ICS 25.040
N 10

中华人民共和国国家标准化指导性技术文件

GB/Z 20177.1—2006

控制网络 LONWORKS 技术规范
第 1 部分：协议规范

广东省网络空间安全协会受控资料

Control network LONWORKS technology specification—
Part 1: Protocol specification

2006-05-08 发布

中华人民共和国国家质量监督检验检疫总局
中国国家标准化管理委员会 发布

目　　次

前言 …… Ⅶ
引言 …… Ⅷ
1　范围 …… 1
2　规范性引用文件 …… 1
3　术语和定义 …… 1
4　符号和缩略语 …… 3
4.1　符号和图形表示 …… 3
4.2　缩略语 …… 4
5　协议分层概述 …… 5
6　MAC 子层 …… 6
6.1　提供的服务 …… 6
6.2　链路层接口 …… 6
6.3　物理层接口 …… 7
6.4　MPDU 格式 …… 7
6.5　可预测 *p*-保持 CSMA 概述 …… 8
6.6　空闲信道检测 …… 9
6.7　随机化 …… 9
6.8　信道使用统计 BL(Backlog)估计 …… 9
6.9　可选的优先级 …… 10
6.10　可选的冲突检测 …… 11
6.11　Beta1,Beta2 和前置码定时 …… 11
7　链路层 …… 13
7.1　前提条件 …… 13
7.2　提供的服务 …… 13
7.3　CRC …… 13
7.4　传输算法 …… 14
7.5　接收算法 …… 14
8　网络层 …… 14
8.1　前提条件 …… 14
8.2　提供的服务 …… 15
8.3　服务接口 …… 16
8.4　网络层内部结构 …… 16
8.5　NPDU 格式 …… 16
8.6　地址识别 …… 17
8.7　路由器 …… 17
8.8　路由算法 …… 18
8.9　自学习算法-子网 …… 18
9　事务控制子层 …… 18

广东省网络空间安全协会受控资料

9.1 前提条件 …… 18
9.2 提供的服务 …… 18
9.3 服务接口 …… 18
9.4 状态变量 …… 19
9.5 事务控制算法 …… 19
10 传输层 …… 20
10.1 前提条件 …… 20
10.2 提供的服务 …… 20
10.3 服务接口 …… 20
10.4 TPDU 的类型和格式 …… 21
10.5 协议图解 …… 22
10.6 传输协议状态变量 …… 22
10.7 发送算法 …… 22
10.8 接收算法 …… 22
10.9 接收事务记录池的大小和配置 …… 23
11 会话层 …… 24
11.1 前提条件 …… 24
11.2 提供的服务 …… 24
11.3 服务接口 …… 25
11.4 会话层内部结构 …… 25
11.5 SPDU 类型和格式 …… 26
11.6 协议时序图 …… 27
11.7 请求/响应状态变量 …… 28
11.8 请求/响应协议——客户端部分 …… 29
11.9 请求/响应协议——服务器部分 …… 29
11.10 请求/响应协议定时器 …… 29
11.11 鉴别协议 …… 29
11.12 加密算法 …… 30
11.13 重试与校验和函数的作用 …… 30
11.14 随机数的生成 …… 30
11.15 鉴别的用法 …… 30
12 表示/应用层 …… 30
12.1 前提条件 …… 30
12.2 提供的服务 …… 30
12.3 服务接口 …… 31
12.4 APDU 的类型和格式 …… 32
12.5 协议图 …… 32
12.6 应用协议状态变量 …… 34
12.7 离线状态中的请求/响应报文 …… 34
12.8 网络变量 …… 34
12.9 给应用程序的差错通知 …… 35
13 网络管理和诊断 …… 35
13.1 前提条件 …… 35

广东省网络空间安全协会受控资料

13.2 提供的服务 …… 35
13.3 网络管理和诊断应用结构 …… 35
13.4 节点状态 …… 36
13.5 网络管理服务的用法 …… 36
13.6 路由器网络管理命令的用法 …… 39
13.7 NMPDU 格式和类型 …… 39
13.8 DPDU 的类型和格式 …… 49
附录 A(规范性附录) 参考实现 …… 53
A.1 预测的 CSMA 算法 …… 53
A.2 LPDU 发送算法 …… 139
A.3 LPDU 接收算法 …… 142
A.4 路由算法 …… 146
A.5 自学习算法 …… 147
A.6 事务控制算法 …… 148
A.7 网络层算法 …… 157
A.8 带鉴别的 TPDU 和 SPDU 发送算法 …… 182
A.9 应用层 …… 262
A.10 网络管理命令 …… 345
A.11 配置数据结构 …… 399
A.12 参考实现的 Include 文件 …… 428
A.13 应用协议状态变量和地址识别结构 …… 472
A.14 查询标识数据结构 …… 476
A.15 查询应答数据结构 …… 476
A.16 更新域数据结构 …… 476
A.17 离开域数据结构 …… 477
A.18 更新密钥数据结构 …… 477
A.19 更新地址数据结构 …… 477
A.20 查询地址数据结构 …… 479
A.21 查询网络变量配置数据结构 …… 479
A.22 更新组地址数据结构 …… 480
A.23 查询域数据结构 …… 480
A.24 更新网络变量配置数据结构 …… 480
A.25 设置节点模式数据结构 …… 481
A.26 读存储器数据结构 …… 481
A.27 写存储器数据结构 …… 482
A.28 重新计算校验和数据结构 …… 482
A.29 安装命令数据结构 …… 482
A.30 存储器刷新数据结构 …… 488
A.31 查询自标识数据结构 …… 489
A.32 取网络变量数据结构 …… 489
A.33 手动服务请求报文数据结构 …… 489
A.34 产品查询数据结构 …… 489
A.35 路由器模式数据结构 …… 490

广东省网络空间安全协会受控资料

A.36 路由表清除组或子网表数据结构 …… 490
A.37 路由器组或子网下载数据结构 …… 490
A.38 路由器组转发数据结构 …… 491
A.39 路由器子网转发数据结构 …… 491
A.40 路由器组不转发数据结构 …… 491
A.41 路由器子网不转发数据结构 …… 492
A.42 组/子网表报告数据结构 …… 492
A.43 路由器状态数据结构 …… 492
A.44 查询状态数据结构 …… 493
A.45 代理状态数据结构 …… 493
A.46 清除状态数据结构 …… 494
A.47 查询收发器状态数据结构 …… 494
附录 B(规范性附录) 其他数据结构 …… 495
B.1 固定的只读数据结构 …… 496
B.2 域表 …… 500
B.3 地址表 …… 501
B.4 网络变量表(资料性) …… 506
B.5 自标识结构 …… 508
B.6 配置结构 …… 511
B.7 统计相关结构 …… 513
附录 C(资料性附录) 行为特性 …… 515
C.1 信道容量和吞吐率 …… 515
C.2 网络度量方式 …… 516
C.3 事务度量方式 …… 517
C.4 边界条件——上电 …… 518
C.5 边界条件——重负载 …… 518
附录 D(规范性附录) PDU 概要 …… 519
附录 E(规范性附录) 命名和编址 …… 521
E.1 地址类型和格式 …… 521
E.2 域 …… 521
E.3 子网和节点 …… 521
E.4 组 …… 522
E.5 Unique_Node_ID 和节点地址分配 …… 522
E.6 NPDU 编址 …… 523

图 1 GB/Z 20177 四个部分的关系 …… Ⅶ
图 2 网络拓扑和符号 …… 4
图 3 协议术语表示法 …… 4
图 4 协议分层 …… 5
图 5 MAC 层与链路层的接口 …… 6
图 6 MPDU/LPDU 格式 …… 7
图 7 可预测 *p*-保持 CSMA 概念和参数 …… 8
图 8 在忙碌信道数据包周期中优先时间片的分配 …… 10

广东省网络空间安全协会受控资料

图 9　CRC 寄存器的状态行为示例 …… 14
图 10　单一信道中的拓扑图 …… 15
图 11　典型的树型域拓扑结构图 …… 15
图 12　网络层服务接口 …… 16
图 13　网络层内部结构 …… 16
图 14　NPDU 格式 …… 17
图 15　事务控制服务接口 …… 19
图 16　传输层和上层接口 …… 20
图 17　TPDU 的类型和格式 …… 21
图 18　传输协议图解:多播中报文和确认 TPDU 都丢失的情况 …… 22
图 19　传输协议——发送 FSM …… 22
图 20　传输协议——接收 FSM …… 23
图 21　k 次重试事务完成的概率 …… 23
图 22　定时器值计算方法 …… 24
图 23　会话层到应用层的接口 …… 25
图 24　会话层内部结构 …… 25
图 25　SPDU 类型和格式 …… 26
图 26　有多个 SPDU 丢失的非等幂请求 …… 27
图 27　有多个 SPDU 丢失的安全等幂请求 …… 28
图 28　请求/响应协议——客户端 FSM …… 29
图 29　请求/响应协议——简化的服务器端 FSM …… 29
图 30　应用层接口 …… 31
图 31　APDU 格式 …… 32
图 32　多播确认事务的应用层协议图 …… 33
图 33　多播请求/响应事务的应用层协议图 …… 33
图 C.1　经 k 次中继后成功提交的概率 …… 517
图 D.1　协议 PDU 概要 …… 520
图 E.1　物理拓扑和逻辑编址(单个域) …… 522
图 E.2　NPDU/TPDU/SPDU 编址——物理地址格式 …… 523

表 1　应用层原语 …… 31
表 B.1　缓冲区大小编码 …… 499
表 B.2　缓冲区计数编码 …… 500
表 B.3　定时器字段值编码 …… 505
表 B.4　缓冲区超时编码 …… 513
表 C.1　关键吞吐率参数 …… 516
表 E.1　NPDU/TPDU/SPDU 编址——逻辑地址格式 …… 523

广东省网络空间安全协会受控资料

前　言

GB/Z 20177 总标题为《控制网络 LONWORKS 技术规范》，目前包括以下 4 个部分：

——第 1 部分：协议规范；

——第 2 部分：电力线信道规范；

——第 3 部分：自由拓扑双绞线信道规范；

——第 4 部分：基于隧道技术在 IP 信道上传输控制网络协议的规范。

本部分是 GB/Z 20177《控制网络 LONWORKS 技术规范》指导性技术文件的第 1 部分。

本部分修改采用 ANSI/CEA 709.1《控制网络协议规范》。

本部分与 ANSI/CAE 709.1 的主要差异为：

a) 凡是出现 ANSI/CEA 709 的地方都用 GB/Z 20177 代替；

b) 凡是出现 ANSI/CEA 709.1 的地方都用本部分代替；

c) 根据 GB/T 1.1 进行编辑性修改；

d) 为方便使用，在原文的基础上增加了引言部分。

本部分的附录 A、附录 B、附录 D 和附录 E 是规范性附录。

本部分的附录 C 是资料性附录。

本部分由中国机械工业联合会提出。

本部分由全国工业过程测量和控制标准化技术委员会第四分技术委员会归口。

本部分起草单位：机械工业仪器仪表综合技术经济研究所、西南大学、北京交通大学现代通信研究所、北京宽网社区数字化建设有限公司、威世达通信控制技术（北京）有限公司、埃施朗公司。

本部分主要起草人：梅恪、王春喜、王玉敏、杨玉柱、刘枫、孙昕、史学玲、欧阳劲松、刘运基、戴恋、刘永生、李翔宇。

广东省网络空间安全协会受控资料

引　言

《控制网络 LONWORKS 技术规范》基于 OSI 参考模型(GB/T 9387.1—1998),是一个 7 层模型。GB/Z 20177《控制网络 LONWORKS 技术规范》由四个部分组成。

——第 1 部分:协议规范;

——第 2 部分:电力线信道规范;

——第 3 部分:自由拓扑双绞线信道规范;

——第 4 部分:基于隧道技术在 IP 信道上传输控制网络协议的规范。

第 1 部分是整个技术规范的核心,后三部分是第 1 部分的补充。

GB/Z 20177 四个部分的关系见图 1。

本部分是 GB/Z 20177 的第 1 部分,描述了协议第 2～7 层的服务以及 MAC 子层与物理层的接口,该规范描述的接口支持物理层多种传输媒体。

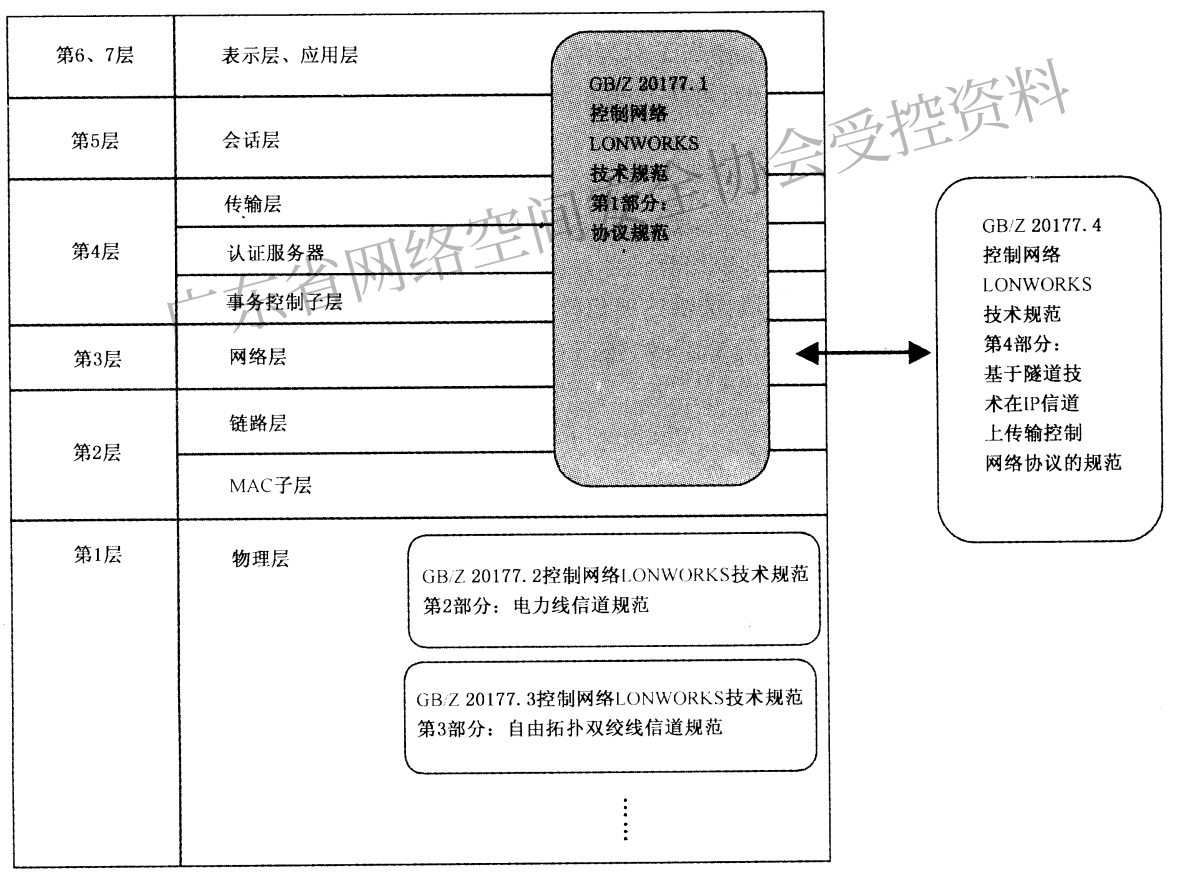


图 1　GB/Z 20177 四个部分的关系

广东省网络空间安全协会受控资料

控制网络 LONWORKS 技术规范
第 1 部分：协议规范

1 范围

GB/Z 20177 的本部分规定了控制网络的通信协议。该协议为网络化控制系统提供了对等式通信，用于实现对等式和主-从式控制策略。

本部分描述第 2～7 层的服务。在第 2 层（数据链路层）规范中，描述了 MAC 子层到物理层的接口。物理层提供传输媒体的选择。本部分描述的接口支持物理层的多种传输媒体。在第 7 层规范中，包含了应用所用的报文类型的描述，包括交换应用数据和网络管理数据所用的报文类型。

本部分适用于自动化控制系统及产品的设计、制造、集成、安装和维护等。

2 规范性引用文件

下列文件中的条款通过 GB/Z 20177 的本部分的引用而成为本部分的条款。凡是注日期的引用文件，其随后所有的修改单（不包括勘误的内容）或修订版均不适用于本部分，然而，鼓励根据本部分达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件，其最新版本适用于本部分。

GB/T 9387.1—1998　信息技术　开放系统互连　基本参考模型　第 1 部分：基本模型（idt ISO/IEC 7498-1：1994）

GB/T 16262—1996　信息处理系统　开放系统互连　抽象语法记法一（ASN.1）规范（idt ISO/IEC 8824：1990）

GB/T 16263—1996　信息处理系统　开放系统互连　抽象语法记法一（ASN.1）基本编码规则规范（idt ISO/IEC 8825：1990）

广东省网络空间安全协会受控资料

3 术语和定义

本章介绍 GB/Z 20177 的本部分采用的基本名词术语。这些术语大部分是通用的，并且在本部分中的含义与通用含义相同。但有些术语有细微的差别。例如，通常网桥是基于第 2 层的目的地址选择性地转发。但在本部分的协议中没有第 2 层编址，所以网桥转发所有数据包，只要数据包中的域地址与网桥所在的域地址相同。路由器通常是执行网络地址的修改，使具有相同传输层但不同网络层的两种协议能互相连接，形成单一的逻辑网络。本部分的路由器可以执行网络地址的修改，但通常它只检查网络地址字段并根据网络层地址字段选择性地转发数据包。

3.1

信道　channel

连接一个或多个通信节点的具有一定带宽的物理单元。参见附录 E 关于本部分的信道和子网之间关系的进一步解释。

3.2

物理层中继器　physical repeater

恢复一个信道段上进入的物理层信号的幅度，并将其转发到其他信道段的设备。

3.3

存储转发中继器　store-and-forward repeater

存储复制数据包并送到另一个信道的设备。

3.4

网桥　bridge

连接两个信道(X、Y)的设备,将所有数据包从 X 信道转发到 Y 信道,或反之。只要这些包源自网桥所从属的域之一。

3.5

配置　configuration

设备用于定制其操作的非易失信息,每一设备中都有为了协议正确运行以及可选的应用运行的配置数据。存储在每一设备中的网络配置数据都有一个与该数据有关的校验和。例如节点地址、通信媒体参数,优先权设置等都是网络配置数据,应用配置信息是专用于应用程序的。

3.6

域　domain

用于管理一个网络单元的虚拟网,组(3.10)和子网(3.8)地址由负责管理这个域的管理员分配,并只在该域中有意义。

3.7

柔性域　flexible domain

用于连结节点唯一标识和广播编址。如果地址相符,节点将对节点唯一标识编址的报文响应,而不管该报文是从哪个域发送来的。为使发送者能接收到响应,该响应必须转送到它被接收的域,并且在该事务处理期间这个域必须被保存,以便对可能的重发进行重复检测。这个在节点中临时的域地址称为柔性域。一个节点能支持多少个柔性域与实现有关,但最少需要一个。

广东省网络空间安全协会受控资料

3.8

子网　subnet

一组通过同样的链路层协议可以访问到的节点;信道的路由抽象。本部分的子网限制最多有 127 个节点。

3.9

节点　node

物理节点的抽象,表示网络中最高程度的地址可分辨性,一个节点在子网中通过其(逻辑的)节点标识符来标识(寻址)。一个物理节点可从属于超过一个子网。这时,对于它所属的每一个子网都分配给它一个(逻辑)节点号。一个物理节点最多可以从属于 2 个子网,这 2 个子网必须在不同的域。一个节点在一个网络中也可以用它的唯一节点标识符(Unique Node ID)(绝对地址)来标识。

3.10

组　group

一个域内可唯一标识的节点的集合。在这个集合中,每个成员由其成员号来标识,组方便了一对多的通信,旨在用于支持功能寻址。

3.11

路由器　router

通过有选择性地从子网转发到子网的方法,将数据包按确定的路径发送到它们相应的目的地的设备。一个路由器总是连结两个子网。路由器可以修改网络层的地址字段。路由器可以设置为下列 4 种模式之一:中继器、网桥、学习路由器和配置路由器模式。在中继器模式,所有接收到的数据包如果没有差错全部转发;在网桥模式,如果接收到的数据包没有差错,且与该路由器所属的域相符则转发;在学习模式,路由器通过检验数据包流量来学习拓扑结构;而在配置模式,路由器存储器存放了网络拓扑结构,完全按配置表的内容作路由的决策。

3.12

(应用)网关　(application)gateway

在网络的最高协议层(通常是两种不同协议)进行网络互联,两个域也可通过应用网关连接。

3.13

时间段　Beta1

紧跟在数据包后面的一个时间段。试图发送的节点侦听信道状态,如果在 Beta1 期间内没有监测到有发送,则确定该信道空闲。

3.14

时间片　Beta2

随机时间片。一个想发送的节点产生一个随机时间延迟 T,该延迟是 Beta2 期间的随机时间片的整倍数。

3.15

网络变量　network variable

应用程序中的变量。在它被赋与一个新值时,其值自动传送到网络上。

3.16

标准网络变量类型　standard network variable types;SNVT

具有标准语义的网络变量。这些变量在所有的应用程序中都以相同方法来解释,是可互操作性的基础。特定 SNVT 的定义超出了本部分的范围。

广东省网络空间安全协会受控资料

3.17

手动服务请求报文　manual service request message

一个包含节点的节点唯一标识的网络管理报文。由接收该报文的网络管理设备用来安装和配置该节点。可由系统或应用产生,也可以由外部硬件事件触发。例如,当"手动服务请求"输入端跳变为低电平时。

3.18

事务　transaction

相互关联的报文序列。例如一个请求和对该请求的响应是一个单次事务的全部。在某事务中当所有期望的报文从所涉及的每个节点最少收到一次时,该事务就完成了。在本部分中,如果在一个事务中任何期待的报文没有收到则事务失败。在发生瞬时差错时,在一个事务中报文重发机制用于提高事务成功的几率。

3.19

控制网络　control network;CN

监测传感器、控制执行器、可靠地通信、管理网络操作和提供对网络数据全面接入的任何装置的集合。GB/Z 20177 特指 LONWORKS 控制网络。

3.20

唯一节点标识　Unique_Node_ID

分配给每个符合 GB/Z 20177 节点的、唯一的 48 比特位的标识符。这个标识符是全世界唯一的,并且在制造时设定。这个标识符的值从节点制造完成后就不再改变。

4　符号和缩略语

4.1　符号和图形表示

图 2 所示是基于本协议的网络基本拓扑结构及用于本部分的符号表示法。

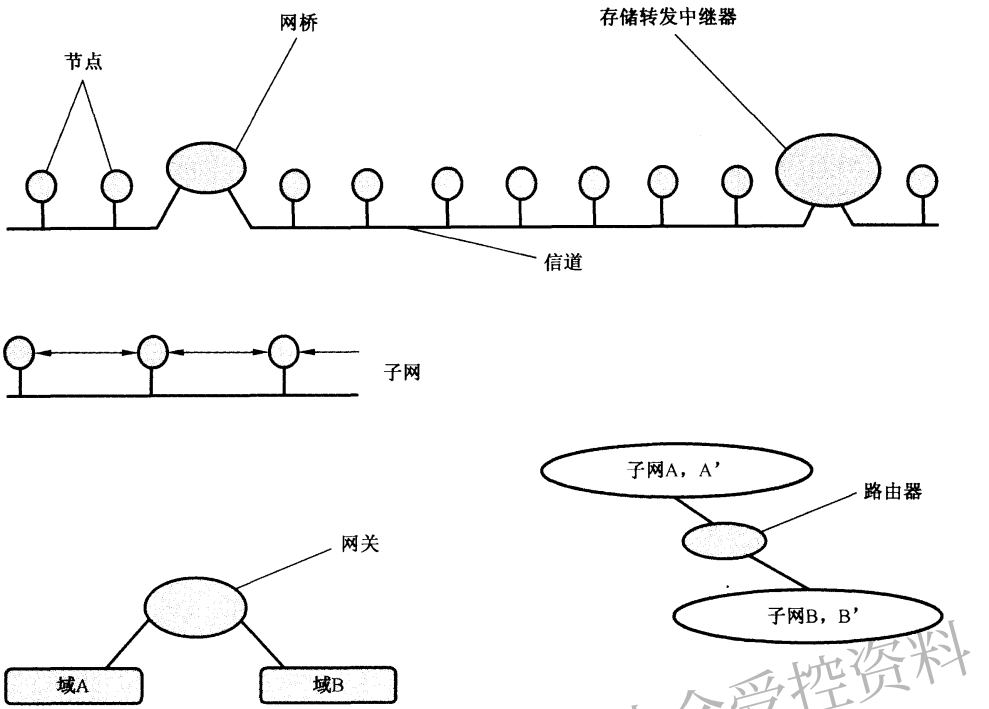


图 2　网络拓扑和符号

广东省网络空间安全协会受控资料

本协议分层使用标准 OSI 的术语描述，如图 3 所示。

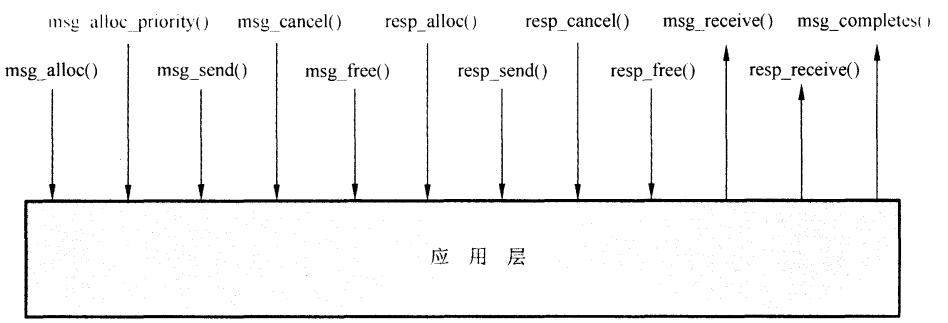


图 3　协议术语表示法

4.2　缩略语

用于本部分中的协议数据单元(PDU)的缩略语有：

PPDU	Physical Protocol Data Unit, or frame	物理协议数据单元(帧)
MPDU	MAC Protocol Data Unit, or frame	MAC 协议数据单元(帧)
LPDU	Link Protocol Data Unit, or frame	链路协议数据单元(帧)
NPDU	Network Protocol Data Unit, or packet	网络协议数据单元(数据包)
TPDU	Transport Protocol Data Unit, or a message/ack	传输协议数据单元(报文/确认)

SPDU	Session Protocol Data Unit,or request/response	会话协议数据单元(请求/响应)
NMPDU	Network Management Protocol Data Unit	网络管理协议数据单元
DPDU	Diagnostic Protocol Data Unit	诊断协议数据单元
APDU	Application Protocol Data Unit	应用协议数据单元
FSM	Finite State Machine (diagram)	有限状态机(图)
CSMA	Carrier Sense,Multiple Access	载波侦听,多路访问
MAC	Medium Access Control	媒体访问控制

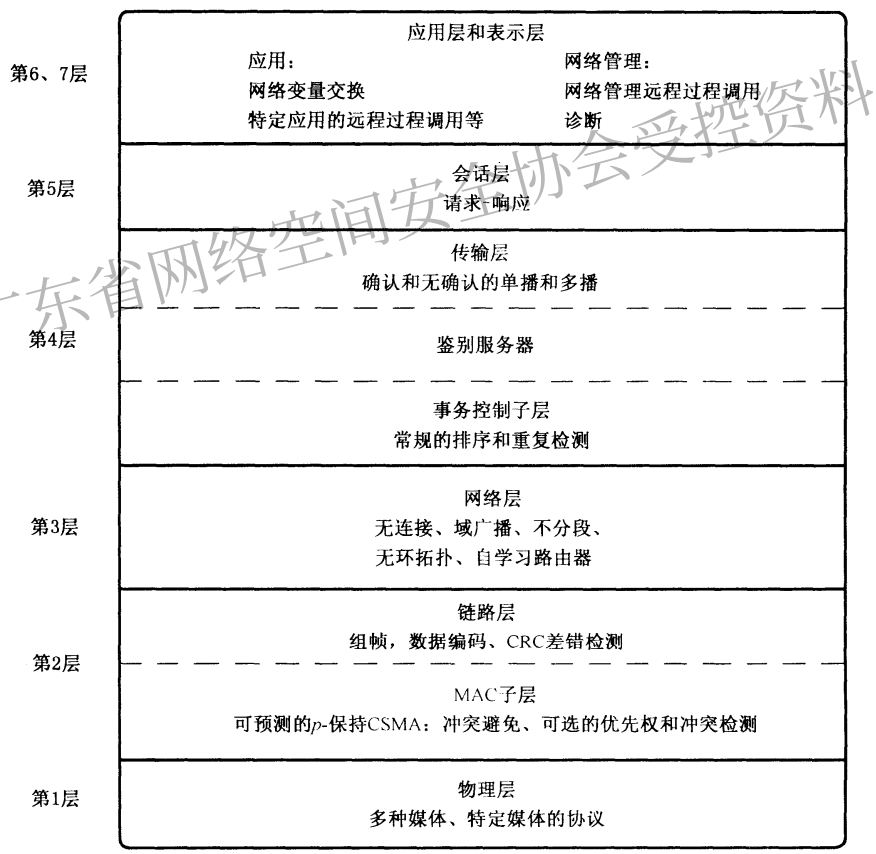
注：附录 D(PDU 汇总)含有这些 PDU 的细节。

5 协议分层概述

本部分规定的协议由图 4 所示的各层所组成,每层描述如下。

在基于本部分的系统中允许多种物理层协议和数据编码方法,每种编码方法与媒体有关。

MAC 子层采用避免冲突的算法,称为可预测 p-保持 CSMA。为了简化和与多播协议兼容等若干原因,链路层支持简单的无连接服务,其功能限于组帧、帧编码和差错检测,无差错恢复的重发机制。



广东省网络空间安全协会受控资料

图 4 协议分层

网络层处理在域内的数据包传输,不提供域间通信,网络服务是无连接、无确认,既不支持报文分段也不支持报文重组。网络层采用学习拓扑的路由算法,假定网络拓扑是树状的;具有配置表的路由器对有物理环路的拓扑结构也可以运行,只要其通信路径逻辑上是树状的。在这种拓扑下,数据包在路由器的包来源一侧不能出现多于一次。单播的路由算法使用最小开销的学习算法,不增加额外的路由流量。单播和组编址均支持使用配置路由表,虽然在许多应用中组编址报文简单地采用扩散法就足够了。

协议层次的核心是传输层和会话层。传输层和会话层公共的事务控制子层处理该两层的事务排序和重复检测。传输层是无连接的，能可靠地将报文提交给单个或多个目的地。报文发送者身份的鉴别包含在传输层服务中，当需要对发送者的安全性进行鉴别时使用。鉴别服务器只要求事务控制子层完成其功能，因此，传输层和会话层报文可以使用除广播之外的任何寻址方式进行鉴别。

会话层提供简单的请求响应机制用于访问远程服务器。该机制提供了一个平台，在此基础上可以建立应用特定的远程过程调用，例如本部分网络管理协议就依赖于会话层的请求-响应机制。

传输层确认报文期待来自远程目的地的报文提交的指示。会话层请求报文期待应用特定的远程任务已完成的指示，一个给定的报文只使用某一种服务，不能使用两种服务。

本部分包括表示层和应用层的最低一级。这些层提供发送和接收应用报文的服务，包括网络变量和其他报文类型，如网络管理、诊断报文消息和外部帧(见第 12 章)。对于网络变量更新，APDU 首部提供如何解释该 APDU 的信息。这一与应用无关的数据解释使得数据能在节点之间共享而无需事先安排。

6 MAC 子层

6.1 提供的服务

本部分 MAC 子层便于媒体访问，带有可选的优先权和可选的冲突检测/冲突解决。它使用被称为可预测 p-保持 CSMA 的协议，类似于 p-保持 CSMA 协议集。

可预测 p-保持 CSMA 是一种避免冲突的技术。利用对信道负载的预测随机化对信道的访问。一个节点想要传输时总是随机延迟一段时间(0..w)后才访问信道。为避免在高负载时吞吐量降低，随机窗口 w 的大小是估计信道使用统计 BL(backlog)的函数，见公式(1)。

$$w = (\mathrm{BL} \times W_{\mathrm{base}}) - 1 \qquad \cdots\cdots (1)$$

式中：

W_{base}——基本窗口大小，以时间来度量，其区间出自 Beta2(见 6.7)，等于 16 个 Beta2 时间片；

BL——信道使用统计估计(见 6.8)。

6.2 链路层接口

如本章所述，MAC 子层紧密耦合到链路层。MAC 子层负责媒体访问，链路层处理所有第 2 层其他的问题，包括组帧和差错检测。为了解释，该两层之间的接口以图 5 的方式描述。

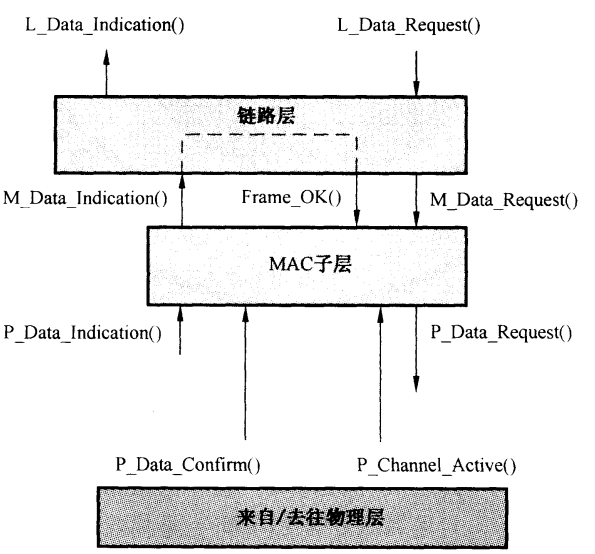


图 5 MAC 层与链路层的接口

广东省网络空间安全协会受控资料

虽然服务接口原语使用类似于编程语言的过程调用语法来定义，但其中并不隐含实现的方法。帧接收完全由链路层处理，并通过 Frame_OK()原语通知 MAC 子层关于信道使用统计的增加。

下面的服务接口原语方便了链路层与 MAC 子层之间的接口：

M_Data_Request(Priority，delta_BL，Alt_Path，LPDU)

该原语被链路层用于将一个输出的 LPDU/MPDU 传递给 MAC 子层。优先权定义了数据帧被发送的优先级别；delta_BL 是期待的由于提交该 MPDU 产生的信道使用统计增量。Alt_Path 是一个二进制标志，指明该 LPDU 是在主信道还是备用信道传输，其波特率等。关于如何设置 Alt_Path 见 6.4。

Frame_OK(delta_BL)

在接收到一个帧并且验证其 CRC 正确无误时，链路层调用该原语，通知 MAC 子层关于与刚接收的帧相关联的信道使用统计增量。

M_Data_Indication()

对每一个进入的 LPDU/MPDU，MAC 子层给链路层提供一次该指示。

6.3 物理层接口

物理层处理二进制数据的实际发送和接收。本部分控制网络协议支持多种物理层协议，提交给链路层的比特误码率必须小于或等于 10^{-4}。为了与高层协议兼容，所有物理层协议必须支持所定义的服务接口(见图 5)：

P_Data_Indication (Frame)

对每个进入的 LPDU/MPDU，物理层给 MAC 子层提供一次该指示。

广东省网络空间安全协会受控资料

P_Data_Request (Frame)

MAC 子层用这一原语将一个编码的 LPDU/MPDU 帧传递给物理层立即发送。比特位传输顺序在附录 D 中定义。

P_Data_Confirm (Status)

物理层返回状态 Status 指明该帧是否已被发送过。Status 可能有三个值：

success(成功)——指明该帧被发送；

request_denied(请求被拒绝)——指示发送前已检测到通信线路有活动；

collision(冲突)——指示发送已开始，但检测到冲突。发送是否中断取决于冲突在什么时候被检测到(见 6.10)。

P_Channel_Active()

物理层使用这一原语将信道状态传递给 MAC 子层，这一指示表明有活动，但不必是有效数据。

6.4 MPDU 格式

MPDU/LPDU 组合的格式如图 6 所示(附录 D 含有 NPDU 帧的细节)。

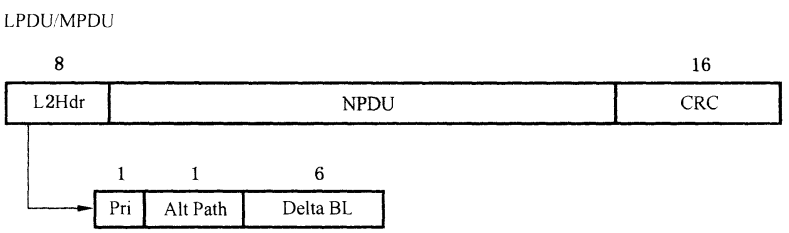


图 6 MPDU/LPDU 格式

MAC 子层使用 L2Hdr 字段，其语法和语义如下：

Pri：1 比特位字段，规定该 MPDU 的优先级：0＝普通，1＝高；

Alt_Path：1 比特位字段，规定使用的信道，供收发器使用。该收发器能够在两条不同信道上传输，并可在任何一条信道上接收而不需要明确指示收发器在某一条信道上接收。除非在每次传输时，要求规定备用路径，传输层在最后两次尝试中设置该标志位（对于确认和请求/响应服务）。对于任何接收到的数据包，如果 Alt_Path 标志位被设置，并且要求确认、响应、质询或应答，那在相应的确认、响应、质询或应答中，Alt_Path 标志位也应该被设置。

Delta_BL：6 比特位无符号字段（≥0）；规定提交该 MPDU 所产生的信道使用统计的增量。

6.5 可预测 *p*-保持 CSMA 概述

如同 CSMA，可预测 *p*-保持 CSMA 在传输前侦听媒体。试图传输的节点监听信道状态（见图 7），如果在 Beta1 周期内检测到没有信息发送，则确定信道处于空闲状态。在该 Beta1 周期内没有数据包要发送的节点在优先时间片期间以及最小的 W_{base} 随机时间片期间保持同步（见 6.10）。这一同步的维持，允许在 Beta1 时间结束后到达 MAC 子层输出队列中的数据包，按照有数据包要传输的其他节点所要求的有效时间片期间内传输。

随后，该节点生成随机延时 T（传输），它处于时间间隔（0...（BL×W_{base}）－1）内，其中 W_{base} 是基本随机窗口内随机时间片的数目，BL 是当前信道使用统计估计值，T（传输）定义为 Beta 2（见 6.7 和 6.8）期间随机时间片的整倍数。如果在延迟时间到时信道空闲，则该节点发送数据包，否则该节点接收入向数据包，然后重复 MAC 算法。在图 7 中，D_{mean} 是数据包之间的平均随机延时，并且，随机延时 T 是均匀分布的，对于小数值的 BL，D_{mean} 的计算见公式（2）：

$$D_{mean} = (W_{base} - 1)/2 \quad \cdots\cdots (2)$$

式中：

D_{mean}——数据包之间的平均随机延时；

W_{base}——基本随机窗口内随机时间片的数目。

广东省网络空间安全协会资料

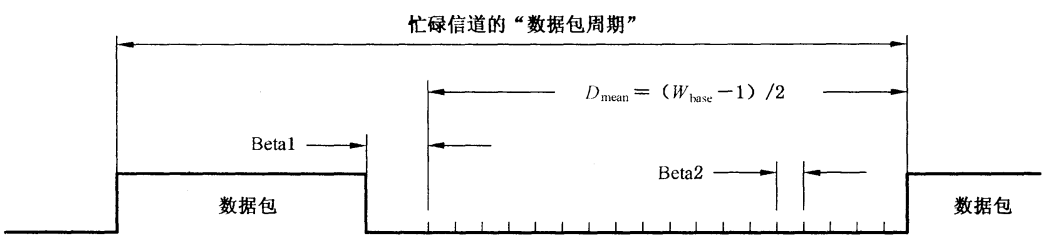


图 7 可预测 *p*-保持 CSMA 概念和参数

通过调整随机窗口的大小作为预测负载的函数，这个算法保持冲突率不变，并且与负载无关。只要估计的信道使用统计大于或等于实际的信道使用统计，则公式（3）成立：

$$\text{冲突率} = \text{差错包周期} / \text{无差错包周期} \leqslant W_{base}/2 \quad \cdots\cdots (3)$$

式中：

W_{base}——基本随机窗口内随机时间片的数目。

GB/Z 20177 采用的基本窗口大小为 16，这就意味着在相邻数据包之间平均有 8 个宽度为 Beta2 的随机时间片和 1 个宽度为 Beta1 的时间片。而且，Beta2 周期的宽度对信道使用效率是极重要的。

预测 CSMA 算法见附录 A.1。

6.6 空闲信道检测

当满足下列两个条件时，判定信道为空闲：

a) 当前信道状态由物理层通过 P_Data_Indication ( )原语报告为低，且，

b) 在最后一个 Beta1 周期无跳变被检测到。注意，MAC 子层可以被配置为在 Beta1 周期某一部分时间内忽略跳变，这部分忽略跳变的时间(在这段时间内即使出现跳变也被假定为信道空闲)被称为不确定时间(详见 6.11)。

Beta1 周期长度由公式(4)确定：

$$\text{Beta1} > 1\text{ 比特位时间} + (2 \times \text{Tau}_p + \text{Tau}_m) \quad \cdots\cdots (4)$$

式中：

Beta1——紧跟在数据包后面的时间片(见 3.13)；

Tau_p——由信道媒体长度确定的物理传播延时；

Tau_m——MAC 子层检测和回转延时。即从检测到信道空闲条件的时间开始到输出点出现第一个输出跳变。在有载波的媒体上，该时间必须包括从打开载波到媒体上判定出现有效载波之间的一段时间。

第一项假定数据编码方法保证在每 1 比特位时间跳变和/或载波。如果所用的编码方法不满足该限制条件，则第一项必须调整为信道可以表现为空闲但不是出现空闲的最长时间，即合法数据传输的最长时间，中间无须判定在媒体上的跳变和/或载波。第二项负责传播和回转延时。

6.7 随机化

在随机化周期开始时，想要发送的节点产生一个随机延时 T(发送)，它处于时间间隔 (0...(BL×W_{base})−1)内。然后该节点在这段时间内等待，同时继续监听信道状态，当延时已到且信道仍然空闲，该节点则发送数据。

发送延时 T(发送)是 Beta2 时段内随机时间片的整倍数；随机时间片的长度必须满足公式(5)：

$$\text{Beta2} > 2 \times \text{Tau}_p + \text{Tau}_m \quad \cdots\cdots (5)$$

式中：

Beta2——随机时间片(见 3.14)；

Tau_p——由信道媒体长度确定的物理传播延时；

Tau_m——MAC 子层检测和回转延时。即从检测到信道空闲条件的时间开始到输出点出现第一个输出跳变。在有载波的媒体上，该时间必须包括从打开载波到媒体上判定出现有效载波之间的一段时间。

6.8 信道使用统计 BL(Backlog)估计

MAC 算法的预测方面是基于信道使用统计的估算。每一节点都保持一个当前信道使用统计 BL 的估计值。即由于发送或接收一个 MPDU 而产生的增量或周期性的减量——每一数据包周期一次。信道使用统计被编码到链路层首部，表示该数据包在被接收时产生的报文数目。信道使用统计初始化时置为 1，在发送或接收非零信道使用统计增量的数据包之后，该节点的信道使用统计估计值由该信道使用统计的增量来增加。最大信道使用统计值是 63，如果超过 63，则信道使用溢出统计值加 1(见 13.8.1 和 B.7)。

在下列条件之一的情况下信道使用统计减 1：

在等待发送时：如果 W_{base} 随机时间片过去而没有信道活动。

在接收时：如果一个信道使用统计增量为“0”的数据包被接收。

在发送时：如果一个信道使用统计增量为“0”的数据包被发送。

在空闲时：如果一个数据包周期时间结束而没有信道活动。

每当信道使用统计改变时数据包周期定时器复位为初始值。每当 MAC 层变为空闲数据包周期定

广东省网络空间安全协会受控资料

时器开始计时(从其当前值开始递减)。MAC 层空闲定义为:

a) 没有接收;

b) 没有发送;

c) 不在等待发送;

d) 不在 Beta1 计时时段;

e) 不在等待优先时间片;

f) 不在等待第一个 W_{base} 随机窗口完成。

在从空闲转变到发送或接收时,数据包周期定时器暂停计时。

信道使用统计值总是≥1。本算法是通过与被发送的 MPDU 相关联的数量对信道使用统计进行后增量而非前增量,因为期望的响应数在发送该 MPDU 之前并不重要。

6.9 可选的优先级

本协议支持基于某一信道的优先级。如果有优先时间片,则它紧跟在数据包传输后的 Beta1 周期之后(见图 8)。每个信道的优先时间片数目为 0～127,优先时间片通常不是通过竞争来使用,而是唯一指定给信道上的节点。被分配了优先时间片的节点并非每个报文都必须使用它,是否使用所分配的优先时间片,是由节点根据每个报文的情况来决定的,决定的依据是检查 LPDU 首部(图 6)的优先级标识位。

可以分配给同一信道上所有节点相同的优先时间片。这在某些情况下是有意义的。例如,一个有对等式活动后台的结构,且单个主节点周期性地对每一个节点做某事(例如网络管理,轮询等)。通过给予每一节点以相同的优先时间片,并且只用于该目的,则这些事务(从单个主节点到多个从节点)将先于后台业务量完成之前完成。

应用可以决定一个报文是高优先级并且试图优先发送。如果节点没有被分配给优先时间片,则该报文以通常的方式发出,除了在第 2 层首部的优先标识位置位以外。因而,如果该数据包通过在其目的地信道上有优先时间片的路由器,则该数据包应使用该路由器的优先级传送。如果该路由器没有优先时间片,该报文将被以通常方式转发,而其第 2 层首部的优先标识位保持置位。

广东省网络空间安全协会受控资料

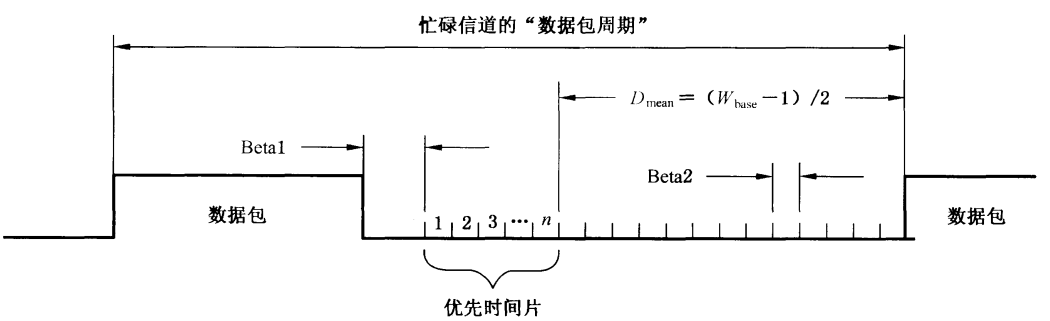


图 8 在忙碌信道数据包周期中优先时间片的分配

本协议不提供节点间的同步。因此,如果信道空闲时间长于随机化周期(Beta1+优先时间片数+D_{mean}(见以上所述)),对链路的访问是随机的而不管其优先级。一旦链路回到忙碌状态,对链路的访问就应按优先顺序。

如果一个优先报文要求:

——确认(确认报文服务);

——响应(请求/响应报文服务);

——质询(对鉴别确认报文或鉴别请求的响应);或

——应答(对鉴别质询的响应)。

响应的节点将通过在第2层首部设置优先级标识位来试图发送优先确认/响应/质询/应答。如果高优先级报文在节点内部生成,则它先于任何普通优先级的排队等待数据包被发送。多个高优先级的数据包按FIFO(先进先出)的顺序发送。如果应用试图在节点发送数据包期间发送高优先报文,则让正在发送的数据包先发送完。

如果信道具有优先时间片,而节点内部有多个报文排队等待,则不应在连续的数据包周期内发送这些报文。在节点有数据包要发送,而在前一个数据包周期内已发送了一个数据包的情况下,则该节点在当前周期内不使用它的常规的媒体访问算法,而使用稍加修改的非优先MAC算法来访问媒体。修改的算法是发送节点在刚刚发送的优先数据包后等待Beta1时间,然后等所有优先时间片过去,然后等待一个随机时间片窗口过去(16个Beta2时间片),这个时候它使用当前信道使用统计估计值随机化对网络的访问。如果该节点在当前数据包周期内不成功,则它可在后续数据包周期使用其优先时间片。

6.10 可选的冲突检测

物理层可以选择通知MAC子层检测到冲突。MAC子层可以配置为在数据包传输期间的任何时候检查有无冲突检测通知。

冲突解决可以通过冲突检测来实现。可以在前置码字段前插入对信道上的每一节点唯一的位模式来解决冲突。这样物理层必须定义在仲裁模式后面有足够长的前置码,以便在数据包周期内丢失仲裁的发送节点能够回转,从刚赢得仲裁的节点接收入向数据包。

如果采用冲突检测,则应仅当发送数据包时才启用。当在MAC子层检测到冲突时,信道使用统计加1。接收方并不检测冲突,而把冲突看作是太短的数据包,或CRC不正确,或者有时候被看成是不能达到位同步的虚假的跳变。当检测到冲突时,信道使用统计加1,冲突统计也加1(见13.8.1和B.7)。

MAC子层在接到冲突通知时,试图重发该数据包,它遵守下列规则:

a) 如果在节点优先时间片内两次连续试图发送优先数据包时检测到冲突,则下一次尝试发送优先数据包将不使用配置的优先时间片,而是使用按照非优先MAC算法挑出一个时间片。在这种情况下该数据包第2层首部优先级标识位仍保持置位。

b) 每当发送节点检测到冲突时,这个发送节点将其信道使用统计估计值加1。

c) 当发送一个数据包时连续尝试256次都检测到冲突,则该数据包被丢弃。

6.11 Beta1,Beta2和前置码定时

本部分节点可实现支持一种或多种媒体。如果实现支持多种媒体,其前置码的长度和Beta1及Beta2的时间必须能够配置以支持相应的物理通信媒体。Beta2是媒体访问时间片(随机时间片或优先时间片)的时间段。用于Beta1的MAC层定时器提供给前一个数据包周期的发送者以及接收者使用。这是为了纠正前一个数据包结束时发送者和接收者之间的时基差。此外,有2个通信配置(见B.6中的Comm_type)用不同的公式根据类型来计算Beta1。

一个节点必须能够生成5个基本时间周期之一,以产生正确时间段的Beta1和Beta2时间片。

Beta2公式和前置码见公式(6)。

$$\mathrm{Beta2} = \mathrm{CT} \times (40 + 20 \times v) \quad \cdots\cdots (6)$$

式中:

Beta2——随机时间片(见3.14);

CT——用于生成Beta1及Beta2时间片和前置码的时基。根据所支持的媒体类型不同,每一节点必须支持一个或多个下列CT值:600 ns,1.2 μs,2.4 μs,4.8 μs,9.6 μs;

v——调节参数,取值范围为0～255。

广东省网络空间安全协会受控资料

计算 Beta1 的公式有 4 个，用哪一个公式取决于配置结构中 Comm_type 的比特位字段，见 B.6，以及计时是否立即在接收或发送数据包后进行。Comm_type 的值可以为 1 或 2，选择哪一个值决定了 MAC 子层的行为，详述如下：

Comm_type = 1：

a) 在 Beta1 周期中有一个不确定时间，在这个时间内信道上所有跳变都被忽略，这个时间周期跟在任何发送或接收数据包结束后即开始。其区间定义如下：在不确定时间之后及第 1 个 Beta2 时间片之前，MAC 子层重复地等待一个 Beta2 时间片的时间段周期过去，且在该信道上进行传输前没有跳变(注：如果在此期间检测到一个有效的前置码，则应尝试接收输入的数据包)。这个 Beta2 时间片时间长度为总的 Beta1 时间内的 Beta2 时间段。

b) 发送的前置码长度在 MAC 子层控制下。

c) MAC 子层忽略在发送的前置码的前 25%期间发生的冲突。它还可选择地忽略在 CRC 传输后但在传输结束前所报告的冲突。(根据 B.6 中描述的 cd_tail 字段。)

d) 如果冲突在前置码传输期间被检测到，MAC 子层可终止该数据包，如果这样配置的话。在前置码已经发送后检测到的冲突不终止传输。

Comm_type = 2：

a) 在 MAC 子层没有不确定时间定义，如果有不确定时间，则应由物理层加以强制。

b) MAC 子层配置为或者控制前置码传输长度，或者由物理层控制前置码传输长度。

c) MAC 子层总是在接收到冲突的通知时终止数据包传输。

d) 如有必要，MAC 子层在传输前重新配置物理层，通知它所选择传输路径的改变(有备用路径与无备用路径)。

广东省网络空间安全协会反控资料

对应于 Comm_type=1 和 Comm_type=2 时的发送、接收，这四种情况的 Beta1 计算见公式(7)～(10)：

$$\text{Beta1(发送后)} = \text{CT} \times (583 + f(\text{xmit\_interpacket})) + \text{Beta2}\ \{\text{comm\_type}=1\} \quad \cdots\cdots(7)$$

$$\text{Beta1(接收后)} = \text{CT} \times (565 + f(\text{recv\_interpacket})) + \text{Beta2}\ \{\text{comm\_type}=1\} \quad \cdots\cdots(8)$$

$$\text{Beta1(发送后)} = \text{CT} \times (624 + f(\text{xmit\_interpacket})) + \text{Beta2}\ \{\text{comm\_type}=2\} \quad \cdots\cdots(9)$$

$$\text{Beta1(接收后)} = \text{CT} \times (602 + f(\text{recv\_interpacket})) + \text{Beta2}\ \{\text{comm\_type}=2\} \quad \cdots\cdots(10)$$

式中：

Beta1——紧跟在数据包后面的时间片(见 3.13)；

Beta2——随机时间片(见 3.14)；

CT——见公式(6)；

xmit_interpacket——取值范围 0～255，被存储在配置结构中(B.6)；

recv_interpacket——取值范围 0～255，被存储在配置结构中(B.6)；

$f()$——一个函数，返回一个时间延迟，用于补偿信道上其他较慢的节点，这些节点实际产生的 CT 值可能低于信道额定值。其计算见公式(11)。

$$f(v) = (v < 128)?(41 \times v) : 145 \times (v - 128) \quad \cdots\cdots\cdots\cdots(11)$$

式中：

v——见公式(6)。

MAC 子层不确定时间的计算见公式(12)、(13)：

$$\text{IDT(发送后)} = \text{CT} \times (313 + f(\text{xmit\_interpacket}))\{\text{comm\_type}=1\} \quad \cdots\cdots\cdots\cdots(12)$$

$$\text{IDT(接收后)} = \text{CT} \times (295 + f(\text{recv\_interpacket}))\{\text{comm\_type}=1\} \quad \cdots\cdots\cdots\cdots(13)$$

式中：

IDT——MAC 子层不确定时间；

CT——见公式(6)；

$f(v)$——见公式(11)；

xmit_interpacket——见公式(7)；

recv_interpacket——见公式(7)。

注：在所有情况下，不确定时间都限制为小于 Beta1 总时间。

前置码长度由协议处理器或者收发器控制。当由协议处理器控制前置码时，前置码长度所有可能值见公式(14)。

$$\text{前置码长度} = \text{CT} \times (219 + 32 \times v) \quad \cdots\cdots (14)$$

式中：

CT——见公式(6)；

v——取值范围 0～253。

当收发器控制前置码长度(当 comm_type = 2 时可选)时，允许的最小前置码长度为 181 μs。该值是和具有 600ns 计时器值的协议处理器相对应的，该值随时基值 CT 的变化而变化。当收发器控制前置码长度时，最大前置码长度由收发器控制。

Beta1、Beta2 和每个 GB/Z 20177 兼容收发器类型的前置码定时在 GB/Z 20177 相应的物理层规范中作了规定。

7 链路层

7.1 前提条件

链路层假设由于冲突和传输差错可能造成一定的 CRC 差错率 P_e，由于 P_e 很小，因而不需要链路级的差错恢复机制。

基于以上前提条件，只有当通信路径的差错率总和远远低于 1 时，这种端到端的通信才是成功的。

$$\text{SUM}(P_e) \ll 1$$

在 P_e 为常数的网络中，最大通信距离 D(网络或组的直径，即成功事务所能跨越的信道的数目)必须满足：

$$D \ll 1/P_e$$

广东省网络空间安全协会受控资料

7.2 提供的服务

链路层提供对子网范围内的、有序的、无确认的 LPDU 提交，带差错检测，但没有差错恢复。一旦 CRC 校验有错，则将损坏的帧丢弃。

7.3 CRC

CRC 计算整个 NPDU，包括 L2Hdr 字段。利用公式(15)确定的多项式 $G(X)$生成 CRC：

$$G(X) = X^{16} + X^{12} + X^{5} + 1 \quad \cdots\cdots (15)$$

式中：

X——输入位。

对于每一 NPDU，CRC 寄存器初始化为全 1。图 9 是应用以上多项式的 CRC 移位寄存器的状态行为示例。寄存器的各比特位由标号为比特位 0～比特位 15 的各列表示，CRC 将要校验的数据在标号为 crc_in 列中。寄存器初始化为全 1，此后数据被串行地送入寄存器，从第 1 个字节的第 7 比特位开始。表中的每一行显示在某一比特位时间的寄存器状态。只要计算模式为真(mode compute=1)，寄存器将连续输入和计算。计算模式为假，表示数据结束和 CRC 计算结束。

数据输入后，标号为 nrz_out 的列中的 16 比特位 CRC 数据串行移出。

当链路层接收到一个 CRC 校验出错的数据包，或者长度小于 8 个字节数据包时，传输差错统计数加 1(见 13.8.1)，数据包周期计时器启动但并没有被重置，这和正确的数据包接收后不一样。如果一个数据包被接收，但它的长度对于输入缓冲区来说太长，或者已没有输入缓冲区，丢失数据包统计数将加 1(见 13.8.1)。

16 位 CRC 计算的规范描述，参见 A.2 的伪代码。

	计算模式	crc_in	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8	bit 9	bit 10	bit 11	bit 12	bit 13	bit 14	bit 15	nrz_out
初始值			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
字节 1 bit7	1	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	0
6	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1
5	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
4	1	1	0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	1
3	1	1	1	0	0	0	1	0	1	1	1	0	1	1	0	1	1	1	1
2	1	0	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0
1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	0	0	0	1	0
0	1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	0	0	0	1
字节 2 bit7	1	1	1	0	1	1	1	1	0	1	0	0	1	1	0	0	0	0	1
6	1	0	0	1	0	1	1	1	1	0	1	0	0	1	1	0	0	0	0
5	1	0	0	0	1	0	1	1	1	1	0	1	0	0	1	1	0	0	0
4	1	1	1	0	0	1	0	0	1	1		0	1	0	1	1	1	0	1
3	1	1	1	1	0	0	1	1	0	1	1	1	0	1	1	1	1	1	1
2	1	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1	1	1	0
1	1	0	1	1	1	1	0	1	0	1	0	1	1	1	1	0	1	1	0
0	1	0	1	1	1	1	1	1	1	0	1	0	1	1	0	1	0	1	0
字节 3 bit7	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1	0	1	0	1
6	1	1	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	1
5	1	1	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1
4	1	0	0	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0
3	1	0	1	0	0	1	0	0	1	1	0	1	1	1	1	1	0	0	0
2	1	0	0	1	0	0	1	0	0	1	1	0	1	1	1	1	1	0	0
1	1	0	0	0	1	0	0	1	0	0	1	1	0	1	1	1	1	1	0
0	1	0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	1	1	0
crc bit15	0		0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	1	0
14	0		0	0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	0
13	0		0	0	0	1	0	0	1	0	1	1	0	0	1	1	0	0	0
12	0		0	0	0	0	1	0	0	1	0	1	1	0	0	1	1	0	1
11	0		0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	1	1
10	0		0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0
9	0		0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	0
8	0		0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	1
7	0		0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1
6	0		0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0
5	0		0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
4	0		0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
3	0		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
2	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
1	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 9　CRC 寄存器的状态行为示例

广东省网络空间安全协会受控资料

7.4　传输算法

见 A.2 的传输算法伪代码。

7.5　接收算法

一个有效帧以信道活动状态开始，以信道空闲状态终止。对于接收而言，有效帧的处理过程描述如下：无效帧将被丢弃。接收算法伪代码见 A.3。

8　网络层

8.1　前提条件

本部分协议支持多种拓扑结构，以满足多种应用领域的需要。在单一信道内，拓扑结构可以是总线

型、环型、星型或“自由”型(见图 10)。自由拓扑是这样定义的,整个网络布线无其他规则,只用一个终端器放在网络的任意位置。所以自由拓扑可以包括环型、星型、总线型,或者是它们的任意组合。

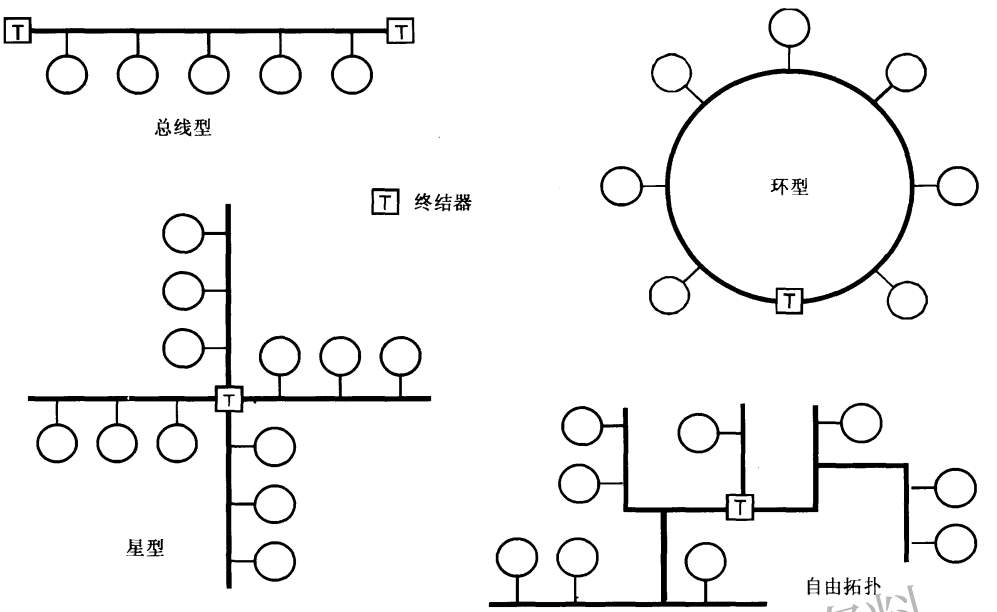


图 10 单一信道中的拓扑图

广东省网络空间安全协会受控资料

该协议支持物理层中继器以及存储转发中继器,从一个信道向另一信道重发数据包。该协议也支持网桥,从该网桥所在域的一个信道向另一个信道重发所有数据包。此外,还支持自学习型和配置型路由器,对通信量进行分割,从而提高系统总体性能。

在网络中,从一个节点到另一个节点可能存在多条路径,这就存在着数据包无限循环的危险。在这些网络中,必须采用配置路由器在物理环路拓扑的上面强行建立逻辑树状拓扑。

如果需要使用中继器、网桥、自学习型路由器,就必须控制拓扑结构,以保证无环路存在。为防止在一个域范围内出现路由环路,域拓扑结构必须采用图 11 中描述的树状结构。只有两个不同的信道相连时才可以使用存储转发中继器;在同一信道中不支持存储转发中继器。这个约束对于顺序保持的网络是必须的。

如果任何节点接收到的数据包中的协议版本号是它所不支持的,则该数据包被丢弃。

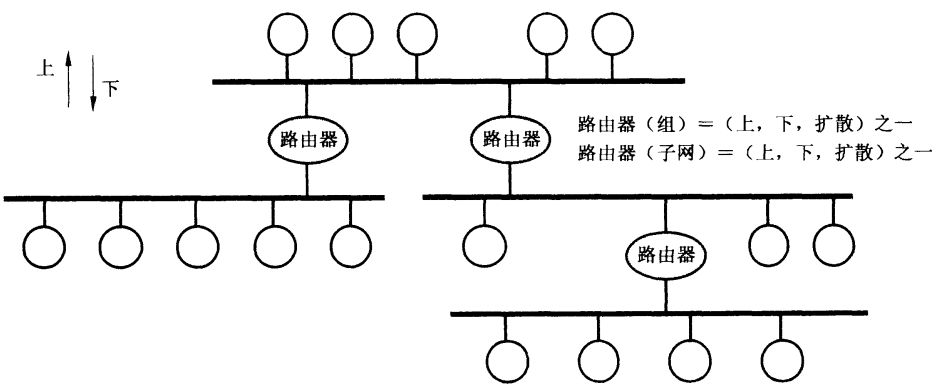


图 11 典型的树型域拓扑结构图

8.2 提供的服务

GB/Z 20177 网络层提供无连接的网络服务,方便域范围内的数据包交付,具有以下属性:

a) 无确认单播、多播、广播：这取决于它的目的地址，提交的数据包被交付到单个节点、多个节点或域内的所有节点（或域内规定子网的所有节点）。这种交付方式发生的概率 $p \leqslant 1$。

b) 丢失：网络层不支持重发和确认机制。穿过的信道数目多，将使提交概率下降。

c) 顺序保持：无环路拓扑结构（逻辑上通过配置型路由器或物理上通过拓扑控制来实现）并且无单一信道的存储转发中继器存在，提供了自然排序。

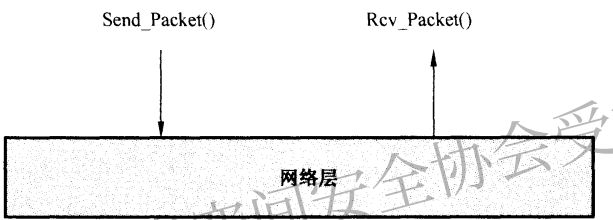
d) 无分割：在网络层不允许对报文进行分割或重组。

当使用自学习型路由器时，路由器通过检查数据包的网络层地址段来发现其拓扑。这种学习算法不会给网络增加额外的通信量开销。它假定域内没有环路，通过检查被路由的 NPDU 源地址来学习子网位置。成组编址的 NPDU 通过扩散法被路由，在整个域范围内传播。

如果节点不是处于硬离线状态（见 12.4）或配置状态（即非配置状态），网络层将禁止出向报文，人工服务请求报文、响应和 ACK 除外，这些特殊的报文可在任何状态发送。

8.3 服务接口

网络服务接口由 Send_Packet()服务请求和 Rcv_Packet()指示组成，如图 12 所示。再有，提供这一接口是为了解释的目的。然而在实际实现中，该层可以和传输层合并，对外只提供传输层的接口。



广东省网络空间安全协会受控资料

图 12 网络层服务接口

这两个接口原语的语法为：

Send_Packet (address_pair, pduType, PDU, priority, delta_BL, Alt_Path)

Rcv_Packet (address_pair, pduType, PDU, priority)

8.4 网络层内部结构

网络层执行两种功能，地址识别和路由控制，见图 13。

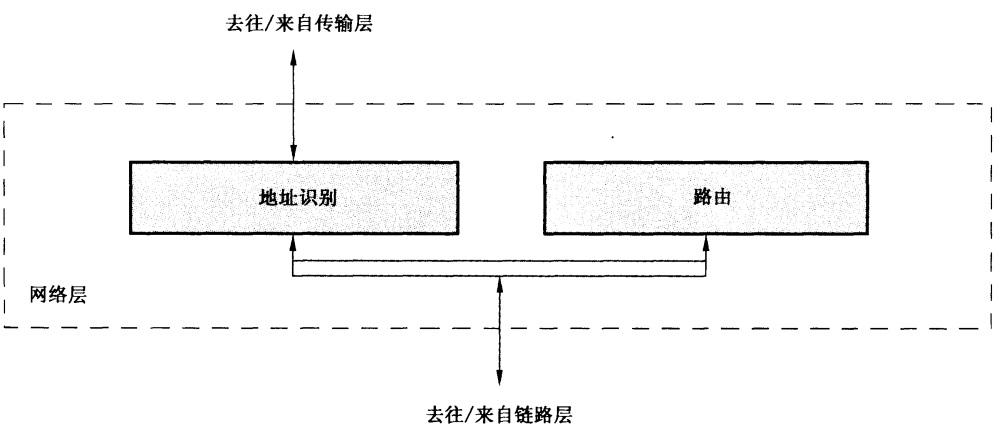


图 13 网络层内部结构

8.5 NPDU 格式

NPDU 格式如图 14 所示，NPDU 携带并封装 TPDU、或 SPDU、或 AuthPDU、或 APDU。没有定义网络层内部使用的 NPDU。图 14 中每一个字段上方的数字指出该字段的位数。图中符号字段的值

按所示的枚举范围(1,2,3,…,n)的顺序赋值。例如,在 PDU_Fmt 字段,值 3 表明其封装的 PDU 是一个 APDU。更多的详细信息,包括位/字节传输顺序,参见附录 D。

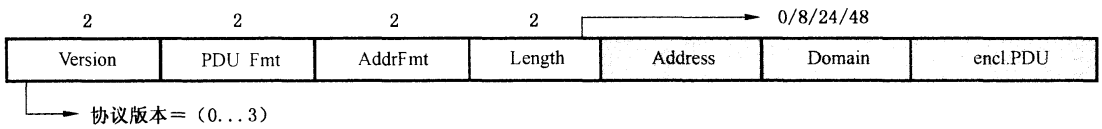


图 14 NPDU 格式

8.6 地址识别

每一个 GB/Z 20177 节点都执行地址识别。节点的地址识别取决于该节点的状态。在未配置状态下,所有的广播报文以及符合 Unique_Node_ID 的报文都被接收。在配置状态下,报文必须要么匹配 Unique_Node_ID,要么匹配域且是广播、组(多播)或单播报文。为了匹配域,域长度和域地址两者都必须匹配。对于广播报文,还必须匹配子网,除非广播报文是发送到整个域范围。对于组报文,则该节点必须是该组成员,即它的组地址之一必须匹配。对于单播报文,子网和节点号必须匹配。节点号的匹配是比较整个字节,即使节点号是定义成 7 个比特位的。

在本条中,每个节点必须保存要被识别的信息,但没有规定地址匹配算法,这一算法与特定的实现相关。这些数据结构具体实现的例子参见 A.17。

当节点接收到带有它所不支持的协议版本号的任何数据包时,丢弃该数据包。

当数据包的源子网/节点地址匹配当前域内的子网/节点地址时,丢弃该数据包。

当节点不在配置状态或硬离线状态,只有广播(域或子网范围)报文或匹配 Unique_Node_ID 的报文才被接收。接收事务的源地址是基于该报文被实际接收的域(柔性域)被保存。以 0x00/0x80 的子网/节点号的任何响应被回送到该域中。其实现要么所有的接收事务支持共享一个柔性域,要么每一个接收事务支持一个柔性域。

广东省网络空间安全协会宣传资料

8.7 路由器

路由器执行特定域内的路由功能,路由功能独立于网络层的地址识别功能。因此,路由器也有自身的网络地址,可以像其他节点一样被寻址,用于网络管理和诊断的目的。路由器连接两个子网,一个路由器是一个逻辑的而不是一个物理的实体,一个物理的包装可以装有多个路由器。路由器可由网络管理报文设置成四种工作模式:中继器、网桥、自学习和配置模式。当设置成自学习模式时,它将自动学习网络的子网拓扑结构。当设置成配置模式时,其路由表由网络管理报文静态配置,强制形成一个特定的拓扑并提供关于组编址拓扑信息。类似地,路由器也可以用这种网络管理报文来设置成网桥或中继器模式,网桥或中继器的两边可以存在相同的子网。

路由器使用三个路由功能函数:$ROUTE_{uc}()$,$ROUTE_{mc}()$,和 $ROUTE_{bc}()$来转发 NPDU。第一个函数指定如何转发编址到子网的 NPDU,第二个函数指定如何转发编址到组的 NPDU,第三个函数指定当广播时如何转发 NPDU。函数以表的形式出现,每一项有如下形式:

$ROUTE_{uc}(DestSubnet) = [forward|discard]$

$ROUTE_{mc}(DestGroup) = [forward|discard]$

$ROUTE_{bc}(DestSubnet) = [forward|discard]$

地址 X 的项指明编址为 X 的 NPDU 是要转发还是丢弃。这些函数在路由器的数据包接收方被调用,A.4 中的算法在路由器两侧独立地执行,并对到达该侧的数据包作出所有的路由决策。路由器的每一侧都有不同的一组表,保存有路由信息,决定是否允许该数据包通过该路由器。对配置型路由器,这些表是在该路由器的网络配置时被初始化,如果网络拓扑发生变化时必须及时更新。自学习型路由器通过观察数据包的源地址发现其子网表的内容(见 A.5),但不能学习组拓扑。因此,自学习型路由器总是将组报文转发到另一侧。

8.8 路由算法

路由算法的伪代码见 A.4。

8.9 自学习算法-子网

定义路由函数 $ROUTE_{uc}()$子网路由表是依据 A.5 的算法建立的。在初始化时，所有子网的地址被转发，算法接着通过观察被路由的 NPDU 源地址来学习子网的位置。再有，该算法在路由器两侧独立地执行。

9 事务控制子层

9.1 前提条件

这一节要描述的事务排序协议用 4 个比特位的事务号，它是由发送端分配，接收端用它来检查重复报文。该子层提供端到端的协议完整性，它通过每次只允许接收端处理一个数据包来保证，即使最初的数据包几次重试都被接收端正确接收。它基于这样的前提条件：网络是顺序保持的。如果不是的话，数据包在从源地址到目的地址的多条路径上产生的延时大致相同。从源地址到目的地址多条路径数据包传播的时间差异必须小于从该源地址完成一次事务(在最短路径上有一个确认)并开始下一个到同一目的地址的事务所需的时间。晚到的确认和响应被认为是重复的。但如果晚到的数据包是启动一个事务，并在一个后续事务开始后到达，则不被认为是重复的。

事务控制子层可能将事务递交到高层作进一步的处理。在某些实现方式中，上层可能永远不会对事务控制子层作响应。如果存在这样的情况，则事务控制子层可以保护自己免受上层影响。在事务控制子层等待上层响应的时间内，即使接收时钟超时，该条事务控制记录将被锁定不被解除。这样的状态只有在上层响应后才解除，或者，当没有更多的事务记录可用，但又有需要，于是一条已经被锁定的记录可被用于新的事务。这种机制的使用与实现方式有关。

一般来说，并发的出向事务限定为一条优先级事务和一条非优先级事务。这个限制是基于这样的事实，即确认/响应是对来自一个子网/节点编址的报文/请求还是来自唯一节点标识编址的报文/请求，是无法辨别的。实现时限制只使用这两种编址模式之一，可支持多个并发事务。接收事务在传输层和会话层共享，事务在每一层都可以进行重复检测。

在重复检测机制中，事务定时器、接收定时器、重发次数共同作用保证了系统的可靠性。这里假定接收定时器设置得足够长，可以覆盖全部的重发次数时间和。同时又足够短，事务 ID 不致于从头开始，以致一个新的事务被差错地检测为重复的旧事务。在实现中要求所有优先级事务有一个事务空间，所有非优先级事务有另外一个事务空间。如果一个发送者要发送一个数据包到给定的目的地，且发送到该目的地的上一条报文的事务 ID 等于当前的事务 ID，则将该事务 ID 加 1。一个可选的改进是对每一个目的地址都分别有优先事务和非优先事务空间。这时发送者就记下每一个目的地址所用的最后一个事务 ID。不管如何具体实现，都应保证避免不能分辨确认/响应是来自子网/节点编址还是唯一节点标识编址的情形。只要这种重复检测机制保持在所有出错情况下，在节点使用的所有编址模式中非常可靠，则其他的实现方式也是允许的。

9.2 提供的服务

事务控制子层负责与事务排列顺序、重复检测有关的公共功能。它提供的服务有：

——出向顺序控制。为保证出向的传输层报文和会话层的请求次序，事务控制子层控制发送事务号的分配。它限制到任何目的地的并发事务：小于或等于 1 个优先级事务，小于或等于 1 个非优先级事务。

——入向顺序控制和重复检测。事务控制子层提供重复检测。

9.3 服务接口

访问事务控制服务通过图 15 的接口描述。

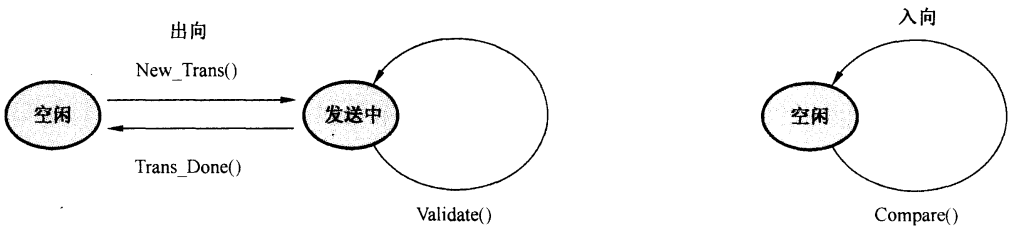


图 15　事务控制服务接口

接口原语的语法语义描述如下：

New_Trans (Priority,Dest_addr) -> (Trans_No)

用在获得一个新的出向事务的事务号

Validate (Priority,Trans_No,Dest_addr) -> (result)

这里,result = (current,not_current)之一,核实 Trans_No 在发送窗口中

Trans_Done (Priority,Trans_No,Dest_addr)

出向事务完成的通知

Compare (T1,T2) -> (result)

这里 result 是(new,duplicate)之一;定义 T1 和 T2 的相对关系,这里 T1 和 T2 是接收事务号。

广东省网络空间安全协会受控资料

9.4　状态变量

事务控制子层利用目的地记录来完成事务号码的分配,如下所示：

```
Transaction_CTRL_Record = record
  PriTX            True or False
  Trans_No:        (0..15); initial value = 0;
  In_Progress:     Boolean;
  end;
```

9.5　事务控制算法

A.6 中的伪代码算法是提供两个事务空间的一种事务控制版本,一种对应于所有优先级事务,另一种对应于所有非优先级事务。在所有实现中,对该实现方法所提供的每个事务空间,由 New_Trans()提供的第一个事务 ID 在复位后应当为 0。此后 New_Trans()在 1～15 的范围内增加该事务 ID,再从 1 开始。因此序列号 0 只有在复位后每个事务空间的第一个事务才会用到。

一般而言,所有事务控制算法均应遵循以下基本操作：

——当某节点需要用事务控制算法发送数据的时候,该发送节点选取一个事务 ID,然后再发送。

——当接收到一个数据包时,接收节点查找当前的接收事务,找出与该数据源地址、目的地址、优先级属性及事务 ID 相匹配的事务。

——如果没有找到匹配事务,则分配一个新的记录。

——如果除了事务 ID 外都匹配,并且现存的事务没有被锁定,那么就认为正在进行的事务处理已经成功完成,接收事务的记录可以被重新使用,即插入该新的事务 ID。只有分配给它一个事务记录后,相应的入向数据包才可以进行其处理过程。

——一旦接收者已经分配好一个事务记录，立即启动一个接收事务定时器。对于一个用唯一节点标识地址的报文，接收事务定时器的时间是 8 s。对于子网/节点地址的报文，或者组地址报文，接收事务定时器可以进行相应配置，该记录在定时器超时后会被重新分配，除非该事务记录被锁定(参见 9.1)。

——当节点接收到后来到的数据包与前面已经接收到的数据包事务 ID、源地址、目的地址、优先级属性都一致，则认为该数据包是一个重复报文。

——重复数据包依据协议的服务要求决定如何处理，但一般都不会交付到应用层，即使被交付到应用层，也要有重复标志的提示。

10 传输层

10.1 前提条件

除了依赖事务控制子层对 TPDU 序列进行正确的排序和重复检测外，传输控制协议没有其他的条件。

10.2 提供的服务

传输层提供下列服务：

a) 可靠的多播或单播。传输协议支持发送者在组内或发送者在组外的多播传输。如果服务要求确认或响应，对于发送者在组外的情况，传输层应被告知组的大小，其大小是组实际大小加 1。所有可靠的服务均具有以下属性：

——通过多次重试尽力确保可靠交付。

——假定第 4 层定时器均正确设置，除了发送者或接收者刚刚被复位的情况外，在所有情况下均提供了重复检测。复位节点的事务号应从 0 开始。这可能导致两个节点间进行中的事务完全不起作用，只是接收者对发送者发送确认或响应。

——部分排序——保持次序，但当报文经指定重发次数后交付仍然失败时，则不再提交。

——立即重新同步——紧随着网络的分割，交付最早发送的报文。

b) 无确认——重复的单播或多播。该服务和上述的可靠传输的主要区别在于没有确认信息，报文重复发送，直到最大重发次数。当应用这种服务时，没有组成员数为 64 的限制，应用这种服务时对组成员数目的唯一限制是一个域的节点数目。

GB/Z 20177 中的组在这个意义上是对称的，即组内的每个成员都既能发送也能接收。

10.3 服务接口

该层对会话层和应用层提供的服务接口方式如图 16 所示。

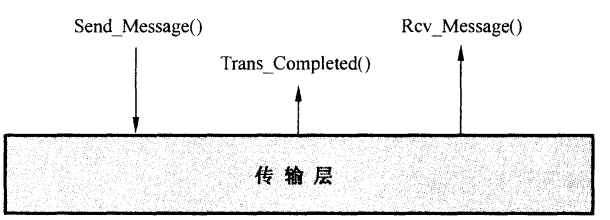


图 16 传输层和上层接口

传输层的接口语法和语义是：

Send_Message (Address, APDU, priority, ServiceType, Alt_Path) -> (TID)

Trans_Completed (TID, Result)

广东省网络空间安全协会受控资料

Rcv_Message (APDU)

其中 TID 是该事务唯一标识符。

10.4 TPDU 的类型和格式

TPDU 的语法如图 17 所示。每个字段上面的数字表示该字段长度位数。图中所示符号字段的值按次序映射到数值范围(0,1,2,3,…)上。更详细的内容,如位/字节的传输顺序,在附录 D 中定义。用于重发的定时器和计数器、重试次数和超时事务 ID,在 B.3.11 中描述。

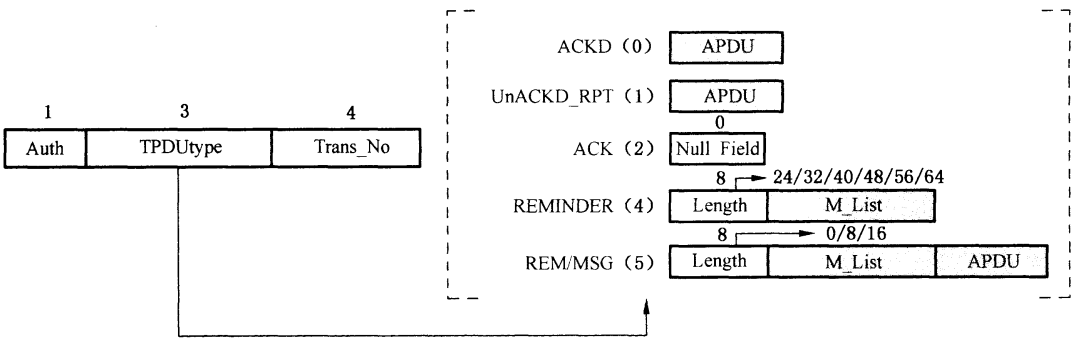


图 17 TPDU 的类型和格式

广东省网络空间安全协会受控资料

确认报文(ACKD)TPDU:(见图 D.1)被用在报文的第一次传输中,它使用编址格式 #1、#2a,并在某些条件下用格式 #3。与无确认 TPDU 报文不同,它必须得到所有被寻址接收者的确认。因此传输层提供链路层期望的确认数目并编入链路层报文首部。这种 TPDU 用在确认的初始 TPDU(单播和多播)及单播的提示中。多播提示将在下面描述。见图 E.2 或图 D.1 的编址格式。

无确认-重复报文(UnACKD_RPT)TPDU:与相应的确认报文相同,但有一点不同,即在接收到报文时没有确认返回到发送者。这种 TPDU 可以不加修改用在无确认-重复服务上。对这类报文,传输层提供一个信道使用统计增量,它等于对第一个报文的重试次数,而对该事务内所有其他后续信道使用统计增量为零。简单无确认报文只发送一次,没有 TPDU 首部,因此也没有重复检测。

报文提示(REM/MSG)TPDU:有助于对多播事务确认的选择请求。当接收到确认的发送者所在组的最高成员号小于 16 时,使用 REM/MSG 类型 5;这种 TPDU 包含成员列表(M_List[]),它是一个比特位数组,还包含 APDU。其长度字段指定 M_List[]字段大小(字节数)。比特位字段 M_List[X]值为 0 表示成员 X 的确认未被发送者接收到,为 1 则表示已经被发送者接收到。字节内的比特位从右到左分配,比特位 0 在右边。当长度为 0 时,M_List[]不出现,意思是所有成员都要确认。对该报文信道使用统计增量是尚未接收到的确认的数目。

类型 4:是一种简单的提示,它没有 APDU(见图 17)。它用在有确认报文的最高组成员号大于或等于 16 的情形。它使用 TPDU 对(REMINDER,ACKD)请求确认,该 TPDU 对逻辑上等同于单一类型 5 REM/MSG TPDU,它用在需要确认的成员可以以类型 5 的格式编码时。(这种解决方案用在大的组中,因为需要限制最大 TPDU 的大小)。这种报文信道使用统计增量总为 0,跟在传输后的 ACKD 报文将其信道使用统计增量设置成尚未接收到的确认数目。

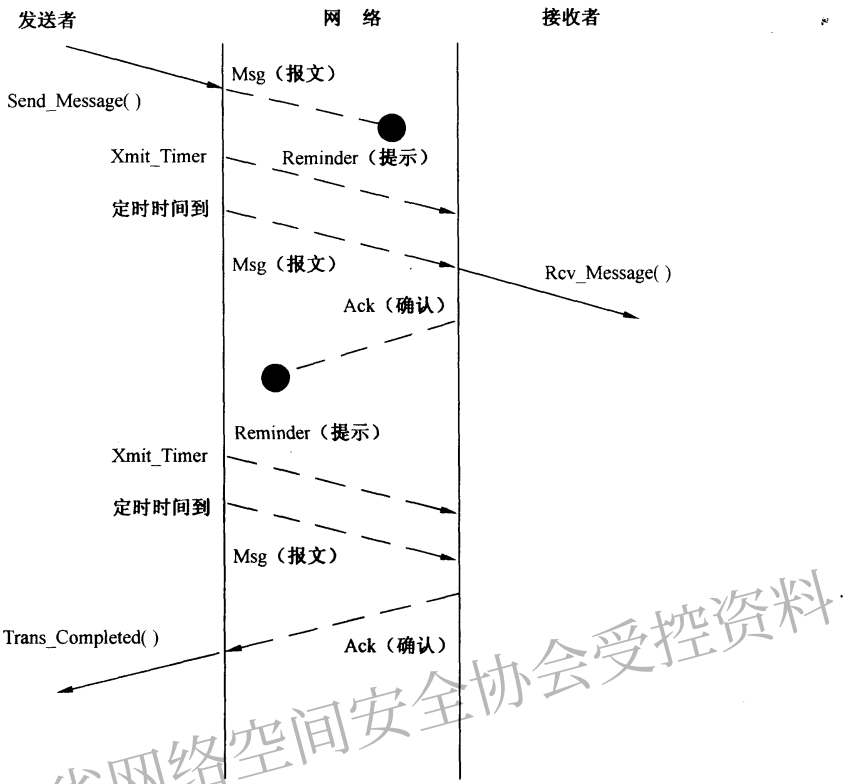
确认(ACK)TPDU:是空的,它采用 #2a 编址格式(单播确认)或采用 #2b 编址格式(多播确认)。Trans_No 字段表明被确认的事务,确认总是从原始报文中继承 priority、Alt_Path 等属性信息。

任何需要确认的 TPDU 都可以将 Auth 位置"1"来标记为需鉴别的数据包。

类型 3、6、7 保留,带这些编码的数据包都将被丢弃。

10.5 协议图解

图 18 对 10.7～10.8 的协议描述作更详细的说明，但并非取代它们。



广东省网络空间安全协会受控资料

图 18 传输协议图解：多播中报文和确认 TPDU 都丢失的情况

10.6 传输协议状态变量

传输协议包含两个独立的功能——发送和接收。为支持发送功能，传输层对每个正在进行的事务保留一个传输记录。一个共享的接收记录池便于报文的接收处理。发送和接收的数据结构见 A.7 和 A.8 的描述。

10.7 发送算法

图 19 是简化的发送 FSM，A.9 中描述了它的详细算法。每当接收到确认或质询时，算法将重试定时器(Xmit_Timer)复位，而不同于每次超时复位一次。在最后两次尝试时该层设置数据包的替换路径位并通知低层每次尝试的替换路径属性。

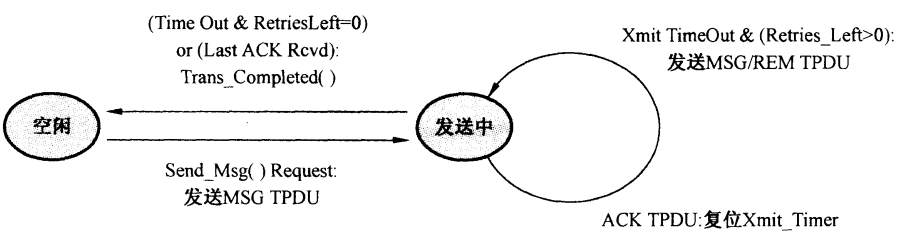


图 19 传输协议——发送 FSM

10.8 接收算法

报文接收使用基于时间的机制，图 20 描述了单个事务接收有限状态机，详细的算法在 A.10 中说明。

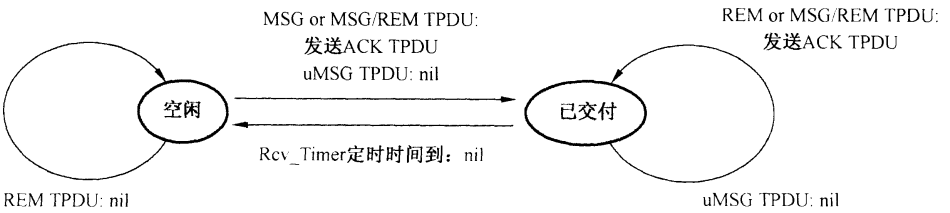


图 20　传输协议——接收 FSM

10.9　接收事务记录池的大小和配置

接收事务记录(RTR)池已经在 9.5 和 A.8 中定义，它在实现时可以是固定的，也可以在应用程序链接到协议代码时分配。接收事务记录池的大小限制节点并行接收操作的数目(并行是指在 Rcv_Timer 的作用时间内)，它应该根据接收定时器的时段内预计并行事务的数目进行配置。如果需要分配接收事务记录，但没有可用的空闲记录时，接收事务满统计值加 1(见 13.8.1)。

10.9.1　重试次数

重试次数应足够大以保证报文提交的成功率达到可接受的水平，如大于或等于 99%。但在 GB/Z 20177 中重试次数的上限为 15。

如果一次尝试发送的失败概率是 p，那么在一个有 n 个节点的组内小于或等于 k 次成功的概率按公式(16)、(17)计算(假定对所有的目标节点一次尝试的概率都为 p)。

$$P\{\text{no retry}\} = (1-p)^n \qquad (16)$$

$$P\{\leqslant k \text{ retries}\} = \sum_{i=0}^{k}\binom{k+1}{i} p^i (1-p)^{(k-i-1)} (1-p^{(k-i+1)})^{(n-1)} \qquad (17)$$

式中：

p——失败概率。

注：对于单信道 $p=(w/2+p_e)$，其中 w 是 MAC 层的随机窗口大小，p_e 是传输出错导致的数据包丢失的概率。

上述的概率情况在图 21 中描述。给定组的大小和失败概率 p，如成功率在 99.5%，那么就可以从图中得到要求的重试次数。

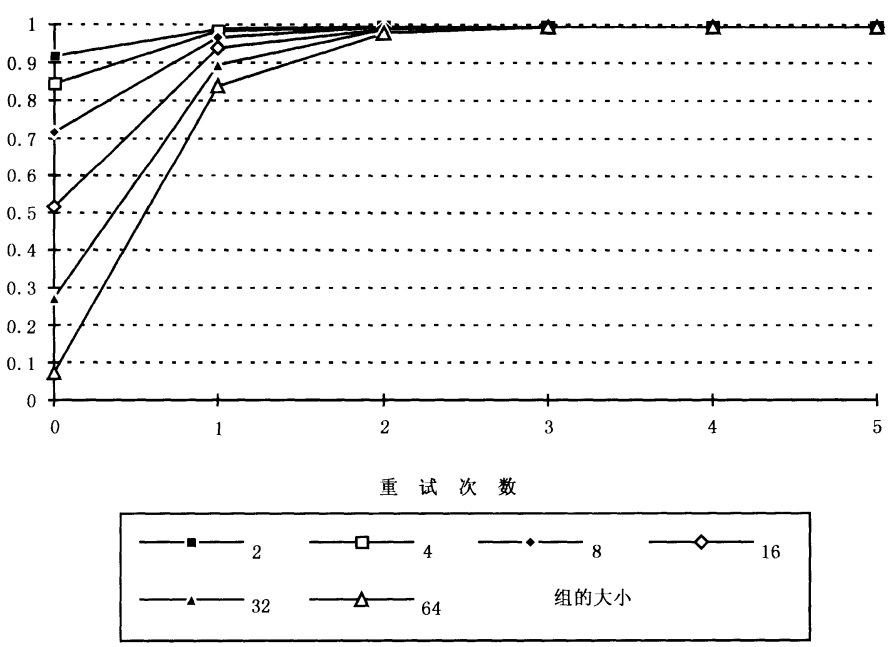


图 21　k 次重试事务完成的概率

广东省网络空间安全协会受控资料

在实际的系统设计中应考虑实效性。比如，对于周期性重复的事务就没有必要进行大量的重试操作。此外，对多播报文的确认会明显占有较大比例的信道带宽，无确认重复服务会比确认服务报文更有可能发送成功。实际上，在大部分情况 Retry_Count（重试次数）的取值范围为 2～5。

10.9.2 传输层定时器

传输层协议使用三种定时器，分别是：

Xmit_Timer	第 4 层重传定时器
Repeat_Interval_Timer	UNACKD_RPT（无确认重复）间隔定时器
Rcv_Timer	接收记录定时器（见 A.8 和 10.9）

这些定时器在每个事务期间是活动的。对发送方，Repeat_Interval_Timer 或 Xmit_Timer 是活动的，而对接收方，只有 Rcv_Time 是活动的。Xmit_Timer 在接收到每一个 ACK TPDU 或鉴别质询信息后复位，而 Rcv_Timer 则在每次接收到对该目的地节点的具有新事务号码的 MSG 或 MSG/REM TPDU 时复位。Repeat_Interval_Timer 决定发送者发送 UNACKD_RPT 报文的时间间隔。推荐的计算定时器值的方法如图 22 所示。这些建议是对单一信道中的单播事务的。对多信道网络，计算要根据路由器的速度和缓冲区数目。多播事务需要较长的接收事务值，因为多播事务即使不需要重试，完成也需要较长的时间。缓冲和时钟速率都是可调的，系统的设计者应采取相应措施合理设置每一个节点的定时器和重试次数。

Retry_Count = 2～5　　（见 10.9.1 和图 21）

Xmit_Timer≥3×packet cycle time+margin　　（margin = 最好情况下传输完成时间）

Rcv_Timer≥Xmit_Timer×(Retry_Count+2)

这里：

"3×packet cycle time"假定该信道上一般的站点必须等待 2 个数据包周期延迟后才能访问网络。然后，紧跟在网络访问后，是平均数据包传输时间（即数据包周期），加到定时器值中。最后，加上接收者处理该数据包并送出确认的时间。这是相关联节点时钟速率的函数。

中国电子信息产业集团公司网络空间安全协会受控资料

图 22　定时器值计算方法

所有使用 Unique_Node_ID 地址的报文使用 8s 的接收定时器，而不管非组型接收定时器的配置值是多少。所有非组型报文使用非组型接收定时器。

11　会话层

11.1　前提条件

除根据事务控制子层对 SPDU 正确排序和重复检测外，会话层不作其他假设。

11.2　提供的服务

会话层提供的服务为：

——请求/响应。该服务提供应用通信，类似于远程过程调用。尤其是，允许客户端对远程的服务器发出请求并接收对该请求的响应。

——非等幂的事务最多只执行一次（即：恰好一次或根本不执行）。非等幂事务是指那些依赖于先前状态的操作，比如：将阀门开度值增加 10%。只有响应长度小于或等于一个字节时，协议才将事务看作是非等幂的。

——响应的长度大于一个字节的请求则被看作是等幂的，这样的请求可能会被执行若干次（即零次或多次）。等幂事务是指动作可以被反复执行若干次，效果还是一样的。等幂命令的一个例子是"读前 10 个表值"。

——等幂和非等幂事务的区别在于响应的字节数，这样便于限制节点内所需存储的事务记录数量。

如果一个事务响应的长度大于1个字节，但执行可能不超过一次，那么应用程序负责保存响应并重发。对一个重复的请求，会话层向应用层提示。

——除非服务器响应在请求响应定时器设定的时间内产生或重传，否则认为请求响应事务失败（见10.8）。

11.3 服务接口

与应用层的服务接口的格式如图23所示。

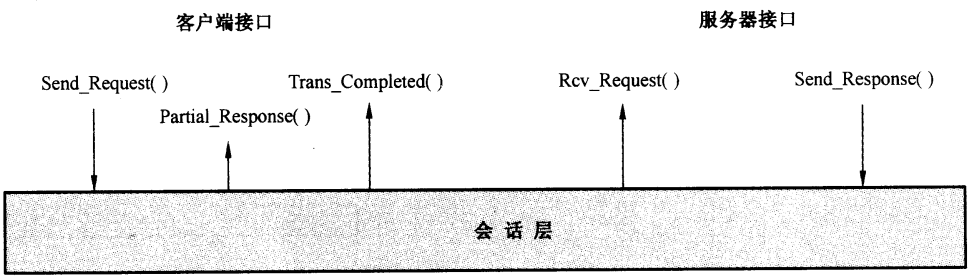


图23 会话层到应用层的接口

下面给出服务接口原语的语法。TID是该事务的唯一标识，详见A.6的实现方式细节。duplicate是一个布尔量，为真表示客户端正在重试一个先前执行的请求。Result也是一个布尔量，真和假分别表示成功和失败。

Send_Request (Address,APDU,priority) -> (TID) {客户端}

Partial_Response (TID,APDU,priority)

Trans_Completed (TID,Result)

Rcv_Request (TID,APDU,duplicate,priority) {服务器}

Send_Response (TID,APDU,priority)

广东省网络空间安全协会受控资料

11.4 会话层内部结构

会话层内部结构如图24所示。

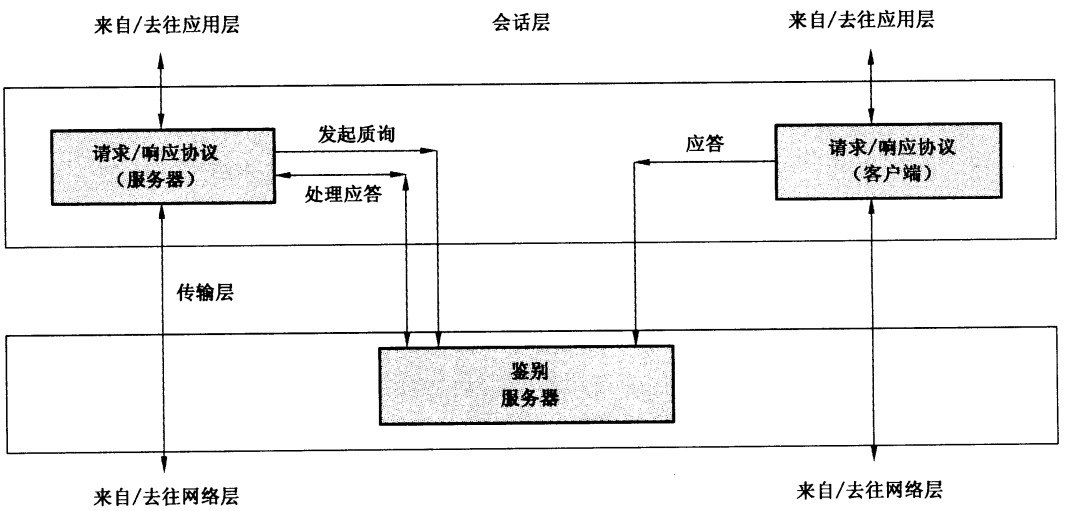


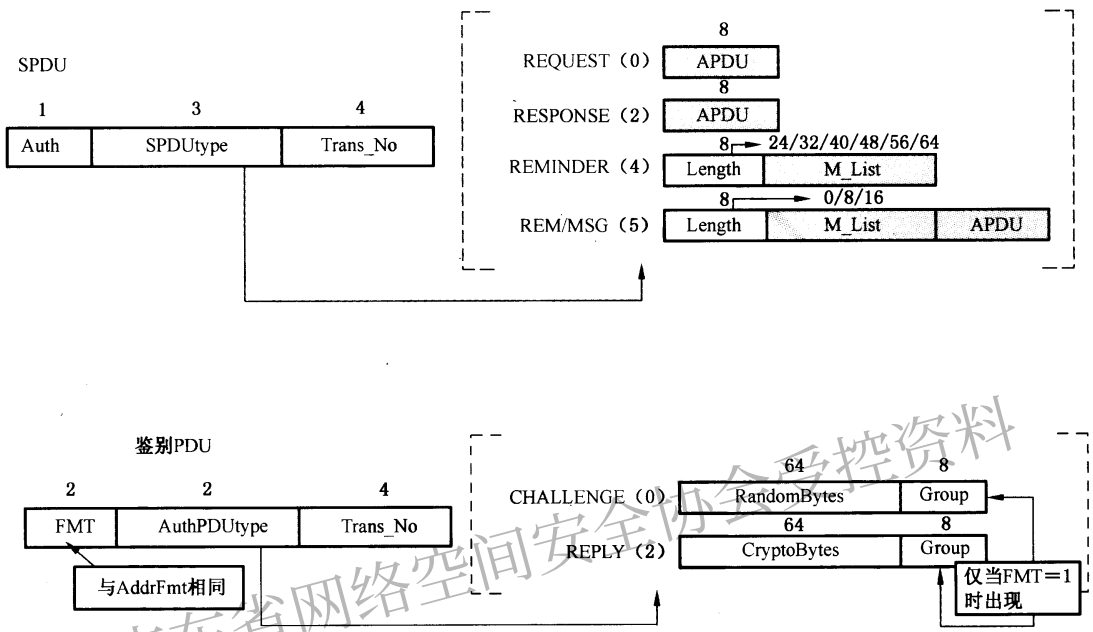
图24 会话层内部结构

请求响应协议通过下面三个调用访问鉴别服务器（详见11.11）。传输层也通过三个同样的调用访问鉴别服务器。

Initiate_Challenge(RR,PDU) -> (null)　　{质询者}
Reply(XR,PDU)-> (null)　　{被质询者}
Process_Request(RR,PDU) -> (pass/fail);　　{质询者}

11.5 SPDU 类型和格式

SPDU 格式如图 25 所示，其中每个字段上面的数字表明字段宽度的比特位大小，图中符号值按所示顺序与数值(0,1,2,3,…)对应。



广东省网络空间安全协会受控资料

图 25　SPDU 类型和格式

请求/响应协议使用三种基本 PDU 类型：请求(REQUEST)、响应(RESPONSE)和请求提示组合(REM/MSG)。这三种基本 SPDU 的语法(数据包格式)和语义(数据包处理)与 ACKD、ACK 以及 REM/MSG 的 TPDU 相近。注意：尽管会话层内部使用传输层一些功能特性，但在带有 SPDU 的数据包中不会另外出现传输层的编码特征，也就是说，一个数据包要么有 TPDU，要么有 SPDU。

在请求的第一次传输中使用请求(REQUEST)SPDU。请求(REQUEST)SPDU 采用编址格式#1、#2a，某些条件下使用格式#3 和格式#0(见图 D.1)。编址格式#0 是由特殊的网络管理命令使用的，用于广播一个请求来寻找尚未配置的节点。编址格式#3 则用于配置对这个特殊的网络管理命令响应的节点。会话层为较低层提供一个等于预期响应报文数目的信道使用统计增量。

请求提示(REM/MSG)SPDU：提供有选择的响应请求。在需确认的最多节点数小于 16 组中使用 REM/MSG 类型 5；这种 SPDU 包含为一个比特位数组的成员列表(M_List[])和 APDU(即请求本身)。长度字段规定(用字节数)M_List[]字段的大小。M_List[X]中的值 0 表示请求者没有接收到成员 X 的响应，值 1 表示请求者接收到响应。从右到左分配比特位，每个字节的最右端比特位为第一个比特位(bit 0)。用 REM/MSG 编码的信道使用统计增量等于尚未接收到的响应数。

类型 4：是一个不带有请求的简单提示请求(见图 25)。当需确认的最多成员数大于或等于 16 时，使用类型 4。在这种情况下，使用 SPDU 对(REMINDER-类型 4，REQUEST-类型 0)请求响应，并且在逻辑上该 SPDU 对相当于在小的组中所使用的单个类型 4 REM/MSG SPDU。(对大的组提供分离解决方案，因为需要限制最大 SPDU 大小。)最后，当长度=0 时，M_List[]字段不出现，其含义为“所有成员都应确认”。类型 4 REMINDER 报文的信道使用统计增量总为 0，而紧跟其后的 REQUEST 包含尚未接收到的响应数的信道使用统计增量。

响应(RESPONSE)SPDU:使用编址格式#2a(单播确认)或#2b(组确认)。Trans_No 字段承载正在被确认的事务。APDU 长度隐含定义了事务类型:如果该响应可被存放在 1 个字节中,那么认为该事务是非等幂的,否则认为该事务是等幂的。响应报文继承了对应请求数据包的 priority 和 Alt_Path 属性。

鉴别 SPDU(设置 Auth 比特位为"1"):表明接收者鉴别的请求。在所有其他方面,它们都与不需要鉴别的 SPDU 相同。

鉴别:鉴别服务器是一个第 4 层中所包含的服务器,可用于传输层和会话层协议。它提供了单向鉴别服务。当需要时,由客户端负责启动鉴别事务。它是通过在 SPDU 或 TPDU 中设置 Auth 比特位来实现。当接收到带有被设置 Auth 比特位的 TPDU 或 SPDU 时,服务器应使用"质询"AuthPDU 进行质询。应将鉴别 PDU 中的 FMT 字段值设置成与被质询报文中 NPDU 首部地址格式字段值相同。该客户端根据服务器的质询、该客户端发送的原始 APDU 和该客户的鉴别密钥计算出换算值。使用"应答"AuthPDU 向服务器发送换算值结果。应将该应答中的 FMT 字段值设置成与相应质询 AuthPDU 中的 FMT 字段值相同。当服务器收到该应答,将其内容与该服务器换算的值进行比较,如果相匹配,该事务通过鉴别。在所有情况下,不管该鉴别是失败还是成功,将该 SPDU/TPDU 传递到应用层与通知一起处理。注意:如果服务器节点的应用层不要求对特定事务进行鉴别,即使鉴别失败,那么它也可以选择接受该请求。如果应用层选择不接受该请求,那么它将简单地丢弃该 APDU 不作进一步处理。

11.6 协议时序图

图 26 和图 27 的协议时序旨在为会话层协议提供一个直观的感受并加强对 11.7～11.13 协议规范的理解(而不是取代这些章节)。注意:对于组成员数小于 16 时需要的确认,REM/MSG SPDU 被使用,并在功能上等同于(REMINDER,REQUEST)对。

广东省网络空间安全协会管控资料

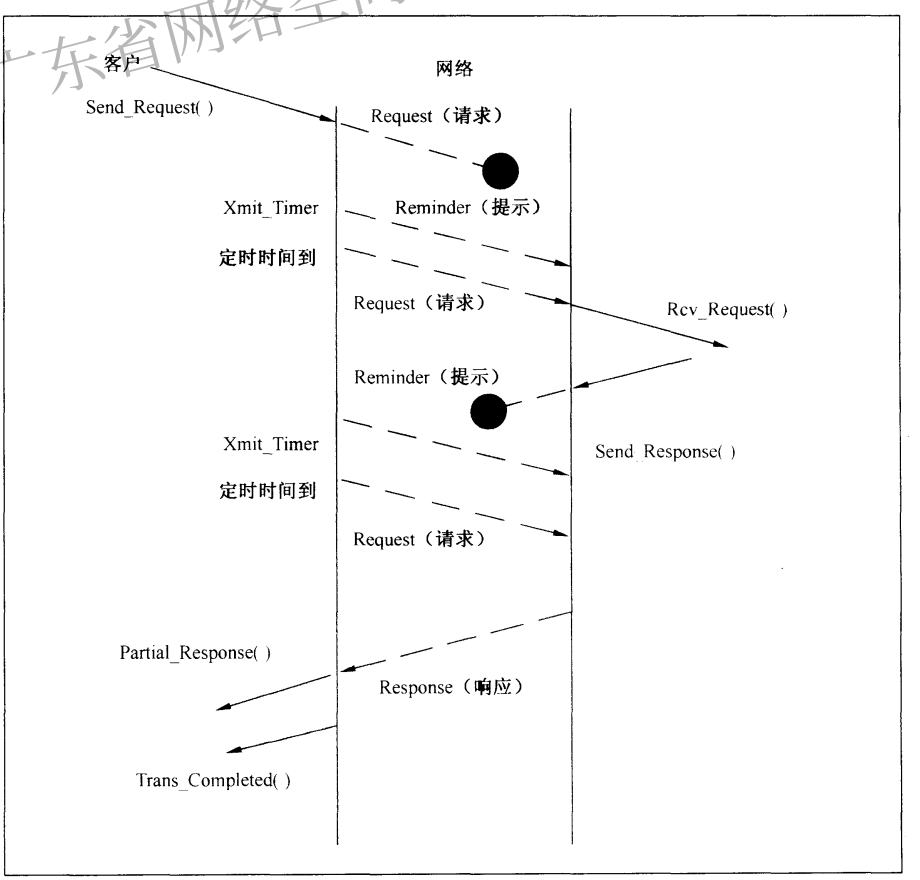
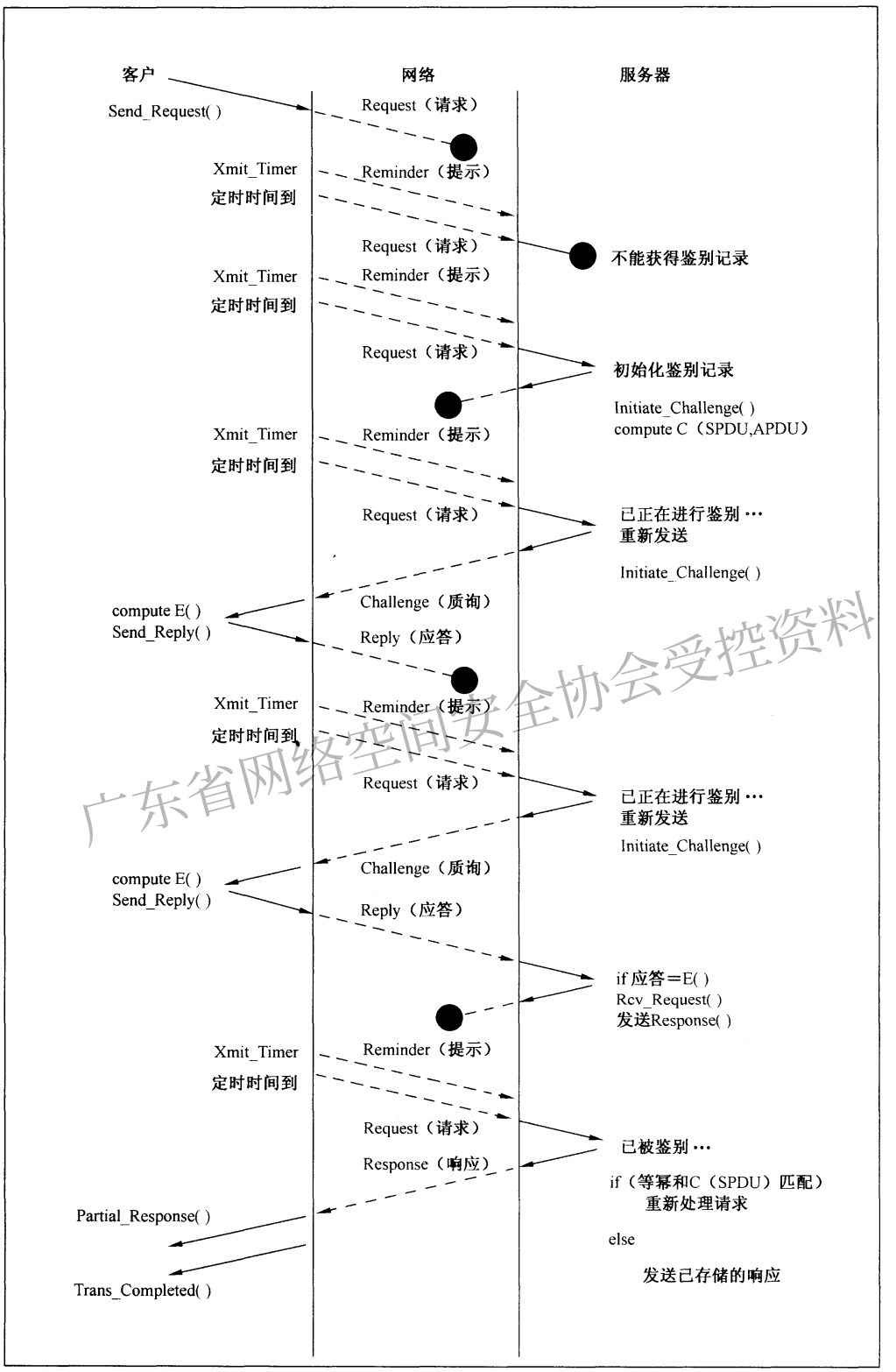


图 26 有多个 SPDU 丢失的非等幂请求



广东省网络空间安全协会受控资料

图 27　有多个 SPDU 丢失的安全等幂请求

11.7　请求/响应状态变量

如传输层协议那样，对每个正在进行的事务，请求/响应协议保存一个传送记录，并保存一个共享的接收记录池便于进行报文接收。见 A.11，这些记录与传输层协议使用的那些记录在细节上略有不同。

11.8 请求/响应协议——客户端部分

请求/响应协议客户端部分的简化的 FSM 如图 28 所示。详细说明见 A.12。

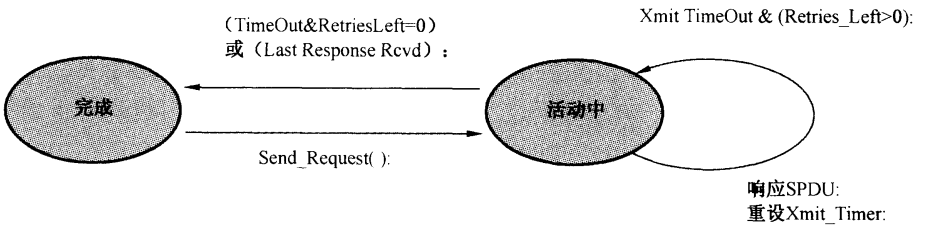
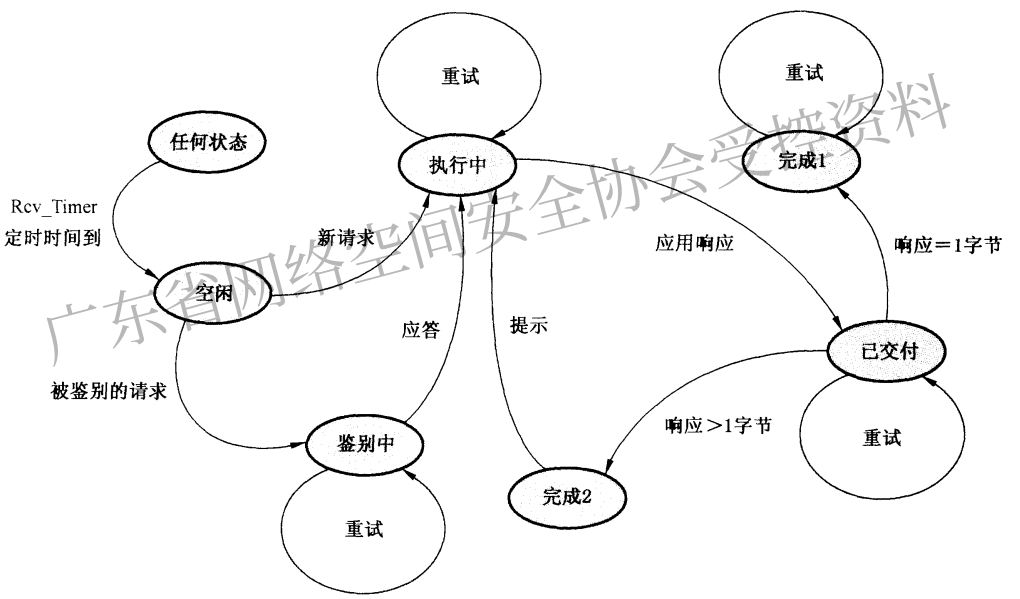


图 28 请求/响应协议——客户端 FSM

11.9 请求/响应协议——服务器部分

请求/响应协议服务器部分的简化的 FSM 如图 29 所示,完整的描述见 A.13。协议把所有响应大小>1 个字节的事务视为等幂的,意味着它们可能执行不止一次,响应大小为 1 个字节的事务执行不会超过一次。



广东省网络空间安全协会受控资料

图 29 请求/响应协议——简化的服务器端 FSM

11.10 请求/响应协议定时器

请求/响应协议使用的两个定时器——Xmit_Timer 和 Rcv_Timer,遵循传输层定时器的功能和格式,建议值与 10.9.2 中定义的传输层所建议值相同,例外情况是如果请求占了服务器上相当多的处理时间(相对于事务时间),在计算过程中应包括这个时间。

11.11 鉴别协议

鉴别服务器是在传输层实现的,它可被会话层和传输层访问。本节中讨论的是包括会话层需要的附加交互作用的解释,它依赖于事务控制子层中的重复检测机制,而不是其他的传输层服务。鉴别允许一个服务器验证请求者的身份。通常通过网络管理命令控制这种服务的使用,该网络管理命令规定被鉴别的报文/网络变量的改变。只有使用确认或请求/响应服务的事务可以被鉴别,在柔性域上收到的报文不被鉴别。仅当节点在配置状态或在硬离线状态时,网络管理鉴别比特位才被使用。在节点网络配置存储器上的一个校验和差错将导致节点进入未配置状态。

鉴别协议有两个不对称的部分:质询者和被质询者。质询者通过产生随机数字 X 初始化鉴别过

程；然后，被质询者利用 Y＝E(X,msg)进行响应，使用一个专用密钥对 X 和原始报文进行加密；最后，质询者将 Y 与它自身版本 E(X,msg)进行比较，根据比较的结果进行通过/失败的判定。A.14 中描述的鉴别算法定义了质询者和被质询者的函数。所有服务器的调用都是同步的。

11.12 加密算法

A.8 中详细描述的本文件加密算法提供单向编码，而不提供真正加密。加密算法使用一个 48 比特位的加密密钥 K、一个可变长度 APDU A[len]、以及一个 64 比特位输入字符串 R，生成一个 64 比特位输出字符串 Y。11.14 中定义随机数 R 的理想属性。任何 48 比特位的数字都可作为有效的加密密钥。

11.13 重试与校验和函数的作用

A.16 中定义的校验和函数被用于客户端重试时确认 APDU。如果任何原报文、质询、答复或确认/响应被丢失，那么客户端应重试。当收到一个重试时，服务器端所采取的操作是下列事务状态的一个函数：

等待中　　服务器正在等待鉴别记录。在这种情况下，服务器将设法再次分配记录。

鉴别中　　服务器已发出质询，并正在等一个应答。在这种情况下，服务器简单地重新发出相同质询(使用相同随机数)。

鉴别成功　　已完成鉴别交换，且带有成功确认。如果原报文被确认，那么确认被重新发出，并这次重试被丢弃。如果原报文是一个未知类型的请求，那么假定应用程序仍然正在组建响应，因此这次重试也被丢弃。如果原报文是一个非等幂请求，那么应答被重新发出，这次重试被丢弃。如果原来的报文是一个等幂请求，那么为节省内存，要么重试 APDU 将被保存，要么对它计算校验和，二者选其一。将结果与从原报文保存的内容(实际报文的内容或校验和)相比较。如果他们不匹配，重试就标记为未被鉴别。在任何情况下，将重试交付到应用程序。

鉴别失败　　已完成鉴别交换，且不带有成功确认。所采取的操作与“已被鉴别”状态相同，除非等幂请求不进行加密/比较。

注：在柔性域上接收到的报文不作鉴别。

11.14 随机数的生成

鉴别协议使用的随机数生成器应该具有下列性质：

a) 生成的数字 R 是数学随机数且是不可预测的；

b) 在电源故障或重新启动等事件之后，发生器不会产生可预测的值。

11.15 鉴别的用法

GB/Z 20177 鉴别模式必须被正确地使用，以提供最大安全性。用户应该知道，使用网络管理命令公开地进行密钥传输可能存在问题。可以通过使用鉴别密钥增量网络管理命令来解决这个问题(见 13.7.5 和 A.22)，而不是使用提供绝对密钥值的网络管理命令来解决这个问题(见 13.7.3 和 A.20)。

12 表示/应用层

12.1 前提条件

应用层除了依赖事务控制子层对 TPDU/SPDU 进行正确地排序和重复检测外，会话层不作其他假设。表示层提供的功能被规定作为 APDU 首部的一部分。特别地，当 APDU 首部指示 APDU 是网络变量更新时，首部含有表示层信息编码，以此告之节点如何去解释该 APDU 数据。

12.2 提供的服务

表示/应用层提供 6 种服务：

——网络变量传播。这种服务发送被接收方解释为网络变量更新的报文。一个特殊的两字节的首部用来传递表示层信息，指明该 APDU 要被解释成一个网络变量。网络变量使用任何协议服务传播。当网络变量被轮询时，使用请求/响应服务。

广东省网络空间安全协会受控资料

——网络变量别名。这种服务允许若干个网络变量输入或输出别名到一个基本的网络变量。从而当应用更新一个基本网络变量输出时，也为对应于基本网络变量的网络变量别名发送额外的数据包。当基本输入或别名输入从网络被更新时，应用接收一个关于基本变量已改变其值的事件。

——一般报文传递。一个应用可以构造一条任意的报文，使用任何寻址模式寻址。

——网络管理报文。这些报文在第 13 章中详细描述。

——网络诊断报文。这些报文一般被一个网络管理工具初始化，用来测试节点完全可操作并围绕有问题的区域采取纠正措施。这些报文在第 13 章中详细描述。

——外部帧传输。这些报文来自本文件环境的外部，并且目的地节点也在该环境的外部。提供这项功能，是把本协议用作这样两个外部节点之间的一个网关，或使用隧道技术使其他协议通过本协议。

12.3 服务接口

应用程序的服务接口形式如图 30 所示。

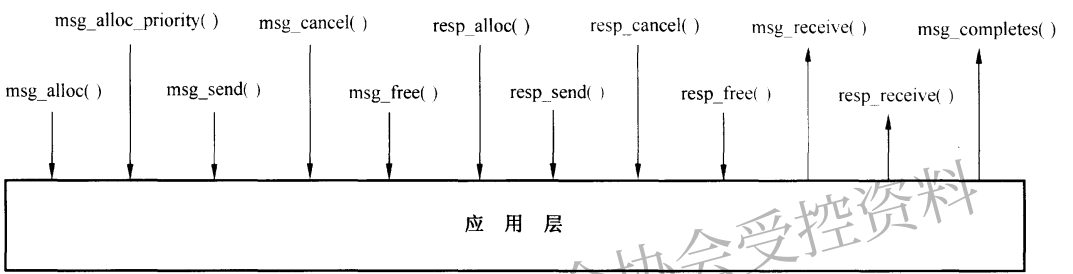


图 30 应用层接口

广东省网络空间安全协会受控资料

应用层接口的服务原语在下面的表 1 中描述。它们中的大多数不需要参数，其操作的数据结构 msg_out、msg_in、resp_out 和 resp_in 在 A.13 中描述。

表 1 应用层原语

服务接口原语	描　　述
msg_alloc()	该布尔原语为出向报文分配一个没有优先级的缓冲区。如果一个 msg_out 的数据结构被分配，它返回 TRUE；如果不能分配 msg_out 的数据结构，它返回 FALSE。如果它返回 FALSE，应用程序必要时可继续进行其他的处理，而不是等待一个空闲的报文缓冲区
msg_alloc_priority()	该布尔原语为出向报文分配一个有优先级的缓冲区。如果一个有优先级的 msg_out 数据结构被分配，它返回 TRUE；如果不能分配有优先级的 msg_out 数据结构，它返回 FALSE。如果它返回 FALSE，应用程序必要时可继续进行其他的处理，而不是等待一个空闲的优先级报文缓冲区
msg_send(msg_out)	该原语以 msg_out 数据结构发送一条报文
msg_cancel()	该原语取消当前正在构造的报文。它释放相关缓冲区，允许构造另一条报文
msg_free()	该原语为一条入向报文释放 msg_in 数据结构
resp_alloc()	该布尔原语为出向响应分配一个缓冲区。如果 resp_out 数据结构被分配，它返回 TRUE；如果 resp_out 数据结构不能分配，它返回 FALSE
resp_send(resp_out)	该原语以 resp_out 数据结构发送一个响应
resp_cancel()	该原语取消当前正在构造的响应。它释放相关的 resp_out 数据结构，允许构造另一个响应
resp_free()	该原语为一个入向响应释放 resp_in 数据结构
msg_receive(msg_in)	该原语接收一条 msg_in 数据结构的报文
resp_receive(resp_in)	该原语接收一个 resp_in 数据结构的响应
msg_completes()	当一条出向报文完成时(可能成功也可能失败)，该布尔原语的值为 TRUE

注：msg_out 和 msg_in 可以具有 12.4 中描述的任何格式。

12.4 APDU 的类型和格式

APDU 由一个首部和后面跟随的应用数据组成。首部是单个字节，只当在首部指定后面是网络变量信息时，后面有第二个字节跟随。APDU 的数据结构如下：

```
struct message
{
    byte destin_type;
    byte data[];
}
```

这里 data[]是一个未限制尺寸的数组，destin_type 取值如下：

00xxxxxx　　通用应用报文(64 种编码)。

1dxxxxxx　　一条网络变量报文；“d”指示方向：1 为出向，0 为入向。其余的代码位与第一个数据字节相结合形成一个 14 比特位的网络变量选择号。

011xxxxx　　一条网络管理报文(32 种编码)。

0101xxxx　　一条诊断报文(16 种编码)。

0100xxxx　　外部帧 (16 种编码)。

APDU 的其余部分按收到的第一个字节作为最左字节、收到的最后一个字节作为最右字节来定义。APDU 中存储的任何 2 字节或 4 字节的数据是按左起最高有效字节存储。在一个字节中最左边的位是最高有效位。数组是按左起最低编号的元素存储。结构字段也是按从左到右存储。

每个符合 GB/Z 20177 的节点将能至少接收到一个 16 个字节数据字段加上 destin_type 字段的 APDU。

应用层协议数据单元(APDU)的格式如图 31 所示。

广东省网络空间安全协会学习资料

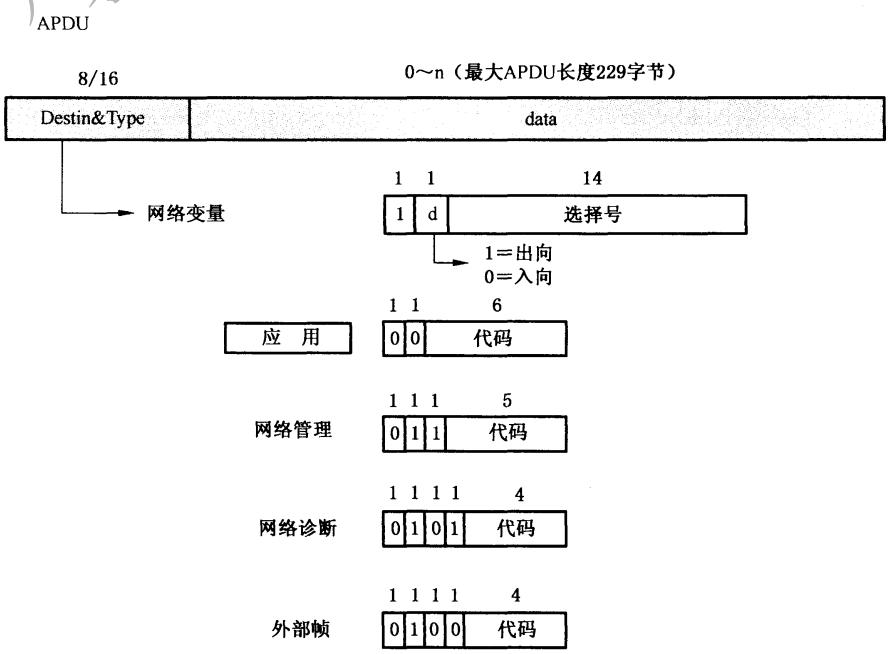


图 31　APDU 格式

网络管理报文编码的描述见 13.7；诊断报文编码见 13.8。

12.5 协议图

图 32 显示丢失初始 APDU 和 ACK TPDU 的非等幂多播事务。

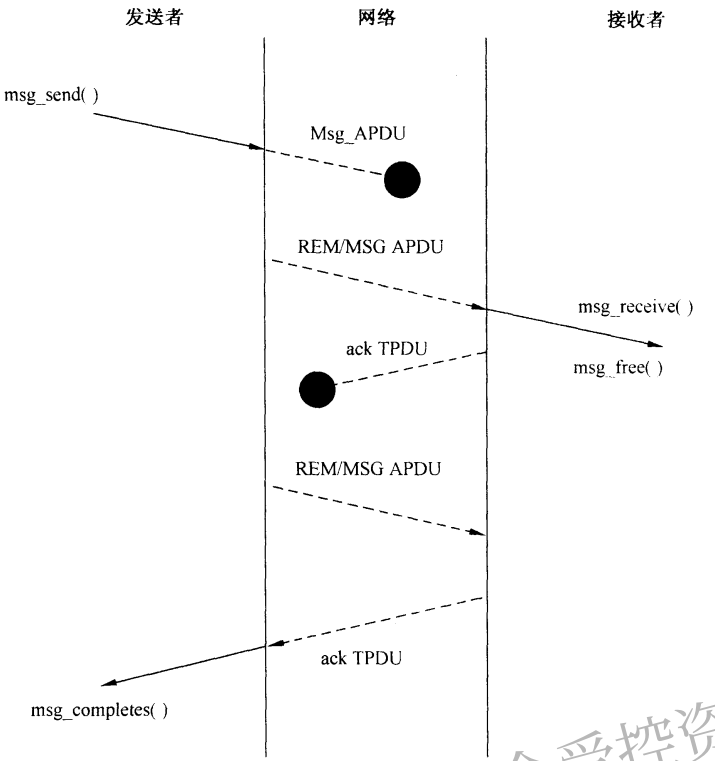


图 32　多播确认事务的应用层协议图

图 33 显示丢失请求和响应的等幂多播请求/响应事务。

广东省网络空间安全协会受控资料

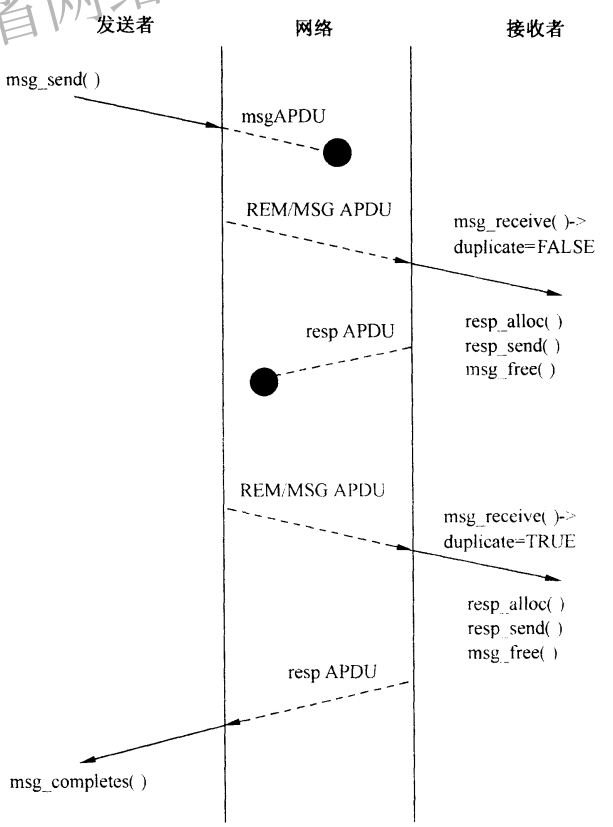


图 33　多播请求/响应事务的应用层协议图

12.6 应用协议状态变量

地址格式的数据结构在 A.13 中列出。msg_out、msg_in、resp_out 和 resp_in 结构使用这些地址格式引导报文到达它们的目的地。

12.7 离线状态中的请求/响应报文

当明确地或隐含地将请求/响应机制用于网络变量轮询时，有可能出现这种情况：发送请求给应用程序离线的节点，因而不能得到响应。当出现这种情况时，会发生什么事情取决于该响应是对网络变量轮询的响应还是对其他报文的响应。如果响应是对于一条应用程序报文，APDU 的 destin_type 将被设置为 63。当响应是对于一个外部帧时，APDU 的 destin_type 将被设置为 79。当响应是对于网络变量轮询时，该响应将包含网络变量选择号，但不包含任何相关的数据。如果试图轮询一个节点上的没有匹配选择号的网络变量，或者如果一个鉴别轮询失败，也会收到这样的响应。

12.8 网络变量

网络变量是一个节点上的逻辑输入和输出。可以在任何节点上定义多个网络变量，这样网络变量报文发送就允许特定的数据值从一个节点向一个或多个其他节点传播。网络变量作为一种特殊类型的应用报文实现，这里 APDU 最高有效位被设为 1。它们具有方向属性（输入或输出）、传播属性（被轮询、同步）和通信服务属性（优先级、确认、无确认、无确认重复、鉴别）。网络变量可能处于连接中。输入网络变量只有当从一个输出网络变量中接收到一个新值时，才用该新值更新。被轮询的网络变量只有当请求者给该节点发送一条包含该网络变量选择号的请求/响应报文时，其值才被传播。

网络变量根据应用的判断传播。例如，网络变量可能在被写或者被改变的任何时候被传播。在任何情况下，应用必须保证为一个多字节网络变量传播的数据在该应用内部一致。一个非轮询的输出网络变量可以是‘同步’的，在这种情况下，应用写的所有值将被传播到网络。反之，一个非同步的网络变量不要求赋给该网络变量的所有值实际传播到网络。应用如何传播非同步的网络变量依赖于实现方式。当使用更低层协议服务，如优先级、鉴别和确认等进行连接传播时，网络变量要被配置。

当发送方节点上的一个网络变量被配置为有一个选择号，该值与一个或多个接收节点上定义的网络变量的选择号匹配，并且该网络变量通过一个地址表项与一个或多个其他节点绑定，这时就定义了一个网络变量连接。一个节点可以把定义在它上面的一个输出网络变量连接到它自己的一个输入网络变量，这个连接被称为一个回转连接，即使该输出也通过网络被送到另一个接收方节点。

当节点处于离线状态时不发送网络变量报文。唯一的例外是当一个离线节点上的一个网络变量被轮询时，在这种情况下，节点用网络变量的合适的选择号响应该网络变量轮询报文，但是没有该网络变量值的数据。

网络变量可以有别名。一个别名的网络变量继承了被取别名的（基本）网络变量的长度、方向和类型。别名的网络变量总是有不同的网络变量选择号，并且可以有与基本网络变量不同的目的地址。这实际上是允许多个选择号与单个网络变量联系起来。单个网络变量选择号的限制可能会导致选择号冲突（即在单个节点上的多个变量使用相同的选择号）时，多个选择号对于网络变量连接是有用的。

12.8.1 网络变量处理

只要应用程序更新一个输出网络变量，应用层首先检查看网络上是否有该变量的任何读取者。这是通过检查有关联的地址表项的网络变量来完成的。如果有这样的一项，应用层就编码形成一个数据包，含有网络变量选择号和网络变量值，这个数据包然后送到更低的层进行处理。然后，应用层检查该变量是否有别名，如果别名存在，就用别名选择号与基本网络变量值一起形成数据包，每个这样的数据包也送到更低的层进行处理。

当网络变量更新到达正在接收的应用层时，选择号与接收方节点内的选择号比较。如果不匹配，更新被丢弃。如果匹配，则针对匹配的选择号作第二次检查，看它是否对应一个基本网络变量或一个别名。如果选择号对应一个基本网络变量，那么应用层更新该基本值。如果选择号是针对一个网络变量别名，那么应用层更新基本网络变量的值。

广东省网络空间安全协会受控资料

12.9 给应用程序的差错通知

不管应用程序是否正在使用网络变量或发送报文，它总是可以得到它最近一次事务的状态。

12.9.1 报文的差错通知

报文有三个事件与它们相联系：完成、成功或失败。完成意味着事务的结束（成功或者失败）。对于确认的事务，成功被定义为所有的确认都已收到。对于请求/响应事务，成功被定义为从所有预期的接收方接收到了响应。在这种情况下，要靠应用程序检查响应代码以了解预期的接收节点不是在离线状态。对于无确认的事务，当事务完成时（报文被送出）事务就成功了。对于无确认重复的事务，当报文被送出要求的次数时，事务完成并且成功。

本部分没有为无确认的事务定义任何失败事件。然而，特定的实现方法可能对于无确认事务，由于一条报文传输期间可能发生的内部差错，造成的失败事件进行标记。在没有收到期望的确认次数时，确认的事务提供失败通知给应用。当没有收到一个或多个响应时，请求/响应事务失败。

12.9.2 网络变量的差错通知

对网络变量来说，当事务完成时完成事件被标记，这与报文相同。还有，和报文类似，由于在发送的节点上检测到差错条件，无确认的更新能够标记失败事件。对于确认的网络变量更新，成功被定义为所有期望的确认都已收到，失败被定义为没有收到一个或多个期望的确认。当网络变量被轮询时，总是使用请求/响应机制。当目标节点返回了所有值时，被轮询的网络变量更新成功。当发生下列情况之一时应用标记一个失败事件：

a) 所有响应的目标没有有效的数据（无匹配的网络变量或离线），或；

b) 一个或多个期望的响应没有到达。

注意，对别名而言，规则 b)同时适用于针对基本和别名网络变量的响应。

规则 a)有一个例外：一个被轮询的网络变量，既连接到同一节点的另一个网络变量，又连接到其他节点的一个或多个网络变量，如果所有其他节点对该轮询都返回离线响应，就给应用标记一个失败事件。即使该节点的回转连接成功地应答它自己的轮询，也标志一个失败事件。

广东省网络空间安全协会受控资料

13 网络管理和诊断

13.1 前提条件

网络管理和网络诊断（NM/ND）服务是运用会话层的应用级的进程。这意味着只有当会话层（以及所有以下各层）正常工作时，网络管理和诊断才是可能的。

有一些例外，所有的网络管理和网络诊断命令以这样或那样的方式检验或修改存储器特定区域的内容。驻留在非易失性存储器中各种各样数据元素的一部分，例如地址分配，在协议处理器内由它们各自的网络管理和网络诊断命令支持，进行报告、更新和允许更可控地执行这些操作，其他的区域能使用读写存储器命令的特殊寻址模式进行读或写。这样，移动或变更命令类型的用户就不必关注他们自己非易失性存储器的物理分布。需要下载应用程序的那些用户必须了解非易失性存储器的物理分布（尽管该信息可能完全包含在一个下载文件中）。

13.2 提供的服务

网络管理和诊断服务提供下列功能：

——地址分配：所有地址成分的分配（除了 Unique_Node_ID）；

——节点查询：查询节点状态和基本的统计信息；

——路由器表的维护。

有一些小的例外，网络管理操作是作为远程过程来实现的，远程过程使用由会话层提供的低层请求/响应服务，见 11.5。

13.3 网络管理和诊断应用结构

网络管理是一个分布式的应用，有多个客户端和多个服务器。服务器功能必须被所有节点支持，而

客户端功能仅仅需要作为网络管理设备的节点的支持。

13.4 节点状态

节点可处于四种状态之一，这些状态保存在非易失性存储器中，它的值在状态名字后面的括号中列出。在对网络诊断命令中的 Query Status(查询状态命令)的响应中报告出这些状态。括号中的数字是 node_state(节点状态)的枚举值(见 13.8.1)。

Applicationless(无应用状态)：(3) 应用程序还没被装入，正在装载应用程序的过程中，或者由于应用程序校验和差错认为应用程序是损坏的。在这个状态没有应用程序运行。节点状态指示灯(一个基于 GB/Z 20177 的节点上的可选择利用的诊断帮助工具)连续亮。

注：节点状态指示灯的标准颜色是黄色的(资料性注释)。

Unconfigured(未配置状态)：(2) 应用程序已装载，但网络配置存储器或者是没有装载，或者是正在重新装入，或者由于网络配置存储器校验和差错被认为是损坏的。GB/Z 20177 特定的实现可能允许一个程序让节点进入未配置状态，由程序决定一个应用程序是否处在这种状态下运行。节点状态指示灯以每秒钟一次的频率在闪烁。(应用程序启动到未配置状态的转变将清除鉴别密钥。)

Hard-offline(硬离线状态)：(6) 标志应用程序已装载但没有运行。在这个状态网络配置存储器的内容认为是有效的；网络管理鉴别标志位是有效的。节点状态指示灯关闭。

Configured(已配置状态)：(4) 正常的节点状态。应用程序正在运行，网络配置存储器的内容被认为有效。这是唯一的应用层接收报文的状态，在所有其他状态，应用层报文被丢弃。节点状态指示灯关闭。已配置状态有另外的修改符，即在线/离线条件，该条件未必需要保存在非易失性的存储器中，状态和在线/离线条件经由不同的机制控制。但是，它们在状态命令中一起被报告。

注意，称一个节点在未配置状态或已配置状态和称一个节点已配置或未配置有细小的区别。节点在未配置状态或已配置状态如上所述。然而，如果节点在硬离线状态或在已配置状态(每一个情况都有有效的网络配置存储器内容)，则节点被称为已配置。如果节点在无应用状态或在未配置状态(每一个情况都无有效的网络配置存储器内容)，则节点被称为未配置。仅当节点在硬离线状态或在已配置状态时，网络管理鉴别标志位是有效的。

广东省网络空间安全协会反控资料

13.5 网络管理服务的用法

大多数网络管理 PDU(NMPDU)在会话层请求与/或响应时被传送。在缺省情况下，NMPDU 继承了 SPDU 封装中的请求或响应的属性。然而，使用 TPDU 的命令是这样表示的，一些请求是在 TPDU 而不是在 SPDU 中传送。一些命令，例如 Query_ID( )，可以使用广播寻址和请求/响应服务发送。当至少有一个响应从网络返回时，这样的一个命令就成功了。

当这样配置时，为了起作用，大多数网络管理或网络诊断事务必须被鉴别。对于使用 Unique_Node_ID 寻址的报文，当服务器和发起请求的客户端不在同一个域时，鉴别是不可能的。在这种情况下的网络管理或网络诊断命令不要求执行鉴别。

如果节点不理解一个特定网络管理请求，它将返回一个失败的响应。见 13.7 的网络管理响应编码。

13.5.1 寻址注意事项

客户端节点的传输事务定时器值必须被扩展到能够处理涉及改变非易失性存储器的任何命令的长延迟。当使用 Unique_Node_ID 寻址时，服务器节点自动地将非组接收事务定时器值扩展到大约 8s。这允许该定时器根据正常应用流量调节，不需要担心过长的网络管理事务。

在一个节点上最初使用这些命令的推荐寻址模式是 Unique_Node_ID。一旦节点被指定了一个适当的非组接收定时器值(用于重复检测)、域、子网和节点字段，则推荐使用子网/节点寻址模式。

无论从哪个域送出，以 Unique_Node_ID 寻址的报文都能被接收。未配置的节点不论其所在域，也将接收任何子网或域范围的广播。在这两种情况下，在收到报文的域用源子网/节点对值为 0/0 返回确认和响应。报文被不是域中成员的节点收到(节点未配置或不在该域中)被称为报文在柔性域上被接

收。某些命令在这些情形下是不允许的，它们在 13.7.1 开始的网络管理命令描述中指出。

使用 Unique_Node_ID 为网络管理命令寻址的一个重要优点是，如果节点偶然变成未配置（例如，在改变网络配置存储器时，由于电源问题导致的一个校验和差错），网络管理工具不至于失去它与该节点通信的能力。

13.5.2 网络配置变更

在一个节点内的网络配置变更流程范例如下：

a) 改变节点状态或条件（可选）；

b) 进行单个或多个改变；

c) 更新配置校验和（仅仅在第 b）步未做时需要）；

d) 如果还有需要去改变，转步骤 b）；

e) 如果在步骤 a）变化了，恢复节点状态或条件；

f) 如果通信参数改变了又想让它们生效，重新复位节点。

13.5.3 下载一个应用程序

下载应用程序的流程范例如下：

a) 使节点离线；

b) 改变节点状态到 Applicationless（无应用状态）；

c) 发送报文给节点，把应用程序装进该节点的存储器；

d) 使节点复位；

e) 计算应用程序校验和；

f) 进入未配置状态。

此时，可以装载网络配置存储器。注意，当装载一个应用程序后，跟着装载网络配置存储器，节点处于离线的状态下。

山东省网络空间安全协会受控资料

13.5.4 差错处理条件（资料性）

有几类差错需要考虑：

事务失败：如果事务失败（即未收到期望的确认或响应），最好是设法得到一个已知的状态而不是简单地重试该事务。如果网络管理鉴别已打开，返回一个已知的状态应该包括尝试使用不同密钥（例如，当前的密钥，以前的密钥等）进行鉴别事务直到成功。

节点复位或上电周期：如果当一个网络管理命令在进行时节点复位，该复位使节点可能表现为自身存在通信问题或事务失败。当涉及非易失性存储器写时，取决于实现的技术，在复位时正被修改的位置的内容可能被破坏。

在存储器刷新命令期间的复位可能导致网络配置存储器或应用程序的损坏。任何一个问题都可能是灾难性的，这取决于网络管理工具认知的范围。这里的一种选择是在网络管理工具中放置早期掉电检测，只在电源表现稳定时（假设客户端和服务器共享一个公共电源）发布刷新命令（不带重试）。

违反读/写保护：如果节点被读/写保护，试图写到应用程序代码区就会被拒绝。因此通过读 read_only_data（只读数据结构）结构的 read_write_protect 字段，客户端能验证试图写存储器失败。

其他可能产生问题的情况，如地址表和域的改变，都需要客户端小心地处理和理解，例如，一个请求能产生一个新的域地址的响应。

每个网络节点维持两个校验和，一个在网络配置存储器上，另一个在应用存储器上。随着改变网络配置存储器的任何网络寻址命令的结束，计算出一个新的配置校验和并进行更新。这增加了这些命令执行的时间，客户端在把下一条报文送到目标节点前应该考虑到这一点。该延时总是按非易失性存储器的写入时间乘以改变的字节数来考虑，每字节的延时可长达 20 ms，因此一条更新地址（update address）命令的传输事务的超时时间就应该至少为 $20\times5+30$，即 130 ms。

要注意自动更新校验和的命令。

某种实现方式可以选择对一个给定网络管理命令的参数进行有效性检查。并非所有网络管理命令的发送方实际上发送的命令都有准确的长度，这使得对命令长度进行有效性检查更加困难。网络管理命令的发送方不能超过以下所列长度。

对一个无效网络管理或网络诊断命令的响应应该是一个失败的响应。长度有效性检查基于下面的描述。注意，如果对于一个给定的命令列出了不止一个数字，那么对较小的数字有效的参数集合如果被填补到较大的数字中也必须被接受。通过设置命令参数为 0xFF，发布退出命令，退出命令和规则在本章和附录 A 的数据结构中提及。

命令	代码(16 进制)	数据大小	
Query Status	51	0	
Proxy Agent	52	12	唯一节点标识寻址
Proxy Agent	52	6	所有其他地址格式
Proxy Target	52	1	
Clear Status	53	0	
Report Transceiver Sts	54	0	
Query ID	61	1	
Query ID (conditional)	61	$5+n$	n=conditional 字节数
Respond To Query	62	1	
Join Domain	63	16	
Leave Domain	64	1	
Security	65	7	
Modify Address	66	5	只用于回转
		6	只用于非回转
Report Address	67	1	
Report NV	68	1	只用于非退出
		3	只用于退出
Update Address	69	5	
Query Domain	6a	1	
Modify NV	6b	4	常规更新
		5	别名更新
		5	空的别名更新，基本变量=0xFF
		6	退出常规
		7	退出别名
		9	双退出别名
Node Mode	6c	1	不用做状态改变
		2	用做状态改变
Read Memory	6d	4	
Write Memory	6e	$5+n$,16	n=写入的字节数
Checksum Recalculate	6f	1	
Install	70	0	基本的应用闪烁
Install	70	1	对多个堆栈/节点
Install	70.	9	网络变量定义
Install	70	5	网络变量删除

广东省网络空间安全协会受控资料

Install	70	7	查询网络变量自文档内容
Install	70	4	所有其他的查询网络变量命令
Install	70	5	查询节点信息
Install	70	20	更新网络变量名称
Install	70	7+*N*	更新网络变量自文档内容
Install	70	7	更新网络变量速率估算
Install	70	5	更新网络变量 SNVT 索引
Memory Refresh	71	4	
Query SI Data	72	3	
NV Fetch	73	1	只用于非退出
		3	只用于退出

注：回转意指设备的输出 NV 连接到该设备的输入 NV。非回转意指该连接不是设备的输出 NV 与该设备的输入 NV 的连接。

13.6 路由器网络管理命令的用法

路由器要遵循正常的协议处理器“状态”，并且必须处于应用/已配置状态以便完全作为一个路由器来运行。

所有影响路由表的命令仅仅影响路由器的一侧。用于 OFFLINE(离线)、ONLINE(在线)和 RESTART(重启)的网络管理节点模式命令将自动影响路由器的两侧。

对路由器来说，ONLINE 意味着路由器将按描述正常运行，OFFLINE 意味着路由器不进行转发；出现在数据缓冲循环队列中的、非寻址到该路由器的所有数据包被丢弃，除了丢掉这些数据以外，OFFLINE 路由器继续正常运行。

路由器将忽略某组网络管理命令，这些命令是广播节点模式命令。忽略这组命令是为了防止广播的重启或 OFFLINE 命令使路由器停止，并防止同样的广播命令到达该路由器另一侧的目的地。这样，在需要时路由器就必须被单独重启或使之离线。为了做到这一点，可能必须要为路由器发起一个唯一的手动服务请求。

广东省网络空间安全协会受控资料

13.7 NMPDU 格式和类型

本条使用类似于 C 语言结构定义的符号，列出了网络管理 APDU(NMPDU)。在附录 D 中定义的比特位和字节的顺序规则适用情况是：每个字节最高有效比特位首先被传送；一个记录的第一个字节被认为是该记录最低有效字节。在描述中的数值部分，该数值对应报文的一个命令号或响应代码。

所有 NMPDU 的第一个字节包含目的地/类型数据，对于网络管理请求和命令而言它总是(二进制的)：

011xxxxx

<xxxxx>字段包含命令代码。

执行这些网络管理命令所产生的响应指向应用程序，这由 APDU 的第一个字节指定：

00pxxxxx

如果操作成功，<p>字段设为 1，如果失败设为 0。失败通常是由于超出范围(表的边界)的差错或非易失性存储器写失败。<xxxxx>字段返回原来的网络管理命令代码。

所有网络诊断报文的 APDU 的第一个字节都包含目的地/类型数据：

0101xxxx

<xxxx>字段包含命令代码。

网络诊断响应有下面的格式，这里<p>字段和网络管理响应中一样，<xxxx>字段反映原来的命令：

00plxxxx

这隐含着所有网络管理或网络诊断请求都被递交到网络管理或网络诊断层，同时所有网络管理或网络诊断响应被递交到应用层。这里我们假定响应要在应用层获得处理。

在本文档中仅仅描述了命令字段值，<p>字段和目的地代码没被包括，但是被假定在正确位置。

对于提供给网络管理操作的单字节的响应，被认为是非等幂的。

注：13.7 中对 8 比特项用“byte”，对 16 比特项用“uint16”。

13.7.1 查询标识(Query ID)

Query_ID( )请求一个节点或一组节点向请求者报告其 Unique_Node_ID。典型情况下，这种请求在单个域上作为子网范围内的广播寻址，意味着客户端至少知道该域，并且可能为了询问一组节点正在子网地址内按序探查。使用的数据结构在 A.14 中描述。

要查询未配置的节点，查询标识请求(Query_ID_Request)记录内的选择号设为‘0’。为了查询其“查询响应”位已被设置(见后面的命令)的节点，选择号设为“1”。要查询未配置且其“查询响应”位已被设置的节点，选择号设为“2”。这个命令通常使用子网或域范围内的广播寻址模式。它从来不要求执行鉴别。

如果提供，地址和数据字段被用作附加的限定符。地址模式和地址字段被用来形成一个地址(见该处理过程描述的“读存储器”部分)；从那个地址开始的数据“计数”字节(1～11)与提供的数据进行比较。只有当它们匹配时查询标识才继续进行(按“选择号”指定的方式)。

仅仅对这个命令，假定读是始终允许的。如果地址和计数落在一个读保护的区域里(例如在那儿存储有鉴别密钥)，则没有响应返回。

13.7.2 查询响应(Respond to Query)

这个命令用来设置或清除目标节点中的“respond to query”比特位。在该比特位被设置时，目标将对选择号为‘1’的 Query_ID( )请求做出响应，节点在这个模式下一直持续到节点被复位或通过命令清除该标志位。这条命令被用于询问网络拓扑，“on”版本通常按子网广播寻址，使用无确认的重复服务。而“off”版本一旦被查询，则寻址到特定的节点。这个命令从来不要求执行鉴别。数据结构在 A.15 被定义。

广东省网络空间安全协会受控资料

13.7.3 更新域(Update Domain)

这个命令使用 A.16 中的数据结构，更新服务器的域的一个表项。注意，该节点字段中最高有效比特位必须要置位。这条命令的执行更新网络配置存储器校验和。如果节点只能在单个域中，试图分配域索引‘1’将返回一个差错。如果被更新的域与发出修改报文的域相同，并且节点在“已配置”状态，那么，响应将返回到新的域，这样发送方就不能收到。这个命令也可以被用来离开一个域，可通过将域长度设为 0xFF，把子网和节点地址设为零，并且让鉴别密钥无效来实现。注意，使用该命令离开一个域时不会造成节点复位和进入未配置状态的副作用，而这些影响在使用 Leave Domain(离开域)命令时会自动产生。

因为加密密钥以明文传播，所以这个请求只应该在物理网络安全能够保证(或者安全通过其他的方式可以获得)时才使用。

13.7.4 离开域(Leave Domain)

即使在这域内还有分配的地址，节点也必须支持这个命令。在内部，节点的域长度被设为 0xFF，子网和节点地址被设为零。另外，鉴别密钥被清除。在这个命令的执行期间网络配置存储器校验和要被更新。如果要离开的域就是接收请求的域，并且节点是已配置状态，则没有响应送出。如果要离开的域是已配置节点的最后一个域，则节点自动地进入未配置状态，并复位。使用的数据结构在 A.17 中给出。

13.7.5 更新密钥(Update Key)

这个命令用于更新加密密钥，使用 A.18 中的数据结构。要使用的域在报文中指定。密钥各字节按字节方式加到已有的密钥中(不进位)。网络配置存储器校验和被该命令更新。

13.7.6 **更新地址(Update Address)**

这个命令按索引执行。如果该地址表项地址不存在,在应答中的<p>比特位应为零。网络配置存储器校验和被这个命令更新。不对重复地址或组执行交叉检验。

地址表中的每个地址表项的格式在 A.19 中描述。

13.7.7 **查询地址(Query Address)**

这个命令根据一个给定的索引号报告出节点地址表中的一个表项,使用的数据结构在 A.20 中描述。

13.7.8 **查询网络变量配置(Query Network Variable Configuration)**

这个命令根据索引号报告出节点网络变量配置表(nv_config table)中的表项地址,该表项地址在表中必须存在,使用的数据结构在 A.21 中列出。

这个命令也能用来查询节点的别名表。用于别名查询的数据结构与网络变量查询的数据结构相同(见 A.21)。当用于别名时,命令数据结构中索引是别名表项的索引,主索引是对应被索引别名的基本网络变量的索引。

13.7.9 **更新组地址(Update Group Address)**

这个命令被用来更新在一个地址表中的一个组表项;它一般是按组寻址的。组的大小、定时器索引和重试计数器都要被更新,组成员字段保留不变。表项更新是基于接收命令的域,因为组地址只在特定的域内是唯一的。因此,该命令不允许在柔性域使用。该命令更新网络配置存储器校验和,使用的数据结构在 A.22 中。

这个操作仅仅更新该组地址中第一个匹配的实例。

广东省网络空间安全协会受控资料

13.7.10 **查询域(Query Domain)**

这个命令被用来检索一个节点上两个域中一个域的域信息。如果请求第二个域,而只有一个域存在的空间,则响应包含命令失败指示。使用的数据结构在 A.23 中。

13.7.11 **更新网络变量配置(Update Network Variable Configuration)**

这个命令被用来根据索引号增加或修改来自或到达节点网络变量配置表(nv_config table)的表项地址。网络变量配置表中必须有用于该表项的剩余空间。地址表索引可以被设为 0~15,其中 15 指明没有地址表表项与该网络变量相联系。网络配置存储器校验和被该命令更新。对于索引为 X 的网络变量,网络变量选择号为(0x3FFF−X),它暗示网络变量不在一个逻辑连接中(即没被绑定)。使用的数据结构在 A.24 中。

这个命令也能被用来更新节点的别名表。用于别名配置表更新的数据结构与网络变量配置表更新的数据结构相同(见 A.24)。当用于更新别名配置表时,命令数据结构中索引是别名表项的索引,主索引是对应于被索引别名的基本网络变量的索引。

13.7.12 **设置节点模式(Set Node Mode)**

这个请求指示应用程序进入离线或在线状态,通过一个内部复位命令复位整个节点,或改变节点的状态。当离线时,应用程序被挂起,网络管理或网络诊断命令继续得到处理。在线请求指示应用程序调度器离开离线状态,继续应用程序的运行。离线状态的一种用法是在下载应用程序非易失性存储器时,暂时停止应用程序的运行。

用于这个命令的服务类型是变化的。对于在线和离线而言,没有响应返回,因此不能使用请求/响应。虽然接收方可以使用请求/响应服务接受一个请求,但是在线和离线状态的变迁阻止响应被送出。状态改变的确认是通过发布一个状态请求命令获得的。对于状态变化,应该使用请求/响应。既然状态是应用程序的一部分,应用程序校验和就要被更新。对于复位而言,只使用无确认的服务类型(如果要求鉴别的话则使用确认的服务类型)。这条报文要用一系列网络管理命令确认,首先是节点的状态被清除,然后向节点发出复位命令,最后向节点发出状态请求命令。注意,确认复位的失败可能表明最初的无确认报文被丢失,必须重试该交换过程。使用的数据结构在 A.25 中。

13.7.13 读存储器(Read Memory)

这个命令被用来读存储器。如果读/写保护被设置,只有 read_only_data、config_data(配置数据)和 RAM 或非易失性存储器数据区能被读。

"计数(count)"字段包含要读的字节数,这个数字不应该超过 16,除非目标节点有足够大的缓冲区来容纳额外的数据。要读的地址可以指定为只读数据结构(见 B.1)的相对地址、配置数据结构(见 B.6)的相对地址、统计数据结构(B.7)的相对地址,或可以是绝对地址。能被读的绝对寻址的存储器空间长达 65 535 个字节,从实现时特定的物理地址偏移量开始。在绝对地址 0 读一个字节时,返回原始系统映像版本号。当固件版本号指定它是一个定制的版本时,这能用来决定系统的能力。定制的版本假定来自原始系统的映像,并且继承原始系统的能力。使用的数据结构见 A.26 中。

13.7.14 写存储器(Write Memory)

这个命令有两种形式:一种形式在写后复位协议处理器,另一种不复位。复位形式的确认必须通过使用读存储器读回存储内容来完成。非复位形式产生响应并且通过请求/响应传送,复位形式使用无确认或无确认-重复的服务传送。网络配置存储器校验和与/或应用校验和可选择性地被更新。一个符合 GB/Z 20177 的节点不要求支持跨越存储器技术的写存储器命令,例如,单个写命令不需要既支持写 flash EPROM 存储器又支持写 EEPROM 存储器。

如果设置了读/写保护,则仅有 config_data_struct(见附录 B)可写。字节数应少于 11,除非目标节点拥有足够大的缓冲区来处理更多的数据。写入的地址可以指定为配置数据结构(见 B.6)的相对地址、统计数据结构的相对地址(见 B.7)或可能是绝对地址。和读存储器命令类似,绝对地址可以是从应用程序特定物理地址偏移量开始的65 535 字节范围内。所使用的数据结构参见 A.27。

13.7.15 校验和重计算(Checksum Recalculate)

这个命令强制协议处理器计算和存储新的网络配置存储器或应用程序的校验和。这个命令应该在改变网络配置非易失性存储器映像或应用程序非易失性存储器/RAM 映像的任何网络管理会话结束时使用(除非那些命令已经特别执行了这个操作,如地址命令)。所有的节点都将计算其网络配置非易失性存储器的校验和。应用程序映像的校验和是可选的,不计算应用程序校验和的节点当通过此命令重新计算其应用程序校验和时,总是报告校验成功。使用的数据结构见 A.28。

13.7.16 安装(Install)

这个命令在各种目的的安装期间使用。主要目的是强制执行应用特定的 wink(闪烁)功能,例如可用闪烁功能闪烁连接到节点上的指示灯以确定节点的标识。闪烁功能如果已经实现,即使节点在非配置状态闪烁功能都将被执行。

这个命令的第二个目的是允许多个协议处理器共存于一个应用中。每个协议处理器都有其自己唯一节点标识和相联系的域和地址表。该命令允许为安装目的个别地查询协议处理器。使用的情形是通过发出手动的服务请求取得唯一节点标识来安装应用,然后这个命令用于查询应用有多少个协议处理器与它相关联。对于每个请求,应用都回复一个包含在手动服务响应报文中的下一个协议处理器的信息,以及网络接口号。一旦所有的网络接口被鉴别,获取下一个网络接口的命令都将失败。这个命令很少使用,由于 GB/Z 20177 所支持的更广范围的协议处理器能力,将来可能也不会使用。

本命令的第三个目的是支持网络变量的动态(在运行/安装的时候,而不是在编译的时候)创建和删除。由于资源所限,对这个特性的支持是可选择的。作为安装命令的扩展,这些命令提供了查询自标识/自文档(SI/SD)数据、更新自标识/自文档数据的方法,和通知节点增加了一个新的网络变量或删除了一个现有网络变量的方法。对所有三种目的的安装命令的一般格式是:

<应用命令> <应用特定的数据字段>

对这个命令响应的第一个字节应该是指示命令成功的 0x30,或者指示命令失败的 0x10。如果有响应的其他字节,取决于应用命令的类型。应用命令的值为:

APP_WINK (0)　　命令应用执行一些可视的或可听的动作

广东省网络空间安全协会交流资料

APP_INSTALL (1)	查找一个应用的所有网络接口
APP_NV_DEFINE (2)	创建一个新的动态网络变量定义
APP_NV_REMOVE (3)	删除一个现有动态网络变量的定义
APP_QUERY_NV_INFO (4)	查询一个网络变量的自标识/自文档数据
APP_QUERY_NODE_INFO (5)	查询节点的自标识/自文档数据
APP_UPDATE_NV_INFO (6)	更新一个网络变量的自标识/自文档数据

下面是按命令类型的对应用特定的数据字段的定义。

APP_NV_DEFINE

这个命令增加一个新的或修改一个已有网络变量的定义。如果企图增加或修改的节点固定网络变量接口部分的网络变量的定义，或者变量索引超出了节点支持的最大值，那么节点将报告命令失败。

一个成功的定义之后，应用程序负责将网络变量配置成缺省值(非绑定的选择号、正确的方向等)，这样后续的网络变量配置查询命令就可以显示最新定义网络变量的正确的缺省属性。

报文的数据结构包含下列字段：

APP_NV_DEFINE (2)		如上述定义
unsigned	nvIndexHi	网络变量索引的高位字节
unsigned	nvIndexLo	网络变量索引的低位字节
unsigned	arrayLenHi	网络变量数组中元素个数的高位字节 如果网络变量不是数组，则值为 0
unsigned	arrayLenLo	网络变量数组中元素个数的低位字节 如果网络变量不是数组，则值为 0
unsigned	nvLen	网络变量值中字节的个数 (1～31)
unsigned	nv_dflts	网络变量缺省的可配置的属性，编码为单个字节 包括方向、优先级、鉴别和服务类型
unsigned	nv_attr	网络变量的属性，编码为单个字节(参见下面的定义)

广东省网络空间安全协会资料

APP_NV_REMOVE

这个命令删除一个或多个已有的网络变量定义。网络变量索引必须在指定为动态的接口区域内。如果指定的网络变量索引在节点的固定接口区域内，则命令将失败。即使指定的网络变量集目前未定义或者超过了节点支持网络变量的最大个数，命令也将成功。本命令包含下列字段：

APP_NV_REMOVE(3)		如上述定义
unsigned	nvIndexHi	被删除的第一个网络变量定义的网络变量索引的高位字节
unsigned	nvIndexLo	被删除的第一个网络变量定义的网络变量索引的低位字节
unsigned	nvCountHi	删除网络变量个数的高位字节
unsigned	nvCountLo	删除网络变量个数的低位字节

APP_QUERY_NV_INFO

本命令查询特定网络变量的自文档数据。包含下列字段：

APP_QUERY_NV_INFO (4)		
unsigned	nv_info	请求的自文档数据。当前值为：
NV_INFO_DESC (0)		网络变量的基本属性、数组信息、网络变量如何定义和可附加自文档的指示

	NV_INFO_RATE_EST (1)	为网络变量定义的平均和最大速率估计
	NV_INFO_NAME (2)	网络变量名
	NV_INFO_SD_TEXT (3)	和网络变量相联系的自文档文本字符串
	NV_INFO_SNVT_INDEX (4)	标准网络变量类型(SNVT)的索引
unsigned	nvIndexHi	网络变量索引的高位字节
unsigned	nvIndexLo	网络变量索引的低位字节
unsigned	offset_hi	如果 nv_info 是 NV_INFO_SD_TEXT,则为请求数据偏移量的高位字节
unsigned	offset_lo	如果 nv_info 是 NV_INFO_SD_TEXT,则为请求数据偏移量的低位字节
unsigned	length	如果 nv_info 是 NV_INFO_SD_TEXT,则为请求的字节数响应的数据结构取决于请求的数据

如果请求的数据是 NV_INFO_DESC,则发送下列响应:

unsigned length	:5	网络变量值的长度,以字节为单位。如果网络变量目前未定义,则编码为 0
unsigned origin	:3	网络变量如何被创建。为下列值之一:
	NV_UNDEFINED (0)	网络变量目前未定义
	NV_STATIC (1)	网络变量是静态定义的(不能被删除)
	NV_DYNAMIC (2)	网络变量是动态定义的(可以被删除)
unsigned	nv_dflts	缺省可配置的网络变量属性,编码为单个字节。包括方向、优先级、鉴别和服务类型
unsigned	nv_attr	网络变量的基本属性
unsigned	nv_exten	扩展位。可用的附加自标识/自文档数据的指示符
unsigned	nv_array	网络变量数组属性
Unsigned	nv_name[16]	可选字段。如果有,在 ext 字段的 nm_supplied 比特位必须被设置,网络变量的名字不带数组下标

如果请求的数据是 NV_INFO_RATE_EST,则发送下列响应:

unsigned nv_rate_est	对该网络变量的平均速率估计编码
Unsigned nv_max_rate_est	对该网络变量的最大速率估计编码

如果请求的数据是 NV_INFO_NAME,则发送下列响应:

unsigned nv_name[16]	网络变量的名字。如果网络变量是一个数组的一部分,则名字不包括数组下标。如果长度小于 16 个字符,则以“0”终止

如果请求的数据是 NV_INFO_SD_TEXT,则发送下列响应:

unsigned length	返回的自文档文本数据的字节数
unsigned text[*]	自文档文本数据段。实际的文本数组大小取决于前面的长度值

如果请求的数据是 NV_INFO_SNVT_INDEX,则发送下列响应:

unsigned snvt_type_index	与网络变量相关的标准网络变量类型(SNVT)的索引。如果网络变量不是 SNVT,则编码为零

APP_QUERY_NODE_INFO

这个命令查询节点的自文档数据。包含下列字段:

广东省网络空间安全协会受控资料

APP_QUERY_NODE_INFO (5)

unsigned	node_info	被请求的自文档数据。值为：
	NODE_INFO_SD_TEXT (3)	与节点相关的自文档文本字符串
unsigned	offset_hi	如果 node_info 是 NODE_INFO_SD_TEXT，则为请求数据偏移量的高位字节
unsigned	offset_lo	如果 node_info 是 NODE_INFO_SD_TEXT，则为请求数据偏移量的低位字节
unsigned	length	如果 node_info 是 NODE_INFO_SD_TEXT，则为请求的字节数

应答数据结构取决于请求的数据。如果请求的数据是 NODE_INFO_SD_TEXT，则发送下列响应：

unsigned	length	返回自文档文本数据的字节数
unsigned	text[*]	自文档文本数据段。实际文本数组大小取决于前面的长度值

APP_UPDATE_NV_INFO

这个命令更新特定网络变量的自文档数据。一般地，更新报文不能靠目标应用程序进行有效性验证。报文发出者必须执行所有的检验。这个报文包含下列字段：

APP_UPDATE_NV_INFO (6)

unsigned	nv_info	更新的自文档数据。当前的值为：
	NV_INFO_RATE_EST (1)	对该网络变量定义的平均和最大速率估计
	NV_INFO_NAME (2)	网络变量名
	NV_INFO_SD_TEXT (3)	与网络变量相关的自文档文本字符串
	NV_INFO_SNVT_INDEX (4)	标准网络变量类型的索引
unsigned	nvIndexHi	网络变量索引的高位字节
unsigned	nvIndexLo	网络变量索引的低位字节

其余的数据字段取决于指定的 nv_info 值。如果指定的 nv_info 是 NV_INFO_RATE_EST，则提供下列数据：

unsigned	clear_mre :1	如果置位，则清除网络变量的最大速率估计值，并指示这个数据不可用
unsigned	clear_re :1	如果置位，则清除网络变量的速率估计值，并指示这个数据不可用
unsigned	update_mre :1	如果置位，则更新网络变量的最大速率估计值，并同时指示最大速率估计数据是可用的
unsigned	update_re :1	如果置位，则更新网络变量的速率估计值，并同时指示速率估计数据是可用的
unsigned	nv_rate_est	对网络变量的平均速率估计编码
unsigned	nv_max_rate_est	对网络变量的最大速率估计编码

如果指定的 nv_info 是 NV_INFO_NAME，则提供下列数据：

unsigned	nv_name[16]	网络变量的名字以“null(0)”终止。名字不包括数组下标

如果指定的 nv_info 是 NV_INFO_SD_TEXT，则提供下列数据：

unsigned	length	被更新的自文档文本数据的字节数

广东省网络空间安全协会受控资料

unsigned offset_hi 被更新数据偏移量的高位字节

unsigned offset_lo 被更新数据偏移量的低位字节

unsigned text[*] 自文档文本数据段。实际文本数组大小取决于前面的长度值

如果指定的 nv_info 是 NV_INFO_SNVT_INDEX，则提供下列数据：

unsigned snvt_type_index 与网络变量相关的标准网络变量类型(SNVT)的索引。如果网络变量不是 SNVT，则编码为零

在前面的报文定义中引用了下列数据结构：

nv_attr 这个数据结构描述了特定网络变量的基本属性。包含下列字段：

unsigned nv_sync :1 如果置位，则赋给网络变量的所有值都以其原始的次序传播。与 nv_polled 互斥

unsigned nv_polled :1 如果置位，输出网络变量的值仅在响应轮询时才被发送。当应用程序更新网络变量值时，不产生更新

unsigned nv_offline :1 如果置位，在更新网络变量值之前，节点应离线

unsigned nv_service_type_config :1 如果置位，则网络变量的服务类型属性是可配置的

unsigned nv_priority_config :1 如果置位，则网络变量的优先级属性是可配置的

unsigned nv_auth_config :1 如果置位，则网络变量的鉴别属性是可配置的

unsigned nv_config_class :1 如果置位，则网络变量是一个配置网络变量

unsigned snvt_type_index 与这个网络变量相关的标准网络变量类型(SNVT)的索引。如果网络变量不是标准网络变量类型(SNVT)，则为零

nv_exten 这个数据结构指示对一个特定网络变量哪些附加的自文档数据是可用的。包含下列字段：

unsigned mre :1 如果置位，则网络变量的最大速率估计是可用的

unsigned re :1 如果置位，则网络变量的平均速率估计是可用的

unsigned nm :1 如果置位，则网络变量的名字是可用的

unsigned sd :1 如果置位，则网络变量具有一个自文档的文本字符串

unsigned nm_supplied :1 仅应用于对 NV_INFO_DESC 的 APP_QUERY_NV_INFO 响应。如果该位置 1，则在响应中，随已有数据字段同时提供 nv_name

nv_array 这个数据结构为特定的网络变量提供数组属性。包含下列字段：

unsigned count_hi 数组中网络变量的总数。一个 16 位的无符号数

unsigned count_lo

unsigned element_hi 网络变量数在数组中的索引。如果当前的网络变量

广东省网络空间安全协会受控资料

unsigned element_lo 不是数组的成员，则编码为零

nv_dflts 这个数据结构为特定的网络变量提供可配置属性的缺省值。包含下列字段：

unsigned nv_direction :1 网络变量的缺省方向

unsigned nv_auth :1 网络变量的缺省鉴别设置

unsigned nv_priority :1 网络变量的缺省优先级设置

unsigned nv_service :2 网络变量的缺省服务类型。

使用的数据结构在 A.29 中描述。

13.7.17 存储器刷新(Memory Refresh)

这个命令使节点重写指定位置中非易失性存储器指定字节数的已有内容。也可用于周期性地重写非易失性存储器的内容来延长存储器内容的保持时间。如果请求了一个存储器刷新但在指定地址不存在任何内容，则返回一个差错。如果“地址偏移量”超出了非易失性存储器的区域的末端，也将返回一个差错。这样，命令的发出者就可以简单地增加偏移量直到返回一个差错为止。如果目标是在线的，则次数应限定为 8；如果目标是离线的，则次数可高达 38。使用的数据结构在 A.30 中。如果非易失性存储器仅允许用扇区的方式写入，则次数和偏移量值须用完整扇区数代替字节数。

13.7.18 查询自标识(Query SI)

这个命令用于检索节点及其网络变量的自标识数据，如果自标识信息位于协议处理器可寻址的存储空间之外(如在协处理器的存储空间)时，在 B.1.1 中描述的数据结构中，字段“si”是指向节点的自标识数据地址的指针。Null(0)地址表示节点不提供自标识数据。不包括全 1 的非空地址指向数据所在的地址。在这种情况下，直接用读存储器命令将数据读出来。包含全 1 的地址指针表示自标识信息位于协议处理器可寻址的存储空间之外。查询自标识命令必须在协议处理器上实现来读取自标识数据。

发出这个命令的设备不需要了解协处理器的物理地址，请求者仅需处理偏移量。请求者从偏移量 0 开始，对于每个后续的请求仅需增加偏移量值。除非目标节点具有足够大的缓冲区来处理较大的计数，字节计数应限定为 16。使用的数据结构见 A.31。

13.7.19 取网络变量值(Network Variable Value Fetch)

这个报文用于轮询网络变量。和网络变量轮询报文相比有两个优点：它使用网络变量索引并因此与网络变量选择号分配无关；不管节点是在线还是离线状态，它都可以得到数值。使用的数据结构见 A.32。

13.7.20 手动服务请求报文(Manual Service Request Message)

这个报文不像其他的网络管理报文，它是主动提供的报文。当启动节点的手动服务请求时，该报文从节点发送到网络上。该报文以域范围内广播的方式发送，域长为 0、源子网为 0、并且节点地址为 0。使用的数据结构见 A.33。

13.7.21 网络管理转义代码(Network Management Escape Code)

网络管理命令代码之一保留为转义代码。转义代码的值为 0x7D。发送转义代码作为网络管理命令会使 APDU 的前两个字节被解释为附加的命令代码。这个能力允许网络管理协议可以以产品特定的方式扩展。

如果一个节点响应使用网络管理转义代码的网络管理报文，那么节点将总是响应产品查询(Product Query)命令。所有其他的命令都是产品特定的、有关文档随产品提供。响应这个命令有两种形式。短的形式仅包含一个单独字节来指定产品，完整的形式包含：

——响应代码；

——产品字节(和短形式一致)；

——用作型号的两字节字段；

——用作固件版本的一个单独字节；

广东省网络空间安全协会受控资料

——用作设备配置或模式的一个字节，和

——用作收发器类型的一个字节。

单字节产品规格代码是保留的并且必须被分配，以保证实现之间没有冲突。在完整的响应形式中，除非设备可被置为几种模式或设备配置，返回的设备配置选项/模式字节值为零。在这种情况下，字节包含设备的当前模式或配置。

成功或失败是以转义码而不是子代码或命令的形式报告。使用的数据结构见 A.34。

13.7.22 路由器模式(Router Mode)

这个请求指示路由器执行几个路由相关任务中的一个。“resume”命令使路由器从“all flood”状态返回。“init router tables”命令将全部的路由表从非易失性存储器拷贝到 RAM 表中(如果是可配置路由器)，或者将全部的 RAM 表置为 flood(如果是自学习路由器)，这和在节点复位之后出现的动作相同。“mode all flood”命令使路由器转发该域内的所有数据包。路由器模式命令影响路由器两侧的行为，这个命令是通过请求/响应协议传递的。需要注意的是可以用正常的网络管理节点模式请求使整个路由器离线和在线。使用的数据结构见 A.35。

13.7.23 路由器清除组或子网表(Router Clear Group or Subnet Table)

这个请求用于清除路由器一侧中一个单独域的组或子网路由表中的所有表项。为了防止过长的非易失性存储器的写操作，这个命令以 8 个字节为一组分段操作。这个命令是通过请求/响应协议传递的。本命令将更新非易失性存储器中的网络配置存储器的校验和。使用的数据结构见 A.36。

13.7.24 路由器组或子网表下载(Router Group or Subnet Table Download)

这个请求用于配置路由器一侧中一个指定域的非易失性存储器中的全部组或子网表。下载功能被分成 8 字节的段。这个命令是通过请求/响应协议传递的。本命令将更新非易失性存储器中的网络配置存储器的校验和。使用的数据结构见 A.37。

广东省网络空间安全协会受控资料

13.7.25 路由器组转发(Router Group Forward)

这个请求设置路由表中指定域中的一个给定组的转发标志。这个命令是通过请求/响应协议传递的。如果校验和发生变化，则本命令将更新非易失性存储器中的网络配置存储器的校验和。使用的数据结构见 A.38。

13.7.26 路由器子网转发(Router Subnet Forward)

这个请求设置路由表中指定域中的一个给定子网的转发标志。这个命令是通过请求/响应协议传递的。如果校验和发生变化，则本命令将更新非易失性存储器中的网络配置存储器的校验和。使用的数据结构见 A.39。

13.7.27 路由器不转发组(Router Do Not Forward Group)

这个请求清除路由表中指定域中的一个给定组的转发标志。这个命令是通过请求/响应协议传递的。如果校验和发生变化，则本命令将更新非易失性存储器中的网络配置存储器的校验和。使用的数据结构见 A.40。

13.7.28 路由器不转发子网(Router Do Not Forward Subnet)

这个请求清除路由表中指定域中的一个给定子网的转发标志。这个命令是通过请求/响应协议传递的。如果校验和发生变化，则本命令将更新非易失性存储器中的网络配置存储器的校验和。使用的数据结构见 A.41。

13.7.29 路由器组或子网表报告(Router Group or Subnet Table Report)

这个请求用于报告路由器一侧的指定域的在非易失性存储器或 RAM 中的组或子网表的当前设置。报告功能被分成 8 字节的段。这个命令是通过请求/响应协议传递的。使用的数据结构见 A.42。

13.7.30 路由器状态(Router Status)

这个请求用于报告路由器网络配置存储器和扩散/正常模式。这个命令是通过请求/响应协议传递的。使用的数据结构见 A.43。

13.7.31 路由器一侧转义代码(Router Half Escape Code)

尽管其本身不是网络管理命令,但为了完整性,本条中包括了这个代码。当这个代码放在 APDU 的开始并且后面紧跟着任何网络管理或网络诊断命令时,该命令将被传送到路由器另一侧进行处理。任何响应都是以正常的方式返回的。

```
byte command; /* Destination: NM,code: 30 */
```

13.8 DPDU 的类型和格式

大多数诊断 PDU(DPDU)都是在会话层的请求和/或响应中传递的。这样 DPDU 就缺省地继承了封装 SPDU 的请求或响应属性。

本条列出了 GB/Z 20177 的 DPDU。在附录 A 中定义的比特位和字节次序规则是,其中每个字节的最高有效位最先传送;一个记录的第一个字节是该记录的最低有效字节。在描述值域时,值对应着该报文的命令号或响应代码。另外,字符串"+pass/fail"表示在该响应的高位中的一个单独的标志位被置位,用于指示命令是成功还是失败。

13.8.1 查询状态(Query Status)

这个命令给出一个节点状态的快照。它传递差错统计、复位信息、节点状态、差错日志、系统映像版本和协议处理器型号。差错统计、复位原因和差错日志可以通过"clear status"命令全部清除。需注意的是节点复位时,也将清除统计。这个命令不要求执行鉴别。使用的数据结构在 A.44 中。

各字段定义如下:

——传输差错(transmission_errors):这是在网络上出现的传送差错次数的计数。在数据包接收期间检查到 CRC 差错或当一个数据包长小于 8 字节时,则表示发生传送差错。这可能是由于冲突、媒体噪声、信号衰减等造成的。

——事务超时(transaction_timeouts):这是在试图进行确认或请求/响应事务时出现超时次数的计数。当一个节点在配置时间间隔内重试了配置的次数后,还不能接收到所有期望的确认或响应,则出现一次超时。

——接收事务满(receive_transaction_full):这个计数器反映的是由于没有接收事务记录可用,造成入向的无确认重复、确认或请求报文丢失的次数。

——丢弃报文(lost_messages):这是由于没有可用的应用缓冲区而丢弃的送到该节点的报文个数。

——丢失报文(missed_messages):这是由于报文没有可用的网络缓冲区(数据包缓冲区)或对于接收报文网络缓冲区太小,在网络上传输而不能接收的报文的个数。

——复位原因(reset_cause):本字节包含复位原因信息。它标识了最近复位的来源。本字节的值如下:(X =>无关):

上电复位	0bXXXXXXX1
外部复位	0bXXXXXX10
WDT 复位	0bXXXX1100
软件发出的复位	0bXXX10100

——节点状态(node_state):包含节点状态和节点条件(在网络管理一节的"节点状态"部分定义的)。

结构 node_state 如下所示:

```
struct node_state
{
    unsigned                    :4;   // application specific
    unsigned      offline       :1;   // 1=>offline,0=>online
    unsigned      state         :3;   // 2=>Unconfigured
```

广东省网络空间安全协会受控资料

```
                                    // 3=>Applicationless
                                    // 4=>Configured
                                    // 6=>Hard offline
};
```

通常一个节点只有处于已配置状态，才能置位离线字段。然而，如果一个节点在离线状态被告知从已配置状态转移到另一个状态，node_state 将显示离线字段被置位，并且不是已配置状态。

当被看作一个字节字段时，node_state 值的例子如下所示：

```
Unconfigured            0x02
Unconfig/App-less       0x03
Configured/online       0x04
Configured/hard-offline 0x06     /* Permanent offline          */
Configured/offline      0x0C     /* Non-reset retained offline */
                                 /* (note this is actually an  */
                                 /* encoding of the node state */
                                 /* of "configured" and the    */
                                 /* offline condition)
Configured              0x8C     /* Non-reset retained bypass- */
                                 /* mode offline Like config-  */
                                 /* ured/offline except that   */
                                 /* the application went off-  */
                                 /* line in bypass mode        */
```

广东省网络空间安全协会受控资料

——版本(version)：版本号反映了在协议处理器中协议实现的版本，它被网络管理工具用于计算对于不被标准网络管理地址分配/报告命令支持的非易失性存储器数据字段的地址。对于不包含附加的、专为应用定制的实现的版本号是 1～127；对于包含附加的、专为应用定制特定的实现版本号是 128～255。255 是特别的转义版本号，表示通过其他命令可得到更多的版本号信息。

——差错日志(error_log)：差错日志包含最近系统记录的差错。0 表示没有差错。1～127 范围内的任何一个差错都表示应用差错，并且相对该应用差错是唯一的。128～255 范围内的差错表示系统差错。给定的协议实例检测和报告的差错是面向特定实现的，目前分配给系统差错的差错号如下：

0x81 Bad Event　　应用程序处理器的调度程序有一个事件表，包含对程序中的各种 when 语句的编码的表示。如果调度程序遇到一个非法的事件编码，则记录这个差错。

0x82 NV Length　　如果入向的网络变量更新与和它相匹配的应用程序中的输入网络变量长度不一致，则记录这个差错并丢弃该网络变量更新。

0x83 NV Short　　如果入向的网络变量更新丢失了选择号的第二个字节，则记录这个差错。

0x84 EE Write Fail	每次写入 EEPROM 后，结果将回读。如果回读结果不匹配，则记录这个差错。
0x85 Bad Address Type	如果地址表中的类型字段是非法的，则记录这个差错。
0x88 Sync NV	当网络变量被写入时，更新一个位数组来指示更新需要被传送。在“post_events”中，将检查这个位数组，并且为如此标记的每个网络变量都产生一个报文并且相应位被清除。然而，如果缓冲区不可用，应用程序将进入占先模式。在占先模式下，如果置位的同步网络变量被应用程序写入，那么这个差错将被记录。
0x8A Invalid Domain	试图使用长度字节与 0xF8 进行“AND”操作不为 0 的域表的表项发送一条报文。该报文将被丢弃。
0x8D Invalid Address Index	试图用超出范围的索引访问地址表。在使用任何与网络管理命令相关的地址表时，或使用地址表发送显式报文或网络变量时，可能会出现这种情况。
0x93 Invalid NV Index	试图使用超出范围的索引访问网络变量配置或网络变量别名表。当使用任何与网络管理命令相关的网络变量表时，或形成一个网络变量更新报文时，可能会出现这种情况。
0x9A Illegal Transceiver Register Address	试图使用超出范围的索引访问一个特殊目的模式收发器寄存器。当使用“retrieve_xcvr_status”功能访问收发器的寄存器时，可能会出现这种情况。
0x9B Transceiver Register Operation Failure	等待 MAC 层获得一个查询特殊目的模式收发器的机会时，出现超时。
0xA1 Self-Installation Semaphore 0xA2 Read/Write Semaphore 0xC0-DF State Byte Semaphore	差错日志字节，用于特定情况下，保存意外的状态信息。

——型号(model_number)：型号表示协议处理器的型号。

型号和版本号在一起，用于决定节点使用的准确固件映像。

13.8.2 代理状态(Proxy Status)

这个命令可用于通过一个代理节点将命令传送到一个或多个目标节点。代理命令随同 APDU 中目的地地址一起被发送到一个代理节点。代理节点将命令转发到目标节点，并将响应转发回原始的请求者。除了目的地地址被删除外，代理将原封不动地重复来自请求者的报文。代理命令仅可用于转发状态请求或查询标识(未配置)请求。尽管这个命令本身并不要求执行鉴别，但如果原始请求者标记为鉴别的，那么转发到目标的请求也将作如此标记。

广东省网络空间安全协会受控资料

原始请求者通过 APDU 中地址表项形式的数据指定目的地地址。在处理过程中代理节点用它来决定目的地址和重试/超时值。存在一个例外情况——域位被忽略(报文总是在接收报文的相同域中转发)。需要注意的是,在目的地地址给出的重试/超时值将产生一个事务处理周期,它应该比原始请求者使用的处理周期要短。一般来说,如果代理和目标在同一个信道上,这个命令工作得最好。使用的数据结构见 A.45。

原始请求者接收到的响应数据和直接对该目标发请求所接收到的结果应该是一样的。需要注意的是,在查询标识(未配置)广播的情况下,直接的报文将导致多个响应,而用广播目标发送到一个单独代理的代理命令将对原始请求者仅产生一个响应。

如果代理在一个柔性域接收请求,则这个命令是不允许的。

最后,如果代理节点正处在出向事务处理过程中,它可能不能立即传送转发的请求。取决于延迟多长时间,甚至在几次重试后,原始请求者可能还未能及时接收到响应。因此,代理命令的事务失败可能比其他事务失败要多;在推导结论时,应考虑这种情况。另外,为了使代理命令能够工作,代理节点必须至少有两个应用输入缓冲器。

13.8.3 清除状态(Clear Status)

这个命令清除状态响应信息中的一个子集。这个命令将清除统计信息、扩展的统计信息、复位原因寄存器和差错日志。周期性执行状态请求的节点可以在每个成功的状态响应之后,选择使用清除命令。使用的数据结构见 A.46。

13.8.4 查询收发器状态(Query Transceiver Status)

这个命令检索收发器的状态寄存器信息。如果节点上没有收发器,或者与收发器的通信失败,则此命令失败。不管收发器实际支持多少个寄存器,都将返回相当于 7 个寄存器的数据。由控制器决定有多少个寄存器有效。使用的数据结构见 A.47。

广东省网络空间安全协会受控资料

附 录 A
（规范性附录）
参 考 实 现

本附录包含协议中定义的函数的伪代码算法和相关数据结构的描述。这些算法是从可实行的参考实现中提取出来的。为了使可实行的实现能够验证这些算法，有必要在协议处理器和物理层之间定义边界和接口，就像在协议应用层和实际应用程序之间的边界一样。这些接口只是提供信息，并且在参考实现中的一些特定函数可能有几种实现方法。这些也作为例外情况提了出来。最后，参考实现是在摩托罗拉 MC68360 处理器上实行的，因此在代码中就存在一些与该处理器相关的实现细节。

A.1 预测的 CSMA 算法

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        File:          spm. c

     Version:          1. 7

   Reference:              None

     Purpose:          MAC sublayer,comm_type = 2 using SPI port on 360.
                       This does not support continuous frame exchange.
                       This files also has functions to support
                       io buttons for testing purpose such as
                       manual service request,reset,io pin,and LEDs.

        Note:        None.


       To Do:        None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* START INFORMATIVE — Direct Mode */
/* This implementation is for transceivers that work with the MAC
* sublayer configured for comm_type = 2 only.
* If the implementation needs to support transceivers with the MAC
* sublayer configured for comm_type = 1,
* this example serves only as a starting point for such an
* implemenation. */
/* END INFORMATIVE — Direct Mode */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

广东省网络空间安全协会受控资料

```
Section: Includes
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include <string.h>
#include <stdlib.h>

#include <CEA709_1.h>
#include <node.h>
#include <link.h>
#include <physical.h>

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Section: Constant Definitions
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Used to indicate how often to check status of i/o buttons */
#define PHYIO_CHECK_INTERVAL 0.1 /* In Seconds */

/* Macro sets bit B of 16 bit word X */
#define SET_BIT(B,X) (((0x0001U << (B)) | (X))

/* Macro Clears bit B of 16 bit word X */
#define CLEAR_BIT(B,X) (( ~(0x0001U << (B)) ) & (X))

/* Dual Port Ram Base on Arnewsh Board */
#define DPRB              0x01000000UL

/* SPI Parameter Ram Base */
#define SPIB              (DPRB + 0xD80UL)

/* SCC2 Parameter Ram Base */
#define SCC2B             (DPRB + 0xD00UL)

/* Register Base = DPRB + 4 k */
#define REGB              (DPRB + 0x1000UL)

/* 16bit CPM Command Register */
#define CR                (REGB + 0x5C0UL)

/* 16 bit Serial DMA Config Register */
#define SDCR              (REGB + 0x51EUL)

/* 24(32)bit CPM Interrupt Config Reg. */
#define CICR              (REGB + 0x540UL)
```

广东省网络空间安全协会受控资料

```
/* 32 bit CPM Interrupt Pending Reg. */
#define CIPR            (REGB + 0x544UL)

/* 32 bit CPM Interrupt Mask Reg. */
#define CIMR            (REGB + 0x548UL)

/* Clear CISR bit by writing a 1 */
/* 32 bit CPM Interrupt Service Register */
#define CISR            (REGB + 0x54CUL)

/* 32 bit SI Clock Route */
#define SICR            (REGB + 0x6ECUL)

/* 16 bit SPI Mode Register */
#define SPMODE          (REGB + 0x6A0UL)

/* Clear SPIE bit by writing a 1 */
/* 8 bit SPI Event Register */
#define SPIE            (REGB + 0x6A6UL)

/* 8 bit SPI Mask Register */
#define SPIM            (REGB + 0x6AAUL)

/* 8 bit SPI Command Register */
#define SPCOM           (REGB + 0x6ADUL)

/* 32 bit SCC2 general mode register low */
#define GSMRL2          (REGB + 0x620UL)

/* 32 bit SCC2 general mode register high */
#define GSMRH2          (REGB + 0x624UL)

/* 16 bit SCC2 protocol specific mode reg */
#define PSMR2           (REGB + 0x628UL)

/* 16 bit SCC2 transmit on demand */
#define TODR2           (REGB + 0x62CUL)

/* 16 bit SCC2 Data sync register */
#define DSR2            (REGB + 0x62EUL)

/* Clear SCCE bit by writing a 1 */
/* 16 bit SCC2 event register */
```

广东省网络空间安全协会受控资料

```
#define SCCE2            (REGB + 0x630UL)

/* 16 bit SCC2 mask register */
#define SCCM2            (REGB + 0x634UL)

/* 8 bit SCC2 status register */
#define SCCS2            (REGB + 0x637UL)

/* 32 bit Baud rate gen config register */
#define BRGC2            (REGB + 0x5F4UL)

/* 16 bit port A data direction reg */
#define PADIR            (REGB + 0x550UL)

/* 16 bit port A pin assignment reg */
#define PAPAR            (REGB + 0x552UL)

/* 16 bit port A open drain register */
#define PAODR            (REGB + 0x554UL)

/* 16 bit port A data register */
#define PADAT            (REGB + 0x558UL)

/* 32 bit port B direction register */
#define PBDIR            (REGB + 0x6B8UL)

/* 32 bit port B pin assignment register */
#define PBPAR            (REGB + 0x6BCUL)

/* 16 bit port B open drain register */
#define PBODR            (REGB + 0x6C2UL)

/* 32 bit port B data register */
#define PBDAT            (REGB + 0x6C4UL)

/* 16 bit port C direction register */
#define PCDIR            (REGB + 0x560UL)

/* 16 bit port C pin assignment reg */
#define PCPAR            (REGB + 0x562UL)

/* 16 bit port C special options */
#define PCSO             (REGB + 0x564UL)
```

广东省网络空间安全协会受控资料

```
/* 16 bit port C data register */
#define PCDAT              (REGB + 0x566UL)

/* 16 bit port C interrupt register */
#define PCINT              (REGB + 0x568UL)

/* Timer Related Constants */
/* 16 bit timer general config register */
#define TGCR        (REGB + 0x580UL)

/* 16 bit timer reference   reg 1 */
/* or 32 bit cascaded 1 & 2        */
#define TRR1        (REGB + 0x594UL)

/* 16 bit timer counter reg 1 */
/* or 32 bit cascaded 1 & 2    */
#define TCN1        (REGB + 0x59CUL)


/* 16 bit timer mode register 4 */
#define TMR2        (REGB + 0x592UL)

/* 16 bit timer reference register 4 */
#define TRR2        (REGB + 0x596UL)

/* 16 bit timer capture register 4 */
#define TCR2        (REGB + 0x59AUL)

/* 16 bit timer counter register 4 */
#define TCN2        (REGB + 0x59EUL)

/* 16 bit timer event register 4 */
#define TER2        (REGB + 0x5B2UL)


/* ISR defines */
/* Interrupt vector base on Arnewsh Board */
#define VBASE              0x00000000UL

/* User Interrupt vector num 3 MSbits */
#define UIVN_MSB           0x4U
```

广东省网络空间安全协会受控资料

```
/* SPI Int. vector num 5 LSBits from CPIC table */
#define SPIVN_LSB          0x5U

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
8 bit vector number = 3 msb set by user,5 lsb from CPIC table shift
UIVN_MSB left 5 to bits 7 6 5 then OR with SPIVN_LSB to get vector
number vector address is 4 times vector number + vector base
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* 32 bit SPI Int. vector address */
#define SPIV        ((((UIVN_MSB << 5) | SPIVN_LSB) * 4UL) + VBASE)


/* need an ISR for int error since int. error cannot be masked */
/* CPM int. error int. vector. num 5 LSB  */
#define CERRVN_LSB         0x0UL

/* cpm error iv */
#define CERRV   ((((UIVN_MSB << 5) | CERRVN_LSB) * 4UL) + VBASE)

/* spurious interrupt exception  vector */
#define SPURINTV 0x60U + VBASE

/* bus error  exception  vector */
#define BUSERRV 0x8U + VBASE

/* SPI Channel Number for CPM Command Register Commands */
#define SPI_CH_NUM         0x5U

/* SCC2 channel number for CPM Command Register Command */
#define SCC2_CH_NUM        0x4U

/* Opcode for CPM Com. Reg. to Init Tx Rx Param Ram */
#define INIT_TRP_OPCODE 0x0U

/* number of receive (transmit) BDs for SPI */
#define NUM_BD 0x1U

/* RBD_BASE and TBD_BASE must be divisible by 8 */
/* receive buffer descriptor base */
#define RBD_BASE (DPRB + 0x500UL)

/* put Transmit buffers right after Receive,  8bytes per BD */
/* transmit buffer descriptor base */
```

广东省网络空间安全协会受控资料

```
#define TBD_BASE (RBD_BASE + (NUM_BD * 0x8U))

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   FCR_INIT
         bits 7—5 unused,
         bit 4,Mot big endian = 1
         bits 3—0,function code dma = 8
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* SPI Param Function Code Register Init */
#define FCR_INIT          0x18U

/* max length of receive buffers 2 bytes per buffer */
#define MAX_BUF_LEN 0x2U

/* CP Interrupt Request Level */
#define CPIR_LEVEL 0x4U

#define NUM_BD_SCC2       0x1U

/* scc2 receive buffer descriptor. base */
#define RBD_BASE_SCC2 (DPRB + 0x440UL)

/* scc2 transmit bd base */
#define TBD_BASE_SCC2 (RBD_BASE_SCC2 + (NUM_BD_SCC2 * 8U))

/* max buffer length scc2 */
#define MAX_BL_SCC2       16U

/* length of history arrays */
#define NUM_HIST 512

/* time limit before xcvr resets */
#define RESET_COUNT_LIMIT 0xFFFF

/* number of ticks to delay during restart processing */
#define RESTART_DELAY_TICKS 165000UL

/*   Constants related to Channel Access Algorithm */
/* maximum size of backlog for access algorithm */
#define MAX_BACKLOG 63U

/* # of additional slots per backlog increment */
#define W_BASE 16U
```

广东省网络空间安全协会受控资料

```
/* used in conversion from this specification's
   reference time to clock ticks on the 360 */
#define NICS_TICKS_BASE 480UL

/* used in computation of SPI—SPM bit clock */
#define BIT_CLOCK_BASE 312500UL

/* used in computation of bit clock */
#define RATIO_BASE 3UL

/*****************************************************************
Implementation specific timing adjustments to channel access algorithm
This is where one would add or subtract from the respective times to
account for differences between the implementation specific delays and
latencies and those in this specification.
*****************************************************************/
/* adjustment to the beta1 slot time */
#define BETA1_ADJUST_TICKS 0

/* adjustment to the beta2 slot time */
#define BETA2_ADJUST_TICKS 0

/* adjustment to the cycle timer time */
#define CYCLE_ADJUST_TICKS 0

/* adjustment to the begin of packet tx */
#define WAIT_TX_ADJUST_TICKS 0

/*****************************************************************
Section: Type Definitions
*****************************************************************/
/* 68360 communications processor register definitions */

typedef struct            /* CP Command Register 16 bit */
{
   unsigned reset   : 1; /* reset CPM */
   unsigned         : 3; /* unused */
   unsigned opcode : 4; /* opcode for command */
   unsigned chnum   : 4; /* channel number selects which port
                            SPI,SCC etc */
   unsigned         : 3; /* unused */
   unsigned flag    : 1; /* flag,command executes when set,
```

广东省网络空间安全协会受控资料

```
                                cp clears when done */
} CPCommandReg;

typedef struct              /* Serial DMA Config Register 16 bit */
{
   unsigned        : 1;    /* unused */
   unsigned frz    : 2;    /* freeze */
   unsigned        : 2;    /* unused */
   unsigned sism : 3;      /* SDMA interrupt service mask */
   unsigned        : 1;    /* unused */
   unsigned said : 3;      /* SDMA arbitration ID */
   unsigned        : 2;    /* unused */
   unsigned inte : 1;      /* interrupt error */
   unsigned intb : 1;      /* interrupt break point */
} SDMAConfigReg;

typedef struct   /* CPM interrupt configuration register 32 bit */
{
   unsigned                : 8; /* unused */
   unsigned          scdp : 2; /* scc port for priority slot d */
   unsigned          sccp : 2; /* scc port for priority slot c */
   unsigned          scbp : 2; /* scc port for priority slot b */
   unsigned          scap : 2; /* scc port for priority slot a */
   unsigned          irl  : 3; /* interrupt request level */
   unsigned          hpi  : 5; /* highest priority interrupt */
   unsigned          vba  : 3; /* vector base address number offset */
   unsigned                : 4; /* reserved */
   unsigned          sps  : 1; /* spread priority scheme */
} CIConfigReg;

/* CPM interrupt source 32 bit,same type for CIPR CIMR CISR */
typedef struct
{ /* names correspond to sources */
   unsigned pc0     : 1;              /* bit 31 */
   unsigned scc1    : 1;
   unsigned scc2    : 1;
   unsigned scc3    : 1;
   unsigned scc4    : 1;
   unsigned pc1     : 1;
   unsigned timer1 : 1;
   unsigned pc2     : 1;              /* bit 24  */
   unsigned pc3     : 1;              /* bit 23 */
   unsigned sdma    : 1;
```

广东省网络空间安全协会受控资料

```
    unsigned idma1   : 1;
    unsigned idma2   : 1;
    unsigned         : 1;            /* unused */
    unsigned timer2 : 1;
    unsigned rtt     : 1;
    unsigned         : 1;            /* bit 16 unused */
    unsigned pc4     : 1;            /* bit 15 */
    unsigned pc5     : 1;
    unsigned         : 1;            /* unused */
    unsigned timer3 : 1;
    unsigned pc6     : 1;
    unsigned pc7     : 1;
    unsigned pc8     : 1;
    unsigned         : 1;            /* bit 8 unused */
    unsigned timer4 : 1;             /* bit 7 */
    unsigned pc9     : 1;
    unsigned spi     : 1;            /* bit 5 */
    unsigned smc1    : 1;
    unsigned smc2    : 1;            /* also pip */
    unsigned pc10    : 1;
    unsigned pc11    : 1;
    unsigned         : 1;            /* bit 0 unused */
} CISourceReg;

/* serial interface clock route register 32 bit */
typedef struct
{
    unsigned gr4    : 1;    /* grant support for scc4 */
    unsigned sc4    : 1;    /* connection for scc4 */
    unsigned r4cs   : 3;    /* receive clock source for scc4 */
    unsigned t4cs   : 3;    /* transmit clock source for scc4 */
    unsigned gr3    : 1;    /* grant support for scc3 */
    unsigned sc3    : 1;    /* connection for scc3 */
    unsigned r3cs   : 3;    /* receive clock source for scc3 */
    unsigned t3cs   : 3;    /* transmit clock source for scc3 */
    unsigned gr2    : 1;    /* grant support for scc2 */
    unsigned sc2    : 1;    /* connection for scc2 */
    unsigned r2cs   : 3;    /* receive clock source for scc2 */
    unsigned t2cs   : 3;    /* transmit clock source for scc2 */
    unsigned gr1    : 1;    /* grant support for scc1 */
    unsigned sc1    : 1;    /* connection for scc1 */
    unsigned r1cs   : 3;    /* receive clock source for scc1 */
    unsigned t1cs   : 3;    /* transmit clock source for scc1 */
```

广东省网络空间安全协会受控资料

```
} SIClockRouteReg;

typedef struct              /* SPI command register 8 bit */
{
   unsigned str : 1;      /* start transmit and receive */
   unsigned res : 7;      /* reserved write with zeros */
} SPICommandReg;

typedef struct              /* SPI mode register 16 bit */
{
   unsigned        : 1;        /* unused */
   unsigned loop   : 1;        /* local loop back */
   unsigned ci     : 1;        /* clock invert */
   unsigned cp     : 1;        /* clock phase */
   unsigned div16  : 1;        /* divide brgclk by 16 for spiclk */
   unsigned rev    : 1;        /* reverse data from big endian to
                                  little endian */
   unsigned ms     : 1;        /* master slave */
   unsigned en     : 1;        /* enable SPI */
   unsigned len    : 4;        /* number of bits per char = len + 1 */
   unsigned pms    : 4;        /* prescale modulus select,divide
                                  down clock */
} SPIModeReg;

typedef struct   /* SPI event and mask registers 8 bit */
{
   unsigned      : 2;    /* unused */
   unsigned mme  : 1;    /* multi-master error */
   unsigned txe  : 1;    /* transmit error */
   unsigned      : 1;    /* unused */
   unsigned bsy  : 1;    /* busy error */
   unsigned txb  : 1;    /* transmit buffer complete */
   unsigned rxb  : 1;    /* receive buffer complete */
} SPIEventMaskReg;

typedef struct              /* SPI Parameter RAM  40 bytes */
{
   uint16 rbase;   /* base address of receive BD's,set by user */
   uint16 tbase;   /* base address of transmit BD's,set by user */
   uint8  rfcr;    /* receive function code,set by user  */
   uint8  tfcr;    /* transmit function code,set by user  */
   uint16 mrblr;   /* maximum receive buffer length register,
                      set by user  */
```

广东省网络空间安全协会受控资料

```
    uint32 rstate; /* rx internal state      */
    uint32 ridp;   /* rx internal data ptr */
    uint16 rbptr;  /* rx BD ptr */
    uint16 ribc;   /* rx internal byte count */
    uint32 rtemp;  /* rx temp */
    uint32 tstate; /* tx internal state      */
    uint32 tidp;   /* tx internal data ptr */
    uint16 tbptr;  /* tx BD ptr */
    uint16 tibc;   /* tx internal byte count */
    uint32 ttemp;  /* tx temp */
} SPIParamRam;


typedef struct              /* Receive Buffer Descriptor 8 bytes */
{
    unsigned e   : 1;  /* empty buffer ready to receive */
    unsigned     : 1;  /* unused */
    unsigned w   : 1;  /* wrap final bd so wrap to top */
    unsigned i   : 1;  /* interrupt,set rxb bit in spie register
                          when filled */
    unsigned l   : 1;  /* last,if slave then buffer has last bit */
    unsigned     : 1;  /* unused */
    unsigned cm  : 1;  /* continuous mode buffer can be overwritten
                          e always 1 */
    unsigned     : 7;  /* unused */
    unsigned ov  : 1;  /* receiver overrun only when slave */
    unsigned me  : 1;  /* multiple master error */
    uint16   dataLen; /* data length = number of bytes written
                          into buffer */
    SPMRxFrame * dataPtr;  /* ptr to buffer of data */
} RBufferDesc;

typedef struct              /* Transmit Buffer Descriptor 8 bytes */
{
    unsigned r   : 1; /* ready to transmit buffer */
    unsigned     : 1; /* unused */
    unsigned w   : 1; /* wrap final bd so wrap to top */
    unsigned i   : 1; /* interrupt,set txb bit in spie register
                         when sent */
    unsigned l   : 1; /* last,if slave then buffer has last bit */
    unsigned     : 1; /* unused */
    unsigned cm  : 1; /* continuous mode buffer can be re-sent
                         r always 1 */
```

广东省网络空间安全协会受控资料

```
    unsigned      : 7; /* unused */
    unsigned un : 1; /* transmit underrun only when slave */
    unsigned me : 1; /* multiple master error */
    uint16   dataLen; /* data length = number of bytes to send
                         from this buffer */
    SPMTxFrame * dataPtr;   /* ptr to buffer of data */
} TBufferDesc;

/* scc structs */

typedef struct          /* general scc mode register  high 32 bit */
{
    unsigned      : 15;     /* unused */
    unsigned gde  : 1;      /* glitch detect enable */
    unsigned tcrc : 2;      /* transparent crc */
    unsigned revd : 1;      /* reverse data */
    unsigned trx  : 1;      /* transparent receiver */
    unsigned ttx  : 1;      /* transparent transmitter */
    unsigned cdp  : 1;      /* cd pulse */
    unsigned ctsp : 1;      /* cts pulse */
    unsigned cds  : 1;      /* cd sampling */
    unsigned ctss : 1;      /* cts sampling */
    unsigned tfl  : 1;      /* transmit fifo length */
    unsigned rfw  : 1;      /* receive fifl width */
    unsigned txsy : 1;      /* transmitter synchronized to receiver */
    unsigned synl : 2;      /* sync length */
    unsigned rtsm : 1;      /* rts mode */
    unsigned rsyn : 1;      /* receive sync timing */
} GSMRegHigh;


typedef struct          /* general scc mode register  low 32 bit */
{
    unsigned      : 1;      /* unused */
    unsigned edge : 2;      /* DPLL clock edge */
    unsigned tci  : 1;      /* transmit clock invert */
    unsigned tsnc : 2;      /* transmit sense bits */
    unsigned rinv : 1;      /* dpll receive invert data */
    unsigned tinv : 1;      /* dpll transmit invert data */
    unsigned tpl  : 3;      /* transmit preamble length */
    unsigned tpp  : 2;      /* transmit preamble pattern */
    unsigned tend : 1;      /* transmit frame ending */
    unsigned tdcr : 2;      /* transmit divide clock rate */
```

广东省网络空间安全协会受控资料

```
    unsigned rdcr : 2;        /* receive dpll clock rate */
    unsigned renc : 3;        /* receiver decoding method */
    unsigned tenc : 3;        /* transmitter encoding method */
    unsigned diag : 2;        /* diagnostic mode */
    unsigned enr  : 1;        /* enable receive */
    unsigned ent  : 1;        /* enable transmit */
    unsigned mode : 4;        /* channel protocol mode */
} GSMRegLow;

typedef struct            /* SCC event/mask register 16 bit */
{
    unsigned      : 3;     /* unused */
    unsigned glr : 1;      /* glitch on receive */
    unsigned glt : 1;      /* glitch on tranmit */
    unsigned dcc : 1;      /* dpll carrier sense changed */
    unsigned      : 2;     /* unused */
    unsigned gra : 1;      /* graceful stop complete */
    unsigned      : 2;     /* unused */
    unsigned txe : 1;      /* transmit error */
    unsigned rch : 1;      /* receive character or long word */
    unsigned bsy : 1;      /* busy condition */
    unsigned tx  : 1;      /* buffer transmitted */
    unsigned rx  : 1;      /* buffer received */
} SCCEventMaskReg;

typedef struct            /* scc status register 8 bit */
{
    unsigned      : 6;  /* unused */
    unsigned cs : 1;  /* carrier sense */
    unsigned      : 1;  /* unused */
} SCCStatusReg;

typedef struct    /* SCC Parameter RAM Transparent mode 56 bytes */
{
    uint16 rbase; /* base address of receive BD's,set by user */
    uint16 tbase; /* base address of transmit BD's,set by user */
    uint8  rfcr;  /* receive function code,set by user   */
    uint8  tfcr;  /* transmit function code,set by user   */
    uint16 mrblr; /* maximum receive buffer length register,
                     set by user   */
    uint32 rstate; /* rx internal state      */
    uint32 ridp;   /* rx internal data ptr */
    uint16 rbptr;  /* rx BD ptr */
```

广东省网络空间安全协会受控资料

```
    uint16 ribc;    /* rx internal byte count */
    uint32 rtemp;   /* rx temp */
    uint32 tstate;  /* tx internal state      */
    uint32 tidp;    /* tx internal data ptr */
    uint16 tbptr;   /* tx BD ptr */
    uint16 tibc;    /* tx internal byte count */
    uint32 ttemp;   /* tx temp */
    uint32 rcrc;    /* temp receive crc */
    uint32 tcrc;    /* temp transmit crc */
    uint32 crcp;    /* crc preset for transparent mode */
    uint32 crcc;    /* crc constant for transparent mode */
} SCCParamRam;

typedef struct      /* SCC Receive Buffer Descriptor 8 bytes */
{
    unsigned e  : 1; /* empty buffer ready to receive */
    unsigned    : 1; /* unused */
    unsigned w  : 1; /* wrap final bd so wrap to top */
    unsigned i  : 1; /* interrupt,set rxb bit in scce register
                        when filled */
    unsigned l  : 1; /* this buffer last in frame */
    unsigned f  : 1; /* first in frame */
    unsigned cm : 1; /* continuous mode buffer can be overwritten
                        e always 1 */
    unsigned    : 1; /* unused */
    unsigned de : 1; /* dpll error */
    unsigned    : 2; /* unused */
    unsigned no : 1; /* non octet error */
    unsigned    : 1; /* unused */
    unsigned cr : 1; /* crc error */
    unsigned ov : 1; /* receiver overrun    */
    unsigned cd : 1; /* carrier detect lost */
    uint16   dataLen; /* data length = number of bytes written
                         into buffer */
    Byte    * dataPtr; /* ptr to buffer of data */
} SCCReceiveBD;

typedef struct    /* SCC Transmit Buffer Descriptor 8 bytes */
{
    unsigned r  : 1; /* ready to transmit buffer */
    unsigned    : 1; /* unused */
    unsigned w  : 1; /* wrap final bd so wrap to top */
    unsigned i  : 1; /* interrupt,set txb bit in scce register
```

广东省网络空间安全协会受控资料

```
                                when sent */
    unsigned l   : 1; /* last byte in frame in this buffer */
    unsigned tc : 1; /* transmit crc */
    unsigned cm : 1; /* continuous mode buffer can be re-sent r always 1 */
    unsigned     : 7; /* unused */
    unsigned un : 1; /* transmit underrun */
    unsigned ct : 1; /* cts lost during frame transmission */
    uint16   dataLen; /* data length = number of bytes to send
                                from this buffer */
    Byte   * dataPtr; /* ptr to buffer of data */
} SCCTransmitBD;

typedef struct   /* BRGC baud rate generator config register 32 bit */
{
    unsigned        : 14;     /* unused */
    unsigned rst    : 1;      /* reset brg */
    unsigned en     : 1;      /* enable brg count */
    unsigned extc   : 2;      /* enable external clock source */
    unsigned atb    : 1;      /* autobaud */
    unsigned cd     : 12;     /* clock divider */
    unsigned div16 : 1;       /* div 16 clock */
} BRGConfigReg;

typedef struct   /* port a direction register 16 bit */
{
    unsigned dr15 : 1;        /* pin direction 0 = input 1 = output */
    unsigned dr14 : 1;
    unsigned dr13 : 1;
    unsigned dr12 : 1;
    unsigned dr11 : 1;
    unsigned dr10 : 1;
    unsigned dr9  : 1;
    unsigned dr8  : 1;
    unsigned dr7  : 1;
    unsigned dr6  : 1;
    unsigned dr5  : 1;
    unsigned dr4  : 1;
    unsigned dr3  : 1;
    unsigned dr2  : 1;
    unsigned dr1  : 1;
    unsigned dr0  : 1;
} PADirectionReg;
```

广东省网络空间安全协会受控资料

```
typedef struct              /* port a pin assignment register 16 bit */
{
   unsigned dd15 : 1;          /* 0 = general purpose io */
   unsigned dd14 : 1;
   unsigned dd13 : 1;
   unsigned dd12 : 1;
   unsigned dd11 : 1;
   unsigned dd10 : 1;
   unsigned dd9  : 1;
   unsigned dd8  : 1;
   unsigned dd7  : 1;
   unsigned dd6  : 1;
   unsigned dd5  : 1;
   unsigned dd4  : 1;
   unsigned dd3  : 1;
   unsigned dd2  : 1;
   unsigned dd1  : 1;
   unsigned dd0  : 1;
} PAPinAssignmentReg;

typedef struct              /* port a open drain register 16 bit */
{
   unsigned      : 8;    /* unused */
   unsigned od7  : 1;    /* 0 = active drive 1 = open drain */
   unsigned od6  : 1;
   unsigned od5  : 1;
   unsigned od4  : 1;
   unsigned od3  : 1;
   unsigned      : 1;     /* unused */
   unsigned od1  : 1;
   unsigned      : 1;     /* unused */
} PAOpenDrainReg;

typedef struct              /* port a data register 16 bit */
{
   unsigned d15 : 1;           /* value of pin */
   unsigned d14 : 1;
   unsigned d13 : 1;
   unsigned d12 : 1;
   unsigned d11 : 1;
   unsigned d10 : 1;
   unsigned d9  : 1;
   unsigned d8  : 1;
```

广东省网络空间安全协会受控资料

```
    unsigned d7   : 1;
    unsigned d6   : 1;
    unsigned d5   : 1;
    unsigned d4   : 1;
    unsigned d3   : 1;
    unsigned d2   : 1;
    unsigned d1   : 1;
    unsigned d0   : 1;
} PADataReg;

typedef struct              /* PBDIR port b direction register 32 bit */
{
    unsigned        : 14;           /* unused */
    unsigned dr17 : 1;              /* pin with given # */
    unsigned dr16 : 1;              /* pin with same # */
    unsigned dr15 : 1;              /* pin with same # */
    unsigned dr14 : 1;              /* pin with same # */
    unsigned dr13 : 1;              /* pin with same # */
    unsigned dr12 : 1;              /* pin with same # */
    unsigned dr11 : 1;              /* pin with same # */
    unsigned dr10 : 1;              /* pin with same # */
    unsigned dr9  : 1;              /* pin with same # */
    unsigned dr8  : 1;              /* pin with same # */
    unsigned dr7  : 1;              /* pin with same # */
    unsigned dr6  : 1;              /* pin with same # */
    unsigned dr5  : 1;              /* pin with same # */
    unsigned dr4  : 1;              /* pin with same # */
    unsigned dr3  : 1;              /* pin with same # */
    unsigned dr2  : 1;              /* pin with same # */
    unsigned dr1  : 1;              /* pin with same # */
    unsigned dr0  : 1;              /* pin with same # */
} PBDirectionReg;

typedef struct   /* PBPAR port b pin assignment register 32 bit */
{
    unsigned        : 14;           /* unused */
    unsigned dd17 : 1;              /* pin with given # */
    unsigned dd16 : 1;              /* pin with same # */
    unsigned dd15 : 1;              /* pin with same # */
    unsigned dd14 : 1;              /* pin with same # */
    unsigned dd13 : 1;              /* pin with same # */
    unsigned dd12 : 1;              /* pin with same # */
    unsigned dd11 : 1;              /* pin with same # */
```

广东省网络空间安全协会受控资料

```
    unsigned dd10 : 1;              /* pin with same # */
    unsigned dd9  : 1;              /* pin with same # */
    unsigned dd8  : 1;              /* pin with same # */
    unsigned dd7  : 1;              /* pin with same # */
    unsigned dd6  : 1;              /* pin with same # */
    unsigned dd5  : 1;              /* pin with same # */
    unsigned dd4  : 1;              /* pin with same # */
    unsigned dd3  : 1;              /* pin with same # */
    unsigned dd2  : 1;              /* pin with same # */
    unsigned dd1  : 1;              /* pin with same # */
    unsigned dd0  : 1;              /* pin with same # */
} PBPinAssignmentReg;

typedef struct          /* PBODR port b open drain register 16 bit */
{
    unsigned od15 : 1;              /* pin with same # */
    unsigned od14 : 1;              /* pin with same # */
    unsigned od13 : 1;              /* pin with same # */
    unsigned od12 : 1;              /* pin with same # */
    unsigned od11 : 1;              /* pin with same # */
    unsigned od10 : 1;              /* pin with same # */
    unsigned od9  : 1;              /* pin with same # */
    unsigned od8  : 1;              /* pin with same # */
    unsigned od7  : 1;              /* pin with same # */
    unsigned od6  : 1;              /* pin with same # */
    unsigned od5  : 1;              /* pin with same # */
    unsigned od4  : 1;              /* pin with same # */
    unsigned od3  : 1;              /* pin with same # */
    unsigned od2  : 1;              /* pin with same # */
    unsigned od1  : 1;              /* pin with same # */
    unsigned od0  : 1;              /* pin with same # */
} PBOpenDrainReg;

typedef struct        /* PBDAT port b data register 32 bit */
{
    unsigned      : 14;             /* unused */
    unsigned d17 : 1;               /* pin with given # */
    unsigned d16 : 1;               /* pin with same # */
    unsigned d15 : 1;               /* pin with same # */
    unsigned d14 : 1;               /* pin with same # */
    unsigned d13 : 1;               /* pin with same # */
    unsigned d12 : 1;               /* pin with same # */
    unsigned d11 : 1;               /* pin with same # */
```

广东省网络空间安全协会受控资料

```
    unsigned d10 : 1;                /* pin with same # */
    unsigned d9  : 1;                /* pin with same # */
    unsigned d8  : 1;                /* pin with same # */
    unsigned d7  : 1;                /* pin with same # */
    unsigned d6  : 1;                /* pin with same # */
    unsigned d5  : 1;                /* pin with same # */
    unsigned d4  : 1;                /* pin with same # */
    unsigned d3  : 1;                /* pin with same # */
    unsigned d2  : 1;                /* pin with same # */
    unsigned d1  : 1;                /* pin with same # */
    unsigned d0  : 1;                /* pin with same # */
} PBDataReg;

typedef struct              /* port c direction register 16 bit */
{
    unsigned       : 4;   /* unused */
    unsigned dr11 : 1;   /* pin direction 0 = input 1 = output */
    unsigned dr10 : 1;
    unsigned dr9  : 1;
    unsigned dr8  : 1;
    unsigned dr7  : 1;
    unsigned dr6  : 1;
    unsigned dr5  : 1;
    unsigned dr4  : 1;
    unsigned dr3  : 1;
    unsigned dr2  : 1;
    unsigned dr1  : 1;
    unsigned dr0  : 1;
} PCDirectionReg;

typedef struct              /* port c pin assignment register 16 bit */
{
    unsigned       : 4;   /* unused */
    unsigned dd11 : 1;   /* 0 = general purpose io */
    unsigned dd10 : 1;
    unsigned dd9  : 1;
    unsigned dd8  : 1;
    unsigned dd7  : 1;
    unsigned dd6  : 1;
    unsigned dd5  : 1;
    unsigned dd4  : 1;
    unsigned dd3  : 1;
    unsigned dd2  : 1;
```

广东省网络空间安全协会受控资料

```
    unsigned dd1   : 1;
    unsigned dd0   : 1;
} PCPinAssignmentReg;

typedef struct              /* port c data register 16 bit */
{
    unsigned      : 4;   /* unused */
    unsigned d11 : 1;   /* value of pin */
    unsigned d10 : 1;
    unsigned d9   : 1;
    unsigned d8   : 1;
    unsigned d7   : 1;
    unsigned d6   : 1;
    unsigned d5   : 1;
    unsigned d4   : 1;
    unsigned d3   : 1;
    unsigned d2   : 1;
    unsigned d1   : 1;
    unsigned d0   : 1;
} PCDataReg;

typedef struct              /* port c interrupt control register 16 bit */
{
    unsigned        : 4;   /* unused */
    unsigned edm11 : 1;   /* edge detect mode for line */
    unsigned edm10 : 1;
    unsigned edm9   : 1;
    unsigned edm8   : 1;
    unsigned edm7   : 1;
    unsigned edm6   : 1;
    unsigned edm5   : 1;
    unsigned edm4   : 1;
    unsigned edm3   : 1;
    unsigned edm2   : 1;
    unsigned edm1   : 1;
    unsigned edm0   : 1;
} PCInterruptReg;

typedef struct              /* port c special options register 16 bit */
{
    unsigned        : 4;                /* unused */
    unsigned cd4   : 1;                /* Carrier detect */
    unsigned cts4 : 1;                /* clear to send */
```

广东省网络空间安全协会受控资料

```
    unsigned cd3   : 1;              /* Carrier detect */
    unsigned cts3 : 1;               /* clear to send */
    unsigned cd2   : 1;              /* Carrier detect */
    unsigned cts2 : 1;               /* clear to send */
    unsigned cd1   : 1;              /* Carrier detect */
    unsigned cts1 : 1;               /* clear to send */
    unsigned       : 4;              /* unused */
} PCSpecialOptionsReg;

/* Timer related TypeDefs */

typedef struct     /* timer general config register  16 bit */
{
    unsigned cas4 : 1;   /* cascade timer 4 */
    unsigned frz4 : 1;   /* freeze timer 4 */
    unsigned stp4 : 1;   /* stop timer 4 */
    unsigned rst4 : 1;   /* reset timer 4 */
    unsigned gm2   : 1;   /* gate mode pin 2 */
    unsigned frz3 : 1;   /* freeze timer 3 */
    unsigned stp3 : 1;   /* stop timer 3 */
    unsigned rst3 : 1;   /* reset timer 3 */
    unsigned cas2 : 1;   /* cascade timer 2 */
    unsigned frz2 : 1;   /* freeze timer 2 */
    unsigned stp2 : 1;   /* stop timer  2 */
    unsigned rst2 : 1;   /* reset timer 2 */
    unsigned gm1   : 1;   /* gate mode pin 1 */
    unsigned frz1 : 1;   /* freeze timer 1 */
    unsigned stp1 : 1;   /* stop timer 1 */
    unsigned rst1 : 1;   /* reset timer 1 */
} TimerGenConfigReg;

typedef struct     /*   timer mode register 16 bit */
{
    unsigned ps    : 8;     /* prescaler */
    unsigned ce    : 2;     /* capture edge */
    unsigned om    : 1;     /* output mode */
    unsigned ori   : 1;     /* output ref interrupt enable */
    unsigned frr   : 1;     /* free run/restart */
    unsigned iclk : 2;     /* input clock source */
    unsigned ge    : 1;     /* gate enable */
} TimerModeReg;

typedef struct     /* timer event register 16 bit */
```

广东省网络空间安全协会受控资料

```
{
   unsigned      : 14;    /* unused */
   unsigned ref : 1;      /* output reference event */
   unsigned cap : 1;      /* input capture event */
} TimerEventReg;


/* structure used for byte by byte crc checking on receive */

typedef struct
{
   uint16 poly;
   uint16 crc;
   uint8 crcBit;
   uint8 dataBit;
   uint8 dataByte;
} CRCParam;

/* structures used for debugging purposes */

typedef struct   /* contains record of SPM frame */
{
   SPMState state;  /* state when received status byte */
   SPMRxFrame rf;      /*   RX frame */
   SPMTxFrame tf;      /* TX frames */
   uint32 duration;   /* time to run isr */
   uint32 start;      /* start time of isr */
   uint16 rb;          /* index to receive buffer */
   uint16 tb;          /* index to transmit bufer */
} Record;


typedef struct /* contains history of frame records */
{
   int index; /* current record wraps around when full */
   Record records[NUM_HIST];   /* array of records */
} History;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Section: Local Globals
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* pointers to SPI registers   all are constant pointers to volatiles
   volatile is needed to insure mem access occurs at each reference */
```

广东省网络空间安全协会受控资料

```
static volatile CPCommandReg    * const crPtrGbl =
            (volatile CPCommandReg * ) CR;
static volatile SDMAConfigReg    * const sdcrPtrGbl =
            (volatile SDMAConfigReg * ) SDCR;

static volatile CIConfigReg    * const cicrPtrGbl =
            (volatile CIConfigReg * ) CICR;
static volatile CISourceReg    * const ciprPtrGbl =
            (volatile CISourceReg * ) CIPR;
static volatile CISourceReg    * const cimrPtrGbl =
            (volatile CISourceReg * ) CIMR;
static volatile CISourceReg    * const cisrPtrGbl =
            (volatile CISourceReg * ) CISR;

static volatile SIClockRouteReg    * const sicrPtrGbl =
            (volatile SIClockRouteReg * ) SICR;

static volatile SPIModeReg * const spmodePtrGbl =
            (volatile SPIModeReg * ) SPMODE;
static volatile SPIEventMaskReg    * const spiePtrGbl =
            (volatile SPIEventMaskReg * ) SPIE;

static volatile SPIEventMaskReg    * const spimPtrGbl =
            (volatile SPIEventMaskReg * ) SPIM;
static volatile SPICommandReg    * const spcomPtrGbl =
            (volatile SPICommandReg * ) SPCOM;

static volatile SPIParamRam    * const spiParamPtrGbl =
            (volatile SPIParamRam * ) SPIB;
static volatile RBufferDesc    * const rbdPtrGbl =
            (volatile RBufferDesc * ) RBD_BASE;
static volatile TBufferDesc    * const tbdPtrGbl =
            (volatile TBufferDesc * ) TBD_BASE;


/* SCC2 registers */
static volatile GSMRegLow    * const gsmrl2PtrGbl =
            (volatile GSMRegLow * ) GSMRL2;
static volatile GSMRegHigh    * const gsmrh2PtrGbl =
            (volatile GSMRegHigh * ) GSMRH2;
static volatile uint16 * const dsr2PtrGbl =
            (volatile uint16 * ) DSR2;
```

广东省网络空间安全协会受控资料

```
static volatile uint16 * const todr2PtrGbl =
            (volatile uint16 * ) TODR2;
static volatile SCCEventMaskReg   * const scce2PtrGbl =
            (volatile SCCEventMaskReg * ) SCCE2;
static volatile SCCEventMaskReg   * const sccm2PtrGbl =
            (volatile SCCEventMaskReg * ) SCCM2;
static volatile SCCStatusReg   * const sccs2PtrGbl =
            (volatile SCCStatusReg * ) SCCS2;
static volatile BRGConfigReg   * const brgc2PtrGbl =
            (volatile BRGConfigReg * ) BRGC2;

static volatile SCCParamRam   * const scc2ParamPtrGbl =
            (volatile SCCParamRam * ) SCC2B;
static volatile SCCReceiveBD   * const scc2rbdPtrGbl =
            (volatile SCCReceiveBD * ) RBD_BASE_SCC2;
static volatile SCCTransmitBD   * const scc2tbdPtrGbl =
            (volatile SCCTransmitBD * ) TBD_BASE_SCC2;

/* allocate storage for scc transmit and receive buffers,
   init to zeros */
static volatile Byte scc2rBufGbl[NUM_BD_SCC2][MAX_BL_SCC2] = {0};
static volatile Byte scc2tBufGbl[NUM_BD_SCC2][MAX_BL_SCC2] = {0};

/* Port A registers */
static volatile PADirectionReg   * const padirPtrGbl =
            (volatile PADirectionReg * ) PADIR;
static volatile PAPinAssignmentReg   * const paparPtrGbl =
            (volatile PAPinAssignmentReg * ) PAPAR;
static volatile PAOpenDrainReg   * const paodrPtrGbl =
            (volatile PAOpenDrainReg * ) PAODR;
static volatile PADataReg   * const padatPtrGbl =
            (volatile PADataReg * ) PADAT;

/* Port B registers */
static volatile PBDirectionReg   * const pbdirPtrGbl =
            (volatile PBDirectionReg * ) PBDIR;
static volatile PBPinAssignmentReg   * const pbparPtrGbl =
            (volatile PBPinAssignmentReg * ) PBPAR;
static volatile PBOpenDrainReg   * const pbodrPtrGbl =
            (volatile PBOpenDrainReg * ) PBODR;
static volatile PBDataReg   * const pbdatPtrGbl =
            (volatile PBDataReg * ) PBDAT;
/* Port C registers */
```

广东省网络空间安全协会受控资料

```
static volatile PCDirectionReg    * const pcdirPtrGbl =
            (volatile PCDirectionReg * ) PCDIR;
static volatile PCPinAssignmentReg    * const pcparPtrGbl =
            (volatile PCPinAssignmentReg * ) PCPAR;
static volatile PCDataReg    * const pcdatPtrGbl =
            (volatile PCDataReg * ) PCDAT;
static volatile PCSpecialOptionsReg * const pcsoPtrGbl =
            (volatile PCSpecialOptionsReg * ) PCSO;
static volatile PCInterruptReg * const pcintPtrGbl =
            (volatile PCInterruptReg * ) PCINT;

/* const ptr to ptr to function returning void with void params */
static volatile void  ( * * const spivPtrGbl)(void) =
            (volatile void  ( * * )(void)) SPIV; /* ptr to vector */
static volatile void  ( * * const cerrvPtrGbl)(void) =
            (volatile void  ( * * )(void)) CERRV; /* ptr to vector */
static volatile void  ( * * const spurintvPtrGbl)(void) =
            (volatile void  ( * * )(void)) SPURINTV; /* ptr to vector */
static volatile void  ( * * const buserrvPtrGbl)(void) =
            (volatile void  ( * * )(void)) BUSERRV; /* ptr to vector */

/* Timer 1—2 stuff */
static volatile TimerGenConfigReg    * const tgcrPtrGbl =
            (volatile TimerGenConfigReg * ) TGCR;

static volatile uint32    * const trr12PtrGbl =
            (volatile uint32 * ) TRR1; /* cascaded 32 bit ref */
static volatile uint32    * const tcn12PtrGbl =
            (volatile uint32 * ) TCN1; /* cascaded 32 bit timer */

static volatile TimerModeReg    * const tmr2PtrGbl =
            (volatile TimerModeReg * ) TMR2;
static volatile uint16    * const trr2PtrGbl =
            (volatile uint16 * ) TRR2;
static volatile uint16    * const tcr2PtrGbl =
            (volatile uint16 * ) TCR2;
static volatile uint16    * const tcn2PtrGbl =
            (volatile uint16 * ) TCN2;
static volatile TimerEventReg    * const ter2PtrGbl =
            (volatile TimerEventReg * ) TER2;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Section: Globals
```

广东省网络空间安全协会受控资料

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* parameters for SPMIsr execution history for debugging purposes */
volatile MACParam macGbl;   /* extern in physical.h */
volatile SPMParam spmGbl;   /* extern in physical.h */
volatile CRCParam crcGbl;

volatile History hBufGbl = {0}; /* Initialize to zero */

/* exception count for debugging puposes. Incremented in exception
   service routines for bus error and spurious interrupt */
volatile uint32 exceptions = 0;

/* Timer for PHYIO */
MsTimer phyIOTimer;

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Section: Local Function Prototypes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Special Purpose Mode Init */
static int16    SPMInit(void);

/* Special Purpose Mode Int. Service Routine */
static void     SPMIsr(void);

/* CPM int error Int. Service Routive */
static void     CErrIsr(void);

/* Spurious interrupt error ISR */
static void     SpurIntIsr(void);

/* bus error ISR */
static void     BusErrIsr(void);

/* Initialize MAC hardware timer */
static uint32 * MACTimerInit(void);

/* delay function waits for delay ticks */
static Boolean DelayTicks(uint32 delay);

/* uses MACTimer hardware timer */
static void     UpdateElapsedTimer(TimerData32 * t);
static void     StartElapsedTimer(TimerData32 * t);
/* increments backlog */
```

广东省网络空间安全协会受控资料

```
static void      IncrementBacklog(uint8 deltaBacklog);
/* decrement backlog */
static void      DecrementBacklog(uint8 deltaBacklog);

/*-----------------------------------------------------------------
Section: Function Definitions
-----------------------------------------------------------------*/
/* #define SPM_TEST */      /* To include a main to test SPM only */
/* #define SPM_HISTORY */ /* To debug without break points */


/*****************************************************************
Function:  main
Returns:
Reference:
Purpose:   To test mac layer only
Comments:
*****************************************************************/

#ifdef SPM_TEST
void main()
{
   long count = 0;

   PHYInitSPM();


   macGbl.tl = 8;   /* 5 byte packet */
   macGbl.tc = 0;
   macGbl.tPkt[0] = 1;
   macGbl.tPkt[1] = 2;
   macGbl.tPkt[2] = 3;
   macGbl.tPkt[3] = 4;
   macGbl.tPkt[4] = 5;
   macGbl.tPkt[5] = 6;
   macGbl.tPkt[6] = 7;
   macGbl.tPkt[7] = 8;
   macGbl.tpr = TRUE;

   count = 0;
   while (macGbl.tpr ! = FALSE )
   {
      count++;
```

广东省网络空间安全协会受控资料

```
    }


    count = 0;
    while (spmGbl.mode ! = STOP )
    {
        count++;
    }
    return;
}

#endif

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   PHYInitSPM
Returns:
Reference:
Purpose:    Set up special purpose mode
            SPMInit() initializes all the 68360 registers
            MACTimerInit() starts up a 32 bit hardware timer (25Mhz)
            PHYEnableSPMIsr() configures the Interrupt Service Routine
Comments:
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void PHYInitSPM(void)
{
    int16 initOK = 0;

    exceptions = 0; /* for debugging */

    initOK = SPMInit();

    /* initialize and start MAC hardware timer */
    spmGbl.clock = MACTimerInit();

    /* reconfigure SPI SPMODE */
    /* spmodePtrGbl->loop   = 1; */    /* local loop back for test */

    hBufGbl.index = 0;   /* start history at 0 */


    /* updates communications parameters
       brings out of reset and starts frame exchange
       and loads config regs
```

广东省网络空间安全协会受控资料

```
    */
    PHYEnableSPMIsr();

    return;
}

/*****************************************************************
Function:   SPMInit
Returns:    1 if ok,0 if error.
Reference:
Purpose:    set up special purpose mode
            Basic approach: Use SPI to tx and rx frames from xcvr.
            The spi bit clock is wired externally to clk3 input
            This is used to clock the scc2 port that generates
            the frame clock.
Comments:
*****************************************************************/
static int16 SPMInit()
{
    int16 i;                        /* for loop counter */
    SDMAConfigReg sdcrTemp;         /* scratch to set up sdcr */
    CPCommandReg crTemp;            /* scratch to set up cr */
    SCCReceiveBD srbdTemp;          /* scratch */
    SCCTransmitBD stbdTemp;         /* scratch */
    SCCEventMaskReg scceTemp;       /* scratch */
    CIConfigReg cicrTemp;           /* scratch to set up cicr */
    SPIEventMaskReg spiemTemp;      /* scratch to set up spie or spim */
    SPIModeReg spmodeTemp;          /* scratch to set up spmode */

    /* initialize registers and vectors */

    /* initialize SDMA Config Register */
    sdcrTemp.frz = 0;           /* ignore freeze */
    sdcrTemp.sism = 7;          /* level 7 interrupt mask */
    sdcrTemp.said = 4;          /* level 4 bus arbitration priority */
    sdcrTemp.inte = 0;          /* no error interrupts */
    sdcrTemp.intb = 0;          /* no break interrupts */
    *sdcrPtrGbl = sdcrTemp; /* write out sdcr */


    /* set sicr */
    sicrPtrGbl->sc2 = 0;        /* scc2 in nmsi mode */
    sicrPtrGbl->r2cs = 6;       /* receive clock from clk3 */
```

广东省网络空间安全协会受控资料

```
sicrPtrGbl->t2cs = 6;    /* transmit clock from clk3 */

/*****************************************************************/
/* initialize SCC2 Port for frame clock */
/*****************************************************************/
/* initialize RBASE and TBASE in SCC2 Param RAM */
scc2ParamPtrGbl->rbase = (uint16)  RBD_BASE_SCC2;
scc2ParamPtrGbl->tbase = (uint16)  TBD_BASE_SCC2;

/* initialize other parts of SCC2 parameter ram using CP command */
crTemp.reset =0;                    /* no cpm reset */
crTemp.chnum = SCC2_CH_NUM;         /* SCC2 Channel */
crTemp.opcode = INIT_TRP_OPCODE; /* init param ram */
crTemp.flag = 1;                    /* start command */
*crPtrGbl = crTemp;                 /* write to command register */

#ifndef SIMULATION
/* wait for command to finish */
while ( crPtrGbl->flag == 1)
{
    ; /* Do nothing */
}
#endif

/* initialize function codes for param ram  and buffer length
   for scc2 */
/* endian and function code */
scc2ParamPtrGbl->rfcr = (uint8) FCR_INIT;
/* endian and function code */
scc2ParamPtrGbl->tfcr = (uint8) FCR_INIT;
/*   max receive buffer length */
scc2ParamPtrGbl->mrblr = (uint16) 2;

/* initialize scc receive buffer descriptors */
srbdTemp.e = 1;          /* 1 = buffer ready to receive */
srbdTemp.w = 0;          /* 0 = don't wrap this bd */
srbdTemp.i = 0;          /* 0 = disable interrupts */
srbdTemp.l = 0;          /* 0 = not last */
srbdTemp.cm = 1;         /* 1 = put in continuous mode */
srbdTemp.de = 0;         /* clear out dpll error */
srbdTemp.no = 0;         /* clear out non octet error */
srbdTemp.cr = 0;         /* clear out crc error */
srbdTemp.ov = 0;         /* clear out receiver overrun error */
```

广东省网络空间安全协会受控资料

```
srbdTemp. cd = 0;            /* clear out carrier detect lost error */
srbdTemp. dataLen = 0;       /* superfluous   set by cp */
/* set to first buffer but change below */
srbdTemp. dataPtr = &scc2rBufGbl[0][0];

for (i = 0; i < NUM_BD_SCC2; i++) /* loop thru buf descriptors */
{
    scc2rbdPtrGbl[i] = srbdTemp; /* assign defaults from temp */
    /* assign ptr to buffer */
    scc2rbdPtrGbl[i]. dataPtr = &scc2rBufGbl[i][0];
}
scc2rbdPtrGbl[NUM_BD_SCC2 - 1]. w = 1; /* wrap last rbd */

/* initialize scc transmit buffer descriptors */
stbdTemp. r = 1;             /* 1 = buffer ready to transmit */
stbdTemp. w = 0;             /* 0 = don't wrap this bd */
stbdTemp. i = 0;             /* 0 = disable interrupts   */
stbdTemp. l = 0;             /* 0 = not last */
stbdTemp. tc = 0;            /* 0 = don't transmit crc */
stbdTemp. cm = 1;            /* 1 = put in continuous mode */
stbdTemp. un = 0;            /* clear out transmit underrunn error */
stbdTemp. ct = 0;            /* clear out cts lost error */
stbdTemp. dataLen = 2;       /* set to buffer length should be
                                2 bytes */
/* set to first buffer but change below */
stbdTemp. dataPtr = &scc2tBufGbl[0][0];

for (i = 0; i < NUM_BD_SCC2; i++) /* loop thru buf descriptors */
{
    scc2tbdPtrGbl[i] = stbdTemp; /* assign defaults from temp */
    /* assign ptr to buffer */
    scc2tbdPtrGbl[i]. dataPtr = &scc2tBufGbl[i][0];
}
scc2tbdPtrGbl[NUM_BD_SCC2 - 1]. w = 1; /* wrap last tbd */

/* initialize port A and C for SCC2 Transparent mode operation */
padirPtrGbl ->dr2 = 0;    /* RXD for scc2 */
paparPtrGbl ->dd2 = 1;    /* rxd connect internal */

paodrPtrGbl ->od3 = 0;    /* not open drain   for txd */
padirPtrGbl ->dr3 = 0;    /* TXD for scc2 */
paparPtrGbl ->dd3 = 1;    /* txd   connect internal */
```

广东省网络空间安全协会受控资料

```
padirPtrGbl->dr10 = 0;   /* clk3 sc2 */
paparPtrGbl->dd10 = 1;   /*   connect internal */
/* enable rts for scc2 */
pcdirPtrGbl->dr1 = 0;
pcparPtrGbl->dd1 = 1;
/* general purpose io so CD always asserted */
pcdirPtrGbl->dr7 = 1;
pcparPtrGbl->dd7 = 0;
pcsoPtrGbl->cd2 = 0;
/* use cts2 to initiate transmission. configure as gen purpose
   io so always asserted low */
pcdirPtrGbl->dr6 = 1;
pcparPtrGbl->dd6 = 0;
pcsoPtrGbl->cts2 = 0; /* configure as low always */

/* set gsmr */

gsmrh2PtrGbl->tcrc = 0; /* 16 bit ccitt crc */
gsmrh2PtrGbl->revd = 1; /* 1= send msbit of each byte out first */
gsmrh2PtrGbl->trx = 1;   /* receiver in transparent mode */
gsmrh2PtrGbl->ttx = 1;   /* transmitter in transparent mode */
gsmrh2PtrGbl->cdp = 1;   /* cd pulse mode */
gsmrh2PtrGbl->ctsp = 1; /* cts pulse mode */
gsmrh2PtrGbl->cds = 1;   /* cd synchronous   mode */
gsmrh2PtrGbl->ctss = 1; /* cts synchronous mode */
gsmrh2PtrGbl->synl = 0; /* receive sync on cd not on
                           sync pattern */
gsmrh2PtrGbl->tfl = 1;   /* 1 byte fifo for lower transmit
                           latency   */

gsmrl2PtrGbl->diag = 1;   /* local loop back for test */

/* initialize transparent mode crc type */
scc2ParamPtrGbl->crcp = 0x0000FFFFUL;   /* 16 bit CCITT CRC */
scc2ParamPtrGbl->crcc = 0x0000F0B8UL;   /* 16 bit CCITT CRC */

/* clear any previous scc2 interrupt events,write 1 to clear */
scceTemp.glr = 1;
scceTemp.glt = 1;
scceTemp.dcc = 1;
scceTemp.gra = 1;
scceTemp.txe = 1;
scceTemp.rch = 1;
```

广东省网络空间安全协会受控资料

```
    scceTemp.bsy = 1;
    scceTemp.tx  = 1;
    scceTemp.rx  = 1;
    *scce2PtrGbl = scceTemp;   /* write it out to SCCE2 */

    /* enable/disable interrupts write 1 to enable */
    scceTemp.glr = 0;
    scceTemp.glt = 0;
    scceTemp.dcc = 0;
    scceTemp.gra = 0;
    scceTemp.txe = 0;
    scceTemp.rch = 0;
    scceTemp.bsy = 0;
    scceTemp.tx  = 0;
    scceTemp.rx  = 0;
    *sccm2PtrGbl = scceTemp;   /* write it out to sccm2 */

    /* clear any old scc2 interrupts from CISR */
    /* clear by writing 1,spi interrupt in service */
    cisrPtrGbl->scc2 = 1;

    /* enable/disable system interrupt for spi in CIMR */
    cimrPtrGbl->scc2 = 0;              /* enable by writing 1 */

    /*****************************************************************/
    /*    SPI Port Initialization */
    /*****************************************************************/

    /* initialize RBASE and TBASE in SPI Param RAM */
    /* start of Receive BDs */
    spiParamPtrGbl->rbase = (uint16) RBD_BASE;
    /* start of transmit BDs */
    spiParamPtrGbl->tbase = (uint16) TBD_BASE;

    /* initialize other parts of SPI parameter ram using CP command */
    crTemp.reset =0; /* no cpm reset */
    crTemp.chnum = SPI_CH_NUM; /* SPI Channel */
    crTemp.opcode = INIT_TRP_OPCODE; /* init param ram */
    crTemp.flag = 1; /* start command */
    *crPtrGbl = crTemp; /* write to command register */

#ifndef  SIMULATION
    /* wait for command to finish */
```

广东省网络空间安全协会受控资料

```
    while ( crPtrGbl->flag == 1)
    {
        ;
    }
#endif

    /* initialize function codes for param ram   and buffer length */
    /* endian and function code */
    spiParamPtrGbl->rfcr = (uint8) FCR_INIT;
    /* endian and function code */
    spiParamPtrGbl->tfcr = (uint8) FCR_INIT;
    /* maximum buffer length */
    spiParamPtrGbl->mrblr = (uint16) MAX_BUF_LEN;

    /* initialize receive buffer descriptors */
    rbdPtrGbl->e = 1;              /* 1 = buffer ready to receive */
    rbdPtrGbl->w = 1;              /* 1 = wrap this bd */
    rbdPtrGbl->i = 1;              /* 1 = enable interrupts of rxb */
    rbdPtrGbl->cm = 1;             /* 1 = continuous mode */
    rbdPtrGbl->l = 0;              /* clear out */
    rbdPtrGbl->ov = 0;             /* clear out */
    rbdPtrGbl->me = 0;             /* clear out */
    rbdPtrGbl->dataLen = 0;        /* superfluous set by cp */
    rbdPtrGbl->dataPtr = &spmGbl.rf;          /* assign ptr to buffer */

    /* initialize transmit buffer descriptors */
    tbdPtrGbl->r = 1; /* 1 = buffer ready to transmit */
    tbdPtrGbl->w = 1; /* 1 = wrap this bd */
    tbdPtrGbl->i = 0; /* 0 = don't enable interrupts of txb */
    tbdPtrGbl->cm = 1; /* 1 = continuous mode */
    tbdPtrGbl->l = 1; /* 1 = last   */
    tbdPtrGbl->un = 0; /* clear out */
    tbdPtrGbl->me = 0; /* clear out */
    tbdPtrGbl->dataLen = 2; /* set to buffer length should be 2bytes */
    tbdPtrGbl->dataPtr = &spmGbl.tf; /* assign ptr to buffer */

/*****************************************************************
  Initialize Globals Structure for Channel Access Algorithm
  and Special Purpose Mode
*****************************************************************/

    /* initialize receive frame buffers just to prevent garbage
       in case out of sync */
```

广东省网络空间安全协会受控资料

```
spmGbl. rf. setTxFlag = 0;      / * frame status not transmitting * /
spmGbl. rf. clrTxReqFlag = 0; / * don't clear * /
spmGbl. rf. rxDataValid = 0;   / * no valid data this frame * /
spmGbl. rf. txDataCTS = 0;      / * no clear to send * /
spmGbl. rf. setCollDet = 0;    / * no collision detected * /
spmGbl. rf. rxFlag = 0;         / * not receiving * /
spmGbl. rf. rwAck = 0;          / * not acknowledged * /
spmGbl. rf. txOn = 0;           / * not receiving * /
spmGbl. rf. data = 0;           / *    * /

/ * initialize receive frame buffers just to prevent garbage in case out of sync * /
spmGbl. tf. txFlag = 0;         / * frame status not transmitting * /
spmGbl. tf. txReqFlag = 0;     / * don't req * /
spmGbl. tf. txDataValid = 0; / * no valid data this frame * /
spmGbl. tf. blank = 0;          / * unused * /
spmGbl. tf. txAddrRW = 0;       / * write * /
spmGbl. tf. txAddr = 0;         / * default address * /
spmGbl. tf. data = 0;           / *    * /

/ * Initialize spm global parameters * /

spmGbl. mode = STOP;            / * no errors yet * /
spmGbl. state = IDLE;           / * state machine start state * /
macGbl. tpr = FALSE;            / * packet not ready to transmit * /
macGbl. tc = 0;                 / * transmit byte count 0 * /
macGbl. tl = 0;                 / * last byte is 0 * /
macGbl. rpr = FALSE;            / * packet not yet received * /
macGbl. rc = 0;                 / * receive byte count 0 * /
macGbl. rl = 0;                 / * last byte is 0 * /
spmGbl. crw = FALSE;            / * don't write config register * /
spmGbl. cra = 0;                / * address zero nothing * /
spmGbl. crData = 0;             / * empty data * /
spmGbl. srr = FALSE;            / * don't read from status register * /
spmGbl. sra = 0;                / * empty address * /
spmGbl. srData = 0;             / * put nothing in * /
spmGbl. resetCount = 0;         / * zero to start * /
spmGbl. collisionsThisPkt = 0 ; / * # of collisions this packet * /
macGbl. priorityPkt = FALSE; / * FALSE = not a priority packet * /

/ * FALSE = channel access algo not complete * /
spmGbl. accessApproved = FALSE;
/ * TRUE means cycle timer to be reset and started * /
spmGbl. cycleTimerRestart = TRUE;
```

广东省网络空间安全协会受控资料

```
spmGbl. backlog = 0;   /* current channel backlog */
/* delta backlog on current transmit packet */
macGbl. deltaBLTx = 0;
macGbl. deltaBLRx = 0;   /* delta backlog on last receive packet */
macGbl. altPathBit = 0; /* alternate path bit */
/* signal to write alternate path bit */
spmGbl. writeAltPathBit = FALSE;
/* alt path bit written state for this pkt */
spmGbl. altPathBitWritten = FALSE;
spmGbl. nodePriority = 0; /* node's priority slot number */
/* comm parameters for this node */
for ( i = 0; i < NUM_COMM_PARAMS; i++)
{
    spmGbl. configData[i] = 0;
}
spmGbl. phase = RANDOM_IDLE; /* Idle for a long time */

spmGbl. kind = POST_RX;      /* beta1 time slot type */
spmGbl. nicsToTicks = 15;   /* conversion factor this specification's
                                time base to 68360 ticks */
spmGbl. bitClockRate = 156250; /* in units of Hz */
spmGbl. beta2Ticks = 0;    /* length of beta2 in 68360 ticks
                              40ns each */
spmGbl. beta1Ticks = 0;    /* length of beta1 in 68360 ticks
                              40ns each */
spmGbl. beta1PostTxTicks = 0;    /* length of beta1 in 68360 ticks
                                    40ns each */
spmGbl. beta1PostRxTicks = 0;    /* length of beta1 in 68360 ticks
                                    40ns each */
spmGbl. baseTicks = 0; /* duration of wbase in 68360 ticks
                          40ns each */
spmGbl. cycleTicks = 0;    /* duration of avg packet cycle in
                              68360 ticks 40ns each */
spmGbl. priorityChPostTxTicks = 0; /* duration of channel priority
                                      slots */
spmGbl. priorityChPostRxTicks = 0; /* duration of channel priority
                                      slots */
spmGbl. priorityIdleTicks = 0;   /* duration of priority idle wait */
/* duration until node's priority slot */
spmGbl. priorityNodeTicks   = 0;
spmGbl. randomTicks         = 0;
/* timers for channel access algorithm */
```

广东省网络空间安全协会受控资料

```
spmGbl. idleTimerStart        = 0;
spmGbl. baseTimerStart        = 0;
spmGbl. cycleTimerStart       = 0;
spmGbl. transmitTimerStart = 0;
spmGbl. elapsed               = 0;
spmGbl. stopped               = 0;
spmGbl. lastTime              = 0;

/* initialize crcGbl   for good form */
crcGbl. poly = 0;
crcGbl. crc = 0;
crcGbl. crcBit = 0;
crcGbl. dataBit = 0;
crcGbl. dataByte = 0;

/* initialize port B for SPI Master operation */
/* pin0 = spisel or chip select if single master */
/* pin1 = spiclk    clock */
/* pin2 = spimosi master out slave in */
/* pin3 = spimiso master in slave out */
/* pin5 = ~reset */

pbodrPtrGbl->od0 = 0;   /*0=active driven 1=open drain(3 state),if output */
pbodrPtrGbl->od1 = 0;
pbodrPtrGbl->od2 = 0;
pbodrPtrGbl->od3 = 0;
pbodrPtrGbl->od5 = 0;   /* not open drain for reset? */

pbparPtrGbl->dd0 = 0;   /* general purpose pin chip select */
pbparPtrGbl->dd1 = 1;   /* internal connect spiclk */
pbparPtrGbl->dd2 = 1;   /* internal connect spimosi */
pbparPtrGbl->dd3 = 1;   /* internal connect spimiso */
pbparPtrGbl->dd5 = 0;   /* general purpose i/o */

pbdirPtrGbl->dr0 = 1;   /* output */
pbdirPtrGbl->dr1 = 1;   /* output */
pbdirPtrGbl->dr2 = 1;   /* output */
/* although input must config as out else brgo4 */
pbdirPtrGbl->dr3 = 1;
pbdirPtrGbl->dr5 = 1;   /* output */


/* initialize port B for IO pins : reset switch,manual service request,
```

广东省网络空间安全协会受控资料

```
    ioswitch,and LEDs reset-service and IOLED
    pb8 = led reset service
    pb9 = general purpose out led
    pb12 = reset switch
    pb13 = manual service request
    pb14 = general purpose in switch
*/

pbodrPtrGbl->od8 = 0;    /* 0 = active driven not open drain    */
pbodrPtrGbl->od9 = 0;
/* since input NA,however open drain for pull down */
pbodrPtrGbl->od12 = 1;
/* since input NA,however open drain for pull down */
pbodrPtrGbl->od13 = 1;
/* since input NA,however no pull down */
pbodrPtrGbl->od14 = 0;

pbparPtrGbl->dd8 = 0;          /* general purpose pin chip select */
pbparPtrGbl->dd9 = 0;          /* general purpose pin chip select */
pbparPtrGbl->dd12 = 0;         /* general purpose pin chip select */
pbparPtrGbl->dd13 = 0;         /* general purpose pin chip select */
pbparPtrGbl->dd14 = 0;         /* general purpose pin chip select */

pbdirPtrGbl->dr8 = 1;          /* output */
pbdirPtrGbl->dr9 = 1;          /* output */
pbdirPtrGbl->dr12 = 0;         /* input */
pbdirPtrGbl->dr13 = 0;         /* input */
pbdirPtrGbl->dr14 = 0;         /* input */

/* set initial values */
pbdatPtrGbl->d8 = 0;
pbdatPtrGbl->d9 = 0;
pbdatPtrGbl->d12 = 1; /* input so NA */
pbdatPtrGbl->d13 = 1; /* input so NA */
pbdatPtrGbl->d14 = 0; /* input so NA */

/* reset xcvr */
/* driven low to reset hold in reset until change.  */
pbdatPtrGbl->d5 = 0;

/* clear any previous spi interrupt events write 1 to clear */
spiemTemp.mme = 1;
spiemTemp.txe = 1;
```

广东省网络空间安全协会管控资料

```
spiemTemp. bsy = 1;
spiemTemp. txb = 1;
spiemTemp. rxb = 1;
* spiePtrGbl = spiemTemp;   /* write it out to SPIE */

/* assign address of isr functions to vector table entry */
* spivPtrGbl = &SPMIsr;            /* spi isr */
* cerrvPtrGbl = &CErrIsr;          /* comm error isr */
* spurintvPtrGbl = &SpurIntIsr; /* spurious interrupt isr */
/*  * buserrvPtrGbl = &BusErrIsr; */ /* spurious interrupt isr */

/* enable interrupts write 1 to enable */
spiemTemp. mme = 0;
spiemTemp. txe = 0;
spiemTemp. bsy = 0;
spiemTemp. txb = 0;   /* isr will run after transmit */
spiemTemp. rxb = 1;   /* isr will run after receive buffer filled */
* spimPtrGbl = spiemTemp;   /* write it out to SPIM */

/* init CICR register */
cicrTemp. scdp = 3; /* scc4 has priority d */
cicrTemp. sccp = 2; /* scc3 has priority c */
cicrTemp. scbp = 1; /* scc2 has priority b */
cicrTemp. scap = 0; /* scc1 has priority a */
cicrTemp. irl   = CPIR_LEVEL; /* level 4 interrupt request level */
cicrTemp. hpi   = 31; /* default highest priority = 31 = 1F hex */
cicrTemp. vba   = UIVN_MSB; /* user interrupt vector number offset */
cicrTemp. sps   = 0; /* use grouped priority scheme */
* cicrPtrGbl = cicrTemp; /* write to cicr */

/* clear any old SPI interrupts from CISR */
cisrPtrGbl ->spi = 1; /* clear by writing 1,
                         spi interrupt in service */
/* enable system interrupt for spi in CIMR */
cimrPtrGbl ->spi = 1; /* enable by writing 1 */

/* init SPMODE register */
spmodeTemp. loop   = 0; /* 1 = local loop back
                           0 = no local loop back */
spmodeTemp. ci     = 0; /* don't invert clock */
spmodeTemp. cp     = 1; /* toggle clock at begin of data transfer */
spmodeTemp. div16 = 0; /* don't divide brgclk by 16 */
spmodeTemp. rev    = 1; /* use bigendian MSB first */
```

广东省网络空间安全协会受控资料

```
    spmodeTemp.ms      = 1; /* master mode */
    spmodeTemp.len     = 7; /* 8 bits per character = len + 1 */
    spmodeTemp.pms     = 4; /* prescale modulus select,
                               4 gives 1.25 Mbps */
    spmodeTemp.en      = 1; /* enable spi    */
    *spmodePtrGbl = spmodeTemp; /* write to spmode register */

    /* initialize SPCOM (probably don't need to do */
    spcomPtrGbl->res = 0;   /* should never have to change */
    spcomPtrGbl->str = 0;   /* superfluous auto cleared one
                               clock cycle after set */

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     Enable interrupts by setting interrupt priority mask level
     in Status Register to 0,must use in line assembly code
     assumes CPU is in supervisor mode,Bits 10 9 8 are
     interrupt priority mask level
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

    asm(" ANDI #~0x0700,SR"); /* sets priority mask to level 000 */


    /* Set up frame clock using scc2 */
    /* put data into transmit buffer */
    scc2tBufGbl[0][0] = 0x01U;      /* first byte 7 bit delay
                                       to sync frame to spi */
    scc2tBufGbl[0][1] = 0x00U;      /* second byte */
    scc2tbdPtrGbl[0].dataLen = 2; /* send just two bytes */


    /* enable frame clock */
    gsmrl2PtrGbl->enr = 1;   /* receive */
    gsmrl2PtrGbl->ent = 1;   /* transmit */


    return(1);
}


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   MACTimerInit
Returns:    pointer to hardware timer counter register
Reference:
```

广东省网络空间安全协会受控资料

```
Purpose:    initialize MAC   hardware timer
Comments:
*****************************************/

uint32 * MACTimerInit(void)
{
   tgcrPtrGbl->cas2 = 1; /* cascade 1 and 2 */
   tgcrPtrGbl->frz2 = 0; /* ignore freeze pin */
   tgcrPtrGbl->stp2 = 0; /* normal unstopped operation */
   tgcrPtrGbl->rst2 = 0; /* reset timer */

   tmr2PtrGbl->ps   = 0; /* prescaler */
   tmr2PtrGbl->ce   = 0; /* input capture disabled */
   tmr2PtrGbl->om   = 0; /* output mode pulse */
   tmr2PtrGbl->ori  = 0; /* diable output reference interrupt */
   tmr2PtrGbl->frr  = 0; /* 0= free running counter */
   tmr2PtrGbl->iclk = 1; /* 1 = internal general system clk source */
   tmr2PtrGbl->ge   = 0; /* tgate is ignored */

   /* set reference to max all ones */
   *trr12PtrGbl = (uint32) 0xFFFFFFFFU;

   /* writing a 1 resets the reference event bit */
   ter2PtrGbl->ref = 1;
   /* writing a 1 resets the capture event bit */
   ter2PtrGbl->cap = 1;

   *tcn12PtrGbl = 0; /* start at zero */

   tgcrPtrGbl->rst2 = 1; /* start timer running */

   return(tcn12PtrGbl);
}


/****************************************
Function:   DelayTicks
Returns:    Boolean state of mac timer running = true stopped = false
Reference:
Purpose:    Delay for a "delay" number of ticks
            Used during various startup routines
            to wait for xcvr hardware to respond
Comments:
```

广东省网络空间安全协会受控资料

```
***************************************/

Boolean DelayTicks(uint32 delay)
{
   uint32 start;

   if (tgcrPtrGbl->rst2 == 0)
   {
      /* timer not running so exit immediately with false */
      return (FALSE);
   }
   else
   {
      start = *tcn12PtrGbl;

      while ((*tcn12PtrGbl - start) < delay)
      {
         ; /* keep waiting */
      }

      return (TRUE);
   }
}


/***************************************
Function:  PHYSoftResetSPMXCVR
Returns:
Reference:
Purpose:   Software reset Special Purpose Mode XCVR by reading
           from register zero
Comments:
***************************************/

void PHYSoftResetSPMXCVR(void)
{
   uint16 count;

   spmGbl.sra = 0;
   spmGbl.srr = TRUE;

   count = 0;
   while ((spmGbl.srr != FALSE) && (count < 0xFFFF))
```

广东省网络空间安全协会受控资料

```
    {
        count++;
    }
    spmGbl.srData = 0;

    return;
}

/*****************************************************************
Function:   PHYHardResetSPMXCVR
Returns:
Reference:
Purpose:    hardware reset Special Purpose Mode XCVR by
            asserting reset pin. then configures xcvr
            this is used when xcvr hangs in tx_on mode
            e.g. spmGbl.resetCount times out
Comments:
*****************************************************************/

void PHYHardResetSPMXCVR(void)
{
    unsigned long count;
    int i;
    Boolean good;

    pbdatPtrGbl->d5 = 0; /* drive xcvr into reset */

    /* wait for a while for everything to settle down */
    good = DelayTicks(RESTART_DELAY_TICKS);


    pbdatPtrGbl->d5    = 1; /* bring xcvr out of reset */

    /* initialize config registers on PLT-20 transceiver */

    /* load in reverse order */
    i = NUM_COMM_PARAMS - 1; /* should be 6 */
    while ( i >= 0)
    {
        spmGbl.crData = spmGbl.configData[i];
        /* config registers start at 7 down to 1 */
        spmGbl.cra = (Byte) i + 1;
        spmGbl.crw = TRUE;
```

广东省网络空间安全协会受控资料

```
        count = 0;
        /* wait for ack but time out so not infinite loop */
        while ((spmGbl.crw ! = FALSE) && (count < 0xFFFFF))
        {
            count++;
        }
        if (spmGbl.crw == TRUE) /* xcvr never acked try again */
        {
            pbdatPtrGbl->d5    = 0; /* reset xcvr */

            /* wait for a while for everything to settle down */
            good = DelayTicks(RESTART_DELAY_TICKS);

            pbdatPtrGbl->d5    = 1; /* bring xcvr out of reset */
            i = NUM_COMM_PARAMS - 1; /* start over again */
        } /* NOTE!!! if the xcvr never acks this code will hang */
        else
        {
            i--;
        }
    }

    spmGbl.crData = 0;
    spmGbl.cra = 0;

    return;
}

/*********************************************************************
Function:   PHYDisableSPMIsr
Returns:
Reference:
Purpose:    To diable the Isr temporarily on node reset.
            Must use PHYEnableIsr to reenable correctly.
Comments:
*********************************************************************/

void PHYDisableSPMIsr(void)
{
    /* once the first line below is executed,the ISR will stop
       running. the next time it runs */
    spmGbl.mode   = STOP;
```

广东省网络空间安全协会受控资料

```
    pbdatPtrGbl->d5 = 0; /* drive xcvr into reset */

    spmGbl.state = DEBUG;


    macGbl.tpr    = FALSE;
    macGbl.tc     = 0;
    macGbl.tl     = 0;
    macGbl.rpr    = FALSE;
    macGbl.rc     = 0;
    macGbl.rl     = 0;
    spmGbl.crw    = FALSE;
    spmGbl.srr    = FALSE;
    spmGbl.sra = 0;

    return;
}

/*****************************************************************
Function:   PHYEnableSPMIsr
Returns:
Reference:
Purpose:    To Enable the Isr after Being disabled on node reset
            Also to enable on powerup etc.
            Computes clock rates etc based on node comm parameters
Comments:
*****************************************************************/

void PHYEnableSPMIsr(void)
{
    long  i;
    uint32 inputClock;
    uint32 commClock;
    uint32 rxPad;
    uint32 txPad;
    uint32 beta2;
    Boolean good;

    /* compute conversion factor nics_To_Tick   */
    /* converts from this specification's time base to 683660 ticks */
    inputClock = (uint32) eep->configData.inputClock;
    commClock = (uint32) eep->configData.commClock;
    spmGbl.nicsToTicks = NICS_TICKS_BASE >> inputClock;
```

广东省网络空间安全协会受控资料

```
/* compute channel access algorithm parameters */

beta2 = (eep->configData.reserved[2] * 20) + 40;

/* compute beta1 = postpacket + networkidlewait +
                   interpacketpad + prepacket */

if (eep->configData.reserved[4] < 128)
{
    rxPad = eep->configData.reserved[4] * 41;
}
else
{
    rxPad = (eep->configData.reserved[4] - 128) * 145;
}

if (eep->configData.reserved[3] < 128)
{
    txPad = eep->configData.reserved[3] * 41;
}
else
{
    txPad = (eep->configData.reserved[3] - 128) * 145;
}

/* set beta1 time as per formula with implementation
   specific adjustment BETA1_ADJUST_TICKS */
spmGbl.beta1PostRxTicks =
   (285 + beta2 + rxPad + 317) * spmGbl.nicsToTicks +
   BETA1_ADJUST_TICKS;
spmGbl.beta1PostTxTicks =
   (307 + beta2 + txPad + 317) * spmGbl.nicsToTicks +
   BETA1_ADJUST_TICKS;

/* set beta2 time as per formula with implementation
   specific adjustment BETA2_ADJUST_TICKS */
spmGbl.beta2Ticks =
   beta2 * spmGbl.nicsToTicks + BETA2_ADJUST_TICKS;
spmGbl.baseTicks = spmGbl.beta2Ticks * W_BASE;
/* set cycle time as per formula with implementation
   specific adjustment CYCLE_ADJUST_TICKS */
spmGbl.cycleTicks =
```

广东省网络空间安全协会受控资料

```
    eep->configData.reserved[1] * 1794 * spmGbl.nicsToTicks;

spmGbl.beta1Ticks = 0; /* this is set by isr to either post rx or
                          post tx beta1 as appropriate */

spmGbl.nodePriority =
    eep->configData.nodePriority; /* set node's priority slot # */

/* Add W_Base slots to wait time for post tx access only when there
   are non zero number of channel priority slots */
if (eep->configData.channelPriorities > 0)
{
    /* duration of channel priority slots */
    spmGbl.priorityChPostTxTicks = (eep->configData.channelPriorities +
                                    W_BASE) * spmGbl.beta2Ticks;
    /* duration of channel priority slots */
    spmGbl.priorityChPostRxTicks = eep->configData.channelPriorities
                                   * spmGbl.beta2Ticks;
}
else
{
    /* duration of channel priority slots */
    spmGbl.priorityChPostTxTicks = 0;
    spmGbl.priorityChPostRxTicks = 0;
}
/* duration of priority idle wait */
spmGbl.priorityIdleTicks = spmGbl.priorityChPostTxTicks;

/* set time of nodes priority slot. If eep->configData.nodePriority is 0,
   then this field will be negative,but it is not used in this case. */
spmGbl.priorityNodeTicks =
    (eep->configData.nodePriority - 1) * spmGbl.beta2Ticks;
spmGbl.randomTicks = 0;
spmGbl.bitClockRate = (BIT_CLOCK_BASE << inputClock) >>
    (RATIO_BASE + commClock);


/* make local copy of xcvr parameters so can reset xcvr free of changes to eep
   without following node reset. Needed in case xcvr locks up,See
   PHYHardResetSPMXCVR(void)
   */
for (i = 0; i < NUM_COMM_PARAMS; i++)
{
```

广东省网络空间安全协会受控资料

```
    spmGbl. configData[i] = eep->configData. param. xcvrParams[i];
}

/* reinitialize remainder of spmGbl */
spmGbl. mode = RUN;          /* get ready to go */
spmGbl. state = IDLE;       /* state machine start state */
macGbl. tpr = FALSE;        /* packet not ready to transmit */
macGbl. tc = 0;             /* transmit byte count 0 */
macGbl. tl = 0;             /* last byte is 0 */
macGbl. rpr = FALSE;        /* packet not yet received */
macGbl. rc = 0;             /* receive byte count 0 */
macGbl. rl = 0;             /* last byte is 0 */
spmGbl. crw = FALSE;        /* don't write config register */
spmGbl. cra = 0;            /* address zero nothing */
spmGbl. crData = 0;         /* empty data */
spmGbl. srr = FALSE;        /* don't read from status register */
spmGbl. sra = 0;            /* empty address */
spmGbl. srData = 0;         /* put nothing in */
spmGbl. resetCount = 0;     /* zero to start */
spmGbl. collisionsThisPkt = 0 ; /* # of collisions this packet */
macGbl. altPathBit = 0; /* alternate path bit */
spmGbl. writeAltPathBit = FALSE; /* signal to write
                                    alternate path bit */
spmGbl. altPathBitWritten = FALSE; /* alt path bit written
                                      state for this pkt */
macGbl. priorityPkt = FALSE; /* FALSE = not a priority packet */

spmGbl. accessApproved = FALSE; /* FALSE = channel access algo
                                   not complete */
spmGbl. cycleTimerRestart = TRUE;     /* indicates whether cycle
                                    timer can be updated */

spmGbl. backlog = 0;   /* current channel backlog */
macGbl. deltaBLTx = 0;   /* delta backlog on curr transmit packet */
macGbl. deltaBLRx = 0;   /* delta backlog on last receive packet */
spmGbl. phase = RANDOM_IDLE; /* Idle for a long time */

spmGbl. kind = POST_RX;      /* beta1 time slot type */
spmGbl. idleTimerStart = 0;
spmGbl. baseTimerStart = 0;
spmGbl. transmitTimerStart = 0;
spmGbl. elapsed = 0;
/* spmGbl. clock is initialized in PHYInitSPM */
```

广东省网络空间安全协会受控资料

```
spmGbl.stopped = *(spmGbl.clock);
spmGbl.cycleTimerStart = spmGbl.stopped; /* start up 1st time */
spmGbl.lastTime = spmGbl.stopped;


/* configure bit clock */
/* div16 = 1 and pms = 4 gives 78.125 khz */
/* div16 = 1 and pms = 3 gives 97.656 khz */
/* div16 = 1 and pms = 2 gives 130.208 khz */
/* div16 = 1 and pms = 1 gives 195.312 khz */
/* div16 = 1 and pms = 0 gives 390.625 khz */
/* div16 = 0 and pms = 9 gives 625.000 khz */

/*****************************************************************
    Since the 360 implementation runs in noncontinuous mode the
    effective frame period is the actual frame period plus the
    processing time for the ISR at the end of the frame. Therefore
    we need to run at a higher bit clock than specified by the
    network management so that the effective frame rate is the same
    or better than the network manager specifies. For a bit rate of
    156.25 Khz the frame period is 16 * 6.4 usec = 102.4
    usec. Suppose the worst case time for ISR is 58 micro seconds
    then the time left over for the actual frame is 102.4 - 58 =
    44.4 usec. This corresponds to a bit rate of 16 * 1/(44.4 usec)
    = 360.36 khz. The closest bit rate greater than this that the
    360 supports is 390.625 khz. The effective frame period becomes
    58 usec + (16 * (1/390.625 khz)) = 98.96 usec. This is faster so
    OK. Effective bit rate is 16 * 1/(98.96 usec) = 161.68 kHz.
    Unless we can more than half the time for the ISR to run we
    can't run fast enough to make the next step that is the 312.5
    Khz bit rate.
*****************************************************************/

switch (spmGbl.bitClockRate)
{
    case 78125:
        /* run at 130 khz to get same or faster effective rate */
        spmodePtrGbl->div16 = 1; /* divide by 16    */
        spmodePtrGbl->pms   = 2; /* prescale modulus select */
        break;
    case 156250:
        /* run at 390 khz to get same or faster effective rate */
        spmodePtrGbl->div16 = 1; /* divide by 16    */
```

广东省网络空间安全协会受控资料

```
        spmodePtrGbl->pms    = 0; /* prescale modulus select */
        break;
    default:
        /* 156250 */
        /* run at 390 khz to get same or faster eff rate */
        spmodePtrGbl->div16 = 1; /* divide by 16    */
        spmodePtrGbl->pms    = 0; /* prescale modulus select */
        break;
}

/* reinitialize receive frame buffers */
spmGbl.rf.setTxFlag      = 0; /* frame status not transmitting */
spmGbl.rf.clrTxReqFlag = 0; /* don't clear */
spmGbl.rf.rxDataValid   = 0; /* no valid data this frame */
spmGbl.rf.txDataCTS      = 0; /* no clear to send */
spmGbl.rf.setCollDet    = 0; /* no collision detected */
spmGbl.rf.rxFlag         = 0; /* not receiving */
spmGbl.rf.rwAck          = 0; /* not acknowledged */
spmGbl.rf.txOn           = 0; /* not receiving */
spmGbl.rf.data           = 0; /*    */

/* reinitialize receive frame buffers */
spmGbl.tf.txFlag         = 0; /* frame status not transmitting */
spmGbl.tf.txReqFlag      = 0; /* don't req */
spmGbl.tf.txDataValid = 0; /* no valid data this frame */
spmGbl.tf.blank          = 0; /* unused */
spmGbl.tf.txAddrRW       = 0; /* write */
spmGbl.tf.txAddr         = 0; /* default address */
spmGbl.tf.data           = 0; /*    */

/* initialize crcGbl   for good form */
crcGbl.poly = 0;
crcGbl.crc = 0;
crcGbl.crcBit = 0;
crcGbl.dataBit = 0;
crcGbl.dataByte = 0;


/* Send First Frame   ISR keeps it going */
tbdPtrGbl->r       = 1; /* buffer ready to transmit */
spcomPtrGbl->str = 1;

/* wait for a while for everything to settle down */
```

广东省网络空间安全协会受控资料

```
    good = DelayTicks(RESTART_DELAY_TICKS);


    PHYHardResetSPMXCVR();   /* this also writes config registers */

    return;

}

/*****************************************************************
Function:   PHYIOInit
Returns:    None
Reference:  None
Purpose:    To Init variables related to PHYIO function.
Comments:   None
*****************************************************************/
void PHYIOInit(void)
{
    SetMsTimer(&phyIOTimer,(uint16)(PHYIO_CHECK_INTERVAL * 1000));
    gp->resetPinPrevState       = 1; /* Allow Reset Button Push */
    gp->manualServiceRequestPrevState = 1; /* Allow Manual Service Request Push */
    gp->ioInputPin0PrevState = 1;
}

/*****************************************************************
Function:   PHYIO
Returns:    None
Reference:  None
Purpose:    To handle Input switches (reset service and IO) and
            to set value of ouput leds
            Manual Service Request and Reset pin share same LED to indicate
            switch pushed
Comments:   None
*****************************************************************/
void PHYIO(void)
{

    /* Perform Pin related checks only on timer expiry
       to avoid bounce problem */
    if (MsTimerExpired(&phyIOTimer))
    {
        /* check reset switch */
        /* reset pulled low &&
```

广东省网络空间安全协会受控资料

```
   reset not enabled so need to enable &&
   have not yet set this pin push */
if (( pbdatPtrGbl->d12 == 0) &&
     ! gp->resetNode &&
     (gp->resetPinPrevState ! = 0))
{
   gp->resetNode = TRUE; /* signal scheduler there is reset */
   nmp->resetCause = EXTERNAL_RESET;
   /* NodeReset function will clear when reset is finished */
   /* prevents sending more than 1 reset until pin is toggled */
   gp->resetPinPrevState = 0;
}

/* reset pin high */
if (pbdatPtrGbl->d12 == 1)
{
   gp->resetPinPrevState = 1;
}

if ((pbdatPtrGbl->d13 == 0) && /* manual service request pulled low */
     ! gp->manualServiceRequest &&          /* manualServiceRequest not enabled */
     (gp->manualServiceRequestPrevState ! = 0)) /* have not yet set
                                                   this pin push */
{
   gp->manualServiceRequestPrevState = 0;
   gp->manualServiceRequest = TRUE; /* notify stack that
                                       manual service request enabled */
}

if (pbdatPtrGbl->d13 == 1) /* manual service request high */
{
   gp->manualServiceRequestPrevState = 1; /* manual service request toggled so
                                             next press will allow push */
}

if(pbdatPtrGbl->d14 == 1 &&
   (gp->ioInputPin0PrevState ! = 0))   /* debouncer,check
                                          gen purpose io input */
{
   gp->ioInputPin0 = 1;
}
else
{
```

广东省网络空间安全协会受控资料

```
        gp->ioInputPin0 = 0;
    }

    if (pbdatPtrGbl->d14 == 1) /* io pin high */
    {
        gp->ioInputPin0PrevState = 1; /* io pin toggled so next
                                         press will allow push */
    }

    SetMsTimer(&phyIOTimer,
            (uint16)(PHYIO_CHECK_INTERVAL * 1000));
  }


  if (gp->resetNode || gp->manualServiceRequest || gp->ioOutputPin0)
  {
    pbdatPtrGbl->d8 = 1; /* turn on reset-service light */
  }
  else
  {
    pbdatPtrGbl->d8 = 0; /* turn off reset-service light */
  }


  if(gp->ioOutputPin1 == 0) /* set io output pin */
  {
    pbdatPtrGbl->d9 = 0;
  }
  else
  {
    pbdatPtrGbl->d9 = 1;
  }

  return;
}


/*****************************************************************
Function:  IncrementBacklog
Returns:
Reference:
Purpose:  increments backlog and handles backlog overflow
Comments:
```

广东省网络空间安全协会受控资料

```
*****************************************/

static void    IncrementBacklog(uint8 deltaBacklog)
{
   if (deltaBacklog > MAX_BACKLOG)
   {
       return; /* Erroneous deltaBacklog value */
   }
   /* spmGbl.backlog size must allow 2 * MAX_BACKLOG without
       wrap around */
   spmGbl.backlog += deltaBacklog;
   if (spmGbl.backlog > MAX_BACKLOG)
   {
       spmGbl.backlog = MAX_BACKLOG;
       INCR_STATS(nmp->stats.backlogOverflow);
   }
}


/*****************************************
Function:  DecrementBacklog
Returns:
Reference:
Purpose:   decrements backlog and handles backlog underflow
Comments:
*****************************************/

static void    DecrementBacklog(uint8 deltaBacklog)
{
   if (deltaBacklog > MAX_BACKLOG)
   {
       return; /* Erroneous deltaBacklog value */
   }
   if (spmGbl.backlog <= deltaBacklog)
   {
       spmGbl.backlog = 0;
   }
   else
   {
       spmGbl.backlog -= deltaBacklog;
   }
}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   SPMIsr
Returns:    none,no arguments allowed either
Reference:
Purpose:
    Interrupt Service Routine

    This ISR serves two primary functions. the first is to execute the
    channel access algorithm. the second is to performs the special
    purpose mode transfers with the XCVR. For the channel access
    algorithm a single hardware timer is used as reference for a set
    of global values that are the expire times for the various
    channel access algorithm timers. The ISR polls the hardware timer
    each time the ISR runs and checks for expiration of the various
    timers.   The ISR has two coupled state machines. the first
    indicated by spmGbl. phase controls the channel access algorithm
    and the second indicated by spmGbl. state controls the SPM
    transfers.

    The ISR runs after each 16 SPI transfers or clock transitions.
    An Interrupt is generated by an rxb event when a receive buffer
    is full.

Comments:   must use interrupt pragma
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*   interrupt pragma informs compiler to make next function
    an interrupt handler with RTE and saved state of registers */

#pragma interrupt()

static void SPMIsr()
{
   SPIEventMaskReg spieTemp;   /* local copy of event register */
   int j;   /* for loop index */

   spieTemp = *spiePtrGbl;   /* read SPIE into local copy */

   /* clear spie ASAP so as not to miss any events while in isr */
   /* write all ones to clear */
   *((uint8 * ) spiePtrGbl) = (uint8) 0xFFU;

   /* each event handled by this isr needs a separate if statement */
   if ( spieTemp. rxb == 1)
```

广东省网络空间安全协会受控资料

```
{
    /* handle interrupt for receive buffer filled */
    /* transfer can be stopped by physical layer or for error */
    if (spmGbl.mode == STOP) /* stop spm exchanges  */
    {
        spmGbl.state = DEBUG;
        macGbl.tc = 0;
        macGbl.tl = 0;
        macGbl.tpr = FALSE;
        macGbl.rc = 0;
        macGbl.rl = 0;
        macGbl.rpr = FALSE;
        spmGbl.crw = FALSE;
        spmGbl.srr = FALSE;
    }
    else    /* State Machines for Transmit and Receive */
    {

        /* debug */
        /* last received frame is in spmGbl.rf  */
        /* update history for debugging purposes */
#ifdef SPM_HISTORY
        hBufGbl.records[hBufGbl.index].rf = spmGbl.rf;
        hBufGbl.records[hBufGbl.index].state = spmGbl.state;
        hBufGbl.records[hBufGbl.index].rb = spiParamPtrGbl->rbptr;
        hBufGbl.records[hBufGbl.index].tb = spiParamPtrGbl->tbptr;

        hBufGbl.records[hBufGbl.index].start = *(spmGbl.clock);
#endif
        /*********************************************************/
        /* state machine for channel access algorithm */
        /*********************************************************/
        switch (spmGbl.phase)
        {
            case BUSY: /* some node is transmitting */
                /* waits here until channel is idle */

                if ((spmGbl.rf.rxFlag == 0) &&
                    (spmGbl.rf.txOn == 0))
                {
                    /* channel has become idle neither receive or transmit */
                    /* start idle timer */
                    spmGbl.idleTimerStart = *(spmGbl.clock);
```

广东省网络空间安全协会受控资料

```
            /* set beta1 duration based on last packet,
               either rx or tx */
            if (spmGbl.kind == POST_RX)
            {
               spmGbl.beta1Ticks = spmGbl.beta1PostRxTicks;
               spmGbl.priorityIdleTicks = spmGbl.priorityChPostRxTicks;
            }
            else
            {
               spmGbl.beta1Ticks = spmGbl.beta1PostTxTicks;
               spmGbl.priorityIdleTicks = spmGbl.priorityChPostTxTicks;
            }

            /* Previous Packet has ended so reset altPathBit flags.
               Once tpr goes true then we will know to write
               altPathBit once before tx */
            spmGbl.writeAltPathBit = FALSE;
            spmGbl.altPathBitWritten = FALSE;

            /* start waiting out beta1 time */
            spmGbl.phase = BETA1_IDLE;
         }
         else /* channel still busy */
         {
            if (spmGbl.rf.rxFlag == 1)
            {
               /* busy receiving packet */
               spmGbl.kind = POST_RX;
            }
            else
            {
               /* busy transmitting packet */
               spmGbl.kind = POST_TX;
            }


            spmGbl.phase = BUSY;
         }
         break;
      case BETA1_IDLE:   /* waits out beta1 time */

         if ((spmGbl.rf.rxFlag == 1) ||
```

广东省网络空间安全协会受控资料

```
    (spmGbl. rf. txOn == 1))
{
    /* channel is now busy */
    /* other node must have used channel or
       xcvr is in error with bad txOn */

    spmGbl. phase = BUSY;
}
else if ((macGbl. tpr == TRUE) &&
            (spmGbl. writeAltPathBit == FALSE) &&
            (spmGbl. state == IDLE)) /* in case doing status register read */
{
    /* tx packet is ready */
    /* always do unconfirmed write
       of altPathBit before tx */
    spmGbl. writeAltPathBit = TRUE;

    /* wait until next frame before begin channel
       access */
    spmGbl. phase = BETA1_IDLE;
}
else
{
    /* check idle timer to see if beta1 time expired */
    spmGbl. elapsed =
        *(spmGbl. clock) - spmGbl. idleTimerStart;

    if ( spmGbl. elapsed >= spmGbl. beta1Ticks )
    {
        /* idle timer expired */
        /* check if transmit packet is still ready and
           set priority access type */
        /* conditions for priority access
           Packet marked as priority
           Collisions less than 2
           Node has a priority slot
           Only after  received packet
           only after valid crc on last received pkt
        */
        if ( (macGbl. tpr == TRUE) &&
                (macGbl. priorityPkt == TRUE) &&
                (spmGbl. kind == POST_RX) && /* only after  rx */
                (spmGbl. collisionsThisPkt <= 1) &&
```

广东省网络空间安全协会受控资料

```
                (spmGbl. nodePriority > 0) &&
                (spmGbl. state == IDLE))      /* in case doing status      */
                                                /* register read          */
            {

                /* attempt priority slot access */
                spmGbl. transmitTimerStart = spmGbl. idleTimerStart +
                                                spmGbl. beta1Ticks;


                spmGbl. phase = PRIORITY_WAIT_TX;
            }
            else
            {

                /* do not attempt priority slot access */
                /* start waiting out priority slots */
                spmGbl. phase = PRIORITY_IDLE;

            }
        }
        else
        {
            spmGbl. phase = BETA1_IDLE; /* keep waiting */
        }
    }
    break;
case PRIORITY_IDLE:
    if ((spmGbl. rf. rxFlag == 1) ||
        (spmGbl. rf. txOn == 1)) /* busy */
    {
        /* other node must have used channel or
           xcvr is in error with bad txOn */

        spmGbl. phase = BUSY;
    }
    else if ((macGbl. tpr == TRUE) &&
                (spmGbl. writeAltPathBit == FALSE) &&
                (spmGbl. state == IDLE)) /* in case doing status register read */
    {
        /* tx packet is ready */
        /* always do unconfirmed write of altPathBit
           before tx */
```

广东省网络空间安全协会受控资料

```
    spmGbl. writeAltPathBit = TRUE;

    /* have to wait until next frame before
       access channel */
    spmGbl. phase = PRIORITY_IDLE;
}
else
{
    /* check idle timer for priority slots to end */
    spmGbl. elapsed =
        *(spmGbl. clock) - spmGbl. idleTimerStart;
    if ( spmGbl. elapsed >=
          (spmGbl. priorityIdleTicks + spmGbl. beta1Ticks)) /* idle      */
                                                            /* timer expired */
    {
       /* timer expired so check if transmit packet ready */
       if ( (macGbl. tpr == TRUE) &&
           (spmGbl. state == IDLE)) /* in case doing status register read */
       {
           /* start transmit timer */
           spmGbl. transmitTimerStart = spmGbl. idleTimerStart +
                                        spmGbl. beta1Ticks +
                                        spmGbl. priorityIdleTicks;
           /* start base timer */
           /* base timer runs when waiting for randomized tx access */

       /* spmGbl. baseTimerStart = spmGbl. idleTimerStart +
                                        spmGbl. beta1Ticks +
                                        spmGbl. priorityIdleTicks; */

           spmGbl. baseTimerStart = spmGbl. transmitTimerStart; /* faster */
                                                                /* way */

               /* change seed */
           srand((unsigned int) spmGbl. transmitTimerStart);

             /* attempt random slot access */
           /* random slot between 0 and backlog * wbase - 1 */

           spmGbl. randomTicks = (rand() % ((spmGbl. backlog + 1) *
                                        W_BASE)) * spmGbl. beta2Ticks ;

           spmGbl. phase = RANDOM_WAIT_TX; /* go wait for slot */
```

广东省网络空间安全协会受控资料

```
            }
            else   /* priority idle timer has expired but transmit        */
                        /* packet is not ready */
            {
                spmGbl. phase = RANDOM_IDLE; /* go wait */
            }
        }
        else /* timer not elapsed so keep waiting for end of priority slots */
        {
            spmGbl. phase = PRIORITY_IDLE;
        }
    }
    break;
case RANDOM_IDLE:
    if ((spmGbl. rf. rxFlag == 1) || (spmGbl. rf. txOn == 1)) /* busy */
    {
        /* other node must have used channel or
           xcvr is in error with bad txOn */
        spmGbl. phase = BUSY;
    }
    else if ((macGbl. tpr == TRUE) && /* tx packet ready */
             (spmGbl. writeAltPathBit == FALSE) &&
             (spmGbl. state == IDLE)) /* in case doing status register read */
    {
        /* always do unconfirmed write of altPathBit before tx */
        spmGbl. writeAltPathBit = TRUE;

        /* have to wait until next frame to start channel access */
        spmGbl. phase = RANDOM_IDLE;
    }
    else
    {
        /* check if transmit packet still ready   */
        if ( (macGbl. tpr == TRUE)   &&
            (spmGbl. state == IDLE) ) /* in case doing status register read */
        {
            spmGbl. transmitTimerStart = *(spmGbl. clock); /* start transmit   */
                                                                /* timer */

            /* base timer runs when waiting for randomized tx access */
                /* start base timer */
            spmGbl. baseTimerStart = spmGbl. transmitTimerStart;
```

广东省网络空间安全协会受控资料

```
            srand((unsigned int) spmGbl. transmitTimerStart); /* change seed */

            /* attempt random slot access */
            /* random slot between 0 and backlog * wbase - 1 */
            spmGbl. randomTicks = (rand() % ((spmGbl. backlog + 1) * W_BASE))
                * spmGbl. beta2Ticks ;

            spmGbl. phase = RANDOM_WAIT_TX; /* go wait for slot */
        }
        else
        {
            spmGbl. phase = RANDOM_IDLE; /* nothing to send keeping waiting */
        }
    }
    break;
case PRIORITY_WAIT_TX:
    if ((spmGbl. rf. rxFlag == 1) || (spmGbl. rf. txOn == 1)) /* busy */
    {
        /* other node must have used channel or
            xcvr is in error with bad txOn */
        spmGbl. phase = BUSY;
    }
    else
    {
        /* check tx timer wait for our priority slot to come up */
        spmGbl. elapsed = *(spmGbl. clock) - spmGbl. transmitTimerStart;
        if (spmGbl. elapsed >= spmGbl. priorityNodeTicks)
        {
            /* approve transmission */
            spmGbl. accessApproved = TRUE; /* enable spm state machine to */
                                            /* begin tx                    */
            spmGbl. phase = START_TX;
        }
        else
        {
            spmGbl. phase = PRIORITY_WAIT_TX;
        }
    }
    break;
case RANDOM_WAIT_TX:
    if ((spmGbl. rf. rxFlag == 1) || (spmGbl. rf. txOn == 1)) /* busy */
    {
        /* other node must have used channel or
```

广东省网络空间安全协会受控资料

```
            xcvr is in error with bad txOn * /
        spmGbl. phase = BUSY;
    }
    else
    {

        / * Get current time. Since used for two checks we want      * /
        / * to be synchronized                                       * /
        spmGbl. stopped = * (spmGbl. clock);

        / * check tx timer wait for out random slot to come up * /
        spmGbl. elapsed = spmGbl. stopped - spmGbl. transmitTimerStart;
        if (spmGbl. elapsed >= spmGbl. randomTicks)
        {
            / * approve transmission * /
            spmGbl. accessApproved = TRUE; / * enable spm state machine to
                                                                begin tx * /
            spmGbl. phase = START_TX;
        }
        else
        {
            spmGbl. phase = RANDOM_WAIT_TX;
        }


        / * check base timer, base timer running when we do random      * /
            / * slot access   * /
        / * remember baseTimerStart is the same as transmitTimerStart only on
            the first iteration * /
        spmGbl. elapsed = spmGbl. stopped - spmGbl. baseTimerStart;
        if (spmGbl. elapsed >= spmGbl. baseTicks)
        {
            spmGbl. baseTimerStart = spmGbl. stopped; / * restart base timer * /
            DecrementBacklog(1);   / * decrement back log * /
        }

    }
    break;
case START_TX:
    if ((spmGbl. rf. rxFlag == 1) || (spmGbl. rf. txOn == 1)) / * channel busy * /
    {
        / * other node must have used channel or packet has started * /
```

广东省网络空间安全协会受控资料

```
            spmGbl. accessApproved = FALSE;
            spmGbl. phase = BUSY;
        }
        else
        {
            if ( macGbl. tpr == FALSE) / * macGbl. tpr == FALSE packet cancelled * /
            {
                spmGbl. accessApproved = FALSE; / * deny access * /
                spmGbl. phase = RANDOM_IDLE;      / * back to idle * /
            }
            else
            {

                spmGbl. phase = START_TX;
            }

            / * check base timer on more time in case expires while waiting
                for txOn. only if non priority slot access * /

            spmGbl. stopped = * (spmGbl. clock);
            spmGbl. elapsed = spmGbl. stopped - spmGbl. baseTimerStart;
            if (spmGbl. elapsed >= spmGbl. baseTicks)
            {
                spmGbl. baseTimerStart = spmGbl. stopped; / * restart base timer * /
                DecrementBacklog(1);   / * decrement back log * /
            }

        }
        break;
    default:
        spmGbl. phase = RANDOM_IDLE;
        break;
}   / * End access algo switch * /


/ * Cycle Timer only runs when the Mac Layer is Idle that is when
    the node is not tranmitting or receiving or waiting to transmit
    or counting down beta1 or counting down priority slots or counting
    down the extra 16 beta2 slots after a transmission
    This directly corresponds to the Phase RANDOM_IDLE

    Anytime the MAC layer is busy ie in some phase other than RANDOM_IDLE
    then the cycle timer stops.  The cycle timer will then resume when the
```

广东省网络空间安全协会受控资料

```
    mac layer returns to the RANDOM_IDLE phase.  The cycle timer is reset when
    spmGbl. cycleTimerRestart == TRUE.  This is set to true
    1) following a successful transmission
    2) following a valid crc receive

    We use a time difference between the current value of the hardware timer
    and the stored value when we started the cycleTimer to determine
    when the cycleTimer expires.
    Since the hardware timer is free running and does not stop when
    the cycle timer stops ,we must keep shifting the
    cycleTimerStart forward whenever the cycleTimer is stopped.  This keeps
    the relative time difference correct so that when it resumes
    the expiration calculation will be valid.
*/

spmGbl. stopped = *(spmGbl. clock);
if (  (spmGbl. phase == RANDOM_IDLE) ) /* cycle timer is running */
{
    spmGbl. elapsed = spmGbl. stopped - spmGbl. cycleTimerStart;

    if (spmGbl. cycleTimerRestart) /* cycle timer needs to be restarted */
    {
        spmGbl. cycleTimerStart = spmGbl. stopped;
        spmGbl. cycleTimerRestart = FALSE;
    }
    else if (spmGbl. elapsed >= spmGbl. cycleTicks)
    {
        spmGbl. cycleTimerStart = spmGbl. stopped; /* restart */
        DecrementBacklog(1);
    }
}
else  /* shift cycleTimerStart forward since cycle timer stopped */
{
    spmGbl. cycleTimerStart += spmGbl. stopped - spmGbl. lastTime;
}
spmGbl. lastTime = spmGbl. stopped; /* save last stopped value */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  End Channel Access Algorithm State machine
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  Begin SPM Transfer State Machine
```

广东省网络空间安全协会受控资料

```
 * * *
```

```
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 0;
        spmGbl. tf. data = 0;
        */

      *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;  /* fast way */

      /* increment reset count */
      spmGbl. resetCount ++;
      if (spmGbl. resetCount >= RESET_COUNT_LIMIT)
      {
         PHYHardResetSPMXCVR();
      }

      spmGbl. state = IDLE;
    }
    else if (( spmGbl. writeAltPathBit) && (! spmGbl. altPathBitWritten))
    {
      /* need to write alt path bit unacked write */
      /*
        spmGbl. tf. txFlag = 0;
        spmGbl. tf. txReqFlag = 0;
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 1;
        spmGbl. tf. data = spmGbl. configData[0];
        */

      *((uint8 *)&(spmGbl. tf)) = (uint8) 1;  /* fast way */
      if (macGbl. altPathBit == 1)
      {
         *((uint8 *)&(spmGbl. tf. data)) =
            (uint8) ( 0x80 | spmGbl. configData[0]);  /* fast way */
      }
      else
      {
         *((uint8 *)&(spmGbl. tf. data)) = (uint8) ( 0x7F &
                                                spmGbl. configData[0]);
      }
      spmGbl. altPathBitWritten = TRUE;
```

广东省网络空间安全协会受控资料

```
    spmGbl. state = IDLE;
}
else if ((macGbl. tpr == TRUE) && /* transmit packet ready */
          (spmGbl. accessApproved == TRUE)) /* channel access approved */
{
    /*
      spmGbl. tf. txFlag = 0;
      spmGbl. tf. txReqFlag = 1;
      spmGbl. tf. txDataValid = 0;
      spmGbl. tf. blank = 0;
      spmGbl. tf. txAddrRW = 0;
      spmGbl. tf. txAddr = 0;
      spmGbl. tf. data = 0;
      */

    *((uint16 *)&(spmGbl. tf)) = (uint16) 0x4000U;  /* fast way */

    macGbl. tc = 0;
    spmGbl. state = REQ_TX;
}

else if ((spmGbl. crw == TRUE) &&  /* config register write */
          (spmGbl. cra > 0) &&          /* 0< config reg address <=7 */
          (spmGbl. cra <= 7) )
{
    /*
      spmGbl. tf. txFlag = 0;
      spmGbl. tf. txReqFlag = 0;
      spmGbl. tf. txDataValid = 0;
      spmGbl. tf. blank = 0;
      spmGbl. tf. txAddrRW = 0;
      spmGbl. tf. txAddr = spmGbl. cra;
      spmGbl. tf. data = spmGbl. crData;
      */

    *((uint8 *)&(spmGbl. tf)) = (uint8) spmGbl. cra;  /* fast way */
    *((uint8 *)&(spmGbl. tf. data)) = (uint8) spmGbl. crData; /* fast way */

    spmGbl. state = WRITE;
}
else if ( (spmGbl. srr == TRUE) &&   /* status register read */
          (spmGbl. sra >= 0) &&        /* 0< status reg address <=7 */
          (spmGbl. sra <= 7) )
```

广东省网络空间安全协会受控资料

```
        {
            spmGbl. tf. txFlag = 0;
            spmGbl. tf. txReqFlag = 0;
            spmGbl. tf. txDataValid = 0;
            spmGbl. tf. blank = 0;
            spmGbl. tf. txAddrRW = 1;
            spmGbl. tf. txAddr = spmGbl. sra;
            spmGbl. tf. data = 0;

            spmGbl. state = READ;
        }
        else    /* Nothing can be done just wait */
        {
            /*
              spmGbl. tf. txFlag = 0;
              spmGbl. tf. txReqFlag = 0;
              spmGbl. tf. txDataValid = 0;
              spmGbl. tf. blank = 0;
              spmGbl. tf. txAddrRW = 0;
              spmGbl. tf. txAddr = 0;
              spmGbl. tf. data = 0;
              */

            *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */
            spmGbl. state = IDLE;
        }
        break;
    case RECEIVE:
        /*
          spmGbl. tf. txFlag = 0;
          spmGbl. tf. txReqFlag = 0;
          spmGbl. tf. txDataValid = 0;
          spmGbl. tf. blank = 0;
          spmGbl. tf. txAddrRW = 0;
          spmGbl. tf. txAddr = 0;
          spmGbl. tf. data = 0;
          */

        *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */

        if (spmGbl. rf. rxFlag == 0)   /* packed ended */
        {
            macGbl. rl = macGbl. rc; /* set last byte count */
```

广东省网络空间安全协会受控资料

```
/* is packet of valid length */
if ( (macGbl.rl >= 8) && /* minimum valid packet is 8 bytes */
     (macGbl.rl < PKT_BUF_LEN) ) /* must be less than buffer */
{
   /* do last step of crc */
   crcGbl.crc = crcGbl.crc ^ 0xffff;


   /* is crc valid */
   if ( crcGbl.crc == ( *((uint16 *) &macGbl.rPkt[macGbl.rl - 2])) )
   {
      /* copy out backlog */
      macGbl.deltaBLRx = macGbl.rPkt[0] & 0x3F;

      /* update cycle timer and set flag */
      if (macGbl.deltaBLRx <= 0)
      {
         DecrementBacklog(1);
      }
      else
      {
         IncrementBacklog(macGbl.deltaBLRx);
      }
      /* Cycle Timer needs to be restarted after valid crc rx */
      spmGbl.cycleTimerRestart = TRUE;

      /* Attempt to copy packet into Link Layer receive Queue */
      if ( ( *gp->lkInQTailPtr == 0) && /* == 0 means free     */
                                        /* buffer big enough for packet    */
           (gp->lkInBufSize >= macGbl.rl + 3) )
      {
        /* Copy the packet to link layer input queue */
         *(uint16 *)(gp->lkInQTailPtr + 1) = (uint16) macGbl.rl;
                                                  /* copy length */
         memcpy(gp->lkInQTailPtr + 3,macGbl.rPkt,macGbl.rl);
         /* Update Link Layer Input Queue Fields */
         *gp->lkInQTailPtr = 1; /* lets link layer know packet      */
                                          /* is ready to process       */
         gp->lkInQTailPtr += gp->lkInBufSize;
         if (gp->lkInQTailPtr ==
             gp->lkInQ + gp->lkInBufSize * gp->lkInQCnt)
               {/* past end of queue */
            gp->lkInQTailPtr = gp->lkInQ; /* wrap to beginning */
```

广东省网络空间安全协会受控资料

```
                }
            }
            else /* packet discarded */
            {
                /* Discard packet as either there is no buffer available or
                    it is too big for buffer */
                INCR_STATS(nmp->stats.missedMessages); /* update stats */
            }
        }
        else /* invalid crc */
        {
            /* update stats for bad crc packet */
            INCR_STATS(nmp->stats.transmissionErrors);
        }
    }
    else /* update stats invalid packet length */
    {
        INCR_STATS(nmp->stats.transmissionErrors);
    }

    /* Reset macGbl's receive packet fields */
    macGbl.rc = 0; /* reinit to be safe */
    macGbl.rl = 0;
    macGbl.rpr = FALSE;   /* ready to receive another packet */

    spmGbl.state = IDLE; /*   all done */
}
else if (spmGbl.rf.rxDataValid   == 1)    /* valid byte this frame */
{
    if(macGbl.rc < PKT_BUF_LEN) /* don't go past end of buffer */
    {
        macGbl.rPkt[macGbl.rc] = spmGbl.rf.data;   /* copy out byte */
        macGbl.rc++; /* increment counter */

        /* do 1 byte of crc. Delay by two bytes so we do not computer
            crc on the two crc bytes in the packet itself */
        if (macGbl.rc >= 3)
        {
            /* make copy of byte */
            crcGbl.dataByte = macGbl.rPkt[macGbl.rc - 3];
            for (j = 0; j < 8; j++)
            {
                crcGbl.crcBit = (crcGbl.crc & 0x8000) ? 1 : 0;
```

广东省网络空间安全协会受控资料

```
                crcGbl. dataBit = (crcGbl. dataByte & 0x80) ? 1 : 0;
                crcGbl. dataByte = crcGbl. dataByte << 1;
                crcGbl. crc = crcGbl. crc << 1;
                if ( crcGbl. crcBit ! = crcGbl. dataBit)
                {
                    crcGbl. crc = crcGbl. crc ^ crcGbl. poly;
                }
            }
        }
    }



    spmGbl. state = RECEIVE; /* look for more */
}
else  /* no valid data but still receiving */
{
    spmGbl. state = RECEIVE; /* keep looking */
}
break;
case WRITE:
if ( spmGbl. rf. rxFlag == 1)  /* XCVR has detected packet to receive */
{
    macGbl. rc = 0; /* init receive byte count */
    macGbl. rl = 0;
    if ( macGbl. rpr == TRUE )  /* last packet not copyied out */
    {
      spmGbl. mode =  OVERWRITE; /* stepped on last packet */
      macGbl. rpr = FALSE;/* what else to do ? */
    }

    /* initialize  crc stuff */
    crcGbl. poly = 0x1021;
    crcGbl. crc = 0xffff;
    crcGbl. crcBit = 0;
    crcGbl. dataBit = 0;
    crcGbl. dataByte = 0;

    /*
      spmGbl. tf. txFlag = 0;
      spmGbl. tf. txReqFlag = 0;
      spmGbl. tf. txDataValid = 0;
      spmGbl. tf. blank = 0;
```

广东省网络空间安全协会受控资料

```
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 0;
        spmGbl. tf. data = 0;
        */

      *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */

      spmGbl. state = RECEIVE;
    }
    else if ((spmGbl. rf. rwAck == 0) &&   /* still not written */
             (spmGbl. crw == TRUE) &&
             (spmGbl. cra > 0) &&          /* 0< config reg address <=7 */
             (spmGbl. cra <= 7) )
    {
      /*
        spmGbl. tf. txFlag = 0;
        spmGbl. tf. txReqFlag = 0;
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = spmGbl. cra;
        spmGbl. tf. data = spmGbl. crData;
        */

      *((uint8 *)&(spmGbl. tf)) = (uint8) spmGbl. cra;   /* fast way */
      *((uint8 *)&(spmGbl. tf. data)) = (uint8) spmGbl. crData; /* fast way */

      spmGbl. state = WRITE; /* try again */
    }
    else   /* write successful or recalled */
    {
      spmGbl. crw = FALSE;

      /*
        spmGbl. tf. txFlag = 0;
        spmGbl. tf. txReqFlag = 0;
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 0;
        spmGbl. tf. data = 0;
        */
```

广东省网络空间安全协会受控资料

```
        *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;   /* fast way */


        spmGbl.state = IDLE;
    }
    break;
case READ:
    if ( spmGbl.rf.rxFlag == 1) /* XCVR has detected packet to receive */
    {
        macGbl.rc = 0; /* init receive byte count */
        macGbl.rl = 0;
        if ( macGbl.rpr == TRUE )   /* last packet not copied out */
        {
            spmGbl.mode =   OVERWRITE; /* stepped on last packet */
            macGbl.rpr = FALSE;/* what else to do ? */
        }

        /* initialize   crc stuff */
        crcGbl.poly = 0x1021;
        crcGbl.crc = 0xffff;
        crcGbl.crcBit = 0;
        crcGbl.dataBit = 0;
        crcGbl.dataByte = 0;

        /*
          spmGbl.tf.txFlag = 0;
          spmGbl.tf.txReqFlag = 0;
          spmGbl.tf.txDataValid = 0;
          spmGbl.tf.blank = 0;
          spmGbl.tf.txAddrRW = 0;
          spmGbl.tf.txAddr = 0;
          spmGbl.tf.data = 0;
          */

        *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;   /* fast way */

        spmGbl.state = RECEIVE;
    }
    else if ((spmGbl.rf.rwAck == 0) &&   /* still not read */
             (spmGbl.srr == TRUE) &&
             (spmGbl.sra >= 0) &&          /* 0< config reg address <=7 */
             (spmGbl.sra <= 7) )
    {
```

广东省网络空间安全协会受控资料

```
        spmGbl. tf. txFlag = 0;
        spmGbl. tf. txReqFlag = 0;
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 1;
        spmGbl. tf. txAddr = spmGbl. sra;
        spmGbl. tf. data = 0;

        spmGbl. state = READ;   /* keep trying */
    }
    else if ( spmGbl. rf. rwAck == 1)
    {
        spmGbl. srData = spmGbl. rf. data;
        spmGbl. srr = FALSE;

        /*
          spmGbl. tf. txFlag = 0;
          spmGbl. tf. txReqFlag = 0;
          spmGbl. tf. txDataValid = 0;
          spmGbl. tf. blank = 0;
          spmGbl. tf. txAddrRW = 0;
          spmGbl. tf. txAddr = 0;
          spmGbl. tf. data = 0;
          */

        *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */
        spmGbl. state = IDLE;
    }
    else
    {
        /*
          spmGbl. tf. txFlag = 0;
          spmGbl. tf. txReqFlag = 0;
          spmGbl. tf. txDataValid = 0;
          spmGbl. tf. blank = 0;
          spmGbl. tf. txAddrRW = 0;
          spmGbl. tf. txAddr = 0;
          spmGbl. tf. data = 0;
          */

        *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */
        spmGbl. state = IDLE;
```

广东省网络空间安全协会受控资料

```
    }
    break;
case REQ_TX:
    if ( spmGbl. rf. rxFlag == 1)        /* XCVR has detected packet to receive */
    {
        macGbl. rc = 0; /* init receive byte count */
        macGbl. rl = 0;
        if ( macGbl. rpr == TRUE )   /* last packet not copied out */
        {
            spmGbl. mode =   OVERWRITE; /* stepped on last packet */
            macGbl. rpr = FALSE;/* what else to do ? */
        }

        /* initialize   crc stuff */
        crcGbl. poly = 0x1021;
        crcGbl. crc = 0xffff;
        crcGbl. crcBit = 0;
        crcGbl. dataBit = 0;
        crcGbl. dataByte = 0;

        /*
            spmGbl. tf. txFlag = 0;
            spmGbl. tf. txReqFlag = 0;
            spmGbl. tf. txDataValid = 0;
            spmGbl. tf. blank = 0;
            spmGbl. tf. txAddrRW = 0;
            spmGbl. tf. txAddr = 0;
            spmGbl. tf. data = 0;
            */

        *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */

        spmGbl. state = RECEIVE;
    }
    else if (spmGbl. rf. clrTxReqFlag == 1) /* either accept or deny */
    {
        if ( (spmGbl. rf. setTxFlag == 1) && /* request accepted */
                (spmGbl. rf. txDataCTS == 1) )
        {
            /*
            spmGbl. tf. txFlag = 1;
            spmGbl. tf. txReqFlag = 0;
            spmGbl. tf. txDataValid = 1;
```

广东省网络空间安全协会受控资料

```
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 0;
        spmGbl. tf. data =  macGbl. tPkt[macGbl. tc];
        */

        *((uint8 *)&(spmGbl. tf)) = (uint8) 0xA0;  /* fast way */
        *((uint8 *)&(spmGbl. tf. data)) = (uint8) macGbl. tPkt[macGbl. tc];

        macGbl. tc++; /* increment count */

        spmGbl. state = TRANSMIT; /* accepted */
    }
    else  /* request denied */
    {
        /*
        spmGbl. tf. txFlag = 0;
        spmGbl. tf. txReqFlag = 0;
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 0;
        spmGbl. tf. data = 0;
        */

        *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;  /* fast way */

        spmGbl. state = IDLE; /* dont want to tx anymore */

    }
}
else  /* wait for xcvr to respond to request */
{
    /* typically there is one blank frame from xcvr
       after REQ_TX bit is set but before CTS or ClrTxReqFlag

       Possible bug in Echelon powerline spm xcvr.
       Sending multiple REQ_TX without waiting for
       response may cause xcvr to hang. So only
       send zeros while waiting
    */

    /*
```

广东省网络空间安全协会受控资料

```
            spmGbl. tf. txFlag = 0;
            spmGbl. tf. txReqFlag = 0;
            spmGbl. tf. txDataValid = 0;
            spmGbl. tf. blank = 0;
            spmGbl. tf. txAddrRW = 0;
            spmGbl. tf. txAddr = 0;
            spmGbl. tf. data = 0;
            */

        *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */

        spmGbl. state = REQ_TX; /* Wait for response */
    }
    break;
case TRANSMIT:
    if ( spmGbl. rf. setCollDet == 1)   /* collision so start over */
    {
        /*
            spmGbl. tf. txFlag = 0;
            spmGbl. tf. txReqFlag = 0;
            spmGbl. tf. txDataValid = 0;
            spmGbl. tf. blank = 0;
            spmGbl. tf. txAddrRW = 0;
            spmGbl. tf. txAddr = 0;
            spmGbl. tf. data = 0;
            */

        *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */

        macGbl. tc = 0;

        IncrementBacklog(1);   /* collision so increment backlog */

        spmGbl. collisionsThisPkt++; /* increment collison count this packet */

        INCR_STATS(nmp->stats. collisions);/* increment collision stats */

        if (spmGbl. collisionsThisPkt >= 255)
        {
            /* throw away packet */
            macGbl. tc = 0;
            spmGbl. resetCount = 0; /* reset count for txOn */
            spmGbl. collisionsThisPkt = 0;
```

广东省网络空间安全协会受控资料

```
            macGbl. tpr = FALSE;
        }

        spmGbl. state = IDLE; /* start over */
    }
    else if (spmGbl. rf. txDataCTS == 1)
    {
        /*
            spmGbl. tf. txFlag = 1;
            spmGbl. tf. txReqFlag = 0;
            spmGbl. tf. txDataValid = 1;
            spmGbl. tf. blank = 0;
            spmGbl. tf. txAddrRW = 0;
            spmGbl. tf. txAddr = 0;
            spmGbl. tf. data =   macGbl. tPkt[macGbl. tc];
            */

        *((uint8 *)&(spmGbl. tf)) = (uint8) 0xA0;   /* fast way */
        *((uint8 *)&(spmGbl. tf. data)) = (uint8) macGbl. tPkt[macGbl. tc];
                                                                /* fast way */

            macGbl. tc++; /* increment count */

        if( macGbl. tc >= macGbl. tl) /* last byte sent this frame */
        {
            spmGbl. state = DONE_TX; /* dont want to tx anymore */
        }
        else
        {
            spmGbl. state = TRANSMIT;
        }
    }
    else if ((spmGbl. rf. txOn == 1) &&
            (macGbl. tpr == TRUE) &&
            (macGbl. tc < macGbl. tl) ) /* wait for cts */
    {
        /*
            spmGbl. tf. txFlag = 1;
            spmGbl. tf. txReqFlag = 0;
            spmGbl. tf. txDataValid = 0;
            spmGbl. tf. blank = 0;
            spmGbl. tf. txAddrRW = 0;
            spmGbl. tf. txAddr = 0;
```

广东省网络空间安全协会受控资料

```
        spmGbl. tf. data =  0;
        */

      *((uint8 *)&(spmGbl. tf)) = (uint8) 0x80;  /* fast way */
      *((uint8 *)&(spmGbl. tf. data)) = (uint8) 0;  /* fast way */

      spmGbl. resetCount++;
      if (spmGbl. resetCount >= RESET_COUNT_LIMIT)
      {
         PHYHardResetSPMXCVR();
         spmGbl. state = IDLE;
      }
      else
      {
         spmGbl. state = TRANSMIT;
      }
   }
   else /* something wrong here throw away packet and get out */
   {
      /*
        spmGbl. tf. txFlag = 0;
        spmGbl. tf. txReqFlag = 0;
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 0;
        spmGbl. tf. data = 0;
        */

      *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;  /* fast way */

      macGbl. tc = 0;
      spmGbl. resetCount = 0; /* reset count for txOn */
      spmGbl. collisionsThisPkt = 0;
      macGbl. tpr = FALSE;
      spmGbl. state = IDLE;
   }
   break;
case DONE_TX: /* wait for tx_on to go off */
   if ( spmGbl. rf. setCollDet == 1)  /* collision so start over */
   {
      /*
        spmGbl. tf. txFlag = 0;
```

广东省网络空间安全协会受控资料

```
        spmGbl. tf. txReqFlag = 0;
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 0;
        spmGbl. tf. data = 0;
        */

      *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U;   /* fast way */

      macGbl. tc = 0;

      IncrementBacklog(1);   /* collision so increment backlog */

      spmGbl. collisionsThisPkt++; /* increment collision count      */
                                   /* this packet */

      INCR_STATS(nmp->stats. collisions);/* increment collision stats */

      if (spmGbl. collisionsThisPkt >= 255)
      {
        /* throw away packet */
        macGbl. tc = 0;
        spmGbl. resetCount = 0; /* reset count for txOn */
        spmGbl. collisionsThisPkt = 0;
        macGbl. tpr = FALSE;
      }

      spmGbl. state = IDLE; /* start over */
    }
    else if (spmGbl. rf. txOn == 1) /* still sending last byte(s) */
    {
      /*
        spmGbl. tf. txFlag = 0;
        spmGbl. tf. txReqFlag = 0;
        spmGbl. tf. txDataValid = 0;
        spmGbl. tf. blank = 0;
        spmGbl. tf. txAddrRW = 0;
        spmGbl. tf. txAddr = 0;
        spmGbl. tf. data =  0;
        */

      *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0;   /* fast way */
```

广东省网络空间安全协会受控资料

```
    spmGbl. resetCount++;
    if (spmGbl. resetCount >= RESET_COUNT_LIMIT)
    {
        PHYHardResetSPMXCVR();
        spmGbl. state = IDLE;
    }
    else
    {
        spmGbl. state = DONE_TX;
    }
}
else /* (spmGbl. rf. txOn == 0) done sending */
{
    /*
      spmGbl. tf. txFlag = 0;
      spmGbl. tf. txReqFlag = 0;
      spmGbl. tf. txDataValid = 0;
      spmGbl. tf. blank = 0;
      spmGbl. tf. txAddrRW = 0;
      spmGbl. tf. txAddr = 0;
      spmGbl. tf. data =  0;
      */

    *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0;   /* fast way */

    /* successful transmission so update backlog */
    if (macGbl. deltaBLTx <= 0)
    {
        DecrementBacklog(1);
    }
    else
    {
        IncrementBacklog(macGbl. deltaBLTx);
    }

    /* Cycle Timer needs to be restarted after valid tx   */

    spmGbl. cycleTimerRestart = TRUE;


    /* clean up */
    macGbl. tc = 0;
```

广东省网络空间安全协会受控资料

```
                spmGbl. resetCount = 0; /* reset count for txOn */
                spmGbl. collisionsThisPkt = 0;
                macGbl. tpr = FALSE;
                spmGbl. state = IDLE; /* dont want to tx anymore */
            }
            break;
        case DEBUG:
        default:
            /*
                spmGbl. tf. txFlag = 0;
                spmGbl. tf. txReqFlag = 0;
                spmGbl. tf. txDataValid = 0;
                spmGbl. tf. blank = 0;
                spmGbl. tf. txAddrRW = 0;
                spmGbl. tf. txAddr = 0;
                spmGbl. tf. data = 0;
                */

            *((uint16 *)&(spmGbl. tf)) = (uint16) 0x0U; /* fast way */

            spmGbl. state = DEBUG;
    }       /* end switch spm state machine */


#ifdef SPM_HISTORY
        hBufGbl. records[hBufGbl. index]. tf = spmGbl. tf;
        /* time period of  isr */
        /* C unsigned math is congruent mod 2^n so negatives wrap around OK */
        hBufGbl. records[hBufGbl. index]. duration = *(spmGbl. clock) -
            hBufGbl. records[hBufGbl. index]. start;

        hBufGbl. index = (hBufGbl. index + 1) % NUM_HIST;
#endif
        /* Send Frame */
        tbdPtrGbl->r = 1;           /* buffer ready to transmit */
        spcomPtrGbl->str = 1;
    } /* end if spmGbl. mode == STOP */
  }  /* endif rxb event */

  cisrPtrGbl->spi = 1; /* clear spi bit in CISR by writing a 1 */
  return;
```

广东省网络空间安全协会受控资料

```
}

/*****************************************
Function:  CErrIsr
Returns:   none,no arguments allowed either
Reference:
Purpose:   interrupt service routine for CPM Int. Error
Comments:  must use interrupt pragma
*****************************************/
/*  interrupt pragma informs compiler to make next function
        an interrupt handler with RTE and saved state of registers */

#pragma interrupt()

static void CErrIsr(void)
{

   return;
}

/*****************************************
Function:  SpurIntIsr
Returns:   none,no arguments allowed either
Reference:
Purpose:   interrupt service routine for spurious interrupt Error
Comments:  must use interrupt pragma
*****************************************/
/*  interrupt pragma informs compiler to make next function
        an interrupt handler with RTE and saved state of registers */

#pragma interrupt()

static void SpurIntIsr(void)
{
   exceptions++;

   return;
}

/*****************************************
Function:  BusErrIsr
Returns:   none,no arguments allowed either
Reference:
```

广东省网络空间安全协会受控资料

```
Purpose:    interrupt service routine for Bus errors
Comments:   must use interrupt pragma
*****************************************************************/
/*   interrupt pragma informs compiler to make next function
         an interrupt handler with RTE and saved state of registers */

#pragma interrupt()

static void BusErrIsr(void)
{
   exceptions++;

   return;
}

/****************************************************************
Function:   GetTransceiverStatus
Returns:    none
Purpose:    To determine the status of transceiver registers.
            The status registers are different from config registers.
Comments:   The array should have space for all the register values.
*****************************************************************/
void GetTransceiverStatus(Byte transceiverStatusOut[])
{
   int i;
   unsigned long count;
   Boolean good;

   /* read config registers on PLT-20 transceiver */

   /* read in reverse order */
   i = NUM_COMM_PARAMS - 1; /* should be 6 */
   while ( i >= 0)
   {
      /* read status registers starting at 7 down to 1 */
      spmGbl.sra = (Byte) i + 1;
      spmGbl.srr = TRUE;
      count = 0;
      /* wait for ack but time out so not infinite loop */
      while ((spmGbl.srr != FALSE) && (count < 0xFFFFF))
      {
         count++;
      }
```

广东省网络空间安全协会受控资料

```
        if (spmGbl. srr == TRUE) /* xcvr never acked try again */
        {
            pbdatPtrGbl->d5    = 0; /* first reset xcvr */

            /* wait for a while for everything to settle down */
            good = DelayTicks(RESTART_DELAY_TICKS);


            pbdatPtrGbl->d5    = 1; /* bring xcvr out of reset */
            i = NUM_COMM_PARAMS - 1; /* start over again */
        } /* NOTE!!! if the xcvr never acks this code will hang */
        else
        {
            transceiverStatusOut[i] = spmGbl. srData;
            i--;
        }
    }

    spmGbl. srData = 0;
    spmGbl. sra = 0;


    return;

}

/*************** End of spm. c ***************/
```

广东省网络空间安全协会受控资料

A.2 LPDU 发送算法

```
/*****************************************************
Function:   LKSend
Returns:    None
Reference:  None
Purpose:    To take the NPDU from link layer's output queue and put it
            in the queue for the physical layer.
Comments:   We assume that there will be sufficient space as we
            allocated the extra bytes based on header size etc.
*****************************************************/
void LKSend(void)
{
    LKSendParam        * lkSendParamPtr;
    Queue              * lkSendQueuePtr;
    Byte               * npduPtr;
```

```
Byte            * phyQItemPtr, * phyQPtr, * tempPtr;
Byte            * * phyQTailPtrPtr;
uint16          queueCnt,itemSize;
uint16          * pduSizePtr;
LPDUHeader      * lpduHeaderPtr;
Boolean         priority;

/* First,make variables point to the right queue. */
if (! QueueEmpty(&gp->lkOutPriQ))
{
    priority        = TRUE;
    lkSendQueuePtr  = &gp->lkOutPriQ;
    phyQPtr         = gp->phyOutPriQ;
    phyQItemPtr     = gp->phyOutPriQTailPtr;
    phyQTailPtrPtr  = &gp->phyOutPriQTailPtr;
    queueCnt        = gp->phyOutPriQCnt;
    itemSize        = gp->phyOutPriBufSize;
}
else if (! QueueEmpty(&gp->lkOutQ))
{
    priority        = FALSE;
    lkSendQueuePtr  = &gp->lkOutQ;
    phyQPtr         = gp->phyOutQ;
    phyQItemPtr     = gp->phyOutQTailPtr;
    phyQTailPtrPtr  = &gp->phyOutQTailPtr;
    queueCnt        = gp->phyOutQCnt;
    itemSize        = gp->phyOutBufSize;
}
else
{
    return; /* Nothing to send. */
}

lkSendParamPtr = QueueHead(lkSendQueuePtr);
npduPtr        = (Byte *) (lkSendParamPtr + 1);

/* Check if there is space in the physical layer's queue. */
if ( * phyQItemPtr == 1)
{
    return; /* No space in the physical layer queue. */
}

/* Fill the PDU size so that the physical layer knows about it. */
```

广东省网络空间安全协会受控资料

```
    pduSizePtr   = (uint16 * ) ((char * )phyQItemPtr + 1);
    * pduSizePtr = lkSendParamPtr ->pduSize + 3;

    /* First fill the data. */
    tempPtr = phyQItemPtr + 3;

    /* Form the header. */
    lpduHeaderPtr              = (LPDUHeader * )tempPtr;
    lpduHeaderPtr ->priority = priority;
    lpduHeaderPtr ->altPath    = lkSendParamPtr ->altPath;
    lpduHeaderPtr ->deltaBL    = lkSendParamPtr ->deltaBL;
    tempPtr++;

    /* Copy the NPDU. */
    if (lkSendParamPtr ->pduSize <= itemSize)
    {
        memcpy(tempPtr,npduPtr,lkSendParamPtr ->pduSize);
    }
    else
    {
        ErrorMsg("LKSend: NPDU seems too large to fit.\n");
    }

    /* Compute the CRC value. */
    CRC16(phyQItemPtr+3,lkSendParamPtr ->pduSize + 1);

    /* Turn the flag on so that physical layer can send it. */
    * phyQItemPtr = 1;

    /* Increment tail pointer taking care of wraparound. */
    * phyQTailPtrPtr =  * phyQTailPtrPtr + itemSize;
    if ( * phyQTailPtrPtr == (phyQPtr + queueCnt * itemSize))
    {
        * phyQTailPtrPtr = phyQPtr; /* wrap around. */
    }

    DeQueue(lkSendQueuePtr);

    return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:  CRC16
```

广东省网络空间安全协会受控资料

```
Returns:    16 bit CRC computed.
Purpose:    To compute the 16 bit CRC for a given buffer.
Comments:    None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void CRC16(Byte bufInOut[],uint16 sizeIn)
{
   uint16 poly = 0x1021;          /* Generator Polynomial. */
   uint16 crc = 0xffff;
   uint16 i,j;
   unsigned char byte,crcbit,databit;

   for (i = 0; i < sizeIn; i++)
   {
      byte = bufInOut[i];
      for (j = 0; j < 8; j++)
      {
        crcbit = crc & 0x8000 ? 1 : 0;
        databit = byte & 0x80 ? 1 : 0;
        crc = crc << 1;
        if (crcbit != databit)
         {
           crc = crc ^ poly;
         }
        byte = byte << 1;
      }

   }
   crc = crc ^ 0xffff;
   bufInOut[sizeIn]      = (crc >> 8);
   bufInOut[sizeIn + 1] = (crc & 0x00FF);

   return;
}
```

广东省网络空间安全协会受控资料

A.3 LPDU 接收算法

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   LKReceive

Returns:    None

Reference: None
```

Purpose: To receive the incoming LPDUs and process them.

Comments: Each item of the queue gp->lkInQ has the following form:

flag pduSize LPDU

flag is 1 byte long.

pduSize is 2 bytes long.

LPDU has header followed by the rest of the LPDU and then CRC.

The LPDU header is 1 byte long. CRC uses 2 bytes.

If a packet is in lkInQ then it should fit into nwInQ.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void LKReceive(void)
{
    NWReceiveParam  *nwReceiveParamPtr;
    Byte            *npduPtr;
    LPDUHeader      *lpduHeaderPtr;
    Byte            *tempPtr;
    uint16          lpduSize;

    if (*(gp->lkInQHeadPtr) == 0)
    {
        /* There is nothing to receive. */
        return;
    }
```

广东省网络空间安全协会受控资料

```
lpduSize = *(uint16 *)(gp->lkInQHeadPtr + 1);

lpduHeaderPtr      = (LPDUHeader *)(gp->lkInQHeadPtr + 3);


/* Throw away packets that are smaller than 8 bytes long. */

/* this check is now  made in the mac sublayer */

/* Do CRC check. */

/* this check is now made in the mac sublayer */

/* Only packets with valid CRC and >= 8 bytes are placed

   in the lkInQ by mac sublayer. */


INCR_STATS(nmp->stats.layer2Received); /* Got a good packet. */



/* We need to receive this message. */

if (QueueFull(&gp->nwInQ))

{

   /* We are losing this packet. */

   INCR_STATS(nmp->stats.missedMessages);

}

else

{
```

广东省网络空间安全协会受控资料

```
    nwReceiveParamPtr = QueueTail(&gp->nwInQ);
    npduPtr           = (Byte *)(nwReceiveParamPtr + 1);

    nwReceiveParamPtr->priority = lpduHeaderPtr->priority;
    nwReceiveParamPtr->altPath  = lpduHeaderPtr->altPath;
    tempPtr = (Byte *)((char *)lpduHeaderPtr + 1);
    nwReceiveParamPtr->pduSize  = lpduSize - 3;


    /* Copy the NPDU. */
    /* if it was in link layer's queue, then the size should be
       sufficient in network layer's queue as they differ by 3.
       However, let us play safe by checking the size first. */
    if (nwReceiveParamPtr->pduSize <= gp->nwInBufSize)
    {
       memcpy(npduPtr, tempPtr, nwReceiveParamPtr->pduSize);
    }
    else
    {
       ErrorMsg("LKReceive: NPDU size seems too large.\n");
    }
    EnQueue(&gp->nwInQ);
}
```

广东省网络空间安全协会受控资料

```
    *(gp->lkInQHeadPtr) = 0;

    gp->lkInQHeadPtr = gp->lkInQHeadPtr + gp->lkInBufSize;

    if (gp->lkInQHeadPtr ==

        (gp->lkInQ + gp->lkInBufSize * gp->lkInQCnt))

    {

        gp->lkInQHeadPtr = gp->lkInQ; /* wrap around. */

    }



    return;

}
```

A.4 路由算法

```
Input:
    NPDU            the NPDU to be routed

Output:
    Decision        one of (Forward,Drop)

Uses:
    My_Domain       the domain this router is assigned to
    My_Subnet       the subnet within the domain that this side of the router is assigned to
    ROUTE_uc()      routing table
    ROUTE_mc()      routing table
    ROUTE_bc()      routing table
    RouterType      one of: Configured,Learning,Bridge,Repeater

Begin { routingalgorithm }

    If RouterType = Repeater Then Begin
        Decision := Forward;
        Return;
    end;
```

广东省网络空间安全协会受控资料

```
    If RouterType = Learning Then
        Execute ROUTING_EVENT of learning algorithm;

    If NPDU. Domain <> My_Domain Then
        If RouterType = Bridge or RouterType = Learning Then Begin
            Decision := Drop;
            Return;
        end;
        If NPDU. Domain <> Null_Domain Then Begin
            Decision := Drop;
            Return;
            end;
    Else If RouterType = Bridge Then Begin
        Decision := Forward;
        Return;
    end;

    Case NPDU. DestAddrFmt Of
        Subnet/Node:        Decision := ROUTE_uc(NPDU. DestSubnet);
        Group:              Decision := ROUTE_mc(NPDU. DestGroup);
        Broadcast:          Decision := ROUTE_bc(NPDU. DestSubnet);
    end case;

    Return;
end { routing algorithm};
```

广东省网络空间安全协会受控资料

A.5 自学习算法

Inputs:

INIT_EVENT

always occurs on system reboot; may also occur periodically, allowing the router to adapt to changes in network topology

ROUTING_EVENT

NPDU the NPDU to be routed

MySubnet the subnet the router is configured on for this side of the router

Output:

Defines routing function $ROUTE_{uc}()$

```
begin { learning algorithm }
```

```
    case event of
        INIT_EVENT:
            Set ROUTE_uc() := Forward for all subnet addresses;
            Set ROUTE_uc(MySubnet) := Drop;
            Set ROUTE_mc() := Forward for all group addresses;

        ROUTING_EVENT:
            ROUTE_uc(NPDU. SrcSubnet) := Drop
    end case;
end { learning algorithm };
```

A.6 事务控制算法

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        File:          tcs. c

     Version:          1. 7

     Purpose:          Interface file for transaction control sublayer.
                       Outgoing sequencing.
                       Incoming sequencing and duplicate detection.

        Note:          For assigning TIDs,a table is used. We
                       remember the last TID for each unique destination
                       address. When a new TID is requested for a
                       destination,this table is searched for that
                       destination. If found,we make sure that we
                       don't assign the same TID used for that
                       destination. If the destination is not
                       found,we make a new entry in the table.

                       We have an entry in the table for each
                       subnet/node,group,broadcast,subnet
                       broadcast,unique node id. When a table entry
                       is assigned,we remember the time stamp too.
                       If the table does not have space for a new
                       destination address,we get rid of one that has
                       remained more than 24 seconds. If there is no
                       such entry,then we fail to allocate the new
                       transaction ID. The table size is configurable.

      To Do:           None
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

广东省网络空间安全协会受控资料

```
/* *** START INFORMATIVE — Transaction ID Allcation *** */
/* These functions represent an example means of allocating transaction IDs.
 * There are in fact several valid mechanisms for allocating transaction IDs.
 * In addition to the mechanism below, there are at least two other accepted means
 * for allocating transaction IDs.
 * 1. Allocate a transaction ID per unique destination address.  This method should
 *    not be used if acknowledged or request/response using unique ID or broadcast
 *    addressing are using in time proximity with the other addressing modes.
 *    Note that such combinations can be accomplished in this scheme if guardbands
 *    are placed around expected arrivals of acks/responses per transaction ID.
 * 2. Allocate all transaction IDs from a single transaction ID space without
 *    conflict checking.  If this simple scheme is used, it is recommended that
 *    conflict checking be performed by the application.  */

/*-----------------------------------------------------------------------
Section: Includes
-----------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <CEA709_1.h>
#include <node.h>
#include <tes.h>

/*-----------------------------------------------------------------------
Section: Constant Definitions
-----------------------------------------------------------------------*/
/* Minimum amount of time in seconds a record in the priTbl or
   non-priTbl should stay before it can be replaced with a new
   entry. i.e., if the table is full and a new entry is needed,
   we look for an entry that has remained in the table for
   at least MIN_TABLE_TIME seconds.  */
#define MIN_TABLE_TIME 24

/*-----------------------------------------------------------------------
Section: Type Definitions
-----------------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------------
Section: Globals
-----------------------------------------------------------------------*/
/* None */
```

广东省网络空间安全协会受控资料

```
/*-----------------------------------------------------------------
Section: Function Prototypes
-----------------------------------------------------------------*/
/*****************************************************************
Function:   TCSReset
Returns:    None
Reference: Section 7,Protocol Specification.
Purpose:    To initialize all globals to proper values.
Comments:   None.
*****************************************************************/
void TCSReset(void)
{
   gp->priTransID      = 0; /* On node reset,transaction id 0 is used. */
   gp->nonpriTransID = 0;
   gp->priTransCtrlRec.inProgress      = FALSE;
   gp->nonpriTransCtrlRec.inProgress = FALSE;
   /* Reset the tables that keep track of (destination address
      transaction id) pairs only duing powerup or external reset.
      When resetCause is software reset or cleared,we keep this
      table to ensure that we don't send a message to a destination
      with TID same as the one used last time for that destination.
      For power-up or external reset,we also need to ensure that
      using some other technique. We will delay transport or session
      layer sends by a small amount so that no messages are pending in
      target nodes. If we don't follow these guidelines,the target
      node may throw away messages sent after a reset as duplicates. */
   if (nmp->resetCause == POWER_UP_RESET || nmp->resetCause == EXTERNAL_RESET)
   {
      gp->priTblSize        = 0;
      gp->nonpriTblSize     = 0;
   }
}

/*****************************************************************
Function:   NewTrans
Returns:    SUCCESS if a transaction id can be assigned.
            FAILURE if it is not possible to assign an id.
Purpose:    To get a new transaction id.
Comments:   This function implements a new algorithm to assign the
            transaction id. It does not use the one in protocol specification.
Alg Idea:   For each of the following categories,we have an
            entry in the table.
```

广东省网络空间安全协会受控资料

```
            1. Subnet/Node
            2. group
            3. broadcast (domainwide or subnet)
            4. unique node id
            When a new id is requested,we increment the TID from
            the single space. We then search the table for this
            entry. If a matching entry is found,then we check
            if that TID was used last time for the same destination.
            If so,we bump it up by one. If not we use it. In
            either case,we record this TID in the table.
            If there was no such entry,we create a new one.
            If there is no space for the new entry,we release one
            that has remained more than MIN_TABLE_TIME seconds.
            If no such entry,then we fail to assign a TID.
*******************************************/
Status NewTrans(Boolean    priorityIn,DestinationAddress addrIn,
                TransNum  *transNumOut)
{
   uint16              i;
   TransCtrlRecord *transRecPtr;
   TransNum            *transNumPtr;
   TIDTableEntry     *tbl;
   uint16             *tblSize;
   Boolean             found;

   /* Point to the appropriate control record & table. */
   if (priorityIn)
   {
      transRecPtr = &gp->priTransCtrlRec;
      transNumPtr = &gp->priTransID;
      tbl         = gp->priTbl;
      tblSize     = &gp->priTblSize;
   }
   else
   {
      transRecPtr = &gp->nonpriTransCtrlRec;
      transNumPtr = &gp->nonpriTransID;
      tbl         = gp->nonpriTbl;
      tblSize     = &gp->nonpriTblSize;
   }

   /* Check if transaction already in progress. */
   if (transRecPtr->inProgress)
```

广东省网络空间安全协会受控资料

```
{
    /* We can't allow a new transaction. Return failure. */
    return(FAILURE);
}

/* We can allow the transaction. Allocate a new TID. */
transRecPtr->transNum = *transNumPtr;

/* Make sure that this dest did not use this TID last time.
   If it did,increment the TID. */
/* Note: addrIn.addressMode can never be MULTICAST_ACK
         for transactions initiated by a node. */
found = FALSE;
for (i = 0; i < *tblSize; i++)
{
    /* Update timer for this i whether match or not. */
    UpdateMsTimer(&tbl[i].timer);

    /* If domainId does not match,skip entry. */
    if (addrIn.domainIndex != FLEX_DOMAIN &&
        (eep->domainTable[addrIn.domainIndex].len == 0xFF ||
         tbl[i].len != eep->domainTable[addrIn.domainIndex].len ||
         memcmp(eep->domainTable[addrIn.domainIndex].domainId,
                tbl[i].domainId,
                eep->domainTable[addrIn.domainIndex].len) != 0)
       )
    {
       continue; /* Not flex domain but domain mismatch. */
    }

    if (addrIn.domainIndex == FLEX_DOMAIN &&
        (tbl[i].len != addrIn.flexDomainLen ||
         memcmp(addrIn.flexDomainId,
                tbl[i].domainId,
                addrIn.flexDomainLen) != 0)
       )
    {
       continue; /* Flex domain but domain mismatch. */
    }

    switch(addrIn.addressMode)
    {
       case SUBNET_NODE:
```

广东省网络空间安全协会受控资料

```
            if (tbl[i].addressMode == SUBNET_NODE &&
                memcmp(&tbl[i].addr.subnetNode,&addrIn.addr.addr2a,
                        sizeof(SubnetAddress)) == 0)
            {
                found = TRUE;
            }
            break;
        case UNIQUE_NODE_ID:
            if (tbl[i].addressMode == UNIQUE_NODE_ID &&
                memcmp(tbl[i].addr.uniqueNodeId,
                        addrIn.addr.addr3.uniqueId,
                        UNIQUE_NODE_ID_LEN) == 0)
            {
                found = TRUE;
            }
            break;
        case MULTICAST:
            if (tbl[i].addressMode == MULTICAST &&
                tbl[i].addr.group   == addrIn.addr.addr1)
            {
                found = TRUE;
            }
            break;
        case BROADCAST:
            if (tbl[i].addressMode == BROADCAST &&
                tbl[i].addr.subnet == addrIn.addr.addr0)
            {
                found = TRUE;
            }
            break;
        default:
            ErrorMsg("NewTrans: Unexpected addressMode.\n");
            /* Should not come here. */
    }
    if (found)
    {
        break; /* Need to leave for loop with matched i value. */
    }
}

if (found)
{
    /* Found a match. Check if last TID is same or not. */
```

广东省网络空间安全协会受控资料

```
    /* We can reuse this entry and reinitialize timer.    */
    if (tbl[i].tid == *transNumPtr)
    {
        /* Increment TID.  */
        (*transNumPtr)++;
        if (*transNumPtr == 16)
        {
            *transNumPtr = 1; /* Wrap around.  */
        }
        transRecPtr->transNum = *transNumPtr;
    }
    tbl[i].tid    = *transNumPtr;
    SetMsTimer(&tbl[i].timer,
            (uint16)(MIN_TABLE_TIME * 1000));
    *transNumOut = *transNumPtr;
    transRecPtr->inProgress = TRUE;
    return(SUCCESS);
}

/* No match. Make a new entry. If no space. get a space.  */
/* All the timers must have been updated in the for loop above.  */
if (*tblSize == TID_TABLE_SIZE)
{
    /* Table is full. See if any entry can be replaced.  */
    found = FALSE;
    for (i = 0; i < *tblSize; i++)
    {
        if (tbl[i].timer.curTimerValue == 0) /* Expired */
        {
            found = TRUE;
            break;
        }
    }
    if (found)
    {
        /* Replace this entry with last entry of table.  */
        tbl[i] = tbl[*tblSize];
        (*tblSize)--;
        /* Fall through to code below.  */
    }
    else
    {
        /* Unable to find an entry.  */
```

广东省网络空间安全协会受控资料

```
        return(FAILURE);
    }
}

/* Now we have space for an entry. Add new entry. */
/* Store the domain len and domain id in table */
if (addrIn.domainIndex == FLEX_DOMAIN)
{
    memcpy(tbl[*tblSize].domainId,
            addrIn.flexDomainId,
            addrIn.flexDomainLen);
    tbl[i].len = addrIn.flexDomainLen;
}
else
{
    memcpy(tbl[*tblSize].domainId,
            eep->domainTable[addrIn.domainIndex].domainId,
            eep->domainTable[addrIn.domainIndex].len);
    tbl[i].len = eep->domainTable[addrIn.domainIndex].len;
}

tbl[*tblSize].addressMode = addrIn.addressMode;
if (addrIn.addressMode == MULTICAST)
{
    tbl[*tblSize].addr.group = addrIn.addr.addr1;
}
else if (addrIn.addressMode == SUBNET_NODE)
{
    tbl[*tblSize].addr.subnetNode = addrIn.addr.addr2a;
}
else if (addrIn.addressMode == UNIQUE_NODE_ID)
{
    memcpy(tbl[*tblSize].addr.uniqueNodeId,
            addrIn.addr.addr3.uniqueId,
            UNIQUE_NODE_ID_LEN);
}
else if (addrIn.addressMode == BROADCAST)
{
    tbl[*tblSize].addr.subnet = addrIn.addr.addr0;
}
else
{
    /* Should not come here as addressMode was checked before too. */
```

广东省网络空间安全协会受控资料

```
        ErrorMsg("NewTrans: Invalid addressMode at unexpected place\n");
    }
    SetMsTimer(&tbl[*tblSize].timer,(uint16)(MIN_TABLE_TIME * 1000));
    tbl[*tblSize].tid = *transNumPtr;
    *transNumOut      = *transNumPtr;
    (*tblSize)++;
    transRecPtr->inProgress = TRUE;
    return(SUCCESS);
}

/*****************************************************************
Function:   TransDone
Returns:    None
Reference: Section 7 Protocol Spec.
Purpose:    To release the transaction record for future assignments.
Comments:   None
*****************************************************************/
void TransDone(Boolean  priorityIn)
{
    TransCtrlRecord *transRecPtr;
    TransNum        *transNumPtr;

    /* Point to the appropriate control record & table. */
    if (priorityIn)
    {
        transRecPtr = &gp->priTransCtrlRec;
        transNumPtr = &gp->priTransID;
    }
    else
    {
        transRecPtr = &gp->nonpriTransCtrlRec;
        transNumPtr = &gp->nonpriTransID;
    }

    /* Mark transaction as available. */
    transRecPtr->inProgress = FALSE;

    /* Increment the corresponding transaction id. */
    (*transNumPtr)++;
    if (*transNumPtr == 16)
    {
        *transNumPtr = 1; /* Wrap Around to 1. */
    }
```

广东省网络空间安全协会受控资料

```
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   ValidateTrans
Returns:    TRANS_CURRENT if the transNumIn matches transaction
            in progress.
            TRANS_NOT_CURRENT othewise.
Reference: Section 7,Protocol Specification.
Purpose:    To check if a given transNumIn is current or not.
Comments:   None
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
TransStatus ValidateTrans(Boolean   priorityIn,
                          TransNum transNumIn)
{
   TransCtrlRecord * transRecPtr;

   /* Point to the appropriate control record & table. */
   if (priorityIn)
   {
      transRecPtr = &gp->priTransCtrlRec;
   }
   else
   {
      transRecPtr = &gp->nonpriTransCtrlRec;
   }

   if (transRecPtr->inProgress &&
       transRecPtr->transNum == transNumIn)
   {
      return(TRANS_CURRENT);
   }
   else
   {
      return(TRANS_NOT_CURRENT);
   }
}

/* * * * END INFORMATIVE — Transaction ID Allcation * * * */

/* ————————————————End of tcs.c———————————————— */
```

A.7 网络层算法

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

广东省网络空间安全协会受控资料

```
          File:        network.c

       Version:        1.7

     Reference:        Protocol Specification. Section 7.

       Purpose:        To implement network layer functions.

          Note:        Reference implementation does not support
                       special nodes such as routers and bridges.
                       Extra code is needed to implement these.

         To Do:        None
*************************************************************************/
/*-------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>

#include <CEA709_1.h>
#include <node.h>
#include <queue.h>
#include <network.h>

/*-------------------------------------------------------------------------
Section: Constants
-------------------------------------------------------------------------*/
/*  #define DEBUG  */

/*-------------------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------------------*/
/*************************************************************************
   Byte data[1] is used so that variable data has address assigned by
   the compiler. Once we know the size of the record, we will use
   data[0], data[1], etc.

   data[0] is source subnet.
   data[1] is source node.
   Based on the addrFmt field and 1st bit of data[1], the rest of
   the data array is used appropriately.
*************************************************************************/
```

广东省网络空间安全协会受控资料

```
#pragma maxalign(1)
typedef struct
{
   Bits        protocolVersion        :2;
   Bits        pduType                :2;
   Bits        addrFmt                :2;
   Bits        domainLength           :2;
   Byte        data[1];                    /* Variable part */
} NPDU;
#pragma maxalign()

/*-----------------------------------------------------------------
Section: Globals
-----------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------
Section: Local Function Prototypes
-----------------------------------------------------------------*/
static Byte DecodeDomainLength(Byte lengthCode);
static Byte EncodeDomainLength(Byte length);

/*-----------------------------------------------------------------
Section: Function Definitions
-----------------------------------------------------------------*/
/*****************************************************************
Function:   NWReset
Returns:    None
Reference: None
Purpose:    To initialize the queues used by the network layer.
Comments:   None.
*****************************************************************/
void    NWReset(void)
{
   uint16 queueItemSize;

   /* Allocate and initialize the input queue. */
   gp->nwInBufSize    =
      DecodeBufferSize((uint8)eep->readOnlyData.nwInBufSize);
   gp->nwInQCnt       = DecodeBufferCnt((uint8)eep->readOnlyData.nwInBufCnt);
   queueItemSize      = gp->nwInBufSize + sizeof(NWReceiveParam);
```

广东省网络空间安全协会受控资料

```
if (QueueInit(&gp->nwInQ,queueItemSize,gp->nwInQCnt) ! = SUCCESS)
{
    ErrorMsg("NWReset: Unable to init the input queue.\n");
    gp->resetOk = FALSE;
    return;
}

/* Allocate and initialize the output queue. */
gp->nwOutBufSize =
    DecodeBufferSize((uint8)eep->readOnlyData.nwOutBufSize);
gp->nwOutQCnt      = DecodeBufferCnt((uint8)eep->readOnlyData.nwOutBufCnt);
queueItemSize      = gp->nwOutBufSize + sizeof(NWSendParam);

if (gp->nwOutQCnt < 2)
{
    ErrorMsg("NWReset: Network non-pri buffers count should be >= 2.\n");
    gp->resetOk = FALSE;
    return;
}

if (QueueInit(&gp->nwOutQ,queueItemSize,gp->nwOutQCnt)
    ! = SUCCESS)
{
    ErrorMsg("NWReset: Unable to init the output queue.\n");
    gp->resetOk = FALSE;
    return;
}

/* Allocate and initialize the priority output queue. */
gp->nwOutPriBufSize = gp->nwOutBufSize;
gp->nwOutPriQCnt     =
    DecodeBufferCnt((uint8)eep->readOnlyData.nwOutBufPriCnt);
queueItemSize        = gp->nwOutPriBufSize + sizeof(NWSendParam);

if (gp->nwOutPriQCnt < 2)
{
    ErrorMsg("NWReset: Network pri buffers count should be >= 2.\n");
    gp->resetOk = FALSE;
    return;
}

if (QueueInit(&gp->nwOutPriQ,queueItemSize,gp->nwOutPriQCnt)
    ! = SUCCESS)
```

广东省网络空间安全协会受控资料

```
    {
        ErrorMsg("NWReset: Unable to init the priority output queue.\n");
        gp->resetOk = FALSE;
        return;
    }

    return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   NWSend
Returns:    None
Reference: None. No algorithms in protocol specification.
Purpose:    To send outgoing PDUS (APDU or SPDU or TPDU or AuthPDU)
            waiting on the queue (pri or nonpri) for network layer.
            Network layer forms the NPDU and the parameters for
            sending the NPDU and writes to the queue for the
            link/mac layer.
Comments:   Network buffer size is guaranteed to be at least
            20 bytes long as the encoding table's minimum value is 20.
            The NPDU's header's worst case size is 16. So,we are
            OK. No need to check for space when writing headers.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void    NWSend(void)
{
    NWSendParam    * nwSendParamPtr; /* Param in nwOutQ or nwPriOutQ.     */
    LKSendParam    * lkSendParamPtr; /* Param in lkOutQ or lkPriOutQ.     */
    APPReceiveParam * appReceiveParamPtr;
    NPDU           * npduPtr;        /* Pointer to NPDU being formed.     */
    Byte           * pduPtr;         /* Pointer to PDU etc being sent.    */
    Boolean        priority;         /* TRUE if processing pri msg.       */
    Byte           selField;         /* 0 or 1. Used to form NPDU.        */
    uint8          j;                /* For temporary use.                */
    uint16         npduSize;         /* Size of NPDU formed.              */
    uint8          numDomains;       /* Number of domains for this node.  */
    Boolean        flexDomain;       /* True if sending in flex domain.   */
    uint8          domainLength;     /* Length of domain value sent.      */
    Byte           domainId[DOMAIN_ID_LEN]; /* Value of domain.           */

    /* Check if there is work to do and set pointers */
    if ( ! QueueEmpty(&gp->nwOutPriQ) && ! QueueFull(&gp->lkOutPriQ) )
    {
        /* Process priority message if there is one and it can be processed. */
```

广东省网络空间安全协会受控资料

```
        priority          = TRUE;
        nwSendParamPtr    = QueueHead(&gp->nwOutPriQ);
        lkSendParamPtr    = QueueTail(&gp->lkOutPriQ);
    }
    else if ( ! QueueEmpty(&gp->nwOutQ) && ! QueueFull(&gp->lkOutQ) )
    {
        /* Process non-priority message if there is one and can be processed */
        priority          = FALSE;
        nwSendParamPtr    = QueueHead(&gp->nwOutQ);
        lkSendParamPtr    = QueueTail(&gp->lkOutQ);
    }
    else
    {
        /* Either there is nothing to send or there is no space in link layer */
        return;
    }

    /* For application layer messages,we need to give completion event
       using the tag given. This is for consistency with transport/session
       layers. Thus completiong events are streamlined in one place in
       application layer rather than lots of places. */
    if (nwSendParamPtr->pduType == APDU_TYPE && QueueFull(&gp->appInQ))
    {
        /* Can't deliver the indication. Wait until we can send indication */
        return;
    }

    /* Process the waiting PDU,form the NPDU and send it */

    /* ptr to APDU or TPDU or SPDU or AuthPDU. */
    pduPtr   = (Byte *)(nwSendParamPtr + 1);

    /* ptr to NPDU constructed. */
    npduPtr = (NPDU *)(lkSendParamPtr + 1);

    /* Write the NPDU header. */
    npduPtr->protocolVersion = PROTOCOL_VERSION; /* See CEA709_1.h */
    npduPtr->pduType         = nwSendParamPtr->pduType;
    switch (nwSendParamPtr->destAddr.addressMode)
    {
        case BROADCAST:
            npduPtr->addrFmt = 0;
              break;
```

广东省网络空间安全协会受控资料

```
    case MULTICAST:
        npduPtr->addrFmt = 1;
        break;
    case SUBNET_NODE:
    case MULTICAST_ACK:
        npduPtr->addrFmt = 2;
        break;
    case UNIQUE_NODE_ID:
        npduPtr->addrFmt = 3;
        break;
    default:
        ErrorMsg("NWSend: Unknown address mode.\n");
        /* Discard the packet as addrmode is wrong */
        nmp->errorLog = BAD_ADDRESS_TYPE;
        /* Send completion event if it was an APDU */
        if (nwSendParamPtr->pduType == APDU_TYPE)
        {
            appReceiveParamPtr = QueueTail(&gp->appInQ);
            appReceiveParamPtr->indication = COMPLETION;
            appReceiveParamPtr->success    = FALSE;
            appReceiveParamPtr->tag        = nwSendParamPtr->tag;
            EnQueue(&gp->appInQ);
        }
        if (priority)
        {
            DeQueue(&gp->nwOutPriQ);
        }
        else
        {
            DeQueue(&gp->nwOutQ);
        }
        return;
}

/* Write the domain length. */
/* First determine the number of domains for this node */
if (eep->readOnlyData.twoDomains == 1)
{
    numDomains = 2;
}
else
{
    numDomains = 1;
```

广东省网络空间安全协会受控资料

```
}

/* if a node is in in unconfigured state and the message is not in flex
   domain,then we discard the message. We should not use the domain table
   in unconfigured state,irrespective of whether they are valid or not.
   However,we allow acks,response,challenge and reply. The field
   dropIfUnconfigured indicates whether this check is done or not. */
if (nwSendParamPtr->dropIfUnconfigured &&
    nwSendParamPtr->destAddr.domainIndex ! = FLEX_DOMAIN &&
    NodeUnConfigured())
{
   /* drop this packet. */
   /* Send completion event if it was an APDU */
   if (nwSendParamPtr->pduType == APDU_TYPE)
   {
      appReceiveParamPtr = QueueTail(&gp->appInQ);
      appReceiveParamPtr->indication = COMPLETION;
      appReceiveParamPtr->success    = FALSE;
      appReceiveParamPtr->tag        = nwSendParamPtr->tag;
      EnQueue(&gp->appInQ);
   }
   if (priority)
   {
      DeQueue(&gp->nwOutPriQ);
   }
   else
   {
      DeQueue(&gp->nwOutQ);
   }
   return;
}

/* If the domain used is not in use,it cannot send any packet */
if (nwSendParamPtr->destAddr.domainIndex < numDomains &&
    eep->domainTable[nwSendParamPtr->destAddr.domainIndex].len
       == 0xFF)
{
   if (! nwSendParamPtr->dropIfUnconfigured)
   {
      /* It is not ACK,RESP etc. Don't log domain error in this case.
         LNS might use join domain to leave a domain with ACKD. So,the
         ACK send by the transport layer will be in an invalid domain
         but should be ignored. */
```

广东省网络空间安全协会受控资料

```
        /* Discard the packet as the domain table entry is not in use */
        nmp->errorLog = INVALID_DOMAIN;
    }
    /* Send completion event if it was an APDU */
    if (nwSendParamPtr->pduType == APDU_TYPE)
    {
        appReceiveParamPtr = QueueTail(&gp->appInQ);
        appReceiveParamPtr->indication = COMPLETION;
        appReceiveParamPtr->success    = FALSE;
        appReceiveParamPtr->tag        = nwSendParamPtr->tag;
        EnQueue(&gp->appInQ);
    }
    if (priority)
    {
        DeQueue(&gp->nwOutPriQ);
    }
    else
    {
        DeQueue(&gp->nwOutQ);
    }
    return;
}

/* Use destAddr to determine the domain and write it.
   compute and store domainLength and domainId for later use. */
if (nwSendParamPtr->destAddr.domainIndex < numDomains)
{
    /* One of this node's domains. */
    domainLength =
        eep->domainTable[nwSendParamPtr->destAddr.domainIndex].len;
    npduPtr->domainLength =
        EncodeDomainLength(
            eep->domainTable[nwSendParamPtr->destAddr.domainIndex].len
            );
    if (domainLength <= DOMAIN_ID_LEN)
    {
        memcpy(domainId,
                eep->domainTable[
                    nwSendParamPtr->destAddr.domainIndex].domainId,
                domainLength); /* Save id for now */
    }
    flexDomain = FALSE;
```

广东省网络空间安全协会受控资料

```
}
else if (nwSendParamPtr->destAddr.domainIndex == FLEX_DOMAIN)
{
    /* Flex domain message. */
    domainLength = nwSendParamPtr->destAddr.flexDomainLen;
    npduPtr->domainLength =
        EncodeDomainLength(
                nwSendParamPtr->destAddr.flexDomainLen
                );
    if (domainLength <= DOMAIN_ID_LEN)
    {
        memcpy(domainId,
                nwSendParamPtr->destAddr.flexDomainId,
                domainLength); /* Save the id for now */
    }
    flexDomain = TRUE;
}
else
{
    ErrorMsg("NWSend: Domain index is not valid.\n");
    nmp->errorLog = INVALID_DOMAIN;
    /* Send completion event if it was an APDU */
    if (nwSendParamPtr->pduType == APDU_TYPE)
    {
        appReceiveParamPtr = QueueTail(&gp->appInQ);
        appReceiveParamPtr->indication = COMPLETION;
        appReceiveParamPtr->success    = FALSE;
        appReceiveParamPtr->tag        = nwSendParamPtr->tag;
        EnQueue(&gp->appInQ);
    }
    /* Discard the packet as the domain index is invalid. */
    if (priority)
    {
        DeQueue(&gp->nwOutPriQ);
    }
    else
    {
        DeQueue(&gp->nwOutQ);
    }
    return;
}

if (domainLength != 0 && domainLength != 1 &&
```

广东省网络空间安全协会受控资料

```
    domainLength ! = 3 && domainLength ! = 6)
{
    /* Protocol specification indicates that domainLength has to be
       one of the above values. If not,it is a bad value. */
    ErrorMsg("NWSend: Domain length is not valid.\n");
    nmp->errorLog = INVALID_DOMAIN;
    /* Send completion event if it was an APDU */
    if (nwSendParamPtr->pduType == APDU_TYPE)
    {
       appReceiveParamPtr = QueueTail(&gp->appInQ);
       appReceiveParamPtr->indication = COMPLETION;
       appReceiveParamPtr->success     = FALSE;
       appReceiveParamPtr->tag         = nwSendParamPtr->tag;
       EnQueue(&gp->appInQ);
    }
    /* Discard the packet as the domain length is invalid. */
    if (priority)
    {
       DeQueue(&gp->nwOutPriQ);
    }
    else
    {
       DeQueue(&gp->nwOutQ);
    }
    return;
}

/* Write the address. It goes into data[0] onwards. */

/* First,write source subnet. */
if (flexDomain)
{
    npduPtr->data[0] = 0; /* It is 0 for flex domain response. */
}
else
{
    npduPtr->data[0] =
       eep->domainTable[nwSendParamPtr->destAddr.domainIndex].subnet;
}

/* Determine the selField value. */
/* Only MULTICAST_ACK has selector field as 0. For all others,it is 1. */
if (nwSendParamPtr->destAddr.addressMode == MULTICAST_ACK)
```

广东省网络空间安全协会受控资料

```
{
    selField = 0;
}
else
{
    selField = 1;
}

/* Write the source node. */
/* Node value is only 7 bits. The high order bit is selField. */
if (flexDomain)
{
    npduPtr->data[1] = selField << 7; /* SrcNode is 0. */
}
else
{
    npduPtr->data[1] = selField << 7 |
        eep->domainTable[nwSendParamPtr->destAddr.domainIndex].node;
}

/* Write the destination address. */
/* Set j to the index for writing the domain field. */
switch (nwSendParamPtr->destAddr.addressMode)
{
    case BROADCAST:
        npduPtr->data[2] = nwSendParamPtr->destAddr.addr.addr0;
        j = 3;
        break;
    case MULTICAST:
        npduPtr->data[2] = nwSendParamPtr->destAddr.addr.addr1;
        j = 3;
        break;
    case SUBNET_NODE:
        nwSendParamPtr->destAddr.addr.addr2a.selField = 1;
        memcpy(&npduPtr->data[2],
                        &nwSendParamPtr->destAddr.addr.addr2a,
                        2);
        j = 4;
        break;
    case MULTICAST_ACK:
        nwSendParamPtr->destAddr.addr.addr2b.subnetAddr.selField = 1;
        memcpy(&npduPtr->data[2],
                &nwSendParamPtr->destAddr.addr.addr2b,
```

广东省网络空间安全协会受控资料

```
                4);
        j = 6;
        break;
    case UNIQUE_NODE_ID:
        memcpy(&npduPtr->data[2],
                &nwSendParamPtr->destAddr.addr.addr3,
                7);
        j = 9;
        break;
    default:
        ErrorMsg("NWSend: Unknown address format.\n");
        /* Discard the packet as the address Mode is wrong. */
        nmp->errorLog = BAD_ADDRESS_TYPE;
        /* Send completion event if it was an APDU */
        if (nwSendParamPtr->pduType == APDU_TYPE)
        {
            appReceiveParamPtr = QueueTail(&gp->appInQ);
            appReceiveParamPtr->indication = COMPLETION;
            appReceiveParamPtr->success    = FALSE;
            appReceiveParamPtr->tag        = nwSendParamPtr->tag;
            EnQueue(&gp->appInQ);
        }
        if (priority)
        {
            DeQueue(&gp->nwOutPriQ);
        }
        else
        {
            DeQueue(&gp->nwOutQ);
        }
        return;
}

/* Now, j has the index of data field in which domain goes. */
/* Write the domain. We saved this information earlier. */
memcpy(&npduPtr->data[j], domainId, domainLength);
j += domainLength;

/* Write the enclosed PDU. */
if (1 + j + nwSendParamPtr->pduSize > gp->nwOutBufSize)
{
    /* Discard the packet as it is too long. */
    nmp->errorLog = WRITE_PAST_END_OF_NET_BUFFER;
```

广东省网络空间安全协会受控资料

```
        /* Send completion event if it was an APDU */
        if (nwSendParamPtr->pduType == APDU_TYPE)
        {
            appReceiveParamPtr = QueueTail(&gp->appInQ);
            appReceiveParamPtr->indication = COMPLETION;
            appReceiveParamPtr->success    = FALSE;
            appReceiveParamPtr->tag        = nwSendParamPtr->tag;
            EnQueue(&gp->appInQ);
        }
        if (priority)
        {
            DeQueue(&gp->nwOutPriQ);
        }
        else
        {
            DeQueue(&gp->nwOutQ);
        }
        return;
    }

    memcpy(&npduPtr->data[j], pduPtr, nwSendParamPtr->pduSize);
    /* NPDU size is header_size + enclosed PDU size. */
    npduSize = 1 + j + nwSendParamPtr->pduSize;

    /* Write the parameters for the link layer. */
    lkSendParamPtr->deltaBL = nwSendParamPtr->deltaBL;
    lkSendParamPtr->altPath = nwSendParamPtr->altPath;
    lkSendParamPtr->pduSize = npduSize;

    /* Update both queues. */
    if (priority)
    {
        DeQueue(&gp->nwOutPriQ);
        EnQueue(&gp->lkOutPriQ);
#ifdef DEBUG
        DebugMsg("NWSend: Sending a priority packet.");
#endif
    }
    else
    {
        DeQueue(&gp->nwOutQ);
        EnQueue(&gp->lkOutQ);
#ifdef DEBUG
```

广东省网络空间安全协会受控资料

```
        DebugMsg("NWSend: Sending a non-priority packet.");
#endif
    }

    INCR_STATS(nmp->stats.layer3Transmitted);

    /* Send completion event if it was an APDU */
    if (nwSendParamPtr->pduType == APDU_TYPE)
    {
        appReceiveParamPtr = QueueTail(&gp->appInQ);
        appReceiveParamPtr->indication = COMPLETION;
        appReceiveParamPtr->success    = TRUE;
        appReceiveParamPtr->tag        = nwSendParamPtr->tag;
        EnQueue(&gp->appInQ);
    }

    return;
}

/*****************************************************************
Function:   NWReceive
Returns:    None
Reference:  None. No receive algorithms in protocol specification.
Purpose:    To receive packets waiting for the network layer
            from the link layer. The NPDU is retrieved, processed
            and the enclosed PDU is sent to the proper destination queue.
            If the NPDU is not for this node, it is discarded.
Comments:   Discard packets originated from this node itself.
            It might receive such packets in the presence of repeaters.

            Discard packets in which version field is not correct.

            Discard packets that match a domain but not
            subnet or group.

            When a node is not in configured state and not in
            hard-offline state, it can only receive broadcast
            or matching Unique Node ID messages. In such a case,
            the domain on which it is received need not match its
            own domain. If it does not match, then the message is
            said to be received in a flex domain.
*****************************************************************/
void    NWReceive(void)
```

广东省网络空间安全协会受控资料

```
{
    NWReceiveParam        * nwReceiveParamPtr; /* Param in gp->nwInQ.      */
    APPReceiveParam       * appReceiveParamPtr;
    TSAReceiveParam       * tsaReceiveParamPtr;
    SourceAddress         srcAddr;      /* Address of source node.        */
    NPDU                  * npduPtr;    /* ptr to NPDU being received.    */
    Byte                  * pduPtr;     /* ptr to item in target queue.   */
    uint16                pduSize;      /* Size of enclosed PDU.          */
    Boolean               flexDomain; /* TRUE => NPDU in flexdomain.      */
    uint8                 numDomains; /* # of domains of this node.       */
    uint8                 domainLength;  /* Domain length                 */
    Byte                  domainId[DOMAIN_ID_LEN]; /* Temp.              */
    Byte                  uniqueNodeId[UNIQUE_NODE_ID_LEN]; /* Temp.     */
    SubnetAddress         destAddr;     /* Temp.                          */
    uint8                 j;            /* Temp.                          */

    /* First, check if we have any packets to process.  */
    if (QueueEmpty(&gp->nwInQ))
    {
        return; /* Nothing to process.  */
    }

    /* Until we determine what type of PDU we have, we cannot
       check for space availability in destination queue.
       Also, it is possible that the NPDU may very well be
       discarded.  */

    /* Set the pointer to NPDU in nwInQ.  */
    nwReceiveParamPtr = QueueHead(&gp->nwInQ);
    npduPtr           = (NPDU *)(nwReceiveParamPtr + 1);

    /* Discard NPDU if version is not PROTOCOL_VERSION  */
    if (npduPtr->protocolVersion ! = PROTOCOL_VERSION)
    {
        DeQueue(&gp->nwInQ);
#ifdef DEBUG
        DebugMsg(" NWReceive: Discard packet. Wrong version.\n");
#endif
        return;
    }

    /* Determine the source address.  */
    memcpy(&srcAddr.subnetAddr, npduPtr->data, 2);
```

广东省网络空间安全协会受控资料

```
/* Determine the destination address used and set srcAddr properly. */
/* For MULTICAST and MULTICAST_ACK address modes, the
   group and/or member values are copied into srcAddr.group
   or srcAddr.ackNode. For BROADCAST and SUBNET_NODE address
   modes, destAddr is used to store subnet and/or node
   values. For UNIQUE_NODE_ID, uniqueNodeId is used & subnet is ignored */
/* Also, set j to domain field's index. */
switch (npduPtr->addrFmt)
{
   case 0:
      srcAddr.addressMode = BROADCAST;
      destAddr.subnet     = npduPtr->data[2];
      j = 3;
      break;
   case 1:
      srcAddr.addressMode = MULTICAST;
      srcAddr.group       = npduPtr->data[2];
      j = 3;
      break;
   case 2:
      if (srcAddr.subnetAddr.selField == 1)
      {
         srcAddr.addressMode = SUBNET_NODE;
         memcpy(&destAddr, &npduPtr->data[2], 2);
         j = 4;
      }
      else
      {
         srcAddr.addressMode = MULTICAST_ACK;
         memcpy(&destAddr, &npduPtr->data[2], 2);
         memcpy(&srcAddr.ackNode.subnetAddr, &destAddr, 2);
         memcpy(&srcAddr.ackNode.groupAddr, &npduPtr->data[4], 2);
         j = 6;
      }
      break;
   case 3:
      srcAddr.addressMode = UNIQUE_NODE_ID;
      destAddr.subnet = npduPtr->data[2]; /* Routing Purpose */
      memcpy(uniqueNodeId, &npduPtr->data[3], UNIQUE_NODE_ID_LEN);
      j = 3 + UNIQUE_NODE_ID_LEN;
      break;
   default:
```

广东省网络空间安全协会受控资料

```
        /* Discard it as the address format is wrong. */
        ErrorMsg("NWReceive: Unknown addFmt.\n");
        nmp->errorLog = BAD_ADDRESS_TYPE;
        DeQueue(&gp->nwInQ);
        return;
    }

    /* Determine the domain. */
    domainLength = DecodeDomainLength(npduPtr->domainLength);
    if (domainLength != 0 && domainLength != 1 &&
        domainLength != 3 && domainLength != 6)
    {
        ErrorMsg("NWReceive: Domain length is not valid.\n");
        nmp->errorLog = INVALID_DOMAIN;
        /* Discard the packet as the domain length is invalid. */
        DeQueue(&gp->nwInQ);
        return;
    }

    /* domainLength is good. Safe to use memcpy now. */
    memcpy(domainId, &npduPtr->data[j], domainLength);

    j += domainLength; /* Now j points to enclosed PDU. */

    /* Determine the number of domains for this node. */
    if (eep->readOnlyData.twoDomains)
    {
        numDomains = 2;
    }
    else
    {
        numDomains = 1;
    }

    /* Check if the NPDU is received in flexDomain.
        If domainId does not match any of this node's domains,
        then the msg is said to have been received in flex domain. */

    flexDomain = FALSE; /* Assume it is not flex domain. */
    if (NodeConfigured() && eep->domainTable[0].len != 0xFF &&
        domainLength == eep->domainTable[0].len &&
        memcmp(domainId, eep->domainTable[0].domainId,
                domainLength) == 0)
```

广东省网络空间安全协会受控资料

```
{
    /* Matches domainId in index 0 */
    srcAddr.domainIndex = 0;
}
else if (NodeConfigured() && numDomains == 2 &&
            eep->domainTable[1].len != 0xFF &&
            domainLength == eep->domainTable[1].len &&
            memcmp(domainId, eep->domainTable[1].domainId,
                    domainLength) == 0)
{
    /* Matches domainId in index 1 */
    srcAddr.domainIndex = 1;
}
else
{
    /* Must be a flex domain. */
    srcAddr.domainIndex = FLEX_DOMAIN;
    srcAddr.flexDomainLen = domainLength;
    memcpy(srcAddr.flexDomainId, domainId, domainLength);
    flexDomain = TRUE;
}

/* Determine if the packet was sent by myself. If so, drop. */
/* We can do this check only in non-flexdomain as
    src subnet and node are 0 in flex domain. */
if (! flexDomain &&
     memcmp(&srcAddr.subnetAddr,
            &eep->domainTable[srcAddr.domainIndex].subnet, 2)
       == 0)
{
    /* Not flex domain and source addr matches. */
    DeQueue(&gp->nwInQ); /* Discard packet. */
#ifdef DEBUG
    DebugMsg("NWReceive. Discarding Self Pck\n");
#endif
    return;
}

/* Drop packet in various address modes if not for us. */
switch(srcAddr.addressMode)
{
    case BROADCAST:
       if (! flexDomain &&
```

广东省网络空间安全协会受控资料

```
            destAddr.subnet ! = 0 && /* subnet broadcast. */
            memcmp(&destAddr.subnet,
                    &eep->domainTable[srcAddr.domainIndex].subnet,
                    1) ! = 0)
        {
            /* Domain matches but destAddr does not. Not for us. */
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard BC pck. Not my subnet.\n");
#endif
            return;
        }
        srcAddr.broadcastSubnet = destAddr.subnet;
        break;
    case MULTICAST:
        if (! flexDomain &&
            ! IsGroupMember(srcAddr.domainIndex,
                            srcAddr.group, NULL) )
        {
            /* Domain matches but group does not. Not for us. */
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard MC pck. Not my group.\n");
#endif
            return;
        }
        break;
    case SUBNET_NODE:
        if (! flexDomain &&
            memcmp(&destAddr,
                    &eep->domainTable[srcAddr.domainIndex].subnet,
                    2) ! = 0)
        {
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard unicast logical packet. "
                     "Not my subnet (or subnode).\n");
#endif
            return;
        }
        break;
    case MULTICAST_ACK:
        /* Make sure the destination subnet/node matches. */
```

广东省网络空间安全协会受控资料

```
        if (! flexDomain &&
            memcmp(&destAddr,
                   &eep->domainTable[srcAddr.domainIndex].subnet,
                   2) ! = 0)
        {
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard multicast ack packet. "
                     "Not my subnet (or subnode).\n");
#endif
            return;
        }
        /* Also make sure that group matches. */
        if (! flexDomain &&
            ! IsGroupMember(srcAddr.domainIndex,
                            srcAddr.ackNode.groupAddr.group,
                            NULL) )
        {
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard multicast ack packet. "
                     "Not my group.\n");
#endif
            return;
        }
        break;
    case UNIQUE_NODE_ID:
        if (memcmp(uniqueNodeId,
                   eep->readOnlyData.uniqueNodeId,
                   UNIQUE_NODE_ID_LEN) ! = 0)
        {
            /* Unique Node Id message but not for our id. */
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard Unique Node ID packet. Not my Id.\n");
#endif
            return;
        }
        break;
    default:
        ; /* Null statement. */
        /* Error message has been already printed in the previous switch. */
        /* Control should not come here. But, let us play safe. */
```

广东省网络空间安全协会受控资料

```
        DeQueue(&gp->nwInQ);
        return;
    }


    /* If a node is in unconfigured state,
       only broadcast and Unique Node ID messages can be received. */
    if (NodeUnConfigured() &&
        srcAddr.addressMode != BROADCAST &&
        srcAddr.addressMode != UNIQUE_NODE_ID)
    {
        /* Drop the packet. */
        DeQueue(&gp->nwInQ);
#ifdef DEBUG
        DebugMsg("NWReceive: Discard packet. We are not online.\n");
#endif
        return;
    }

    /* Drop packets received on flexDomain if the state
       is not unconfigured and it is not Unique Node ID addressed. */
    /* Unique Node ID addressed packets are always received. */
    /* i.e if node is configured, flexdomain is not possible
       unless it is Unique Node ID addressed. */
    if (flexDomain &&
        NodeConfigured() &&
        srcAddr.addressMode != UNIQUE_NODE_ID)
    {
        /* Drop the packet. */
        DeQueue(&gp->nwInQ);
#ifdef DEBUG
        DebugMsg("NWReceive: Discard packet. Flex domain & not Neu. Id.\n");
#endif
        return;
    }

    /* We now got a packet that must be received. */
    INCR_STATS(nmp->stats.layer3Received);

    /* pduSize = npduSize - npduHeaderSize. */
    /* j is length of the variable part header of NPDU. */
    /* The fixed portion of NPDU header is always 1 byte. */
    pduSize = nwReceiveParamPtr->pduSize - j - 1;
```

广东省网络空间安全协会受控资料

```
/* Set the pdu pointer properly. */
switch (npduPtr->pduType)
{
    case APDU_TYPE:
      if (QueueFull(&gp->appInQ) ||
          pduSize > gp->appInBufSize)
      {
          /* No space or insufficient space. Discard packet. */
          if (pduSize > gp->appInBufSize)
          {
             nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
          }
          INCR_STATS(nmp->stats.lostMessages);
          DeQueue(&gp->nwInQ);
#ifdef DEBUG
          DebugMsg("NWReceive: Discard packet. Insufficient space.\n");
#endif
          return;
      }
      /* Queue is not full and buffer has sufficient space. */
      appReceiveParamPtr = QueueTail(&gp->appInQ);
      pduPtr = (Byte *) appReceiveParamPtr +
                          sizeof(APPReceiveParam);

      appReceiveParamPtr->indication = MESSAGE;
      appReceiveParamPtr->srcAddr    = srcAddr;
      appReceiveParamPtr->priority   = nwReceiveParamPtr->priority;
      appReceiveParamPtr->altPath    = nwReceiveParamPtr->altPath;
      appReceiveParamPtr->pduSize    = pduSize;
      appReceiveParamPtr->auth       = FALSE;
      appReceiveParamPtr->service    = UNACKD;
      memcpy(pduPtr, &npduPtr->data[j], pduSize);
      EnQueue(&gp->appInQ);
      INCR_STATS(nmp->stats.layer6_7MsgsRcvd);
      DeQueue(&gp->nwInQ);
      return;
    case TPDU_TYPE: /* Fall through. */
    case SPDU_TYPE: /* Fall through. */
    case AUTHPDU_TYPE:
      if (QueueFull(&gp->tsaInQ) ||
          pduSize > gp->tsaInBufSize)
      {
```

广东省网络空间安全协会受控资料

```
            /* No space or insufficient space. Discard packet. */
            if (pduSize > gp->tsaInBufSize)
            {
               /* Buffer sizes are based on app buf sizes. See
                  TSAReset function. */
               nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
            }
            INCR_STATS(nmp->stats.lostMessages);
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard packet. Insufficient space.\n");
#endif
            return;
         }
         /* Queue is not full and buffer has sufficient space. */
         tsaReceiveParamPtr = QueueTail(&gp->tsaInQ);
         pduPtr = (Byte *) tsaReceiveParamPtr +
                              sizeof(TSAReceiveParam);
         tsaReceiveParamPtr->pduType    = npduPtr->pduType;
         tsaReceiveParamPtr->srcAddr    = srcAddr;
         tsaReceiveParamPtr->priority   = nwReceiveParamPtr->priority;
         tsaReceiveParamPtr->altPath    = nwReceiveParamPtr->altPath;
         tsaReceiveParamPtr->pduSize    = pduSize;
         memcpy(pduPtr, &npduPtr->data[j], pduSize);
         EnQueue(&gp->tsaInQ);
         DeQueue(&gp->nwInQ);
         return;
      default:
         ErrorMsg("NWReceive: Unknown PDU was received.\n");
         nmp->errorLog = UNKNOWN_PDU;
         DeQueue(&gp->nwInQ);
         return;
   }

   /* Should not come here. */
}

/*****************************************************************
Function:   DecodeDomainLength
Returns:    Decoded value of domain length code.
Reference:  None.
Purpose:    To compute the actual domain length from code.
Comments:   None.
```

广东省网络空间安全协会受控资料

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static Byte DecodeDomainLength(Byte lengthCodeIn)
{
   switch(lengthCodeIn)
   {
       case 0:
                return(0);
       case 1:
                return(1);
       case 2:
                return(3);
       case 3:
                return(6);
       default:
          /* Impossible to come here as lengthCode is 2 bits. */
          ;
   }
   return(0); /* To silence the compiler from complaining. */
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   EncodeDomainLength
Returns:    Encode value of domain length.
Reference: None
Purpose:    To compute the encoded value of domain length given.
Comments:
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static Byte EncodeDomainLength(Byte lengthIn)
{
   switch(lengthIn)
   {
       case 0:
                return(0);
       case 1:
                return(1);
       case 3:
                return(2);
       case 6:
                return(3);
       default:
                return(0); /* should not come here. But has to return
                                  something. Chose 0 arbitrarily */
   }
```

广东省网络空间安全协会受控资料

```
}

/*----------------------End of network.c------------------------*/
```

A.8 带鉴别的 TPDU 和 SPDU 发送算法

```
/*****************************************************************
          File:          tsa.c (Transport Session Authentication)

       Version:          1.7

     Reference:          Sections 9, 10, Protocol Specification.

       Purpose:          Transport, Session and Authentication Layers.

          Note:          None.

         To Do:          None.
*****************************************************************/
/*---------------------------------------------------------------
Section: Includes
---------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <CEA709_1.h>
#include <node.h>
#include <queue.h>
#include <tcs.h>
#include <tsa.h>
#include <app.h> /* For TAG related macros and constants */

/*---------------------------------------------------------------
Section: Constant Definitions.
---------------------------------------------------------------*/
/* #define DEBUG */
/* The last few tries for a message are sent using alternate path.
   The following constant determines how many are sent like this.
   A message is sent on alternate path if retries_left <= ALT_PATH_COUNT.
   Thus actual # of messages sent on alternate path is ALT_PATH_COUNT + 1 */
#define ALT_PATH_COUNT 1
```

山东省网络空间安全协会受控资料

```
/*-----------------------------------------------------------------
Section: Type Definitions.
-----------------------------------------------------------------*/
typedef enum
{
   TRANSPORT,
   SESSION
} Layer;

typedef enum
{
   ACKD_MSG          = 0,  /* for Transport */
   REQUEST_MSG       = 0,  /* for Session   */
   CHALLENGE_MSG     = 0,  /* for Authentication */
   UNACK_RPT_MSG     = 1,  /* for Transport */
   ACK_MSG           = 2,  /* for Transport */
   RESPONSE_MSG      = 2,  /* for Session */
   REPLY_MSG         = 2,  /* for Authentication */
   REMINDER_MSG      = 4,  /* for Transport and Session */
   REM_MSG_MSG       = 5   /* for Transport and Session */
} PDUMsgType;  /* Type of msg sent in the PDU
                  (TPDU or SPDU or AuthPDU) */

#pragma maxalign(1)
typedef struct
{
   Bits        auth        :1;   /* Needs authentication? */
   Bits        pduMsgType  :3;   /* See PDUMsgType above */
   Bits        transNum    :4;
   Byte        data[1];          /* Variable length field */
} TSPDU; /* Transport or Session PDU */

typedef TSPDU *TSPDUPtr;

typedef struct
{
   Bits  fmt          :2;   /* Same as addrfmt.                  */
   Bits  pduMsgType   :2;   /* Type of AuthPDU. See PDUMsgType. */
   Bits  transNum     :4;   /* Transaction number.               */
   union {
      Byte randomBytes[8];    /* Random number for challenge.     */
      Byte cryptoBytes[8];    /* Encrypted value in response.     */
```

广东省网络空间安全协会受控资料

```
    } value;
    Byte group;                    /* Present only if fmt = 1.          */
} AuthPDU;
#pragma maxalign()

typedef AuthPDU *AuthPDUPtr;

/*-----------------------------------------------------------------------
Section: Globals.
-----------------------------------------------------------------------*/
/* None */


/* Array to convert address format to address mode. If format is 2,
   we convert to SUBNET_NODE instead of MULTICAST_ACK */
static Byte addrFmtToMode[4] =
{
    BROADCAST,      /* 0 */
    MULTICAST,      /* 1 */
    SUBNET_NODE,    /* 2 */
    UNIQUE_NODE_ID /* 3 */
};

/* Array to convert address mode to format. For example,
   BROADCAST to 0 MULTICAST to 1 etc. */
static Byte addrModeToFmt[6] =
{
    0,
    2, /* SUBNET_NODE    */
    3, /* UNIQUE_NODE_ID */
    0, /* BROADCAST      */
    1, /* MULTICAST      */
    2  /* MULTICAST_ACK */
};

/*-----------------------------------------------------------------------
Section: Local Function Prototypes.
-----------------------------------------------------------------------*/
/* Authentication related functions. */
static void InitiateChallenge(uint16 rrIndexIn);
static void SendReplyToChallenge(void);
static void ProcessReply(void);
```

广东省网络空间安全协会受控资料

```
/* Transport layer related functions. */
static void TPSendAck(uint16 rrIndexIn);
static void TPReceiveAck(void);

/* Session layer related functions. */
static void SNSendResponse(uint16 rrIndexIn, Boolean nullResponse);
static void SNReceiveResponse(void);

/* Functions that are common to both transport and session layers. */
static void XmitTimerExpiration(Layer layerIn, Boolean priorityIn);
static void TerminateTrans(Boolean priorityIn);
static void SendNewMsg(Layer layerIn, Boolean priorityIn);
static void ReceiveNewMsg(Layer layerIn);
static void ReceiveRem(Layer layerIn);
static void Deliver(uint16 rrIndexIn);

static int16 AllocateRR(void);
static int16 RetrieveRR(SourceAddress srcAddrIn, Boolean priorityIn);

static uint16 ComputeRecvTimerValue(AddrMode        addrModeIn,
                                    MulticastAddress group);
static void Encrypt(Byte rand[], APDU *apdu, uint16 apduSize,
            Byte domainIndex, Byte encryptValue[]);

/*-----------------------------------------------------------------
Section: Function Definitions.
-----------------------------------------------------------------*/
/*****************************************************************
Function:   TSAReset
Returns:    None
Reference:  None
Purpose:    To initialize the queues used by the transport and session
            layers and to initialize transmit and receive records.
Comments:   Sets gp->resetOk to FALSE if unable to reset properly.
*****************************************************************/
void TSAReset(void)
{
   uint16   queueItemSize;
   uint16   i;

   /* Allocate and initialize the input queue. */

   /* Some TSPDUs have APDU attached and others do not.
```

广东省网络空间安全协会受控资料

```
    The max # bytes for TSPDU with no APDU is 10 (REMINDER).
    The max header size for those with APDU is 4 (REM/MSG).  */
gp->tsaInBufSize =
    DecodeBufferSize((uint8)eep->readOnlyData.appInBufSize) + 4;
gp->tsaInBufSize = MAX(gp->tsaInBufSize, 10);
gp->tsaInQCnt      = DecodeBufferCnt((uint8)eep->readOnlyData.appInBufCnt);
queueItemSize      = gp->tsaInBufSize + sizeof(TSAReceiveParam);

if (QueueInit(&gp->tsaInQ, queueItemSize, gp->tsaInQCnt)
    != SUCCESS)
{
    ErrorMsg("TSAReset: Unable to initialize the input queue.");
    gp->resetOk = FALSE;
    return;
}

/* Allocate and initialize the output queue.  */
gp->tsaOutBufSize =
    DecodeBufferSize((uint8)eep->readOnlyData.appOutBufSize) + 4;
gp->tsaOutBufSize = MAX(gp->tsaOutBufSize, 10);
gp->tsaOutQCnt     =
    DecodeBufferCnt((uint8)eep->readOnlyData.appOutBufCnt);
queueItemSize      = gp->tsaOutBufSize + sizeof(TSASendParam);

if (QueueInit(&gp->tsaOutQ, queueItemSize, gp->tsaOutQCnt)
    != SUCCESS)
{
    ErrorMsg("TSAReset: Unable to initialize the output queue.");
    gp->resetOk = FALSE;
    return;
}

/* Allocate and initialize the priority output queue.  */
gp->tsaOutPriBufSize = gp->tsaOutBufSize;
gp->tsaOutPriQCnt =
    DecodeBufferCnt((uint8)eep->readOnlyData.appOutBufPriCnt);
queueItemSize      = gp->tsaOutPriBufSize + sizeof(TSASendParam);

if (QueueInit(&gp->tsaOutPriQ, queueItemSize, gp->tsaOutPriQCnt)
    != SUCCESS)
{
    ErrorMsg("TSAReset: Unable to initialize the priority output queue.");
    gp->resetOk = FALSE;
```

广东省网络空间安全协会受控资料

```
    return;
}

/* Allocate and initialize the responses queue. */
gp->tsaRespBufSize = gp->tsaOutBufSize;
gp->tsaRespQCnt    = gp->tsaOutQCnt;
queueItemSize      = gp->tsaRespBufSize + sizeof(TSASendParam);

if (QueueInit(&gp->tsaRespQ, queueItemSize, gp->tsaRespQCnt)
    != SUCCESS)
{
    ErrorMsg("TSAReset: Unable to initialize the responses queue.");
    gp->resetOk = FALSE;
    return;
}

/* Initialize the transmit records. */
gp->xmitRec.status     = UNUSED_TX;
gp->priXmitRec.status = UNUSED_TX;

/* Initialize the receive records. */
gp->recvRecCnt = RECEIVE_TRANS_COUNT;
gp->recvRec    = AllocateStorage((uint16)(gp->recvRecCnt *
                                  sizeof(ReceiveRecord)));
if (gp->recvRec == NULL)
{
    ErrorMsg("TSAReset: Insufficient space for allocating receive records.");
    gp->resetOk = FALSE;
    return;
}

for (i = 0; i < gp->recvRecCnt; i++)
{
    gp->recvRec[i].response =
        AllocateStorage(DecodeBufferSize((uint8) eep->readOnlyData.appOutBufSize));
    gp->recvRec[i].apdu =
        AllocateStorage(DecodeBufferSize((uint8) eep->readOnlyData.appInBufSize));
    if (gp->recvRec[i].response == NULL ||
        gp->recvRec[i].apdu     == NULL)
    {
        ErrorMsg("TSAReset: Insufficient space for response or apdu.");
        gp->resetOk = FALSE;
        return;
```

广东省网络空间安全协会受控资料

```
        }
        gp->recvRec[i].status = UNUSED_RR;
    }

    /* Initialize the running count for request id assignment. */
    gp->reqId = 0;

    return;
}

/*****************************************************************
Function:   TPSend
Returns:    None
Reference: Section 8, Protocol Specification.
Purpose:    To implement send algorithm for transport layer.
            If there is anything to be sent by transport layer,
            it processes that message and calls the right function
            that sends it.
Comments:   Update the priority transaction timer, if it exists.
            Update the non-priority transaction timer, if it exists.
            If the priority transaction timer expired then
                process this event.
            else if there is a priority message to be sent and there is space
                  in priority queue of the network layer then
                process the priority message.
            else if non-priority transaction timer expired then
                process that event.
            else if there is a non-priority message to be sent and there
                  is space in the non-priority queue of the network layer then
                process the non-priority message.
            else
                  there is nothing to do. return.
Note:
*****************************************************************/
void TPSend(void)
{

    /* Delay TPSend after power-up or external reset. */
    if ( gp->tsDelayTimer.curTimerValue > 0 &&
          (nmp->resetCause == POWER_UP_RESET ||
           nmp->resetCause == EXTERNAL_RESET)
        )
    {
```

广东省网络空间安全协会受控资料

```
    UpdateMsTimer(&gp->tsDelayTimer);
    return; /* Do nothing */
}

/* Update transmit timers, if they do exist. */
if (gp->priXmitRec.status == TRANSPORT_TX &&
     gp->priXmitRec.xmitTimer.curTimerValue > 0)
{
    UpdateMsTimer(&gp->priXmitRec.xmitTimer);
}
if (gp->xmitRec.status == TRANSPORT_TX &&
     gp->xmitRec.xmitTimer.curTimerValue > 0)
{
    UpdateMsTimer(&gp->xmitRec.xmitTimer);
}

/*****************************************************************
  Priority transaction timer expired event.
  *****************************************************************/
if (gp->priXmitRec.status == TRANSPORT_TX &&
     gp->priXmitRec.xmitTimer.curTimerValue == 0)
{
    XmitTimerExpiration(TRANSPORT, TRUE);
    return;
}
/*****************************************************************
  Send a new priority message event.
  *****************************************************************/
else if (gp->priXmitRec.status == UNUSED_TX &&
            ! QueueEmpty(&gp->tsaOutPriQ)        &&
            ! QueueFull(&gp->nwOutPriQ) )
{
    SendNewMsg(TRANSPORT, TRUE);
    return;
}
/*****************************************************************
  Non-priority transaction timer expired event.
  *****************************************************************/
else if (gp->xmitRec.status == TRANSPORT_TX &&
            gp->xmitRec.xmitTimer.curTimerValue == 0)
{
    XmitTimerExpiration(TRANSPORT, FALSE);
}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   Send a new non-priority message.
   * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
else if (gp->xmitRec.status == UNUSED_TX &&
            ! QueueEmpty(&gp->tsaOutQ)        &&
            ! QueueFull(&gp->nwOutQ) )
{
    SendNewMsg(TRANSPORT, FALSE);
}
else
{
    /* Either there is no work or there is no space. */
    return;
}

return;

}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   TerminateTrans
Returns:    None
Reference:  None
Purpose:    To terminate a transaction for transport or session
            layer and send the completion indication to application
            layer. If the application layer's input queue is full,
            we don't terminate the transaction.
Comments:   layerIn is not passed as it is not needed.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static void TerminateTrans(Boolean priorityIn)
{
    Queue              *tsaQPtr;       /* Pointer to source queue */
    TSASendParam       *tsaSendParamPtr;
    APPReceiveParam    *appReceiveParamPtr;
    TransmitRecord     *xmitRecPtr;    /* Ptr to xmit rec (pri or nonpri) */
    Boolean            success;

    if (QueueFull(&gp->appInQ))
    {
        return; /* Can't send the indication. Come back later. */
    }

    if (priorityIn)
```

广东省网络空间安全协会受控资料

```
    {
        tsaQPtr          = &gp->tsaOutPriQ;
        tsaSendParamPtr = QueueHead(tsaQPtr);
        xmitRecPtr       = &gp->priXmitRec;
    }
    else
    {
        tsaQPtr          = &gp->tsaOutQ;
        tsaSendParamPtr = QueueHead(tsaQPtr);
        xmitRecPtr       = &gp->xmitRec;
    }

    appReceiveParamPtr = QueueTail(&gp->appInQ);

    if (tsaSendParamPtr->service == UNACK_RPT ||
         xmitRecPtr->destCount == xmitRecPtr->ackCount ||
         (xmitRecPtr->nwDestAddr.addressMode == BROADCAST &&
          xmitRecPtr->ackCount >= 1)
        )
    {
        /* UNACK_RPT or ACK and got all acks(or resp). */
        success = TRUE;
    }
    else
    {
        INCR_STATS(nmp->stats.transmitTXFailures);
        success = FALSE; /* REQUEST or ACK and did not get all acks. */
    }
    appReceiveParamPtr->indication = COMPLETION;
    appReceiveParamPtr->success    = success;
    appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
    EnQueue(&gp->appInQ);

    TransDone(priorityIn); /* Call to TCS. */
    xmitRecPtr->status = UNUSED_TX;
#ifdef DEBUG
    if (success)
    {
        DebugMsg("Debug: TermTran: Terminated the transaction. Success.");
    }
    else
    {
        DebugMsg("Debug: TermTran: Terminated the transaction. Fail.");
```

广东省网络空间安全协会受控资料

```
    }
#endif
    /* Remove the transaction from the queue. */
    DeQueue(tsaQPtr);
    return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   XmitTimerExpiration
Returns:    None
Reference:  None
Purpose:    To process the XmitTimer expiration event (pri or nonpri).
            Retranmission and termination of transaction are
            handled. Retransmission might be reply to already
            initiated challenge or it might be the APDU itself.
            If there is no space for retransmission, the retry is
            lost.
Comments:   None
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static void XmitTimerExpiration(Layer layerIn, Boolean priorityIn)
{
    TSASendParam  * tsaSendParamPtr; /* Param in tsaQ (Pri or nonPri).    */
    NWSendParam   * nwSendParamPtr; /* Param in nwQ (Pri or nonPri).     */
    TransmitRecord  * xmitRecPtr;   /* Ptr to xmit rec (pri or nonpri). */
    Queue          * tsaQPtr;       /* Pointer to source queue.          */
    Queue          * nwQPtr;        /* Pointer to target queue.          */
    TSPDUPtr         pduPtr;        /* Pointer to TSPDU being formed.    */
    uint8          deltaBL;
    uint16         pduSize;
    int8           i;
    uint8          length; /* For length of reminder in bytes. */
    uint16         queueSpace;

    if (priorityIn)
    {
        tsaQPtr          = &gp->tsaOutPriQ;
        tsaSendParamPtr = QueueHead(tsaQPtr);
        nwQPtr           = &gp->nwOutPriQ;
        nwSendParamPtr   = QueueTail(nwQPtr);
        xmitRecPtr       = &gp->priXmitRec;
    }
    else
    {
```

广东省网络空间安全协会受控资料

```
        tsaQPtr          = &gp->tsaOutQ;
        tsaSendParamPtr = QueueHead(tsaQPtr);
        nwQPtr           = &gp->nwOutQ;
        nwSendParamPtr  = QueueTail(nwQPtr);
        xmitRecPtr       = &gp->xmitRec;
    }

    /* First, check if we really need to retry the message. */
    if (xmitRecPtr->retriesLeft == 0 ||
         xmitRecPtr->destCount == xmitRecPtr->ackCount ||
         (xmitRecPtr->nwDestAddr.addressMode == BROADCAST &&
          xmitRecPtr->ackCount >= 1)
        )
    {
        /* No More retries left or all acks have been received.
           Terminate the transaction. Send indication to the application
           layer. */
        TerminateTrans(priorityIn);
        return;
    }

    /* Check if there is space in the network buffer for retransmission */
    if (QueueFull(nwQPtr))
    {
        /* We are losing a retry chance locally due to lack of space
           in network queue. If we don't want to lose the retry, we
           simply delete the next two lines of code */
        xmitRecPtr->retriesLeft--;
        /* Start the transmit timer */
        SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);
#ifdef DEBUG
        DebugMsg("XmitTimerExp: Retry failure due to no space in net");
#endif
        return;
    }

    /* Now, we need to retransmit the message again. */
    /* Form the PDU to be sent directly in the target queue. */
    pduPtr              = (TSPDUPtr) (nwSendParamPtr + 1);
    pduPtr->auth         = tsaSendParamPtr->auth;
    pduPtr->transNum     = xmitRecPtr->transNum;

    if (tsaSendParamPtr->service == UNACK_RPT)
```

广东省网络空间安全协会受控资料

```
    {
        pduPtr->pduMsgType      = UNACK_RPT_MSG;
        memcpy(pduPtr->data, xmitRecPtr->apdu,
                xmitRecPtr->apduSize);
        pduSize = xmitRecPtr->apduSize + 1;
#ifdef DEBUG
        DebugMsg("XmitTimerExp: Resending UNACK_RPT packet.");
#endif
    }
    else if (xmitRecPtr->nwDestAddr.addressMode ! = MULTICAST)
    {
        if (layerIn == TRANSPORT)
        {
            pduPtr->pduMsgType = ACKD_MSG;
#ifdef DEBUG
            DebugMsg("XmitTimerExp: Resending ACKD packet.");
#endif
        }
        else if (tsaSendParamPtr->service == REQUEST)
        {
            pduPtr->pduMsgType = REQUEST_MSG;
#ifdef DEBUG
            DebugMsg("XmitTimerExp: Resending REQUEST packet.");
#endif
        }
        else
        {
            /* Response Messages are retried. Something is wrong. */
            /* Force retriesLeft to 0 so that next time we will
               terminate the transaction. */
            xmitRecPtr->retriesLeft = 0;
#ifdef DEBUG
            DebugMsg("XmitTimerExp: Response Message??. What is wrong?");
#endif
            return;
        }
        memcpy(pduPtr->data, xmitRecPtr->apdu,
                xmitRecPtr->apduSize);
        pduSize = xmitRecPtr->apduSize + 1;
    }
    else
    {
        /* Multicast Retry */
```

广东省网络空间安全协会受控资料

```
/* Compute the highest numbered group member that has
   acknowledged and form the M_LIST up to that. Since the
   rest of the nodes have not acknowledged, they will
   respond when they see that their bit is missing.
   However, the last byte of M_LIST should be padded with
   0 as those members need to acknowledge. Note that a node
   will respond if its member number is not even present
   in the M_List. */
/* Group members are 0..MAX_GROUP_NUMBER */
if (xmitRecPtr->ackCount == 0)
{
   length = 0;
}
else
{
   /* ackCount > 0, So, we have at least one ack.
      Find the highest member who have responded. */
   for (i = MAX_GROUP_NUMBER; i >= 0; i--)
   {
      if (xmitRecPtr->ackReceived[i])
      {
         break;
      }
   }
   if (i == -1)
   {
      /* There should have been at least one ack. */
      ErrorMsg("XmitTimerExpiration: Something is wrong."
               " Check code. Atleast one ack member expected.");
      xmitRecPtr->retriesLeft = 0;
      return;
   }
   length = i / 8 + 1; /* Number of bytes in M_List. */
}

if (length <= 2)
{
   pduPtr->pduMsgType      = REM_MSG_MSG;
   pduPtr->data[0]         = length; /* # of bytes in M_List. */
   /* Copy the M_LIST. See Fig 8.2 in Protocol Specification. */
   /* We use the fact that ackReceived[i] is 0 or 1. */
   /* The padding of 0's of last byte is automatic as
      ackReceived[i] for those are anyway 0. */
```

广东省网络空间安全协会受控资料

```
        pduPtr->data[1] = 0; /* Init anyway even if not used. */
        pduPtr->data[2] = 0; /* Init anyway even if not used. */
        for (i = 0; i < 8*length; i++)
        {
            pduPtr->data[1 + i / 8] |=
                (xmitRecPtr->ackReceived[i] << (i % 8));
        }
        /* Copy APDU. */
        memcpy(&pduPtr->data[1+length],
                xmitRecPtr->apdu,
                xmitRecPtr->apduSize);
        /* TSPDU = 1 byte header + 1 byte for length + M_LIST. */
        pduSize = xmitRecPtr->apduSize + 2 + length;
#ifdef DEBUG
        DebugMsg("XmitTimerExp: Resending REMINDER  packet.");
#endif
    }
    else
    {
        /* Length > 2 */
        /* A Pair is sent in this case. First, send the REMINDER
           and then send the ACKD or REQUEST message. */
        /* In this case, we are going to send two msgs.
           So, we need to make sure that the queue has space
           for 2 msgs. If not, return and come back later to
           do this case. */
        queueSpace = QueueCnt(nwQPtr) - QueueSize(nwQPtr);
        if (queueSpace < 2)
        {
            /* We are losing a retry chance locally due to lack
               of space in network queue. */
            xmitRecPtr->retriesLeft--;
            /* Start the transmit timer. */
            SetMsTimer(&xmitRecPtr->xmitTimer,
                    xmitRecPtr->xmitTimerValue);
#ifdef DEBUG
            DebugMsg("XmitTimerExp: Retry failure due to no"
                    " space in network buffer.");
#endif
            return; /* Not enough space in the queue. Come back. */
        }
        /* Send the REMINDER message. */
        if (tsaSendParamPtr->service == ACKD ||
```

广东省网络空间安全协会受控资料

```
 tsaSendParamPtr->service == REQUEST)
{
   nwSendParamPtr = QueueTail(nwQPtr);
   nwSendParamPtr->dropIfUnconfigured = TRUE;
   pduPtr = (TSPDUPtr)
      ((char *)nwSendParamPtr + sizeof(NWSendParam));

   pduPtr->auth          = tsaSendParamPtr->auth;
   pduPtr->pduMsgType    = REMINDER_MSG;
   pduPtr->transNum      = xmitRecPtr->transNum;
   pduPtr->data[0]       = length;

   /* Copy the M_LIST. See Fig 8.2 in Protocol Specification. */
   /* First, initialize all the M_LIST fields to 0. */
   for (i = 1; i <= length; i++)
   {
      pduPtr->data[i] = 0;
   }
   /* Set the bits for M_LIST field based on received acks. */
   for (i = 0; i < 8 * length; i++)
   {
      pduPtr->data[1 + i / 8] |=
         (xmitRecPtr->ackReceived[i] << (i % 8));
   }

   pduSize = 2 + length; /* REMINDER has no APDU. */

   /* Fill in the NWSendParam structure. */
   nwSendParamPtr->destAddr = xmitRecPtr->nwDestAddr;
   if (layerIn == TRANSPORT)
   {
      nwSendParamPtr->pduType    = TPDU_TYPE;
   }
   else
   {
      nwSendParamPtr->pduType    = SPDU_TYPE;
   }
   nwSendParamPtr->deltaBL   = 0; /* REMINDER has deltaBL 0. */
   nwSendParamPtr->pduSize   = pduSize;

   /* UnAck_rpt packets do not use alt path. */
   if (tsaSendParamPtr->service != UNACK_RPT)
   {
```

广东省网络空间安全协会受控资料

```
            nwSendParamPtr->altPath   =
                    (xmitRecPtr->retriesLeft <= (ALT_PATH_COUNT + 1));
        }
        else
        {
            nwSendParamPtr->altPath   = FALSE;
        }

        /* if altPath has override, use it */
        if (tsaSendParamPtr->altPathOverride)
        {
            nwSendParamPtr->altPath = tsaSendParamPtr->altPath;
        }

        /* Add TSPDU into the queue. */
        EnQueue(nwQPtr);
    }

    /* Send the ACKD or REQUEST. */
    nwSendParamPtr = QueueTail(nwQPtr);
    nwSendParamPtr->dropIfUnconfigured = TRUE;
    pduPtr = (TSPDUPtr)
        ((char *)nwSendParamPtr + sizeof(NWSendParam));
    pduPtr->auth          = tsaSendParamPtr->auth;
    pduPtr->transNum      = xmitRecPtr->transNum;
    if (tsaSendParamPtr->service == ACKD)
    {
        pduPtr->pduMsgType      = ACKD_MSG;
    }
    else
    {
        pduPtr->pduMsgType      = REQUEST_MSG;
    }
    memcpy(pduPtr->data,
            xmitRecPtr->apdu,
            xmitRecPtr->apduSize);
    pduSize = xmitRecPtr->apduSize + 1;

#ifdef DEBUG
    DebugMsg("XmitTimerExp: Resending REM/MSG pair.");
#endif
    }
  }
```

广东省网络空间安全协会受控资料

```
if (tsaSendParamPtr->service ! = UNACK_RPT)
{
    INCR_STATS(nmp->stats.transmitTXRetries);
}

/* Compute the delta backlog value. */
deltaBL = 1; /* for subnet and unique id messages. */
if (tsaSendParamPtr->service == UNACK_RPT)
{
    deltaBL = 0; /* Only on first attempt, deltaBL is retries left. */
}
else if (xmitRecPtr->nwDestAddr.addressMode == BROADCAST)
{
    /* Domainwide or subnet BROADCAST. */
    /* If there is no override value for deltaBL, then it is 15. */
    if (tsaSendParamPtr->destAddr.bcast.backlog)
    {
        deltaBL = tsaSendParamPtr->destAddr.bcast.backlog;
    }
    else
    {
        deltaBL = 15;
    }
}
else if (xmitRecPtr->nwDestAddr.addressMode == MULTICAST)
{
    /* deltaBL is outstanding responses or acknowledgements. */
    deltaBL = xmitRecPtr->destCount - xmitRecPtr->ackCount;
}

/* Fill in the NWSendParam structure */
nwSendParamPtr->destAddr = xmitRecPtr->nwDestAddr;
if (layerIn == TRANSPORT)
{
    nwSendParamPtr->pduType    = TPDU_TYPE;
}
else
{
    nwSendParamPtr->pduType    = SPDU_TYPE;
}
nwSendParamPtr->deltaBL    = deltaBL;
```

广东省网络空间安全协会受控资料

```
    /* UnAck_rpt packets do not use alt path. */
    if (tsaSendParamPtr->service ! = UNACK_RPT)
    {
        nwSendParamPtr->altPath  =
                (xmitRecPtr->retriesLeft <= (ALT_PATH_COUNT + 1));
    }
    else
    {
        nwSendParamPtr->altPath  = FALSE;
    }
    /* if altPath has override, use it */
    if (tsaSendParamPtr->altPathOverride)
    {
        nwSendParamPtr->altPath = tsaSendParamPtr->altPath;
    }


    nwSendParamPtr->pduSize  = pduSize;

    xmitRecPtr->retriesLeft--;

    /* Add TSPDU into the queue. */
    EnQueue(nwQPtr);

    /* Start the transmit timer. */
    SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);

    return;
}

/*****************************************************************
Function:   SendNewMsg
Returns:    None
Reference:  None
Purpose:    To process a new request from the application layer that
            is in the tsa output queue (pri or nonpri). Request or ACKD.
Comments:   This fn is called only if there is space in the
            corresponding queue of the network layer.
*****************************************************************/
static void SendNewMsg(Layer layerIn, Boolean priorityIn)
{
    Queue           *tsaQPtr;          /* Pointer to the source queue.      */
```

广东省网络空间安全协会受控资料

```
TSASendParam     * tsaSendParamPtr;/* Param in tsaQ (Pri or non-pri). */
Queue            * nwQPtr;          /* Pointer to target queue.        */
NWSendParam      * nwSendParamPtr; /* Param in nwQ (Pri or non-pri).   */
TransmitRecord * xmitRecPtr;       /* Ptr to xmit rec.                 */
DestinationAddress nwDestAddr;   /* Destination address.              */
TSPDUPtr         pduPtr;           /* Pointer to TSPDU being formed.  */
APPReceiveParam * appReceiveParamPtr;
Status           status;
uint16           rptTimer;
uint8            retryCount;
uint16           txTimer;
uint8            deltaBL;
uint16           nwBufSize;
int8             i;

if (priorityIn)
{
    tsaQPtr          = &gp->tsaOutPriQ;
    tsaSendParamPtr = QueueHead(tsaQPtr);
    nwQPtr           = &gp->nwOutPriQ;
    nwSendParamPtr   = QueueTail(nwQPtr);
    nwBufSize        = gp->nwOutPriBufSize;
    xmitRecPtr       = &gp->priXmitRec;
}
else
{
    tsaQPtr          = &gp->tsaOutQ;
    tsaSendParamPtr = QueueHead(tsaQPtr);
    nwQPtr           = &gp->nwOutQ;
    nwSendParamPtr   = QueueTail(nwQPtr);
    nwBufSize        = gp->nwOutBufSize;
    xmitRecPtr       = &gp->xmitRec;
}

/* If processing a new message, make sure that it is for this
   layer. If not, we are done. */

if (layerIn == TRANSPORT &&
     tsaSendParamPtr->service ! = ACKD &&
     tsaSendParamPtr->service ! = UNACK_RPT)
{
    return;
}
```

广东省网络空间安全协会受控资料

```
if (layerIn == TRANSPORT && NV_LAST_TAG(tsaSendParamPtr->tag))
{
    if (QueueFull(&gp->appInQ))
    {
        return;
    }
    /* Special tag used by application layer for synchronization.
       Send completion indication right away. */
    appReceiveParamPtr              = QueueTail(&gp->appInQ);
    appReceiveParamPtr->indication = COMPLETION;
    appReceiveParamPtr->success     = TRUE;
    appReceiveParamPtr->tag         = tsaSendParamPtr->tag;
    EnQueue(&gp->appInQ);
    DeQueue(tsaQPtr);
    return;
}

if (layerIn == SESSION && tsaSendParamPtr->service != REQUEST)
{
    /* Responses are placed in the response queue. */
    return;
}

/* Make sure that large group size is not used for ack
   or request service. */
if (tsaSendParamPtr->destAddr.group.groupFlag &&
    tsaSendParamPtr->destAddr.group.groupSize == 0 &&
    tsaSendParamPtr->service != UNACK_RPT)
{
    /* Large groups can only use unack or unack_rpt services. */
    /* Indicate failure of this message to application layer. */
    if (! QueueFull(&gp->appInQ))
    {
        appReceiveParamPtr              = QueueTail(&gp->appInQ);
        appReceiveParamPtr->indication = COMPLETION;
        appReceiveParamPtr->success     = FALSE;
        appReceiveParamPtr->tag         = tsaSendParamPtr->tag;
        EnQueue(&gp->appInQ);
        DeQueue(tsaQPtr);
    }
    else
    {
```

广东省网络空间安全协会受控资料

```
        /* We wait until appInQ has space for the indication. */
      }
      return;
    }
    /* Make sure that groupSize is in the proper range. */
    /* *** START INFORMATIVE — Group Size *** */
    /* See "START INFORMATIVE — Group Size" below. */
#ifdef GROUP_SIZE_COMPATIBILITY
    if (tsaSendParamPtr->destAddr.group.groupFlag &&
        (tsaSendParamPtr->destAddr.group.groupSize == 1 ||
         tsaSendParamPtr->destAddr.group.groupSize > MAX_GROUP_NUMBER+1) )
#else
    if (tsaSendParamPtr->destAddr.group.groupFlag &&
         tsaSendParamPtr->destAddr.group.groupSize > MAX_GROUP_NUMBER)
#endif
    /* *** END INFORMATIVE — Group Size *** */
    {
      /* Indicate failure of this message to application layer. */
      if (! QueueFull(&gp->appInQ))
      {
        appReceiveParamPtr              = QueueTail(&gp->appInQ);
        appReceiveParamPtr->indication = COMPLETION;
        appReceiveParamPtr->success     = FALSE;
        appReceiveParamPtr->tag         = tsaSendParamPtr->tag;
        EnQueue(&gp->appInQ);
        DeQueue(tsaQPtr);
      }
      else
      {
        /* We wait until appInQ has space for indication. */
      }
      return;
    }

    /* Make sure there is space in network buffer. If not, we fail. */
    /* apdu + (tran or session header of 1 byte) should fit. */
    if ((tsaSendParamPtr->apduSize + 1) > nwBufSize)
    {
      /* We can't send this message as it is too big for
         the network layer's buffer. */
      /* Right now, we haven't allocated any transmit record. */
      /* So, we directly give the indication to application. */
      if (! QueueFull(&gp->appInQ))
```

广东省网络空间安全协会受控资料

```
    {
        appReceiveParamPtr = QueueTail(&gp->appInQ);
        appReceiveParamPtr->indication = COMPLETION;
        appReceiveParamPtr->success     = FALSE;
        appReceiveParamPtr->tag         = tsaSendParamPtr->tag;
        EnQueue(&gp->appInQ);
        DeQueue(tsaQPtr);
    }
    else
    {
        /* We wait until appInQ has space for indication. */
    }
    return;
}

/* First, compute nwDestAddr from destAddr. */
/* First, initialize domainIndex. Only if it is COMPUTE_DOMAIN_INDEX,
   we need to recompute it based on destAddr field value. */
nwDestAddr.domainIndex = tsaSendParamPtr->domainIndex;
switch (tsaSendParamPtr->destAddr.noAddress)
{
    case UNBOUND:
        /* Not in use or turnaround format. */
        if (! QueueFull(&gp->appInQ))
        {
            ErrorMsg("SendNewMsg: UNBOUND destination address is invalid.");
            appReceiveParamPtr = QueueTail(&gp->appInQ);
            appReceiveParamPtr->indication = COMPLETION;
            appReceiveParamPtr->success     = FALSE;
            appReceiveParamPtr->tag         = tsaSendParamPtr->tag;
            EnQueue(&gp->appInQ);
            DeQueue(tsaQPtr);
        }
        else
        {
            /* We wait until appInQ has space for indication. */
        }
        return;
    case SUBNET_NODE:
        nwDestAddr.addressMode = SUBNET_NODE;
        if (tsaSendParamPtr->domainIndex == COMPUTE_DOMAIN_INDEX)
        {
            /* Use the domainIndex from destAddr. */
```

广东省网络空间安全协会受控资料

```
        nwDestAddr.domainIndex =
            tsaSendParamPtr->destAddr.snode.domainIndex;
    }
    nwDestAddr.addr.addr2a.subnet =
        tsaSendParamPtr->destAddr.snode.subnetID;
    nwDestAddr.addr.addr2a.selField = 1; /* always 1 */
    nwDestAddr.addr.addr2a.node    =
        tsaSendParamPtr->destAddr.snode.node;
    txTimer =
        DecodeTxTimer((uint8) tsaSendParamPtr->destAddr.snode.txTimer);
    rptTimer =
        DecodeRptTimer((uint8) tsaSendParamPtr->destAddr.snode.rptTimer);
    retryCount =
        tsaSendParamPtr->destAddr.snode.retryCount;
    break;
case UNIQUE_NODE_ID:
    nwDestAddr.addressMode = UNIQUE_NODE_ID;
    if (tsaSendParamPtr->domainIndex == COMPUTE_DOMAIN_INDEX)
    {
        nwDestAddr.domainIndex =
            tsaSendParamPtr->destAddr.uniqueNodeId.domainIndex;
    }
    nwDestAddr.addr.addr3.subnet =
        tsaSendParamPtr->destAddr.uniqueNodeId.subnetID;
    memcpy(nwDestAddr.addr.addr3.uniqueId,
            tsaSendParamPtr->destAddr.uniqueNodeId.uniqueId,
            UNIQUE_NODE_ID_LEN);
    txTimer =
        DecodeTxTimer((uint8) tsaSendParamPtr->destAddr.uniqueNodeId.txTimer);
    rptTimer =
        DecodeRptTimer((uint8) tsaSendParamPtr->destAddr.uniqueNodeId.rptTimer);
    retryCount =
        tsaSendParamPtr->destAddr.uniqueNodeId.retryCount;
    break;
case BROADCAST:
    /* The following field is not used in this mode by
        application. It is however used in BROADCAST_GROUP. */
    tsaSendParamPtr->destAddr.bcast.maxResponses = 1;
    /* Fall Through */
/* *** START INFORMATIVE — Broadcast Group */
/* Broadcast group addressing is optional.   It can not be assumed that all devices
 * support this form of addressing. */
case BROADCAST_GROUP:
```

广东省网络空间安全协会受控资料

```
        nwDestAddr.addressMode = BROADCAST;
        if (tsaSendParamPtr->domainIndex == COMPUTE_DOMAIN_INDEX)
        {
            nwDestAddr.domainIndex =
                tsaSendParamPtr->destAddr.bcast.domainIndex;
        }
        nwDestAddr.addr.addr0 =
            tsaSendParamPtr->destAddr.bcast.subnetID;
        txTimer =
            DecodeTxTimer((uint8) tsaSendParamPtr->destAddr.bcast.txTimer);
        rptTimer =
            DecodeRptTimer((uint8) tsaSendParamPtr->destAddr.bcast.rptTimer);
        retryCount =
            tsaSendParamPtr->destAddr.bcast.retryCount;
        break;
    /* *** END INFORMATIVE — Broadcast Group */
    default:
        /* Must be group format unless it is an invalid value. */
        if (tsaSendParamPtr->destAddr.group.groupFlag != 1)
        {
            /* It must be some invalid value. Let us fail. */
            nmp->errorLog = BAD_ADDRESS_TYPE;
            if (! QueueFull(&gp->appInQ))
            {
                ErrorMsg("SendNewMsg: Invalid group format addr.");
                appReceiveParamPtr = QueueTail(&gp->appInQ);
                appReceiveParamPtr->indication = COMPLETION;
                appReceiveParamPtr->success    = FALSE;
                appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
                EnQueue(&gp->appInQ);
                DeQueue(tsaQPtr);
            }
            else
            {
                /* We wait until appInQ has space for indication. */
            }
            return;
        }
        nwDestAddr.addressMode = MULTICAST;
        if (tsaSendParamPtr->domainIndex == COMPUTE_DOMAIN_INDEX)
        {
            nwDestAddr.domainIndex =
                tsaSendParamPtr->destAddr.group.domainIndex;
```

广东省网络空间安全协会受控资料

```
        }
        nwDestAddr.addr.addr1 =
            tsaSendParamPtr->destAddr.group.groupID;
        txTimer =
            DecodeTxTimer((uint8) tsaSendParamPtr->destAddr.group.txTimer);
        rptTimer =
            DecodeRptTimer((uint8) tsaSendParamPtr->destAddr.group.rptTimer);
        retryCount =
            tsaSendParamPtr->destAddr.group.retryCount;
    } /* switch */

    if (nwDestAddr.domainIndex == FLEX_DOMAIN)
    {
        /* flex domain. Copy the domain id. */
        nwDestAddr.flexDomainLen = tsaSendParamPtr->flexDomainLen;
        if (tsaSendParamPtr->flexDomainLen <= DOMAIN_ID_LEN)
        {
            memcpy(nwDestAddr.flexDomainId,
                   tsaSendParamPtr->flexDomainId,
                   tsaSendParamPtr->flexDomainLen);
        }
    }
    /* Get transaction number using nwDestAddr. */
    status = NewTrans(priorityIn, nwDestAddr, &xmitRecPtr->transNum);
    if (status == FAILURE)
    {
        /* Unable to get the transaction number. Give up. Try later. */
        return;
    }

    /* Initialize the xmit record. */
    if (layerIn == TRANSPORT)
    {
        xmitRecPtr->status = TRANSPORT_TX;
    }
    else
    {
        xmitRecPtr->status = SESSION_TX;
    }
    xmitRecPtr->nwDestAddr = nwDestAddr; /* Save it for future use. */
    for (i = 0; i <= MAX_GROUP_NUMBER; i++)
    {
        xmitRecPtr->ackReceived[i] = FALSE;
```

广东省网络空间安全协会受控资料

```
    }

    if (nwDestAddr.addressMode == MULTICAST)
    {
        /* If the node is a member of the group, then groupsize
           field is set to 1 more than actual group size
           by in  app layer or app pgm.
           However, provide an option to set this to the
           true group size and transport and session layers
           will take care of this.  */
        xmitRecPtr->destCount =
           tsaSendParamPtr->destAddr.group.groupSize - 1;
        /*  * * * START INFORMATIVE - Group Size * * *  */
        /* To be fully compatible with all applications, you must use the
              GROUP_SIZE_COMPATIBILITY option.   This behavior only affects explicitly
               addressed messages originated by the application.  */
#ifndef GROUP_SIZE_COMPATIBILITY
        if (! IsGroupMember(tsaSendParamPtr->destAddr.group.domainIndex,
                            tsaSendParamPtr->destAddr.group.group,
                            &groupMember))
        {
           /* node is not a member & group size is true size */
           xmitRecPtr->destCount =
              tsaSendParamPtr->destAddr.group.groupSize;
        }
#endif
        /*  * * * END INFORMATIVE - Group Size * * *  */
        if (xmitRecPtr->destCount == 0)
        {
           /* If the value is incorrect, set it to 1. We need at least
              one acknowledgment or response.  */
#ifdef DEBUG
           DebugMsg("SendNewMsg: groupSize incorrect. Default assumed.");
#endif
           xmitRecPtr->destCount = 1;
        }
    }
    else
    {
        xmitRecPtr->destCount = 1;
    }

    xmitRecPtr->retriesLeft = retryCount;
```

广东省网络空间安全协会受控资料

```
    xmitRecPtr->ackCount      = 0;
    xmitRecPtr->apdu          = (APDU *)(tsaSendParamPtr + 1);
    xmitRecPtr->apduSize      = tsaSendParamPtr->apduSize;
    if (tsaSendParamPtr->service == UNACK_RPT)
    {
        xmitRecPtr->xmitTimerValue = rptTimer;
    }
    else
    {
        xmitRecPtr->xmitTimerValue = txTimer;
    }

    /* Form the TSPDU to be sent directly in queue. */
    pduPtr                    = (TSPDUPtr)
        ((char *)nwSendParamPtr + sizeof(NWSendParam));
    pduPtr->auth              = tsaSendParamPtr->auth;
    /* Save auth status so that if a challenge comes later on
        we can at least verify that it was legitimate challenge */
    xmitRecPtr->auth          = tsaSendParamPtr->auth;
    if (tsaSendParamPtr->service == UNACK_RPT)
    {
        pduPtr->pduMsgType        = UNACK_RPT_MSG;
#ifdef DEBUG
            DebugMsg("SendNewMsg: Sending a new UNACK_RPT packet.");
#endif
    }
    else if (layerIn == TRANSPORT)
    {
        pduPtr->pduMsgType        = ACKD_MSG;
#ifdef DEBUG
            DebugMsg("SendNewMsg: Sending a new ACKD packet.");
#endif
    }
    else if (tsaSendParamPtr->service == REQUEST)
    {
        pduPtr->pduMsgType        = REQUEST_MSG;
#ifdef DEBUG
            DebugMsg("SendNewMsg: Sending a new REQ packet.");
#endif
    }
    else
    {
#ifdef DEBUG
```

广东省网络空间安全协会受控资料

```
        DebugMsg("SendNewMsg: Something is wrong. Unknown service.");
#endif
    }
    pduPtr->transNum      = xmitRecPtr->transNum;
    memcpy(pduPtr->data, xmitRecPtr->apdu,
            xmitRecPtr->apduSize);

    /* Compute the delta backlog value. */
    deltaBL = 1; /* For subnet and unique node id messages. */
    if (tsaSendParamPtr->service == UNACK_RPT_MSG)
    {
        deltaBL = xmitRecPtr->retriesLeft;
    }
    else if (nwDestAddr.addressMode == BROADCAST)
    {
        if (tsaSendParamPtr->destAddr.bcast.backlog)
        {
            deltaBL = tsaSendParamPtr->destAddr.bcast.backlog;
        }
        else
        {
            deltaBL = 15;
        }
    }
    else if (nwDestAddr.addressMode == MULTICAST)
    {
        deltaBL = xmitRecPtr->destCount;
    }

    /* Fill in the NWSendParam structure. */
    nwSendParamPtr->dropIfUnconfigured = TRUE;
    nwSendParamPtr->destAddr = nwDestAddr;
    if (layerIn == TRANSPORT)
    {
        nwSendParamPtr->pduType   = TPDU_TYPE;
    }
    else
    {
        nwSendParamPtr->pduType   = SPDU_TYPE;
    }
    nwSendParamPtr->deltaBL   = deltaBL;

    /* UnAck_rpt packets do not use alternate path. */
```

广东省网络空间安全协会受控资料

```
    if (tsaSendParamPtr->service ! = UNACK_RPT)
    {
        nwSendParamPtr->altPath   = (retryCount <= ALT_PATH_COUNT);
    }
    else
    {
        nwSendParamPtr->altPath   = FALSE;
    }

    /* if altPath has override, use it */
    if (tsaSendParamPtr->altPathOverride)
    {
        nwSendParamPtr->altPath = tsaSendParamPtr->altPath;
    }

    nwSendParamPtr->pduSize   = xmitRecPtr->apduSize + 1;

    /* Add the TSPDU into the queue. */
    EnQueue(nwQPtr);

    /* Start the transmit timer. */
    SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);

    return;
}


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   TPReceive
Returns:    None
Reference: None
Purpose:    To receive and process incoming TPDU's by calling
            apprpriate functions. Also handle receive timers.
Comments:   None
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void TPReceive(void)
{
    TSAReceiveParam *tsaReceiveParamPtr;/* Param in tsaQ. */
    TSPDUPtr        pduInPtr;           /* Pointer to TPDU received. */
    int16           i;

    /* Update all the receive timers, if they do exist */
    for (i = 0; i < gp->recvRecCnt; i++)
```

广东省网络空间安全协会受控资料

```
  {
      if (gp->recvRec[i].status == TRANSPORT_RR)
      {
         if (gp->recvRec[i].recvTimer.curTimerValue > 0)
         {
             UpdateMsTimer(&gp->recvRec[i].recvTimer);
         }
         if (gp->recvRec[i].recvTimer.curTimerValue == 0)
         {
             /* Timer expired. See if RR can be released. */
#ifdef DEBUG
             DebugMsg("TPReceive: Receive timer expired.");
#endif

             /* Deliver the message if it has not been delivered. */
             /* Deliver fn checks for authentication */
             if (gp->recvRec[i].transState == DELIVERED ||
                 gp->recvRec[i].transState == DONE)
             {
                gp->recvRec[i].status = UNUSED_RR; /* Release the RR */
             }
             else
             {
                Deliver(i);
                if (gp->recvRec[i].serviceType == ACKD &&
                    gp->recvRec[i].transState == DELIVERED)
                {
                    TPSendAck(i); /* may get lost due to no space. */
                }
             }
         }
         else if ((gp->recvRec[i].transState ==JUST_RECEIVED &&
                                          ! gp->recvRec[i].needAuth) ||
                 gp->recvRec[i].transState == AUTHENTICATED )
         {
             /* The request does not need authentication and just received
                or it has been authenticated */
             Deliver(i);
             if (gp->recvRec[i].serviceType == ACKD &&
                 gp->recvRec[i].transState == DELIVERED)
             {
                TPSendAck(i); /* may get lost due to no space. */
             }
```

广东省网络空间安全协会受控资料

```
            /* We can't release RR as it is not expired yet. */
        }
    }
}

/* Check if there is TPDU to be processed. */
if (QueueEmpty(&gp->tsaInQ))
{
    /* There is nothing to process. */
    return;
}

tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
pduInPtr           = (TSPDUPtr) (tsaReceiveParamPtr + 1);

/* Check if the PDU is for transport layer. If not, we are done. */
if (tsaReceiveParamPtr->pduType ! = TPDU_TYPE)
{
    return;
}

/* We have a TPDU to be processed. Check the type and
   process accordingly. */
switch (pduInPtr->pduMsgType)
{
    case ACK_MSG:
        TPReceiveAck();
        return;
    case ACKD_MSG:
        /* Fall through. */
    case UNACK_RPT_MSG:
        ReceiveNewMsg(TRANSPORT);
        return;
    case REMINDER_MSG:
        /* Fall through. */
    case REM_MSG_MSG:
        ReceiveRem(TRANSPORT);
        return;
    default:
        ErrorMsg("TPReceive: Unknown TPDU type received.");
        DeQueue(&gp->tsaInQ);
        nmp->errorLog = UNKNOWN_PDU;
}
```

广东省网络空间安全协会受控资料

```
    return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   TPReceiveAck
Returns:    None
Reference: None
Purpose:    To process ACK message received by the transport layer.
Comments:   ACK might correspond to a transmit record or it may be
            stale, in which case we ignore it.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static void TPReceiveAck(void)
{
    TSAReceiveParam  * tsaReceiveParamPtr;
    TSPDUPtr            pduPtr;
    TransmitRecord   * xmitRecPtr;   /* Ptr to xmit rec (pri or nonpri). */

    tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
    pduPtr             = (TSPDUPtr) (tsaReceiveParamPtr + 1);

    /* Set the corresponding transmit pointer. */
    if (tsaReceiveParamPtr->priority)
    {
        xmitRecPtr = &gp->priXmitRec;
    }
    else
    {
        xmitRecPtr = &gp->xmitRec;
    }

    if (ValidateTrans(tsaReceiveParamPtr->priority, pduPtr->transNum)
        == TRANS_NOT_CURRENT)
    {
        /* Stale ACK. Ignore it. */
        DeQueue(&gp->tsaInQ);
#ifdef DEBUG
        DebugMsg("Debug: TPReceiveAck: Stale Ack Discarded");
#endif
        INCR_STATS(nmp->stats.lateAcknowledgements);
        return;
    }
```

广东省网络空间安全协会受控资料

```
  /* Check if the ACK really corresponds to the transaction in progress. */
  if (xmitRecPtr->status != TRANSPORT_TX ||
      xmitRecPtr->nwDestAddr.domainIndex !=
      tsaReceiveParamPtr->srcAddr.domainIndex)
  {
      /* Transmit record is not ours or the domain index for Ack
         and the transaction do not match. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: TPReceiveAck: Stale ack discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
  }

  if (xmitRecPtr->nwDestAddr.addressMode == SUBNET_NODE &&
      (xmitRecPtr->nwDestAddr.addr.addr2a.subnet !=
        tsaReceiveParamPtr->srcAddr.subnetAddr.subnet ||
        xmitRecPtr->nwDestAddr.addr.addr2a.node !=
        tsaReceiveParamPtr->srcAddr.subnetAddr.node))
  {
      /* Source address of ACK does not match destination address of
         transmit record. */
      /* ACK does not seem to correspond to what we are expecting. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: TPReceiveAck: Stale Ack discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
  }

  if (xmitRecPtr->nwDestAddr.addressMode == MULTICAST &&
      (tsaReceiveParamPtr->srcAddr.addressMode != MULTICAST_ACK ||
        xmitRecPtr->nwDestAddr.addr.addr1 !=
        tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.group))
  {
      /* MULTICAST message but not MULTICAST_ACK or the ack's group
         # does not match that of xmitRecord. */
      /* ACK does not seem to correspond to what we are expecting. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: TPReceiveAck: Stale ack discarded.");
```

广东省网络空间安全协会受控资料

```
#endif
        INCR_STATS(nmp->stats.lateAcknowledgements);
        return;
    }

    /* For broadcast messages, ACK can come back with subnet value of 0
        from unconfigured nodes. We should accept these acks too. */
    if (xmitRecPtr->nwDestAddr.addressMode == BROADCAST &&
         tsaReceiveParamPtr->srcAddr.subnetAddr.subnet != 0 &&
         xmitRecPtr->nwDestAddr.addr.addr0 != 0 &&
         xmitRecPtr->nwDestAddr.addr.addr0 !=
             tsaReceiveParamPtr->srcAddr.subnetAddr.subnet)
    {
        /* Subnet broadcast but the ACK's subnet does not match . */
        /* ACK does not seem to correspond to what we are expecting. */
        DeQueue(&gp->tsaInQ);
#ifdef DEBUG
        DebugMsg("Debug: TPReceiveAck: Stale ack discarded.");
#endif
        INCR_STATS(nmp->stats.lateAcknowledgements);
        return;
    }

    switch (xmitRecPtr->nwDestAddr.addressMode)
    {
        case BROADCAST:      /* Fall Through. */
        case SUBNET_NODE:    /* Fall Through. */
        case UNIQUE_NODE_ID:
            /* Normally the first ack should have terminated the transaction.
               If it did not, we can try to terminate again. There is no
               harm in doing this. Also, there is no harm in increment
               ackCount as XmitTimerExpiration checks for ackCount >= 1. */
            xmitRecPtr->ackCount++; /* Got one more ack. */
            TerminateTrans(tsaReceiveParamPtr->priority);
          break;
        case MULTICAST:
          /* Group acknowledgement. */
          if (tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member
               > MAX_GROUP_NUMBER)
          {
              ErrorMsg("TPReceiveAck: Invalid group number.");
              break;
          }
```

广东省网络空间安全协会受控资料

```
        if (! xmitRecPtr->ackReceived[
                tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member])
        {
            /* We did not receive this ack in past. */
            xmitRecPtr->ackReceived[
              tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member]
               = TRUE;
            xmitRecPtr->ackCount++;
#ifdef DEBUG
            DebugMsg("TPReceiveAck: A new multicast ACK recvd.");
#endif
        }
        else
        {
            /* Else, it is a duplicate Ack. Ignore it. */
#ifdef DEBUG
            DebugMsg("TPReceiveAck: A Duplicate Mul. ACK ignored");
#endif
        }
        if (xmitRecPtr->destCount == xmitRecPtr->ackCount)
        {
            TerminateTrans(tsaReceiveParamPtr->priority);
        }
        break;
      default:
        nmp->errorLog = BAD_ADDRESS_TYPE;
        ErrorMsg("TPReceiveAck: Invalid address mode");
    }

#ifdef DEBUG
    DebugMsg("TPReceiveAck: Ack was received.");
#endif

    DeQueue(&gp->tsaInQ);

    /* Restart the Xmit Timer if the Xmit record is still active. */
    if (xmitRecPtr->status ! = UNUSED_TX)
    {
        SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);
    }

    return;
}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   SNReceiveResponse
Returns:    None
Reference: None
Purpose:    To process response message received by the session layer.
Comments:   Response should correspond to current transaction in
            progress or else it is thrown away.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static void SNReceiveResponse(void)
{
   TSAReceiveParam  * tsaReceiveParamPtr;
   TSASendParam     * tsaSendParamPtr;
   TSPDUPtr         pduPtr;
   TransmitRecord   * xmitRecPtr;   /* Ptr to xmit rec (pri or nonpri). */
   APPReceiveParam  * appReceiveParamPtr;
   APDU             * apduPtr;

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduPtr             = (TSPDUPtr) (tsaReceiveParamPtr + 1);

   /* Set the corresponding transmit pointer. */
   if (tsaReceiveParamPtr->priority)
   {
      xmitRecPtr       = &gp->priXmitRec;
      tsaSendParamPtr = QueueHead(&gp->tsaOutPriQ);
   }
   else
   {
      xmitRecPtr       = &gp->xmitRec;
      tsaSendParamPtr = QueueHead(&gp->tsaOutQ);
   }

   if (ValidateTrans(tsaReceiveParamPtr->priority, pduPtr->transNum)
       == TRANS_NOT_CURRENT)
   {
      /* Unsolicited response. Ignore it. */
#ifdef DEBUG
      DebugMsg("SNReceiveResponse: Unsolicited response ignored.");
#endif
      DeQueue(&gp->tsaInQ);
      INCR_STATS(nmp->stats.lateAcknowledgements);
```

广东省网络空间安全协会受控资料

```
        return;
    }

    /* Check if possible if the Response really corresponds to the
        transaction in progress. */
    if (xmitRecPtr->status != SESSION_TX ||
         xmitRecPtr->nwDestAddr.domainIndex !=
         tsaReceiveParamPtr->srcAddr.domainIndex)
    {
        /* Transmit record is not ours or the domain indices for
           the response and the transaction record do not match. */
        DeQueue(&gp->tsaInQ);
#ifdef DEBUG
        DebugMsg("Debug: SNReceiveResponse: Stale response discarded.");
#endif
        INCR_STATS(nmp->stats.lateAcknowledgements);
        return;
    }

    if (xmitRecPtr->nwDestAddr.addressMode == SUBNET_NODE &&
         (xmitRecPtr->nwDestAddr.addr.addr2a.subnet !=
          tsaReceiveParamPtr->srcAddr.subnetAddr.subnet ||
          xmitRecPtr->nwDestAddr.addr.addr2a.node !=
          tsaReceiveParamPtr->srcAddr.subnetAddr.node))
    {
        /* Source address of response does not match dest addr of
           transmit record. */
        /* Response does not seem to correspond to what we are expecting. */
        DeQueue(&gp->tsaInQ);
#ifdef DEBUG
        DebugMsg("Debug: SNReceiveResponse: Stale response discarded.");
#endif
        INCR_STATS(nmp->stats.lateAcknowledgements);
        return;
    }

    if (xmitRecPtr->nwDestAddr.addressMode == MULTICAST &&
         (tsaReceiveParamPtr->srcAddr.addressMode != MULTICAST_ACK ||
          xmitRecPtr->nwDestAddr.addr.addr1 !=
          tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.group))
    {
        /* MULTICAST message but not MULTICAST_ACK or the response's group
           # does not match that of xmitRecord. */
```

广东省网络空间安全协会受控资料

```
        /* Response does not seem to correspond to what we are expecting. */
        DeQueue(&gp->tsaInQ);
#ifdef DEBUG
        DebugMsg("Debug: SNReceiveResponse: Stale response discarded.");
#endif
        INCR_STATS(nmp->stats.lateAcknowledgements);
        return;
    }

    /* For brodacast messages, the subnet in the response can be 0 from
        unconfigured nodes. Another possibility is that the response might
        come through a router in a scenerio where the router replaces the
        subnet with its own. Thus the response can be valid even if the
        subnet does not match. So, we should skip this test. */


    /* Check if there is space in application layer's input queue. */
    if (QueueFull(&gp->appInQ))
    {
        return; /* Can't give the response yet. Come back later. */
    }

    appReceiveParamPtr = QueueTail(&gp->appInQ);
    apduPtr            = (APDU *)(appReceiveParamPtr + 1);

    /* Deliver the partial response to application layer. */
    appReceiveParamPtr->indication = MESSAGE;
    appReceiveParamPtr->srcAddr    = tsaReceiveParamPtr->srcAddr;
    appReceiveParamPtr->service    = RESPONSE;
    appReceiveParamPtr->priority   = tsaReceiveParamPtr->priority;
    appReceiveParamPtr->pduSize    = tsaReceiveParamPtr->pduSize - 1;
    appReceiveParamPtr->auth       = FALSE;
    appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
    memcpy(apduPtr, pduPtr->data, tsaReceiveParamPtr->pduSize - 1);
    /* Don't deliver the response to application yet. It could be
        a duplicate. */
    switch (xmitRecPtr->nwDestAddr.addressMode)
    {
        case BROADCAST:
            /* We deliver up to N responses to app layer where N =
                tsaSendParamPtr->destAddr.bcast.maxResponses.
                But we succeed if at least one resp is received. */
            if (xmitRecPtr->ackCount <
```

广东省网络空间安全协会受控资料

```
            tsaSendParamPtr->destAddr.bcast.maxResponses)
        {
            EnQueue(&gp->appInQ);
            INCR_STATS(nmp->stats.layer6_7RespRcvd);
            xmitRecPtr->ackCount++;
            if (xmitRecPtr->ackCount ==
              tsaSendParamPtr->destAddr.bcast.maxResponses)
            {
                TerminateTrans(tsaReceiveParamPtr->priority);
            }
        }
        /* else, we don't want this response. Ignore it. */
        break;
    case SUBNET_NODE:  /* Fall through. */
    case UNIQUE_NODE_ID:
        if (xmitRecPtr->ackCount == 0)
        {
            EnQueue(&gp->appInQ);
            INCR_STATS(nmp->stats.layer6_7RespRcvd);
            xmitRecPtr->ackCount++; /* First response. */
            TerminateTrans(tsaReceiveParamPtr->priority);
        }
        /* else, it is a duplicate response. Ignore it. */
        break;
    case MULTICAST:
        /* Group request message. Check response address mode. */
        if (tsaReceiveParamPtr->srcAddr.addressMode
            != MULTICAST_ACK)
        {
            ErrorMsg("SNReceiveResponse: RESPONSE should be "
                     "MULTICAST_ACK.");
            break;
        }
        if (tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member
            > MAX_GROUP_NUMBER)
        {
            ErrorMsg("SNReceiveResponse: Invalid group number.");
            break;
        }
        if (! xmitRecPtr->ackReceived[
                tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member])
        {
#ifdef DEBUG
```

广东省网络空间安全协会受控资料

```
            DebugMsg("SNReceiveResp: A multicast resp delivered.");
#endif
            EnQueue(&gp->appInQ);
            INCR_STATS(nmp->stats.layer6_7RespRcvd);
            xmitRecPtr->ackReceived[
              tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member]
               = TRUE;
            xmitRecPtr->ackCount++;
        }
        else
        {
            /* Else, it is a duplicate response. Ignore it */
#ifdef DEBUG
            DebugMsg("SNReceiveResp: A duplicate multicast response ignored.");
#endif
        }
        if (xmitRecPtr->destCount == xmitRecPtr->ackCount)
        {
            TerminateTrans(tsaReceiveParamPtr->priority);
        }
        break;
    default:
        nmp->errorLog = BAD_ADDRESS_TYPE;
        ErrorMsg("SNReceiveResponse: Invalid address mode.");
    }

    DeQueue(&gp->tsaInQ);

    /* Restart the timer if the xmit rec is still active. */
    if(xmitRecPtr->status != UNUSED_TX)
    {
        SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);
    }

    return;
}

/*****************************************************************
Function:   ReceiveNewMsg
Returns:    None
Reference:  None
Purpose:    To process either a new message received by transport
            (ACKD or UNACK_RPT) or session layer (REQUEST).
```

广东省网络空间安全协会受控资料

```
Comments:  The message could be new or a duplicate. If new, we
           need to allocate a RR. If not, we use the existing RR.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static void ReceiveNewMsg(Layer layerIn)
{
   TSAReceiveParam  * tsaReceiveParamPtr;
   TSPDUPtr           pduPtr;
   uint16            recvTimerValue;
   int16             i;
   Boolean           initRR; /* Used to check if RR needs to init. */

   /* We have a new msg in TSA In queue. */

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduPtr             = (TSPDUPtr) (tsaReceiveParamPtr + 1);

   /* First retrieve associated RR, if it exists. */
   /* Since we can receive at most one message from a node
      (one for priority & one for non-priority), we can reuse
      receive records if the sender determines that a message
      transmission is over by transmitting a new message with a
      new transaction number or different service (i.e layer) or
      a new APDU. */

   i = RetrieveRR(tsaReceiveParamPtr->srcAddr,
                  tsaReceiveParamPtr->priority);
   if (i == -1)
   {
      /* No Associated RR found. We need to allocate a new one. */
      i = AllocateRR();
      if (i == -1)
      {
         /* Unable to allocate a new RR. */
         INCR_STATS(nmp->stats.receiveTXFull);
#ifdef DEBUG
         DebugMsg("ReceiveNewMsg: Unable to allocate RR. Msg is lost.");
#endif
         DeQueue(&gp->tsaInQ); /* Remove item from queue. */
         return;
      }
      /* We did allocate a new RR. We need to init this RR. */
      initRR = TRUE;
   }
```

广东省网络空间安全协会受控资料

```
else if (gp->recvRec[i].transNum ! = pduPtr->transNum ||
      gp->recvRec[i].status    ! = (layerIn == TRANSPORT? TRANSPORT_RR:SESSION_RR)
                                    || tsaReceiveParamPtr->pduSize - 1 ! =
                                    gp->recvRec[i].apduSize   ||
    /* * * * START INFORMATIVE - APDU Contents and Duplicate Detection * * *  */
     /* This check causes the contents of the APDU to always be factored into
       * the duplicate detection algorithm.
      * It is considered a requirement that the APDU contents be
      * factored into the duplicate detection algorithm only in the case where the
      * message is authenticated and the request is idempotent (because retries are
      * not re-authenticated). Whether this is done by storing the entire APDU
      * for comparison or storing only the APDU length and a checksum of the APDU
      * data is implementation dependent. If a checksum method is used, the
      * checksum should be at least 24 bits in length. In the case where a
      * non-matching APDU vectors into an existing authenticated transaction, two
      * choices are possible:
      * a. Treat the message as if it were not a duplicate (as is done here).
      * b. Treat the message as a duplicate but mark it as not authenticated (if
      *     it is necessary to deliver the retry to the application - see
      *     "START INFORMATIVE - Saved Response Length" below).  */
        memcmp(pduPtr->data, gp->recvRec[i].apdu,
               gp->recvRec[i].apduSize) ! = 0)
        /* * * * END INFORMATIVE - APDU Contents and Duplicate Detection * * *  */
{
    /* Something does not match. Must be a new message from
       the sender. Let us reuse this RR, even if it is a
       request. If it is a request and response comes later,
       it won't match this record anyway due to reqid.  */
    initRR = TRUE;
    /* If the old message was not deliverd, increment lost msg stat */
    if (gp->recvRec[i].transState ! = DELIVERED &&
        gp->recvRec[i].transState ! = DONE        &&
        gp->recvRec[i].transState ! = RESPONDED)
    {
       INCR_STATS(nmp->stats.lostMessages);
    }
}
else
{
    /* It is an existing RR with everything matching.  */
    /* Nothing to do. We use the information in existing RR.  */
    /* Set the alternate path bit based on new message so that
       acks, challenge etc will use the same path. Change only if
```

广东省网络空间安全协会受控资料

```
        it is not already using alternate path.  */
    if (! gp->recvRec[i].altPath)
    {
        gp->recvRec[i].altPath = tsaReceiveParamPtr->altPath;
    }
    initRR = FALSE;
}

if (initRR)
{
    if (layerIn == TRANSPORT)
    {
        gp->recvRec[i].status          = TRANSPORT_RR;
    }
    else
    {
        gp->recvRec[i].status          = SESSION_RR;
    }
    gp->recvRec[i].srcAddr          = tsaReceiveParamPtr->srcAddr;
    gp->recvRec[i].transNum         = pduPtr->transNum;
    gp->recvRec[i].transState       = JUST_RECEIVED;
    gp->recvRec[i].priority         = tsaReceiveParamPtr->priority;
    gp->recvRec[i].altPath          = tsaReceiveParamPtr->altPath;
    gp->recvRec[i].auth             = FALSE;
    gp->recvRec[i].reqId            = 0; /* Init to invalid reqid.  */
    if (layerIn == TRANSPORT)
    {
        gp->recvRec[i].serviceType      =
            pduPtr->pduMsgType == ACKD_MSG? ACKD: UNACK_RPT;
    }
    else
    {
        gp->recvRec[i].serviceType      = REQUEST;
        /* reqId can wrap around. Never use 0 as reqId so that
            a valid reqId will never match a receive record that
            does not correspond to corresponding request.  */
        if (gp->reqId == 0)
        {
            gp->reqId++;
        }
        gp->recvRec[i].reqId            = gp->reqId++;
    }
    gp->recvRec[i].apduSize = tsaReceiveParamPtr->pduSize - 1;
```

广东省网络空间安全协会受控资料

```
        memcpy(gp->recvRec[i].apdu,
                pduPtr->data,
                gp->recvRec[i].apduSize); /* Store the APDU. */
        /* Compute the recvTimer value to be used. */
        recvTimerValue = ComputeRecvTimerValue(
                            tsaReceiveParamPtr->srcAddr.addressMode,
                            tsaReceiveParamPtr->srcAddr.group);
        SetMsTimer(&gp->recvRec[i].recvTimer, recvTimerValue);
    }

    DeQueue(&gp->tsaInQ); /* Remove item from queue. */

    /* Now, we have a RR for this message. */
    /* Determine if the msg needs authentication and store. */
    /* Allow authentication for UnAck_Rpt. */
    /* flexDomain messages cannot be authenticated. Force it. */

    if (pduPtr->auth && tsaReceiveParamPtr->srcAddr.domainIndex != FLEX_DOMAIN)
    {
        gp->recvRec[i].needAuth = TRUE;
    }
    else
    {
        gp->recvRec[i].needAuth = FALSE;
    }

    /* Authentication is performed if it is a new message or
        in the process of authenticating and it needs to be
        autheneticated. */
    if ((gp->recvRec[i].transState == JUST_RECEIVED ||
          gp->recvRec[i].transState == AUTHENTICATING) &&
         gp->recvRec[i].needAuth)
    {
        /* The message needs authentication. Initiate Challenge. */
        /* Or ReInitiate Challenge as the prev one is probably lost. */
#ifdef DEBUG
        DebugMsg("ReceiveNewMsg: Initiating Challenge.");
#endif
        InitiateChallenge(i);
        return;
    }

    if (gp->recvRec[i].transState != DELIVERED &&
```

广东省网络空间安全协会受控资料

```
        gp->recvRec[i].transState ! = RESPONDED &&
        gp->recvRec[i].transState ! = DONE)
    {
        /* Deliver the message to the application layer. */
        Deliver(i);
    }
    else
    {
#ifdef DEBUG
        DebugMsg("ReceiveNewMsg: Msg received was already delivered.");
#endif
    }

    if (layerIn == TRANSPORT)
    {
        if (gp->recvRec[i].transState   == DELIVERED &&
            gp->recvRec[i].serviceType == ACKD)
        {
            /* Compose and send the acknowledgement. */
            TPSendAck(i);
        }
    }
    else if (gp->recvRec[i].transState == RESPONDED)
    {
        /* Must be a request that was already responded. */
        /* We already have a resp. Simply send it. */
        SNSendResponse(i, FALSE); /* It can't be a null response. */
    }
    else if (gp->recvRec[i].transState == DONE)
    {
#ifdef DEBUG
        DebugMsg("ReceiveNewMsg: Nothing to do.");
#endif
    }
    return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   ReceiveRem
Returns:    None
Reference:  None
Purpose:    To process REMINDER and REM/MSG messages.
            We either do nothing or send an ack (for Transport)
```

广东省网络空间安全协会受控资料

```
            or a response(for Session) if possible.
Comments:   None
*****************************************************************/
static void ReceiveRem(Layer layerIn)
{
   TSAReceiveParam  * tsaReceiveParamPtr;
   TSPDUPtr           pduPtr;
   int16              i;
   APDU               * apduPtr;
   uint16             apduSize;
   uint8              length;
   uint8              member;
   uint8              mlistIndex, mlistOffset;
   Byte               * mlistPtr;
   uint16             recvTimerValue;

   /* We have a REMINDER or REM/MSG in TSA input queue. */

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduPtr             = (TSPDUPtr) (tsaReceiveParamPtr + 1);
   length             = pduPtr->data[0];

   if (pduPtr->pduMsgType == REMINDER_MSG)
   {
      apduSize = 0;
      apduPtr = NULL;
   }
   else
   {
      /* Must be REM_MSG_MSG. */
      /* TSPDU header size before APDU is actual 1 byte header,
         1 byte length field and the length of M_List field. */
      apduSize = tsaReceiveParamPtr->pduSize - 2 - length;
      apduPtr  = (APDU *)&pduPtr->data[1 + length];
   }

   /* First retrieve the associated RR, if it exists. */
   i = RetrieveRR(tsaReceiveParamPtr->srcAddr,
                  tsaReceiveParamPtr->priority);


   /* Check if the REMINDER msg has an associated RR. */
   if (i == -1 && pduPtr->pduMsgType == REMINDER_MSG)
```

广东省网络空间安全协会受控资料

```
    {
        /* No associated RR. Discard this REMINDER msg. */
#ifdef DEBUG
        DebugMsg("ReceiveRem: REMINDER msg has no associated RR");
#endif
        DeQueue(&gp->tsaInQ);
        return;
    }

    if (i != -1 &&
        (gp->recvRec[i].srcAddr.addressMode != MULTICAST ||
         gp->recvRec[i].transNum != pduPtr->transNum ||
         gp->recvRec[i].status != (layerIn == TRANSPORT? TRANSPORT_RR:SESSION_RR) ||
         (apduPtr && apduSize != gp->recvRec[i].apduSize) ||
         (apduPtr &&
            /* *** START INFORMATIVE — APDU Contents and Duplicate Detection *** */
            /* See "START INFORMATIVE — APDU Contents and Duplicate Detection" above. */
            memcmp(apduPtr, gp->recvRec[i].apdu, apduSize) != 0))
            /* *** END INFORMATIVE — APDU Contents and Duplicate Detection *** */
        )

    {
        /* Associate RR does not match properly. Either it is not
           a multicast message or transNum does not match or
           APDU does not match. This is not OK for REMINDER_MSG.
           If it is however REM_MSG_MSG, we want to treat it as a
           new message. */
        if (pduPtr->pduMsgType == REMINDER_MSG)
        {
#ifdef DEBUG
            DebugMsg("ReceiveRem: Associated RR does not match.");
#endif
            DeQueue(&gp->tsaInQ);
            return;
        }
        if (gp->recvRec[i].transState == DELIVERED ||
            gp->recvRec[i].transState == DONE)
        {
            /* Reuse this receive record. We will free this record and allocate
               a new one. */
            gp->recvRec[i].status = UNUSED_RR;
            i = -1;
        }
```

广东省网络空间安全协会受控资料

```
        else
        {
            i = -1; /* Allocate new one. */
        }
    }

    if (i != -1 && gp->recvRec[i].serviceType == UNACK_RPT)
    {
        if (pduPtr->pduMsgType == REMINDER_MSG)
        {
            /* We should not have gotton this REMINDER as the original
               message was UNACK_RPT. So, ignore this message. */
#ifdef DEBUG
            DebugMsg("ReceiveRem: Original msg is UNACK_RPT.");
#endif
            DeQueue(&gp->tsaInQ);
            return;
        }
        /* Treat REM_MSG_MSG as new one. Fack this by setting i to -1. */
        i = -1;
    }

    if (i != -1)
    {
        /* We have already received the corresponding APDU. */
        /* Probably the ack or response sent earlier was lost. */
        /* Sent the ack or response again. */
        if (! IsGroupMember(gp->recvRec[i].srcAddr.domainIndex,
                            gp->recvRec[i].srcAddr.group,
                            &member))
        {
            /* We are not member of the group used for this msg.
               Strange! Ignore this msg too!
               Network layer should not have delivered such msg
               to upper layers. */
#ifdef DEBUG
            DebugMsg("ReceiveRem: Strange! Not a group member!!.");
#endif
            DeQueue(&gp->tsaInQ);
            return;
        }
        mlistIndex  = member / 8;
        mlistOffset = member % 8;
```

广东省网络空间安全协会受控资料

```
    mlistPtr      = &pduPtr->data[1];
    if (
        length == 0
        ||
        mlistIndex >= length
        ||
        (mlistPtr[mlistIndex] & (1 << mlistOffset)) == 0
      )
    {
      /* We are asked to respond. Server did not get our ack or response. */

      /* If it is a REMINDER_MSG, it will be followed by the retry.
         So, we simply ignore it. */
      if (pduPtr->pduMsgType == REMINDER_MSG)
      {
         DeQueue(&gp->tsaInQ);
         return;
      }

      if (gp->recvRec[i].needAuth &&
            (gp->recvRec[i].transState == JUST_RECEIVED ||
             gp->recvRec[i].transState == AUTHENTICATING)
          )
      {
         /* Need authentication. Either a new msg or prev challenge
            was lost. */
#ifdef DEBUG
         DebugMsg("ReceiveRem: Initiating Challenge.");
#endif
         InitiateChallenge(i);
         DeQueue(&gp->tsaInQ);
         return;
      }
      if (gp->recvRec[i].transState ! = DELIVERED &&
          gp->recvRec[i].transState ! = RESPONDED &&
          gp->recvRec[i].transState ! = DONE)
      {
         /* Deliver the message to application layer. */
         Deliver(i);
      }

      /* Send ack or response. */
      if (gp->recvRec[i].transState   == DELIVERED ||
```

广东省网络空间安全协会受控资料

```
            gp->recvRec[i].transState   == RESPONDED)
        {
            DeQueue(&gp->tsaInQ);
            if (layerIn == TRANSPORT)
            {
                /* Compose and send the acknowledgement. */
                TPSendAck(i);
            }
            else if (gp->recvRec[i].transState == RESPONDED)
            {
                SNSendResponse(i, FALSE); /* can't be null resp. */
            }
            else
            {
                /* we are still waiting for a response. */
#ifdef DEBUG
                DebugMsg("ReceiveRem: Don't have response yet"
                         " or have null response.");
#endif
            }
        }
        else
        {
            /* We are still waiting for resp & so can't respond
               or in DONE state indicating server already received
               the ack or response. */
#ifdef DEBUG
            DebugMsg("ReceiveRem: Can't respond or no need to respond.");
#endif
            DeQueue(&gp->tsaInQ);
        }
    }
    else
    {
        /* We are not asked to acknowledege or respond. Ignore msg. */
        if (gp->recvRec[i].transState == DELIVERED ||
            gp->recvRec[i].transState == RESPONDED)
        {
            /* Note down the fact that the ack or resp has been received */
            gp->recvRec[i].transState = DONE;
        }
#ifdef DEBUG
        DebugMsg("ReceiveRem: We are not asked to ack or respond.");
```

广东省网络空间安全协会受控资料

```
#endif
        DeQueue(&gp->tsaInQ);
    }
    return;
}

/* Now we must have a REM_MSG_MSG with no associated RR. */
/* REMINDER msg with no associated record was taken care of
   earlier. */
i = AllocateRR();
if (i == -1)
{
    /* Unable to allocate a new RR. Give up. */
    return;
}
if (layerIn == TRANSPORT)
{
    gp->recvRec[i].status          = TRANSPORT_RR;
}
else
{
    gp->recvRec[i].status          = SESSION_RR;
}
gp->recvRec[i].srcAddr          = tsaReceiveParamPtr->srcAddr;
gp->recvRec[i].transNum         = pduPtr->transNum;
gp->recvRec[i].transState       = JUST_RECEIVED;
gp->recvRec[i].priority         = tsaReceiveParamPtr->priority;
gp->recvRec[i].altPath          = tsaReceiveParamPtr->altPath;
gp->recvRec[i].auth             = FALSE; /* Not authenticated yet. */
if (layerIn == TRANSPORT)
{
    /* We have REM/MSG. So service type must be ACKD. */
    gp->recvRec[i].serviceType      = ACKD;
}
else
{
    gp->recvRec[i].serviceType      = REQUEST;
    /* reqId can wrap around. Never use 0 as reqId so that
       a valid reqId will never match a receive record that
       does not correspond to corresponding request. */
    if (gp->reqId == 0)
    {
        gp->reqId++;
```

广东省网络空间安全协会受控资料

```
        }
        gp->recvRec[i].reqId              = gp->reqId++;
    }
    gp->recvRec[i].apduSize = apduSize;
    memcpy(gp->recvRec[i].apdu, apduPtr, apduSize);
    /* Compute the recvTimer value to be used. */
    recvTimerValue =
        ComputeRecvTimerValue(
                                tsaReceiveParamPtr->srcAddr.addressMode,
                                tsaReceiveParamPtr->srcAddr.group
                              );
    DeQueue(&gp->tsaInQ); /* Remove the item from queue. */

    SetMsTimer(&gp->recvRec[i].recvTimer, recvTimerValue);

    /* Flex domain messages cannot be authenticated. */
    if (pduPtr->auth &&
         tsaReceiveParamPtr->srcAddr.domainIndex ! = FLEX_DOMAIN)
    {
        gp->recvRec[i].needAuth = TRUE; /* Remember that we need auth. */
    }
    else
    {
        gp->recvRec[i].needAuth = FALSE;
    }

    /* Now, we have a RR for this message. */
    if (gp->recvRec[i].needAuth)
    {
        /* The message needs authentication. Initiate Challenge. */
#ifdef DEBUG
        DebugMsg("ReceiveRem: Initiating Challenge.");
#endif
        InitiateChallenge(i);
        return;
    }

    /* Deliver the message to the application layer. */
    Deliver(i);

    if (layerIn == TRANSPORT)
    {
        if (gp->recvRec[i].transState   == DELIVERED)
```

广东省网络空间安全协会受控资料

```
        {
            /* Compose and send the acknowledgement. */
            TPSendAck(i);
        }
    }
    else
    {
        /* We just delivered. Can't have a response yet. */
    }
    return;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   TPSendAck
Returns:    None
Reference:  None
Purpose:    To Send an ACK message for ACKD msg received by the
            transport layer. If there is no space in network
            output queue, then nothing is sent.
Comments:   None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static void TPSendAck(uint16 rrIndexIn)
{
    NWSendParam        * nwSendParamPtr;    /* Ptr to NW Send Param. */
    TSPDUPtr             pduPtr;            /* Ptr to TPDU sent.     */
    Queue              * nwQueuePtr;
    DestinationAddress   destAddr;

    if (gp->recvRec[rrIndexIn].priority)
    {
        nwQueuePtr = &gp->nwOutPriQ;
    }
    else
    {
        nwQueuePtr = &gp->nwOutQ;
    }

    if (QueueFull(nwQueuePtr))
    {
        return; /* Can't send the acknowledgment now. */
    }

    nwSendParamPtr = QueueTail(nwQueuePtr);
```

广东省网络空间安全协会受控资料

```
pduPtr          = (TSPDUPtr) (nwSendParamPtr + 1);

/* Form the TPDU directly in the network layer's queue. */
pduPtr->auth       = FALSE; /* Acks are not authenticated. */
pduPtr->pduMsgType   = ACK_MSG;
pduPtr->transNum = gp->recvRec[rrIndexIn].transNum;

/* Fill in the details for the network layer. */

/* First, form sender's (i.e our) address. */
destAddr.domainIndex = gp->recvRec[rrIndexIn].srcAddr.domainIndex;
if (destAddr.domainIndex == FLEX_DOMAIN)
{
    /* Flex domain was used. Copy flex domain information. */
    destAddr.flexDomainLen =
       gp->recvRec[rrIndexIn].srcAddr.flexDomainLen;
    /* Make sure we have a good length field. */
    if (destAddr.flexDomainLen <= DOMAIN_ID_LEN)
    {
       memcpy(destAddr.flexDomainId,
              gp->recvRec[rrIndexIn].srcAddr.flexDomainId,
              destAddr.flexDomainLen);
    }
}

/* We send unicast ACK or multicast ACK depending on msg recvd. */
if (gp->recvRec[rrIndexIn].srcAddr.addressMode == MULTICAST)
{
    /* Send ACK in address format 2b. */
    destAddr.addressMode = MULTICAST_ACK;
    destAddr.addr.addr2b.subnetAddr =
       gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
    destAddr.addr.addr2b.groupAddr.group   =
       gp->recvRec[rrIndexIn].srcAddr.group;
    if (! IsGroupMember(destAddr.domainIndex, destAddr.addr.addr2b.groupAddr.group,
                        &destAddr.addr.addr2b.groupAddr.member))
    {
        ErrorMsg("SendACKTPDU: Ack msg for a non-existing group.");
        return;
    }
}
else
{
```

广东省网络空间安全协会受控资料

```
        /* Send Ack in address format 2a. */
        destAddr.addressMode = SUBNET_NODE;
        destAddr.addr.addr2a =
            gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
    }

    nwSendParamPtr->dropIfUnconfigured = FALSE; /* acks are not dropped */
    nwSendParamPtr->destAddr = destAddr;
    nwSendParamPtr->pduType  = TPDU_TYPE;
    nwSendParamPtr->deltaBL  = 0;
    nwSendParamPtr->altPath  = gp->recvRec[rrIndexIn].altPath;
    nwSendParamPtr->pduSize  = 1;

#ifdef DEBUG
    DebugMsg("Debug: TPSendAck. Sending an ACK.");
#endif
    EnQueue(nwQueuePtr);
}

/*****************************************************************
Function:   SNSendResponse
Returns:    None
Reference:  None
Purpose:    To send a response message for REQUEST msg received by the
            session layer. If there is no space in network
            output queue, then nothing is sent.
            If it is a null response, then the receive record is freed
            and no response goes out.
Comments:   This function is called only when the application layer
            has already given the response. (Or else how can this
            function know about how to respond anyway?)
*****************************************************************/
static void SNSendResponse(uint16 rrIndexIn, Boolean nullResponse)
{
    NWSendParam         *nwSendParamPtr;    /* Ptr to NW Send Param. */
    TSPDUPtr            pduPtr;             /* Ptr to TPDU sent.     */
    Queue               *nwQueuePtr;
    DestinationAddress destAddr;

    if (gp->recvRec[rrIndexIn].priority)
    {
        nwQueuePtr = &gp->nwOutPriQ;
    }
```

广东省网络空间安全协会受控资料

```
    else
    {
        nwQueuePtr = &gp->nwOutQ;
    }

    if (QueueFull(nwQueuePtr) && ! nullResponse)
    {
        return; /* Can't sent the response now. */
    }

    if (gp->recvRec[rrIndexIn].transState ! = RESPONDED &&
         gp->recvRec[rrIndexIn].transState ! = DONE)
    {
        ErrorMsg("SNSendResponse: How can I be called without "
                 " atleast one resp?");
        return;
    }

    if (nullResponse)
    {
        /* Set state to DONE and do not send the response. */
#ifdef DEBUG
        DebugMsg("SendResponse: Null response. Nothing goes out.");
#endif
        gp->recvRec[rrIndexIn].transState = DONE;
        return;
    }

    nwSendParamPtr = QueueTail(nwQueuePtr);
    pduPtr         = (TSPDUPtr) (nwSendParamPtr + 1);

    if (gp->nwOutBufSize < gp->recvRec[rrIndexIn].rspSize + 1)
    {
        ErrorMsg("SNSendResponse: Network buf too small for response.");
        return;
    }
    pduPtr->auth       = FALSE; /* Responses are not authenticated. */
    pduPtr->pduMsgType   = RESPONSE_MSG;
    pduPtr->transNum = gp->recvRec[rrIndexIn].transNum;
    /* Copy the existing response from RR */
    if (gp->recvRec[rrIndexIn].rspSize <=
          DecodeBufferSize((uint8) eep->readOnlyData.appOutBufSize))
    {
```

广东省网络空间安全协会受控资料

```
    /* *** START INFORMATIVE — Saved Response Length *** */
/* This implementation saves the entire response in the receive transaction record.
 * Thus, if and when a retry of the request is received, the response can be
 * re-transmitted without re-delivering the request to the application for
 * construction of a response.  It is acceptable to save only responses of certain
 * length (e.g., as little as 1 byte) in order to minimize RAM use.
 * Under these conditions, a retry of a request that illicited a response of length
 * greater than that minimium length is considered to be idempotent (can be safely
 * executed again) and may be re-delivered to the application for reformulation of the
 * response.  It is also required that under these conditions that the application be
 * informed of the duplicate nature of the request in case the application chooses to
 * treat the request as non-idempotent by saving the response for retransmission
 * without recomputation.  In this case a key must also be provided for the
 * application to use to map the request to a saved response. */
        memcpy(pduPtr->data, gp->recvRec[rrIndexIn].response,
                gp->recvRec[rrIndexIn].rspSize);
    /* *** END INFORMATIVE — Saved Response Length *** */
    }
    else
    {
#ifdef DEBUG
        DebugMsg("SNSendResponse: Discarding a long response.");
#endif
        return;
    }
    /* Fill in the details for the network layer. */

    destAddr.domainIndex = gp->recvRec[rrIndexIn].srcAddr.domainIndex;
    if (destAddr.domainIndex == FLEX_DOMAIN)
    {
        /* Flex domain was used. Copy flex domain information. */
        destAddr.flexDomainLen =
            gp->recvRec[rrIndexIn].srcAddr.flexDomainLen;
        if (destAddr.flexDomainLen <= DOMAIN_ID_LEN)
        {
            memcpy(destAddr.flexDomainId,
                    gp->recvRec[rrIndexIn].srcAddr.flexDomainId,
                    destAddr.flexDomainLen);
        }
    }

    /* We send unicast response or multicast response depending on
        the request received. */
```

广东省网络空间安全协会受控资料

```
    if (gp->recvRec[rrIndexIn].srcAddr.addressMode == MULTICAST)
    {
        /* Send the response in address format 2b. */
        destAddr.addressMode = MULTICAST_ACK;
        destAddr.addr.addr2b.subnetAddr =
            gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
        destAddr.addr.addr2b.groupAddr.group   =
            gp->recvRec[rrIndexIn].srcAddr.group;
        if (! IsGroupMember(destAddr.domainIndex, destAddr.addr.addr2b.groupAddr.group,
                            &destAddr.addr.addr2b.groupAddr.member))
        {
            ErrorMsg("SNSendResponse: Response for a non-existing group.");
            return;
        }
    }
    else
    {
        /* Send response in address format 2a. */
        destAddr.addressMode = SUBNET_NODE;
        destAddr.addr.addr2a =
            gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
    }

    nwSendParamPtr->dropIfUnconfigured = FALSE; /* responses are not dropped */
    nwSendParamPtr->destAddr = destAddr;
    nwSendParamPtr->pduType   = SPDU_TYPE;
    nwSendParamPtr->deltaBL   = 0;
    nwSendParamPtr->altPath   = gp->recvRec[rrIndexIn].altPath;
    nwSendParamPtr->pduSize   = gp->recvRec[rrIndexIn].rspSize + 1;

#ifdef DEBUG
    DebugMsg("SNSendResponse: Sending a response.");
#endif
    EnQueue(nwQueuePtr);
}

/*****************************************************************
Function:   Deliver
Returns:    None
Reference:  None
Purpose:    To deliver a message received by tranport or session layer
            to the application layer.
Comments:   layerIn is not needed as it is not used.
```

广东省网络空间安全协会受控资料

```
*************************************/
static void Deliver(uint16 rrIndexIn)
{
   APPReceiveParam      *appReceiveParamPtr;
   char                 *apduInPtr;
   uint16                i = rrIndexIn; /* Use i instead of rrIndexIn. */

   if (gp->recvRec[i].needAuth && gp->recvRec[i].transState != AUTHENTICATED)
   {
      /* The message needs authentication but was not authenticated. */
      gp->recvRec[i].transState = DONE;
      return;
   }

   if (QueueFull(&gp->appInQ))
   {
      /* We can wait, but then we may not be able to guarantee delivery in
         sequence from a given source node. So, it is better to drop the
         message and let the retry mechanism take care of redelivery. */
      INCR_STATS(nmp->stats.lostMessages);
      gp->recvRec[i].transState = DONE; /* indicate that this one is done */
      return;
   }

   appReceiveParamPtr = QueueTail(&gp->appInQ);
   apduInPtr          = (char *)(appReceiveParamPtr + 1);

   appReceiveParamPtr->indication = MESSAGE;
   appReceiveParamPtr->srcAddr     = gp->recvRec[i].srcAddr;
   appReceiveParamPtr->service     = gp->recvRec[i].serviceType;
   appReceiveParamPtr->priority    = gp->recvRec[i].priority;
   appReceiveParamPtr->altPath     = gp->recvRec[i].altPath;
   appReceiveParamPtr->auth        = gp->recvRec[i].auth;
   appReceiveParamPtr->pduSize     = gp->recvRec[i].apduSize;
   appReceiveParamPtr->reqId       = gp->recvRec[i].reqId;

   if (gp->appInBufSize < gp->recvRec[i].apduSize)
   {
      ErrorMsg("Deliver: APDU size too big");
      /* We can never deliver this APDU. So, make it look like
         it was delivered so that it can be discarded. We don't
         want to send any ACK or response for this message. */
      gp->recvRec[i].transState       = DONE;
```

广东省网络空间安全协会受控资料

```
        nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
        return;
    }
    /* Now it should be safe to do memcpy. */
    memcpy(apduInPtr, gp->recvRec[i].apdu, gp->recvRec[i].apduSize);
    EnQueue(&gp->appInQ);
    INCR_STATS(nmp->stats.layer6_7MsgsRcvd);
    gp->recvRec[i].transState        = DELIVERED;
#ifdef DEBUG
    DebugMsg("Debug: Deliver: Packet has been delivered"
             " to the application layer.");
#endif
}

/*****************************************************************
Function:   RetrieveRR
Returns:    Index of RR Table that matches given input parameters.
            -1 if none exists.
Reference: None
Purpose:    To retrieve the receive record from the RR table
            that corresponds to a message received. For matching,
            we use priority, domainIndex, addressMode, and source address.
            We also match subnet if the message is broadcast
            or group if the message is multicast.
Comments:   None
*****************************************************************/
static int16 RetrieveRR(SourceAddress srcAddrIn,
                        Boolean priorityIn)
{
    int16 i;

    /* Search through all the receive records for a match. */

    for (i = 0; i < gp->recvRecCnt; i++)
    {
        if (
            priorityIn == gp->recvRec[i].priority
            &&
            /* Destination subnet/node match is based on domainIndex. */
            srcAddrIn.domainIndex == gp->recvRec[i].srcAddr.domainIndex
            &&
            (srcAddrIn.addressMode == gp->recvRec[i].srcAddr.addressMode)
            &&
```

广东省网络空间安全协会受控资料

```
            /* Source node address should always match. */
            (memcmp(&srcAddrIn.subnetAddr,
                    &gp->recvRec[i].srcAddr.subnetAddr,
                    sizeof(SubnetAddress)) == 0)
            &&
            /* Make sure BROADCAST address matches for broadcast messages. */
            ( srcAddrIn.addressMode ! = BROADCAST ||
              srcAddrIn.broadcastSubnet == gp->recvRec[i].srcAddr.broadcastSubnet )
            &&
            /* Make sure MULTICAST address matches for multicast messages. */
             ( srcAddrIn.addressMode ! = MULTICAST ||
               srcAddrIn.group == gp->recvRec[i].srcAddr.group )
          )

        {
          break;
        }
    }

    if (i == gp->recvRecCnt)
    {
        return(-1); /* Matching RR was not found. */
    }

    return(i); /* Found matching RR. */
}


/*****************************************************************
Function:   AllocateRR
Returns:    Index of RR table that can be used for a new msg.
Reference: None
Purpose:    To find an index in the RR table that is UNUSED.
Comments:   None
*****************************************************************/
static int16 AllocateRR(void)
{
    int16 i;

    /* First search for one that is unused. */
    for (i = 0; i < gp->recvRecCnt; i++)
    {
        if (gp->recvRec[i].status == UNUSED_RR)
```

广东省网络空间安全协会受控资料

```
        {
            return(i); /* Found one that is unused. */
        }
    }

    return(-1);
}

/*****************************************************************
Function:   ComputeRecvTimerValue
Returns:    Receive Timer value in milli seconds.
Reference:  None
Purpose:    To compute the value to be used for receive timer
            based on the address format of the received message
            and group id.
Comments:   None
*****************************************************************/
static uint16 ComputeRecvTimerValue(AddrMode        addrModeIn,
                                    MulticastAddress groupIdIn)
{
    uint16 i, max = 0, temp;

    if (addrModeIn == UNIQUE_NODE_ID)
    {
        return((uint16)(NGTIMER_SPCL_VAL * 1000.0));
    }
    if (addrModeIn == MULTICAST)
    {
        /* Search through the address table to find the receiver timer val. */
        /* If there is more than one entry with the same group,
           use the one with the max rcv timer value */
        for (i = 0; i < NUM_ADDR_TBL_ENTRIES; i++)
        {
            if (eep->addrTable[i].addrFormat >= 128 &&
                eep->addrTable[i].groupEntry.groupID == groupIdIn)
            {
                /* Group format match */
                temp = DecodeRcvTimer((uint8)
                            eep->addrTable[i].groupEntry.rcvTimer);
                /* *** START INFORMATIVE - Multicast Receive Timer *** */
                /* Using the maximum receive timer for the group is not required.  It
                 * is acceptable to use the receive timer for the first group entry
                 * found in the table. */
```

广东省网络空间安全协会受控资料

```
            if (temp > max)
            {
               max = temp;
            }
            /* *** END INFORMATIVE — Multicast Receive Timer *** */
         }
      }
      return(max);
   }
   /* All other messages use non-group timer value. */
   return(DecodeRcvTimer((uint8) eep->configData.nonGroupTimer));
}

/*****************************************************************
Function:  SNSend
Returns:   None
Reference: Section 8, Protocol Spec.
Purpose:   To implement Send algorithm for the session layer.
           If there is anything to be sent by the session layer,
           this function will process that message and sends it.
           If there is any ongoing message, it might need some
           processing such as retry, timer expiry etc. This
           function will process such events too.
Comments:  Update the priority transmit timer, if it exists.
           Update the non-priority transmit timer, if it exists.
           If the priority transmit timer expired then
              process this event.
           else if there is priority message to be sent and there is space
                 in priority queue of the network layer then
              process the priority message.
           else if the non-priority transmit timer expired then
              process that event.
           else if there is non-priority message to be sent and there is space
                 in non-priority queue of network layer then
              process the non-priority message.
           else
                 nothing to do. return.
Note:
*****************************************************************/
void SNSend(void)
{
   TSASendParam      *tsaSendParamPtr;
   uint16            i;
```

广东省网络空间安全协会受控资料

```
/* Delay SNSend after power-up or external reset. */
if ( gp->tsDelayTimer.curTimerValue > 0 &&
      (nmp->resetCause == POWER_UP_RESET ||
       nmp->resetCause == EXTERNAL_RESET)
    )
{
    UpdateMsTimer(&gp->tsDelayTimer);
    return; /* Do nothing */
}

/*****************************************************************
  Send response, if any, first. Responses are not like transactions
  and hence it is better to send it first.
*****************************************************************/
if (! QueueEmpty(&gp->tsaRespQ) &&
        ! QueueFull(&gp->nwOutQ))
{
    /* We have a response to be sent out.
       Make sure the response is not stale.
       Response should have a reqId.
       If there is no RR for this reqId, then it is stale.
       Throw the response away, if it is stale. */
    tsaSendParamPtr = QueueHead(&gp->tsaRespQ);

    if (tsaSendParamPtr->service ! = RESPONSE)
    {
       /* Throw away this message. Only responses are
          allowed in this queue. */
#ifdef DEBUG
       DebugMsg("SNSend: Response queue is only for responses.");
#endif
       DeQueue(&gp->tsaRespQ);
       return;
    }

    /* Search for the associated RR for this response. */
    for (i = 0; i < gp->recvRecCnt; i++)
    {
       if (gp->recvRec[i].status == SESSION_RR &&
            gp->recvRec[i].reqId == tsaSendParamPtr->reqId)
       {
            break;
```

广东省网络空间安全协会受控资料

```
        }
    }
    if (i == gp->recvRecCnt ||
        gp->recvRec[i].serviceType != REQUEST ||
        gp->recvRec[i].transState != DELIVERED)
    {
        /* Stale or duplicate response. Ignore it. */
        DeQueue(&gp->tsaRespQ);
#ifdef DEBUG
        DebugMsg("SNSend: Discarding stale or duplicate response.");
#endif
        INCR_STATS(nmp->stats.lateResponses);
        return;
    }
    /* Copy the response to the receive record and send response. */
    gp->recvRec[i].rspSize = tsaSendParamPtr->apduSize;
    /* Resp should fit in response field as it was found to be
        fit in the TSA queue. */
    memcpy(gp->recvRec[i].response,
            (char *) (tsaSendParamPtr + 1),
            gp->recvRec[i].rspSize);
    gp->recvRec[i].transState = RESPONDED;
    SNSendResponse(i, tsaSendParamPtr->nullResponse);
    INCR_STATS(nmp->stats.layer6_7RespSent);
    DeQueue(&gp->tsaRespQ);
    return;
}

/* Update transmit timers, if they do exist */
if (gp->priXmitRec.status == SESSION_TX &&
    gp->priXmitRec.xmitTimer.curTimerValue > 0)
{
    UpdateMsTimer(&gp->priXmitRec.xmitTimer);
}
if (gp->xmitRec.status == SESSION_TX &&
    gp->xmitRec.xmitTimer.curTimerValue > 0)
{
    UpdateMsTimer(&gp->xmitRec.xmitTimer);
}

/*************************************************************
  Priority transmit timer expired event.
  *************************************************************/
```

广东省网络空间安全协会受控资料

```
    if (gp->priXmitRec.status == SESSION_TX &&
        gp->priXmitRec.xmitTimer.curTimerValue == 0)
    {
        XmitTimerExpiration(SESSION, TRUE);
        return;
    }
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      Send a new priority message event.
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    else if (gp->priXmitRec.status == UNUSED_TX &&
            ! QueueEmpty(&gp->tsaOutPriQ)        &&
            ! QueueFull(&gp->nwOutPriQ) )
    {
        SendNewMsg(SESSION, TRUE);
        return;
    }
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      Non-priority timer expired event.
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    else if (gp->xmitRec.status == SESSION_TX &&
            gp->xmitRec.xmitTimer.curTimerValue == 0)
    {
        XmitTimerExpiration(SESSION, FALSE);
    }
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      Send a new non-priority message.
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    else if (gp->xmitRec.status == UNUSED_TX &&
            ! QueueEmpty(&gp->tsaOutQ)        &&
            ! QueueFull(&gp->nwOutQ) )
    {
        SendNewMsg(SESSION, FALSE);
    }
    else
    {
        /* Either there is no work or there is no space. */
        return;
    }

    return;

}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:  SNReceive
Returns:   None
Reference: None
Purpose:   To receive and process incoming SPDU's.
Comments:  None
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void SNReceive(void)
{
   TSAReceiveParam *tsaReceiveParamPtr;/* Param in tsa input queue.    */
   TSPDUPtr        spduInPtr;          /* Pointer to SPDU received.  */
   int16           i;

   /* Update Receive Timers, if they do exist */
   for (i = 0; i < gp->recvRecCnt; i++)
   {
      if (gp->recvRec[i].status == SESSION_RR)
      {
         if (gp->recvRec[i].recvTimer.curTimerValue > 0)
         {
            UpdateMsTimer(&gp->recvRec[i].recvTimer);
         }
         if (gp->recvRec[i].recvTimer.curTimerValue == 0)
         {
            /* Timer expired. Release the receive record.  */
#ifdef DEBUG
            DebugMsg("SNReceive: Receive timer expired.");
#endif
            /* Deliver the request if it has not been delivered.
               Deliver fn checks for authentication.  */
            if (gp->recvRec[i].transState == DELIVERED ||
                gp->recvRec[i].transState == DONE ||
                gp->recvRec[i].transState == RESPONDED)
            {
               gp->recvRec[i].status = UNUSED_RR; /* Release the RR */
            }
            else
            {
               Deliver(i);
            }
         }
         else if ((gp->recvRec[i].transState == JUST_RECEIVED &&
                   ! gp->recvRec[i].needAuth) ||
```

广东省网络空间安全协会受控资料

```
                gp->recvRec[i].transState == AUTHENTICATED )
        {
            /* The request does not need authentication and was just received
               or it has been authenticated. */
            Deliver(i);
        }
    }
}

/* Check if there is SPDU to be processed. */
if (QueueEmpty(&gp->tsaInQ))
{
    /* There is nothing to process. */
    return;
}

tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
spduInPtr          = (TSPDUPtr) ((char *)tsaReceiveParamPtr
                                 + sizeof(TSAReceiveParam));

/* Check if the PDU is for session layer. If not, we are done. */
if (tsaReceiveParamPtr->pduType != SPDU_TYPE)
{
    return;
}

/* We have a SPDU to be processed. Check the type and
   process accordingly. */
switch (spduInPtr->pduMsgType)
{
    case RESPONSE_MSG:
        SNReceiveResponse();
        return;
    case REQUEST_MSG:
        ReceiveNewMsg(SESSION);
        return;
    case REMINDER_MSG:
        /* Fall through. */
    case REM_MSG_MSG:
        ReceiveRem(SESSION);
        return;
    default:
        nmp->errorLog = UNKNOWN_PDU;
```

广东省网络空间安全协会受控资料

```
        ErrorMsg("SNReceive: Unknown SPDU type received.");
        DeQueue(&gp->tsaInQ);
    }

    return;
}

/*****************************************************************
Function:   AuthSend
Returns:    None
Reference:  None
Purpose:    To send challenges that are pending in receive records.
            These are challenges that could not be sent earlier due
            to unavailable space in network queue.
Comments:   None
*****************************************************************/
void AuthSend(void)
{
    int16 i;

    /* Only thing the authentication layer can do here is to check if any
       challenges need to be sent. */
    for (i = 0; i < gp->recvRecCnt; i++)
    {
        if (gp->recvRec[i].status != UNUSED_RR &&
            gp->recvRec[i].needAuth &&
            gp->recvRec[i].transState == JUST_RECEIVED)
        {
            InitiateChallenge(i);
        }
    }

    return;
}

/*****************************************************************
Function:   AuthReceive
Returns:    None
Reference:  None
Purpose:    To receive an incoming AUTH_PDU packet and process.
Comments:   None
*****************************************************************/
```

广东省网络空间安全协会受控资料

```
void AuthReceive(void)
{
    TSAReceiveParam *tsaReceiveParamPtr; /* Parameter in tsa input queue. */
    AuthPDUPtr      pduInPtr;            /* Ptr to PDU received. */

    /* Check if there is AuthPDU to be processed. */
    if (QueueEmpty(&gp->tsaInQ))
    {
        /* There is nothing to process. */
        return;
    }

    tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
    pduInPtr           = (AuthPDUPtr) (tsaReceiveParamPtr + 1);

    /* Check if the PDU is for session layer. If not, we are done. */
    if (tsaReceiveParamPtr->pduType ! = AUTHPDU_TYPE)
    {
        return;
    }

    /* We have a AuthPDU to be processed. Check the type and
       process accordingly. */
    if (pduInPtr->pduMsgType == CHALLENGE_MSG)
    {
        SendReplyToChallenge();
    }
    else if (pduInPtr->pduMsgType == REPLY_MSG)
    {
        ProcessReply();
    }
    else
    {
        ErrorMsg("AuthReceive: Unknown pdu type received.");
        nmp->errorLog = UNKNOWN_PDU;
        DeQueue(&gp->tsaInQ);
        return;
    }

}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:  InitiateChallenge
```

山东省网络空间安全协会受控资料

```
Returns:    None
Reference: None
Purpose:    To send a challenge message for a message received.
Comments:   None
****************************************/
static void InitiateChallenge(uint16 rrIndexIn)
{
   AuthPDUPtr         pduOutPtr;              /* Ptr to PDU sent. */
   NWSendParam        *nwSendParamPtr;
   Queue              *nwQueuePtr;
   Byte               randomValue[8];
   uint8              i;

   /* Since authentication is not allowed in flex domain, if
      this fn is called when a message received is in flex domain,
      let us force the authentication to be a failure */
   if (gp->recvRec[rrIndexIn].srcAddr.domainIndex == FLEX_DOMAIN)
   {
      gp->recvRec[rrIndexIn].auth = FALSE;
      gp->recvRec[rrIndexIn].transState = AUTHENTICATED;
      return;
   }

   if (gp->recvRec[rrIndexIn].priority)
   {
      nwQueuePtr = &gp->nwOutPriQ;
   }
   else
   {
      nwQueuePtr = &gp->nwOutQ;
   }

   if (QueueFull(nwQueuePtr))
   {
      /* No space to send challenge anyway. Come back later. */
      return;
   }

   nwSendParamPtr = QueueTail(nwQueuePtr);
   pduOutPtr      = (AuthPDU *)(nwSendParamPtr + 1);

   /* First compute the random bytes to be sent */
   if (gp->recvRec[rrIndexIn].transState != AUTHENTICATING)
```

广东省网络空间安全协会受控资料

```
{
    /* Generate random number only the first time we are called
       for this message. subsequent calls use the same rand. */
    for (i = 0; i < 8; i++)
    {
      randomValue[i] = (Byte)((gp->prevChallenge[i] + rand() % 256 +
                        GetCurrentTime() % 256) % 256);
      gp->prevChallenge[i] = randomValue[i];
    }
    memcpy(gp->recvRec[rrIndexIn].rand, randomValue, 8); /* Save */
}

/* Form the challenge AuthPDU. */
pduOutPtr->fmt = addrModeToFmt[gp->recvRec[rrIndexIn].srcAddr.addressMode];
if (gp->recvRec[rrIndexIn].srcAddr.addressMode == MULTICAST)
{
    /* For Multicast message, send the group info with AuthPDU. */
    nwSendParamPtr->pduSize = 10;
    pduOutPtr->group      = gp->recvRec[rrIndexIn].srcAddr.group;
}
else
{
    nwSendParamPtr->pduSize = 9;
}
pduOutPtr->pduMsgType   = CHALLENGE_MSG;
pduOutPtr->transNum = gp->recvRec[rrIndexIn].transNum;
memcpy(pduOutPtr->value.randomBytes,
        gp->recvRec[rrIndexIn].rand,
        8);

/* Now fill in the NWSendParam structure. */
nwSendParamPtr->dropIfUnconfigured = FALSE; /* Challenges are not dropped */
/* Challenge must have come from a particular node.
   Use format 2b for multicast and format 2a for others. */
nwSendParamPtr->destAddr.domainIndex =
    gp->recvRec[rrIndexIn].srcAddr.domainIndex;
if (gp->recvRec[rrIndexIn].srcAddr.addressMode == MULTICAST)
{
    nwSendParamPtr->destAddr.addressMode = MULTICAST_ACK;
    nwSendParamPtr->destAddr.addr.addr2b.subnetAddr =
       gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
    nwSendParamPtr->destAddr.addr.addr2b.groupAddr.group =
       gp->recvRec[rrIndexIn].srcAddr.group;
```

广东省网络空间安全协会受控资料

```
        if (! IsGroupMember(gp->recvRec[rrIndexIn].srcAddr.domainIndex,
                            gp->recvRec[rrIndexIn].srcAddr.group,
                            &nwSendParamPtr->destAddr.addr.addr2b.groupAddr.member))
        {
            ErrorMsg("InitiateChallenge: Strange: We are not member of group???.");
            return; /* Don't challenge. This case should not happen. */
        }
    }
    else
    {
        nwSendParamPtr->destAddr.addressMode = SUBNET_NODE;
        nwSendParamPtr->destAddr.addr.addr2a =
            gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
    }

    nwSendParamPtr->pduType = AUTHPDU_TYPE;
    nwSendParamPtr->deltaBL = 0;
    nwSendParamPtr->altPath = gp->recvRec[rrIndexIn].altPath;
    gp->recvRec[rrIndexIn].transState = AUTHENTICATING;
    EnQueue(nwQueuePtr);
#ifdef DEBUG
    DebugMsg("InitiateChallenge: Sending a challenge.");
#endif
    return;
}

/*****************************************************************
Function:   SendReplyToChallenge
Returns:    None
Reference: None
Purpose:    To send a reply to a challenge just received.
Comments:   None
*****************************************************************/
static void SendReplyToChallenge(void)
{
    TSAReceiveParam *tsaReceiveParamPtr; /* Parameter in tsa input queue. */
    AuthPDUPtr      pduInPtr;            /* Ptr to PDU received. */
    AuthPDUPtr      pduOutPtr;           /* Ptr to PDU sent.     */
    NWSendParam     *nwSendParamPtr;
    Queue           *nwQueuePtr;
    TransmitRecord  *xmitRecPtr;
    Byte            encryptValue[8];
```

广东省网络空间安全协会受控资料

```
tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
pduInPtr           = (AuthPDUPtr) (tsaReceiveParamPtr + 1);

if (tsaReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN)
{
    /* Challenge was received in flex domain that is not
       possible. We don't initiate challenge in flex domain.
       Ignore the challenge. */
    DeQueue(&gp->tsaInQ);
    return;
}

if (tsaReceiveParamPtr->priority)
{
    nwQueuePtr = &gp->nwOutPriQ;
    xmitRecPtr = &gp->priXmitRec;
}
else
{
    nwQueuePtr = &gp->nwOutQ;
    xmitRecPtr = &gp->xmitRec;
}


/* Make sure that this challenge for current transaction
   in progress. If not, it is stale. Ignore it. */
/* Also do not reply if we did not set auth bit or
   the group value in challenge msg, if present,
   does not match the one in transmit record. */
if (
     xmitRecPtr->status == UNUSED_TX ||
     ! xmitRecPtr->auth ||
     pduInPtr->transNum ! = xmitRecPtr->transNum ||
     addrFmtToMode[pduInPtr->fmt] ! = xmitRecPtr->nwDestAddr.addressMode ||
     (pduInPtr->fmt == 1 &&
       xmitRecPtr->nwDestAddr.addr.addr1 ! = pduInPtr->group)
    )
{
    DeQueue(&gp->tsaInQ);
    INCR_STATS(nmp->stats.lateChallenges);
    return;
```

广东省网络空间安全协会受控资料

```
}

if (QueueFull(nwQueuePtr))
{
    /* No Space to send reply anyway. Come back later. */
    return;
}
nwSendParamPtr = QueueTail(nwQueuePtr);
pduOutPtr      = (AuthPDU *)(nwSendParamPtr + 1);

/* First compute the cryptoBytes to be sent. */
Encrypt(pduInPtr->value.randomBytes,
        xmitRecPtr->apdu, xmitRecPtr->apduSize,
        tsaReceiveParamPtr->srcAddr.domainIndex,
        encryptValue);

/* Form the reply AuthPDU. */
pduOutPtr->fmt = pduInPtr->fmt;
if (pduInPtr->fmt == 1)
{
    pduOutPtr->group = pduInPtr->group;
    nwSendParamPtr->pduSize = 10;
}
else
{
    nwSendParamPtr->pduSize = 9;
}
pduOutPtr->pduMsgType = REPLY_MSG;
pduOutPtr->transNum = xmitRecPtr->transNum;
memcpy(pduOutPtr->value.cryptoBytes, encryptValue, 8);

/* Now fill in the NWSendParam structure */
nwSendParamPtr->dropIfUnconfigured = FALSE; /* Replies are not dropped */
/* Challenge must have come from a particular node.
   Use subnet addressing to send to that node. */
nwSendParamPtr->destAddr.domainIndex =
    tsaReceiveParamPtr->srcAddr.domainIndex;
nwSendParamPtr->destAddr.addressMode = SUBNET_NODE;
nwSendParamPtr->destAddr.addr.addr2a =
    tsaReceiveParamPtr->srcAddr.subnetAddr;
nwSendParamPtr->pduType = AUTHPDU_TYPE;
nwSendParamPtr->deltaBL = 0;
nwSendParamPtr->altPath = tsaReceiveParamPtr->altPath;
```

广东省网络空间安全协会受控资料

```
    EnQueue(nwQueuePtr);
    DeQueue(&gp->tsaInQ);
#ifdef DEBUG
    DebugMsg("SendReply: Sending a reply msg.");
#endif
    /* Restart the transmit timer. */
    SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);
    return;
}

/*****************************************************************
Function:   ProcessReply
Returns:    None
Reference:  None
Purpose:    To process a reply msg received to see if it meets the
            challenge sent earlier.
Comments:   None
*****************************************************************/
static void ProcessReply(void)
{
    TSAReceiveParam *tsaReceiveParamPtr;  /* Parameter in tsa input queue. */
    AuthPDUPtr      pduInPtr;             /* Ptr to PDU received. */
    int16           i;
    Byte            encryptValue[8];
    Byte            domainIndex;

    tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
    pduInPtr           = (AuthPDUPtr) (tsaReceiveParamPtr + 1);

    if (tsaReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN)
    {
        /* Reply received in flex domain. Ignore it. */
        DeQueue(&gp->tsaInQ);
        return;
    }

    /* If the reply indicates that the original address format is
       multicast, then make sure that the receive record matches
       this group number. Force it so that RetriveRR will do the
       match. */
    tsaReceiveParamPtr->srcAddr.group = pduInPtr->group;

    /* The address format of the reply will be different from the address
```

广东省网络空间安全协会受控资料

```
    format of the original message. The original address format is in
    the Reply PDU. Change the addressmode in source address before
    calling RetrieveRR. */
tsaReceiveParamPtr->srcAddr.addressMode = addrFmtToMode[pduInPtr->fmt];

/* Retrieve the associated record.

    If the original address format is broadcast, we don't have
    the subnet number in the current reply(as the reply itself does
    not use broadcast address mode) and hence we can't match
    subnet number. So, we don't support authentication in
    broadcast mode. */

i = RetrieveRR(tsaReceiveParamPtr->srcAddr,
               tsaReceiveParamPtr->priority);

if (i == -1)
{
    /* We did not find an associated RR. Ignore reply. */
    DeQueue(&gp->tsaInQ);
    INCR_STATS(nmp->stats.lateReplies);
    return;
}

/* We have a RR. */

/* Check if we have already authenticated. */
if (gp->recvRec[i].transState != AUTHENTICATING)
{
    /* Ignore this reply as it is probably a duplicate. */
    DeQueue(&gp->tsaInQ);
    return;
}

/* Check if other things match to make sure that we are
    authenticating the right RR. */
if (pduInPtr->fmt == 1 &&
     pduInPtr->group != gp->recvRec[i].srcAddr.group)
{
    /* Group in AuthPDU does not match one in RR. */
#ifdef DEBUG
    DebugMsg("ProcessReply: Group does not match. Ignore reply.");
#endif
```

广东省网络空间安全协会受控资料

```
        DeQueue(&gp->tsaInQ);
        return;
    }

    if (pduInPtr->transNum ! = gp->recvRec[i].transNum)
    {
        /* Transaction Number does not match. Ignore reply. */
#ifdef DEBUG
        DebugMsg("ProcessReply: TID does not match. Ignore reply.");
#endif
        DeQueue(&gp->tsaInQ);
        return;
    }

    /* Now check if the reply matches our encryption. */
    /* First compute the value of E(range, apdu) */

    domainIndex = gp->recvRec[i].srcAddr.domainIndex;
    if (domainIndex == FLEX_DOMAIN)
    {
        /* We should not have initiated challenge in flexDomain. */
        /* Somthing is wrong. Ignore this reply. */
#ifdef DEBUG
        DebugMsg("ProcessReply: Oops! Challenge was unnecessary.");
#endif
        gp->recvRec[i].auth = FALSE; /* Flex Domain? Force it */
        DeQueue(&gp->tsaInQ);
        return;
    }

    Encrypt(gp->recvRec[i].rand,
            gp->recvRec[i].apdu, gp->recvRec[i].apduSize,
            domainIndex, encryptValue);

    if (memcmp(encryptValue, pduInPtr->value.cryptoBytes, 8) == 0)
    {
        /* Matches */
#ifdef DEBUG
        DebugMsg("ProcessReply: Reply matches.");
#endif
        gp->recvRec[i].auth = TRUE;
    }
    else
```

广东省网络空间安全协会受控资料

```
    {
#ifdef DEBUG
        DebugMsg("ProcessReply: Reply Does not match");
#endif
        nmp->errorLog = AUTHENTICATION_MISMATCH;
        gp->recvRec[i].auth = FALSE;
    }
    gp->recvRec[i].transState = AUTHENTICATED;

    /* Deliver the message now to the application layer. */
    Deliver(i);

    /* Send ack message if it is for transport layer and ackd msg. */
    if (gp->recvRec[i].status == TRANSPORT_RR &&
        gp->recvRec[i].serviceType == ACKD      &&
        gp->recvRec[i].transState == DELIVERED)
    {
        TPSendAck(i);
    }

    DeQueue(&gp->tsaInQ);
    return;
}

/*****************************************************************
Function:   Encrypt
Returns:    None
Reference:  Protocol Spec (Online version)
Purpose:    To compute the encryption key based on authentication
            key in the domain table, the APDU, and the random
            number given.
Comments:   None
*****************************************************************/
static void Encrypt(Byte randIn[], APDU *apduIn, uint16 apduSizeIn,
                Byte domainIndexIn, Byte encryptValueOut[])
{
    int8 i,j,k;
    Byte m, n;
    Byte *apduBytes = (Byte *)apduIn;
    uint16 apduSize = apduSizeIn;

    for (i = 0; i < 8; i++)
    {
```

广东省网络空间安全协会受控资料

```
        encryptValueOut[i] = randIn[i];
    }

    do
    {
        for (i = 0; i <= 5; i++)
        {
            for (j = 7; j >= 0; j--)
            {
                k = (j + 1) % 8;
                if (apduSize > 0)
                {
                    m = apduBytes[--apduSize];
                }
                else
                {
                    m = 0;
                }
                n = ~(encryptValueOut[j] + j);
                if (eep->domainTable[domainIndexIn].key[i] &
                    (1 << (7 - j)))
                {
                    encryptValueOut[j] =
                        encryptValueOut[k] + m + ((n << 1) + (n >> 7));
                }
                else
                {
                    encryptValueOut[j] =
                        encryptValueOut[k] + m - ((n >> 1) + (n << 7));
                }
            }
        }
    } while (apduSize > 0);
}
/*-------------------End of tsa.c-------------------*/
```

广东省网络空间安全协会受控资料

A.9 应用层

```
/*****************************************************
        File:        app.c

      Version:       1.7
```

```
     Reference:        Section 11, Protocol Specification

       Purpose:        To implement session/application layer.

          Note:        None

         To Do:        None
****************************************************************************/

/****************************************************************************

Overview:

All messages coming into the application layer are handled by
the APPReceive() function, that dispatches messages to the
appropriate "Handle" function. Some of the functions explained here
are local to the application layer and are not available to the application
program. See "Local Function Prototypes" section below.

HandleNormal():     Handles normal messages bound to the application program.
                    These are messages from other nodes that are bound to the
                    application program.

HandleResponse(): Handles normal responses bound to the application program.
                    These are responses due to requests generated by the
                    application program.

ProcessNV():        Handles incoming NV update and poll messages.
                    These are messages from other nodes or responses for
                    requests generated by the application layer.

SendVar(): Takes care of network variable updates waiting to be sent.
            i. e implicit messages generated due to output network variable
            updates in the application program.

PollVar(): Takes care of network variable polls waiting to be dispatched.
            These are messages explicitly generated by the application
            program by calling various forms of poll functions.
            The corresponding variables are input network variables.

HandleNM(): Processes network management messages. Defn is in netmgmt. c.
```

广东省网络空间安全协会受控资料

HandleND(): Processes network diagnostic messages. Defn is in netmgmt.c.

HandleProxyResponse(): Handles responses from proxy messages. Defn is in netmgmt.c. i.e responses received from the target node are relayed back to the original node.

HandleMsgCompletion(): Handles transaction completion indications from transport, session, and network layers. Some of them are for transactions originated by the application layer itself. Others are for transactions originated by the application program. In that case, the application program is notified of these completion events by a call to MsgCompletes or NVUpdateCompletes function.

TryMsgSend(): Helper function that moves messages in the application queue to lower layer queues as appropriate.

All messages generated by the application program (implicit or explicit) are buffered and then sent to lower layers by the APPSend() function.

广东省网络空间安全协会受控资料

APPReset(): Takes care of the application layer reset operations. Allocates memory for all queue structures used by the application. Also, performs some initialization that are done after each reset.

APPInit(): Takes care of initializations that are done only during power up.

AddNV(): Takes care of network variable declarations in the application program. This function is normally called during power up to register all network variables in application program. This is done using the function AppInit(). Note that APPInit() is application layer function whereas AppInit() is application program function.

RefImpToAlt(), AltToRefImp(): Functions to transform data between a format often used by 709 applications (such as msg_in, msg_out, resp_in etc) and the corresponding structures for reference implementation (msgIn, msgOut, etc).

The functions in this file also support the API interface used by application programs. e.g. msg_send, resp_send, propagate() etc.

There is no implicit way of sending network variable updates in the reference

```
implementation. The application program should call Propagate or one
of its variants to actually propagate network variable updates.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* ------------------------------------------------------------------------------
Section: Includes
------------------------------------------------------------------------------ */
#include <stdio.h>
#include <string.h>

#include <CEA709_1.h>
#include <node.h>
#include <queue.h>
#include <app.h>
#include <api.h>
#include <netmgmt.h>

/* ------------------------------------------------------------------------------
Section: Constant and Macro Definitions
------------------------------------------------------------------------------ */
#define MAX_NV_SELF_DOC_LENGTH 1023

/* Depending on whether ALTERNATE_STRUCTURES are used in application
   program or not, define the macros REF_IMP_TO_ALT and ALT_TO_REF_IMP
   accordingly. */
#if defined(ALTERNATE_STRUCTURES_NEEDED)
#define REF_IMP_TO_ALT(a,b) RefImpToAlt(a,b)
#define ALT_TO_REF_IMP(a,b) AltToRefImp(a,b)
#else
#define REF_IMP_TO_ALT(a,b)
#define ALT_TO_REF_IMP(a,b)
#endif

/* ------------------------------------------------------------------------------
Section: Type Definitions
------------------------------------------------------------------------------ */
/* None */

/* ------------------------------------------------------------------------------
Section: Globals
------------------------------------------------------------------------------ */
/* None */

/* ------------------------------------------------------------------------------
```

广东省网络空间安全协会受控资料

```
Section: Local Function Prototypes
------------------------------------------------------------------------*/
static void ProcessNV(APPReceiveParam *appReceiveParamPtr,
                      APDU            *apduPtr);
static void ProcessNVUpdate(APPReceiveParam *appReceiveParamPtr,
                            APDU            *apduPtr);
static void ProcessNVPoll(APPReceiveParam *appReceiveParamPtr,
                          APDU            *apduPtr);

static void HandleMsgCompletion(APPReceiveParam *appReceiveParamPtr,
                                APDU            *apduPtr);
static void HandleResponse(APPReceiveParam *appReceiveParamPtr,
                           APDU            *apduPtr);
static void HandleNormal(APPReceiveParam *appReceiveParamPtr,
                         APDU *apduPtr);

static Status  TryMsgSend(APPSendParam *appSendParamPtr,
                          APDU *apduPtr,
                          Queue    *tsaOutQPtr,
                          Queue    *nwOutQPtr);

static void ReinitMsgOut();
static void ReinitRespOut();

static Status PropagateThisIndex(int16 nvIndexIn, int16 primaryIndex);
static void   PropagateThisPrimary(int16 nvIndexIn);
static void   SendVar(void);

static Status PollThisIndex(int16 nvIndexIn);
static void   PollThisPrimary(int16 nvIndexIn);
static void   PollVar(void);

static Boolean IsArrayNV(int16 nvIndexIn,
                         uint16 *dimOut, int16 *indexOut);

static void ReinitMsgOut(void);
static void ReinitRespOut(void);
/*------------------------------------------------------------------------
Section: Function Definitions
------------------------------------------------------------------------*/

/*****************************************************************
Function:  APPInit
```

广东省网络空间安全协会受控资料

```
Returns:    None
Reference: None
Purpose:    To perform initializations that should be done only
            once when the node is powered on. These are not
            done during node reset.
Comments:   None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void APPInit(void)
{
   uint16  len;

   gp->unboundSelector          = 0x3FFF;  /* Countdown as we assign */
   gp->nvArrayTblSize           = 0;
   gp->nextBindableMsgTag       = 0;
   gp->nextNonbindableMsgTag = NUM_ADDR_TBL_ENTRIES;

   /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     The SNVT area has the following layout (as expected by Network
     management tools):

     Header:
     uint16 length;
     uint8  numNetvars;
     uint8  version;
     uint8  msbNumNetvars;
     uint8  mtagCount;

     SNVT Desc:
     SNVTdescStruct[];   One struct (2 bytes) per var

     Node Self Doc String:
     char[];        Null terminated; just "\0" if empty

     SNVT Extension Records:
     one byte + various lengths

     Alias Field
     three bytes

     In the reference implementation, the SNVT information is
     stored in nmp->snvt, of type SNVTstruct. It can be
     stored in EEPROM too.
```

广东省网络空间安全协会受控资料

```
In SNVTstruct the header fields are stored in explicit
structure members, but the remaining information is stored
in the 'sb' member that is simply a buffer of type char.

When a network variable is added via AddNV():
1. The Alias Field is saved.
2. Room is made for the new SNVT Desc by moving all data
past the last SNVT Desc forward by sizeof(SNVTdesc).
The descPtr pointer indicates where the move starts.
3. The new SNVT Desc is added (and descPtr is incremented).
4. The new extension records are appended after the
existing extension records.
5. The Alias Field is restored (placed after the last
extension record, and pointed to by aliasPtr).

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* Initialize SNVT area */
nmp->snvt.version        = 1;
nmp->snvt.numNetvars     = 0;
nmp->snvt.msbNumNetvars = 0;
nmp->snvt.mtagCount      = 0;

/* Copy NODE_DOC info if there is sufficient space */
/* If not, init to null string. */
len = strlen(NODE_DOC) + 1;
if (len <= SNVT_SIZE - sizeof(AliasField))
{
    strcpy((char *)&nmp->snvt.sb[0], NODE_DOC);
}
else
{
    nmp->snvt.sb[0] = '\0';
    len = 1;
}

/* Initially, there is no network var related info */
nmp->snvt.length    = 6 + len + sizeof(AliasField);
nmp->snvt.descPtr   = (SNVTdescStruct *)&nmp->snvt.sb[0];
nmp->snvt.aliasPtr = (AliasField *)&nmp->snvt.sb[len];
nmp->snvt.aliasPtr->bindingII   = TRUE;
nmp->snvt.aliasPtr->queryStats = TRUE;
nmp->snvt.aliasPtr->aliasCount = 0x3F;
```

广东省网络空间安全协会受控资料

```
    nmp->snvt.aliasPtr->hostAlias   = NV_ALIAS_TABLE_SIZE;

    nmp->nvTableSize   = 0;
}

/*****************************************************************
Function:   APPReset
Returns:    None
Reference:  None
Purpose:    To initialize the Queues used by the application layer.
            May be some more.
Comments:   None.
*****************************************************************/
void APPReset(void)
{
    uint16 queueItemSize;

    /* Allocate and Initialize Input Queue */
    gp->appInBufSize   =
        DecodeBufferSize((uint8)eep->readOnlyData.appInBufSize);
    gp->appInQCnt      = DecodeBufferCnt((uint8)eep->readOnlyData.appInBufCnt);
    queueItemSize      = gp->appInBufSize + sizeof(APPReceiveParam);

    if (QueueInit(&gp->appInQ, queueItemSize, gp->appInQCnt)
        != SUCCESS)
    {
        ErrorMsg("APPReset: Unable to init Input Queue.\n");
        gp->resetOk = FALSE;
        return;
    }

    /* Allocate and Initialize Output Queue */
    gp->appOutBufSize =
        DecodeBufferSize((uint8)eep->readOnlyData.appOutBufSize);
    gp->appOutQCnt     =
        DecodeBufferCnt((uint8)eep->readOnlyData.appOutBufCnt);
    queueItemSize      = gp->appOutBufSize + sizeof(APPSendParam);

    if (QueueInit(&gp->appOutQ, queueItemSize, gp->appOutQCnt)
        != SUCCESS)
    {
        ErrorMsg("APPReset: Unable to init Output Queue.\n");
        gp->resetOk = FALSE;
```

广东省网络空间安全协会受控资料

```
    return;
}

/* Allocate and Initialize Pri Output Queue */
gp->appOutPriBufSize = gp->appOutBufSize;
gp->appOutPriQCnt =
    DecodeBufferCnt((uint8)eep->readOnlyData.appOutBufPriCnt);
queueItemSize      = gp->appOutPriBufSize + sizeof(APPSendParam);

if (QueueInit(&gp->appOutPriQ, queueItemSize, gp->appOutPriQCnt)
    != SUCCESS)
{
    ErrorMsg("APPReset: Unable to init Priority Output Queue.\n");
    gp->resetOk = FALSE;
    return;
}

/* Allocate Queue for NV output variable scheduling */
gp->nvOutIndexQCnt      = MAX_NV_OUT;
gp->nvOutIndexBufSize = 2 + MAX_NV_LENGTH;
if (QueueInit(&gp->nvOutIndexQ, gp->nvOutIndexBufSize,
              gp->nvOutIndexQCnt)  != SUCCESS)
{
    ErrorMsg("APPReset: Unable to init NV Out Index Queue.\n");
    gp->resetOk = FALSE;
    return;
}
gp->nvOutStatus        = SUCCESS; /* Propagate succeeds if all the scheduled
                                     transactions complete successfully. */
gp->nvOutCanSchedule = TRUE;
gp->nvOutIndex         = 0; /* Not relevant initially */

/* Allocate Queue for NV input variable scheduling */
gp->nvInIndexQCnt = MAX_NV_IN;
if (QueueInit(&gp->nvInIndexQ, 2, gp->nvInIndexQCnt)
    != SUCCESS)
{
    ErrorMsg("APPReset: Unable to init NV In Index Queue.\n");
    gp->resetOk = FALSE;
    return;
}
gp->nvInDataStatus   = FAILURE; /* See node.h for usage */
gp->nvInTranStatus   = SUCCESS; /* See node.h for usage */
```

广东省网络空间安全协会受控资料

```
    gp->nvInCanSchedule = TRUE;
    gp->nvInIndex       = 0; /* Not relevant initially */

    /* Set flags to correct state */
    gp->msgReceive      = FALSE;      /* TRUE if data is in gp->msgIn   */
    gp->respReceive     = FALSE;      /* TRUE if data is in gp->respIn  */
    gp->callMsgFree     = FALSE;
    gp->callRespFree    = FALSE;
    gp->selectQueryFlag = FALSE;      /* FALSE until selected           */

    /* Init msg and resp */
    memset(&gp->msgIn,   0, sizeof(gp->msgIn));
    memset(&gp->msgOut,  0, sizeof(gp->msgOut));
    memset(&gp->respIn,  0, sizeof(gp->respIn));
    memset(&gp->respOut, 0, sizeof(gp->respOut));
    memset(&gp->nvInAddr, 0, sizeof(gp->nvInAddr));
    gp->nvArrayIndex = 0;
#if defined(ALTERNATE_STRUCTURES_NEEDED)
    memset(&msg_in,   0, sizeof(msg_in));
    memset(&msg_out,  0, sizeof(msg_out));
    memset(&resp_in,  0, sizeof(resp_in));
    memset(&resp_out, 0, sizeof(resp_out));
    memset(&nv_in_addr, 0, sizeof(nv_in_addr));
    nv_array_index = 0;
#endif
}

/*****************************************************************
Function:   HandleMsgCompletion
Returns:    None
Reference:  None
Purpose:    Pass a message completion event to the application
Comments:   None.
*****************************************************************/
static void HandleMsgCompletion(APPReceiveParam *appReceiveParamPtr,
                                APDU            *apduPtr)
{
    Status stat;
    int16 primaryIndex, baseIndex;
    uint16 dim;
    Boolean pollCompletion = TRUE;

    if(appReceiveParamPtr->success)
```

广东省网络空间安全协会受控资料

```
{
    stat = SUCCESS;
}
else
{
    stat = FAILURE;
}

/* Since App Layer sends messages for NV update
   we may also get completion indication for these.
   See app.h for tag usage. */
if (appReceiveParamPtr->tag < 0)
{
    /* Negative tags belong to application layer. */
    /* See app.h for explanation on how the tag is used. */
    if (PROXY_TAG(appReceiveParamPtr->tag))
    {
        /* Proxy Completion. Release the proxy command structure */
        nmp->pxyData.pxyType = -1;
    }
    else if (MANUAL_SERVICE_REQUEST_TAG(appReceiveParamPtr->tag))
    {
        ; /* Ignore. Nothing to do. */
    }
    else
    {
        /* Must be a tag for NV message generated by application layer. */
        if (NV_POLL_TAG(appReceiveParamPtr->tag))
        {
            /* A network variable poll for an input variable will
               succeed if both gp->nvInDataStatus and gp->nvInTranStatus succeed.
               The gp->nvInDataStatus flag is updated by ProcessNVUpdate
               function. */
            if (NV_LAST_TAG(appReceiveParamPtr->tag))
            {
                primaryIndex    = gp->nvInIndex;
                IsArrayNV(primaryIndex, &dim, &baseIndex);
                gp->nvArrayIndex = primaryIndex - baseIndex;
                REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
                /* Set poll completion status */
                if ( (gp->nvInDataStatus == SUCCESS) &&
                     (gp->nvInTranStatus == SUCCESS)
                    )
```

广东省网络空间安全协会受控资料

```
        {
            stat = SUCCESS;
        }
        else
        {
            stat = FAILURE;
        } /* poll completion status */
        gp->nvInDataStatus = FAILURE; /* Reinit */
        gp->nvInTranStatus = SUCCESS;
        if (AppPgmRuns())
        {
            NVUpdateCompletes(stat, baseIndex, gp->nvArrayIndex);
        }
    }
    else
    {
        /* Update the index. Since the last tag does not have the
            index, we need to save it. */
        gp->nvInIndex = NV_INDEX_OF_TAG(appReceiveParamPtr->tag);
        if (stat == FAILURE)
        {
            /* Set the flag to FAILURE as this transaction failed. */
            gp->nvInTranStatus = FAILURE;
        }
    }
    gp->nvInCanSchedule = TRUE; /* Resume scheduling */
}
else
{
    /* NV_UPDATE_TAG */
    /* A network variable update for an output variable will
        succeed if all the transactions scheduled for that
        variable succeed. So, we need to update status flag. */
    if (NV_LAST_TAG(appReceiveParamPtr->tag))
    {
        primaryIndex    = gp->nvOutIndex;
        IsArrayNV(primaryIndex, &dim, &baseIndex);
        gp->nvArrayIndex = primaryIndex - baseIndex;
        REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
        stat = gp->nvOutStatus;
        gp->nvOutStatus = SUCCESS; /* Reinit */
        if (AppPgmRuns())
        {
```

广东省网络空间安全协会受控资料

```
                NVUpdateCompletes(stat, baseIndex, gp->nvArrayIndex);
            }
        }
        else
        {
            /* Set index. Update gp->nvOutStatus */
            gp->nvOutIndex = NV_INDEX_OF_TAG(appReceiveParamPtr->tag);
            if (stat == FAILURE)
            {
                gp->nvOutStatus = FAILURE;
            }
        }
        gp->nvOutCanSchedule = TRUE; /* Resume scheduling */
    }
  }
}
else
{
    /* Non-negative tags belong to application. */
    MsgCompletes(stat, appReceiveParamPtr->tag);
}

/* Message processing completed - remove it from queue */
DeQueue(&gp->appInQ);
}

/*****************************************************************
Function:   HandleResponse
Returns:    None
Reference:  None
Purpose:    Handle incoming response message, passing it to the
            application via the gp->respIn global.
Comments:   None.
*****************************************************************/
static void HandleResponse(APPReceiveParam *appReceiveParamPtr,
                           APDU            *apduPtr)
{
  /* Discard responses received when the node is not CNFG_ONLINE.
     In CNFG_ONLINE, the application can be either running (online)
     or not running (soft-offline). In either of these cases,
     we receive the response and store it in resp_in. Note that
     one reponse can be received at a time from resp_in. If the
     application is offline, then at most one response can be
```

广东省网络空间安全协会受控资料

```
    stored in resp_in */
if(eep->readOnlyData.nodeState != CNFG_ONLINE)
{
    DeQueue(&gp->appInQ); /* Discard response, nothing to do */
    return;
}

/* If the application has not processed the previous
 * response, then do nothing, we'll try again next time
 */
if(gp->respReceive)
{
    return;
}

/* Pass response to application */

/* Setup gp->respIn */
gp->respIn.tag  = appReceiveParamPtr->tag;
gp->respIn.code = apduPtr->code.allBits;
gp->respIn.len = appReceiveParamPtr->pduSize - 1;

/* Copy domainIndex even if it is 2. respIn.domainIndex is
    only one bit anyway, so the result is 0 or 1 */
gp->respIn.addr.domain = appReceiveParamPtr->srcAddr.domainIndex;
gp->respIn.addr.flexDomain =
    (appReceiveParamPtr->srcAddr.domainIndex == 2);
memcpy(&gp->respIn.addr.srcAddr,
        &appReceiveParamPtr->srcAddr.subnetAddr,
        sizeof(appReceiveParamPtr->srcAddr.subnetAddr));
gp->respIn.addr.srcAddr.snodeFlag =
    appReceiveParamPtr->srcAddr.addressMode != MULTICAST_ACK;
if (gp->respIn.addr.srcAddr.snodeFlag == 0)
{
    memcpy(&gp->respIn.addr.destAddr.group,
            &appReceiveParamPtr->srcAddr.ackNode,
            sizeof(appReceiveParamPtr->srcAddr.ackNode));
}
else if (! gp->respIn.addr.flexDomain)
{
    /* Fill snode entry only for non-flex domain response */
    gp->respIn.addr.destAddr.snode.subnet =
        eep->domainTable[appReceiveParamPtr->
```

广东省网络空间安全协会受控资料

```
                                srcAddr.domainIndex].subnet;
        gp->respIn.addr.destAddr.snode.node    =
            eep->domainTable[appReceiveParamPtr->
                                srcAddr.domainIndex].node;
    }
    if (gp->respIn.len <= gp->appInBufSize)
    {
        memcpy(gp->respIn.data, &apduPtr->data, gp->respIn.len);
        gp->respReceive = TRUE;
#if defined(ALTERNATE_STRUCTURES_NEEDED)
            REF_IMP_TO_ALT(RESP_IN, &resp_in);
#endif
    }
    else
    {
        nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
    }
    /* Message processing completed — remove it from queue */
    DeQueue(&gp->appInQ);
}

/*****************************************************************
Function:   HandleNormal
Returns:    None
Reference:  None
Purpose:    Handle incoming normal message
Comments:   If the application program is not running and a request message
            for the application program is received, we send a offline
            message. Note that the node state could be unconfigured or
            soft off-line.
*****************************************************************/
static void HandleNormal(APPReceiveParam *appReceiveParamPtr,
                                APDU              *apduPtr)
{
    Queue           *tsaOutQPtr;
    TSASendParam    *tsaSendParamPtr;
    APDU            *apduRespPtr;

    if(! AppPgmRuns())
    {
        if(appReceiveParamPtr->service == REQUEST)
        {
            /* If not online, send offline response */
```

广东省网络空间安全协会受控资料

```
        tsaOutQPtr = &gp->tsaRespQ;
        if(QueueFull(tsaOutQPtr))
        {
            /* Can't send response yet — try later */
            return;
        }
        tsaSendParamPtr = QueueTail(tsaOutQPtr);
        tsaSendParamPtr->altPathOverride = FALSE;
        tsaSendParamPtr->service = RESPONSE;
        tsaSendParamPtr->reqId = appReceiveParamPtr->reqId;
        tsaSendParamPtr->nullResponse = FALSE;
        tsaSendParamPtr->apduSize = 1; /* Just the code */
        apduRespPtr = (APDU *)(tsaSendParamPtr + 1);
        if(apduPtr->code.ff.ffFlag == 0x4)
        {
            apduRespPtr->code.allBits = FOREIGN_FRAME_OFFLINE;
        }
        else
        {
            apduRespPtr->code.allBits = APPL_MSG_OFFLINE;
        }
        EnQueue(tsaOutQPtr);
    }
    DeQueue(&gp->appInQ); /* Discard msg, nothing to do */
    return;
}

/* If the application has not processed the previous
 * message, then do nothing, we'll try again next time
 */
if(gp->msgReceive)
{
    return;
}

/* Pass message to application */

/* Setup gp->msgIn */
gp->msgIn.code = apduPtr->code.allBits;
gp->msgIn.len = appReceiveParamPtr->pduSize - 1;
if (gp->msgIn.len <= gp->appInBufSize)
{
    memcpy(gp->msgIn.data, &apduPtr->data, gp->msgIn.len);
```

广东省网络空间安全协会受控资料

```
}
else
{
    nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
    DeQueue(&gp->appInQ);
    return;

}

gp->msgIn.authenticated = appReceiveParamPtr->auth;
gp->msgIn.service = appReceiveParamPtr->service;
gp->msgIn.reqId = appReceiveParamPtr->reqId;
/* Fill in addr structure */
gp->msgIn.addr.domain   = appReceiveParamPtr->srcAddr.domainIndex;
gp->msgIn.addr.flexDomain =
    (appReceiveParamPtr->srcAddr.domainIndex == 2);
/* Copy Source Address */
memcpy(&gp->msgIn.addr.srcAddr,
        &appReceiveParamPtr->srcAddr.subnetAddr,
        sizeof(gp->msgIn.addr.srcAddr));
switch(appReceiveParamPtr->srcAddr.addressMode)
{
    case BROADCAST:
        gp->msgIn.addr.format   = 0;
        gp->msgIn.addr.destAddr.bcastSubnet =
            appReceiveParamPtr->srcAddr.broadcastSubnet;
        break;
    case MULTICAST:
        gp->msgIn.addr.format   = 1;
        gp->msgIn.addr.destAddr.group =
            appReceiveParamPtr->srcAddr.group;
        break;
    case SUBNET_NODE:
        gp->msgIn.addr.format   = 2;
        if (! gp->msgIn.addr.flexDomain)
        {
            gp->msgIn.addr.destAddr.snode.subnet =
                eep->domainTable[gp->msgIn.addr.domain].subnet;
            gp->msgIn.addr.destAddr.snode.node   =
                eep->domainTable[gp->msgIn.addr.domain].node;
        }
        break;
    case UNIQUE_NODE_ID:
```

山东省网络空间安全协会受控资料

```
        gp->msgIn.addr.format   = 3;
        gp->msgIn.addr.destAddr.uniqueNodeId.subnet = 0; /* Not stored */
        memcpy(gp->msgIn.addr.destAddr.uniqueNodeId.uniqueId,
                eep->readOnlyData.uniqueNodeId,
                UNIQUE_NODE_ID_LEN);
        break;
      default:
        /* should not come here */
        gp->msgIn.addr.format   = 5; /* unknown. arbitrary 5 */
    }

    gp->msgReceive = TRUE;
    REF_IMP_TO_ALT(MSG_IN, &msg_in);
    /* Message processing completed — remove it from queue */
    DeQueue(&gp->appInQ);
}


/*****************************************************************
Function:   APPSend
Returns:    None
Reference:  None
Purpose:    Process send side of the application layer
            Send one message of each of the following types if there is space.
              pri msg, nv update message, nv poll message, non-pri msg.
Comments:   Called by scheduler loop.
*****************************************************************/
void APPSend(void)
{
    Queue           *appOutQPtr;
    Queue           *tsaOutQPtr;
    Queue           *nwOutQPtr;
    APPSendParam    *appSendParamPtr;
    APDU            *apduPtr;
    Status          status;

    if(gp->manualServiceRequest)
    {
        ManualServiceRequestMessage();
        gp->manualServiceRequest = FALSE;
    }

    /* Call MsgFree or RespFree implicitely if needed. */
```

广东省网络空间安全协会受控资料

```
if (gp->callMsgFree)
{
    MsgFree();
}
if (gp->callRespFree)
{
    RespFree();
}

/* Send a priority message if we can */
if (! QueueEmpty(&gp->appOutPriQ))
{
    appOutQPtr        = &gp->appOutPriQ;
    tsaOutQPtr        = &gp->tsaOutPriQ;
    nwOutQPtr         = &gp->nwOutPriQ;
    appSendParamPtr = QueueHead(appOutQPtr);
    apduPtr           = (APDU *)(appSendParamPtr + 1);
    status            = TryMsgSend(appSendParamPtr, apduPtr,
                                   tsaOutQPtr, nwOutQPtr);
    if (status == SUCCESS)
    {
        /* We have moved this message. Discard it */
        DeQueue(appOutQPtr);
    }
}

/* Process one NV output variable scheduled, if any. */
SendVar();

/* Process one NV input variable (Poll) scheduled, if any. */
PollVar();

/* Send a non-priority message if we can */
if (! QueueEmpty(&gp->appOutQ))
{
    appOutQPtr        = &gp->appOutQ;
    tsaOutQPtr        = &gp->tsaOutQ;
    nwOutQPtr         = &gp->nwOutQ;
    appSendParamPtr = QueueHead(appOutQPtr);
    apduPtr           = (APDU *)(appSendParamPtr + 1);
    status            = TryMsgSend(appSendParamPtr, apduPtr,
                                   tsaOutQPtr, nwOutQPtr);
    if (status == SUCCESS)
```

广东省网络空间安全协会受控资料

```
        {
            /* We have moved this message. Discard it */
            DeQueue(appOutQPtr);
        }

    }
}


/*****************************************************************
Function:  APPReceive
Returns:   None
Reference: None
Purpose:   Process receive side of the application layer.
Comments:  Called by scheduler loop.
*****************************************************************/
void APPReceive(void)
{
    APPReceiveParam        *appReceiveParamPtr;
    APDU                   *apduPtr;      /* ptr to APDU being received  */

    /* Check if anything to process */
    if(QueueEmpty(&gp->appInQ))
    {
        return; /* Nothing to process */
    }

    /* Set the pointer to APDU in appInQ */
    appReceiveParamPtr = QueueHead(&gp->appInQ);
    apduPtr            = (APDU *)(appReceiveParamPtr + 1);

    if(appReceiveParamPtr->indication == COMPLETION)
    {
        HandleMsgCompletion(appReceiveParamPtr,apduPtr);
    }
    else if(appReceiveParamPtr->service == RESPONSE &&
        appReceiveParamPtr->tag >= 0)
    {
        /* The response belongs to the application program.
            Deliver it to the application irrespective of
            what type of message it is. If the application
            sends network management/diagnostic messages,
            it is its responsibility to handle them */
```

广东省网络空间安全协会受控资料

```
        HandleResponse(appReceiveParamPtr,apduPtr);
    }
    /* Handle Response from a Network Diagnostic Proxy Command */
    /* pxyType is initialized to -1. If a proxy command has been
        received, it is set to the command requested (0 or 1 or 2).
        The command is forwarded using a tag value of PROXY_TAG.
        So, check if this message is a response and is a response
        to a previously forwarded proxy command.
        The response should match (success or failure) of
        original the request */
    else if(nmp->pxyData.pxyType != -1 &&
            PROXY_TAG(appReceiveParamPtr->tag) &&
            appReceiveParamPtr->service == RESPONSE &&
            (apduPtr->code.allBits == 0x32 ||
             apduPtr->code.allBits == 0x12)
           )
    {
        /* Indeed we have a good proxy response. Forward to source */
        HandleProxyResponse(appReceiveParamPtr,apduPtr);
    }
    else if (appReceiveParamPtr->service == RESPONSE &&
            PROXY_TAG(appReceiveParamPtr->tag))
    {
        /* ill-formed proxy response. Ignore. */
        DeQueue(&gp->appInQ);
    }
    else if(apduPtr->code.nm.nmFlag == 0x3)
    {
        /* Network management messages */
        HandleNM(appReceiveParamPtr,apduPtr);
    }
    else if(apduPtr->code.nd.ndFlag == 0x5)
    {
        /* Network diagnostic messages */
        HandleND(appReceiveParamPtr,apduPtr);
    }
    else if(apduPtr->code.nv.nvFlag == 0x1)
    {
        /* Network variable update/poll messages */
        ProcessNV(appReceiveParamPtr,apduPtr);
    }
    else if(apduPtr->code.ap.apFlag == 0x0 || apduPtr->code.ff.ffFlag == 0x4)
    {
```

广东省网络空间安全协会受控资料

```
        /* Messages bound to the application program. App Msg or Foriegn Frames. */
        HandleNormal(appReceiveParamPtr,apduPtr);
    }
    else
    {
        /* Not supported — just discard */
        DeQueue(&gp->appInQ);
    }

}

/*****************************************************************
Function:  TryMsgSend
Returns:   SUCCESS if message sent, FAILURE otherwise
Reference: None
Purpose:   If room is available, move message from app queue to
           to tsa or nw output queue.
Comments:  None.
*****************************************************************/
static Status TryMsgSend(APPSendParam *appSendParamPtr,
                         APDU *apduPtr,
                         Queue *tsaOutQPtr,
                         Queue *nwOutQPtr)
{
    TSASendParam    *tsaSendParamPtr;
    NWSendParam     *nwSendParamPtr;
    APDU            *apduSendPtr;

    /* If the address is bad, don't even bother sending it */
    if(appSendParamPtr->addr.noAddress == UNBOUND)
    {
        /* TurnAround is not possible with MsgOutAddr */
        MsgCompletes(SUCCESS, appSendParamPtr->tag);
        return(SUCCESS);
    }

    /* Simple unacknowledged messages go to network layer */
    if(appSendParamPtr->service == UNACKD)
    {
        if(QueueFull(nwOutQPtr))
        {
            return(FAILURE); /* Can't send message yet — try later */
        }
```

广东省网络空间安全协会受控资料

```
nwSendParamPtr = QueueTail(nwOutQPtr);

nwSendParamPtr->dropIfUnconfigured = TRUE;

switch(appSendParamPtr->addr.noAddress)
{
  case UNBOUND:
      /* Can't happen. We took care above */
      return(SUCCESS); /* doesn't matter what we return */
  case SUBNET_NODE:
      nwSendParamPtr->destAddr.addressMode = SUBNET_NODE;
      nwSendParamPtr->destAddr.domainIndex =
         appSendParamPtr->addr.snode.domainIndex;
      nwSendParamPtr->destAddr.addr.addr2a.subnet =
         appSendParamPtr->addr.snode.subnetID;
      nwSendParamPtr->destAddr.addr.addr2a.node    =
         appSendParamPtr->addr.snode.node;
      break;

  case UNIQUE_NODE_ID:
      nwSendParamPtr->destAddr.addressMode = UNIQUE_NODE_ID;
      nwSendParamPtr->destAddr.domainIndex =
         appSendParamPtr->addr.uniqueNodeId.domainIndex;
      nwSendParamPtr->destAddr.addr.addr3.subnet =
         appSendParamPtr->addr.uniqueNodeId.subnetID;
      memcpy(nwSendParamPtr->destAddr.addr.addr3.uniqueId,
            appSendParamPtr->addr.uniqueNodeId.uniqueId, UNIQUE_NODE_ID_LEN);
      break;

  case BROADCAST:
      nwSendParamPtr->destAddr.addressMode = BROADCAST;
      nwSendParamPtr->destAddr.domainIndex =
         appSendParamPtr->addr.bcast.domainIndex;
      nwSendParamPtr->destAddr.addr.addr0 =
         appSendParamPtr->addr.bcast.subnetID;
      break;

  default: /* MUST BE GROUP FORMAT */
      nwSendParamPtr->destAddr.addressMode = MULTICAST;
      nwSendParamPtr->destAddr.domainIndex =
         appSendParamPtr->addr.group.domainIndex;
      nwSendParamPtr->destAddr.addr.addr1 =
```

广东省网络空间安全协会受控资料

```
            appSendParamPtr->addr.group.groupID;
        break;
    } /* switch */

    nwSendParamPtr->tag      = appSendParamPtr->tag;
    nwSendParamPtr->pduType = APDU_TYPE;
    nwSendParamPtr->deltaBL = 0;
    /* unacknowledged messages do not use alternate path */
    nwSendParamPtr->altPath = FALSE;
    nwSendParamPtr->pduSize = appSendParamPtr->len + 1;

    apduSendPtr = (APDU *)(nwSendParamPtr + 1);
    if (nwSendParamPtr->pduSize <= gp->nwOutBufSize)
    {
        memcpy(apduSendPtr, apduPtr, appSendParamPtr->len + 1);
        EnQueue(nwOutQPtr);
        /* Don't give completion yet. The network layer will send the
           completion indication in gp->appInQ */
    }
    else
    {
        /* Losing this packet as it is too large */
        MsgCompletes(FAILURE, appSendParamPtr->tag);
    }

    return(SUCCESS);
}

if(QueueFull(tsaOutQPtr))
{
    return(FAILURE); /* Can't send message yet — try later */
}

/* All other service types go to TSA layers */
tsaSendParamPtr = QueueTail(tsaOutQPtr);

tsaSendParamPtr->domainIndex = 3;
memcpy(&tsaSendParamPtr->destAddr,
        &appSendParamPtr->addr,
        sizeof(appSendParamPtr->addr));
tsaSendParamPtr->service    = appSendParamPtr->service;
tsaSendParamPtr->auth       = appSendParamPtr->authenticated;
tsaSendParamPtr->reqId      = appSendParamPtr->reqId;
```

广东省网络空间安全协会受控资料

```
    tsaSendParamPtr->tag          = appSendParamPtr->tag;
    tsaSendParamPtr->apduSize     = appSendParamPtr->len + 1;
    tsaSendParamPtr->altPathOverride = FALSE;

    apduSendPtr = (APDU *)(tsaSendParamPtr + 1);
    if (tsaSendParamPtr->apduSize <= gp->tsaOutBufSize)
    {
        memcpy(apduSendPtr, apduPtr, tsaSendParamPtr->apduSize);
        EnQueue(tsaOutQPtr);
    }
    else
    {
        /* Losing this message */
        MsgCompletes(FAILURE, appSendParamPtr->tag);
    }

    return(SUCCESS);
}

static void ReinitMsgOut(void)
{
    /* Reinit whatever fields need to be reinitialized after each msg_send */
    memset(&gp->msgOut, 0, sizeof(gp->msgOut));
#if defined(ALTERNATE_STRUCTURES_NEEDED)
    memset(&msg_out, 0, sizeof(msg_out));
#endif
}

static void ReinitRespOut(void)
{
    /* Reinit whatever fields need to be reinitialized after each resp_send */
    memset(&gp->respOut, 0, sizeof(gp->respOut));
#if defined(ALTERNATE_STRUCTURES_NEEDED)
    memset(&resp_out, 0, sizeof(resp_out));
#endif
}

/*****************************************************************
Function:   MsgAlloc
Returns:    None
Reference:  None
Purpose:    Determines if there is room for a message to be sent.
Comments:   None.
```

广东省网络空间安全协会受控资料

```
**************************************************/
Boolean MsgAlloc(void)
{
   return(! QueueFull(&gp->appOutQ));
}

/* For nc compatibility */
Boolean msg_alloc(void)
{
   return(! QueueFull(&gp->appOutQ));
}

/*************************************************
Function:  MsgAllocPriority
Returns:   None
Reference: None
Purpose:   Determines if there is room for a pri message to be sent.
Comments:  None.
**************************************************/
Boolean MsgAllocPriority(void)
{
   return(! QueueFull(&gp->appOutPriQ));
}

/* For nc compatibility */
Boolean msg_alloc_priority(void)
{
   return(! QueueFull(&gp->appOutPriQ));
}

/*************************************************
Function:  MsgSend
Returns:   None
Reference: None
Purpose:   The application calls this function to send a message.
           The message is placed in the application queue for
           processing by APPSend.
Comments:  None.
**************************************************/
void MsgSend(void)
{
   Queue            *outQptr;
   APPSendParam     *appSendParamPtr;
```

广东省网络空间安全协会受控资料

```
APDU                *apduPtr;
AddrTableEntry      *ap;
uint16              addrIndex;


ALT_TO_REF_IMP(MSG_OUT, &msg_out);

if(gp->msgOut.priorityOn)
{
    outQptr = &gp->appOutPriQ;
}
else
{
    outQptr = &gp->appOutQ;
}

/* Negative tags are reserved for application layer.
   Applications are not allowed to use negative tags */
if(QueueFull(outQptr) || gp->msgOut.tag < 0 ||
    (gp->msgOut.service >= RESPONSE))
{
    /* Bad Tag  OR
       Bad Service OR
       No place to put the message — discard it. This should
       not happen if application called MsgAlloc or
       MsgPriorityAlloc before forming the message */
    MsgCompletes(FAILURE, gp->msgOut.tag);
    ReinitMsgOut();
    return;
}

appSendParamPtr                 = QueueTail(outQptr);
appSendParamPtr->tag            = gp->msgOut.tag;
appSendParamPtr->len            = gp->msgOut.len;
appSendParamPtr->authenticated  = gp->msgOut.authenticated;
appSendParamPtr->service        = gp->msgOut.service;
appSendParamPtr->addr           = gp->msgOut.addr;
apduPtr                         = (APDU *)(appSendParamPtr + 1);
apduPtr->code.allBits           = gp->msgOut.code;
if (appSendParamPtr->len + 1 <= gp->appOutBufSize)
{
    /* There is space in the queue item to copy data */
    memcpy((char *)apduPtr+1,
```

广东省网络空间安全协会受控资料

```
            gp->msgOut.data,
            appSendParamPtr->len);
    }
    else
    {
        /* We are losing this message as it is too big. */
        MsgCompletes(FAILURE, appSendParamPtr->tag);
        ReinitMsgOut();
        return;
    }

    /* Use implicit addressing if the tag value corresponds to an address
        table entry and the explicit address is unbound or turnaround.
        Thus explicit address can be used to override implicit addressing.
        Note that turnaround is not allowed with explicit messages. */
    if(appSendParamPtr->tag < NUM_ADDR_TBL_ENTRIES &&
        gp->msgOut.addr.noAddress == UNBOUND)
    {
        addrIndex = appSendParamPtr->tag;
        ap = AccessAddress(addrIndex); /* ap cannot be NULL */
        if (ap == NULL || ap->addrFormat == UNBOUND)
        {
            /* ap cannot be NULL, but we can be safe in checking it anyway.
                We lose this message as the address table entry is unbound
                or turnaround. */
            MsgCompletes(FAILURE, appSendParamPtr->tag);
            ReinitMsgOut();
            return;
        }
        memcpy(&appSendParamPtr->addr,
                ap,
                sizeof(MsgOutAddr));
    }
    EnQueue(outQptr);
    ReinitMsgOut();
}

/* For nc compatibility */
void msg_send(void)
{
    MsgSend();
}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   MsgCancel
Returns:    None
Reference:  None
Purpose:    The application calls this function to cancel a
            previous message allocation.
Comments:   None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void MsgCancel(void)
{
   /* Nothing to do */
}

/* For nc compatibility */
void msg_cancel()
{
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   MsgFree
Returns:    None
Reference:  None
Purpose:    Releases gp->msgIn (by clearning gp->msgReceive) so that the
            next message can be copied to gp->msgIn.
Comments:   None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void MsgFree(void)
{
   gp->msgReceive = FALSE; /* TRUE when data is in gp->msgIn   */
   gp->callMsgFree = FALSE;
}

/* For nc compatibility */
void msg_free(void)
{
   gp->msgReceive = FALSE;
   gp->callMsgFree = FALSE;
}

/* TRUE if there is msg to be received */
Boolean   msgReceive(void)
{
   if (gp->msgReceive)
```

广东省网络空间安全协会受控资料

```
    {
        /* There is a message to be received. Need to call msg_free() */
        gp->callMsgFree = TRUE;
    }
    return(gp->msgReceive);
}

/* For nc compatibility */
Boolean  msg_receive(void)
{
    if (gp->msgReceive)
    {
        /* There is a message to be received. Need to call msg_free() */
        gp->callMsgFree = TRUE;
    }
    return(gp->msgReceive);
}

/*****************************************************************
Function:  RespAlloc
Returns:   None
Reference: None
Purpose:   Check if there is space for sending a response.
Comments:  None.
*****************************************************************/
Boolean RespAlloc(void)
{
    if (! QueueFull(&gp->tsaRespQ))
    {
        return(TRUE);
    }
    return(FALSE);
}

/* For nc compatibility */
Boolean resp_alloc(void)
{
    return(RespAlloc());
}

/*****************************************************************
Function:  RespSend
Returns:   None
```

广东省网络空间安全协会受控资料

```
Reference: None
Purpose:   Reads gp->respOut, sends the response message
Comments:  None.
*****************************************************************/
void RespSend(void)
{
   TSASendParam  * tsaSendParamPtr;
   APDU          * apduRespPtr;

   ALT_TO_REF_IMP(RESP_OUT, &resp_out);

   if (gp->respOut.reqId == 0)
   {
      /* Application did not initialize it. Let us use as default
         the reqId of message currently in gp->msgIn */
      gp->respOut.reqId = gp->msgIn.reqId;
   }

   /* Place the response in tsaRespQ. If it is full, discard the
      response. There should be space if application called
      RespAlloc before forming the response */
   if (! QueueFull(&gp->tsaRespQ))
   {
      tsaSendParamPtr                  = QueueTail(&gp->tsaRespQ);
      apduRespPtr                      = (APDU *)(tsaSendParamPtr + 1);
      tsaSendParamPtr->altPathOverride = FALSE;
      tsaSendParamPtr->service         = RESPONSE;
      tsaSendParamPtr->reqId           = gp->respOut.reqId;
      tsaSendParamPtr->apduSize        = gp->respOut.len + 1;
      tsaSendParamPtr->nullResponse = gp->respOut.nullResponse;
      apduRespPtr->code.allBits        = gp->respOut.code;
      if (tsaSendParamPtr->apduSize <= gp->tsaRespBufSize)
      {
         memcpy((char *)apduRespPtr + 1,
                gp->respOut.data,
                gp->respOut.len);
         EnQueue(&gp->tsaRespQ);
      }
      else
      {
         /* Lose this response as it is too big to fit */
         /* There is no errorLog or stat to record this */
         ;
```

广东省网络空间安全协会受控资料

```
        }
    }
    else
    {
        /* Sorry! Application is sending a response without checking
           for space first */
        ;
    }
    ReinitRespOut();
}

/* For nc compatibility */
void resp_send(void)
{
    RespSend();
}

/*****************************************************************************
Function:   RespCancel
Returns:    None
Reference:  None
Purpose:    The application calls this function to cancel a
            previous response allocation.
Comments:   None.
*****************************************************************************/
void RespCancel(void)
{
    /* Nothing to do */
}

/* For nc compatibility */
void resp_cancel(void)
{
}


/*****************************************************************************
Function:   RespFree
Returns:    None
Reference:  None
Purpose:    Sets gp->respReceive to FALSE, which will allow a new
            response to be received.
Comments:   None.
```

广东省网络空间安全协会受控资料

```
***************************************/
void RespFree(void)
{
    gp->respReceive = FALSE; /* TRUE when data is in gp->respIn */
    gp->callRespFree = FALSE;
}

/* For nc compatibility */
void resp_free(void)
{
    gp->respReceive = FALSE;
    gp->callRespFree = FALSE;
}

/* Returns TRUE if there is resp to be received */
Boolean RespReceive(void)
{
    if (gp->respReceive)
    {
        /* There is a response to be received. Need to call resp_free() */
        gp->callRespFree = TRUE;
    }
    return(gp->respReceive);
}

/* For nc comptability */
Boolean resp_receive(void)
{
    if (gp->respReceive)
    {
        /* There is a response to be received. Need to call resp_free() */
        gp->callRespFree = TRUE;
    }
    return(gp->respReceive);
}

void RefImpToAlt(RefImpAltStruct whatIn, void *destAddr)
{
    switch(whatIn)
    {
        case MSG_IN:
            memcpy(destAddr, &gp->msgIn, sizeof(gp->msgIn));
            break;
```

广东省网络空间安全协会受控资料

```
        case MSG_OUT:
            memcpy(destAddr, &gp->msgOut, sizeof(gp->msgOut));
            break;
        case RESP_IN:
            memcpy(destAddr, &gp->respIn, sizeof(gp->respIn));
            break;
        case RESP_OUT:
            memcpy(destAddr, &gp->respOut, sizeof(gp->respOut));
            break;
        case NV_IN_ADDR:
            memcpy(destAddr, &gp->nvInAddr, sizeof(gp->nvInAddr));
            break;
        case NV_ARRAY_INDEX:
            memcpy(destAddr, &gp->nvArrayIndex, sizeof(gp->nvArrayIndex));
            break;
        default:
            /* Do nothing */
            ;
    }
}
void AltToRefImp(RefImpAltStruct whatIn, void * srcAddr)
{
    switch(whatIn)
    {
        case MSG_IN:
            memcpy(&gp->msgIn, srcAddr, sizeof(gp->msgIn));
            break;
        case MSG_OUT:
            memcpy(&gp->msgOut, srcAddr, sizeof(gp->msgOut));
            break;
        case RESP_IN:
            memcpy(&gp->respIn, srcAddr, sizeof(gp->respIn));
            break;
        case RESP_OUT:
            memcpy(&gp->respOut, srcAddr, sizeof(gp->respOut));
            break;
        case NV_IN_ADDR:
            memcpy(&gp->nvInAddr, srcAddr, sizeof(gp->nvInAddr));
            break;
        case NV_ARRAY_INDEX:
            memcpy(&gp->nvArrayIndex, srcAddr, sizeof(gp->nvArrayIndex));
            break;
        default:
```

广东省网络空间安全协会受控资料

```
        /* Do nothing */
        ;
    }
}

/*****************************************************************
Function:   AddNV
Returns:    Network variable index. For arrays, the base index is returned.
Reference:  None
Purpose:    Adds a new network variable. This involves
            adding an entry into nvConfigTable, nvFixedTable,
            SNVT information, if present etc.
            The return value is the index assigned to the variable.
            For arrays, each element is like a separate network
            variable. So, multiple entries are added to the
            tables. However, only the base index is returned.
Comments:   Format of the snvt.sb space is as follows:
    Self-Id for each network variable (SNVTdescStruct)
        (For arrays, one entry for each element)
    Node Self-Doc string.
    Self-Doc for each network variable
        (SNVTExtension & variable part)
    Self-Id for binding and status (AliasField)
*****************************************************************/
int16 AddNV(NVDefinition *dp)
{
    uint16          i, nvSelfIdCnt;
    uint16          nvNameLen; /* Length for name of network variable. */
    uint16          docLen;    /* Length for self-doc for network var. */
    uint16          selectorVal;
    uint16          sizeNeeded;
    uint16          dim;
    uint16          jumpBy;    /* Number of bytes by that we shift sb array. */
    uint8           remainder,quotient;
    SNVTextension   *se;
    SNVTdescStruct  *sd;
    AliasField      saveAlias; /* To save old alias field. */
    char            *p;
    char            *extPtr;   /* Points to where the new ext rec can be stored. */
    char            *endOfSb;  /* Points to end of sb array. */

    /* Initialize local pointers for structure information in dp so that
       we can use field names in these structures instead of explicit
```

广东省网络空间安全协会受控资料

```
    bit operations */
se = (SNVTextension *)  &dp->snvtExt;
/* snvtType info is not part of the snvtDesc. We only want to access the
    first byte of the structure, anyway. */
sd = (SNVTdescStruct *) &dp->snvtDesc;

/* First determine the number of entries to be added to nvConfigTable */
if (dp->arrayCnt > 0)
{
    dim = dp->arrayCnt; /* Array Variable. dim can still be 1. */
}
else
{
    dim = 1; /* Simple variable */
}

if (dp->nvName == NULL || dp->varAddr == NULL)
{
    return(-1); /* Network variable name and address is a must. */
}

if (nmp->nvTableSize + dim > NV_TABLE_SIZE)
{
    /* Not enough space in network variable table. */
    return(-1);
}

/* Save the original alias field. */
saveAlias = *nmp->snvt.aliasPtr;

/* Make endOfSb point to last element of sb array. We never want to go
    past this element */
endOfSb = (char *)&nmp->snvt.sb[SNVT_SIZE - 1];

/* Compute the size needed for this variable in SNVT structure. */
if (dp->arrayCnt > 0 && dp->explodeArray)
{
    /* We need one entry for each network variable. */
    nvSelfIdCnt = dim;
}
else
{
    nvSelfIdCnt = 1;
```

广东省网络空间安全协会受控资料

```
}
sizeNeeded = sizeof(SNVTdescStruct) * nvSelfIdCnt;

/* Fixed extRec. One for each entry in self id desc part. */
if (sd->extRec) /* Check ext_rec bit */
{
    /* Need space for fixed extension record too */
    sizeNeeded = sizeNeeded + sizeof(SNVTextension) * nvSelfIdCnt;
}

/* Variable Part of Extension Record. One for each entry in self id desc part. */
/* See Page 9—23 of Technology Device Data Book Rev 3 */
if (sd->extRec && se->mre)
{
    sizeNeeded += nvSelfIdCnt; /* One byte for maximum rate */
}
if (sd->extRec && se->re)
{
    sizeNeeded += nvSelfIdCnt; /* One byte for average rate */
}
if (sd->extRec && se->nm)
{
    nvNameLen = strlen(dp->nvName) + 1;
    if (dp->arrayCnt > 0 && dp->explodeArray)
    {
        /* Need space for [ ] and up to 3 bytes for index */
        nvNameLen += 5;
        if (nvNameLen > 22)
        {
            return(-1);
        }
    }
    else if (nvNameLen > 17)
    {
        return(-1); /* Only up to 17 bytes supported */
    }

    sizeNeeded += (nvNameLen * nvSelfIdCnt);
}
if (sd->extRec && se->sd)
{
    if (dp->nvSdoc)
```

广东省网络空间安全协会受控资料

```
    {
        docLen = strlen(dp->nvSdoc) + 1;
    }
    else
    {
        docLen = 1; /* For null character */
    }
    if (docLen > MAX_NV_SELF_DOC_LENGTH)
    {
        return(-1);
    }
    sizeNeeded += (docLen * nvSelfIdCnt);
}

if (sd->extRec && se->nc)
{
    /* 16 bit count for # of network variables of this type. dim? */
    /* Reference implementation uses ver 1. */
    sizeNeeded += (2 * nvSelfIdCnt);
}

if ((char *)nmp->snvt.aliasPtr + sizeof(AliasField) + sizeNeeded
    > endOfSb)
{
    /* No Space for the new snvt_desc_struct and extension rec. */
    return(-1);
}

/* For arrays, we need space in gp->nvArrayTblSize */
if (dp->arrayCnt > 0 && gp->nvArrayTblSize == MAX_NV_ARRAYS)
{
    /* No more space to save array information */
    return(-1);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    dp->bind = TRUE ==> the network variable is bindable. Bindable
    variables are automatically given selector numbers in the range
    0x3000-0x3FFF (unbound network variables). bindable means they
    are currently unbound.

    dp->bind = FALSE ==> the network variable is non-bindable. i.e
    the application program assigns selector numbers and these variables
```

广东省网络空间安全协会受控资料

```
    are cannot be bound by the binder tools. In this case, it is
    the responsibility of the application program to assign these
    selector numbers in a reasonable way. For array variables, the
    application program indicates the selector for the first element
    and the other elements automatically get the previous selector
    numbers. The selector numbers count down.
************************************************************/

/* We need dim many selectors. dim can be 1. */
/* bindable variables use unboundSelectors    */
selectorVal = gp->unboundSelector; /* initialize */
if (dp->bind && gp->unboundSelector - dim + 1 < 0x3000)
{
    /* Not enough selector numbers available for assigning */
    return(-1);
}
else if (! dp->bind)
{
    /* selectors are specified by the application program.
       For arrays, only the selector for first element is
       given. Application program has the responsibility to
       assign unique selectors for all variables */

    selectorVal = dp->selectorHi;
    selectorVal = (selectorVal << 8) + dp->selectorLo;
    /* We are using dim selectors. Make sure that all are in range */
    if (selectorVal + dim - 1 > 0x2FFF)
    {
       /* Nonbindable variables should have value in 0-0x2FFF */
       return(-1);
    }
}

/* Now selectorVal has the selector number to be assigned */

/* Everything is fine. We are now ready to add this variable */

/* Make extPtr point to where the new extension rec would go */
/* aliasPtr points to the byte following the last byte of ext records */
extPtr = (char *)nmp->snvt.aliasPtr + sizeof(SNVTdescStruct) * nvSelfIdCnt;

/* Add entry or entries into the nvConfigTable and Fixed Table */
/* Add one entry for each array element irrespective of whether it
```

```
    is exploded or not.  */
for (i = nmp->nvTableSize; i < nmp->nvTableSize + dim; i++)
{
    /* nv config table */
    nmp->nvConfigTable[i].nvPriority    = dp->priority;
    nmp->nvConfigTable[i].nvDirection   = dp->direction;
    nmp->nvConfigTable[i].nvSelectorHi = selectorVal >> 8;
    nmp->nvConfigTable[i].nvSelectorLo = selectorVal & 0xFF;
    selectorVal--; /* automatic countdown - for both bind & non-bind */
    if (dp->bind)
    {
        gp->unboundSelector--; /* Update this as we just used up one */
    }
    nmp->nvConfigTable[i].nvTurnaround = dp->turnaround;
    nmp->nvConfigTable[i].nvService     = dp->service;
    nmp->nvConfigTable[i].nvAuth        = dp->auth;
    nmp->nvConfigTable[i].nvAddrIndex   = 0xF;

    /* nv fixed table */
    nmp->nvFixedTable[i].nvSync       = sd->nvSync;
    nmp->nvFixedTable[i].nvLength     = dp->nvLength;
    /* For arrays, make sure we compute the address of each item.  */
    nmp->nvFixedTable[i].nvAddress    = (char *) dp->varAddr +
                                        (i - nmp->nvTableSize) * dp->nvLength;
}

/* If we are adding an array, save this info in gp->nvArrayTbl */
/* Array with dim 1 is considered like an array */
if (dp->arrayCnt > 0)
{
    gp->nvArrayTbl[gp->nvArrayTblSize].nvIndex = nmp->nvTableSize;
    gp->nvArrayTbl[gp->nvArrayTblSize++].dim   = dim;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Format of the SNVT structure: (See APPReset for more info).
    snvtheader nv-self-id-desc node-self-doc nv-self-doc alias-field

    snvtheader is part of the nmp->snvt structure, its initial members.
    nmp->snvt.sb array is used for [nv-self-id-desc ... alias-field]
    alias-fields has binding and status information.
    nmp->snvt.descPtr points to beginning address of node-self-doc.
            i.e the address where the nv-self-id-desc for new network
```

广东省网络空间安全协会受控资料

```
            variable should be placed.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* Add data to snvt structure */

/* Update snvt descriptor count in snvt structure.
   nmp->snvt.msbNumNetvars is the most significant byte.
   nmp->snvt.numNetvars is the least significant byte.
   We want to add the value of dim to this number */
quotient  = (nmp->snvt.numNetvars + nvSelfIdCnt) / 256;
remainder = (nmp->snvt.numNetvars + nvSelfIdCnt) % 256;
nmp->snvt.numNetvars      = remainder;
nmp->snvt.msbNumNetvars += quotient;

/* Now we need to move nv-self-doc and node-self-doc to make
   for self-id-desc for the network variable being added.  */

/* Make p point the last byte of nv-self-doc */
p = (char *)nmp->snvt.aliasPtr - 1;
/* Determine how many bytes everything moves by. */
jumpBy = sizeof(SNVTdescStruct) * nvSelfIdCnt;
while(p >= (char *)(nmp->snvt.descPtr))
{
   *(p + jumpBy) = *p;
   p--;
}

/* Copy dp->snvtDesc information. One for each array item.         */
/* All array items have the same information. i.e homogeneous.  */
sd = nmp->snvt.descPtr; /* Initialize */
for (i = 0; i < nvSelfIdCnt; i++)
{
   memcpy(sd, &dp->snvtDesc, 1); /* can't assign without casting. */
   sd->snvtTypeIndex = dp->snvtType;
   sd++;
}
nmp->snvt.descPtr = sd;

sd = (SNVTdescStruct *)&dp->snvtDesc; /* Reinitialize */
/* We are now ready to add the extension information */
if(sd->extRec)
{
   /* This variable has extension record following node-self-doc */
```


广东省网络空间安全协会受控资料

```
/* Add one for each entry in the self id desc part. */
for (i = 0; i < nvSelfIdCnt; i++)
{
   /* extPtr has already been set to point to the right place      */
   /* se has already been set to point to dp->snvtExt              */
   /* extPtr is of type (Byte *)                                   */
   *extPtr = dp->snvtExt;
   extPtr += sizeof(SNVTextension);
   if(se->mre)
   {
       *extPtr = dp->maxrEst;
       extPtr++;
   }

   if(se->re)
   {

       *extPtr = dp->rateEst;
       extPtr++;
   }

   if(se->nm)
   {
       /* We need to store the name of the variable. null terminated. */
       nvNameLen = strlen(dp->nvName) + 1;
       if (dp->arrayCnt > 0 && dp->explodeArray)
       {
          int index = i + 1;
          /* We support maximum of 999 array items */
          /* 999 is the maximum index we can represent with 3 digits */
          sprintf(extPtr, "%s_", dp->nvName);
          extPtr += nvNameLen; /* Should now point to null character */
          if (index < 10)
          {
              sprintf(extPtr, "%1d", index); /* Single digit index */
              extPtr++;
          }
          else if (index < 100)
          {
              sprintf(extPtr, "%2d", index); /* Double digit index */
              extPtr += 2;
          }
          else
```

广东省网络空间安全协会受控资料

```
            {
                sprintf(extPtr, "%3d", index); /* Triple digit index */
                extPtr += 3;
            }
            *extPtr = '\0';
            extPtr++;
        } /* for j loop */
        else
        {
            /* Simple variable. or Array with one entry. */
            strcpy(extPtr, dp->nvName); /* Adds Null ch too */
            extPtr += nvNameLen;
        } /* else */
    } /* if (se->nm) */

    if(se->sd)
    {
        if (dp->nvSdoc == NULL)
        {
            *extPtr = '\0'; /* nvSdoc is missing */
            extPtr++;
        }
        else
        {
            docLen = strlen(dp->nvSdoc) + 1;
            strcpy(extPtr, dp->nvSdoc);
            extPtr += docLen;
        }
    }

    if(se->nc)
    {
        /* Store the dimension of the variable. Useful only for array
           entered in SNVT that are not exploded. */
        *extPtr = dp->arrayCnt >> 8;    /* High order byte. */
        extPtr++;
        *extPtr = dp->arrayCnt & 0xFF; /* Low order byte */
        extPtr++;
    }
  } /* for */
} /* if (sd->extRec) */

/* Restore Alias */
```

广东省网络空间安全协会受控资料

```
    nmp->snvt.aliasPtr = (AliasField *)extPtr;
    *nmp->snvt.aliasPtr = saveAlias;
    extPtr = extPtr + sizeof(AliasField);

    /* Now extPtr points to the byte following the alias field */

    /* Update nmp->snvt.length */
    /* 6 below refers to the size of the header in snvt structure */
    nmp->snvt.length  = (extPtr - (char *)&nmp->snvt.sb[0]) + 6;

    nmp->nvTableSize += dim;

    return(nmp->nvTableSize - dim); /* Base index for arrays. */
}

/*****************************************************************
Function:   ProcessNV
Returns:    None
Purpose:    To process an incoming network variable message.
            The message can be
              1. NV Update Message (ACKD, UNACKD, UNACKD_RPT)
              2. NV Poll Message. (REQUEST)
Comments:   In either case, the msg should have at least 2 bytes.
*****************************************************************/
static void ProcessNV(APPReceiveParam *appReceiveParamPtr,
                      APDU            *apduPtr)
{
    if (appReceiveParamPtr->service == REQUEST)
    {
        ProcessNVPoll(appReceiveParamPtr, apduPtr);
    }
    else
    {
        ProcessNVUpdate(appReceiveParamPtr, apduPtr);
    }
}

/*****************************************************************
Function:   ProcessNVPoll
Returns:    None
Purpose:    To process an incoming network variable poll message.
            The service should be a request.
Reference:  The Technology Device Data Book Rev 3. Page 9-54
```

广东省网络空间安全协会受控资料

```
Comments:  The message should have 2 bytes.
           The network variable poll message is normally addressed
           to an output variable. However, we will allow either
           input or output variable to be polled. We simply respond
           with the data value of the matching variable.
           The application layer in reference implementation
           never sends network variable poll messages addressed
           to input variables. A network variable monitor tool
           might do this to get the value of input variables.
           (NVFetch is another way).

           If we are offline, send a response with no data.
           Else
               If we have one matching network variable (primary
               or alias), then send the response with the data.

               If we have two matching primary network variables
               then we ignore this message.

               If we don't have a matching network variable,
               then send a response with no data.

           The response is network variable update message
           with the direction flipped from what was received.

           If we have more than one primary variable with matching
           selector, then we have a problem. The sender is expecting
           only one response from each node receiving the poll message.
           So, it is not clear which one to send. If we have this
           situation, we will ignore this message and not respond
           at all. Note that it is not meaningful to have a primary
           and an alias of that primary to have the same selector.
           If the incoming variable is connected to two different primary
           variables on this node, then the incoming variable
           should not have been polled.
*********************************************/
static void ProcessNVPoll(APPReceiveParam *appReceiveParamPtr,
                          APDU             *apduPtr)
{
   int16        i;
   uint8        nvDirection;
   uint16       selector, thisSelector;
   int16        matchingIndex;
```

广东省网络空间安全协会受控资料

```
uint16          matchingPrimaryIndex;
Queue           * tsaOutQPtr;
TSASendParam    * tsaSendParamPtr;
APDU            * apduRespPtr;
NVStruct        * thisNVStrPtr, * matchingNVStrPtr;
Boolean         authOK;
int16           nvAliasTableSize;
Boolean         noData; /* Should data go out? */

nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

if(appReceiveParamPtr->pduSize ! = 2)
{
    /* The message does not have correct size */
    nmp->errorLog = NV_MSG_TOO_SHORT;
    DeQueue(&gp->appInQ);
    return;
}

tsaOutQPtr = &gp->tsaRespQ;

if(QueueFull(tsaOutQPtr))
{
    /* Can't send response yet-try later. */
    return;
}

if (NodeUnConfigured())
{
    /* Ignore this message in this state. See Tech Data Book Rev 3 p9—47. */
    tsaSendParamPtr                = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service         = RESPONSE;
    tsaSendParamPtr->nullResponse = TRUE;
    tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);
    return;
}

/* Determine selector and nvDirection for variable in the poll message. */
selector     = (apduPtr->code.nv.nvCode << 8) | apduPtr->data[0];
nvDirection = apduPtr->code.nv.nvDir;
```

广东省网络空间安全协会受控资料

```
noData = TRUE; /* Assume that we will respond with no data */
matchingIndex = -1; /* Initialize to indicate that no match yet. */

/* If application is not running, then we should return with no data */
/* We know that the node is configured at this point */
if(AppPgmRuns())
{
    /* Search for matching network variable. Search both primary
       and alias entries */
    for(i = 0; i < nmp->nvTableSize+nvAliasTableSize; i++)
    {
        thisNVStrPtr      = GetNVStructPtr(i);
        thisSelector =
            (thisNVStrPtr->nvSelectorHi << 8) | thisNVStrPtr->nvSelectorLo;
        if(thisNVStrPtr->nvDirection == nvDirection &&
            thisSelector                 == selector)
        {
            if (matchingIndex == -1)
            {
                matchingIndex = i; /* First match. */
            }
            else if (GetPrimaryIndex(matchingIndex) ==
                    GetPrimaryIndex(i))
            {
                /* We have two distinct primary variables with same
                   selector. Ignore this message. */
                tsaSendParamPtr                = QueueTail(tsaOutQPtr);
                tsaSendParamPtr->altPathOverride = FALSE;
                tsaSendParamPtr->service         = RESPONSE;
                tsaSendParamPtr->nullResponse = TRUE;
                tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
                EnQueue(tsaOutQPtr);
                DeQueue(&gp->appInQ);
                return;
            }
        }
        else
        {
            continue; /* Skip this entry. Does not match. */
        }
    }
}
```

广东省网络空间安全协会受控资料

```
/* Send the response with either data or nodata. */
tsaSendParamPtr                    = QueueTail(tsaOutQPtr);
tsaSendParamPtr->altPathOverride = FALSE;
tsaSendParamPtr->service         = RESPONSE;
tsaSendParamPtr->nullResponse = FALSE;
tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
apduRespPtr                        = (APDU *)(tsaSendParamPtr + 1);
apduRespPtr->code.nv.nvFlag    = 0x1;
apduRespPtr->code.nv.nvDir      = 1 - nvDirection; /* Opposite */
/* Keep the same selector as the one received */
apduRespPtr->code.nv.nvCode    = apduPtr->code.nv.nvCode;
apduRespPtr->data[0]             = apduPtr->data[0];

if (matchingIndex != -1)
{
    noData                  = FALSE;
    matchingPrimaryIndex = GetPrimaryIndex(matchingIndex);
    matchingNVStrPtr       = GetNVStructPtr(matchingIndex);
}

if(appReceiveParamPtr->auth ||
    matchingIndex == -1        ||
    ! matchingNVStrPtr->nvAuth)
{
    authOK = TRUE;
}
else
{
    authOK = FALSE;
}

if (noData || matchingIndex == -1 || authOK == FALSE)
{
    /* Send a response with no data */
    tsaSendParamPtr->apduSize = 2;
}
else
{
    /* Send a response with data */
    if (NV_LENGTH(matchingPrimaryIndex) + 2 <= gp->tsaRespBufSize)
    {
        memcpy(&apduRespPtr->data[1],
```

广东省网络空间安全协会受控资料

```
                NV_ADDRESS(matchingPrimaryIndex),
                NV_LENGTH(matchingPrimaryIndex));
        tsaSendParamPtr->apduSize = 2 + NV_LENGTH(matchingPrimaryIndex);
    }
    else
    {
        tsaSendParamPtr->apduSize = 2;
    }
  }
  EnQueue(tsaOutQPtr);

  /* Message processing completed — remove it from queue */
  DeQueue(&gp->appInQ);
  return;
}

/*****************************************************************
Function:   ProcessNVUpdate
Returns:    None
Purpose:    To process an incoming network variable update message.
            The service can be ACKD, UNACKD, UNACKD_RPT.
            The service can be RESPONSE too for poll responses.
Reference:  The Technology Device Data Book Rev 3. Page 9—53
Comments:   The message should have > 2 bytes.
            The network variable update message is addressed
            to an input variable.

            If we are unconfigured then discard the message.

            There can be only one variable (primary or alias) with
            matching selector. Once a match is found, break.

            If we update one ore more network variables and we are
            not soft-offline then we don't give NVUpdateOccurs event.
            Otherwise, we do give NVUpdateOccurs to the application program.

            The prefix 'this' is used for local variables of this function
            related to information regarding network variables searched.
*****************************************************************/
static void ProcessNVUpdate(APPReceiveParam *appReceiveParamPtr,
                            APDU            *apduPtr)
{
  int16          i;
```

广东省网络空间安全协会受控资料

```
uint16          dataLength, matchingDataLength;
uint8           nvDirection;
uint16          selector, thisSelector;
int16           matchingIndex;
uint16          matchingPrimaryIndex;
NVStruct        * thisNVStrPtr, * matchingNVStrPtr;
Boolean         authOK;
int16           nvAliasTableSize;
uint16          thisDim;
int16           thisBaseIndex;

nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

if(appReceiveParamPtr->pduSize <= 2)
{
    /* The message does not have any correct size or data field. */
    nmp->errorLog = NV_MSG_TOO_SHORT;
    DeQueue(&gp->appInQ);
    return;
}

if (eep->readOnlyData.nodeState == APPL_UNCNFG)
{
    /* Ignore this message in this state. See Tech Data Book Rev 3 p9-47. */
    DeQueue(&gp->appInQ);
    return;
}

/* Determine selector and nvDirection for variable in the update message. */
selector    = (apduPtr->code.nv.nvCode << 8) | apduPtr->data[0];
nvDirection = apduPtr->code.nv.nvDir;
if (nvDirection == NV_OUTPUT)
{
    /* Ignore this message */
    nmp->errorLog = NV_UPDATE_ON_OUTPUT_NV;
    DeQueue(&gp->appInQ);
    return;
}

dataLength = appReceiveParamPtr->pduSize - 2; /* data length in message */

/* Go through network input variables looking for a match. Once
   a match is found, update it and break. */
```

广东省网络空间安全协会受控资料

```
matchingIndex = -1;
for(i = 0; i < nmp->nvTableSize + nvAliasTableSize; i++)
{
    thisNVStrPtr = GetNVStructPtr(i);
    thisSelector =
        (thisNVStrPtr->nvSelectorHi << 8) | thisNVStrPtr->nvSelectorLo;
    if (thisNVStrPtr->nvDirection == NV_OUTPUT)
    {
        continue; /* Skip network output variables */
    }
    if (thisSelector == selector)
    {
        matchingIndex = i;
        break;
    }
}

if (matchingIndex ! = -1)
{
    /* Need to update the network input variable  */
    /* matchingIndex can be primary or alias index */
    matchingPrimaryIndex = GetPrimaryIndex(matchingIndex);
    matchingDataLength   = NV_LENGTH(matchingPrimaryIndex);
    matchingNVStrPtr     = GetNVStructPtr(matchingPrimaryIndex);

    /* If the data size does not match, don't update. ignore. */
    if (dataLength ! = matchingDataLength)
    {
        nmp->errorLog = NV_LENGTH_MISMATCH;
        DeQueue(&gp->appInQ);
        return;
    }

    if(appReceiveParamPtr->auth || ! matchingNVStrPtr->nvAuth)
    {
        authOK = TRUE;
    }
    else
    {
        authOK = FALSE;
    }

    if (! authOK)
```

山东省网络空间安全协会受控资料

```
{
   DeQueue(&gp->appInQ);
   return; /* Skip the update as authentication did not succeed. */
}

/* Update the variable */
if (dataLength > 0 && appReceiveParamPtr->service == RESPONSE)
{
   /* We have a response to poll message. Update gp->nvInDataStatus flag. */
   gp->nvInDataStatus = SUCCESS;
}
memcpy(NV_ADDRESS(matchingPrimaryIndex),
        &apduPtr->data[1],
        dataLength);
if (AppPgmRuns())
{
   /* Notify application program only if it is running. */
   gp->nvInAddr.domain = appReceiveParamPtr->srcAddr.domainIndex;
   gp->nvInAddr.flexDomain =
         (appReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN);
   memcpy(&gp->nvInAddr.srcAddr,
          &appReceiveParamPtr->srcAddr.subnetAddr,
          sizeof(gp->nvInAddr.srcAddr));

   switch(appReceiveParamPtr->srcAddr.addressMode)
   {
      case BROADCAST:
         gp->nvInAddr.format   = 0;
         break;
      case MULTICAST:
         gp->nvInAddr.format   = 1;
         gp->nvInAddr.destAddr.group =
             appReceiveParamPtr->srcAddr.group;
         break;
      case SUBNET_NODE:
         gp->nvInAddr.format   = 2;
         break;
      case UNIQUE_NODE_ID:
         gp->nvInAddr.format   = 3;
         break;
      default:
         /* should not come here */
         gp->nvInAddr.format   = 5; /* unknown */
```

广东省网络空间安全协会受控资料

```
        }
        IsArrayNV(matchingPrimaryIndex, &thisDim, &thisBaseIndex);
        gp->nvArrayIndex = matchingPrimaryIndex - thisBaseIndex;
        REF_IMP_TO_ALT(NV_IN_ADDR, &nv_in_addr);
        REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
        NVUpdateOccurs(thisBaseIndex, gp->nvArrayIndex);
    }
  }

  DeQueue(&gp->appInQ);
  return;
}


/*****************************************************************
Function: PropagateThisIndex
Returns:  SUCCESS if the index is scheduled.
          FAILURE if the queue is full and hence not scheduled
                  or for sync network output variables, the queue
                  buffer size is not sufficient for this variable.
          INVALID if the index does not correspond to NV_OUTPUT
Purpose:  To schedule a specific index of a network variable
          (primary or alias), polled or not.
Comment:
          This function is local to this file and used by API functions
          Propagate, PropagateNV, and PropagateArrayNV.
          The address table entry for NV_OUTPUT can be turnarouud.
          So, if it is UNBOUND, then we check for turnaround field too.

          primaryIndex is passed for efficiency to avoid recomputation.
          nvIndexIn is always valid.
*****************************************************************/
static Status PropagateThisIndex(int16 nvIndexIn, int16 primaryIndex)
{
  int16      *indexPtr;
  Queue      *indexQPtr;
  uint16     addrIndex;
  NVStruct   *nvStructPtr;
  uint16     nvAliasTableSize;
  char       *valPtr;
  uint16     bufSize;

  nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
```

广东省网络空间安全协会受控资料

```
    nvStructPtr        = GetNVStructPtr(nvIndexIn);
    addrIndex          = nvStructPtr->nvAddrIndex;

    /* If the variable is not output, then we can't propagate. */
    if (nvStructPtr->nvDirection != NV_OUTPUT)
    {
        return(INVALID);
    }

    indexQPtr = &gp->nvOutIndexQ;
    bufSize   = gp->nvOutIndexBufSize;

    if (QueueFull(indexQPtr))
    {
        return(FAILURE); /* Could not schedule all. */
    }

    indexPtr  = QueueTail(indexQPtr);
    *indexPtr = nvIndexIn;
    if (NV_SYNC(primaryIndex))
    {
        /* Copy current value for synchronous variables */
        /* In the queue, the 2 byte index should follow the value */
        valPtr = (char *)(indexPtr + 1);
        if (NV_LENGTH(primaryIndex) <= bufSize - 2)
        {
            memcpy(valPtr, NV_ADDRESS(primaryIndex), NV_LENGTH(primaryIndex));
            EnQueue(indexQPtr);
        }
        else
        {
            return(FAILURE);
        }
    }
    else
    {
        EnQueue(indexQPtr);
    }

    return(SUCCESS);
}

/*****************************************************************
```

广东省网络空间安全协会受控资料

```
Function: PropagateThisPrimary
Returns:  None
Purpose:  To schedule a specific primary network variable.
          If the variable is bound
          then
            this schedules the primary and any alias entries for this
            primary using PropagateThisIndex.
            If nothing was scheduled
            then
                generate failure completion event.
          else
            generate success completion event.
Comment:  After scheduling the primary and all related alias entries,
          this function will add -1 to the queue to indicate end.
*************************************************************************/
void  PropagateThisPrimary(int16 nvIndexIn)
{
   int16      *indexPtr;
   Queue      *indexQPtr;
   NVStruct   *nvStructPtr;
   uint16     nvAliasTableSize;
   uint16     count, dim;
   int16      baseIndex;
   int16      j;
   uint16     queueSpace;

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
   nvStructPtr      = GetNVStructPtr(nvIndexIn);

   indexQPtr = &gp->nvOutIndexQ;

   queueSpace = QueueCnt(indexQPtr) - QueueSize(indexQPtr);

   /* Schedule primary network output variables for NVUpdate. */
   if (IsNVBound(nvIndexIn))
   {
      /* We need space for at least 2 entries to schedule.
         i.e we need to reserve one space for -1 at the end. */

      count = 0;
      if (queueSpace > 1 && PropagateThisIndex(nvIndexIn, nvIndexIn) == SUCCESS)
      {
         count++;
```

广东省网络空间安全协会受控资料

```
        queueSpace--;
    }
    /* Schedule all alias entries that map to this primary entry.
       If queue does not have much space, stop scheduling rest. */
    for (j = nmp->nvTableSize;
         j < nmp->nvTableSize + nvAliasTableSize && queueSpace > 1;
         j++)
    {
        if (GetPrimaryIndex(j) ! = nvIndexIn)
        {
            continue;
        }
        if (PropagateThisIndex(j, nvIndexIn) == SUCCESS)
        {
            count++;
            queueSpace--;
        }
    }
    if (count == 0)
    {
        IsArrayNV(nvIndexIn, &dim, &baseIndex);
        gp->nvArrayIndex = nvIndexIn - baseIndex;
        REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
        NVUpdateCompletes(FAILURE, baseIndex, gp->nvArrayIndex);
    }
    else
    {
        /* Schedule a -1 to indicate end of indices for this primary. */
        /* There should be at least one space left in queue */
        indexPtr  = QueueTail(indexQPtr);
        *indexPtr = -1;
        EnQueue(indexQPtr);
    }
  }
  else
  {
    IsArrayNV(nvIndexIn, &dim, &baseIndex);
    gp->nvArrayIndex = nvIndexIn - baseIndex;
    REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
    NVUpdateCompletes(SUCCESS, baseIndex, gp->nvArrayIndex); /* Not bound. */
  }
}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   Propagate
Returns:    None
Purpose:    To propagate all output network variables.
Comment:      This function is called by the application program to propagate
            all output network variables (declared as polled or not).

            Schedule all NV_OUTPUT indices with a valid address table
            entry regardless of whether it is primary or alias.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call SendVar function to actually send out
            NV Updates messages. If a network output variable is not
            bound, then there is no need to schedule it.

            The addressing information is implicit. If the network
            output variables does not correspond to an address table
            entry, then it can't be scheduled as we don't know how to
            generate the destination address.

            It is possible that not all variables can be scheduled
            due to space limitation in the queues. For guranteed
            scheduling of all possible network ouput variables, the
            nv output queue should be large enough.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void  Propagate(void)
{
  int16     i;

  /* Schedule primary network output variables.  */
  for (i = 0; i < nmp->nvTableSize;  i++)
  {
      PropagateThisPrimary(i);
  }
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   PropagateNV
Returns:    None
Purpose:    To propagate a specific output network variable (simple or array)
Comment:      This function is called by the application program to propagate
```


广东省网络空间安全协会受控资料

```
            a specific network output variable (declared as polled or not).
            If the index corresponds to the first item of an array,
            then all items of the array are propagated.

            If the index does not correspond to first item of an array.
            only that index is propagated.

            The variable must be a primary variable. Alias entries that
            map to this primary (for array, any of the entries), that
            are bound to some valid address table entry. are also
            scheduled.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call SendVar function to actually send out
            NV Update messages.

            There is no gurantee that all possible variables are scheduled
            due to space limitation in the schedule queue. To ensure that
            this does not happen, the queue size for the scheduling should
            be made larger.

            The addressing information is implicit. If the network
            output variables does not correspond to an address table
            entry, then it can't be scheduled as we don't know how to
            generate the destination address.
*****************************************/
void  PropagateNV(int16 nvIndexIn)
{
   uint16     dim;
   int16      baseIndex;
   int16      i;

   if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize )
   {
      return; /* Must be a valid primary index. */
   }


   IsArrayNV(nvIndexIn, &dim, &baseIndex);
   if (nvIndexIn != baseIndex)
   {
      dim = 1; /* nvIndexIn is not the first item of array */
```

```
    }

    /* Scheduled one or more (for array) primary indices.  */
    /* For array, nvIndexIn = baseIndex */
    for (i = nvIndexIn; i < nvIndexIn + dim; i++)
    {
        PropagateThisPrimary(i);
    }
}


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   PropagateArrayNV
Returns:    None
Purpose:    To propagate a specific element of an array output network
            variable or any other network variable (i. e non-array nv).
Comment:     This function is called by the application program to propagate
            a specific item of an array network output variable
            (declared as polled or not) or a simple network variable.

            Once the primary is scheduled, we also schedule alias
            entries that map to this primary to make sure that we
            reach all possible input connections.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call SendVar function to actually send out
            NV Update messages.

            The addressing information is implicit. If the network
            input variables does not correspond to an address table
            entry, then it can be scheduled as we don't know how to
            generate the destination address.

            If arrayNVIndexIn corresponds to the baseindex of an array
            network variable then indexIn is used to compute the specific
            item of the array. Otherwise indexIn is set to 0 so that this
            function will propagate the given index only (a specific array
            item or a simple network variable).
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void  PropagateArrayNV(int16 arrayNVIndexIn, int16 indexIn)
{
    uint16     dim;
```

广东省网络空间安全协会受控资料

```
    int16      baseIndex;
    int16      nvIndex;

    if (arrayNVIndexIn < 0 || arrayNVIndexIn >= nmp->nvTableSize)
    {
        return; /* Invalid index. */
    }

    /* if arrayNVIndexIn is not an array variable, then dim is set to 1
       and baseIndex is set to arrayNVIndexIn by IsArrayNV function */
    if (! IsArrayNV(arrayNVIndexIn, &dim, &baseIndex) )
    {
        indexIn = 0; /* Simple network variable */
    }
    else if (baseIndex ! = arrayNVIndexIn)
    {
        indexIn = 0; /* Any other array item. don't use indexIn passed. */
    }

    if (indexIn < 0 || indexIn >= dim)
    {
        return; /* Invalid index */
    }

    nvIndex = arrayNVIndexIn + indexIn;
    PropagateThisPrimary(nvIndex);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   SendVar
Returns:    None.
Purpose:    Generates NV Update message for the given index.
            The given index can be either primary or alias.
            For sync network output variables, we are also
            given the corresponding value. For nonsync variables,
            valPtr is null and current value is used.

            If the network output variable has nvTurnaround on,
            then we need to search for a network input variable with
            matching selector number. Once found, we update it, and
            send NVUpdateOccurs event to the application program.
            Note that we search through both primary and alias
            entries for the turnaround. If a primary is connected to this
```

广东省网络空间安全协会受控资料

```
        output variables, we will find it eventually either direcly
        or indirectly (via alias).

Comments:
        We need to handle only the given index. Propagate functions
        take care of scheduling all needed alias indices properly.
        Implicit NV Updates are handleded through Propagate.

        It is possible that variables are scheduled, but by the time
        they are processed in this function, the application is
        offline or the node is unconfigured. So, check for this
        too before updating network variables for turnaround.

        Also, it is possible that some attributes of the variable
        has been changed by a network management tool after the
        index was scheduled.

        If the index is -1, then it represents the end of scheduling for a
        primary. Generate a message to transport layer with a special tag
        that will be recognized and sent back immediately in the
        indication to application layer.
**************************************************************/
static void SendVar()
{
    Queue           *indexQPtr;
    int16           i;              /* For loop. */
    uint16          bufSize;        /* bufSize for the target queue */
    int16           nvIndex;
    int16           primaryIndex;   /* For nvIndex. */
    uint16          nvLength;       /* For nvIndex. */
    uint16          selector;       /* For nvIndex. */
    Byte            *nvPtr;         /* For nvIndex. Points to storage. */
    uint16          dimIn;          /* For input network variable */
    int16           primaryIndexIn; /* For input network variable */
    int16           baseIndexIn;    /* For input network variable */
    uint16          nvLengthIn;     /* For input network variable */
    Byte            *nvPtrIn;       /* For input network variable */
    uint16          selectorIn;     /* For input network variable */
    Queue           *nwOutQPtr, *tsaOutQPtr;
    TSASendParam    *tsaSendParamPtr;
    NWSendParam     *nwSendParamPtr;
    APDU            *apduPtr;
    uint16          nvAliasTableSize;
```


广东省网络空间安全协会受控资料

```
NVStruct        * nvStrPtr, * nvStrPtrIn;
uint16          addrIndex;
AddrTableEntry  * ap;
Boolean         turnAroundOnly; /* does not mean turnaround for sure. Means that
                                   the variable does not have addr table entry
                                   or the address table entry is unbound or
                                   it is turnaround entry. */
Boolean         foundMatchingInputVar;
int16           * indexPtr;
char            * valPtr;

indexQPtr = &gp->nvOutIndexQ;

if (! gp->nvOutCanSchedule || QueueEmpty(indexQPtr))
{
    return; /* Nothing to do. */
}
indexPtr  = QueueHead(indexQPtr);
nvIndex   = * indexPtr;
valPtr    = (char *)(indexPtr + 1);

/* If the node enters unconfigued state before processing this nv update
   we do not want to schedule these indices. We simply do nothing and wait
   for the node to go configured. */
if (eep->readOnlyData.nodeState == APPL_UNCNFG)
{
    return;
}

nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

/* nvIndexIn > 0 => nv variable. nvIndex == -1 => end of scheduling */
if (nvIndex >= 0)
{
    /* Make nvStrPtr point to the right NVStruct for nvIndexIn */
    nvStrPtr      = GetNVStructPtr(nvIndex);
    primaryIndex = GetPrimaryIndex(nvIndex);
    nvPtr         = NV_ADDRESS(primaryIndex);
    nvLength      = NV_LENGTH(primaryIndex);
    selector      = (nvStrPtr->nvSelectorHi << 8) | nvStrPtr->nvSelectorLo;
    addrIndex     = nvStrPtr->nvAddrIndex;
    /* The variable is turnaround only if addrIndex is 0xF or the address
       table entry is unbound (turnaround or not). */
```

广东省网络空间安全协会受控资料

```
/* *** START INFORMATIVE — Unbound Network Variable */
/* It is acceptable to require that turnaround NVs have an address table
 * entry assigned.  This entry may be used for determining retry counts
 * and tx_timer values for retrying in the event of resource problems such
 * as input buffer inavailability.  Network management tools are expected
 * to assign an address table entry even if using unackd service. */
/* *** END INFORMATIVE — Unbound Network Variable */
turnAroundOnly  = addrIndex == 0xF ||
                  eep->addrTable[addrIndex].addrFormat == UNBOUND;

if(nvStrPtr->nvPriority)
{
   tsaOutQPtr = &gp->tsaOutPriQ;
   nwOutQPtr  = &gp->nwOutPriQ;
   if (nvStrPtr->nvService == UNACKD)
   {
      bufSize = gp->nwOutPriBufSize;
   }
   else
   {
      bufSize = gp->tsaOutPriBufSize;
   }
}
else
{
   tsaOutQPtr = &gp->tsaOutQ;
   nwOutQPtr  = &gp->nwOutQ;
   if (nvStrPtr->nvService == UNACKD)
   {
      bufSize = gp->nwOutBufSize;
   }
   else
   {
      bufSize = gp->tsaOutBufSize;
   }
}
}
else
{
   /* Let us use tsaOutQ for the special message */
   tsaOutQPtr = &gp->tsaOutQ;
}
```

广东省网络空间安全协会受控资料

```
if (nvIndex == -1 || ! turnAroundOnly)
{
    /* We need to make sure that we have space in transport or network
       layer. If there is no space, we should return without doing any
       processing.
       NV variable messages can be ACK, UNACK, UNACKD_RPT */
    if (nvIndex == -1)
    {
       if (QueueFull(tsaOutQPtr))
       {
           return;
       }
    }
    else if(nvStrPtr->nvService == UNACKD && QueueFull(nwOutQPtr))
    {
       return;
    }
    else if(nvStrPtr->nvService ! = UNACKD && QueueFull(tsaOutQPtr))
    {
       return;
    }
}

if (nvIndex == -1)
{
    /* Form a message with a special tag to transport layer. */
    tsaSendParamPtr                = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service       = ACKD;
    tsaSendParamPtr->tag           = NV_UPDATE_LAST_TAG_VALUE;
    tsaSendParamPtr->apduSize      = 0;
    EnQueue(tsaOutQPtr);
    gp->nvOutCanSchedule = FALSE; /* Only one at a time. */
    DeQueue(indexQPtr);
    return;
}


/* If the variable is flagged as turnaround, then we look for first
    network input variable with matching selector number. Once found,
    we update that input variable, and then send NVUpdateOccurs event
    to the application program. We break as soon as first match is found. */
if (nvStrPtr->nvTurnaround)
```

广东省网络空间安全协会受控资料

```
{
    foundMatchingInputVar = FALSE;
    for(i = 0; i < nmp->nvTableSize+nvAliasTableSize; i++)
    {
        nvStrPtrIn   = GetNVStructPtr(i);
        selectorIn   =
                (nvStrPtrIn->nvSelectorHi << 8) | nvStrPtrIn->nvSelectorLo;
        /* If this variable is not input or does not have matching selector,
           then skip this entry */
        if (nvStrPtrIn->nvDirection ! = NV_INPUT ||
            selector ! = selectorIn)
        {
            continue; /* Not a matching entry */
        }
        /* Found a matching turnaroud entry for nvIndexIn */
        primaryIndexIn = GetPrimaryIndex(i);
        nvPtrIn          = NV_ADDRESS(primaryIndexIn);
        nvLengthIn       = NV_LENGTH(primaryIndexIn);
        if (nvLength ! = nvLengthIn)
        {
            break; /* selector matches but length does not */
        }
        memcpy(nvPtrIn, nvPtr, nvLength);
        foundMatchingInputVar = TRUE;
        /* Notify application if it is running */
        if (AppPgmRuns())
        {
            IsArrayNV(primaryIndexIn, &dimIn, &baseIndexIn);

            gp->nvInAddr.format = 4; /* TURNAROUND */
            memset(&gp->nvInAddr.srcAddr, 0, sizeof(SubnetAddress));
            gp->nvInAddr.domain = 0; /* Not relevant */
            gp->nvArrayIndex     = primaryIndexIn - baseIndexIn;
            REF_IMP_TO_ALT(NV_IN_ADDR, &nv_in_addr);
            REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
            NVUpdateOccurs(baseIndexIn, gp->nvArrayIndex);
        }

        break; /* break after first match */
    } /* for */
} /* if */

if (turnAroundOnly)
```

广东省网络空间安全协会受控资料

```
{
    /* Completion event is given in HandleMsgCompletion function */
    /* Since there is nothing actually scheduled, no need to clear
       gp->nvOutCanSchedule flag. */
    /* Since this index does not go through HandleMsgCompletion, we need
       to set gp->nvOutIndex here. This is to take care of the case when
       a variable is turndaround only with no alias. In this case this index
       is followed by -1 in the queue. Hence, the gp->nvOutIndex would never
       be initialized when HandleMsgCompletion gets NV_UPDATE_LAST_TAG_VALUE.
       Explicit initialization here will fix the problem. */
    gp->nvOutIndex = GetPrimaryIndex(nvIndex);
    DeQueue(indexQPtr);
    return;
}

gp->nvOutCanSchedule = FALSE; /* Only one index at a time */
DeQueue(indexQPtr);

/* Build and send network variable update message. */
ap = AccessAddress(addrIndex); /* ap can't be null. */
/* Fail if we don't have sufficient space in the target queue. */
if (2 + nvLength > bufSize)
{
    /* Discard this index as the space is not sufficient */
    return;
}

if (nvStrPtr->nvService ! = UNACKD)
{
    /* The message should go to the transport layer */
    tsaSendParamPtr                = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->domainIndex = COMPUTE_DOMAIN_INDEX;
    *(AddrTableEntry *)(&tsaSendParamPtr->destAddr) = *ap;
    tsaSendParamPtr->service       = nvStrPtr->nvService;
    tsaSendParamPtr->auth          = nvStrPtr->nvAuth;
    /* See app.h for information on tag usage. */
    tsaSendParamPtr->tag           = GET_NV_UPDATE_TAG(primaryIndex);
    apduPtr                        = (APDU *)(tsaSendParamPtr + 1);
    apduPtr->code.nv.nvFlag        = 0x1;
    apduPtr->code.nv.nvDir         = NV_INPUT;
    apduPtr->code.nv.nvCode        = nvStrPtr->nvSelectorHi;
    apduPtr->data[0]               = nvStrPtr->nvSelectorLo;
```

广东省网络空间安全协会受控资料

```
    tsaSendParamPtr->apduSize      = 2 + nvLength;
    if (NV_SYNC(primaryIndex))
    {
       /* Send value given */
       memcpy(&apduPtr->data[1], valPtr, nvLength);
    }
    else
    {
       /* Send the most current value */
       memcpy(&apduPtr->data[1], nvPtr, nvLength);
    }

    EnQueue(tsaOutQPtr);
    return;
}

/* The message is for the network layer */
nwSendParamPtr = QueueTail(nwOutQPtr);
nwSendParamPtr->dropIfUnconfigured = TRUE;
nwSendParamPtr->tag = GET_NV_UPDATE_TAG(primaryIndex);

switch(ap->addrFormat)
{
    case SUBNET_NODE:
       nwSendParamPtr->destAddr.addressMode = SUBNET_NODE;
       nwSendParamPtr->destAddr.domainIndex =
                    ap->snodeEntry.domainIndex;
       nwSendParamPtr->destAddr.addr.addr2a.subnet =
                    ap->snodeEntry.subnetID;
       nwSendParamPtr->destAddr.addr.addr2a.node    =
                    ap->snodeEntry.node;
       break;
    case BROADCAST:
       nwSendParamPtr->destAddr.addressMode = BROADCAST;
       nwSendParamPtr->destAddr.domainIndex =
                ap->bcastEntry.domainIndex;
       nwSendParamPtr->destAddr.addr.addr0 =
                ap->bcastEntry.subnetID;
       break;

    default:
       /* Since the address table entry can't be unbound or turnaround,
          we must have group entry */
```

广东省网络空间安全协会受控资料

```
        nwSendParamPtr->destAddr.addressMode = MULTICAST;
        nwSendParamPtr->destAddr.domainIndex =
                ap->groupEntry.domainIndex;
        nwSendParamPtr->destAddr.addr.addr1 =
                ap->groupEntry.groupID;
    } /* switch */

    nwSendParamPtr->pduType       = APDU_TYPE;
    nwSendParamPtr->deltaBL       = 0; /* No ack generated */
    nwSendParamPtr->altPath       = FALSE;
    nwSendParamPtr->pduSize       = 2 + nvLength;
    apduPtr                       = (APDU *)(nwSendParamPtr + 1);
    apduPtr->code.nv.nvFlag       = 0x1;
    apduPtr->code.nv.nvDir        = NV_INPUT;
    apduPtr->code.nv.nvCode       = nvStrPtr->nvSelectorHi;
    apduPtr->data[0]              = nvStrPtr->nvSelectorLo;
    if (NV_SYNC(primaryIndex))
    {
        /* Send value given. */
        memcpy(&apduPtr->data[1], valPtr, nvLength);
    }
    else
    {
        /* Send the most current value. */
        memcpy(&apduPtr->data[1], NV_ADDRESS(primaryIndex), nvLength);
    }
    EnQueue(nwOutQPtr);

    return;
}

/*****************************************************************
Function: PollThisIndex
Returns:  SUCCESS if the index is scheduled.
          FAILURE if the queue is full and hence not scheduled.
          INVALID if the index does not correspond to NV_INPUT.
Purpose:  To schedule a specific index of a network variable
          (primary or alias).
Comment:
          This function is local to this file and used by API functions
          Poll, PollNV, and PollArrayNV.
          nvIndexIn is always valid.
*****************************************************************/
```

广东省网络空间安全协会受控资料

```
static Status PollThisIndex(int16 nvIndexIn)
{
   int16      * indexPtr;
   Queue       * indexQPtr;
   uint16      addrIndex;
   NVStruct  * nvStructPtr;
   uint16      nvAliasTableSize;

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
   nvStructPtr        = GetNVStructPtr(nvIndexIn);
   addrIndex          = nvStructPtr->nvAddrIndex;
   if (nvStructPtr->nvDirection ! = NV_INPUT)
   {
      return(INVALID);
   }

   indexQPtr = &gp->nvInIndexQ;

   if (QueueFull(indexQPtr))
   {
      return(FAILURE); /* Could not schedule all. */
   }
   indexPtr  = QueueTail(indexQPtr);
   * indexPtr = nvIndexIn;
   EnQueue(indexQPtr);
   return(SUCCESS);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function: PollThisPrimary
Returns:  None
Purpose:  To schedule a specific primary network variable.
          If the variable is bound
          then
            this schedules the primary and any alias entries for this
            primary using PollThisIndex.
            If nothing was scheduled
            then
              generate failure completion event.
          else
            generate success completion event.
Comment:  None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

广东省网络空间安全协会受控资料

```
void  PollThisPrimary(int16 nvIndexIn)
{
   int16      * indexPtr;
   Queue      * indexQPtr;
   NVStruct  * nvStructPtr;
   uint16     nvAliasTableSize;
   uint16     count, dim;
   int16      baseIndex;
   int16      j;
   uint16     queueSpace;

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
   nvStructPtr      = GetNVStructPtr(nvIndexIn);

   indexQPtr = &gp->nvInIndexQ;

   queueSpace = QueueCnt(indexQPtr) - QueueSize(indexQPtr);

   /* Schedule primary network input variables for poll */
   if (IsNVBound(nvIndexIn))
   {
      /* We need space for at least 2 entries to schedule.
      i.e we need to reserve one space for -1 at the end. */
      count = 0;
      if (queueSpace > 1 && PollThisIndex(nvIndexIn) == SUCCESS)
      {
         count++;
         queueSpace--;
      }
      /* Schedule all alias entries that map to this primary entry.
         If queue does not have much space, stop scheduling rest. */
      for (j = nmp->nvTableSize;
            j < nmp->nvTableSize + nvAliasTableSize && queueSpace > 1;
            j++)
      {
         if (GetPrimaryIndex(j) ! = nvIndexIn)
         {
            continue;
         }
         if (PollThisIndex(j) == SUCCESS)
         {
            count++;
            queueSpace--;
```

广东省网络空间安全协会受控资料

```
        }
    }
    if (count == 0)
    {
        IsArrayNV(nvIndexIn, &dim, &baseIndex);
        gp->nvArrayIndex = nvIndexIn - baseIndex;
        REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
        NVUpdateCompletes(FAILURE, baseIndex, gp->nvArrayIndex);
    }
    else
    {
        /* Schedule -1 to indicate end of indices for this primary. */
        /* There should be at least one space left in queue */
        indexPtr  = QueueTail(indexQPtr);
        *indexPtr = -1;
        EnQueue(indexQPtr);
    }
  }
  else
  {
    IsArrayNV(nvIndexIn, &dim, &baseIndex);
    gp->nvArrayIndex = nvIndexIn - baseIndex;
    REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
    NVUpdateCompletes(SUCCESS, baseIndex, gp->nvArrayIndex); /* Not bound. */
  }
}

/*********************************************************************
Function:   Poll
Returns:    None
Purpose:    To poll all input network variables.
Comment:    This function is called by the application program to poll
            all network input variables.

            Schedule all NV_INPUT indices with a valid address table
            entry regardless of whether it is primary or alias.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call PollVar function to actually send out
            polling requests.

            It is possible that not all variables can be scheduled
```

广东省网络空间安全协会受控资料

```
            due to space limitation in the queues. For guranteed
            scheduling of all possible network input variables, the
            polling queue should be large enough.

            NVUpdateCompletes for input variables means that they
            are completion indication of corresponding poll requests.
*****************************************************************/
void  Poll(void)
{
   int16       i;

   /* Schedule primary network input variables for poll */
   for (i = 0; i < nmp->nvTableSize;  i++)
   {
      PollThisPrimary(i);
   }
}

/*****************************************************************
Function:   PollNV
Returns:    None
Purpose:    To poll a specific input network variable (simple or array)
Comment:    This function is called by the application program to poll
            a specific network input variable.
            If the index corresponds to the first item of an array,
            then all items of the array are polled.

            If the index does not correspond to first item of an array.
            only that index is polled.

            The variable must be a primary variable. All alias entries
            for this primary are also scheduled.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call PollVar function to actually send out
            polling requests.

            There is no gurantee that all possible variables are scheduled
            due to space limitation in the schedule queue. To ensure that
            this does not happen, the queue size for the scheduling should
            be made larger.
```

广东省网络空间安全协会受控资料

```
                The addressing information is implicit. If the network
                input variables does not correspond to an address table
                entry, then it can't be scheduled as we don't know how to
                generate the destination address.

                NVUpdateCompletes for input variables means that they
                are completion indication of corresponding poll requests.
**************************************************************************/
void  PollNV(int16 nvIndexIn)
{
   uint16     dim;
   int16      baseIndex;
   int16      i;

   if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize )
   {
      return; /* Must be a valid primary index. */
   }


   IsArrayNV(nvIndexIn, &dim, &baseIndex);
   if (nvIndexIn != baseIndex)
   {
      dim = 1; /* nvIndexIn is not the first item of array */
   }

   /* Scheduled one or more (for array) primary indices. */
   /* For array, nvIndexIn = baseIndex */
   for (i = nvIndexIn; i < nvIndexIn + dim; i++)
   {
      PollThisPrimary(i);
   }
}

/*************************************************************************
Function:   PollArrayNV
Returns:    None
Purpose:    To poll a specific element of an array input network variable
            or any other network variable (i.e non-array).
Comment:    This function is called by the application program to poll
            a specific item of an array network input variable or any
            other network variable (non-array).
```

广东省网络空间安全协会受控资料

```
            Once the primary is schedules, we also schedule alias
            entries that map to this primary to make sure that we
            reach all possible output connections.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call PollVar function to actually send out
            polling requests.

            The addressing information is implicit. If the network
            input variables does not correspond to an address table
            entry, then it can be scheduled as we don't know how to
            generate the destination address.

            If arrayNVIndexIn corresponds to the base index of an array,
            the indexIn is used to get the specific item of the array to
            be polled. Othewise, indexIn is set to 0 so that any other
            array item or any other network variable can be passed.

            NVUpdateCompletes for input variables means that they
            are completion indication of corresponding poll requests.
*******************************************************************/
void  PollArrayNV(int16 arrayNVIndexIn, int16 indexIn)
{
   uint16     dim;
   int16      baseIndex;
   int16      nvIndex;


   if (arrayNVIndexIn < 0 || arrayNVIndexIn >= nmp->nvTableSize)
   {
       return; /* Invalid index. */
   }

   /* if arrayNVIndexIn is not an array variable, then dim is set to 1
       and baseIndex is set to arrayNVIndexIn by IsArrayNV function */
   if (! IsArrayNV(arrayNVIndexIn, &dim, &baseIndex))
   {
       indexIn = 0; /* Simple network variable */
   }
   else if (baseIndex != arrayNVIndexIn)
   {
       indexIn = 0; /* Any other array item. don't use indexIn passed. */
```

广东省网络空间安全协会受控资料

```
    }

    if (indexIn < 0 || indexIn >= dim)
    {
        return; /* Invalid index */
    }

    nvIndex = arrayNVIndexIn + indexIn;
    PollThisPrimary(nvIndex);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   PollVar
Returns:    None
Reference:  Technology Device Data Book Rev 3. page 9—54.
Purpose:    Issues a poll message for a single network input variable.
            Sends NV Poll message for the connected output variables.

            If there exists a unique turnaround output, we update
            the network variable directly and send NVUpdateOccurs event
            to the application program. Thus, updates from externally
            connected output variables will arrive later.

Comments:   Note that this return value does not mean that the
            poll transaction is complete. It only means that the
            polling request has been delivered to the session layer.

            nvIndexIn must be good as it was previously scheduled.
            nvIndexIn can be either primary or alias.

            Do not check turnaround entries unless the session layer
            has space in the request queue for scheduling the poll.
            Otherwise, we may do the turnaround but stuck with not
            being able send the poll request. If we keep the entry
            in the queue after turnaround entries are processed, we
            may end up with double processing of turnaround entries.

            The NV Variable Poll message must be Request. So, use
            session layer.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
static void PollVar(void)
{
    Queue           *indexQPtr;
```


广东省网络空间安全协会受控资料

```
int16           nvIndex;
int16           * indexPtr;
int16           i;          /* For loop. */
int16           primaryIndex; /* For nvIndex. */
uint16          nvLength;     /* For nvIndex. */
uint16          dim;          /* For nvIndex, if it is array */
int16           baseIndex;    /* For nvIndex, if it is array */
uint16          selector;     /* For nvIndex. */
Byte            * nvPtr;        /* For nvIndex. Points to storage. */
int16           primaryIndexOut; /* For output network variable */
uint16          nvLengthOut;     /* For output network variable */
Byte            * nvPtrOut;        /* For output network variable */
uint16          selectorOut;     /* For output network variable */
Queue           * tsaOutQPtr;
TSASendParam    * tsaSendParamPtr;
APDU            * apduPtr;
uint16          nvAliasTableSize;
NVStruct        * nvStrPtr, * nvStrPtrOut;
uint16          addrIndex, addrIndexOut;
AddrTableEntry  * ap;
Boolean         turnAroundOnly; /* does not mean turnaround for sure. Means that
                                   the variable does not have addr table entry
                                   or the address table entry is unbound or
                                   it is turnaround entry. */
Boolean         foundMatchingOutputVar;
int16           matchingIndexOut;

indexQPtr = &gp->nvInIndexQ;

if (! gp->nvInCanSchedule || QueueEmpty(indexQPtr))
{
    return;
}
indexPtr  = QueueHead(indexQPtr);
nvIndex = * indexPtr;

/* If the node enters unconfigued state before processing this nv poll
   we do not want to schedule these indices. We simply do nothing and wait
   for the node to go configured. */
if (eep->readOnlyData.nodeState == APPL_UNCNFG)
{
    return;
}
```

广东省网络空间安全协会受控资料

```
nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

if (nvIndex >= 0)
{
    /* Make nvStrPtr point to the right NVStruct for nvIndex */
    nvStrPtr      = GetNVStructPtr(nvIndex);
    /* START INFORMATIVE — Network Variable Alias Priority */
    /* The bracketed code in conjunction with other code in this implementation
     * is structured in a way that allows network variable aliases
     * to have a different priority attribute than the primary and for that
     * difference in priority to be honored.  It is acceptable, however, to send
     * all aliases using the same priority attribute as that of the primary. */
    if(nvStrPtr->nvPriority)
    {
       tsaOutQPtr = &gp->tsaOutPriQ;
    }
    else
    {
       tsaOutQPtr = &gp->tsaOutQ;
    }
    /* END INFORMATIVE — Network Variable Alias Priority */

    primaryIndex = GetPrimaryIndex(nvIndex);
    nvPtr        = NV_ADDRESS(primaryIndex);
    nvLength     = NV_LENGTH(primaryIndex);
    selector     = (nvStrPtr->nvSelectorHi << 8) | nvStrPtr->nvSelectorLo;
    addrIndex    = nvStrPtr->nvAddrIndex;
    /* The variable is turnaround only if addrIndex is 0xF or it is unbound
       (turnaround or not) */
    turnAroundOnly  = (addrIndex == 0xF) ||
                      (eep->addrTable[addrIndex].addrFormat == UNBOUND);
}
else
{
    tsaOutQPtr = &gp->tsaOutQ;
}
/* If the address table entry is turnaround only, then we don't send out
    any nv poll messages and hence we don't need to check the queue
    for space availability */
if(nvIndex == -1 || ! turnAroundOnly)
{
    if (QueueFull(tsaOutQPtr))
```

广东省网络空间安全协会受控资料

```
    {
        /* Can't send request yet — try later. Don't even try turnaround. */
        return;
    }
}

if (nvIndex == -1)
{
    /* Form a message with a special tag to transport layer. */
    tsaSendParamPtr                = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service       = ACKD;
    tsaSendParamPtr->tag           = NV_POLL_LAST_TAG_VALUE;
    tsaSendParamPtr->apduSize      = 0;
    EnQueue(tsaOutQPtr);
    gp->nvInCanSchedule = FALSE; /* Only one at a time. */
    DeQueue(indexQPtr);
    return;
}

/* If nvIndex is part of an array, we need to get the dimension and
    the baseIndex so that we can report nvInAddr and nvArrayIndex
    properly, in case there is a turn around connected. Alias entries
    can't be used to determine whether it is array. Use primaryIndex.
    We will use baseIndex for NVUpdateOccurs event */
IsArrayNV(primaryIndex, &dim, &baseIndex);

/* If the variable is flagged as turnaround, then we look for first
    network output variables with matching selector number. Once found,
    we update this input variable, and then send NVUpdateOccurs event
    to the application program. */
if (nvStrPtr->nvTurnaround)
{
    foundMatchingOutputVar = FALSE;
    matchingIndexOut = -1;
    for(i = 0; i < nmp->nvTableSize+nvAliasTableSize; i++)
    {
        nvStrPtrOut   = GetNVStructPtr(i);
        addrIndexOut = nvStrPtrOut->nvAddrIndex;
        /* Skip input network variables */
        if (nvStrPtrOut->nvDirection == NV_INPUT)
        {
            continue;
```

广东省网络空间安全协会受控资料

```
        }
        /* It is a network output variable. Match selector */
        selectorOut =
            (nvStrPtrOut->nvSelectorHi << 8) | nvStrPtrOut->nvSelectorLo;
        if (selector != selectorOut)
        {
            continue; /* Selector does not match */
        }
        /* Found a matching output variable for nvIndex */
        matchingIndexOut = i;
        break;
    }

    if (matchingIndexOut != -1)
    {
        primaryIndexOut = GetPrimaryIndex(matchingIndexOut);
        nvPtrOut        = NV_ADDRESS(primaryIndexOut);
        nvLengthOut     = NV_LENGTH(primaryIndexOut);

        /* Update the input variable provided the length matches */
        if (nvLength == nvLengthOut)
        {
            /* We will skip authentication check as it is turnaround */
            memcpy(nvPtr, nvPtrOut, nvLength);
            foundMatchingOutputVar = TRUE;

            /* turnaround updates are ignored for valid data check for polls.
               The only time it is ignored is if the nv is turnaround only.
               So, do not update gp->nvInDataStatus flag here. */

            /* Notify application program if it is running. */
            if (AppPgmRuns())
            {
                gp->nvInAddr.format = 4; /* TURNAROUND */
                memset(&gp->nvInAddr.srcAddr, 0, sizeof(SubnetAddress));
                gp->nvInAddr.domain = 0; /* Not relevant */
                /* gp->nvArrayIndex is 0 for simple var */
                gp->nvArrayIndex     = primaryIndex - baseIndex;
                REF_IMP_TO_ALT(NV_IN_ADDR, &nv_in_addr);
                REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
                /* same as primaryIndex for simple var */
                NVUpdateOccurs(baseIndex, gp->nvArrayIndex);
            }
```

广东省网络空间安全协会受控资料

```
        }
    }
}

if (turnAroundOnly)
{
    /* NV completion event is given in HandleMsgCompletion. */
    /* Don't clear the nvInCanSchedule flag as we can continue */
    /* Since this index does not go through HandleMsgCompletion, we need
       to set gp->nvInIndex here. This is to take care of the case when
       a variable is turndaround only with no alias. In this case, this index
       is followed by -1 in the queue. Hence, the gp->nvInIndex would never
       be initialized when HandleMsgCompletion gets NV_POLL_LAST_TAG_VALUE.
       Explicit initialization here will fix the problem. */
    gp->nvInIndex = GetPrimaryIndex(nvIndex);
    if (nvStrPtr->nvTurnaround && matchingIndexOut != -1)
    {
        /* We did find a matching output variable and updated the polled variable */
        /* Note that even if one of the indices (primary or alias) is turnaround
           only, this flag is set to true. */
        gp->nvInDataStatus = SUCCESS; /* to enable poll to succeed */
    }
    DeQueue(indexQPtr);
    return;
}

gp->nvInCanSchedule = FALSE;
DeQueue(indexQPtr);

/* Build and send netvar poll message. It is a REQUEST message. */
ap = AccessAddress(addrIndex); /* ap can't be null */

tsaSendParamPtr                = QueueTail(tsaOutQPtr);
tsaSendParamPtr->altPathOverride = FALSE;
tsaSendParamPtr->domainIndex = COMPUTE_DOMAIN_INDEX;
*(AddrTableEntry *)(&tsaSendParamPtr->destAddr) = *ap;
tsaSendParamPtr->service       = REQUEST; /* Poll Message */
tsaSendParamPtr->auth          = nvStrPtr->nvAuth;
/* See app.h for details on tag usage. */
tsaSendParamPtr->tag           = GET_NV_POLL_TAG(primaryIndex);
apduPtr                        = (APDU *)(tsaSendParamPtr + 1);
apduPtr->code.nv.nvFlag        = 0x1;
apduPtr->code.nv.nvDir         = NV_OUTPUT;
```

广东省网络空间安全协会受控资料

```
    apduPtr->code.nv.nvCode          = nvStrPtr->nvSelectorHi;
    apduPtr->data[0]                 = nvStrPtr->nvSelectorLo;
    tsaSendParamPtr->apduSize        = 2;

    EnQueue(tsaOutQPtr);
    return;
}

/*****************************************************************
Function: NewMsgTag
Returns:  A new message tag of the requested type (bindable or non-bindable).
Purpose:  To allocate message tags for use by application program.
Comment:  Reference implementation does not support rate information
          for tags.
*****************************************************************/
MsgTag NewMsgTag(BindNoBind bindStatusIn)
{
    if (bindStatusIn == BIND)
    {
        if (gp->nextBindableMsgTag < NUM_ADDR_TBL_ENTRIES &&
            nmp->snvt.mtagCount < 0xFF)
        {
            nmp->snvt.mtagCount++;
            return(gp->nextBindableMsgTag++);
        }
        else
        {
            /* Ran out of addr table entries or no room in mtagCount. */
            return(-1);
        }
    }

    if (gp->nextNonbindableMsgTag < 0xFFFF)
    {
        return(gp->nextNonbindableMsgTag++); /* We have lots of this */
    }
    else
    {
        return(-1); /* Extremely unlikely to happen. */
    }
}

/*****************************************************************
```

广东省网络空间安全协会受控资料

```
Function:  IsArrayNV
Returns:   Returns TRUE if a given primary index corresponds to a
           network array variable.
           The index can be any of the indices for the array
           elements.
Reference: None
Purpose:   To determine if an index corresponds to a network
           array variable. This is needed as this info is not part
           of the nv config or fixed tables.
           For simple variables, dimOut is set to 1 and baseIndexOut
           is set to the given index. (By default).
Comments:   gp->nvArrayTbl is an array of structures with two
           fields: nvIndex and dim. The table contains nvIndex
           and dim only for array variables. The table is populated
           during AddNV.
           For array variables, dim (dimension of array) and array
           base index passed back if provided with non-null addresses.
***************************************************************/
static Boolean IsArrayNV(int16 nvIndexIn,
                         uint16 *dimOut, int16 *baseIndexOut)
{
  int16 i;

  if (dimOut)
  {
     *dimOut = 1; /* Default for simple variables */
  }
  if (baseIndexOut)
  {
     *baseIndexOut = nvIndexIn;
  }
  /* Sequential search. OK if there are not too many arrays. */
  if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize)
  {
     return(FALSE); /* Only primary variables can be arrays */
  }
  for (i = 0; i < gp->nvArrayTblSize; i++)
  {
     if (nvIndexIn >= gp->nvArrayTbl[i].nvIndex &&
        nvIndexIn <  gp->nvArrayTbl[i].nvIndex + gp->nvArrayTbl[i].dim)
     {
       /* Lies in the range for this array variable */
       if (dimOut)
```

广东省网络空间安全协会受控资料

```
            {
                * dimOut    = gp->nvArrayTbl[i].dim;
            }
            if (baseIndexOut)
            {
                * baseIndexOut = gp->nvArrayTbl[i].nvIndex;
            }
            return(TRUE);
        }
    }
    return(FALSE); /* Not found. Must be a simple variable */
}

/* Application can call this fn to put itself offline */
void GoOffline(void)
{
    OfflineEvent();
    gp->appPgmMode = OFF_LINE;
}

/* Appplication can call this fn to put itself unconfigured */
void GoUnconfigured(void)
{
    int i, numDomains;

    eep->readOnlyData.nodeState = APPL_UNCNFG;
    /* Set appPgmMode to OFF_LINE so that when we configured again, the
       application program will be soft-off-line.
       See P9—47 of Tech Device Data Book Rev 3 */
    gp->appPgmMode                  = OFF_LINE;
    if (eep->readOnlyData.twoDomains)
    {
        numDomains = 2;
    }
    else
    {
        numDomains = 1;
    }

    /* Overwrite all domain information. Destroys auth key */
    for (i = 0; i < numDomains; i++)
    {
        memcpy(eep->domainTable[i].domainId, "gmrdwf", DOMAIN_ID_LEN);
```

广东省网络空间安全协会受控资料

```
        eep ->domainTable[i].subnet = 0;
        eep ->domainTable[i].cloneDomain = 0;
        eep ->domainTable[i].node = 0;
        eep ->domainTable[i].len = 0xFF;
        memcpy(eep ->domainTable[i].key, "NO_KEY", AUTH_KEY_LEN);
    }
}

/*-----------------------End of app.c-----------------------*/
```

A.10 网络管理命令

```
/*****************************************************
        File:        netmgmt.c

     Version:        1.7

   Reference:        Section 12, Protocol Specification.

     Purpose:        App Layer/Network Management

                     The functions in this file handle the network
                     management messages (HandleNM()) and network
                     diagnostic messages (HandleND()).

        Note:
                          Discard if     Honor even if     Never
                          Not Request    read/write prot   Authcated
                          -----          ---------         --------
Network Management Messages:
NM_QUERY_ID            0x61   YES          YES              YES
NM_RESPOND_TO_QUERY    0x62   no           YES              YES
NM_UPDATE_DOMAIN       0x63   no           YES              no
NM_LEAVE_DOMAIN        0x64   no           YES              no
NM_UPDATE_KEY          0x65   no           YES              no
NM_UPDATE_ADDR         0x66   no           YES              no
NM_QUERY_ADDR          0x67   YES          YES              no
NM_QUERY_NV_CNFG       0x68   YES          YES              no
NM_UPDATE_GROUP_ADDR   0x69   no           YES              no
NM_QUERY_DOMAIN        0x6A   YES          YES              no
NM_UPDATE_NV_CNFG      0x6B   no           YES              no
```

广东省网络空间安全协会受控资料

```
NM_SET_NODE_MODE        0x6C   no          YES          no
NM_READ_MEMORY          0x6D   YES         Limited      no
NM_WRITE_MEMORY         0x6E   no          Limited      no
NM_CHECKSUM_RECALC      0x6F   no          YES          no
NM_WINK                 0x70   no          YES          no
NM_MEMORY_REFRESH       0x71   no          YES          no
NM_QUERY_SNVT           0x72   YES         YES          no
NM_NV_FETCH             0x73   YES         YES          no

Network Diagnostic Messages:
ND_QUERY_STATUS         0x51   YES         YES          YES
ND_PROXY_COMMAND        0x52   YES         YES          YES
ND_CLEAR_STATUS         0x53   NO          YES          no
ND_QUERY_XCVR           0x54   YES         YES          no

Manual Service Request Message:
NM_MANUAL_SERVICE_REQUEST
                        0x1F-na-        -na-         -na-

The HandleNM() function is called for each network management
message, and does the appropriate processing.  Some messages
are simple enough to be processed right in HandleNM(), others
have their own function that is called by HandleNM().

The HandleND() function is called for each network diagnostic
message.  Like HandleNM(), the HandleND() function takes care
of simple messages, and more complicated messages are handled
by their own function.

*************************************************************/
/*-------------------------------------------------------------
Section: Includes
-------------------------------------------------------------*/
#include <string.h>

#include <CEA709_1.h>
#include <node.h>
#include <app.h>
#include <api.h>
#include <netmgmt.h>

/*-------------------------------------------------------------
Section: Constant Definitions
```

广东省网络空间安全协会受控资料

```
-----------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------
Section: Type Definitions
-----------------------------------------------------------------*/
#pragma maxalign(1)

typedef struct
{
   uint16   transmissionErrors;
   uint16   transmitTXFailures;
   uint16   receiveTXFull;
   uint16   lostMessages;
   uint16   missedMessages;
   uint8    resetCause;
   uint8    nodeState;
   uint8    versionNumber;
   uint8    errorLog;
   uint8    modelNumber;
} NDQueryStat;

#pragma maxalign()

/*-----------------------------------------------------------------
Section: Globals
-----------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------
Section: Function Prototypes.
-----------------------------------------------------------------*/
/* Local Functions */
static void RecomputeChecksum(void);
static void NMNDRespond(NtwkMgmtMsgType msgType,
                        Status success,
                        APPReceiveParam *appReceiveParamPtr,
                        APDU *apduPtr);

/* Network management related functions */
void HandleNMQueryId(APPReceiveParam *appReceiveParamPtr,
                     APDU *apduPtr);
```

广东省网络空间安全协会受控资料

```
void HandleNMWink(APPReceiveParam * appReceiveParamPtr,
                  APDU * apduPtr);

void HandleNMLeaveDomain(APPReceiveParam * appReceiveParamPtr,
                         APDU * apduPtr);

void HandleNMQueryAddr(APPReceiveParam * appReceiveParamPtr,
                       APDU * apduPtr);

void HandleNMQueryNvCnfg(APPReceiveParam * appReceiveParamPtr,
                         APDU * apduPtr);

void HandleNMQuerySIData(APPReceiveParam * appReceiveParamPtr,
                         APDU * apduPtr);

void HandleNMNVFetch(APPReceiveParam * appReceiveParamPtr,
                     APDU * apduPtr);

void HandleNMReadMemory(APPReceiveParam * appReceiveParamPtr,
                        APDU * apduPtr);

void HandleNMWriteMemory(APPReceiveParam * appReceiveParamPtr,
                         APDU * apduPtr);

void HandleNMQueryDomain(APPReceiveParam * appReceiveParamPtr,
                         APDU * apduPtr);

/* Network diagnostics related functions */
void HandleNDQueryStatus(APPReceiveParam * appReceiveParamPtr,
                         APDU * apduPtr);

void HandleNDProxyCommand(APPReceiveParam * appReceiveParamPtr,
                          APDU              * apduPtr);

void HandleNDTransceiverStatus(APPReceiveParam * appReceiveParamPtr,
                               APDU              * apduPtr);

void HandleNDClearStatus(APPReceiveParam * appReceiveParamPtr,
                         APDU              * apduPtr);

/* This function is for use by HandleNDProxyCommand */
void HandleNDQueryUnconfig(APPReceiveParam * appReceiveParamPtr,
                           APDU * apduPtr);
```

广东省网络空间安全协会受控资料

```
/* External function from physical layer to get the transceiver status. */
extern void GetTransceiverStatus(Byte transceiverStatusOut[]);

/*-----------------------------------------------------------------
Section: Function Definitions
-----------------------------------------------------------------*/
/*****************************************************************
Function:  RecomputeChecksum
Returns:   None.
Reference: None
Purpose:   To compute the configuration checksum and store.
Comments:  None.
*****************************************************************/
static void RecomputeChecksum(void)
{
   eep->configCheckSum = ComputeConfigCheckSum();
}


/*****************************************************************
Function:  ManualServiceRequestMessage
Returns:   TRUE if the message is sent, FALSE otherwise.
Reference: None
Purpose:   Produces a manual service request message.
Comments:  Prototype in api.h so that application program can use this
           function too. Returns TRUE or FALSE so that application
           program can determine whether the message was sent or not.
*****************************************************************/
Boolean ManualServiceRequestMessage(void)
{
   NWSendParam        *nwSendParamPtr;
   APDU               *apduRespPtr;

   if(QueueFull(&gp->nwOutQ))
   {
      return(FALSE); /* Can't send it now. Try later. */
   }

   /* Send unack domain wide broadcast message. */
   nwSendParamPtr          = QueueTail(&gp->nwOutQ);
   nwSendParamPtr->pduSize = 1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN;
   if (nwSendParamPtr->pduSize > gp->nwOutBufSize)
```

广东省网络空间安全协会受控资料

```
    {
        return(FALSE); /* Do not have sufficient space to send the message. */
    }
    apduRespPtr                          = (APDU *)(nwSendParamPtr + 1);
    apduRespPtr->code.allBits            = 0x7F;  /* Manual Service Request. */
    nwSendParamPtr->destAddr.domainIndex   = FLEX_DOMAIN;
    nwSendParamPtr->destAddr.flexDomainLen = 0;
    nwSendParamPtr->pduType                = APDU_TYPE;
    nwSendParamPtr->destAddr.addressMode   = BROADCAST;
    nwSendParamPtr->destAddr.addr.addr0    = 0; /* Domain wide broadcast. */
    nwSendParamPtr->deltaBL                = 0;
    nwSendParamPtr->altPath                = 0; /* don't use alternate path. */
    nwSendParamPtr->tag                    = MANUAL_SERVICE_REQ_TAG_VALUE;

    memcpy(apduRespPtr->data, eep->readOnlyData.uniqueNodeId,
            UNIQUE_NODE_ID_LEN);
    memcpy(&(apduRespPtr->data[UNIQUE_NODE_ID_LEN]),
            eep->readOnlyData.progId, ID_STR_LEN);
    EnQueue(&gp->nwOutQ);
    gp->manualServiceRequest = FALSE;
    return(TRUE);
}


/*****************************************************************
Function:   NMNDRespond
Returns:    None
Reference:  None
Purpose:    Respond with success or failure code to current message.
            Can be called either for ND or NM messages.
Comments:   None
*****************************************************************/
static void NMNDRespond(NtwkMgmtMsgType msgType,
                        Status success,
                        APPReceiveParam *appReceiveParamPtr,
                        APDU            *apduPtr)
{
    Queue           *tsaOutQPtr;
    TSASendParam    *tsaSendParamPtr;
    APDU            *apduRespPtr;

    /* If service type is not request, nothing to do. */
    if(appReceiveParamPtr->service != REQUEST)
```

广东省网络空间安全协会受控资料

```
    {
        /* Does not make sense to respond if the original is not a request. */
        return;
    }

    if (QueueFull(&gp->tsaRespQ))
    {
        return; /* Can't send the response. Do nothing. */
    }

    /* Send response. */
    tsaOutQPtr                      = &gp->tsaRespQ;
    tsaSendParamPtr                 = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service        = RESPONSE;
    tsaSendParamPtr->nullResponse   = FALSE;
    tsaSendParamPtr->reqId          = appReceiveParamPtr->reqId;
    tsaSendParamPtr->apduSize       = 1;
    apduRespPtr                     = (APDU *)(tsaSendParamPtr + 1);
    if (msgType == NM_MESSAGE)
    {
        apduRespPtr->code.allBits       =
            (success == SUCCESS? NM_resp_success:NM_resp_failure)
                | (apduPtr->code.allBits & 0x1F);
    }
    else
    {
        apduRespPtr->code.allBits       =
            (success == SUCCESS? ND_resp_success:ND_resp_failure)
                | (apduPtr->code.allBits & 0x0F);

    }
    EnQueue(tsaOutQPtr);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   HandleNMLeaveDomain
Returns:    None
Reference:  None
Purpose:    Handle incoming NM LeaveDomain message.
                — Delete the indicated domain table entry.
                — Recompute the configuration checksum.
                — If message was received on the indicated domain,
```

广东省网络空间安全协会受控资料

```
                do not respond
              - If node no longer belongs to any Domain:
                + become unconfigured
                + reset
Comments:  None.
*************************************************************************/
void HandleNMLeaveDomain(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
  /* Fail if message is not 2 bytes long */
  /* *** START INFORMATIVE - Parameter Validation *** */
  /* It is not required that the processing of network management commands include
    * parameter validation.  In general, it is the responsibility of the configuration
    * tool to ensure that requests are well formed. */
  if(appReceiveParamPtr->pduSize != 2)
  {
     NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
     DeQueue(&gp->appInQ);
     return;
  }
  /* If the domain index is bad, fail */
  if (apduPtr->data[0] != 0 && apduPtr->data[0] != 1)
  {
     nmp->errorLog = INVALID_DOMAIN;
     NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
     DeQueue(&gp->appInQ);
     return;
  }
  /* *** END INFORMATIVE - Parameter Validation *** */

  /* Leave the domain */
  memcpy(eep->domainTable[apduPtr->data[0]].domainId,
         "gmrdwf",
         DOMAIN_ID_LEN);
  eep->domainTable[apduPtr->data[0]].subnet       = 0;
  eep->domainTable[apduPtr->data[0]].cloneDomain = 1;
  eep->domainTable[apduPtr->data[0]].node         = 0;
  eep->domainTable[apduPtr->data[0]].len          = 0xFF; /* not in use */
  memcpy(eep->domainTable[apduPtr->data[0]].key,
         "NO_KEY",
         AUTH_KEY_LEN);

  /* Recompute the configuration checksum */
```

广东省网络空间安全协会受控资料

```
    RecomputeChecksum();

    /* If message not received on domain just left, then respond */
    if(apduPtr->data[0] ! = appReceiveParamPtr->srcAddr.domainIndex)
    {
        NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr,apduPtr);
    }

    /* If not a member of any domain, go unconfigured and reset. */
    if ((! eep->readOnlyData.twoDomains &&
            eep->domainTable[0].len == 0xFF) ||
        (eep->readOnlyData.twoDomains      &&
          eep->domainTable[0].len == 0xFF &&
          eep->domainTable[1].len == 0xFF) )
    {
        eep->readOnlyData.nodeState = APPL_UNCNFG;
        nmp->resetCause             = SOFTWARE_RESET;
        gp->resetNode               = TRUE; /* Scheduler will reset the node */
    }

    DeQueue(&gp->appInQ);
}

/*****************************************************************
Function:   HandleNMQueryAddr
Returns:    None
Reference:  None
Purpose:    Handle incoming NM Query Address message.
Comments:   None.
*****************************************************************/
void HandleNMQueryAddr(APPReceiveParam *appReceiveParamPtr,
                       APDU            *apduPtr)
{
    Queue              *tsaOutQPtr;
    TSASendParam       *tsaSendParamPtr;
    APDU               *apduRespPtr;
    AddrTableEntry     *ap;

    tsaOutQPtr = &gp->tsaRespQ;
    /* *** START INFORMATIVE — Parameter Validation *** */
    /* See "START INFORMATIVE — Parameter Validation" above. */
    /* Fail if message is not 2 bytes long. */
    if(appReceiveParamPtr->pduSize ! = 2)
```

广东省网络空间安全协会受控资料

```
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* *** END INFORMATIVE — Parameter Validation *** */
    /* Fail if the address table index is bad and set statistics. */
    if(apduPtr->data[0] >= NUM_ADDR_TBL_ENTRIES)
    {
        nmp->errorLog = INVALID_ADDR_TABLE_INDEX;
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* Fail if the buffer is insufficient for the response. */
    if (1 + sizeof(AddrTableEntry) > gp->tsaRespBufSize )
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }

    ap = AccessAddress(apduPtr->data[0]); /* ap should not NULL. */

    /* Send response */
    tsaSendParamPtr                 = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service         = RESPONSE;
    tsaSendParamPtr->nullResponse    = FALSE;
    tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
    apduRespPtr                     = (APDU *)(tsaSendParamPtr + 1);
    apduRespPtr->code.allBits       = NM_resp_success | NM_QUERY_ADDR;
    tsaSendParamPtr->apduSize       = 1 + sizeof(AddrTableEntry);
    memcpy(apduRespPtr->data, ap, sizeof(AddrTableEntry));
    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);
}

/*****************************************************************
Function:   HandleNMQueryNvCnfg
Returns:    None
Reference:  None
Purpose:    Handle incoming NM Query Netvar Config message.
```

广东省网络空间安全协会受控资料

```
Comments:  None.
*************************************************************/
void HandleNMQueryNvCnfg(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
    Queue          *tsaOutQPtr;
    TSASendParam   *tsaSendParamPtr;
    APDU           *apduRespPtr;
    uint16         n;
    uint16         nvAliasTableSize;

    /* Fail if the request does not have correct size */
    /* *** START INFORMATIVE - Parameter Validation *** */
    /* See "START INFORMATIVE - Parameter Validation" above. */
    if (appReceiveParamPtr->pduSize != 2 && appReceiveParamPtr->pduSize != 4)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* *** END INFORMATIVE - Parameter Validation *** */

    n = apduPtr->data[0];
    /* *** START INFORMATIVE - Parameter Validation *** */
    /* See "START INFORMATIVE - Parameter Validation" above. */
    if (n == 255 && appReceiveParamPtr->pduSize != 4)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* *** END INFORMATIVE - Parameter Validation *** */

    /* Calculate current alias table size */
    nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

    /* Decode index */

    /* *** START INFORMATIVE - Escaped NV Index *** */
    /* In implementations that handle a maximum number of network variables that is less
     * than 255, it is not necessary to check for network variable escapes of 255. */
    if(n == 255)
    {
```

广东省网络空间安全协会受控资料

```
        n = (uint16) apduPtr->data[1];
        n = (n << 8) | apduPtr->data[2];
    }
    /* *** END INFORMATIVE — Escaped NV Index *** */

    /* Fail if there is insufficient space to send the response */
    /* *** START INFORMATIVE — Parameter Validation *** */
    /* See "START INFORMATIVE — Parameter Validation" above. */
    if (
        (n < nmp->nvTableSize && (1 + sizeof(NVStruct)) > gp->tsaRespBufSize ) ||
        (n >= nmp->nvTableSize && n < nmp->nvTableSize + nvAliasTableSize &&
          (1 + sizeof(AliasStruct)) > gp->tsaRespBufSize )
        )
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* *** END INFORMATIVE — Parameter Validation *** */

    /* Send response */
    tsaOutQPtr                    = &gp->tsaRespQ;
    tsaSendParamPtr               = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service       = RESPONSE;
    tsaSendParamPtr->nullResponse = FALSE;
    tsaSendParamPtr->reqId         = appReceiveParamPtr->reqId;
    apduRespPtr                    = (APDU *)(tsaSendParamPtr + 1);
    apduRespPtr->code.allBits      = NM_resp_success | NM_QUERY_NV_CNFG;
    if(n < nmp->nvTableSize)
    {
        /* Copy the NVStruct entry */
        tsaSendParamPtr->apduSize = 1 + sizeof(NVStruct);
        memcpy(apduRespPtr->data, &(nmp->nvConfigTable[n]),
          sizeof(NVStruct));
    }
    else if(n < nmp->nvTableSize + nvAliasTableSize)
    {
        /* Copy the alias table entry. */
        tsaSendParamPtr->apduSize = 1 + sizeof(AliasStruct);
        n = n - nmp->nvTableSize;
        memcpy(apduRespPtr->data, &(nmp->nvAliasTable[n]),
          sizeof(AliasStruct));
```

广东省网络空间安全协会受控资料

```
    }
    else
    {
        nmp ->errorLog                  = INVALID_NV_INDEX;
        apduRespPtr ->code. allBits = NM_resp_failure | NM_QUERY_NV_CNFG;
        tsaSendParamPtr ->apduSize = 1;
    }

    EnQueue(tsaOutQPtr);
    DeQueue(&gp ->appInQ);
}

/*************************************************************************
Function:   HandleNMNVFetch
Returns:    None
Reference: None
Purpose:    Handle incoming NM NV Fetch message.
Comments:   None.
*************************************************************************/
void HandleNMNVFetch(APPReceiveParam *appReceiveParamPtr,
                     APDU              *apduPtr)
{
    Queue              *tsaOutQPtr;
    TSASendParam       *tsaSendParamPtr;
    APDU               *apduRespPtr;
    uint16          n;
    uint8          i;

    /* Check if the message has correct size. If not, fail. */
    /* *** START INFORMATIVE - Parameter Validation *** */
    /* See "START INFORMATIVE - Parameter Validation" above. */
    if (appReceiveParamPtr ->pduSize != 2 &&
        appReceiveParamPtr ->pduSize != 4)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp ->appInQ);
        return;
    }
    /* *** END INFORMATIVE - Parameter Validation *** */
    n = apduPtr ->data[0];
    /* *** START INFORMATIVE - Parameter Validation *** */
    /* See "START INFORMATIVE - Parameter Validation" above. */
    if (n == 255 && appReceiveParamPtr ->pduSize != 4)
```

广东省网络空间安全协会受控资料

```
{
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** END INFORMATIVE — Parameter Validation *** */
tsaOutQPtr = &gp->tsaRespQ;
/* Send response */
tsaSendParamPtr                  = QueueTail(tsaOutQPtr);
tsaSendParamPtr->altPathOverride = FALSE;
tsaSendParamPtr->service         = RESPONSE;
tsaSendParamPtr->nullResponse    = FALSE;
tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
apduRespPtr                      = (APDU *)(tsaSendParamPtr + 1);

/* *** START INFORMATIVE — Escaped NV Index *** */
/* See "START INFORMATIVE — Escaped NV Index" above. */
if(n == 255)
{
    n = apduPtr->data[1];
    n = (n << 8) | apduPtr->data[2];
    memcpy(apduRespPtr->data, apduPtr->data, 3); /* copy the index */
    i = 3; /* index where value of nv is copied */
}
/* *** END INFORMATIVE — Escaped NV Index *** */
else
{
    apduRespPtr->data[0] = (char) n;
    i = 1;
}
if (n < nmp->nvTableSize)
{
    /* Make sure there is sufficient space for the response. Else, fail. */
    /* *** START INFORMATIVE — Parameter Validation *** */
    /* See "START INFORMATIVE — Parameter Validation" above. */
    if (nmp->nvFixedTable[n].nvLength + i + 1 > gp->tsaRespBufSize)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* *** END INFORMATIVE — Parameter Validation *** */
    memcpy(&apduRespPtr->data[i],
```

广东省网络空间安全协会受控资料

```
            nmp->nvFixedTable[n].nvAddress,
            nmp->nvFixedTable[n].nvLength);
        tsaSendParamPtr->apduSize = nmp->nvFixedTable[n].nvLength + i + 1;
        apduRespPtr->code.allBits = NM_resp_success | NM_NV_FETCH;
        EnQueue(tsaOutQPtr);

    }
    else
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    }
    DeQueue(&gp->appInQ);
}

/*****************************************************************
Function:   HandleNMQuerySIData
Returns:    None
Reference:  None
Purpose:    Handle incoming NM Query SI Data message.
Comments:   None.
*****************************************************************/
void HandleNMQuerySIData(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
    Queue           *tsaOutQPtr;
    TSASendParam    *tsaSendParamPtr;
    APDU            *apduRespPtr;
    uint16          offset;
    uint8           count;

    tsaOutQPtr = &gp->tsaRespQ;

    /*  *** START INFORMATIVE — Query SI Data ***  */
    /* It is only necessary to respond to Query SI Data messages (AKA Query SNVT) if
     * the implementation does not support direct memory read/[write] of the device
     * self identification data.   Only a device with a "ReadOnlyDataStruct.snvtStruct"
     * field with value 0xffff must implement this function.  */

    /* Fail if message is not 4 bytes long */
    if(appReceiveParamPtr->pduSize != 4)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
```

广东省网络空间安全协会受控资料

```
        return;
    }

    /* Decode offset and count */
    offset = apduPtr->data[0];
    offset = (offset << 8) | apduPtr->data[1];
    count  = apduPtr->data[2];
    /* Check if we have enough space to respond for this message */
    if (count + 1 > gp->tsaRespBufSize)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }

    /* Send response */
    tsaSendParamPtr                  = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service         = RESPONSE;
    tsaSendParamPtr->nullResponse    = FALSE;
    tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
    apduRespPtr                      = (APDU *)(tsaSendParamPtr + 1);
    apduRespPtr->code.allBits        = NM_resp_success | NM_QUERY_SNVT;
    tsaSendParamPtr->apduSize        = 1 + count;
    memcpy(apduRespPtr->data,  offset + (char *)&(nmp->snvt), count);
    EnQueue(tsaOutQPtr);
    /*  * * * END INFORMATIVE — Query SI Data * * *  */
    DeQueue(&gp->appInQ);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   HandleNMWink
Returns:    None
Reference: None
Purpose:    Handle incoming NM Wink message.
Comments:   None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void HandleNMWink(APPReceiveParam *appReceiveParamPtr,
                  APDU            *apduPtr)
{
    Queue           *tsaQueuePtr;
    TSASendParam    *tsaSendParamPtr;
    APDU            *apduRespPtr;
```

广东省网络空间安全协会受控资料

```
int8                subcmd;
int8                niIndex;

subcmd = 0;

if (appReceiveParamPtr->pduSize > 1)
{
    subcmd   = apduPtr->data[0];
}

if (appReceiveParamPtr->pduSize > 2)
{
    niIndex = apduPtr->data[1];
}

/* *** START INFORMATIVE — Parameter Validation *** */
/* See "START INFORMATIVE — Parameter Validation" above. */
if (appReceiveParamPtr->pduSize > 3)
{
    /* Incorrect size */
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** END INFORMATIVE — Parameter Validation *** */

/* *** START INFORMATIVE — Wink Handling *** */
/* Handling of wink subcommands is not required.   That is, it is acceptable
 * to treat all requests with this code as if they were simple wink requests.
 * Wink subcommands are a set of backward compatible extensions to the wink
 * command code. */
if(appReceiveParamPtr->pduSize <= 1 || subcmd == 0)
{
    if (appReceiveParamPtr->service ! = REQUEST)
    {
       /* Any service except request/response */
       Wink();   /* Simple Wink */
    }
    else
    {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    }
    DeQueue(&gp->appInQ);
```

广东省网络空间安全协会受控资料

```
        return;
    }

    /* must be requesting SEND_ID_INFO. */
    if(appReceiveParamPtr->service ! = REQUEST)
    {
        DeQueue(&gp->appInQ); /* Discard */
        return; /* The message should be a request. */
    }

    tsaQueuePtr = &gp->tsaRespQ;
    if (QueueFull(tsaQueuePtr))
    {
        return;
    }
    tsaSendParamPtr                    = QueueTail(tsaQueuePtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    /* Note: This implementation only has one NI Interface. i.e 0 */
    /* Send response. */
    tsaSendParamPtr->service          = RESPONSE;
    tsaSendParamPtr->reqId            = appReceiveParamPtr->reqId;
    tsaSendParamPtr->nullResponse = FALSE;
    apduRespPtr                       = (APDU *)(tsaSendParamPtr + 1);

    if (niIndex == 0 &&
         UNIQUE_NODE_ID_LEN + ID_STR_LEN + 2 <= gp->tsaRespBufSize )
    {
        tsaSendParamPtr->apduSize        = ID_STR_LEN + UNIQUE_NODE_ID_LEN + 2;
        apduRespPtr->data[0]             = 0; /* Interface not down. */
        apduRespPtr->code.allBits        = NM_resp_success | NM_WINK;
        memcpy(&apduRespPtr->data[1], eep->readOnlyData.uniqueNodeId,
                UNIQUE_NODE_ID_LEN);
        memcpy(&(apduRespPtr->data[1+UNIQUE_NODE_ID_LEN]),
                eep->readOnlyData.progId, ID_STR_LEN);
    }
    else
    {
        tsaSendParamPtr->apduSize   = 1;
        apduRespPtr->code.allBits   = NM_resp_failure | NM_WINK;
    }
    EnQueue(tsaQueuePtr);
    /* *** END INFORMATIVE — Wink Handling *** */
    DeQueue(&gp->appInQ);
```

广东省网络空间安全协会受控资料

```
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   HandleNMQueryId
Returns:    None
Reference:  None
Purpose:    Handle incoming NM Query ID message.
Comments:   The message must be a request. There must be space in
            response queue. See HandleNM (We do these checks first).
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void HandleNMQueryId(APPReceiveParam *appReceiveParamPtr,
                     APDU            *apduPtr)
{
   NMQueryIdRequest   *pid;
   char               *memp;
   Queue              *tsaOutQPtr;
   TSASendParam       *tsaSendParamPtr;
   APDU               *apduRespPtr;
   Boolean            allowed;
   uint16             offset;

   /* Fail if message does not have the correct size. Should be 2 or 6+n */
   /*  * * * START INFORMATIVE — Parameter Validation * * *  */
   /* See "START INFORMATIVE — Parameter Validation" above. */
   if(appReceiveParamPtr->pduSize ! = 2 && appReceiveParamPtr->pduSize < 6)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /*  * * * END INFORMATIVE — Parameter Validation * * *  */

   tsaOutQPtr = &gp->tsaRespQ;

   /* Init some fields here. Assume we may fail. */
   tsaSendParamPtr                = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   apduRespPtr                    = (APDU *)(tsaSendParamPtr + 1);
   tsaSendParamPtr->service       = RESPONSE;
   tsaSendParamPtr->nullResponse  = FALSE;
   tsaSendParamPtr->reqId         = appReceiveParamPtr->reqId;
   tsaSendParamPtr->apduSize      = 1;
   /* Since there are a lot of fail cases, init code to indicate failed
```

广东省网络空间安全协会受控资料

```
    response. */
apduRespPtr->code.allBits      = NM_resp_failure | NM_QUERY_ID;

pid = (NMQueryIdRequest *)&apduPtr->data;
offset = pid->offset;

/* if optional fields are present, check that the data field has
    sufficient bytes. */
/* *** START INFORMATIVE — Parameter Validation *** */
/* See "START INFORMATIVE — Parameter Validation" above. */
if (appReceiveParamPtr->pduSize > 2 &&
     appReceiveParamPtr->pduSize ! = (6 + pid->count))
{
    /* The message does not have sufficient data or it has too much data. */
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** END INFORMATIVE — Parameter Validation *** */

switch(pid->selector)
{
    case UNCONFIGURED:
       if(! NodeUnConfigured())
       {
           /* Not unconfigured — don't respond. */
           tsaSendParamPtr->nullResponse = TRUE;
           EnQueue(tsaOutQPtr);
           DeQueue(&gp->appInQ);
           return;
       }
       break;
    case SELECTED:
       if(! gp->selectQueryFlag)
       {
           /* Not selected — don't respond. */
           tsaSendParamPtr->nullResponse = TRUE;
           EnQueue(tsaOutQPtr);
           DeQueue(&gp->appInQ);
           return;
       }
       break;
    case SELECTED_UNCFG:    /* must be selected and unconfigured */
```

广东省网络空间安全协会受控资料

```
        if(! gp->selectQueryFlag)
        {
            /* Not selected — don't respond. */
            tsaSendParamPtr->nullResponse = TRUE;
            EnQueue(tsaOutQPtr);
            DeQueue(&gp->appInQ);
            return;
        }
        if(! NodeUnConfigured())
        {
            /* Not unconfigured — don't respond */
            tsaSendParamPtr->nullResponse = TRUE;
            EnQueue(tsaOutQPtr);
            DeQueue(&gp->appInQ);
            return;
        }
        break;
    default:
        EnQueue(tsaOutQPtr); /* Failed response. */
        DeQueue(&gp->appInQ);
        return;
}

/* If memory matching is present, check memory match */
if(appReceiveParamPtr->pduSize > 2)
{
    switch(pid->mode)
    {
        case ABSOLUTE_MEM_ADDR:
            memp = (char *)nmp;
            if(offset >= 0xF000)
            {
                memp = (char *)eep - 0xF000;
            }
            else if (offset >= 0xFC00)
            {
                memp = (char *)nmp->memMapSpace - 0xFC00;
            }
            break;
        case CONFIG_RELATIVE:
            memp = (char *)&(eep->configData);
            break;
        case READ_ONLY_RELATIVE:
```

广东省网络空间安全协会受控资料

```
            memp = (char *)&(eep->readOnlyData);
            break;
        default:
            EnQueue(tsaOutQPtr);
            DeQueue(&gp->appInQ);
            return;    /* Failed response. */
    }

    memp += offset;

    /* Absolute addressing to read snvt is not possible */
    allowed = (memp >= (char *)&eep->readOnlyData &&
                memp + apduPtr->data[3] < (char *)&eep->domainTable[0]);
    if (! allowed)
    {
        EnQueue(tsaOutQPtr);
        DeQueue(&gp->appInQ);
        return; /* Failed response. */
    }

    if(memcmp(pid->data, memp, pid->count) ! = 0)
    {
        /* Compare failed - don't reply. */
        tsaSendParamPtr->nullResponse = TRUE;
        EnQueue(tsaOutQPtr);
        DeQueue(&gp->appInQ);
        return;
    }
}

/* Send response */
if (1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN <= gp->tsaRespBufSize )
{
    tsaSendParamPtr->apduSize = 1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN;
    apduRespPtr->code.allBits = NM_resp_success | NM_QUERY_ID;
    memcpy(apduRespPtr->data, eep->readOnlyData.uniqueNodeId,
            UNIQUE_NODE_ID_LEN);
    memcpy(&(apduRespPtr->data[UNIQUE_NODE_ID_LEN]),
            eep->readOnlyData.progId, ID_STR_LEN);
}

EnQueue(tsaOutQPtr);
DeQueue(&gp->appInQ);
```

广东省网络空间安全协会受控资料

```
}


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   HandleNMQueryDomain
Returns:    None
Reference: None
Purpose:    Handle incoming NM QueryDomain message.
Comments:   Must be a Request.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void HandleNMQueryDomain(APPReceiveParam  * appReceiveParamPtr,
                         APDU             * apduPtr)
{
   Queue              * tsaOutQPtr;
   TSASendParam         * tsaSendParamPtr;
   APDU                 * apduRespPtr;
   int                n;  /* Domain Index */


   /* Fail if message does not have the correct size.  */
   /*  * * * START INFORMATIVE — Parameter Validation * * *  */
   /* See "START INFORMATIVE — Parameter Validation" above.  */
   if(appReceiveParamPtr->pduSize ! = 2)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /*  * * * END INFORMATIVE — Parameter Validation * * *  */

   tsaOutQPtr = &gp->tsaRespQ;
   n = apduPtr->data[0];  /* Domain Index */
   /* If domain index is other than 0 or 1 or if the node is in only one
      domain and the index is 1, then fail.  */
   if ((n ! = 0 && n ! = 1) || (eep->readOnlyData.twoDomains == 0 && n == 1))
   {
      /* Domain index is bad.  */
      nmp->errorLog = INVALID_DOMAIN;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* If there is not enough space, fail too.  */
```

广东省网络空间安全协会受控资料

```
    if (1 + sizeof(DomainStruct) > gp->tsaRespBufSize)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }

    /* Send response */
    tsaSendParamPtr                 = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service         = RESPONSE;
    tsaSendParamPtr->nullResponse = FALSE;
    tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
    tsaSendParamPtr->apduSize        = 1 + sizeof(DomainStruct);
    apduRespPtr                     = (APDU *)(tsaSendParamPtr + 1);
    apduRespPtr->code.allBits       = NM_resp_success | NM_QUERY_DOMAIN;
    memcpy(apduRespPtr->data, &eep->domainTable[n], sizeof(DomainStruct));

    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);
}


/*********************************************************************
Function:   HandleNMReadMemory
Returns:    None
Reference: None
Purpose:    Handle incomming NM ReadMemory message
Comments:   None.
*********************************************************************/
void HandleNMReadMemory(APPReceiveParam *appReceiveParamPtr,
                        APDU            *apduPtr)
{
    Queue               *tsaOutQPtr;
    TSASendParam        *tsaSendParamPtr;
    APDU                *apduRespPtr;
    char                *memp;
    uint16              offset;
    Boolean             allowed;

    tsaOutQPtr = &gp->tsaRespQ;

    /* Fail if message is not 5 bytes long. */
```

广东省网络空间安全协会受控资料

```
/* *** START INFORMATIVE — Parameter Validation *** */
/* See "START INFORMATIVE — Parameter Validation" above. */
if(appReceiveParamPtr->pduSize ! = 5)
{
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** END INFORMATIVE — Parameter Validation *** */

offset = (apduPtr->data[1] << 8) | apduPtr->data[2];

/* Assemlbe response */
tsaSendParamPtr                    = QueueTail(tsaOutQPtr);
tsaSendParamPtr->altPathOverride = FALSE;
tsaSendParamPtr->service         = RESPONSE;
tsaSendParamPtr->nullResponse = FALSE;
tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
tsaSendParamPtr->apduSize        = 1 + apduPtr->data[3];

apduRespPtr = (APDU *)(tsaSendParamPtr + 1);

apduRespPtr->code.allBits = NM_resp_success | NM_READ_MEMORY;

switch(apduPtr->data[0])
{
    case ABSOLUTE_MEM_ADDR:
        memp = (char *)nmp;
        if(offset >= 0xF000)
        {
            memp = (char *)eep — 0xF000;
        }
        else if (offset >= 0xFC00)
        {
            memp = (char *)nmp->memMapSpace — 0xFC00;
        }
        break;
    case READ_ONLY_RELATIVE:
    default:
        memp = (char *)&(eep->readOnlyData);
        break;
    case CONFIG_RELATIVE:
        memp = (char *)&(eep->configData);
```

广东省网络空间安全协会受控资料

```
        break;
    case STAT_RELATIVE:
        memp = (char *)&(nmp->stats);
        break;
}

memp += offset;

/* If readWriteProtect flag is on, then only readonly data structure,
   snvt structures, and configuration structure can be read.
   The one byte read of location 0 (firmware number) is also allowed.
   Reference implementation does not support snvt reading through
   absolute addressing. readOnlyData.snvtStruct is 0xFFFF */


if(eep->readOnlyData.readWriteProtect == TRUE)
{
    allowed = (memp >= (char *)&eep->readOnlyData &&
               memp + apduPtr->data[3] < (char *)&eep->domainTable[0]) ||
               (memp == (char *)&nmp && apduPtr->data[3] == 1);

    if(! allowed)
    {
        tsaSendParamPtr->apduSize = 1;
        apduRespPtr->code.allBits = NM_resp_failure | NM_READ_MEMORY;
        EnQueue(tsaOutQPtr);
        DeQueue(&gp->appInQ);
        return;
    }
}

if (apduPtr->data[3] <= gp->tsaRespBufSize)
{
    memcpy(apduRespPtr->data, memp, apduPtr->data[3]);
}
else
{
    tsaSendParamPtr->apduSize = 1;
}

if (memp == (char *)nmp && apduPtr->data[3] == 1)
{
    /* trap for absolute read of location 0 and write
```

广东省网络空间安全协会管理资料

```
        version number (hard code it as 11) */
      apduRespPtr->data[0] = 11;
    }
    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);
}

/*****************************************************************
Function:  HandleNMWriteMemory
Returns:   None
Reference: None
Purpose:   Handle incoming NM WriteMemory message.
Comments:  None.
*****************************************************************/
void HandleNMWriteMemory(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
    Queue                  *tsaOutQPtr;
    TSASendParam           *tsaSendParamPtr;
    APDU                   *apduRespPtr;
    NMWriteMemoryRequest   *pr;
    char                   *memp;
    uint16                 offset;
    Boolean                allowed;

    tsaOutQPtr = &gp->tsaRespQ;

    /* Fail if message is not at least 6 bytes long */
    /*  *** START INFORMATIVE - Parameter Validation ***  */
    /* See "START INFORMATIVE - Parameter Validation" above. */
    if(appReceiveParamPtr->pduSize < 6)
    {
        NMNDRespond(NM_MESSAGE, FAILURE,appReceiveParamPtr,apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /*  *** END INFORMATIVE - Parameter Validation ***  */

    /* Pointer to struct describing memory request */
    pr = (NMWriteMemoryRequest *)&apduPtr->data[0];

    offset = pr->offset;
```

广东省网络空间安全协会受控资料

```
/* Fail if message length doesn't match count. Poke can use 16 data bytes.
   Allow that. Note that the code takes one byte. */
/* *** START INFORMATIVE — Parameter Validation *** */
/* See "START INFORMATIVE — Parameter Validation" above. */
if(appReceiveParamPtr->pduSize ! = 6 + pr->count &&
   appReceiveParamPtr->pduSize ! = 17)
{
   NMNDRespond(NM_MESSAGE, FAILURE,appReceiveParamPtr,apduPtr);
   DeQueue(&gp->appInQ);
   return;
}
/* *** END INFORMATIVE — Parameter Validation *** */

if (appReceiveParamPtr->service == REQUEST)
{
   /* Assemble response */
   tsaSendParamPtr                  = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service         = RESPONSE;
   tsaSendParamPtr->nullResponse    = FALSE;
   tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
   tsaSendParamPtr->apduSize        = 1;
   apduRespPtr = (APDU *)(tsaSendParamPtr + 1);
}

switch(pr->mode)
{
   case ABSOLUTE_MEM_ADDR:
      memp = (char *)nmp;
      if(offset >= 0xF000)
      {
         memp = (char *)eep - 0xF000;
      }
      else if (offset >= 0xFC00)
      {
         memp = (char *)nmp->memMapSpace - 0xFC00;
      }
      break;
   case CONFIG_RELATIVE:
      memp = (char *)&(eep->configData);
      break;
   case STAT_RELATIVE:
      memp = (char *)&(nmp->stats);
```

广东省网络空间安全协会受控资料

```
        break;
    case READ_ONLY_RELATIVE:
        memp = (char *)&(eep->readOnlyData);
        break;
    default:
        /* Invalid Mode */
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
}

memp += offset;

/* Check if the range of memory cells written is good. */
allowed = (memp >= (char *)nmp &&
            (memp + pr->count) <= ((char *)nmp + 64 * 1024)) ||
            (memp >= (char *)eep &&
                    (memp + pr->count) <= ((char *)eep + sizeof(EEPROM)));
if (! allowed)
{

    /* Send failure response if the message was a request */
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}

/* If readwrite flag is on, then only config structure can be written. */
if(eep->readOnlyData.readWriteProtect == TRUE)
{
    allowed = (memp >= (char *)&eep->configData &&
                memp + pr->count < (char *)&eep->domainTable[0]);

    if(! allowed)
    {
        /* Send failure response if the message was a request */
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
}

/* Need to copy only the data array in NM_Req structure.
```

广东省网络空间安全协会受控资料

```
        The header is 5 bytes long */
    /* We have to assume that pr->count is good. Max is 255 */
    /* Reference implementation has no application check sum.
        Only config checksum */
    memcpy(memp, apduPtr->data+5, pr->count);

    if (pr->form & CNFG_CS_RECALC) {
        RecomputeChecksum();
    }
    if (pr->form & ACTION_RESET) {
        gp->resetNode    = TRUE;
        nmp->resetCause = SOFTWARE_RESET;
    }

    /* There is no harm in responding even when the node is reset. It will
        be lost anyway */
    if(appReceiveParamPtr->service == REQUEST)
    {
        apduRespPtr->code.allBits = NM_resp_success | NM_WRITE_MEMORY;
        EnQueue(tsaOutQPtr);
    }
    DeQueue(&gp->appInQ);
}
```

广东省网络空间安全协会受控资料

```
/*****************************************************************
Function:   HandleProxyResponse
Returns:    None
Reference: None
Purpose:    Handle Proxy Response Message.
Comments:   None.
*****************************************************************/
void HandleProxyResponse(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
    Queue           *tsaOutQPtr;
    TSASendParam    *tsaSendParamPtr;
    APDU            *apduRespPtr;

    /* Send response */
    tsaOutQPtr = &gp->tsaRespQ;
```

```
    if(QueueFull(tsaOutQPtr))
    {
        return;  /* Return without processing message */
    }

    tsaSendParamPtr                  = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service         = RESPONSE;
    tsaSendParamPtr->nullResponse    = FALSE;
    tsaSendParamPtr->reqId           = nmp->pxyData.reqId;
    tsaSendParamPtr->apduSize        = appReceiveParamPtr->pduSize;
    apduRespPtr                      = (APDU *)(tsaSendParamPtr + 1);
    if (appReceiveParamPtr->pduSize <= gp->tsaRespBufSize)
    {
        memcpy(apduRespPtr, apduPtr, appReceiveParamPtr->pduSize);
    }
    else
    {
        tsaSendParamPtr->apduSize = 1;
    }

    EnQueue(tsaOutQPtr);

    /* Done processing */
    DeQueue(&gp->appInQ);

    /* Reset the flag as we have now responded */
    nmp->pxyData.pxyType = -1;
}

/*****************************************************************
Function:   HandleNDQueryStatus
Returns:    none
Reference: None
Purpose:    Handle Network Diagnostics Query Status Message.
Comments:   None.
*****************************************************************/
void HandleNDQueryStatus(APPReceiveParam *appReceiveParamPtr,
                         APDU *apduPtr)
{
    NDQueryStat     ndq;
    Queue           *tsaOutQPtr;
    TSASendParam    *tsaSendParamPtr;
```

广东省网络空间安全协会受控资料

```
APDU            *apduRespPtr;

tsaOutQPtr = &gp->tsaRespQ;
if (((apduPtr->code.allBits & 0x0F) == ND_QUERY_STATUS) &&
      appReceiveParamPtr->pduSize ! = 1)
{
    /* Incorrect size. Fail. */
    NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}

/* *** START INFORMATIVE — Parameter Validation *** */
/* See "START INFORMATIVE — Parameter Validation" above. */
if (((apduPtr->code.allBits & 0x0F) == ND_PROXY_COMMAND) &&
      appReceiveParamPtr->pduSize ! = 2)
{
    /* Incorrect size. Fail. */
    NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** END INFORMATIVE — Parameter Validation *** */

ndq.transmissionErrors = nmp->stats.transmissionErrors;
ndq.transmitTXFailures = nmp->stats.transmitTXFailures;
ndq.receiveTXFull      = nmp->stats.receiveTXFull;
ndq.lostMessages       = nmp->stats.lostMessages;
ndq.missedMessages     = nmp->stats.missedMessages;
ndq.resetCause         = nmp->resetCause;
if (eep->readOnlyData.nodeState == CNFG_ONLINE &&
     gp->appPgmMode == OFF_LINE)
{
    ndq.nodeState = SOFT_OFFLINE;
}
else
{
    ndq.nodeState = eep->readOnlyData.nodeState;
}
ndq.versionNumber = 128;
ndq.errorLog      = nmp->errorLog;
ndq.modelNumber   = 128;
  /* Send response */
```

广东省网络空间安全协会受控资料

```
    tsaSendParamPtr                    = QueueTail(tsaOutQPtr);
    tsaSendParamPtr->altPathOverride = FALSE;
    tsaSendParamPtr->service         = RESPONSE;
    tsaSendParamPtr->nullResponse = FALSE;
    tsaSendParamPtr->reqId           = appReceiveParamPtr->reqId;
    tsaSendParamPtr->apduSize        = 1 + sizeof(NDQueryStat);
    apduRespPtr                        = (APDU *)(tsaSendParamPtr + 1);
    /* The response code is formed using apduPtr->code.allBits as the
       actual could be a proxy command and the response code will be
       different. Thus, using apduPtr->code.allBits works for both
       native query status command as well as proxy based query status */
    if (tsaSendParamPtr->apduSize <= gp->tsaRespBufSize)
    {
       apduRespPtr->code.allBits        =
          ND_resp_success | (apduPtr->code.allBits & 0x0F);
       memcpy(apduRespPtr->data, &ndq,   sizeof(NDQueryStat));
    }
    else
    {
       apduRespPtr->code.allBits        =
          ND_resp_failure | (apduPtr->code.allBits & 0x0F);
       tsaSendParamPtr->apduSize        = 1;
    }
    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);

}

/*****************************************************************
Function:   HandleNDProxyCommand
Returns:    none
Reference: None
Purpose:    Handle network diagnostics proxy request message.
Comments:   None.
*****************************************************************/
void HandleNDProxyCommand(APPReceiveParam *appReceiveParamPtr,
                          APDU            *apduPtr)
{
    Queue          *tsaOutQPtr;
    TSASendParam   *tsaSendParamPtr;
    APDU           *apduSendPtr;
```

广东省网络空间安全协会受控资料

```
/* See if the proxy command is to be forwarded or it is for us
   to process. If the address information in the packet is missing,
   then there is no forwarding information and hence we respond directly */
if (appReceiveParamPtr->pduSize == 2)
{
    /* Address portion is missing. This node is the proxy target. */
   tsaOutQPtr = &gp->tsaRespQ;
   /* Send the response directly back to the sender */
   switch(apduPtr->data[0])
   {
      case 0:
          HandleNDQueryUnconfig(appReceiveParamPtr, apduPtr);
          break;
      case 1:
          HandleNDQueryStatus(appReceiveParamPtr, apduPtr);
          break;
      case 2:
          HandleNDTransceiverStatus(appReceiveParamPtr, apduPtr);
          break;
      default:
          /* Invalid sub_command. Send failure response */
          NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
          DeQueue(&gp->appInQ);
   }
   return;
}

/* Send failure response if a proxy command is already in progress
   or we are proxy agent and the message received is on flex domain
   or it does not have correct size */
if (
    nmp->pxyData.pxyType ! = -1 ||
    appReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN ||
    (apduPtr->data[1] == UNIQUE_NODE_ID &&
          appReceiveParamPtr->pduSize ! =
             (2 + sizeof(AddrTableEntry) + UNIQUE_NODE_ID_LEN) ) ||
    (apduPtr->data[1] ! = UNIQUE_NODE_ID &&
          appReceiveParamPtr->pduSize ! = (2 + sizeof(AddrTableEntry)) )
   )
{
   NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   return;
```

广东省网络空间安全协会受控资料

```
}

/* We need to forward the proxy now. Set the target queue ptrs */
if(appReceiveParamPtr->priority)
{
    tsaOutQPtr = &gp->tsaOutPriQ;
}
else
{
    tsaOutQPtr = &gp->tsaOutQ;
}

/* Check if the target queue has space for forwarding this request. */
if(QueueFull(tsaOutQPtr))
{
    return;  /* Return without processing message */
}

/* Save the proxy data for processing the response later.
   Also, pxyType will no longer be equal to -1 that serves
   as a kind of lock on this record. Future proxy commands
   will be ignored until this proxy command is done. The proxy
   command is done as soon as we get the response and we forward
   it to the original sender or when the transaction forwarding
   the proxy times out. Session layer will not give us duplicate
   requests and hence we don't need to handle that here. Also,
   session layet will take care of sending response again for
   duplicate requests. */
nmp->pxyData.pxyType = apduPtr->data[0];
nmp->pxyData.reqId   = appReceiveParamPtr->reqId;

/* Generate request message */
tsaSendParamPtr                  = QueueTail(tsaOutQPtr);
/* First copy the msg_out_addr info in the proxy command */
if (apduPtr->data[1] == UNIQUE_NODE_ID)
{
    memcpy(&tsaSendParamPtr->destAddr,&apduPtr->data[1],
           sizeof(MsgOutAddr) + UNIQUE_NODE_ID_LEN);
}
else
{
    memcpy(&tsaSendParamPtr->destAddr,&apduPtr->data[1],
```

广东省网络空间安全协会受控资料

```
            sizeof(MsgOutAddr));
    }
    /* Set the domain index to be the one in which it was received.
       Note that this cannot be flex domain. Transport or session
       will use this instead of the one in the destAddr (i.e msg_out_addr) */
    tsaSendParamPtr->domainIndex    = appReceiveParamPtr->srcAddr.domainIndex;
    tsaSendParamPtr->service    = REQUEST;
    tsaSendParamPtr->auth       = FALSE;
    tsaSendParamPtr->tag        = (MsgTag) 0xFFFF;
    /* Proxy relays the message, the altpath bit should be same as the one
       used for the proxy message received */
    tsaSendParamPtr->altPathOverride = TRUE;
    tsaSendParamPtr->altPath = appReceiveParamPtr->altPath;
    tsaSendParamPtr->apduSize = 2; /* Always two bytes: code + sub_command */
    apduSendPtr   = (APDU *)(tsaSendParamPtr + 1);
    /* Copy the code as it is */
    apduSendPtr->code.allBits = apduPtr->code.allBits;
    apduSendPtr->data[0] = apduPtr->data[0];
    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);
}

/*********************************************************************
Function:   HandleNDTransceiverStatus
Returns:    none
Reference: None
Purpose:    Handle network diagnostics transceiver status message.
Comments:   None.
*********************************************************************/
void HandleNDTransceiverStatus(APPReceiveParam *appReceiveParamPtr,
                               APDU            *apduPtr)
{
    Queue           *tsaOutQPtr;
    TSASendParam    *tsaSendParamPtr;
    APDU            *apduRespPtr;
    Byte            transceiverStatus[NUM_COMM_PARAMS];
    uint8           n;

    /* Check for proper size of the message. Regular pdusize is 1 byte.
       If this fn is called due to proxy request, then it is 2 bytes. */
    /*  *** START INFORMATIVE — Parameter Validation *** */
    /* See "START INFORMATIVE — Parameter Validation" above. */
    if (appReceiveParamPtr->pduSize > 2)
```

广东省网络空间安全协会受控资料

```
{
    NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** END INFORMATIVE — Parameter Validation *** */
else if (appReceiveParamPtr->pduSize == 2)
{
    /* Make sure it is a proxy request. Or else, length is wrong. */
    if (apduPtr->code.nd.ndFlag == 0x5 &&
        apduPtr->code.nd.ndCode == ND_PROXY_COMMAND &&
        apduPtr->data[0] == 2)
    {
        ; /* It is indeed a proxy command. Length ok. Proceed. */
    }
    else
    {
        NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
}

if (eep->configData.commType == SPECIAL_PURPOSE)
{
    GetTransceiverStatus(transceiverStatus);
    n = NUM_COMM_PARAMS;
}
else
{
    n = 0;
}

/* Send response */
tsaOutQPtr = &gp->tsaRespQ;
tsaSendParamPtr                = QueueTail(tsaOutQPtr);
tsaSendParamPtr->altPathOverride = FALSE;
tsaSendParamPtr->service       = RESPONSE;
tsaSendParamPtr->nullResponse  = FALSE;
tsaSendParamPtr->reqId         = appReceiveParamPtr->reqId;
tsaSendParamPtr->apduSize      = 1 + n;
apduRespPtr                    = (APDU *)(tsaSendParamPtr + 1);
```

广东省网络空间安全协会受控资料

```
    /* If a node does not have transceiver status registers, fail */
    if ( n > 0 &&
         tsaSendParamPtr->apduSize <= gp->tsaRespBufSize)
    {
        /* No registers or not enough space */
        apduRespPtr->code.allBits       = ND_resp_success |
                                          (apduPtr->code.allBits & 0x0F);

        memcpy(apduRespPtr->data, transceiverStatus, n);
    }
    else
    {
        apduRespPtr->code.allBits       = ND_resp_failure |
                                          (apduPtr->code.allBits & 0x0F);

        tsaSendParamPtr->apduSize       = 1;
    }
    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);
}

/*****************************************************************
Function:   HandleNDClearStatus
Returns:    none
Reference: None
Purpose:    Handle network diagnostics clear status message.
Comments:   None.
*****************************************************************/
void HandleNDClearStatus(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{

    /* Check for proper size of the message */
    /* *** START INFORMATIVE — Parameter Validation *** */
    /* See "START INFORMATIVE — Parameter Validation" above. */
    if (appReceiveParamPtr->pduSize != 1)
    {
        NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* *** END INFORMATIVE — Parameter Validation *** */
```

广东省网络空间安全协会受控资料

```
    /* Clear Status */
    nmp->stats.transmissionErrors    = 0;
    nmp->stats.transmitTXFailures    = 0;
    nmp->stats.receiveTXFull         = 0;
    nmp->stats.lostMessages          = 0;
    nmp->stats.missedMessages        = 0;
    nmp->stats.layer2Received        = 0;
    nmp->stats.layer3Received        = 0;
    nmp->stats.layer3Transmitted     = 0;
    nmp->stats.transmitTXRetries     = 0;
    nmp->stats.backlogOverflow       = 0;
    nmp->stats.lateAcknowledgements  = 0;
    nmp->stats.collisions            = 0;
    nmp->resetCause                  = CLEARED;
    nmp->errorLog                    = NO_ERRORS;   /* Cleared */

    nmp->stats.layer6_7MsgsSent      = 0;
    nmp->stats.layer6_7RespSent      = 0;
    nmp->stats.layer6_7MsgsRcvd      = 0;
    nmp->stats.layer6_7RespRcvd      = 0;
    nmp->stats.lateResponses         = 0;
    nmp->stats.lateChallenges        = 0;
    nmp->stats.lateReplies           = 0;

    /* NMNDRespond will send response only if the msg is REQUEST */
    NMNDRespond(ND_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
}

/*****************************************************************
Function:   HandleNDQueryUnconfig
Returns:    None
Reference:  None
Purpose:    Handle Query Unconfig through Proxy message.
Comments:   This is a simplified version of HandleNMQueryId.
            Only unconfig version of Query ID is supported.
*****************************************************************/
void HandleNDQueryUnconfig(APPReceiveParam *appReceiveParamPtr,
                           APDU            *apduPtr)
{
    Queue           *tsaOutQPtr;
    TSASendParam    *tsaSendParamPtr;
    APDU            *apduRespPtr;
```

广东省网络空间安全协会受控资料

```
/* Check for proper size of the message */
/* *** START INFORMATIVE — Parameter Validation *** */
/* See "START INFORMATIVE — Parameter Validation" above. */
if (appReceiveParamPtr->pduSize ! = 2)
{
    NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** END INFORMATIVE — Parameter Validation *** */

tsaOutQPtr = &gp->tsaRespQ;

/* Init some fields here. */
tsaSendParamPtr                 = QueueTail(tsaOutQPtr);
tsaSendParamPtr->altPathOverride = FALSE;
apduRespPtr                     = (APDU *)(tsaSendParamPtr + 1);
tsaSendParamPtr->service        = RESPONSE;
tsaSendParamPtr->nullResponse = FALSE;
tsaSendParamPtr->reqId          = appReceiveParamPtr->reqId;
tsaSendParamPtr->apduSize       = 1;
/* apduPtr->code.allBits should be proxy code */
apduRespPtr->code.allBits       = ND_resp_failure |
                                  (apduPtr->code.allBits & 0x0F);


if(! NodeUnConfigured())
{
    /* Not unconfigured — don't respond. */
    tsaSendParamPtr->nullResponse = TRUE;
    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);
    return;
}

/* Send response */
if (1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN <= gp->tsaRespBufSize )
{
    tsaSendParamPtr->apduSize = 1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN;
    apduRespPtr->code.allBits = ND_resp_success |
                                (apduPtr->code.allBits & 0x0F);
```

广东省网络空间安全协会受控资料

```
        memcpy(apduRespPtr->data, eep->readOnlyData.uniqueNodeId,
               UNIQUE_NODE_ID_LEN);
        memcpy(&(apduRespPtr->data[UNIQUE_NODE_ID_LEN]),
               eep->readOnlyData.progId, ID_STR_LEN);
    }

    EnQueue(tsaOutQPtr);
    DeQueue(&gp->appInQ);
}


/**************************************************************************
Function:   HandleND
Returns:    None
Reference: None
Purpose:    Handle incomming network diagnostic message.
Comments:   None.
**************************************************************************/
void HandleND(APPReceiveParam  *appReceiveParamPtr,
              APDU              *apduPtr)
{
    if(appReceiveParamPtr->service == RESPONSE)
    {
        /* It is not legal for a response to be an ND command */
        DeQueue(&gp->appInQ);
        return;
    }

    /* If network diagnostics messages need authentication
        and the message did not pass authentication and
        the node is not unconfigured
        then discard those messages that should be discarded and return. */
    if(! NodeUnConfigured() && eep->configData.nmAuth &&
        ! appReceiveParamPtr->auth)
    {
        /* Only two messages are allowed. Others should be discarded */
        if (apduPtr->code.nd.ndCode != ND_QUERY_STATUS &&
            apduPtr->code.nd.ndCode != ND_PROXY_COMMAND)
        {
            if(QueueFull(&gp->tsaRespQ))
            {
                return;  /* Can't send response now. Try later. */
            }
```

广东省网络空间安全协会受控资料

```
        nmp->errorLog = AUTHENTICATION_MISMATCH;
        NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
}

/* Only clear status command need not be request message. Others
    must be request/response */
if (apduPtr->code.nd.ndCode ! = ND_CLEAR_STATUS &&
     appReceiveParamPtr->service ! = REQUEST)
{
    /* Discard. We can't respond anyway as it is not a request */
    DeQueue(&gp->appInQ);
    return;
}

if (appReceiveParamPtr->service == REQUEST &&
     QueueFull(&gp->tsaRespQ))
{
    return; /* Can't send a response. Try later. */
}

/* Handle various network diagnostic message codes */
switch(apduPtr->code.nd.ndCode)
{
    case ND_QUERY_STATUS:
        HandleNDQueryStatus(appReceiveParamPtr, apduPtr);
        return;
    case ND_PROXY_COMMAND:
        HandleNDProxyCommand(appReceiveParamPtr, apduPtr);
        return;
    case ND_CLEAR_STATUS:
        HandleNDClearStatus(appReceiveParamPtr, apduPtr);
        return;
    case ND_QUERY_XCVR:
        HandleNDTransceiverStatus(appReceiveParamPtr, apduPtr);
        return;
    default:
        /* Discard unrecognized diagnostic command */
        NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
}
```

广东省网络空间安全协会受控资料

```
}

/****************************************************************
Function:   HandleNM
Returns:    None
Reference: None
Purpose:    Handle incomming network management messages.
Comments:   None.
****************************************************************/
void HandleNM(APPReceiveParam  *appReceiveParamPtr,
              APDU             *apduPtr)
{
    int16               i;
    uint16              n; /* for nv index */
    AddrTableEntry      *ap;
    GroupAddrMode       *groupStrPtr;
    uint16              addrIndex;
    NVStruct            *np;
    uint16              nvAliasTableSize;
    uint8               pduSize;

    if(appReceiveParamPtr->service == RESPONSE)
    {
        /* It is not legal for a response to be an NM command. */
        DeQueue(&gp->appInQ);
        return;
    }

    /* If network management messages need authentication
        and the message did not pass authentication and
        the node is not unconfigured
        then discard those messages that should be discarded and return. */
    if(! NodeUnConfigured() && eep->configData.nmAuth && ! appReceiveParamPtr->auth)
    {
        /* Only two messages are allowed. Others should be discarded */
        if (apduPtr->code.nm.nmCode != NM_QUERY_ID &&
            apduPtr->code.nm.nmCode != NM_RESPOND_TO_QUERY)
        {
            if(QueueFull(&gp->tsaRespQ))
            {
                return; /* Can't send response now. Try later. */
            }
            nmp->errorLog = AUTHENTICATION_MISMATCH;
```

广东省网络空间安全协会受控资料

```
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
}

/* If the service type is not request, discard certain messages. */
if(appReceiveParamPtr->service != REQUEST)
{
    switch(apduPtr->code.nm.nmCode)
    {
        /* messages that need request/resp. */
        case NM_QUERY_ID:
        case NM_QUERY_ADDR:
        case NM_QUERY_NV_CNFG:
        case NM_QUERY_DOMAIN:
        case NM_READ_MEMORY:
        case NM_QUERY_SNVT:
        case NM_NV_FETCH:
            DeQueue(&gp->appInQ);
            return;
        default:
            break;
    }
}

/* If service type is request, and there is no room for response
    return without processing — so message will be processed later. */
if(appReceiveParamPtr->service == REQUEST)
{
    if(QueueFull(&gp->tsaRespQ))
    {
        return; /* Wait until there is room for response. */
    }
}

/* Handle various network mgmt message codes */
switch(apduPtr->code.nm.nmCode)
{
    case NM_QUERY_ID:
        HandleNMQueryId(appReceiveParamPtr,apduPtr);
        return;
    case NM_RESPOND_TO_QUERY:
```

广东省网络空间安全协会受控资料

```
    /* Fail if message is not 2 bytes long or the byte is bad. */
    if(appReceiveParamPtr->pduSize ! = 2 ||
        (apduPtr->data[0] ! = 0 && apduPtr->data[0] ! = 1) )
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    gp->selectQueryFlag = apduPtr->data[0];
    NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr,apduPtr);
    DeQueue(&gp->appInQ);
    return;
case NM_UPDATE_DOMAIN:
    /* Fail if message is not the right size or the domain_index is bad. */
    if(appReceiveParamPtr->pduSize ! = 2 + sizeof(DomainStruct))
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    if (apduPtr->data[0] ! = 0 && apduPtr->data[0] ! = 1)
    {
        nmp->errorLog = INVALID_DOMAIN;
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* If the node is in only one domain and the request is to update
        domain index 1, then we send a failure response. */
    if (eep->readOnlyData.twoDomains == 0 &&
         apduPtr->data[0] == 1)
    {
        nmp->errorLog = INVALID_DOMAIN;
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* Reference implementation allows even clone domain update. */
    if (((DomainStruct *)&apduPtr->data[1])->cloneDomain == 1)
    {
        UpdateDomain((DomainStruct *)&apduPtr->data[1], apduPtr->data[0]);
    }
    else
```

广东省网络空间安全协会受控资料

```
        {
            UpdateCloneDomain((DomainStruct *)&apduPtr->data[1],
                              apduPtr->data[0]);
        }
        NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
        RecomputeChecksum();
        DeQueue(&gp->appInQ);
        return;
    case NM_LEAVE_DOMAIN:
        HandleNMLeaveDomain(appReceiveParamPtr, apduPtr);
        return;
    case NM_UPDATE_KEY:
        /* Fail if message is not of correct length or domain index is bad. */
        /* *** START INFORMATIVE — Parameter Validation *** */
        /* See "START INFORMATIVE — Parameter Validation" above. */
        if(appReceiveParamPtr->pduSize != 2 + AUTH_KEY_LEN)
        {
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
        }
        if (apduPtr->data[0] != 0 && apduPtr->data[0] != 1)
        {
            nmp->errorLog = INVALID_DOMAIN;
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
        }
        /* *** END INFORMATIVE — Parameter Validation *** */

        for(i = 0; i < AUTH_KEY_LEN; i++)
        {
            eep->domainTable[apduPtr->data[0]].key[i] +=
                apduPtr->data[i+1];
        }
        RecomputeChecksum();
        NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr,apduPtr);
        DeQueue(&gp->appInQ);
        return;
    case NM_UPDATE_ADDR:
        /* Check for incorrect size. Allow for padding. */
        /* *** START INFORMATIVE — Parameter Validation *** */
        /* See "START INFORMATIVE — Parameter Validation" above. */
```

广东省网络空间安全协会受控资料

```
    if (  appReceiveParamPtr->pduSize < 6 ||
          appReceiveParamPtr->pduSize > 7)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    if (apduPtr->data[1] ! = UNBOUND)
    {
        /* Not turnaround. Must be 7 bytes. */
        if (appReceiveParamPtr->pduSize ! = 7)
        {
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
        }
    }
    /* * * * END INFORMATIVE — Parameter Validation * * * */
    /* Fail if the address table index is bad. */
    if (apduPtr->data[0] >= NUM_ADDR_TBL_ENTRIES)
    {
        nmp->errorLog = INVALID_ADDR_TABLE_INDEX;
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }

    UpdateAddress((AddrTableEntry *)&apduPtr->data[1], apduPtr->data[0]);
    RecomputeChecksum();
    NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
case NM_QUERY_ADDR:
    HandleNMQueryAddr(appReceiveParamPtr, apduPtr);
    return;
case NM_QUERY_NV_CNFG:
    HandleNMQueryNvCnfg(appReceiveParamPtr, apduPtr);
    return;
case NM_UPDATE_GROUP_ADDR:
    /* This message must be delivered with group addressing and is
       updated based on the domain in which it was received. Hence,
       flex domain is not allowed. */
    if (appReceiveParamPtr->srcAddr.addressMode ! = MULTICAST ||
```

广东省网络空间安全协会受控资料

```
    appReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN)
{
    /* This message should be sent in MULTICAST. Fail */
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** START INFORMATIVE — Parameter Validation *** */
/* See "START INFORMATIVE — Parameter Validation" above. */
if (appReceiveParamPtr->pduSize ! = 1 + sizeof(AddrTableEntry))
{
    /* Incorrect size */
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
/* *** END INFORMATIVE — Parameter Validation *** */
/* For accessing the corresponding address table entry,
    let us use the domainIndex in which it was received and
    the group in which it was received. It makes sense to use
    the domainIndex in which the message was received rather than
    the domain index in the packet(that will be same for all
    recipients) as it may be different for different nodes. */
groupStrPtr = (GroupAddrMode *)&apduPtr->data[0];
if (groupStrPtr->groupFlag == 1)
{
    addrIndex = AddrTableIndex(appReceiveParamPtr->srcAddr.domainIndex,
                               appReceiveParamPtr->srcAddr.group);
}

/* Make sure we got a good index. */
if (groupStrPtr->groupFlag ! = 1 || addrIndex == 0xFF)
{
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}
ap = AccessAddress(addrIndex); /* ap cannot be NULL */
/* Only group size and timer values should be changed */
ap->groupEntry.groupSize   = groupStrPtr->groupSize;
ap->groupEntry.rptTimer    = groupStrPtr->rptTimer;
ap->groupEntry.retryCount  = groupStrPtr->retryCount;
ap->groupEntry.rcvTimer    = groupStrPtr->rcvTimer;
```

广东省网络空间安全协会受控资料

```
    ap->groupEntry.txTimer      = groupStrPtr->txTimer;
    RecomputeChecksum();
    NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
case NM_QUERY_DOMAIN:
    HandleNMQueryDomain(appReceiveParamPtr,apduPtr);
    return;
case NM_UPDATE_NV_CNFG:
    /* Check for pdu size. Fail if the size is too small or too big */
    /* *** START INFORMATIVE — Parameter Validation *** */
    /* See "START INFORMATIVE — Parameter Validation" above. */
    if (appReceiveParamPtr->pduSize < 5 ||
        appReceiveParamPtr->pduSize > 10)
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /* *** END INFORMATIVE — Parameter Validation *** */
    nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
    /* Decode index */

    n = apduPtr->data[0];
    /* *** START INFORMATIVE — Escaped NV Index *** */
    /* See "START INFORMATIVE — Escaped NV Index" above. */
    if(n == 255)
    {
        n = (uint16)apduPtr->data[1];
        n = (n << 8) | apduPtr->data[2];
        if (n < nmp->nvTableSize)
        {
            pduSize = sizeof(NVStruct) + 4; /* escaped regular update */
        }
        else
        {
            /* Escaped alias update. Assume that host_primary is
               absent for now. */
            pduSize = (sizeof(AliasStruct) - 2) + 4;
        }
        np = (NVStruct *)(&apduPtr->data[3]);
    }
    /* *** END INFORMATIVE — Escaped NV Index *** */
```

广东省网络空间安全协会受控资料

```
else
{
    if (n < nmp->nvTableSize)
    {
        pduSize = sizeof(NVStruct) + 2; /* regular update */
    }
    else
    {
        /* Alias update. Assume that host_primary is absent for now. */
        /* last 2 is for index + code */
        pduSize = (sizeof(AliasStruct) - 2) + 2;
    }
    np = (NVStruct *)(&apduPtr->data[1]);
}

/* Update nv config or alias table */
if(n < nmp->nvTableSize)
{
    if (appReceiveParamPtr->pduSize >= pduSize)
    {
        memcpy(&nmp->nvConfigTable[n], np, sizeof(NVStruct));
    }
    else
    {
        /* Incorrect size */
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
}
else if(n < nmp->nvTableSize + nvAliasTableSize)
{
    n = n - nmp->nvTableSize; /* Alias table index */
    /* Check for various forms of alias update */
    if (((AliasStruct *)np)->primary == 0xFF &&
        appReceiveParamPtr->pduSize == pduSize)
    {
        /* host_primary missing. default to 0xffff. Null alias update. */
        ((AliasStruct *)np)->hostPrimary = 0xffff;
    }
    else if (((AliasStruct *)np)->primary == 0xFF)
    {
        /* escaped alias. hostPrimary is present */
```

广东省网络空间安全协会受控资料

```
            pduSize += 2;
        }
        /* Update the nv alias table */
        if (appReceiveParamPtr->pduSize >= pduSize)
        {
            memcpy(&nmp->nvAliasTable[n], np, sizeof(AliasStruct));
        }
        else
        {
            /* Incorrect size */
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
        }
    }
    else
    {
        /* Invalid nv table index */
        nmp->errorLog = INVALID_NV_INDEX;
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }

    /* Recompute checksum and send response */
    RecomputeChecksum();
    NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;

/* NM_SET_NODE_MODE:
 *                                                                  Possible
 * Description                    State   Mode   Service LED   in ref imp?
 * ------------------------------------------------------------------------
 * Applicationless, unconfigured   3      —      On           NO
 * Unconfigured (w/application)    2      —      Flashing     YES
 * Configured, Hard Offline        6      —      Off          NO
 * Configured                      4      1      Off          YES
 * Configured, Soft offline        4      0      Off          YES
 *
 * The NM_SET_NODE_MODE message encompasses a lot of functionality,
 * and impacts some other areas of the implementation.
 * 1) Mode is not maintained in EEPROM
```

广东省网络空间安全协会受控资料

```
 * 2) A node that is soft-offline will go on-line when it is reset
 * 3) The hard-offline state is preserved across reset
 * 4) For either hard or soft offline, the scheduler is disabled
 * 5) When soft-offline:
 *      A) Polling an NV will return NULL data
 *      B) Incomming network variable updates are handled normally
 *      C) But nv_update_occurs events will be lost
 * 6) In all other states except configured:
 *      A) No response is returned on NV polls
 *      B) Incomming NV updates are discarded
 * 7) If a node is in a non-configured state, is reset and then issued
 *      a command to go configured, it will come up soft offline
 * 8) If a set node mode message changes the mode to offline or online
 *      the approprite task (if any) is executed
 * 9) Changing the node state recomputes the configuration checksum
 */
case NM_SET_NODE_MODE:
  /* Fail if message is not 2 or 3 bytes long.  */
  /*  * * * START INFORMATIVE — Parameter Validation * * * */
  /* See "START INFORMATIVE — Parameter Validation" above.  */
  if(appReceiveParamPtr->pduSize < 2 || appReceiveParamPtr->pduSize > 3)
  {
     NMNDRespond(NM_MESSAGE, FAILURE,appReceiveParamPtr, apduPtr);
     DeQueue(&gp->appInQ);
     return;
  }
  if ( (apduPtr->data[0] ! = 3 && (appReceiveParamPtr->pduSize < 2 ||
            appReceiveParamPtr->pduSize > 3)) ||
       (apduPtr->data[0] == 3 && appReceiveParamPtr->pduSize ! = 3)
     )
  {
     /* Incorrect size */
     NMNDRespond(NM_MESSAGE, FAILURE,appReceiveParamPtr, apduPtr);
     DeQueue(&gp->appInQ);
     return;
  }
  /*  * * * END INFORMATIVE — Parameter Validation * * *  */

  /* Mode on-line and mode off-line messages must not be request
     messages. See Page 9—47 Technology Device Data Book. Rev 3 */
  if (
        (apduPtr->data[0] == 0 || apduPtr->data[0] == 1)
       &&
```

广东省网络空间安全协会受控资料

```
        appReceiveParamPtr->service == REQUEST
    )
{
    /* Fail. */
    NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
}

switch(apduPtr->data[0])
{
    case 0:  /* Go to soft offline state */
        if (AppPgmRuns())
        {
            OfflineEvent();  /* Indicate to application program. */
        }
        eep->readOnlyData.nodeState = CNFG_ONLINE;
        gp->appPgmMode    = OFF_LINE;
        gp->ioOutputPin1 = FALSE; /* LED off */
        /* No response given as the message is not a request. */
        break;
    case 1:  /* Go on-line */
        OnlineEvent(); /* Indicate to application program. */
        eep->readOnlyData.nodeState = CNFG_ONLINE;
        gp->appPgmMode    = ON_LINE;
        gp->ioOutputPin1 = FALSE; /* LED off */
        /* No response given as the message is not a request. */
        break;
    case 2:  /* Application reset */
        gp->resetNode    = TRUE;
        nmp->resetCause = SOFTWARE_RESET; /* Software reset. */
        /* No response since the node is being reset. */
        break;
    case 3:  /* Change State */
        /* Fail if message is not 3 bytes long. */
        if(appReceiveParamPtr->pduSize ! = 3)
        {
            NMNDRespond(NM_MESSAGE, FAILURE,
                        appReceiveParamPtr, apduPtr);
            break;
        }
        eep->readOnlyData.nodeState = apduPtr->data[1];
        /* Preserve the state of appPgmMode except for
```

广东省网络空间安全协会受控资料

```
            NO_APPL_UNCNFG.  */
        if (eep->readOnlyData.nodeState == NO_APPL_UNCNFG)
        {
            gp->appPgmMode = NOT_RUNNING;
        }
        RecomputeChecksum();
        /* Respond with success if the message was a request.  */
        NMNDRespond(NM_MESSAGE, SUCCESS,
                    appReceiveParamPtr, apduPtr);
        break;
      default:
        /* Let us reset the node for this case */
        gp->resetNode    = TRUE;
        nmp->resetCause = SOFTWARE_RESET;
        break;
    }
    DeQueue(&gp->appInQ);
    return;
  case NM_READ_MEMORY:
    HandleNMReadMemory(appReceiveParamPtr, apduPtr);
    return;
  case NM_WRITE_MEMORY:
    HandleNMWriteMemory(appReceiveParamPtr, apduPtr);
    return;
  case NM_CHECKSUM_RECALC:
    /* Fail if the message does not have correct size or has bad value.  */
    /*  * * * START INFORMATIVE — Parameter Validation * * *  */
    /* See "START INFORMATIVE — Parameter Validation" above.  */
    if (appReceiveParamPtr->pduSize ! = 2 ||
          (apduPtr->data[0] ! = 1 && apduPtr->data[0] ! = 4))
    {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        return;
    }
    /*  * * * END INFORMATIVE — Parameter Validation * * *  */
    /* We don't have checksum for application. Just config checksum */
    RecomputeChecksum();
    NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
    DeQueue(&gp->appInQ);
    return;
  case NM_WINK:    /* Same as NM_INSTALL */
    HandleNMWink(appReceiveParamPtr,apduPtr);
```

广东省网络空间安全协会受控资料

```
        return;
    case NM_MEMORY_REFRESH:
        /* *** START INFORMATIVE — Memory Refresh *** */
        /* It is acceptable to also respond with a failure to the memory
         * refresh request if the implementation does not include memory
         * that requires refreshing.  An example of responding to memory
         * refresh with non failure responses is not provided here. */
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ);
        /* *** END INFORMATIVE — Memory Refresh *** */
        return;
    case NM_QUERY_SNVT:
        HandleNMQuerySIData(appReceiveParamPtr, apduPtr);
        return;
    case NM_NV_FETCH:
        HandleNMNVFetch(appReceiveParamPtr, apduPtr);
        return;
    case NM_MANUAL_SERVICE_REQUEST:
        /* This is unsolicited message from a node. Reference
           implementation ignores manual service request message from other nodes */
        DeQueue(&gp->appInQ);
        return;
    default:
        /* This is where any message that is not taken care of should be
           handled. An example is product query command. For now, we treat
           everything else as unrecognized network management message. */
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
        DeQueue(&gp->appInQ); /* Simply discard it */
    }
}

/*****************************************************************/
```

广东省网络空间安全协会受控资料

A.11 配置数据结构

```
/*****************************************************************
        File:  node.c

     Version:  1.7
```

```
      Reference:  Annex B, CEA 709.

        Purpose:  Configuration Data Strutures that contain
                  information about this node.
                  Also, define all type defintions needed.

           Note:  RefImp supports any number of stacks.
                  A global structure called ProtocolStackData
                  is defined in node.h. An array of such
                  structures is used so that each stack has
                  its own data that it works on. A global
                  pointer gp points to the right structure
                  before the stack code is executed. This
                  is done by the scheduler.

                  The support for multiple-stacks does not include support
                  for mac layer to handle multiple stacks or multiple application
                  programs. A true multi-stack system needs some extra coding.
          To Do:  None
**************************************************************************/
/*------------------------------------------------------------------------
Section: Includes
------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <CEA709_1.h>
#include <custom.h>
#include <node.h>
#include <physical.h>

/*------------------------------------------------------------------------
Section: Constant Definitions
------------------------------------------------------------------------*/
/* None */

/*------------------------------------------------------------------------
Section: Type Definitions
------------------------------------------------------------------------*/
/* None */

/*------------------------------------------------------------------------
```

广东省网络空间安全协会受控资料

```
Section: Globals
-----------------------------------------------------------------*/
EEPROM              *eep; /* actual structure is in eeprom.c */
NmMap               *nmp;
NmMap               nm[NUM_STACKS] = {{0}};
ProtocolStackData *gp;
ProtocolStackData   protocolStackDataGbl[NUM_STACKS];

#if defined(ALTERNATE_STRUCTURES_NEEDED)
msg_in_type         msg_in;
msg_out_type        msg_out;
resp_in_type        resp_in;
resp_out_type       resp_out;
nv_in_addr_type     nv_in_addr;
int16               nv_array_index;
#endif

/*-----------------------------------------------------------------
Section: Local Globals
-----------------------------------------------------------------*/
/* See Tech Device Data Book for the tables with these values */

/* Table on page 9-9. Field Value 1 is missing.
   We assume the value 20 for 1 */
static uint16 bufSizeCodeLGbl[16] =
{
   255,20,20,21,22,24,26,30,34,42,50,66,82,114,146,210
};
/* Table is on page 9-10. Field value 1 is missing.
   We assume the value 1 for 1 */
static uint16 bufCntCodeLGbl[16] =
{
   0,1,1,2,3,5,7,11,15,23,31,47,63,95,127,191
};
static uint16 rptTimerCodeLGbl[16] =
{
   16,24,32,48,64,96,128,192,256,384,512,768,1024,1536,2048,3072
};
static uint16 rcvTimerCodeLGbl[16] =
{
   128,192,256,384,512,768,1024,1536,2048,3072,4096,6144,8192,12288,
   16384,24576
};
```

广东省网络空间安全协会受控资料

```
static uint16 txTimerCodeLGbl[16] =
{
   16,24,32,48,64,96,128,192,256,384,512,768,1024,1536,2048,3072
};
static uint16 nonGroupTimerCodeLGbl[16] =
{
   128,192,256,384,512,768,1024,1536,2048,3072,4096,6144,8192,
   12288,16384,24576
};

/*-----------------------------------------------------------------
Section: Local Function Prototypes
-----------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------
Section: Function Definitions
-----------------------------------------------------------------*/
/*****************************************************************
Function:  AccessDomain
Returns:   Address of structure corresponding to given index
Purpose:   To return the address of the structure that has domain
           information for this node
Comments:  If an invalid index is given, log error message.
*****************************************************************/

DomainStruct *AccessDomain(uint8 indexIn)
{
   if (indexIn < MAX_DOMAINS)
   {
      return(&eep->domainTable[indexIn]);
   }
   return(NULL);
}


/*****************************************************************
Function:  UpdateDomain
Returns:   None
Purpose:   To Change the domain table entry with given structure.
Comments:  If an invalid index is given, log error message.
*****************************************************************/
void UpdateDomain(DomainStruct *domainInp, uint8 indexIn)
```

广东省网络空间安全协会受控资料

```
{
   if (domainInp && (indexIn < MAX_DOMAINS))
   {
      eep->domainTable[indexIn] = *domainInp;
      eep->domainTable[indexIn].cloneDomain = 1;
      return;

   }
   if (domainInp)
   {
      ErrorMsg("UpdateDomain: Invalid index.\n");

   }
   else
   {
      ErrorMsg("UpdateDomain: NULL domainInp passed.\n");
   }
}

/*****************************************************************
Function:   UpdateCloneDomain
Returns:    None
Purpose:    To Change the domain table entry with given structure.
Comments:   If an invalid index is given, log error message.
            If this fn is called, then the node can only receive
            messages addressed with Unique Node ID, group unack or
            broadcast addressing mode. CloneDomain is set to 0.
            Clone Domain allows a node to have same subnet and node ID
            as another node and yet receive messages of above types.
            All messages in network have this bit set to 1.
*****************************************************************/
void UpdateCloneDomain(DomainStruct *domainInp, uint8 indexIn)
{
   if (domainInp && (indexIn < MAX_DOMAINS))
   {
      eep->domainTable[indexIn] = *domainInp;
      eep->domainTable[indexIn].cloneDomain = 0;
      return;

   }
   if (domainInp)
   {
      ErrorMsg("UpdateDomain: Invalid index.\n");
```

广东省网络空间安全协会受控资料

```
    }
    else
    {
        ErrorMsg("UpdateDomain: NULL domainInp passed.\n");
    }
}

/*****************************************************************
Function:   AccessAddress
Returns:    Address of structure at given index
Reference:  Tech Device Data Book Rev 1 p.9-12
Purpose:    To access address table entry
Comments:   None
*****************************************************************/
AddrTableEntry *AccessAddress(uint16 indexIn)
{
    if (indexIn < NUM_ADDR_TBL_ENTRIES)
    {
        return(&eep->addrTable[indexIn]);
    }
    return(NULL);
}

/*****************************************************************
Function:   UpdateAddress
Returns:    None
Reference:  Tech Device Data Rev 1 p.9-12
Purpose:    To update an address table entry.
Comments:   None
*****************************************************************/
void UpdateAddress(AddrTableEntry *addrEntryInp, uint16 indexIn)
{
    if (addrEntryInp && indexIn < NUM_ADDR_TBL_ENTRIES)
    {
        eep->addrTable[indexIn] = *addrEntryInp;
        return;
    }
    if (addrEntryInp)
    {
        nmp->errorLog = INVALID_ADDR_TABLE_INDEX;
        ErrorMsg("UpdateAddress: Invalid index.\n");
    }
```

广东省网络空间安全协会受控资料

```
    else
    {
        ErrorMsg("UpdateAddress: NULL addrEntryInp passed.\n");
    }
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   IsGroupMember
Returns:    TRUE if this node belongs to given group. FALSE, else.
Reference: None
Purpose:    To check if a node belongs to a given group in the
            given domain. If it does, also get the member number.
Comments:   If groupMemberOut is NULL, then it is not used.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
Boolean IsGroupMember(Byte domainIndexIn, uint8 groupIn,
                      uint8 *groupMemberOut)
{
    uint16 i;

    for (i = 0; i < NUM_ADDR_TBL_ENTRIES; i++)
    {
        if (eep->addrTable[i].addrFormat >= 128)
        {
            /* Group Format */
            if (eep->addrTable[i].groupEntry.groupID == groupIn &&
              eep->addrTable[i].groupEntry.domainIndex == domainIndexIn)
            {
                break;
            }
        }
    }
    if (i == NUM_ADDR_TBL_ENTRIES)
    {
        return(FALSE); /* Not Found */
    }
    if (groupMemberOut)
    {
        *groupMemberOut = eep->addrTable[i].groupEntry.member;
    }
    return(TRUE); /* Found */
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

广东省网络空间安全协会受控资料

```
Function:    AddrTableIndex
Returns:     The index of the address table for the given domain
             and group. 0xFF if not found.
Reference: None
Purpose:     To get the addr table index for a given group and domain.
             If there is no such entry in the addr table, return 0xff.
Comments:    None
**************************************************************************/
uint16 AddrTableIndex(uint8 domainIndexIn, uint8 groupIn)
{
   uint16 i;

   for (i = 0; i < NUM_ADDR_TBL_ENTRIES; i++)
   {
      if (eep->addrTable[i].addrFormat >= 128)
      {
         /* Group Format */
         if (eep->addrTable[i].groupEntry.groupID == groupIn &&
            eep->addrTable[i].groupEntry.domainIndex == domainIndexIn)
         {
            return(i);
         }
      }
   }

   return(0xFF); /* Not Found */
}


/**************************************************************************
Function:    DecodeBufferSize
Returns:     Actual Buffer Size
Reference: Tech Device Data Rev 1 p.9-9
Purpose:     To compute the actual buffer size from code
Comments:    None
**************************************************************************/
uint16  DecodeBufferSize(uint8 bufSizeIn)
{
   if (bufSizeIn <= 15)
   {
      return(bufSizeCodeLGbl[bufSizeIn]);
   }
   ErrorMsg("DecodeBufferSize: Invalid code.\n");
```

广东省网络空间安全协会受控资料

```
    return(0);
}

/*****************************************************************
Function:   DecodeBufferCnt
Returns:    Actual Buffer Count
Reference: Tech Device Data Rev 1 p.9—10
Purpose:    To compute the actual buffer count from code
Comments:   None
*****************************************************************/
uint16  DecodeBufferCnt(uint8 bufCntIn)
{
    if (bufCntIn <= 15)
    {
        return(bufCntCodeLGbl[bufCntIn]);
    }
    ErrorMsg("DecodeBufferCnt: Invalid code.\n");
    return(0);
}

/*****************************************************************
Function:   DecodeRptTimer
Returns:    Actual timer value in ms
Reference: Tech Device Data Rev 1 p.9—17
Purpose:    To compute the actual rpt timer value from code
Comments:   None
*****************************************************************/
uint16 DecodeRptTimer(uint8 rptTimerIn)
{
    if (rptTimerIn <= 15)
    {
        return(rptTimerCodeLGbl[rptTimerIn]);
    }
    ErrorMsg("DecodeRptTimer: Invalid code.\n");
    return(0);
}

/*****************************************************************
Function:   DecodeRcvTimer
Returns:    Actual Receive Timer value in ms
Reference: Tech Device Data Rev 1 p.9—17
Purpose:    To compute the actual rcv timer value in ms from code
Comments:   None
```

广东省网络空间安全协会受控资料

```
****************************************/
uint16 DecodeRcvTimer(uint8 rcvTimerIn)
{
   if (rcvTimerIn <= 15)
   {
      return(rcvTimerCodeLGbl[rcvTimerIn]);
   }
   ErrorMsg("DecodeRcvTimer: Invalid code.\n");
   return(0);
}

/****************************************
Function:   DecodeTxTimer
Returns:    Actual Transmit Timer Value in ms
Reference: Tech Device Data Rev 1 p.9-17
Purpose:    To compute the actual transmit timer value from code
Comments:   None
****************************************/
uint16 DecodeTxTimer(uint8  txTimerIn)
{
   if (txTimerIn <= 15)
   {
      return(txTimerCodeLGbl[txTimerIn]);
   }
   ErrorMsg("DecodeTxTimer: Invalid code.\n");
   return(0);
}

/****************************************
Function:   DecodeNonGroupTimer
Returns:    Actual non-group timer value in ms
Reference: Tech Device Data Rev 1 p.9-11
Purpose:    To Compute the actual non-group timer value from code
Comments:   None
****************************************/
uint16 DecodeNonGroupTimer(uint8 nonGroupTimerIn)
{
   if (nonGroupTimerIn <= 15)
   {
      return(nonGroupTimerCodeLGbl[nonGroupTimerIn]);
   }
   ErrorMsg("DecodeNonGroupTimer: Invalid code.\n");
   return(0);
```

广东省网络空间安全协会受控资料

```
}

/*****************************************************************
Function:  AccessNV
Returns:   Address of NV conf table entry
Reference: Tech Device Data Rev 1 p. 9—18
Purpose:   To Access the NV Config Table Entry given the index
Comments:  None
*****************************************************************/
NVStruct *AccessNV(uint16 indexIn)
{
   if (indexIn < nmp->nvTableSize)
   {
       return(&nmp->nvConfigTable[indexIn]);
   }
   ErrorMsg("AccessNV: Invalid index.\n");
   return(NULL);
}

/*****************************************************************
Function:  UpdateNV
Returns:   None
Reference: Tech Device Data Rev 1 p. 9—18
Purpose:   To update an entry in NV Config Table
Comments:  None
*****************************************************************/
void UpdateNV(NVStruct *nvStructInp, uint16 indexIn)
{
   if (nvStructInp && indexIn < nmp->nvTableSize)
   {
       nmp->nvConfigTable[indexIn] = *nvStructInp;
       return;
   }
   if (nvStructInp)
   {
       ErrorMsg("UpdateNV: Invalid index.\n");
   }
   else
   {
       ErrorMsg("UpdateNV: NULL nvStructInp.\n");
   }
}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   NVTableIndex
Returns:    index of NV Config Table
Reference:  Tech Device Data Rev 1 p. 9—18.
Purpose:    To retrieve the index corresponding to a network varname.
Comments:   We don't need this fn for Ref Imp. as the app pgm already
            has the index for all variables.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
uint16 NVTableIndex(char varNameIn[])
{
   return(0);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   ErrorMsg
Returns:    None
Reference:  None
Purpose:    To store error msgs produced by these functions.
Comments:   It is just a sequence of Bytes that is large.
            If there is no more space, it wraps around and
            logs. Thus, if there are too many error logs,
            we will only have the latest ones.
            Each Log is automatically given a number.
            The output has log number followed by message.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void ErrorMsg(char errMessageIn[])
{
   uint16 j, msgNumber;
   uint16 spaceNeeded;

   /* Set the starting index for copy */
   /* If there is insufficient space, wrap around */
   /* spaceNeeded includes space for the null character */
   /* header(xxx. ) + msg + \0 */
   spaceNeeded = 5 + strlen(errMessageIn) + 1;
   if (gp->errorMsgIndex + spaceNeeded > ERROR_MSG_SIZE)
   {
      gp->errorMsgIndex = 0; /* Wrap around */
   }

   /* Compute the Log Number. Make sure it is no more
      than 3 digits */
   gp->errorMsgNumber++;
```


广东省网络空间安全协会受控资料

```
    if (gp->errorMsgNumber > 999)
    {
        gp->errorMsgNumber = 1;
    }
    msgNumber = gp->errorMsgNumber;

    j = gp->errorMsgIndex;
    sprintf(gp->errorMsg + j, "%3d. ", msgNumber);
    j = j + 5;

    /* Write the message */
    strcpy(gp->errorMsg + j, errMessageIn);

    /* Update errorMsgIndex. Don't count the space for null char */
    gp->errorMsgIndex = gp->errorMsgIndex + spaceNeeded - 1;
}

/*****************************************************************
Function:   DebugMsg
Returns:    None
Reference:  None
Purpose:    To Print Debugging Messages for stacks.
Comments:   Actually not recorded anywhere. One needs to set
            breakpoint at the end of this fn and print temp
            to see the msg.
*****************************************************************/
void DebugMsg(char debugMsgIn[])
{
    char temp[200];
    uint8 i;

    if (strlen(debugMsgIn) > 170)
    {
        return; /* Too large for temp. Ignore it. */
    }

    /* Find our Stack # */
    for (i = 0; i < NUM_STACKS; i++)
    {
        if (gp == &protocolStackDataGbl[i])
        {
            break;
        }
```

广东省网络空间安全协会受控资料

```
    }
    sprintf(temp, "Time:%5u: Stack %2d: ", GetCurrentMsTime(), i+1);
    strcpy(temp + strlen(temp), debugMsgIn);
}

/*****************************************************************
Function:   AllocateStorage
Returns:    Pointer to data storage allocated or NULL
Reference:  None
Purpose:    A Simple version of storage allocator similar to malloc.
            A Global array is used to allocate the srorage.
            If no more space, NULL is retured.
Comments:   There is no function similar to free. There is no need
            for such a funcion in the Reference Implementation.
*****************************************************************/
void *AllocateStorage(uint16 sizeIn)
{
    Byte *ptr;

    if (gp->mallocUsedSize + sizeIn > MALLOC_SIZE)
    {
        nmp->errorLog = MEMORY_ALLOC_FAILURE;
        return(NULL); /* No space for requested size */
    }

    ptr = gp->mallocStorage + gp->mallocUsedSize;
    gp->mallocUsedSize += sizeIn;

    return(ptr);
}

/*****************************************************************
Function:   GetCurrentTime
Returns:    Current Time (not necessarily real time)
Reference:
Purpose:    To get the current time.
Comments:   None
*****************************************************************/
uint32 GetCurrentTime(void)
{
    return(*(gp->currentTime));
}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:  GetCurrentMsTime
Returns:   Current Time (not necessarily real time)
Reference:
Purpose:   To get the current time.
Comments:  None
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
uint32 GetCurrentMsTime(void)
{
   return( * (gp->currentTime)/CLOCK_TICKS_PER_MS);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:  SetMsTimer
Returns:   None
Reference: None
Purpose:   To set a timer value to a given value in ms.
Comments:  The timer is set to given value. The lastUpdateTime is
           set to current time. UpdateTimer is called later to
           update the timer value.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void SetMsTimer(MsTimer * timerOut, uint16 initValueIn)
{
   timerOut->curTimerValue   = initValueIn * CLOCK_TICKS_PER_MS;
   timerOut->lastUpdatedTime = GetCurrentTime();
   /* If the initial value is 0, then we assume that the timer is
      actually disabled.  */
   if (initValueIn > 0)
   {
      timerOut->expired = FALSE;
   }
   else
   {
      timerOut->expired = TRUE;
   }
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:  UpdateMsTimer
Returns:   None
Reference: None
Purpose:   To Update a timer value.
Comments:  Get CurrentTime. See the difference between cur time
```

广东省网络空间安全协会受控资料

```
            and last updated time. Subtract that qty from timer
            value. Make sure that the timer value is not
            reduced below 0 (that will make it a big positive).
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
void UpdateMsTimer(MsTimer * timerInOut)
{
   uint32 curTime, elapsedTime;

   if (timerInOut->curTimerValue == 0)
   {
      return; /* Already expired. */
   }

   /* Since the timer values are unsinged, C gurantees that simple
      subtraction will give the right value even if the timer
      has already wrapped. We just have to make sure that we
      update the timer within one cycle of the clock. For 32
      bit clock, this is long enough and hence no problem. */
   curTime      = GetCurrentTime();
   elapsedTime = curTime - timerInOut->lastUpdatedTime;
   if (elapsedTime >= timerInOut->curTimerValue)
   {
      timerInOut->curTimerValue = 0;
   }
   else
   {
      timerInOut->curTimerValue -= elapsedTime;
   }

   timerInOut->lastUpdatedTime = curTime;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:   MsTimerExpired
Returns:    TRUE if the timer has expired. FALSE if the timer has
              not expired or it has expired but has already been
              reported as expired.
Reference: None
Purpose:    To update given timer and test if it expired.
Comments:   None
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
Boolean MsTimerExpired(MsTimer * timerInOut)
{
```

广东省网络空间安全协会受控资料

```
    if (timerInOut->curTimerValue > 0)
    {
        UpdateMsTimer(timerInOut); /* Not expired. First update it. */
    }
    if (timerInOut->expired)
    {
        return(FALSE); /* Already expired. */
    }
    if (timerInOut->curTimerValue == 0)
    {
        /* First time expiry. */
        timerInOut->expired = TRUE; /* Remember this expiry */
        return(TRUE);
    }
    return(FALSE); /* Not expired yet */
}


/*********************************************************************
Function:   NodeReset
Returns:    None
Reference:
Purpose:    Initialization of node data structures.
Comments:
*********************************************************************/
void NodeReset(Boolean firstReset)
{
    void APPReset(void), TCSReset(void), TSAReset(void),
         NWReset(void),  LKReset(void),  PHYReset(void);
    void AppReset(void);
    uint32 seed;
#ifndef SIMULATION
    uint32 *StackTimerInit(void);
#endif
    int i;

#ifndef SIMULATION
    if (! firstReset)
    {
        PHYDisableSPMIsr();
    }
#endif
```

广东省网络空间安全协会受控资料

```
  /* Init variables that are not in EEPROM */
  memset(&nmp->stats, 0, sizeof(StatsStruct));
  if (firstReset)
  {
      nmp->errorLog          = NO_ERRORS;
  }
  nmp->pxyData.pxyType       -1;
  gp->errorMsgIndex      = 0;
  gp->errorMsgNumber     = 0;
  gp->softwareTime       = 0;
  gp->prevPinState[0]    = 0;

#ifdef SIMULATION
  gp->currentTime        = &gp->softwareTime;
#else
  if (firstReset)
  {
      gp->currentTime        = StackTimerInit();
  }
#endif
  /* A node in soft off-line state should go on-line state */
  if (eep->readOnlyData.nodeState == CNFG_ONLINE && gp->appPgmMode == OFF_LINE)
  {
      gp->appPgmMode = ON_LINE; /* Normal state. on-line. */
  }

  /* If a node is reset while in unconfigured state, it will come back in
     offline mode when asked to go configured later. */
  if (NodeUnConfigured())
  {
      gp->appPgmMode = OFF_LINE;
  }

  /* First, Let each layer determine the address of all its
     data strcutures */
  gp->mallocUsedSize = 0;


  APPReset(); /* Application Layer */
  if (! gp->resetOk)
  {
      return;
  }
```

广东省网络空间安全协会受控资料

```
    TCSReset();
    if (! gp->resetOk)
    {
        return;
    }
    TSAReset();
    if (! gp->resetOk)
    {
        return;
    }
    NWReset();
    if (! gp->resetOk)
    {
        return;
    }
    LKReset();
    if (! gp->resetOk)
    {
        return;
    }
    PHYReset();

    if (! gp->resetOk)
    {
        return;
    }

    /* Call this function last as this function can call functions
       that will use data structures that should already exist. */
    AppReset(); /* Application Program Reset */
    if (! gp->resetOk)
    {
        return;
    }

#ifndef SIMULATION
    if (firstReset)
    {
        PHYInitSPM();
    }
    else
    {
        PHYEnableSPMIsr();
```

广东省网络空间安全协会受控资料

```
    }
#endif

    if (firstReset)
    {
        gp->prevChallenge[i] = 0; /* Init prevRand. Don't change during other reset. */
    }

    /* Init Seed to some unpredictable value at start */
    seed = 0;
#ifndef SIMULATION
    seed = GetCurrentTime(); /* Current Value of Timer */
#endif
    srand((unsigned int) seed);

    if (nmp->resetCause == EXTERNAL_RESET || nmp->resetCause == POWER_UP_RESET)
    {
        SetMsTimer(&gp->tsDelayTimer, TS_RESET_DELAY_TIME);
    }
    else
    {
        SetMsTimer(&gp->tsDelayTimer, 0); /* Disable */
    }
    gp->resetNode          = FALSE;
}

/*****************************************************************
Function:   InitEEPROM
Returns:    None
Reference:  None
Purpose:    To initialize the EEPROM data items based on constants
            in custom.h and values set in custom.c
Comments:   Incomplete Initialization. Make sure it has the var you
            want or else add it here or in custom.h or custom.c
                depending on where it fits.
*****************************************************************/
void    InitEEPROM(void)
{
    int i;
    char *p;

    /* Init the entire readOnlyData to 0 first. */
    memset(&eep->readOnlyData, 0, sizeof(eep->readOnlyData));
```

广东省网络空间安全协会受控资料

```
/* Init the entire configData to 0 first. */
memset(&eep->configData, 0, sizeof(eep->configData));

/* Init Based on custom.h and default values */
eep->readOnlyData.modelNum            = MODEL_NUM;
eep->readOnlyData.minorModelNum       = MINOR_MODEL_NUM;
eep->readOnlyData.checkSum            = 0;

eep->readOnlyData.nvFixed[0] = 0xFF; /* not useful */
eep->readOnlyData.nvFixed[1] = 0xFF;

eep->readOnlyData.runWhenUnconf       = RUN_WHEN_UNCONF;
eep->readOnlyData.nvCount             = 0;
/* MIP uses 0xFFFF for snvtStruct field. p.9-8 */
eep->readOnlyData.snvtStruct[0]       = 0xFF;
eep->readOnlyData.snvtStruct[1]       = 0xFF;
eep->readOnlyData.nodeState           = CNFG_ONLINE;
/* NUM_ADDR_TBL_ENTRIES can be larger than 15, but
   addressCnt is set to min(15, NUM_ADDR_TBL_ENTRIES).
   The remaining entries are not seen by the lonbuilder tool */
eep->readOnlyData.addressCnt          =
   (NUM_ADDR_TBL_ENTRIES <= 15)? NUM_ADDR_TBL_ENTRIES:15;
eep->readOnlyData.receiveTransCnt     =
   (RECEIVE_TRANS_COUNT < 16)? RECEIVE_TRANS_COUNT-1:15;
eep->readOnlyData.appOutBufSize       = APP_OUT_BUF_SIZE;
eep->readOnlyData.appInBufSize        = APP_IN_BUF_SIZE;
eep->readOnlyData.nwOutBufSize        = NW_OUT_BUF_SIZE;
eep->readOnlyData.nwInBufSize         = NW_IN_BUF_SIZE;
eep->readOnlyData.nwOutBufPriCnt      = NW_OUT_PRI_Q_CNT;
eep->readOnlyData.appOutBufPriCnt     = APP_OUT_PRI_Q_CNT;
eep->readOnlyData.appOutBufCnt        = APP_OUT_Q_CNT;
eep->readOnlyData.appInBufCnt         = APP_IN_Q_CNT;
eep->readOnlyData.nwOutBufCnt         = NW_OUT_Q_CNT;
eep->readOnlyData.nwInBufCnt          = NW_IN_Q_CNT;
eep->readOnlyData.msgTagCnt           = 0;

eep->readOnlyData.readWriteProtect = READ_WRITE_PROTECT;
eep->readOnlyData.txByAddress         = 0;
eep->readOnlyData.aliasCnt            = 0; /* Host based node */


/* Initialize configData */
eep->configData.channelId             = 0;
```

广东省网络空间安全协会受控资料

```
eep->configData.commClock            = 3;
eep->configData.inputClock           = 5;
eep->configData.commType             = SPECIAL_PURPOSE;
eep->configData.commPinDir           = 0x1E; /* 0x17 if wake-up pin is input */
eep->configData.reserved[0]          = 0x00; /* for special purpose mode. */
eep->configData.reserved[1]          = 0x3F; /* packet_cycle */
eep->configData.reserved[2]          = 0xA6; /* beta2 control */
eep->configData.reserved[3]          = 0x77; /* xmit_interpacket */
eep->configData.reserved[4]          = 0x67; /* recv_interpacket */
eep->configData.nodePriority         = 1; /* 0—255. 0 => no priority slot. */
eep->configData.channelPriorities    = 8; /* 0—255 */
eep->configData.param.xcvrParams[0] = 0x0e;
eep->configData.param.xcvrParams[1] = 0x01;
eep->configData.param.xcvrParams[2] = 0;
eep->configData.param.xcvrParams[3] = 0;
eep->configData.param.xcvrParams[4] = 0;
eep->configData.param.xcvrParams[5] = 0;
eep->configData.param.xcvrParams[6] = 0;
/* dirParams only used for direct mode not special purpose mode */
/* eep->configData.param.dirParams.bitSyncThreshHold = 1; */
eep->configData.nonGroupTimer        = NON_GROUP_TIMER;
eep->configData.nmAuth               = NM_AUTH;
eep->configData.preemptionTimeout    = 0;

/* Initializaiton based on custom.c */
memcpy(eep->readOnlyData.uniqueNodeId, cp->uniqueNodeId, UNIQUE_NODE_ID_LEN);
eep->readOnlyData.twoDomains = cp->twoDomains;
memcpy(eep->readOnlyData.progId, cp->progId, ID_STR_LEN);
memcpy(eep->configData.location, cp->location, LOCATION_LEN);
for (i = 0; i <= cp->twoDomains; i++)
{
    eep->domainTable[i].len = cp->len[i];
    memcpy(eep->domainTable[i].domainId, cp->domainId[i],
                cp->len[i]);
    eep->domainTable[i].subnet = cp->subnet[i];
    eep->domainTable[i].node   = cp->node[i];
    eep->domainTable[i].cloneDomain = 1;
    memcpy(eep->domainTable[i].key, cp->key[i], AUTH_KEY_LEN);
}
/* Init Address Table based on custom.c */
for (i = 0; i < NUM_ADDR_TBL_ENTRIES; i++)
{
    memcpy(&eep->addrTable[i], &cp->addrTbl[i], 5);
```

广东省网络空间安全协会受控资料

```
    }

    /* Init Alias Table based in custom.c */
    /* Since C initializes missing elements with 0 we use
       any non-zero value for hostPrimary field to indicate that
       we did initialize an entry. We don't need 0 anyway for
       hostPrimary as we can use primary for such entries. */

    for (i = 0; i < NV_ALIAS_TABLE_SIZE; i++)
    {
        memcpy(&nmp->nvAliasTable[i], &cp->aliasTbl[i], 6);
    }

    nmp->nvTableSize   = 0;

    /* Initialize Alias Tables that are not initialized in custom.h */
    for (i = 0; i < NV_ALIAS_TABLE_SIZE; i++)
    {
        /* Init only those that are not given meaningful values
           in custom.h */
        if (nmp->nvAliasTable[i].hostPrimary != 0)
        {
            continue; /* Skip this as it was initialized in custom.c */
        }

        p = (char *)&nmp->nvAliasTable[i];
        *p          = (char) 0x70;
        *(p + 1)    = (char) 0x00;
        *(p + 2)    = (char) 0x0F;
        *(p + 3)    = (char) 0xFF;
        *(p + 4)    = (char) 0xFF;
        *(p + 5)    = (char) 0xFF;
    }
}

/*****************************************************************
Function:   GetPrimaryIndex
Returns:    The primary index of the given variable.
Reference:  None
Purpose:    To compute the primary index
Comments:   Given index can be either primary or alias.
*****************************************************************/
int16 GetPrimaryIndex(int16 nvIndex)
```

广东省网络空间安全协会受控资料

```
{
   int16    primaryIndex;
   uint16   nvAliasTableSize;

   /* Fetch the alias table size */
   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   if (nvIndex < 0 || nvIndex >= nmp->nvTableSize + nvAliasTableSize)
   {
      return(-1); /* Bad index value. */
   }

   if (nvIndex < nmp->nvTableSize)
   {
      primaryIndex = nvIndex; /* Primary index itself. */
   }
   else
   {
      nvIndex = nvIndex - nmp->nvTableSize; /* Get alias table index. */
      /* Compute the primary index. */
      primaryIndex = nmp->nvAliasTable[nvIndex].primary;
      if (primaryIndex == 0xFF)
      {
         primaryIndex = nmp->nvAliasTable[nvIndex].hostPrimary;
      }
      if (primaryIndex >= nmp->nvTableSize)
      {
         return(-1); /* Bad index in alias structure. */
      }
   }
   return(primaryIndex);
}

/*****************************************************************
Function:  GetNVStructPtr
Returns:   Pointer to the network variable structure.
Reference: None
Purpose:   To compute the pointer to the network variable structure.
Comments:  The given index can be either primary or alias.
*****************************************************************/
NVStruct *GetNVStructPtr(int16 nvIndexIn)
{
   uint16   nvAliasTableSize;
```


广东省网络空间安全协会受控资料

```
    /* Fetch the alias table size */
    nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

    if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize + nvAliasTableSize)
    {
        return(NULL); /* Bad index value. */
    }

    if (nvIndexIn < nmp->nvTableSize)
    {
        return(&nmp->nvConfigTable[nvIndexIn]);
    }

    return(&nmp->nvAliasTable[nvIndexIn - nmp->nvTableSize].nvConfig);
}

/****************************************
Function:   CheckSum4
Returns:    4 bit checksum of a given data.
Reference: None
Purpose:    To Compute the checksum of an array of bytes of
            a given length. The check sum is the successive
            application of exclusive or of successive 4 bits.
Comments:   None
****************************************/
uint8  CheckSum4(void *dataIn, uint16 lengthIn)
{
    unsigned char *p;
    uint16 i;
    uint8 result = 0; /* Final checksum in low order 4 bits. */

    p = dataIn;
    for (i = 0; i < lengthIn; i++)
    {
        result = result ^ (*p >> 4); /* exclusive or with high order
                                        4 bits */
        result = result ^ (*p & 0x0F); /* With low order 4 bits */
        p++;
    }
    return(result);
}
```

广东省网络空间安全协会受控资料

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:  CheckSum8
Returns:   8 bit checksum of a given data
Reference: None
Purpose:   To compute the checksum of an array of bytes of
           a given length. The check sum is the successive
           application of exclusive or of successive 4 bits.
Comments:  None
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
uint8   CheckSum8(void *dataIn, uint16 lengthIn)
{
   unsigned char *p;
   uint16 i;
   uint8 result = 0; /* Final checksum */

   p = dataIn;
   for (i = 0; i < lengthIn; i++)
   {
      result = result ^ *p;
      p++;
   }
   return(result);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Function:  ComputeConfigCheckSum
Returns:   The configuration checksum.
Reference: None
Purpose:   To compute the configuration checksum.
Comments:  Do not include reserved portion as it does not have
           any configuration data. Since, reference implementation
           has nv tables in nmp structure rather than EEPROM,
           we should include them in the computation.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
uint8 ComputeConfigCheckSum(void)
{
   uint8 checkSum;
   uint16 size;

   size = (char *)&eep->addrTable[NUM_ADDR_TBL_ENTRIES] - (char *)eep;
   checkSum = CheckSum8(eep, size);
   size = (char *)&nmp->nvFixedTable[NV_TABLE_SIZE] -
          (char *)&nmp->nvConfigTable[0] + 2;
```

广东省网络空间安全协会受控资料

```
    checkSum = checkSum ^ CheckSum8(&nmp->nvConfigTable, size);
    return(checkSum);
}

/*****************************************************************
Function:   IOChanges
Returns:    TRUE if the state of input pin changed.
Reference: None
Purpose:    To determine whether there is a state change in input
            Pin 0.
Comments:   None
*****************************************************************/
Boolean IOChanges(uint8 pinNumberIn)
{
    if (pinNumberIn != 0)
    {
        return(FALSE); /* Only Input Pin 0 is supported for now */
    }

    if (gp->prevPinState[0] == 0 && gp->ioInputPin0)
    {
        /* Prev state = released  curstate = pressed */
        gp->prevPinState[0] = 1;
        return(TRUE);
    }
    if (gp->prevPinState[0] == 1 && ! gp->ioInputPin0)
    {
        /* prevstate = pressed and curstate = released */
        gp->prevPinState[0] = 0;
        return(TRUE);
    }
    return(FALSE);
}

Boolean IsTagBound(uint8 tagIn)
{
    return(tagIn < nmp->snvt.mtagCount &&
            tagIn < NUM_ADDR_TBL_ENTRIES &&
            eep->addrTable[tagIn].addrFormat != UNBOUND);
}

/*****************************************************************
Function: IsNvBound
```

广东省网络空间安全协会受控资料

```
Returns:  TRUE if the variable is bound. FALSE otherwise.
Purpose:  To determine if a primary variable is bound or not.
Comment:  A variable is bound if its address index is not 0xF
          or there is an alias attached to it whose address
          index is not 0xF.
*********************************************/
Boolean IsNVBound(int16 nvIndexIn)
{
   uint16 i;
   int16 primaryIndex;
   uint16 addrIndex;
   uint16 nvAliasTableSize;

   if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize)
   {
      return(FALSE); /* not an index of a primary network variable */
   }

   /* If the primary has a valid address table index and the address
      table entry is not unbound, then the variable is bound */
   addrIndex = nmp->nvConfigTable[nvIndexIn].nvAddrIndex;
   if ( addrIndex != 0x0F &&
        (eep->addrTable[addrIndex].addrFormat != UNBOUND ||
         eep->addrTable[addrIndex].turnaEntry.turnaround == 1) )
   {
      return(TRUE);
   }

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   /* Primary is not bound. See if there is an alias for this variable
      that is bound. */
   for (i = 0; i < nvAliasTableSize; i++)
   {
      primaryIndex = GetPrimaryIndex(i + nmp->nvTableSize);
      addrIndex    = nmp->nvAliasTable[i].nvConfig.nvAddrIndex;
      /* If the alias matches the primary, has a valid address table
         index and the address table entry is not UNBOUND, then
         the primary variable is bound */
      if (primaryIndex == nvIndexIn &&
          addrIndex != 0x0F         &&
          (eep->addrTable[addrIndex].addrFormat != UNBOUND ||
           eep->addrTable[addrIndex].turnaEntry.turnaround == 1) )
```

广东省网络空间安全协会受控资料

```
        {
            return(TRUE);
        }
    }
    return(FALSE); /* No alias entry for this primary that is bound. */
}

/*****************************************************************
Function: AppPgmRuns
Returns:  TRUE if the application program is running on the node.
          FALSE otherwise.
Purpose:  To determine whether the application program is running or not.
          This is used to determine whether to deliver messages, responses,
          and events to the application. Also used to determine whether
          to call DoApp or not.
*****************************************************************/
Boolean AppPgmRuns(void)
{
    /* Normal Mode. Configured and running. */
    if (eep->readOnlyData.nodeState == CNFG_ONLINE &&
        gp->appPgmMode == ON_LINE)
    {
        return(TRUE);
    }

    /* Unconfigured and running. */
    if (eep->readOnlyData.nodeState == APPL_UNCNFG &&
        eep->readOnlyData.runWhenUnconf            &&
        gp->appPgmMode == ON_LINE)
    {
        return(TRUE);
    }

    return(FALSE);
}

/*****************************************************************
Function: NodeConfigured
Returns:  TRUE if the node configured is valid.
Purpose:  To determine whether currently the node is configured.
*****************************************************************/
Boolean NodeConfigured(void)
{
```

广东省网络空间安全协会受控资料

```
   return(eep->readOnlyData.nodeState == CNFG_ONLINE   ||
          eep->readOnlyData.nodeState == CNFG_OFFLINE);
}

/*****************************************************
Function: NodeUnConfigured
Returns:  TRUE if the node is configured.
Purpose:  To determine whether currently node is unconfigured.
*****************************************************/

Boolean NodeUnConfigured(void)
{
   return(eep->readOnlyData.nodeState == APPL_UNCNFG   ||
          eep->readOnlyData.nodeState == NO_APPL_UNCNFG);
}

/*****************End of node.c*****************/
```

A.12 参考实现的 Include 文件

```
/*****************************************************
  File:          CEA709_1.h

  Version:       1.7

  Purpose:       To define constants and types that are needed
                 by all files. Most .c files will include
                 CEA709_1.h either directly or indirectly.

  Note:          Reference implementation does not support
                 special purpose nodes such as routers, repeaters
                 etc. Additional code is required to implement
                 these nodes.

  To Do:         None.
*****************************************************/
#ifndef _CEA709_1_H
#define _CEA709_1_H

/*-----------------------------------------------------
Section: Includes
-----------------------------------------------------*/
/* None */

/*-----------------------------------------------------
```

广东省网络空间安全协会受控资料

```
Section: Macro Definitions
-------------------------------------------------------------------------*/
#define MIN(x,y) (((x)<(y))? (x):(y))
#define MAX(x,y) (((x)>(y))? (x):(y))
#define INCR_STATS(x) ((x < (uint16)0xFFFF)? x++:x)

/*-------------------------------------------------------------------------
Section: Constant Definitions
-------------------------------------------------------------------------*/
/* Reference implementation supports the use of alternate structures such
   as msg_in, msg_out etc for ease of porting application
   programs. However, the disadvantage is the extra copy needed to
   use these structures. Uncomment this constant if you do not want
   to use the alternate structures (i.e you prefer to use reference implementation
   structures that use a different naming convention.) */
#define ALTERNATE_STRUCTURES_NEEDED

#define UNIQUE_NODE_ID_LEN 6 /* Length of the Unique Node Id.              */
#define ID_STR_LEN         8   /* Length of the program id string.          */
#define AUTH_KEY_LEN       6   /* Length of the authentication key.         */
#define DOMAIN_ID_LEN      6   /* Maximum length for a domain id.           */
#define LOCATION_LEN       6   /* Maximum length for location string.       */
#define NUM_COMM_PARAMS    7   /* Max # of parameters for a tranceiver.     */
#define PROTOCOL_VERSION 0   /* 0 for reference implementaion.            */
#define MAX_DOMAINS        2   /* Maximum # of domains allowed.             */

/* Set the size of the array to log error messages from the protocol stack.
   The error messages wrap around, if there are too many errors.
   Errors seldom happen. So, there is no need for this to be too large. */
#define ERROR_MSG_SIZE   1000   /* 20 messages each with 50 chars */

#define FLEX_DOMAIN              2   /* Indicates that the message was received
                                     in flex domain when domain index is 2 */
#define COMPUTE_DOMAIN_INDEX 3   /* When application layer communicates
                                     with transport or session layer,
                                     the domainIndex for the outgoing message
                                     can be either set by the application layer
                                     or computed by transport or session layer
                                     based on the destination address.
                                     This value is used only in TSASenParam
                                     structure. */
#define MAX_GROUP_NUMBER 63      /* Maximum number of a node in a group */
```

广东省网络空间安全协会受控资料

```
/*-----------------------------------------------------------------
Section: Type Definitions
-----------------------------------------------------------------*/
/* The following type definitions need to be changed based on the
   compiler used. The rest of the files should use only int8,
   int16, uint8 etc. Application programs should use
   nint (8-bit int), nlong etc as much as possible. */
typedef char                int8;
typedef short int           int16;
typedef long int            int32;
typedef unsigned char       uint8;
typedef unsigned short int  uint16;
typedef unsigned long int   uint32;

/* Typical 709 definitions for int long etc. */
typedef int8                nshort;
typedef int8                nint;
typedef uint8               nuint;
typedef uint8               nushort;
typedef int16               nlong;
typedef uint16              nulong;

typedef unsigned char Byte;
typedef unsigned char Bits;
typedef int16         MsgTag; /* Lots of Tags!!! */

/* This type definition may need #ifndef macro wrapped
   around when porting to other platforms. */
typedef enum
{
   FALSE = 0,
   TRUE  = 1
} Boolean;

typedef Boolean Boolean; /* For nc compatibility. */

typedef enum
{
   NOBIND = 0,
   NON_BINDABLE = 0,   /* Same as NOBIND. */
   BIND   = 1,
   BINDABLE = 1        /* Same as BIND.   */
} BindNoBind;
```

广东省网络空间安全协会受控资料

```
typedef enum
{
   NV_INPUT   = 0,
   NV_OUTPUT = 1
} NVDirection;

/* Address Types. */
typedef enum
{
   UNBOUND            = 0,

   SUBNET_NODE        = 1,

   UNIQUE_NODE_ID     = 2,

   BROADCAST          = 3,

   MULTICAST          = 4,

   MULTICAST_ACK      = 5,

   BROADCAST_GROUP    = 0x23 /* Used for broadcast req/resp requiring
                                 up to N responses to be delivered. */
} AddrMode;

/* Reference: page 9—49 of Tech Device Data Book. Revision 2.
   In reference implementation, the application is always loaded
   when downloading the code into the system. There is no provision
   to download application program from management tools. However,
   application can be placed in offline by calling GoOffline fn.
   Thus the node state NO_APPL_UNCNFG is not possible.
   gp->appPgmMode indicates the state of the application program. */
typedef enum
{
   /* For nodeState */
   APPL_UNCNFG       = 2, /* Application is loaded but conf is not */
   NO_APPL_UNCNFG    = 3, /* Application is not loaded yet or bad  */
   CNFG_ONLINE       = 4, /* Normal operation mode                 */
   CNFG_OFFLINE      = 6, /* same as hard offline state            */
   HARD_OFFLINE      = 6,

   SOFT_OFFLINE      = 0xC,   /* For reporting purpose              */
```

广东省网络空间安全协会受控资料

```
   CNFG_BYPASS      = 0x8C   /* Not supported in reference imp.    */
} NodeState;

/* These constants are used to represent the mode when the node is
   configured. */
typedef enum
{
   OFF_LINE     = 0,  /* For soft off-line */
   ON_LINE      = 1,  /* For normal mode   */
   NOT_RUNNING = 2   /* For hard-offline. Not used in reference impl.
                        unless commanded to enter this state. */
} ConfigMode;

/* The order of the first 4 items is important as it is used by
   network layer to determine the type of PDU. */
typedef enum
{
   /* Do not change the order. The values are sent across the network */
   TPDU_TYPE,
   SPDU_TYPE,
   AUTHPDU_TYPE,
   APDU_TYPE,

   /* Something extra for internal use of the protocol stack */
   NPDU_TYPE,
   LPDU_TYPE,
} PDUType;

/* Services offered to application program. These are not sent over
   the network */
typedef enum
{
   ACKD,        /* Transport Layer */
   UNACK_RPT,   /* Transport Layer */
   UNACKD,      /* Network   Layer */
   REQUEST,     /* Session   Layer */

   RESPONSE     /* Session   Layer. Used by resp_send function */
} ServiceType;

/* Tranceiver types. Only the constants are used */
typedef enum
{
```

广东省网络空间安全协会受控资料

```
  BLANK              = 0,
  SINGLE_ENDED       = 1,
  SPECIAL_PURPOSE    = 2,
  DIFFERENTIAL       = 5
} TranceiverType;

/* Return status for all functions. */
typedef enum
{
  SUCCESS            = 0,
  FAILURE            = 1,
  INVALID            = 2
} Status;

/* Reset cause */
typedef enum
{
  POWER_UP_RESET     = 0x01,
  EXTERNAL_RESET     = 0x02,
  WATCHDOG_RESET     = 0x0C,
  SOFTWARE_RESET     = 0x14,
  CLEARED            = 0x00
} ResetCause;

/* Turn alignment on so that structures packed tightly */
#pragma maxalign(1)

/* Software Timer definition. */
typedef struct
{
  uint32 curTimerValue;     /* Number of clock ticks left in timer.          */
  uint32 lastUpdatedTime;   /* The time when the timer was last updated.     */
  Boolean expired;          /* TRUE => it has already expired.               */
} MsTimer;


typedef uint8  TransNum;      /* For transaction numbers. */
typedef uint16 RequestId;     /* For matching responses with requests. */

typedef Byte BroadcastAddress;
typedef Byte MulticastAddress;

typedef struct
```

广东省网络空间安全协会受控资料

```
{
   Byte   subnet;
   Bits   selField    :1;   /* Used by nekwork layer for 2b.  */
   Bits   node        :7;
} SubnetAddress;

typedef struct
{
   MulticastAddress group;
   Byte member;
} GroupAddress;

typedef struct
{
   SubnetAddress subnetAddr;
   GroupAddress   groupAddr;    /* Acknowledging group member.  */
} MulticastAckAddress;

typedef struct
{
   Byte    subnet;   /* For routing purpose.  */
   Byte    uniqueId[UNIQUE_NODE_ID_LEN];
} UniqueNodeIdAddress;


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   DestinationAddress is used to indicate network layer that address
   mode is used to send the message.  The destination address
   is always one of the five possibilities.
   domainIndex indicates the domain table to use to determine
   domain length and domain id.
   If domainIndex = FLEX_DOMAIN, then it is flexdomain.
   In this case, src subnet/node is 0/0.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
typedef struct
{
   Byte   domainIndex;
   uint8 flexDomainLen; /* Use if domainIndex == FLEX_DOMAIN  */
   Byte   flexDomainId[DOMAIN_ID_LEN]; /* Use if domainIndex == FLEX_DOMAIN  */
   AddrMode addressMode;
   union {
       BroadcastAddress        addr0;
       MulticastAddress        addr1;
```


广东省网络空间安全协会受控资料

```
        SubnetAddress            addr2a;
        MulticastAckAddress      addr2b;
        UniqueNodeIdAddress      addr3;
    } addr;
} DestinationAddress;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    SourceAddress is used by network layer to indicate to upper levels
    who sent the message and what mode was used.
    Thus this structure is used only for receiving messages.
    domainIndex is used to respond back in the domain in which
    the message was received.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
typedef struct
{
    SubnetAddress          subnetAddr;    /* Subnet of source node. */
    AddrMode               addressMode;   /* What mode used. */
    Byte                   domainIndex;   /* 0 or 1 or FLEX_DOMAIN (i.e 2) */
    uint8                  flexDomainLen; /* Used if domainIndex == FLEX_DOMAIN. */
    Byte  flexDomainId[DOMAIN_ID_LEN]; /* Used if domainIndex == FLEX_DOMAIN. */
    /* group is used only if addressMode is MULTICAST.
       It is the group of the source node sending this message. */
    MulticastAddress       group;
    /* ackNode is used only if addressMode is MULTICAST_ACK.
       It is the destSubnet used and the group of the node sending the ack
       or response. */
    MulticastAckAddress    ackNode;
    /* Destination subnet for broadcast messages. */
    Byte broadcastSubnet;
} SourceAddress;

#pragma maxalign()

#endif    /* #ifndef _CEA709_1 */
/* * * * * * * * * * * * * * * * CEA709_1.h * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   File:          node.h   (node.h)

   Version:       1.7

   Purpose:       To define all the type definitions needed by all
                  the upper layers of the stack.
                  To also define all function to interface with
```

广东省网络空间安全协会受控资料

```
                    some of these data structures as specified in
                    the Device Data Book.

  Note:             None

  To Do:            None.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#ifndef _NODE_H
#define _NODE_H

/* ---------------------------------------------------------------------------
Section: Includes
--------------------------------------------------------------------------- */
#include <CEA709_1.h>
#include <api.h>
#include <queue.h>

/* ---------------------------------------------------------------------------
Section: Constant Definitions
--------------------------------------------------------------------------- */
/* Define the size of the table maintained by the transaction control
   sublayer that keeps track of each possible destination address
   in packets sent to make sure that we don't assign the same TID as in the
   last transaction to that destination. This constant is needed here
   so that we can use it in the type definition of protocol stack data. */
#define TID_TABLE_SIZE 100

/* Init this constant to reflect the number of clock ticks in
   the 32 bit clock for every 1 ms. Some computation may be
   needed to determine this number. For simulation, we consider
   each clock tick as equivalent to 1 ms. */
#ifdef SIMULATION
#define CLOCK_TICKS_PER_MS 1
#else
#define CLOCK_TICKS_PER_MS 0x61
#endif

/* Given a valid primary index of a network variable, get its address */
#define NV_ADDRESS(i) (nmp->nvFixedTable[i].nvAddress)

/* Given a valid primary index of a network variable, get its length */
#define NV_LENGTH(i)  (nmp->nvFixedTable[i].nvLength)
```

广东省网络空间安全协会受控资料

```
/* Given a valid primary index of a network variable, check if it is sync */
#define NV_SYNC(i)      (nmp->nvFixedTable[i].nvSync)

/*-----------------------------------------------------------------
Section: Type Definitions
-----------------------------------------------------------------*/
/* Reference: Tech Device Data: Page 9-6. */

/* Turn off alignment by compiler to make sure that the structure sizes
   are as we intend them to be with no padding */
#pragma maxalign(1)
typedef struct
{
   Byte  uniqueNodeId[UNIQUE_NODE_ID_LEN]; /* 48-bit unique ID.                  */
   Byte  modelNum;                          /* Model Number for Ref. Impl.        */
   Bits  checkSum          : 4;             /* For Unique Node ID.                */
   Bits  minorModelNum     : 4;             /* 0-128.                             */
   Byte  nvFixed[2];                        /* Location of nv fixed table.        */
   Bits  readWriteProtect  : 1;             /* read+write protect flag.           */
   Bits  runWhenUnconf     : 1;             /* 1=> Application runs.              */
   Bits  nvCount           : 6;             /* 0 for reference implementation.    */
   Byte  snvtStruct[2];                     /* 0xFFFF for reference implementation. */
   char  progId[ID_STR_LEN];                /* Program Id string.                 */
   Bits                    : 1;             /* nvProcessingOff not used in Ref. Impl. */
   Bits  twoDomains        : 1;             /* 1 if node is in 2 domains.         */
   Bits                    : 1;             /* explicitAddr not used in Ref. Impl. */
   Bits                    : 1;             /* Unused.                            */
   Bits                    : 1;             /* Unused.                            */
   Bits  nodeState         : 3;             /* Node State. See CEA709_1.h         */
   Bits  addressCnt        : 4;             /* # of entries in address table.     */
   Bits                    : 4;             /* Unused.                            */
   Bits                    : 4;             /* Unused.                            */
   Bits  receiveTransCnt   : 4;             /* RR Cnt = this field + 1            */
   Bits  appOutBufSize     : 4;             /* Special Size Encoding.             */
   Bits  appInBufSize      : 4;             /* Special Size Encoding.             */
   Bits  nwOutBufSize      : 4;             /* Special Size Encoding.             */
   Bits  nwInBufSize       : 4;             /* Special Size Encoding.             */
   Bits  nwOutBufPriCnt    : 4;             /* Special Count Enconding.           */
   Bits  appOutBufPriCnt   : 4;             /* Special Count Enconding.           */
   Bits  appOutBufCnt      : 4;             /* Special Count Enconding.           */
   Bits  appInBufCnt       : 4;             /* Special Count Enconding.           */
   Bits  nwOutBufCnt       : 4;             /* Special Count Enconding.           */
   Bits  nwInBufCnt        : 4;             /* Special Count Enconding.           */
```

广东省网络空间安全协会受控资料

```
    Byte   reserved0;                    /* Unused                                    */
    Byte   reserved1[2];                 /* Unused.                                   */
    Byte   reserved2[3];                 /* Unused.                                   */
    Bits                        : 6;     /* Unused.                                   */
    Bits   txByAddress          : 1;     /* 0 in reference implementation.            */
    Bits                        : 1;     /* Unused.                                   */
    Bits                        : 2;     /* Unused.                                   */
    Bits aliasCnt               : 6;     /* 0 in reference implementation.            */
    Bits msgTagCnt              : 4;     /* 0 in reference implementation.            */
    Bits                        : 4;     /* Unused.                                   */
    Byte reserved3[3];                   /* Unused.                                   */
} ReadOnlyDataStruct;

/* Reference: Tech Device Data Book. Page 9—11. */
typedef struct
{
    Byte domainId[DOMAIN_ID_LEN];
    Byte subnet;
    Bits cloneDomain : 1;
    Bits node        : 7;
    Byte len;
    Byte key[AUTH_KEY_LEN];
} DomainStruct;

typedef struct
{
    AddrMode   addrMode;                 /* UNBOUND (0) => turnaround.                */
    Byte       turnaround;               /* 1=>turnaround; 0=>not in use. */
    Bits       unacdkRptTimer   : 4;     /* unackd_rpt timer.                         */
    Bits       retryCount       : 4;     /* retry count.                              */
    Bits                        : 4;     /* UNUSED.                                   */
    Bits       txTimerIndex     : 4;     /* transmit timer index.                     */
} TurnaroundStruct;

/* Other types used in the following structure are defined in api.h. */
typedef union
{
    uint8              addrFormat; /* Format of entry. */
    GroupAddrMode      groupEntry;
    SNodeAddrMode      snodeEntry;
    BcastAddrMode      bcastEntry;
    TurnaroundStruct   turnaEntry;
} AddrTableEntry;
```

广东省网络空间安全协会受控资料

```
/* Reference: Tech Device Data: Page 9-24 */
typedef struct
{
   Bits collisionDetect     :1;
   Bits bitSyncThreshHold   :2; /* Encoded. See page 9-28 */
   Bits filter              :2;
   Bits hysteresis          :3;
   Bits unUsed              :6;
   Bits cdTail              :1;
   Bits cdPreamble          :1;
} DirectParamStruct;

typedef struct
{
   int16      channelId;                  /* Id of channel assigned */
   Byte       location[LOCATION_LEN]; /* 6 byte ascii string      */
   Bits       commClock     :5;  /* bit rate ratio      */
   Bits       inputClock    :3;  /* input clk osc freq      */
   Bits       commType      :3;          /* type of receiver      */
   Bits       commPinDir    :5;          /* dir comm port pins      */
   Byte       reserved[5];
   Byte       nodePriority;          /* priority slot used      */
   Byte       channelPriorities;        /* # of priority slots      */
   union {
      Byte                xcvrParams[NUM_COMM_PARAMS];
      DirectParamStruct   dirParams;
   } param;
   Bits       nonGroupTimer      :4;
   Bits       nmAuth             :1; /* NetMgt msgs needs auth? */
   Bits       preemptionTimeout :3; /* free buffer wait time in
                                          preemption mode          */
} ConfigDataStruct;

/* Reference: Tech Device Data: Page 9-17 */
/* Network Variable Tables */
typedef struct
{
   Bits nvPriority      :1;  /* NV uses priority messaging        */
   Bits nvDirection     :1;  /* 1 => output var                   */
   Bits nvSelectorHi    :6;  /* Hi & Lo form NV selector          */
   Byte nvSelectorLo      ;  /* Range 0-0x3FFF                    */
   Bits nvTurnaround    :1;  /* 1 ==> bound to a nv in this node */
```

广东省网络空间安全协会受控资料

```
   Bits nvService      :2;  /* ACKD or UNACKD_RPT or UNACKD      */
   Bits nvAuth         :1;  /* 1 => NV used auth transactions    */
   Bits nvAddrIndex    :4;  /* index into addr table. 15 is spcl */
} NVStruct;

typedef struct
{
   NVStruct nvConfig;
   uint8    primary;    /* index into NV Cfg tbl. 0xFF => next fld */
   uint16   hostPrimary; /* NV cfg tbl index. For host nodes */
} AliasStruct;

typedef struct
{
   Bits nvSync     :1; /* 1 ==> Synchronous network var */
   Bits            :2; /* unused */
   Bits nvLength   :5; /* # of bytes in the Network Var */
   void *nvAddress;    /* Ptr to variable's data */
} NVFixedStruct;

/* Statistics */
typedef struct
{
   uint16 transmissionErrors;
   uint16 transmitTXFailures;
   uint16 receiveTXFull;
   uint16 lostMessages;
   uint16 missedMessages;
   uint16 layer2Received;
   uint16 layer3Received;
   uint16 layer3Transmitted;
   uint16 transmitTXRetries;
   uint16 backlogOverflow;
   uint16 lateAcknowledgements;
   uint16 collisions;
   uint8  eepromLock;
   uint16 expansion[11]; /* Room for expansion */

   /* Reference Implementation Specific Statistics */
   uint16 layer6_7MsgsRcvd;
   uint16 layer6_7RespRcvd;
   uint16 layer6_7MsgsSent;
   uint16 layer6_7RespSent;
```

广东省网络空间安全协会受控资料

```
    uint16 lateResponses;      /* No matching req */
    uint16 lateChallenges;     /* No matching transmit record */
    uint16 lateReplies;        /* No matching receive  record */
} StatsStruct;

/* yes: Yes, error is implemented in reference implementation */
/* na:  Not applicable (not implemented) in ref. imp.         */
typedef enum
{
    NO_ERRORS                          = 0,    /* yes */
    BAD_EVENT                          = 129,  /* na  */
    NV_LENGTH_MISMATCH                 = 130,  /* yes */
    NV_MSG_TOO_SHORT                   = 131,  /* yes */
    EEPROM_WRITE_FAIL                  = 132,  /* na  */
    BAD_ADDRESS_TYPE                   = 133,  /* yes */
    PREEMPTION_MODE_TIMEOUT            = 134,  /* na  */
    ALREADY_PREEMPTED                  = 135,  /* na  */
    SYNC_NV_UPDATE_LOST                = 136,  /* na  */
    INVALID_RESP_ALLOC                 = 137,  /* na  */
    INVALID_DOMAIN                     = 138,  /* yes */
    READ_PAST_END_OF_MSG               = 139,  /* na  */
    WRITE_PAST_END_OF_MSG              = 140,  /* na  */
    INVALID_ADDR_TABLE_INDEX           = 141,  /* yes */
    INCOMPLETE_MSG                     = 142,  /* na  */
    NV_UPDATE_ON_OUTPUT_NV             = 143,  /* na  */
    NO_MSG_AVAIL                       = 144,  /* na  */
    ILLEGAL_SEND                       = 145,  /* na  */
    UNKNOWN_PDU                        = 146,  /* yes */
    INVALID_NV_INDEX                   = 147,  /* yes */
    DIVIDE_BY_ZERO                     = 148,  /* na  */
    INVALID_APPL_ERROR                 = 149,  /* na  */
    MEMORY_ALLOC_FAILURE               = 150,  /* yes */
    WRITE_PAST_END_OF_NET_BUFFER       = 151,  /* na  */
    APPL_CS_ERROR                      = 152,  /* na  */
    CNFG_CS_ERROR                      = 153,  /* na  */
    INVALID_XCVR_REG_ADDR              = 154,  /* na  */
    XCVR_REG_TIMEOUT                   = 155,  /* na  */
    WRITE_PAST_END_OF_APPL_BUFFER      = 156,  /* na  */
    IO_READY                           = 157,  /* na  */
    SELF_TEST_FAILED                   = 158,  /* na  */
    SUBNET_ROUTER                      = 159,  /* na  */
    AUTHENTICATION_MISMATCH            = 160,  /* yes */
    SELF_INST_SEMAPHORE_SET            = 161,  /* na  */
```

广东省网络空间安全协会受控资料

```
    READ_WRITE_SEMAPHORE_SET          = 162,  /* na   */
    APPL_SIGNATURE_BAD                = 163,  /* na   */
    ROUTER_FIRMWARE_VERSION_MISMATCH = 164   /* na   */
} ErrorLog;

/*-----------------------------------------------------------------
  NWSendParam is used to process Send Request to Network Layer
  from other layers. When other layers deposit a new PDU into the
  Output Queue, they should also fill in the details of this
  data structure to indicate the kind of request.

  altPath is set by transport/session layers for the last 2 retries.
  destAddr has the domainIndex that is used to determine the
  domainId, source subnet/node. The possible values for the
  domainIndex are 0 or 1 or 2. 0 or 1 ==> use the corresponding
  domainTable. 2 ==> use the flex domain.
-----------------------------------------------------------------*/
typedef struct
{
    DestinationAddress    destAddr; /* To give dest addr  */
    PDUType               pduType;  /* APDU, SPDU, TPDU, AuthPDU */
    MsgTag                tag;      /* used only for APDU */
    uint8                 deltaBL;  /* Upper Layers supply this */
    Boolean               altPath;
    uint16                pduSize;  /* Size of the PDU sent */

    Boolean               dropIfUnconfigured; /* drop the packet if the
                              node is unconfigured. */
} NWSendParam;

/*-----------------------------------------------------------------
  NWReceiveParam: Is used by Link Layer when it supplies an incoming
  NPDU to the network layer. This data structure contains information
  regarding the incoming NPDU.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
typedef struct
{
    Boolean               priority;  /* Was it a priority message? */
    Boolean               altPath;
    uint16                pduSize;   /* Size of NPDU in queue */
} NWReceiveParam;

/* Type Definitions for Application Layer */
```

广东省网络空间安全协会受控资料

```
typedef struct
{
   DestinType code;
   Byte data[MAX_DATA_SIZE];
} APDU;

typedef APDU * APDUPtr;

/* APPSendParam is used by API to give info to app layer
   regarding APDU that is going out. Most info is available
   in msg_out in the Application Out Buffers */

/* APPSendParam has everything in MsgOut except code and data. */
typedef struct
{
   MsgTag        tag;              /* to correlate completion codes   */
   uint8         len;              /* message length in app data      */
   Boolean       authenticated;    /* TRUE if to be authenticated     */
   ServiceType   service;          /* service type used to send msg   */
   RequestId     reqId;            /* Request ID for responses */
   MsgOutAddr    addr;             /* destination address (see above) */
   Boolean       nullResponse;     /* For responses                   */
} APPSendParam;

/* Types of messages that are received by application layer */
typedef enum {
   MESSAGE      = 0,
   COMPLETION = 1,   /* Indication by Transport/Session layers */
} APIndType;


/* APPReceiveParam is used by Application Layer to figure out
   information regarding incoming APDU. */
typedef struct
{
   APIndType        indication; /* Type of APDU received */
   Boolean          success;    /* Used if indication ==
                                   COMPLETION */
   MsgTag           tag;        /* Tag for indication
                                   or for matching response */
   SourceAddress    srcAddr;    /* Who sent it?              */
   ServiceType      service;    /* Which service type?       */
   Boolean          priority;   /* Was it a priority msg?    */
```

广东省网络空间安全协会受控资料

```
    Boolean                 altPath;      /* Was it sent in alt path? */
    uint16                  pduSize;      /* Size of incoming APDU    */
    uint16                  taIndex;      /* Turnaround var Index     */
    Boolean                 auth;         /* Was it authenticated?    */
    RequestId               reqId;        /* Assigned by session to match
                                                responses later        */
} APPReceiveParam;

/* Type Definition for Transaction Control SubLayer */
typedef struct
{
    TransNum      transNum;        /* initial value = 0. Range 0..15 */
    Boolean       inProgress;      /* Is the transaction in progress */
} TransCtrlRecord;

/* Type definition for table used while assigning new TID. */
/* For more information, see tcs.h or tcs.c */
typedef struct
{
    Byte   domainId[DOMAIN_ID_LEN];
    uint8 len; /* domain length */
    AddrMode                addressMode;
    union
    {
        SubnetAddress          subnetNode;
        MulticastAddress       group;    /* group number of multicast */
        BroadcastAddress       subnet;   /* 0 if domainwide broadcast */
        Byte                   uniqueNodeId[UNIQUE_NODE_ID_LEN];
    } addr;
    MsTimer                    timer;
    TransNum                   tid;      /* Last TID Used for this addr */
} TIDTableEntry;

/* Type Definitions for transport, Session, Auth Layers */

typedef enum
{
    TRANS_CURRENT,
    TRANS_NOT_CURRENT,
    TRANS_NEW,
    TRANS_DUPLICATE
} TransStatus;
```

广东省网络空间安全协会受控资料

```
typedef enum
{
   UNUSED_TX,          /* Transmit Record is unused */
   SESSION_TX,         /* Transmit Record is used by Session */
   TRANSPORT_TX        /* Transmit Record is used by Transport */
} TXStatus;

typedef enum
{
   UNUSED_RR,          /* Receive Record is unused */
   SESSION_RR,         /* Receive Record is used by Session */
   TRANSPORT_RR        /* Receive Record is used by Transport */
} RRStatus;

typedef enum
{
   JUST_RECEIVED,      /* Msg just received. Nothing has been done      */
   DELIVERED,          /* Msg has been delivered to app layer. Receive
                       Timer has not expired yet. Waiting for Resp */
   DONE,               /* Msg delivered & a null resp received.
                       Or Msg delivered and Server received has received
                       the response.
                       Timer has not expired yet */
   AUTHENTICATED,      /* Msg has been authenticated. Not delivered      */
   AUTHENTICATING,     /* Msg is being authenticated. reply expected     */
   RESPONDED,          /* Rsp has been recvd from app layer & responded
                       at least once. Recv Timer not expired yet      */
} TransactionState;  /* Constant used by Transport & Session */

typedef struct
{
   TXStatus            status;                /* Who owns it? if not free   */
   DestinationAddress nwDestAddr;             /* Destination Address */
   Boolean             ackReceived[MAX_GROUP_NUMBER+1];
                       /* Array[0..MAX_GROUP_NUMBER] of Boolean     */
   uint8               destCount;             /* Number of destinations     */
   uint8               ackCount;              /* Or respCount               */
   TransNum            transNum;
   uint16              xmitTimerValue;
   MsTimer             xmitTimer;             /* Transmit Timer             */
   uint8               retriesLeft;           /* How many left?             */
   APDU                *apdu;                 /* APDU transmitted           */
   uint16              apduSize;              /* Size of APDU               */
```

广东省网络空间安全协会受控资料

```
   Boolean              auth;                    /* Does this msg need auth?     */
} TransmitRecord;

typedef struct
{
   RRStatus             status;          /* used? Who is using?        */
   SourceAddress        srcAddr;         /* Who sent this?             */
   TransNum             transNum;
   RequestId            reqId;           /* For matching response    */
   MsTimer              recvTimer;       /* receive timer              */
   TransactionState     transState;      /* What state is it in      */
   Boolean              priority;
   Boolean              altPath;         /* Was alt path used?         */
   Boolean              auth;            /* TRUE if auth succeeded */
   Boolean              needAuth;        /* Need authentication        */
   ServiceType          serviceType;     /* What type of service     */
   Byte                 rand[8];         /* For authentication         */
   APDU                 *response;       /* Store the response         */
   uint16               rspSize;
   APDU                 *apdu;           /* Store the APDU received */
   uint16               apduSize;
} ReceiveRecord;

/*********************************************************************
   TSASendParam is used to provide the necessary information
   needed to process the transaction.
   MsgOutAddr has all the necessary information for forming the
   destination address. It also has the domainIndex that can be
   used to determine the domainId, source subnet/node.
   However to facilitate sending of messages even when a node
   is not in any domain (such as sending ManualServiceRequest) this structure
   provides a way to specify the domainIndex with different
   possible values.
   domainIndex = 0 or 1  --> use the corresponding domain table
   domainIndex = 2 ---> use the flexDomain given
   domainIndex = 3 ---> use the domain table as given in destAddr.
   For response messages, several fields such as the destAddr or
   domainIndex is not needed. They are determined from the
   corresponding request message in the receive records.
*********************************************************************/
typedef struct
{
   MsgOutAddr           destAddr;   /* Whom to send? Need Timers too */
```

广东省网络空间安全协会受控资料

```
    Byte   domainIndex;
    uint8 flexDomainLen;
    Byte   flexDomainId[DOMAIN_ID_LEN];
    ServiceType          service;    /* What type of service? */
    Boolean              auth;       /* Need Authentication?  */
    uint16               apduSize;   /* Size of APDU to be sent */
    MsgTag               tag;        /* For service indication  */
    Boolean              nullResponse; /* TRUE => no resp goes out */
    RequestId            reqId;      /* Set if service=RESPONSE */

    Boolean              altPathOverride;
    Boolean              altPath; /* Used only if altPathOverride is true */
} TSASendParam;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    TSAReceiveParam is used to receive the necessary
    information to process incoming PDU (TPDU or SPDU or AuthPDU).
    Receive Timer is set based on info in srcAddr.
    srcAddr has the source subnet/node, domainIndex used, group if
    any etc.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
typedef struct
{
    SourceAddress        srcAddr;    /* Who send it? */
    Boolean              priority;   /* Was it a priority msg?  */
    uint16               pduSize;    /* Size of incoming PDU */
    PDUType              pduType;    /* What type of PDU? */
    Boolean              altPath;    /* Was it sent in alt path? */
} TSAReceiveParam;

typedef struct
{
    uint8   deltaBL;   /* What is the backlog generated by this msg? */
    Boolean altPath;   /* Should altPath be used? */
    uint16  pduSize;   /* Size of NPDU */
} LKSendParam;

/* SNVT data structures */

/* See comment in app.c in APPReset and AddNV for a description */
/* of how the SNVT area is layed out and managed.               */

typedef struct
```

广东省网络空间安全协会受控资料

```
{
   Bits   extRec              : 1;
   Bits   nvSync              : 1;
   Bits   nvPolled            : 1;
   Bits   nvOffline           : 1;
   Bits   nvServiceConfig     : 1;
   Bits   nvPriorityConfig    : 1;
   Bits   nvAuthConfig        : 1;
   Bits   nvConfigClass       : 1;
   uint8 snvtTypeIndex;
} SNVTdescStruct;

typedef struct
{
   Bits          bindingII     : 1;
   Bits          queryStats    : 1;
   Bits          aliasCount    : 6;
   uint16        hostAlias;
} AliasField;

typedef struct
{
   uint16        length;
   uint8         numNetvars;
   uint8         version;
   uint8         msbNumNetvars;
   uint8         mtagCount;
   char          sb[SNVT_SIZE];
   SNVTdescStruct   * descPtr; /* Point to next SNVTdesc entry in sb */
                               /* Actually, it points Node Self-Doc
                                  String. Just before that is the
                                  self-id info for the last netwar */
   AliasField       * aliasPtr; /* Point to alias structure in sb */
} SNVTstruct;

typedef struct
{
   Bits   mre   : 1;     /* Maximum rate     */
   Bits   re    : 1;     /* Rate             */
   Bits   nm    : 1;     /* Name             */
   Bits   sd    : 1;     /* Self Doc String */
   Bits   nc    : 1;     /* Array Count      */
   Bits         : 3;     /* Unused           */
```

广东省网络空间安全协会受控资料

```
} SNVTextension;

/* Partial list of SNVT type index values */
/* Unused in Ref Impl. Given for illustration */
typedef enum
{
    SNVT_STR_ASC   = 36,
    SNVT_LEV_CONT = 21,
    SNVT_LEV_DISC = 22,
    SNVT_COUNT_F   = 51,
} SNVTType;

typedef struct
{
   int16 nvIndex; /* Base index of array network variables */
   int16 dim;     /* The dimension of the array */
} NVArrayTbl;

/* Type Definition for Protocol Stack Data */
typedef struct
{
   /* Number of bytes used so far */
   uint16   mallocUsedSize;

   /* Array of storage space for dynamic allocation of buffers etc */
   Byte mallocStorage[MALLOC_SIZE];

   /* Variables to store error messges generated using ErrorMsg fn */
   char                errorMsg[ERROR_MSG_SIZE];
   uint16              errorMsgNumber;
   uint16              errorMsgIndex;

   /* Variable to keep track of current Time. Points to time */
   uint32              softwareTime;
   uint32             *currentTime; /* Points to softwareTime or
                                       actual hardware timer loc */

   /* Variables for Transaction Control Sublayer */
   TransCtrlRecord priTransCtrlRec;

   TransCtrlRecord nonpriTransCtrlRec;

   TransNum        priTransID;
```

广东省网络空间安全协会受控资料

```
TransNum        nonpriTransID;

TIDTableEntry priTbl[TID_TABLE_SIZE];
TIDTableEntry nonpriTbl[TID_TABLE_SIZE];
/* # entries currently used */
uint16          priTblSize;
uint16          nonpriTblSize;

/* Timer to delay Transport/Session layers after an external or
   power-up reset. */
MsTimer tsDelayTimer;

/* Transmit and Receive Records */
TransmitRecord xmitRec;
TransmitRecord priXmitRec;

ReceiveRecord   *recvRec;   /* Pool of records */
uint16 recvRecCnt;          /* How many Records allocated? */

RequestId reqId; /* Running count for request numbers */
Byte        prevChallenge[8]; /* Used in generation of new challenge. */

/* Various Queues */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  There are 3 queues associated with Each Layer except Physical.
  1. Input  Queue 2. Output Queue 3. Output Priority Queue

  Each item in a queue has appropriate Param Structure
  followed by appropriate PDU.

  Buffer Sizes and Count Values are available in readOnlyData
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Input Queue For App Layer */
Queue        appInQ;
uint16      appInBufSize;
uint16      appInQCnt;

/* Output Queue For APP Layer */
Queue        appOutQ;
uint16      appOutBufSize;
uint16      appOutQCnt;

/* Output Priority Queue for APP Layer */
```

广东省网络空间安全协会受控资料

```
Queue        appOutPriQ;
uint16       appOutPriBufSize;
uint16       appOutPriQCnt;

/* Input Queue For Transport, Session, Authentication Layers */
Queue        tsaInQ;
uint16       tsaInBufSize;
uint16       tsaInQCnt;

/* Output Queue For Transport, Session, Authentication Layers */
Queue        tsaOutQ;
uint16       tsaOutBufSize;
uint16       tsaOutQCnt;

/* Output Priority Queue For Transport, Session, Auth Layers */
Queue        tsaOutPriQ;
uint16       tsaOutPriBufSize;
uint16       tsaOutPriQCnt;

/* Output Queue for Responses. Just one queue is sufficient.
   Priority or Non-priority is determined based on request */
Queue        tsaRespQ;
uint16       tsaRespBufSize;
uint16       tsaRespQCnt;

/* Input Queue For network Layer */
Queue        nwInQ;
uint16       nwInBufSize;
uint16       nwInQCnt;

/* Output Queue For network Layer */
Queue        nwOutQ;
uint16       nwOutBufSize;
uint16       nwOutQCnt;

/* Output Priority Queue For network Layer */
/* Buffer size is same as the buffer size for Output Queue */
Queue        nwOutPriQ;
uint16       nwOutPriBufSize;
uint16       nwOutPriQCnt;

/* Input Queue For Link Layer */
Byte         *lkInQ;
```

广东省网络空间安全协会受控资料

```
uint16          lkInBufSize; /* Size of buffer in lkInPDUQ */
uint16          lkInQCnt;    /* # of Buffers allocated. */
Byte           *lkInQHeadPtr;
Byte           *lkInQTailPtr;

/* Output Queue For Link Layer */
Queue           lkOutQ;
uint16          lkOutBufSize;
uint16          lkOutQCnt;

/* Output Priority Queue For Link Layer */
/* Buffer size is same as the buffer size for Output Queue */
Queue           lkOutPriQ;
uint16          lkOutPriBufSize;
uint16          lkOutPriQCnt;

/* Output Queue For Physical Layer */
Byte       *phyOutQ; /* Not a regular Queue unlike others */
uint16     phyOutBufSize;
uint16     phyOutQCnt;
Byte       *phyOutQHeadPtr;
Byte       *phyOutQTailPtr;

/* Output Priority Queue For Physical Layer */
/* Buffer size is same as the buffer size for Output Queue */
Byte       *phyOutPriQ; /* Not a regular Queue unlike others */
uint16     phyOutPriBufSize;
uint16     phyOutPriQCnt;
Byte       *phyOutPriQHeadPtr;
Byte       *phyOutPriQTailPtr;

/* API Flags */
Boolean msgReceive;     /* TRUE when data in gp->msgIn   */
Boolean respReceive;    /* TRUE when data in gp->respIn */

Boolean callMsgFree;    /* Flag to help implicit call to msg_free after DoApp */
Boolean callRespFree;   /* Flag to help implicit call to resp_free after DoApp */

/* API Variables */
RespIn   respIn;
RespOut respOut;
MsgIn    msgIn;
MsgOut   msgOut;
```

广东省网络空间安全协会受控资料

```
/* Flag to determine if selected for Net Mgmt queries */
Boolean selectQueryFlag;

/* Unbound selector counter for automatic assignement in AddNV. */
uint16 unboundSelector;

/* Table to keep track of array network variables and dim */
NVArrayTbl nvArrayTbl[MAX_NV_ARRAYS];
uint16     nvArrayTblSize;

/* Queue of nvIndex for network output variables.
   This queue stores the indices of network variables
   (primary or alias) that are scheduled for sending out
   NVUpdate messages. Each index takes 2 bytes. The index
   is optionally followed by the value of the variable to
   be used with the NVUpdate message. This is done for sync
   network output variables so that we use the value when the
   request was made rather than the current value.

   Update to one network output variable may involve zero or one (primary)
   and zero or more alias indices. Thus we have a collection of indices to be
   scheduled. Alias entries can have different service type or priority
   attributes. To make scheduling work properly, we will have only one
   queue for storing these indices. After scheduling all the alias
   indices, if any, for the primary, we will add an extra entry
   that is equal to NV_UPDATE_LAST_TAG_VALUE to indicate end of
   one scheduling of a primary. When these indices are actually processed,
   we will process one at a time. i.e. we don't send NV message unless
   the previous one is completed.

   An NVUpdate for a primary succeeds if all the transactions for
   scheduled for the NVUpdates succeed. We keep track of this in
   the flag nvOutStatus. The flag nvOutSchedule is set to TRUE if
   scheduling can be done, FALSE otherwise. This is reset to TRUE at
   the completion of the primary and all alias schedules. */

Queue      nvOutIndexQ;
uint16     nvOutIndexQCnt;
uint16     nvOutIndexBufSize;
Boolean    nvOutStatus;      /* Used to give NVUpdateCompletes for Propagate. */
Boolean    nvOutCanSchedule; /* TRUE -> can continue to schedule. */
int16      nvOutIndex;       /* current primary index scheduled.  */
```

广东省网络空间安全协会受控资料

```
/* Queue of nvIndex for network input variables.
   This queue stores the input variables that scheduled
   to be polled. Each item exactly 2 bytes to store the index
   of the variable scheduled. Also, see the comment above for the
   nvOutIndexQ. The comment is valid for this queue for the poll
   messages too.

   Polling one network input variable may involve zero or one (primary)
   and zero or more alias indices. Thus we have a collection of indices to be
   scheduled. The alias can have different service type or priority.

   A poll succeeds if both nvInDataStatus and nvInTranStatus are true. */

Queue       nvInIndexQ;
uint16      nvInIndexQCnt;
Status      nvInDataStatus; /* true if any valid data from external node
                               is received or nv is turnaround only. */
Status      nvInTranStatus; /* true if all transactions for the poll succeeded. */
Boolean     nvInCanSchedule; /* TRUE -> can continue to schedule. */
int16       nvInIndex;       /* current primary index scheduled.  */

int16       nvArrayIndex;
NvInAddr    nvInAddr;

/* Flag to indicate scheduler whether reset is needed or not */
Boolean resetNode; /* TRUE ==> reset needed */
Boolean resetPinPrevState;  /* For PHYIO's use */

/* To Check if reset was successful */
Boolean resetOk;

/* To Check if the manual service request button was pressed or not */
Boolean manualServiceRequest;
Boolean manualServiceRequestPrevState;  /* For PHYIO's use */

/* To Check if generic io input pin 0 is pressed or not */
Boolean ioInputPin0;
Boolean ioInputPin0PrevState; /* For PHYIO's use */

/* To set generic io output pin 0 */
Boolean ioOutputPin0;
```

广东省网络空间安全协会受控资料

```
    /* To set generic io output pin 1 */
    Boolean ioOutputPin1;

    /* To Remember prev state of the buttons. For IOChanges's use */
    Boolean prevPinState[1]; /* Currently only one pin is supported */

    /* Represents the mode for the application program in configured state. */
    Byte appPgmMode;  /* Possible Values: OFF_LINE OFF_LINE NOT_RUNNING */

    /* For Msg Tag assignments */
    uint16 nextBindableMsgTag;
    uint16 nextNonbindableMsgTag;
} ProtocolStackData;

typedef struct
{
    ReadOnlyDataStruct   readOnlyData;
    ConfigDataStruct     configData;
    DomainStruct         domainTable[MAX_DOMAINS];
    AddrTableEntry       addrTable[NUM_ADDR_TBL_ENTRIES];

    /* Checksum for config structure */
    Byte    configCheckSum; /* Exclusive or of successive bytes in
                               config structure */

    /* Define some space above EEPROM to allow EEPROM expansion */
    /* The following variable makes the EEPROM exactly 3k in size */
    /* If actual EEPROM size is < 3k, it should handle addresses
       that are not in range */
    Byte                 reserved[3 * 1024 -
                                  sizeof(ReadOnlyDataStruct) -
                                  sizeof(ConfigDataStruct)   -
                                  sizeof(DomainStruct) * MAX_DOMAINS  -
                                  sizeof(AddrTableEntry) * NUM_ADDR_TBL_ENTRIES -
                                  1];

} EEPROM;

typedef struct
{
    char        pxyType;     /* Proxy type (Green Book p. 51). -1 => not in use. */
    RequestId   reqId;       /* Request ID of proxy */
} ProxyData;
```

广东省网络空间安全协会受控资料

```
typedef struct
{
   /* Firmware space. Not used for anything in Ref Imp.
      Usable by users */
   Byte            firmwareSpace[16 * 1024]; /* 16 K size */
   Byte            userSpace[42 * 1024];     /* User space */
   /* RAM starts here */
   StatsStruct     stats;
   SNVTstruct      snvt;
   ProxyData       pxyData;
   ErrorLog        errorLog;
   uint8           resetCause;
   Byte            restOfRAM[2 * 1024 -
                     sizeof(StatsStruct) -
                     sizeof(SNVTstruct) -  sizeof(ProxyData) -
                     sizeof(ErrorLog) - sizeof(uint8)];
   /* EEPROM goes here. But kept separate as it can be loaded
      into Actual EEPROM more easily if it is isolated */
   /* Memory Mapped Space */
   Byte            memMapSpace[1024]; /* 1k in size */

   /* Place network varibale information above the 64K nmap so
      that we can support large number of variables. Since these
      are not kept in EEPROM, some other mechanism should be used
      to make sure that these values are preserved after a power up */
   NVStruct            nvConfigTable[NV_TABLE_SIZE];
   AliasStruct         nvAliasTable[NV_ALIAS_TABLE_SIZE];
   NVFixedStruct       nvFixedTable[NV_TABLE_SIZE];
   uint16              nvTableSize; /* Config or Fixed */
} NmMap; /* Memory Map */

/* Turn the alignment off */
#pragma maxalign()

/*-----------------------------------------------------------------
Section: Global Variables
-----------------------------------------------------------------*/
/* Node Data Structures */
extern   ProtocolStackData   protocolStackDataGbl[NUM_STACKS];
extern   ProtocolStackData   *gp; /* Pointer to current Structure */
extern   EEPROM              eeprom[NUM_STACKS];
extern   EEPROM              *eep; /* Pointer to current eeprom str */
```

广东省网络空间安全协会受控资料

```
extern    NmMap                    nm[NUM_STACKS];
extern    NmMap                    * nmp;

#if defined(ALTERNATE_STRUCTURES_NEEDED)
extern msg_in_type       msg_in;
extern msg_out_type      msg_out;
extern resp_in_type      resp_in;
extern resp_out_type     resp_out;
extern nv_in_addr_type nv_in_addr;
extern int16             nv_array_index;
#endif

/*-----------------------------------------------------------------
Section: Function Prototypes
-----------------------------------------------------------------*/
DomainStruct * AccessDomain(uint8 indexIn);
void      UpdateDomain(DomainStruct * domainInp, uint8 indexIn);
void      UpdateCloneDomain(DomainStruct * domainInp, uint8 indexIn);
AddrTableEntry * AccessAddress(uint16 indexIn);
void      UpdateAddress(AddrTableEntry * addrEntryInp, uint16 indexIn);
uint16    AddrTableIndex(Byte domainIndexIn, uint8 groupIn);
Boolean IsGroupMember(Byte domainIndex, uint8 groupIn,
                      uint8 * groupMemberOut);
uint16    DecodeBufferSize(uint8 bufSizeIn);
uint16    DecodeBufferCnt(uint8 bufCntIn);
uint16    DecodeRptTimer(uint8 rptTimerIn);
uint16    DecodeRcvTimer(uint8 rcvTimerIn);
uint16    DecodeTxTimer(uint8 txTimerIn);
uint16    DecodeNonGroupTimer(uint8 nonGroupTimerIn);
NVStruct * AccessNV(uint16 indexIn);
void      UpdateNV(NVStruct * nvStructInp, uint16 indexIn);
AliasStruct * AccessAlias(uint16 indexIn);
void      UpdateAlias(AliasStruct * aliasStructInp, uint16 indexIn);
uint16    NVTableIndex(char varNameIn[]);
uint16    AliasTableIndex(char varNameIn[]);
void      ErrorMsg(char errMessageIn[]);
void      DebugMsg(char debugMsg[]);
void     * AllocateStorage(uint16 size);
uint32    GetCurrentTime(void);
uint32    GetCurrentMsTime(void);
void      SetMsTimer(MsTimer * timerOut, uint16 initValueIn);
void      UpdateMsTimer(MsTimer * timerInOut);
Boolean MsTimerExpired(MsTimer * timerInOut);
```

广东省网络空间安全协会受控资料

```
void      NodeReset(Boolean firstReset);
void      InitEEPROM(void);
Boolean IOChanges(uint8 pinNumberIn);
uint8    CheckSum4(void *data, uint16 lengthIn);
uint8    CheckSum8(void *data, uint16 lengthIn);
uint8    ComputeConfigCheckSum(void);
int16    GetPrimaryIndex(int16 nvIndexIn);
NVStruct *GetNVStructPtr(int16 nvIndexIn);
Boolean IsTagBound(uint8 tagin);
Boolean IsNVBound(int16 nvIndexIn);
Boolean AppPgmRuns(void);
Boolean NodeConfigured(void);
Boolean NodeUnConfigured(void);


#endif   /* #ifndef _NODE_H */
/*********************node.h*********************/
/*********************************************
  File:          custom.h  (custom.h)

  Version:       1.7

  References:    Protocol Spec.

  Purpose:       Contains constant definitions that can be
                 used to customize the characteristics of the
                 node running the reference implementation.

  Note:          None.

  To Do:         None.

*********************************************/
#ifndef _CUSTOM_H
#define _CUSTOM_H

/*-----------------------------------------------------------
Section: Includes
-----------------------------------------------------------*/
#include <CEA709_1.h>   /* To get constants and types (e.g. Byte) */

/*-----------------------------------------------------------
Section: Constant Definitions
```

广东省网络空间安全协会受控资料

```
-----------------------------------------------------------------*/
/* ReadOnlyData. Reference: Tech Device Data Book. Revision 3. p.9-6 */
#define MODEL_NUM                128
#define MINOR_MODEL_NUM          0
#define READ_WRITE_PROTECT       0
#define RUN_WHEN_UNCONF          0 /* Set to 1 if applicaiton needs to run
                                  even if the node is unconfigured */

/*****************************************************************
   NUM_ADDR_TBL_ENTRIES:
   The address table in reference implementation supports more than
   15 entries. However, network management tools might support only
   upto 15 entries. So, other entries should be managed by the
   application program itself. The maximum supported value is 255.
*****************************************************************/
#define NUM_ADDR_TBL_ENTRIES    15      /* # of entries in addr tbl      */

#define RECEIVE_TRANS_COUNT     50      /* Can be > 16 for Ref. Impl */

#define NV_TABLE_SIZE           100     /* Check management tool for any
                                           restriction on maximum size */

#define NV_ALIAS_TABLE_SIZE     10      /* Check management tool for any
                                           restriction on maximum size */

#define SNVT_SIZE               1050    /* Maximum allowed storage space
                                           for SNVT structures */

/*****************************************************************
Warning!!! The following constants are encoded. So, don't use the
actual values
*****************************************************************/
#define APP_OUT_BUF_SIZE        15
#define APP_IN_BUF_SIZE         15      /* 11 ==> buf size is 66          */
#define NW_OUT_BUF_SIZE         15
#define NW_IN_BUF_SIZE          15

#define APP_OUT_Q_CNT           3
#define APP_OUT_PRI_Q_CNT       8       /* 3 ==> 2   8 ==> 15             */
#define APP_IN_Q_CNT            3

#define NW_OUT_Q_CNT            5
#define NW_OUT_PRI_Q_CNT        8
```

广东省网络空间安全协会受控资料

```
#define NW_IN_Q_CNT            3

#define NGTIMER_SPCL_VAL       8.192

#define NON_GROUP_TIMER 8

/* Reference: Tech Device Data Book. Revision 3. p.9—27. */
#define NM_AUTH 0

/* Node Self Documentation String */
#define NODE_DOC  "Ref. Impl. Pgm."

/* If the node is a member of group A(say), then typical 709 applications
   set group size to 1 more than actual group size if node is not
   a member of group A. This is done so that the number of
   acknowledgements to be expected is always (groupsize — 1).
   In reference implementation, there is an option
   to set this to the true group size and let the transport
   and session layers take care of this. Comment the following
   line to do this. For backward compatibility, uncomment it.
*/
#define GROUP_SIZE_COMPATIBILITY

/* Maximum number of array network variables allowed in
   the application program. This constant is used to allocate
   space that keeps track of all arrays and their dimension */
#define MAX_NV_ARRAYS 10

/* Maximum number of network output variables that can
   be scheduled to be sent out at any point in time */
#define MAX_NV_OUT      50

/* To implement synchronous variables, the values of the
   variables are to be stored along with index in the queue.
   Define the maximum size (in bytes) of a network variable
   in the application program. This is used for storage allocation. */
#define MAX_NV_LENGTH 50

/* Maximum number of network input variables that can
   be scheduled to be polled at any point in time */
#define MAX_NV_IN       50

/* Maximum number of bytes in data array for msg_in, msg_out, resp_in etc.
```

广东省网络空间安全协会受控资料

```
   This value is indepedent of application buffer sizes mentioned
   earlier. Clearly it does not make sense for this value to be
   larger than application buffer size (out or in). */
#define MAX_DATA_SIZE 100

/* The following constant is used to delay transport and session
   layers on an external or power-up reset for sending messages
   to make sure that messages sent after a reset are not discarded
   as duplicates. The dealy should be small enough to ensure that
   there will be no message in receive transaction records of
   target nodes. Normal default is 2 seconds. The amount is given
   in milliseconds.
   The timer duration would be set based on the maximum expected receive
   timer value in all target nodes. */
#define TS_RESET_DELAY_TIME 2000

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   Protocol Stack Implementation uses an array to allocate storage
   space dynamically. The size of the array used for this allocation
   is determined by this constant. If it is too low, it may be
   impossible to allocate necessary buffers or data structures.
   If it is too high, some memory is unused.
   Set to some high value, run program, stop, and check
   gp->mallocUsedSize to determine the current usage.
   This array space is allocated during Reset of all layers.
   Tracing through the Reset code of all layers will indicate
   the approximate size of this array necessary.
   If AllocateStorage function in node.c is rewritten to use malloc, then
   this constant will be of no use.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#define MALLOC_SIZE      52000

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Section: Type Definitions
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
typedef Byte DomainId[DOMAIN_ID_LEN];
typedef Byte AuthKey[AUTH_KEY_LEN];

/* Other values that can be initialized are in the following structure.
   The actual values are given in custom.c file. */
typedef struct
{
   /* ReadOnlyData Members */
```

广东省网络空间安全协会受控资料

```
    Byte uniqueNodeId[UNIQUE_NODE_ID_LEN];
    Byte twoDomains;
    char progId[ID_STR_LEN];

    /* ConfigData Members */
    char location[LOCATION_LEN];

    /* Domain Table Members */
    DomainId domainId[MAX_DOMAINS];
    Byte len[MAX_DOMAINS];
    Byte subnet[MAX_DOMAINS]; /* One for each domain */
    Byte node[MAX_DOMAINS];
    AuthKey key[MAX_DOMAINS];   /* 6 byte authentication key */

    /* Address Table Info. Enter all 5 byte values */
    /* Note: LonBuilder will overwrite the first 15 entries during
       Load/Start. This was used for only for testing purpose */
    /* Alias tables are not handled by some management tools
       and hence this initialization might be useful. */
    Byte addrTbl[NUM_ADDR_TBL_ENTRIES][5];
    Byte aliasTbl[NV_ALIAS_TABLE_SIZE][6];

} CustomData;

extern CustomData   customDataGbl[NUM_STACKS];
extern CustomData *cp;


#endif   /* #ifndef _CUSTOM_H */
/*********************End of custom.h*********************/

/*********************************************************
          File:          network.h

       Version:          1.7

     Reference:          Section 6. Network Layer

       Purpose:          This is the interface file for network layer.

          Note:          Ref. Impl. does not support Router Nodes.

         To Do:          None.
```

广东省网络空间安全协会受控资料

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#ifndef _NETWRK_H
#define _NETWRK_H

/*-----------------------------------------------------------------------------
Section: Includes
-----------------------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------------------
Section: Constant Definitions
-----------------------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------------------
Section: Type Definitions
-----------------------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------------------
Section: Globals
-----------------------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------------------
Section: Function Prototypes
-----------------------------------------------------------------------------*/
void    NWReset(void);
void    NWSend(void);
void    NWReceive(void);

#endif
/*-------------------End of network.h--------------------*/
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        File:           tcs.h

     Version:           1.7

     Purpose:           Interface for Transaction Control Sublayer.
                        Outgoing Sequencing
                        Incoming Sequencing and Duplicate Detection.

        Note:           For assigning TIDs, a table is used. We
```

广东省网络空间安全协会受控资料

```
                    remember the last TID for each unique dest
                    addr. When a new TID is requested for a
                    destination, this table is searched for that
                    destination. If found, we make sure that we
                    don't assign the same TID used for that
                    destination. If the destination is not
                    found, we make a new entry in the table.

                    We have an entry in the table for each
                    subnet/node, group, broadcast, subnet
                    broadcast, unique node id. When a table entry
                    is assigned, we remember the time stamp too.
                    If the table does not have space for a new
                    dest addr, we get rid of one that has
                    remained more than 24 sec. If no such entry,
                    then we fail to allocate the new trans ID.
                    The table size is configurable.

        To Do:      None

*****************************************************************/
#ifndef _TCS_H
#define _TCS_H

/*-------------------------------------------------------------
Section: Includes
-------------------------------------------------------------*/
#include <CEA709_1.h>
#include <node.h>

/*-------------------------------------------------------------
Section: Constant Definitions
-------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------
Section: Globals
-------------------------------------------------------------*/
```

广东省网络空间安全协会受控资料

```
/* None */

/*-----------------------------------------------------------------
Section: Function Prototypes
-----------------------------------------------------------------*/
void TCSReset(void);

void    TransDone(Boolean   priorityIn);

/* Return Values:        SUCCESS or FAILURE    */
Status NewTrans(Boolean    priorityIn, DestinationAddress addrIn,
                TransNum  * transNumOut);

/* Return Values: TRAN_CURRENT or TRAN_NOT_CURRENT */
TransStatus ValidateTrans(Boolean   priorityIn, TransNum transNumIn);

#endif
/*-----------------------End of tcs.h-----------------------------*/
/*****************************************************************
          File:          tsa.h (Transport Session Authentication)

       Version:          1.7

     Reference:          Sections 8, 9 Protocol Specification

       Purpose:          Interface file for tsa.c

          Note:          None

         To Do:          None
*****************************************************************/
#ifndef _TSA_H
#define _TSA_H

/*-----------------------------------------------------------------
Section: Includes
-----------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------
Section: Constant Definitions
-----------------------------------------------------------------*/
/* None */
```

广东省网络空间安全协会受控资料

```
/*-----------------------------------------------------------------
Section: Type Definitions
-----------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------
Section: Globals
-----------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------
Section: Function Prototypes
-----------------------------------------------------------------*/
void TSAReset(void);

void TPSend(void);
void TPReceive(void);

void SNSend(void);
void SNReceive(void);

void AuthSend(void);
void AuthReceive(void);

#endif
/*-------------------End of tsa.h-------------------*/
/*****************************************************
        File:        netmgmt.h

     Version:        1.7

   Reference:        Section 10, Protocol Specification

     Purpose:        App Layer / Network Management

        Note:        For more information, see netmgmt.c.

       To Do:        None

*****************************************************/
#ifndef _NETMGMT_H
#define _NETMGMT_H
```

广东省网络空间安全协会受控资料

```
/*-----------------------------------------------------------------
Section: Includes
-----------------------------------------------------------------*/
#include <CEA709_1.h>
#include <queue.h>
#include <tsa.h>
#include <network.h>
#include <api.h>

/*-----------------------------------------------------------------
Section: Constant Definitions
-----------------------------------------------------------------*/
typedef enum
{
   ND_MESSAGE,
   NM_MESSAGE
} NtwkMgmtMsgType;

/* Message codes */

#define NM_QUERY_ID              0x1
#define NM_RESPOND_TO_QUERY      0x2
#define NM_UPDATE_DOMAIN         0x3
#define NM_LEAVE_DOMAIN          0x4
#define NM_UPDATE_KEY            0x5
#define NM_UPDATE_ADDR           0x6
#define NM_QUERY_ADDR            0x7
#define NM_QUERY_NV_CNFG         0x8
#define NM_UPDATE_GROUP_ADDR     0x9
#define NM_QUERY_DOMAIN          0xA
#define NM_UPDATE_NV_CNFG        0xB
#define NM_SET_NODE_MODE         0xC
#define NM_READ_MEMORY           0xD
#define NM_WRITE_MEMORY          0xE
#define NM_CHECKSUM_RECALC       0xF
#define NM_INSTALL               0x10
#define NM_WINK                  0x10 /* Alias for NM_INSTALL */
#define NM_MEMORY_REFRESH        0x11
#define NM_QUERY_SNVT            0x12
#define NM_NV_FETCH              0x13
#define NM_MANUAL_SERVICE_REQUEST 0x1F
```

山东省网络空间安全协会受控资料

```
/* Define offsets and masks for constructing request and response codes */

#define NM_opcode_base        0x60
#define NM_opcode_mask        0x1F
#define NM_resp_mask          0xE0
#define NM_resp_success       0x20
#define NM_resp_failure       0x00

#define ND_opcode_base        0x50
#define ND_opcode_mask        0x0F
#define ND_resp_mask          0xF0
#define ND_resp_success       0x30
#define ND_resp_failure       0x10

/* Network Diagnostic Message Codes */
#define ND_QUERY_STATUS       0x01
#define ND_PROXY_COMMAND      0x02
#define ND_CLEAR_STATUS       0x03
#define ND_QUERY_XCVR         0x04


/*-----------------------------------------------------------------
Section: Type Definitions
-----------------------------------------------------------------*/
#pragma maxalign(1)

typedef enum {
      ABSOLUTE_MEM_ADDR   = 0,
      READ_ONLY_RELATIVE  = 1,
      CONFIG_RELATIVE     = 2,
      STAT_RELATIVE       = 3,
} ModeType;

typedef enum
{
   UNCONFIGURED    = 0,
   SELECTED        = 1,
   SELECTED_UNCFG  = 2,   /* selected * AND * unconfigured */
} Selector;

typedef enum
{
```

广东省网络空间安全协会受控资料

```
    NO_ACTION              = 0,
    BOTH_CS_RECALC          = 1,
    CNFG_CS_RECALC          = 4,
    ACTION_RESET            = 8,
} Form;

typedef struct {
    Selector selector;
    ModeType mode;
    uint16    offset;
    Byte      count;
    Byte      data[MAX_DATA_SIZE];
} NMQueryIdRequest;

typedef struct {
    ModeType mode;
    uint16    offset;
    Byte      count;
    Form form;
    Byte      data[MAX_DATA_SIZE];
} NMWriteMemoryRequest;

#pragma maxalign()

/*-----------------------------------------------------------------
Section: Globals
-----------------------------------------------------------------*/
/* None */

/*-----------------------------------------------------------------
Section: Function Prototypes
-----------------------------------------------------------------*/
void HandleNM(APPReceiveParam *appReceiveParamPtr, APDU *apduPtr);
void HandleND(APPReceiveParam *appReceiveParamPtr, APDU *apduPtr);
void HandleProxyResponse(APPReceiveParam *appReceiveParamPtr, APDU *apduPtr);

#endif
/*-----------------End of netmgmt.h-----------------*/
/*********************************************************

        File:        app.h

      Version:       1.7
```

广东省网络空间安全协会受控资料

```
        Purpose:        App Layer
                        Data Structures for App Layer.
                        (See app.c for more information.)

           Note:        None

         To Do:         None

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#ifndef _APP_H
#define _APP_H

/*-----------------------------------------------------------------------
Section: Includes
-----------------------------------------------------------------------*/
#include <CEA709_1.h>
#include <queue.h>
#include <tsa.h>
#include <network.h>
#include <api.h>

/*-----------------------------------------------------------------------
Section: Constant Definitions
-----------------------------------------------------------------------*/

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Define tags for use by the application layer for the messages it generates.
There are several types of messages the application layer generates. Proxy,
ManualServiceRequest, network variable updates, and network variable polls. The tag
for these messages will be set up in such a way that we can tell what type of
message got completed when the application layer receives completion indication
from transport/session layer. Since tags are also used by the application
program and they are all >= 0, we will use negative tags for the application
layer. For tags for the application layer are used as follows:

0xFFFF   ==> tag for proxy command.
0xFFFE   ==> tag for manual service request.
For all NV tags,
bit15 is 1 (negative tag)
bit14 is 1 => nv update 0 => nv poll
bit13 is 1 => last tag.
bit12-bit0 is the actual primary index of network variable.
```

广东省网络空间安全协会受控资料

```
The tag for which bit13 is set is a special tag value that is recognized
by transport layer. In this case, the transport layer sends an indication
right away with that tag. No message is sent by the transport layer.

Network variable updates and polls are scheduled sequentially. When the
completion event for the last tag is received, completion event is
generated.
*****************************************************************************/
#define PROXY_TAG_VALUE                          ((MsgTag) 0xFFFF)
#define MANUAL_SERVICE_REQ_TAG_VALUE             ((MsgTag) 0xFFFE)
#define NV_UPDATE_LAST_TAG_VALUE                 ((MsgTag) 0xE000)
#define NV_POLL_LAST_TAG_VALUE                   ((MsgTag) 0xA000)

#define PROXY_TAG(tag)                           (tag == PROXY_TAG_VALUE)
#define MANUAL_SERVICE_REQUEST_TAG(tag)          (tag == MANUAL_SERVICE_REQ_TAG_VALUE)

#define NV_UPDATE_TAG(tag)                       ((tag & 0xC000) == 0xC000)
#define NV_POLL_TAG(tag)                         ((tag & 0xC000) == 0x8000)
#define NV_INDEX_OF_TAG(tag)                     (tag & 0x1FFF)
#define NV_LAST_TAG(tag)                         ((tag & 0xA000) == 0xA000)

#define GET_NV_UPDATE_TAG(index)                 (0xC000 | index)
#define GET_NV_POLL_TAG(index)                   (0x8000 | index)

/* Explicit application message codes */
#define APPL_MSG_OFFLINE        0x3F
#define FOREIGN_FRAME_OFFLINE   0x4F
/*------------------------------------------------------------------------
Section: Type Definitions
------------------------------------------------------------------------*/
/* None */

/*------------------------------------------------------------------------
Section: Globals
------------------------------------------------------------------------*/
/* None */

/*------------------------------------------------------------------------
Section: Function Prototypes
------------------------------------------------------------------------*/
void APPInit(void);
void APPReset(void);
```

```
void APPSend(void);
void APPReceive(void);

#endif
/*-------------------End of app.h-------------------*/
```

A.13 应用协议状态变量和地址识别结构

```
#define DOMAIN_ID_LEN      6
#define AUTH_KEY_LEN      6
#define Unique_Node_ID_LEN      6

typedef enum addr_type { UNBOUND, SUBNET_NODE, UNIQUE_NODE_ID, BROADCAST }
addr_type;

typedef struct domain_struct{
    unsigned    id[DOMAIN_ID_LEN];
    unsigned    subnet;
    unsigned            :1;
    unsigned    node    :7;
    unsigned    len;
    unsigned    key[AUTH_KEY_LEN];
} domain_struct;

typedef union address_struct{
    group_struct    gp;
    snode_struct    sn;
    bcast_struct    bc;
    turnaround_struct    ta;
} address_struct;

typedef struct group_struct {
    unsigned    type        : 1;    /* 1 => group                    */
    unsigned    size        : 7;    /* group size (0 => huge group)  */
    unsigned    domain      : 1;    /* domain index                  */
    unsigned    member      : 7;    /* member num                    */
    unsigned    rpt_timer   : 4;    /* unackd_rpt timer              */
    unsigned    retry       : 4;    /* retry count                   */
    unsigned    rcv_timer   : 4;    /* receive timer index           */
    unsigned    tx_timer    : 4;    /* transmit timer index          */
    unsigned    group;              /* group ID                      */
} group_struct;

typedef struct snode_struct {
```

广东省网络空间安全协会受控资料

```
    addr_type   type;                   /* SUBNET_NODE                  */
    unsigned    domain        : 1;      /* domain index                 */
    unsigned    node          : 7;      /* node number                  */
    unsigned    rpt_timer     : 4;      /* unackd_rpt timer             */
    unsigned    retry         : 4;      /* retry count                  */
    unsigned                  : 4;
    unsigned    tx_timer      : 4;      /* transmit timer index         */
    unsigned    subnet;                 /* subnet ID                    */
} snode_struct;

typedef struct bcast_struct {
    addr_type   type;                   /* BROADCAST                    */
    unsigned    domain        : 1;      /* domain index                 */
    unsigned                  : 1;
    unsigned    backlog       : 6;      /* backlog override value       */
    unsigned    rpt_timer     : 4;      /* unackd_rpt timer             */
    unsigned    retry         : 4;      /* retry count                  */
    unsigned                  : 4;
    unsigned    tx_timer      : 4;      /* transmit timer index         */
    unsigned    subnet;                 /* subnet ID (0 = domain bcast  */
} bcast_struct;


typedef struct unid_struct {
    addr_type   type;
    unsigned    domain        : 1;
    unsigned                  : 7;
    unsigned    rpt_timer     : 4;
    unsigned    retry         : 4;
    unsigned                  : 4;
    unsigned    tx_timer      : 4;
    unsigned    subnet;
    unsigned    nid [UNIQUE_NODE_ID_LEN];
} unid_struct;


typedef     union msg_out_addr {
    addr_type     no_address;           /* UNBOUND 0 if no address      */
    group_struct  group;                /* Defined above                */
    snode_struct  snode;                /* Defined above                */
    unid_struct   unid;                 /* Defined above                */
    bcast_struct  bcast;                /* Defined above                */
```

广东省网络空间安全协会受控资料

```
} msg_out_addr;

/* Typedef for 'msg_in_addr', which is the type of the field 'msg_in.addr'      */

typedef struct msg_in_addr {
    unsigned    domain          : 1;
    unsigned    flexible_domain : 1;
    unsigned    format          : 6;  /* NOT the 'addr_type' enum.          */
                                      /* INSTEAD: 0 => Bcast, 1 =>          */
                                      /* Group, 2 => Subnet/Node,           */
                                      /* 3 => Unique_Node_ID                */
    struct    {
        unsigned    subnet;
        unsigned            : 1;
        unsigned    node    : 7;
    } src_addr;
    union {
        unsigned bcast_subnet;
        unsigned group;
        struct    {
            unsigned    subnet;
            unsigned            : 1;
            unsigned    node        : 7;
        } snode;
        struct {
            unsigned    subnet;
            unsigned    nid [UNIQUE_NODE_ID_LEN];
        } unid;
    } dest_addr;
} msg_in_addr;

/* Typedef for 'resp_in_addr', the type of the field 'resp_in.addr'      */


typedef struct resp_in_addr {
    unsigned    domain              : 1;
    unsigned    flexible_domain     : 1;
    struct    {
        unsigned    subnet;
        unsigned    is_snode        : 1;    /* 0=>group resp,      */
                                            /* 1=>snode resp       */
        unsigned    node            : 7;
    } src_addr;
```

山东省网络空间安全协会受控资料

```
    union {
        struct {
            unsigned    subnet;
            unsigned            : 1;
            unsigned    node    : 7;
        } snode;
        struct {
            unsigned    subnet;
            unsigned            : 1;
            unsigned    node    : 7;
            unsigned    group;
            unsigned            : 2;
            unsigned    member  : 6;
        } group;
    } dest_addr;
} resp_in_addr;

struct {
    byte            code;               /* message code                          */
    byte            len;                /* length of message data                */
    byte            data[];             /* message data                          */
    boolean         authenticated;      /* TRUE if message was authenticated     */
    service_type    service;            /* Service type used to send the msg     */
    boolean         duplicate;          /* TRUE if message is a duplicate        */
    unsigned        rcvtx;              /* index to the transaction record       */
    msg_in_addr     addr;               /* destination address (see above)       */
    } msg_in;

struct    {
    boolean         priority_on;        /* TRUE if a priority message            */
    msg_tag         tag;                /* to correlate completion codes         */
    byte            code;               /* message code                          */
    byte            data[];             /* message data                          */
    boolean         authenticated;      /* TRUE if to be authenticated           */
    service_type    service;            /* service type used to send the msg     */
    msg_out_addr    dest_addr;          /* destination address (see above)       */
    } msg_out;

struct    {
    byte            code;       /* message code                                  */
    byte            len;        /* message length                                */
    byte            data[];     /* message data                                  */
    resp_in_addr    addr;       /* destination address (see above)               */
```

广东省网络空间安全协会受控资料

```
    } resp_in;                      /* struct for receiving responses          */

struct    {
    byte            code;           /* message code                            */
    byte            data[];         /* message data                            */
    } resp_out;                     /* structure for sending responses         */
```

A.14 查询标识数据结构

```
struct query_id_request {
    byte command;                   /* value = 1                               */
    byte selector;                  /* 0 = unconfigured nodes                  */
                                    /* 1 = respond to query set                */
                                    /* 2 = respond to query set and unconfigured */
    byte address_mode;              /* See "read memory" command               */
    byte address_hi;
    byte address_lo;
    byte count;
    byte data;                      /* "count" bytes of data */
};

struct query_id_response {
    byte command;                   /* value = 1 */
    byte Unique_Node_ID[6];
    byte program_id_string[8];
};
```

广东省网络空间安全协会受控资料

A.15 查询应答数据结构

```
struct respond_to_query_cmd {
    byte command;                   /* value = 2                               */
    byte mode;                      /* 1 => ON; 0 => OFF                       */
};

struct respond_to_query_resp {
    byte command;           /* value = 2 */
};
```

A.16 更新域数据结构

```
struct update_domain_request {
    byte command;                   /* value = 3                               */
    byte domain_index;              /* 0 or 1                                  */
```

```
    byte domain_id[6];
    byte subnet;
    byte node;                  /* msb must be set                    */
    byte did_length;            /* 0, 1, 3, or 6                      */
    byte encrypt_key[6];
};

struct update_domain_response {
    byte command;          /* value = 3 */
};
```

A.17 离开域数据结构

```
struct leave_domain_request {
    byte command;          /* value = 4          */
    byte domain_index;     /* 0 or 1             */
};

struct leave_domain_response {
    byte command;          /* value = 4 */
};
```

广东省网络空间安全协会受控资料

A.18 更新密钥数据结构

```
struct update_key_request {
    byte command;          /* value = 5          */
    byte domain_index;     /* 0 or 1             */
    byte encrypt_key[6];
};

struct update_key_response {
    byte command;          /* value = 5 */
};
```

A.19 更新地址数据结构

```
struct group_s {
    byte field1;        /* b7: 1                                    */
                        /* b0—b6: group size, 0 for huge group      */
    byte field2;        /* b7: domain ref                           */
                        /* b0—b6: member number, 0 for huge group   */
    byte field3;        /* b4—b7: unackd_rpt timer                  */
                        /* b0—b3: retry count                       */
```

```
    byte field4;        /* b4—b7: receive timer index                    */
                        /* b0—b3: transmit timer index                   */
    byte group;         /* group id.                                     */
};

struct snode_s {
    byte type;          /* 1                                             */
    byte field2;        /* b7: domain ref set by target b0—b6: node number */
    byte field3;        /* b4—b7: unackd_rpt timer                       */
                        /* b0—b3: retry count                            */
byte field4;            /* b0—b3: transmit timer index                   */
byte subnet;            /* subnet                                        */
};

struct bdcst_s {
    byte type;          /* 3                                             */
    byte field2;        /* b7: domain ref set by target                  */
                        /* b0—b5: backlog; 0 == unknown                  */
    byte field3;        /* b4—b7: unackd_rpt timer                       */
                        /* b0—b3: retry count                            */
    byte field4;        /* b0—b3: transmit timer index                   */
    byte subnet;        /* subnet                                        */
};

struct ta_s {           /* Turnaround entry                              */
    byte type;          /* 0                                             */
    byte ta;            /* 1                                             */
    byte field3;        /* b4—b7: unackd_rpt timer                       */
                        /* b0—b3: retry count                            */
    byte field4;        /* b0—b3: transmit timer index                   */
};

struct empty_s {        /* Empty entry                                   */
    byte type           /* 0                                             */
    byte empty;         /* 0                                             */
};

struct update_addr_request {
    byte command;       /* value = 6                                     */
    byte index;         /* 0—14                                          */
    union {
        struct group_s;
        struct snode_s;
```

广东省网络空间安全协会受控资料

```
        struct bdcst_s;
        struct ta_s;
        struct empty_s;
    } address;
};

struct update_addr_response {
    byte command;          /* value = 6 */
};
```

A.20 查询地址数据结构

```
struct query_addr_request {
    byte command;          /* value = 7      */
    byte index;            /* 0—14           */
};

struct query_addr_response {
    byte command;          /* value = 7 */
    union {
        struct group_s;
        struct snode_s;
        struct bdcst_s;
        struct ta_s;
        struct empty_s;
    } address;
};
```

广东省网络空间安全协会受控资料

A.21 查询网络变量配置数据结构

```
struct query_nv_cnfg_request {
    byte command;          /* value = 8                                    */
    byte nv_index;         /* index of nv when < 255                       */
    [uint16 nv_index16;]   /* 16-bit index iff nv_index == 255             */
    byte pri_index;        /* index of primary NV when nv_index refers to an */
                           /* alias                                        */
    [uint16 pri_index16;]  /* 16 bit index of primary NV when pri_index == 255 */
};

struct query_nv_cnfg_response {
    byte command;          /* value = 8          */
    byte field1;           /* b7: priority       */
                           /* b6: direction      */
```

```
                        /* b0—b5: net var selector-msb          */
    byte idlo;          /* net var selector-lsb                 */
    byte field2;        /* b7: turnaround                       */
                        /* b5—b6: service                       */
                        /* b4: authenticated                    */
                        /* b0—b3: address table index           */
};
```

A.22 更新组地址数据结构

```
struct update_group_addr_request {
    byte command;       /* value = 9 */
    struct group_s;
    } address;
};

struct update_group_addr_response {
    byte command;       /* value = 9 */
};
```

A.23 查询域数据结构

广东省网络空间安全协会受控资料

```
struct query_domain_request {
    byte command;       /* value = 10                  */
    byte index;         /* Domain index, 0 or 1        */
};

struct query_domain_response {
    byte command;       /* value = 10 */
    byte domain_id[6];
    byte subnet;
    byte node;
    byte did_length;    /* 0,1,3, or 6 */
    byte encrypt_key[6];
};
```

A.24 更新网络变量配置数据结构

```
struct update_nv_cnfg_request {
    byte command;       /* value = 11                          */
    byte nv_index;      /* 0—255;                              */
    [uint16 nv_index16;] /* 16-bit index iff nv_index == 255   */
```

```
    byte field1;            /* b7: priority                          */
                            /* b6: direction                         */
                            /* b0—b5: net var selector-msb           */
    byte idlo;              /* net var selector-lsb                  */
    byte field2;            /* b7: turnaround                        */
                            /* b5—b6: service                        */
                            /* b4: secure                            */
                            /* b0—b3: address table index            */
    byte pri_index;         /* index of primary nv iff nv_index is to */
    [uint16 nv_index16;]    /* an alias. 16 bit primary index if     */
                            /* pri_nv == 255.                        */
};

struct update_nv_cnfg_response {
    byte command;           /* value = 11 */
};
```

A.25 设置节点模式数据结构

```
struct set_node_mode_request {
    byte command;           /* value = 12                            */
    byte state;             /* 0: offline                            */
                            /* 1: online                             */
                            /* 2: reset                              */
                            /* 3: change state                       */
    byte state_data;        /* Iff state=3; see "Node States" for values */

struct set_node_mode_response {
    byte command;           /* value = 12   Note, no response is provided */
                            /* for offline, online or reset commands */
};
```

广东省网络空间安全协会受控资料

A.26 读存储器数据结构

```
struct read_memory_request {
    byte command;           /* value = 13          */
    byte address_mode;      /* See below           */
    byte address_hi;
    byte address_lo;
    byte count;
};

struct read_memory_response {
    byte command;           /* value = 13 */
    byte data[count];
```

```
};

/* where "address_mode" determines the physical address as follows:                    */
/* 0: address = address_hi * 256 + address_lo                                           */
/* 1: address = non-volatile memory read-only address+address_hi * 256+address_lo      */
/* 2: address = non-volatile memory configuration address + address_hi * 256 +         */
/* address_lo                                                                           */
```

A.27 写存储器数据结构

```
struct write_memory_request {
    byte command;           /* value = 14                                    */
    byte address_mode;      /* See comment in read_memory command above      */
    byte address_hi;
    byte address_lo;
    byte count;
    byte form;              /* 0: no reset, no checksum                      */
                            /* 1: recalculate both checksums                 */
                            /* 4: recalculate just configuration checksum    */
                            /* 8: reset                                      */
                            /* 9: reset, recalc both checksums               */
                            /* 12: reset, recalc configuration checksum      */
    byte data[count];
};

struct write_memory_response { /* used only for non-reset writes             */
    byte command;           /* value = 14                                    */
};
```

广东省网络空间安全协会受控资料

A.28 重新计算校验和数据结构

```
struct checksum_request {
    byte command;           /* value = 15                                    */
    byte which;             /* 1: do application and configuration           */
                            /* 4: do configuration only                      */
};

struct checksum_response {
    byte command;           /* value = 15 */
};
```

A.29 安装命令数据结构

```
typedef struct {
    unsigned        reserved                :3;     /* Must be set to 0 */
    unsigned        nv_direction            :1;
    unsigned        nv_auth                 :1;
    unsigned        nv_priority             :1;
    unsigned        nv_service              :2;
} nv_dflts;

typedef struct {
    unsigned        reserved                :1;         /* Must be set to 0 */
    unsigned        nv_sync                 :1;
    unsigned        nv_polled               :1;
    unsigned        nv_offline              :1;
    unsigned        nv_service_type_config  :1;
    unsigned        nv_priority_config      :1;
    unsigned        nv_auth_config          :1;
    unsigned        nv_config_class         :1;
    byte            snvt_type_index;                    /* use enum SNVT_t */
} nv_attr;

typedef struct {
    unsigned    mre             :1;     /* Max Rate Est data is available          */
    unsigned    re              :1;     /* Rate Estimate data is available         */
    unsigned    nm              :1;     /* NV Name is available                    */
    unsigned    sd              :1;     /* NV SD text is available                 */
    unsigned    nm_supplied     :1;     /* For QUERY_NV_INFO/NV_INFO_DESC,         */
                                        /* name is appended to message             */
    unsigned    reserved1       :3;     /* Must be set to 0                        */
} nv_exten;

typedef struct {
    unsigned    length          :5;     /* # of bytes in NV value                  */
    unsigned    origin          :3;     /* How NV was defined                      */
    nv_dflts    dflts;                  /* Default NV Configuration settings       */
    nv_attr     attr;                   /* Basic NV attributes                     */
    nv_exten    ext;                    /* Available NV Extension attributes       */
    nv_array    array;                  /* NV array attributes                     */
} nv_desc;

typedef enum{
    APP_WINK            = 0,    /* basic command                                   */
    APP_INSTALL         = 1,    /* get information from multiple stacks            */
```

广东省网络空间安全协会受控资料

```
    APP_NV_DEFINE          = 2,    /* define a new network variable on a node      */
    APP_NV_REMOVE          = 3,    /* remove a network variable from the node      */
    APP_QUERY_NV_INFO      = 4,    /* get network variable information             */
    APP_QUERY_NODE_INFO    = 5,    /* get node self documentation information       */
    APP_UPDATE_NV_INFO     = 6     /* modify nv info. Change its type, etc.        */
} AppCommand;
struct install_msg {
    byte command;                  /* value = 16                                   */
    byte subcommand;               /* one of AppCommand above                      */
    byte subdata[ * ];             /* length dependent upon the command/subcommand pair */
};
struct install_response {
    byte command;                  /* value = 16 */
    union {
        struct manual_service_request_message;     /* if subcommand == 0 */
    } data;
};

/* Request to define new NV(s) on the node. The app should respond successfully even
   if the NV is already defined.
*/
typedef struct
{
    byte        app_command;       /* APP_NV_DEFINE (AppCommand enum)              */
    uint16      nv_index;          /* New NV index                                 */
    uint16      array_len;         /* # of elements in NV array,                   */
                                   /* 0 if new NV is not an array                  */
    byte        nv_length;         /* # of bytes in NV value (1—31)                */
    nv_dflts    dflts;             /* Default NV Configuration settings            */
    nv_attr     attr;              /* NV self—documentation attributes             */
} NM_app_define_nv;

typedef struct
{
    byte                code;          /* NM_app_cmd */
    NM_app_define_nv    data;
} CMD_app_define_nv;

typedef struct
{
    byte    code;                  /* NM_app_cmd_fail/NM_app_cmd_succ */
} CMD_app_response;

/* Request to remove a set of NVs. The host should respond successfully
```

广东省网络空间安全协会受控资料

```
    even if 1 or more of the specified NVs does not exist.
 */
typedef struct
{
    byte            app_command;        /* APP_NV_REMOVE (AppCommand enum) */
    uint16          nv_index;           /* Index of first NV to remove     */
    uint16          nv_count;           /* # of NVs to remove              */
} NM_app_remove_nv;

typedef struct
{
    byte                code;               /* NM_app_cmd */
    NM_app_remove_nv    data;
} CMD_app_remove_nv;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    Query Network Variable Self-documentation data
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#define NV_INFO_DESC          0
#define NV_INFO_RATE_EST      1
#define NV_INFO_NAME          2
#define NV_INFO_SD_TEXT       3
#define NV_INFO_SNVT_INDEX    4

typedef struct
{
    byte        app_command;        /* APP_QUERY_NV_INFO (AppCommand enum)   */
    byte        nv_info;            /* Requested NV Info (NV_INFO...)        */
    uint16          nv_index;
    union   {
        struct {
                uint16      offset;
                byte        length;
        } sd;                       /* For requesting NV_INFO_SD_TEXT */
    } addl;
} NM_app_query_nv_info;

typedef struct
{
    byte                    code;               /* NM_app_cmd */
    NM_app_query_nv_info    data;
} CMD_app_query_nv_info;
```

广东省网络空间安全协会受控资料

```
typedef union
{
    /* Response for requested info NV_INFO_DESC */
    struct {
            nv_desc          desc;
            byte             nv_name[NV_NAME_LEN]; /* Optional field — Included only if
                                                      desc.ext.nm_supplied is set */
    } desc;

    /* Response for requested info NV_INFO_RATE_EST */
    struct {
        byte    nv_rate_est;        /* Encoded rate estimate. Only valid if 're'
                                       Is set in NV desc */
        byte    nv_max_rate_est;    /* Encoded max rate estimate. Only valid if
                                       'mre' is set in NV desc */
    } rate;     /* Response for requested info NV_INFO_NAME */
    byte    nv_name[NV_NAME_LEN];   /* NV name. Only valid if 'nm' set in NV desc */

    /* Response for requested info NV_INFO_SD_TEXT */
    struct {
        byte    length;
        byte    text[*];            /* depends upon length field          */
    } sd;

    /* Response for requested info NV_INFO_SNVT_INDEX */
    byte    snvt_type_index;
} NM_app_query_nv_info_response;

typedef struct
{
    byte                            code;   /* NM_app_cmd_fail/NM_app_cmd_succ */
    NM_app_query_nv_info_response   data;
} CMD_app_query_nv_info_response;

/*****************************************************************
    Update Network Variable Self-documentation data
*****************************************************************/

typedef struct
{
    byte    app_command;        /* APP_UPDATE_NV_INFO (AppCommand enum) */
    byte    nv_info;            /* NV Info to be updated (NV_INFO...)   */
```

广东省网络空间安全协会受控资料

```
    uint16      nv_index;           /* NV index — If the target NV represents an array,
                                       then the update applies to all elements of the
                                       array.                                          */
    union {
        /* Data for updated info NV_INFO_NAME */
            nv_name[NV_NAME_LEN];   /* NV name (ascii) — 0 terminated
                                           if < 16 characters */
        /* Data for updated info NV_INFO_SD_TEXT */
        struct {
            byte        length;
            uint16        offset;
            byte        text[*];            /* May not be null (0x00) terminated */   } sd;

        /* Data for updated info NV_INFO_RATE_EST */
        struct {
        unsigned    reserved        : 4;    /* Must be set to 0                          */
        unsigned    clear_mre       : 1;    /* Clear 'ext' rec 'mre' field               */
        unsigned    clear_re        : 1;    /* Clear 'ext' rec 're' field                */
        unsigned    update_mre      : 1;    /* Update max rate est with nv_max_rate_est*/
        unsigned    update_re       : 1;    /* Update rate est with nv_rate_est          */
        byte        nv_rate_est;            /* Encoded rate estimate.                    */
          byte      nv_max_rate_est;        /* Encoded max rate estimate.                */
        } rate;

        /* Data for updated info NV_INFO_SNVT_INDEX */
        byte    snvt_type_index;
    } info;
} NM_app_update_nv_info;

typedef struct
{
    byte                    code;       /* NM_app_cmd */
    NM_app_update_nv_info   data;
} CMD_app_update_nv_info;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    Query Node Self-documentation data
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#define NODE_INFO_SD_TEXT       3

typedef struct
{
```

广东省网络空间安全协会受控资料

```
    byte     app_command;        /* APP_QUERY_NODE_INFO (AppCommand enum)    */
    byte     node_info;          /* Requested Node Info (NODE_INFO...)       */
    union    {
        struct {
            uint16      offset;
            byte        length;
        } sd;                    /* For requesting NODE_INFO_SD_TEXT */
    } addl;
} NM_app_query_node_info;

typedef struct
{
    byte                        code;                /* NM_app_cmd */
    NM_app_query_node_info data;
} CMD_app_query_node_info;

typedef union
{
    /* Response for requested info NODE_INFO_SD_TEXT */
    struct {
        byte     length;
        byte     text[*];        /* May not be 0 terminated */
    } sd;
} NM_app_query_node_info_response;

typedef struct
{
    byte                        code;  /* NM_app_cmd_fail/NM_app_cmd_succ */
    NM_app_query_node_info_response data;
} CMD_app_query_node_info_response;
```

广东省网络空间安全协会受控资料

A.30 存储器刷新数据结构

```
struct memory_refresh_request {
    byte     command;        /* value = 17                                  */
    uint16   offset;         /* 16 bit offset from start of non-volatile memory */
    byte     count;          /* Number of bytes to write                    */
    byte     offchip;        /* 1=> refresh off chip memory                 */
};

struct memory_refresh_response {
    byte command;        /* value = 17 */
};
```

A.31 查询自标识数据结构

```
struct query_SI_request {
    byte     command;        /* value = 18                          */
    uint16   offset;         /* 16 bit offset into micro's SNVT table */
    byte     count;          /* number of bytes to return (up to 16) */
};

struct query_SI_response {
    byte command;        /* value = 18                  */
    byte data[count];    /* return data (count bytes)   */
};
```

A.32 取网络变量数据结构

```
struct nv_fetch_request {
    byte command;        /* value = 19                                  */
    byte index;          /* Network Variable selector                   */
    [uint16 index16;]    /* 16-bit index, used only if index == 255     */
                         /* This format supported only by host nodes    */
};

struct nv_fetch_response {
    byte command;        /* value = 19                                  */
    byte index;          /* Network Variable selector                   */
    [uint16 index16;]    /* Iff index == 255; host nodes only           */
    byte data[NVLEN];    /* return data (based on NV length)            */
};
```

广东省网络空间安全协会受控资料

A.33 手动服务请求报文数据结构

```
struct manual_service_request_message {
    byte command;/* value = 31 */
    byte Unique_Node_ID[6];
    byte program_id_string[8];
    };
```

A.34 产品查询数据结构

```
struct product_query_request {
    byte command;        /* Destination: NM, code: 0x7D (escape)   */
    byte data[2];        /* Product query subcode = 0x01           */
```

```
                        /* Product query command = 0x01                    */
};

struct product_query_short_response {
    byte command;       /* success = 0x3D, failure = 0x1D                  */
    byte product_ID;    /* product identifier                              */
}

struct product_query_complete_response {
    byte response;      /* success = 0x3D, failure = 0x1D                  */
    byte product;       /* product identifier                              */
    byte model[2];      /* model number                                    */
    byte version;       /* version number                                  */
    byte config;        /* current configuration or mode of device         */
    byte Xcvr_ID;       /* transceiver type                                */
};
```

A.35 路由器模式数据结构

```
struct router_mode_request {
    byte command;       /* Destination: NM, code: 20                       */
    byte mode;          /* 0: resume, 1: init subnet table, 2: mode flood  */
};

struct router_mode_response {
    byte command;       /* Destination: APPL, code: 20                     */
};
```

A.36 路由表清除组或子网表数据结构

```
struct router_table_clear_request {
    byte command;       /* Destination: NM, code: 21                       */
    byte field1;        /* b7: 1 = group, 0 = subnet                       */
                        /* b6: domain ref                                  */
                        /* b0—3: 8x index                                  */
};

struct router_group_clear_response {
    byte command;       /* Destination: APPL, code: 21                     */
};
```

A.37 路由器组或子网下载数据结构

```
struct groupsubnet_table_download {
```


广东省网络空间安全协会受控资料

```
    byte command;          /* Destination: NM, code: 22                    */
    byte field1;           /* b7: 1 = group, 0 = subnet                    */
                           /* b6: domain ref                               */
                           /* b0—3: 8x index                               */
    byte table[8];         /* l.s. bit is l.s. group/subnet #              */
};

struct groupsubnet_table_download_response {
    byte command;          /* Destination: APPL, code: 22                  */
};
```

A.38 路由器组转发数据结构

```
struct group_forward_request {
    byte command;          /* Destination: NM, code: 23                    */
    byte field1;           /* b0: 0: RAM only, 1: RAM + non-volatile memory */
                           /* b6: domain ref                               */
    byte group;            /* 0—255                                        */
};

struct group_forward_response {
    byte command;          /* Destination: APPL, code: 23                  */
};
```

广东省网络空间安全协会受控资料

A.39 路由器子网转发数据结构

```
struct subnet_forward_request {
    byte command;          /* Destination: NM, code: 24                    */
    byte field1;           /* b0: 0: RAM only, 1: RAM + non-volatile memory */
                           /* b6: domain ref                               */
    byte subnet;           /* 1—255                                        */
};

struct subnet_forward_response {
    byte command;          /* Destination: APPL, code: 24                  */
};
```

A.40 路由器组不转发数据结构

```
struct group_noforward_request {
    byte command;          /* Destination: NM, code: 25                    */
    byte field1;           /* b0: 0: RAM only, 1: RAM + non-volatile memory */
                           /* b6: domain ref                               */
```

```
    byte group;             /* 0—255                                    */
};

struct group_noforward_response {
    byte command;           /* Destination: APPL, code: 25              */
};
```

A.41 路由器子网不转发数据结构

```
struct subnet_noforward_request {
    byte command;           /* Destination: NM, code: 26                */
    byte field1;            /* b0: 0: RAM only, 1: RAM + non-volatile memory */
                            /* b6: domain ref                           */
    byte subnet;            /* 1—255                                    */
};

struct subnet_noforward_response {
    byte command;           /* Destination: APPL, code: 26              */
};
```

A.42 组/子网表报告数据结构

广东省网络空间安全协会受控资料

```
struct groupsubnet_table_report_request {
    byte command;           /* Destination: NM, code: 27        */
    byte field1;            /* b7: 1 = group, 0 = subnet        */
                            /* b6: domain ref                   */
                            /* b0—3: 8x index                   */
};

struct groupsubnet_table_report_response {
    byte command;           /* Destination: APPL, code: 27      */
    byte table[8];          /* l.s. bit is l.s. group/subnet #  */
};
```

A.43 路由器状态数据结构

```
struct router_status_request {
    byte command;           /* Destination: NM, code: 28        */
};

struct router_status_response {
    byte command;           /* Destination: APPL, code: 28      */
    byte router_cnfg;       /* type: 1 = learning, 0 = configured */
```

```
                        /* 2 = bridge, 3 = bridge_repeater          */
    byte mode;          /* 0 = normal, 2 = flood                    */
};
```

A.44 查询状态数据结构

```
struct query_status_request {
    byte command;       /* value = 1                                */
};

struct query_status_response {
    byte command;               /* value = 1                        */
    struct status_response data; /* see below                       */
};

struct status_response {
    uint15      transmission_errors;
    uint16      transaction_timeouts;
    uint16      receive_transaction_full;
    uint16      lost_messages;
    uint16      missed_messages;
    byte      reset_cause;
    byte      node_state;
    byte      version;
    byte      error_log;
    byte      model_number;
};
```

A.45 代理状态数据结构

```
struct proxy_command {
    byte command;           /* value = 2                                    */
    byte sub_command;       /* 0=> query id (unconfigured); 1=> status      */
                            /* 2=> transceiver status request               */
    union {
        struct group_s;
        struct snode_s;
        struct bdcst_s;
        struct nid_s;
    } address;              /* Address structures as defined in the network
                            management modify address command and below  */
};
```

广东省网络空间安全协会受控资料

```
struct nid_s {
    byte type;          /*    2                                  */
    byte field2;        /*    b7: domain ref set by target       */
    byte field3;        /*    b0—b3: retry count                 */
    byte field4;        /*    b0—b3: transmit timer index        */
    byte subnet;        /*    subnet                             */
    byte nid[6];        /*    Unique_Node_ID                     */
};

struct proxy_response {
    byte command;       /* value = 2                             */
    byte resp_data;     /* data and length are functions of the sub_command  */

};
```

A.46 清除状态数据结构

```
struct clear_status_command {
    byte command;       /* value = 3                             */
};

struct clear_status_response {
    byte command;       /* value = 3                             */
};
```

广东省网络空间安全协会受控资料

A.47 查询收发器状态数据结构

```
struct query_xcvr_status_command {
    byte command;       /* value = 4                             */
};

struct query_xcvr_status_response {
    byte command;       /* value = 4                             */
    byte data[7];       /* register values                       */
};
```

附 录 B
（规范性附录）
其他数据结构

本附录包含关于其他协议处理器数据结构的信息。在下面的所有数据结构中，一个字节内的比特位顺序是按照先高位后低位排列的。一个字段内的字节顺序是按照先高字节后低字节排列的。

在协议处理器中，软件可以分为三个主要部分：系统映像、应用映像和网络映像。

a） 系统映像：包含协议和特定实现相关的运行环境。

b） 应用映像：包含由用户应用程序生成的代码，以及其他应用有关的参数。网络管理工具可以查询这些参数。这些参数包括：

- 网络变量固定的和自标识的数据；
- Program ID（程序标识）字符串；
- 可选的自标识和自说明文档数据；
- 地址表项数目；
- 域表项数目；
- 网络缓冲区数目和大小；
- 应用缓冲区数目和大小；
- 接收事务记录数；
- 收发器类型和比特位速率。

广东省网络空间安全协会受控资料

在这里描述的应用映像数据结构是：

- 固定的只读结构，其大小与该节点上的应用无关；
- 网络变量固定表，对该节点定义的每一个网络变量均有相应的表项；
- 可选的自标识和自说明文档信息，描述该节点及其网络变量。

c） 网络映像，包含：

- 节点的地址分配；
- 连接该网络中节点间网络变量和报文标签的绑定信息；
- 协议中可以在安装时设置的参数，以及
- 该应用程序的配置变量。

网络管理工具通常在节点安装时将网络映像经由网络下载到片内的 EEPROM 中。对简单的网络，节点可以更新它自己的网络映像。

本节旨在为那些应用程序编程员提供信息，他们需要了解协议处理器用于地址分配、绑定及网络配置的内部数据结构。

这些数据结构应可通过经由网络的来自网络管理工具的网络管理报文进行访问。

在这里描述的网络映像数据结构是：

- 域表，对该节点从属的每个域都有一个表项；
- 地址表，对由该节点引用的每个网络地址都有一个表项；
- 网络变量配置表，对该节点定义的每个网络变量都有一些表项；
- 信道配置结构，定义该节点的收发器接口。

后面章节定义的结构的地址从分配给固定只读数据结构的存储器地址 0XF000 开始的。这个地址

不需要映射到该节点的实际的物理存储器中。但是，读和写存储器命令应该提供这个地址，好像这就是物理地址。

B.1 固定的只读数据结构

本结构定义节点标识，以及某些应用映像参数。

```
#define UNIQUE_NODE_ID_LEN      6
#define ID_STR_LEN      8
typedef struct {
    unsigned    unique_node_id[UNIQUE_NODE_ID_LEN];     // offset 0x00
    unsigned    model_num;                              // offset 0x06
    unsigned    Checksum                    : 4;        // XOR of each of the
                                                        // 4 bit quantities
                                                        // in unique_node_id
                                                        // and model_num
    unsigned    minor_model_num             : 4;        // offset 0x07
    const     nv_fixed_struct * nv_fixed;               // offset 0x08
    unsigned    read_write_protect          : 1;        // offset 0x0A
    unsigned    run_when_unconfigured       : 1;
    unsigned    nv_count                    : 6;
    const     snvt_struct * snvt;                       // offset 0x0B
    unsigned    id_string[ID_STR_LEN];                  // offset 0x0D
    unsigned    NV_processing_off           : 1;        // offset 0x15
    unsigned    two_domains                 : 1;
    unsigned    explicit_addr               : 1;
    unsigned     reserved                   : 1;
    unsigned    msg_process                 : 1;
    unsigned    state                       : 3;        // node state
                                                        // 2: unconfigured
                                                        // 3: applicationless
                                                        // 4: configured
                                                        // 6: hard offline
    unsigned    address_count               : 4;        // offset 0x16
    unsigned    reserved                    : 4;
    unsigned    reserved                    : 4;
    unsigned    receive_trans_count         : 4;        // offset 0x17
    unsigned    app_buf_out_size            : 4;        // offset 0x18
    unsigned    app_buf_in_size             : 4;
    unsigned    net_buf_out_size            : 4;        // offset 0x19
    unsigned    net_buf_in_size             : 4;
    unsigned    net_buf_out_priority_count  : 4;        // offset 0x1A
    unsigned    app_buf_out_priority_count  : 4;
    unsigned    app_buf_out_count           : 4;        // offset 0x1B
    unsigned    app_buf_in_count            : 4;
```

广东省网络空间安全协会受控资料

```
    unsigned    net_buf_out_count          : 4;        // offset 0x1C
    unsigned    net_buf_in_count           : 4;
    byte    reserved1[6];
    unsigned                               : 6;        // offset 0x22
    unsigned    tx_by_address              : 1;
    unsigned    idempotent_duplicate       : 1;
    } read_only_data_struct;


typedef struct {
    unsigned                               : 2;
    unsigned    alias_count                : 6;
    unsigned    msg_tag_count              : 4;
    unsigned                               : 4;
    byte    reserved2[3];
} read_only_data_struct_2;

const read_only_data_struct read_only_data;
const read_only_data_struct_2 read_only_data_2;
```

应用程序可以使用全局声明 read_only_data 和 read_only_data_2 从这些数据结构中读取数据，但不可写入这些数据结构。该结构可以用带 address_mode＝1 的 *Read Memory* 和 *Write Memory* 网络管理报文经由网络读取和大部分数据的写入(除前 8 个字节外)。它在下载新的应用映像到该节点时被写入。

广东省网络空间安全协会受控资料

B.1.1 只读结构字段描述

```
unsigned    unique_node_id[UNIQUE_NODE_ID_LEN];   // offset 0x00
```

该字段是由协议处理器制造商指定的 6 字节 ID，对所制造的每一个器件是唯一的。它可以用 *Query ID* 网络管理报文经由网络读取，它也是主动提供的手动服务请求网络管理报文的一部分。

```
unsigned    model_num;                             // offset 0x06
unsigned    minor_model_num            : 4;        // offset 0x07
```

这是指定协议处理器型号的 2 个字段。

```
Unsigned    checksum                   : 4;        // offset 0x07
```

该字段包含由 unique_node_id，model_num 和 minor_model_num 组成的 8 字节中其余的 15 个半字节的异或值。

```
const    nv_fixed_struct *nv_fixed;                // offset 0x08
```

该字段是指向网络变量固定数据表(见 B.4)的指针。如果该节点上没有网络变量，或者该节点是一个网络接口，并且正在处理应用的网络变量选择号，则不使用该指针。

```
unsigned    read_write_protect         : 1;        // offset 0x0A
```

这一个比特位规定部分协议处理器的存储器不能用 *Read Memory* 和 *Write Memory* 网络管理报

文经由网络读出或写入。如果该比特位置位，则仅有只读结构(B.1)、SNVT 结构(B.5)、配置结构(B.6)和应用数据区可读，且仅有配置结构可写。被写保护的数据包括 read_write_protect 比特位本身，因此一旦设置，该比特位就不能经由网络重置。

Unsigned　　run_when_unconfigured　　: 1;

这一个比特位规定当节点处于未配置状态时应用程序也将运行。

unsigned　　nv_count　　: 6;

该字段规定在这个节点运行的应用程序中已声明的网络变量数目(0～62)。网络变量数组的每个元素是分别计算的。如果该节点是一个网络接口，并且正在处理应用的网络变量选择号，则该字段为 0。

const　　si_struct * si;　　// offset 0x0B

该字段是指向给出网络变量自标识信息的数据结构的指针(见 B.5)。自标识信息可以直接用读存储器命令读取。如果没有自标识信息，则为空(0)指针。一个指针全为 1 意味着该自标识信息位于协议处理器可寻址空间以外(例如，在协处理器中)。在这种情况下可用查询自标识命令(Query SI)(见 13.7.18)读取自标识信息。

unsigned　　id_string[ID_STR_LEN];　　// offset 0x0D

该字段包含 8 字节程序标识信息，由下列指令之一规定：

#pragma set_id_string "*ssssssss*"

或

#pragma set_std_prog_id *fm*:*mm*:*mm*:*cc*:*cc*:*ss*:*ss*:*nn*

第二个指令格式保留用于符合相关机构可互操作性指南的节点。这一格式的比特位分配如下：

- 首 4 比特位为格式码(0x8 — 0xF)，格式码 8 保留用于通过了相关机构一致性测试的设备，格式码 9 则应用于带有标准程序 ID 但尚未通过一致性检查的设备。格式 0xA — 0xF 保留作将来使用。
- 其后 20 比特位为制造商代码。由相关机构分配给制造商一个固定的代码。
- 其后 16 比特位为设备类别，从相关机构预定义的类别定义注册信息中提取。
- 其后 16 比特位为设备子类别，从相关机构预定义的子类别定义注册信息中提取。
- 最后 8 比特位为制造商指定的型号。

程序 ID 字符串可以用 *Query ID* 网络管理报文经由网络读取。它同时也是主动提供的手动服务请求网络管理报文的一部分。

unsigned　　two_domains　　: 1;

这一个比特位指明域表有 2 个表项(见 B.2)。如果该比特位为零，则域表只有一个表项。

unsigned　　explicit_addr　　: 1;

这一个比特位指明节点在其应用中使用显式报文寻址。如果该比特位置位，则应用缓冲区包含一个 11 字节的显式地址字段。对任何使用显式寻址或 *nv_in_add* 结构的应用，该比特位均置位。

Unsigned　　msg_process　　: 1;

当这一个比特位置为 1 时意味着该应用可以处理显式报文。

Unsigned　　state　　: 3;

广东省网络空间安全协会受控资料

这 3 个比特的字段表示节点状态，其不同状态的值在 13.4 中描述。

```
unsigned    address_count              : 4;        // offset 0x16
```

该字段指明地址表(见 B.3)中表项的数目(0—15)。

```
unsigned    receive_trans_count        : 4;        // offset 0x17
```

该字段指明接收事务的数目。接收事务的数目比这个字段中指明的数目多 1。值 0xF 意为有 16 个或多于 16 个接收事务。

```
unsigned    app_buf_out_size           : 4;        // offset 0x18
unsigned    app_buf_in_size            : 4;
unsigned        net_buf_out_size       : 4;        // offset 0x19
unsigned        net_buf_in_size        : 4;
```

这些字段指明应用和网络缓冲区的大小，字段编码见表 B.1。

表 B.1 缓冲区大小编码

字段值	缓冲器大小
0	255
2	20
3	21
4	22
5	24
6	26
7	30
8	34
9	42
10	50
11	66
12	82
13	114
14	146
15	210

广东省网络空间安全协会受控资料

```
unsigned    net_buf_out_priority_count : 4;        // offset 0x1A
unsigned    app_buf_out_priority_count : 4;
unsigned    app_buf_out_count          : 4;        // offset 0x1B
unsigned    app_buf_in_count           : 4;
unsigned    net_buf_out_count          : 4;        // offset 0x1C
unsigned    net_buf_in_count           : 4;
```

这些字段指明应用和网络缓冲区的数目。注意，如果有一个优先输出缓冲区计数是零，则两个(应

用和网络)优先输出缓冲区计数必须都为零。这些字段的编码见表 B.2。

表 B.2 缓冲区计数编码

字段值	缓冲区数目
0	0
2	1
3	2
4	3
5	5
6	7
7	11
8	15
9	23
10	31
11	47
12	63
13	95
14	127
15	191

广东省网络空间安全协会受控资料

```
unsigned    tx_by_addr                  : 1;
```

这一个比特位指明该节点对地址表中每一个唯一的目的地地址都保留单独的出向事务空间。

```
unsigned    idempotent_duplicate        : 1;
```

这一个比特指明当一个重试请求送到应用程序时,协议处理器将该应用缓冲区内的等幂重试比特位置位。

```
unsigned    alias_count                 : 6;
```

该字段指明网络变量别名表中的表项数。如果该节点是一个网络接口,并且正在处理应用的网络变量选择号,则该字段为 0。这个字段的最大值为 62。

```
unsigned    msg_tag_count               : 4;
```

该字段指明节点使用的可绑定报文标签的数目。其范围为 0～15。

B.2 域表

域表定义该节点从属的域。域表存在非易失性存储器中,并且是节点安装时写入的网络映像的一部分。在开发期间,域表的内容在加载节点时下载。

```
#define AUTH_KEY_LEN    6
#define DOMAIN_ID_LEN   6

typedef struct {
    unsigned    id[ DOMAIN_ID_LEN ];                    // offset 0x00
```

```
    unsigned    subnet;                          // offset 0x06
    unsigned                          : 1;
    unsigned    node                  : 7;       // offset 0x07
    unsigned    len;                             // offset 0x08
    unsigned    key[ AUTH_KEY_LEN ];             // offset 0x09
} domain_struct;
```

B.2.1 域表字段描述

unsigned id[DOMAIN_ID_LEN];

在 GB/Z 20177 网络中每个域有一个唯一的、长度为 0、1、3 或 6 字节的 ID。如果 ID 短于 6 个字节，则它在该字段中左对齐。

unsigned subnet;

该字段指明在该节点从属的域内的子网 ID。对每一个域，子网 ID 的范围可以为 1～255。0 是无效的子网 ID。

unsigned node;

该字段指明节点在这个子网内的节点 ID(1～127)。0 是无效的节点 ID。

unsigned len;

该字段指明域 ID 的字节长度(为 0、1、3 或 6 字节)。值 0xFF (255)意味着这个域表项没有使用。

unsigned key[AUTH_KEY_LEN];

该字段指明在这个域中用于鉴别事务的 6 字节鉴别密钥。该密钥必须与这个域内所有参与和该节点鉴别事务的其他节点的密钥相匹配。鉴别密钥可以使用 *Update Key* 网络管理报文经由网络进行增量更新。

广东省网络空间安全协会受控资料

B.3 地址表

地址表定义了该节点可能发送隐式寻址报文和网络变量前往的那些网络地址。它还定义了该节点从属的那些组。地址表存在非易失存储器中，并且是节点安装期间写入的网络映像的一部分。

```
typedef enum { UNBOUND, SUBNET_NODE, Unique_Node_ID, BROADCAST } addr_type;

typedef union {
    group_struct        gp;
    snode_struct        sn;
    bcast_struct        bc;
    turnaround_struct   ta;
} address_struct;

const address_struct *access_address(byte index);
update_address(const address_struct * address_entry, byte index);
    byte addr_table_index(message_tag_name);
```

应用程序可以使用 access_address() 和 update_address()访问程序读和写地址表中的任何一个表项。对应于该程序中声明的任意一个报文标签的地址表项的索引号可以用 addr_table_index()函数确定。地址表由多达 15 个表项组成,每个表项长度为 5 字节。

每个表项可以是 5 种格式中的一种:组地址,子网/节点地址,广播地址,回转地址,或未使用。组地址用于多播寻址,在一个网络变量或报文标签被用于具有多于 2 个成员的连接时使用。子网/节点地址用于单播寻址,在一个输出网络变量或报文标签被用于与另一个节点连接时使用。回转地址用于网络变量只绑定到同一节点的其他网络变量,而不绑定到其他节点的任何网络变量时。48 比特位 Unique_Node_ID 格式的目的地地址不在地址表中使用,但可能在显式寻址的报文中用作目的地地址。出于完整性考虑,该格式在下面描述。

地址表中的表项可以用 *Query Address* 和 *Update Address* 网络管理报文经由网络进行读和写。一个使用组地址格式的表项可以用 *Update Group Address* 网络管理报文经由网络进行更新。

地址表的表项的第一个字节指明该表项的格式:

0　　未使用 / 回转格式

1　　(子网,节点)格式

3　　广播格式

128～255　　组格式

在应用程序中声明的任意一个可绑定的报文标签都按声明顺序从第一个表项开始分配在地址表中。然后是网络变量使用的表项。通常,这些表项由绑定工具分配。

重复定时器、重试次数、接收定时器和事务定时器对这些格式中的几种是通用的,见 B.3.11 中的描述。

广东省网络空间安全协会受控资料

B.3.1　组地址格式声明

```
typedef struct {
    unsigned    type        : 1;        // offset 0x00
    unsigned    size        : 7;
    unsigned    domain      : 1;        // offset 0x01
    unsigned    member      : 7;
    unsigned    rpt_timer   : 4;        // offset 0x02
    unsigned    retry       : 4;
    unsigned    rcv_timer   : 4;        // offset 0x03
    unsigned    tx_timer    : 4;
    unsigned    group       : 8;        // offset 0x04
} group_struct;
```

B.3.2　组地址字段描述

unsigned　type　: 1;

这一个比特位对组地址是 1,对所有其他格式为 0。

unsigned　size　: 7;

该字段指明组的大小(2～64)。组的大小包括该报文的发送者。如果这个字段是 0,则组的大小没有限制,并且必须使用无确认或重复服务。

unsigned　domain　: 1;

该字段指明这个地址在域表中的索引号(0 或 1)。

unsigned　member　: 7;

该字段指明这个节点在这个组内的成员 ID 号(0～63)。大小没有限制的组这个字段的值为 0。成员 ID 用于确认,以便允许一个确认多播报文的发送者记住哪些节点已经响应。

```
unsigned    group                    : 8;
```

该字段指明在这个域内的这个组的 ID 号。组 ID 的范围可以是 0～255。组 ID 通常由网络变量绑定工具分配。

B.3.3 子网/节点地址格式声明

```
typedef struct {
    addr_type    type;                          // offset 0x00
    unsigned     domain          : 1;           // offset 0x01
    unsigned     node            : 7;
    unsigned     rpt_timer       : 4;           // offset 0x02
    unsigned     retry           : 4;
    unsigned                     : 4;           // offset 0x03
    unsigned     tx_timer        : 4;
    unsigned     subnet          : 8;           // offset 0x04
} snode_struct;
```

B.3.4 子网/节点地址字段描述

```
addr_type    type;
```

该字段包含值 SUBNET_NODE (1)。

广东省网络空间安全协会受控资料

```
unsigned    domain                   : 1;
```

该字段指明这个地址在域表中的索引号(0 或 1)。

```
unsigned    node                     : 7;
```

该字段指明在指定子网和域内的节点的 ID 号(1～127)。0 不是有效的节点 ID。

```
unsigned    subnet                   : 8;
```

该字段指明在指定域内的子网 ID 号(1～255)。0 不是有效的子网 ID。

B.3.5 广播地址格式声明

```
typedef struct {
    addr_type    type;                          // offset 0x00
    unsigned     domain          : 1;           // offset 0x01
    unsigned                     : 1;
    unsigned     backlog         : 6;
    unsigned     rpt_timer       : 4;           // offset 0x02
    unsigned     retry           : 4;
    unsigned                     : 4;           // offset 0x03
    unsigned     tx_timer        : 4;
    unsigned     subnet          : 8;           // offset 0x04
} bcast_struct;
```

B.3.6 广播地址字段描述

```
addr_type    type;
```

该字段包含值 BROADCAST (3)。

```
unsigned    domain                          : 1;
```

该字段指明这个地址在域表中的索引号(0 或 1)。

```
unsigned    backlog                         : 6;
```

该字段指明由于使用这个地址的确认或请求/响应报文广播所造成的信道使用统计的估计。它应被设置为一个预期的确认或响应数目。例如,这可以是在最坏情况下一个信道上预期响应 Query ID 报文的节点数目。如果这个数目未知,该字段可以被设置为 0,在这个情况下假设信道使用统计数为 15。

```
unsigned    subnet                          : 8;
```

该字段指明在指定域内的子网号(1～255)。该报文提交给这个子网中的所有节点。如果子网号为 0,则该报文提交给这个域中的所有节点。如果请求/响应或确认服务使用广播地址,一旦收到第一个响应或确认,该事务就结束了。其后的响应或确认被丢弃。

B.3.7 回转地址格式声明

```
typedef struct {
    addr_type    type;                                  // offset 0x00
    unsigned     turnaround;                            // offset 0x01
    unsigned     rpt_timer                  : 4;        // offset 0x02
    unsigned     retry                      : 4;
    unsigned                                : 4;        // offset 0x03
    unsigned     tx_timer                   : 4;
} turnaround_struct;
```

B.3.8 回转地址字段描述

```
addr_type    type;
```

包含值 UNBOUND (0)

广东省网络空间安全协会受控资料

```
unsigned turnaround;
```

该字段包含值 1。如果回转字段是 0,则这个地址表项未使用。

B.3.9 协议处理器地址格式声明

注:这个格式在地址表中不使用,但它可以作为显式寻址报文的目的地地址。

```
typedef struct {
    addr_type    type;                                  // offset 0x00
    unsigned     domain                     : 1;        // offset 0x01
    unsigned                                : 7;
    unsigned     rpt_timer                  : 4;        // offset 0x02
    unsigned     retry                      : 4;
    unsigned                                : 4;        // offset 0x03
    unsigned     tx_timer                   : 4;
    unsigned     subnet                     : 8;        // offset 0x04
    unsigned     nid[UNIQUE_NODE_ID_LEN];  // offset 0x05
} unid_struct;
```

B.3.10 协议处理器地址字段描述

```
addr_type    type;
```

该字段包含值 UNIQUE_NODE_ID (2)。

unsigned domain :1;

该字段指明目的地地址在域表中的索引号(0或1)。但是,用Unique_Node_ID寻址的报文可以被发送到任何域中。未配置的节点将接受任何域上以Unique_Node_ID格式寻址的报文。当发生这种情况时,称该报文在柔性域上被接收,任何响应或确认都被送回接收它的那个域,其源子网和节点标识符为零。

unsigned subnet :8;

该字段指明在指定域内的目的地子网号(1～255)。它只用于报文的路由。如果该报文要经过该域中的所有路由器,则它将被设置为零。

unsigned nid[Unique_Node_ID_LEN];

该字段指明目标节点的48位Unique_Node_ID。

B.3.11 定时器字段描述

unsigned rpt_timer :4;

该字段指明在使用重复服务时出向报文重复发送之间的时间间隔。这个字段的编码见表B.3。

unsigned retry :4;

该字段指明确认、请求/响应或重复服务的重试次数(0～15)。最多的报文发送的次数为这个数字加1。

unsigned rcv_timer :4;

当节点接收多播(组)报文时,接收定时器被设置为由这个字段指定的时间间隔。如果在接收定时器时间未到之前收到一个具有相同事务ID的报文,则它被认为是前一个报文的重试。这个字段的编码见表B.3。

广东省网络空间安全协会受控资料

unsigned tx_timer :4;

该字段指明在使用确认或请求/响应服务时重试之间的时间间隔。在每一次试图重试时,以及在任何确认或响应被收到时(最后一个除外),事务重试定时器重新启动。对于请求/响应服务,请求节点在设置事务定时器时,应考虑到应用响应所必须的延时。例如,对于要写入非易失性存储器的网络管理报文,这一点非常重要。事务定时器字段的编码见表B.3。

表B.3中定义的non_group_timer字段描述见B.6.1。

表B.3 定时器字段值编码

单位为毫秒

字段值	rpt_timer	rcv_timer	tx_timer	non_group_timer
0	16	128	16	128
1	24	192	24	192
2	32	256	32	256
3	48	384	48	384
4	64	512	64	512
5	96	768	96	768
6	128	1 024	128	1 024
7	192	1 536	192	1 536

表 B.3(续)

单位为毫秒

字段值	rpt_timer	rcv_timer	tx_timer	non_group_timer
8	256	2 048	256	2 048
9	384	3 072	384	3 072
10	512	4 096	512	4 096
11	768	6 144	768	6 144
12	1 024	8 192	1 024	8 192
13	1 536	12 288	1 536	12 288
14	2 048	16 384	2 048	16 384
15	3 072	24 576	3 072	24 576

B.4 网络变量表(资料性)

有三个表与网络变量有关:网络变量配置表、网络变量别名表和网络变量固定表。网络变量配置表定义节点中网络变量的配置属性,它存在非易失性存储器中,以便在节点安装期间能被修改,它也是节点安装期间写入的网络映像的一部分。网络变量别名表定义节点中别名网络变量的配置属性,它存在非易失性存储器中,紧跟在网络变量配置表之后。在节点中网络变量固定表的结构是每个实现都不同的,这里编入文档的结构与由相应的网络管理报文使用的结构相同,因此本节中数据的描述和使用与附录 A 中对网络变量数据的描述和使用相匹配。

广东省网络空间安全协会宣传资料

```
typedef struct {
    unsigned    nv_priority          : 1;        // offset 0x00
    unsigned    nv_direction         : 1;
    unsigned    nv_selector_hi       : 6;
    unsigned    nv_selector_lo       : 8;        // offset 0x01
    unsigned    nv_turnaround        : 1;        // offset 0x02
    unsigned    nv_service           : 2;
    unsigned    nv_auth              : 1;
    unsigned    nv_addr_index        : 4;
} nv_struct;

typedef struct {
    nv_struct    nv_cnfg;                         // offset 0x00
    unsigned    primary;                          // offset 0x03
    unsigned    long host_primary;                // offset 0x04
} alias_struct;

const nv_struct * access_nv(byte index);
void update_nv(const nv_struct * nv_entry, byte index);
const alias_struct * access_alias (byte index);
void update_alias (const alias_struct * alias_entry, byte index);
```

```
typedef struct {
    unsigned    nv_sync                 : 1;        // offset 0x00
    unsigned                            : 2;
    unsigned    nv_length               : 5;
    void      * nv_address;                         // offset 0x01
} nv_fixed_struct;
```

网络变量配置表中的表项可以用 Query/Update Network Variable Configuration 网络管理报文经由网络读和写。网络变量固定表不能被写，下载应用映像时除外。

网络变量别名表可以有多达 62 个表项。

网络变量配置表字段描述(资料性)

unsigned nv_priority : 1;

如果网络变量使用优先报文服务，这个比特位设置为 1。

unsigned nv_direction : 1;

如果这个网络变量是输出网络变量，该比特位置 1；如果是输入网络变量，该比特位置 0。这个比特位不能被应用程序或 Update Network Variable Configuration 网络管理报文更改。从技术上说，该比特位不是网络配置存储器数据，放在网络变量配置表中只是为了提高效率。

广东省网络空间安全协会受控资料

unsigned nv_selector_hi : 6;

unsigned nv_selector_lo : 8;

这两个字段构成一个 14 比特位的网络变量选择号，取值范围 0～0x3FFF 。选择号 0x3000～0x3FFF 保留作为非绑定网络变量使用，其选择号等于 0x3FFF 减去网络变量索引号。选择号 0～0x2FFF 用于绑定的网络变量。在任何一个节点上的所有输入网络变量必须具有不同的选择号。在任何一个节点上的所有输出网络变量必须具有不同的选择号。

unsigned nv_turnaround : 1;

如果这是一个回转网络变量，即它被绑定到同一节点上的另一个网络变量时，该比特位设置为 1。

unsigned nv_service : 2;

该字段指明用于交付这个网络变量的服务类型。

unsigned nv_auth : 1;

如果这个网络变量使用鉴别事务，则该比特位设置为 1。

unsigned nv_addr_index : 4;

该字段指明用这个网络变量进入地址表的索引号(0～14)。如果该网络变量没有和地址表项关联，则使用索引号值 15(0x0F)。多个网络变量可以使用同一个地址表的索引号。

网络变量别名表字段描述（资料性）

nv_struct nv_cnfg;

这个定义和上述的网络变量配置表相同。

unsigned primary;

这是进入网络变量配置表的索引号。如果值为 0xFF ,则表示应使用随后的两个字节作为索引号。

unsigned long host_primary;

网络变量配置表索引。这个字段只有在前面的 primary 字段设置为 0xFF 才会出现。只有当所用的网络变量索引号大于 254 时,这些字节才是必须的。

网络变量固定表字段描述(资料性)

unsigned nv_sync : 1;

如果这是一个同步网络变量,该比特位设置为 1。

unsigned nv_length : 5;

该字段指明网络变量的字节数(1～31)。

void * nv_address;

该字段是指向网络变量数据在存储器中位置的一个指针。

B.5 自标识结构

有下列四种结构与自标识相关联:

- 固定大小的自标识结构;
- 包含对每个网络变量的自标识描述符的表;
- 节点的自说明文档字符串;
- 网络变量的自说明文档信息;
- 绑定和状态信息的自标识数据。

这些信息构成了在节点制造期间被写入的应用映像的一部分。

广东省网络空间安全协会受控资料

```
typedef struct {
    unsigned long   length;                          // offset 0x00
    unsigned     num_netvars;                        // offset 0x02
    unsigned      version;                           // offset 0x03
    union {
        struct {
            unsigned msb_num_netvars;                // offset 0x04
            unsigned mtag_count;                     // offset 0x05
        } ver1;
        struct {
            unsigned mtag_count;                     // offset 0x04
        } ver0;
    }  variable_part;
} snvt_struct;

typedef struct {
    unsigned     ext_rec                  : 1;       // offset 0x00
    unsigned     nv_sync                  : 1;
```

```
    unsigned    nv_polled                    : 1;
    unsigned    nv_offline                   : 1;
    unsigned     nv_service_type_config : 1;
    unsigned    nv_priority_config           : 1;
    unsigned    nv_auth_config               : 1;
    unsigned    nv_config_class              : 1;
    unsigned    snvt_type_index;                          // offset 0x01
} snvt_desc_struct;
typedef struct {
    unsigned    binding_II     :1;
    unsigned    query_stats     :1;
    unsigned    alias_count     :6;
    unsigned long host_alias;                             // Host node only
} alias_field;
```

自标识首部结构是5字节长。网络变量描述符表紧随其后，并且它对于在应用程序中声明的每一个网络变量都有一个表项。版本0格式对网络变量数组的每个元素都分别有一个表项，版本1格式允许网络变量数组的元素共享一个表项。网络变量描述符表中的每个表项是2字节长。

紧随网络变量描述符表之后的是以空(null)终结的文本字符串，该文本字符串包含该节点的自说明文档。这个字符串的长度可长达255个字节。即使没有自说明文档字符串，空终结符字节仍会出现。

紧随该节点自说明文档字符串之后的是网络变量的扩展记录，用于那些需要扩展记录的网络变量。

紧随网络变量扩展记录之后的是自标识位，自标识位反映了各种绑定和查询状态条件的情况。

广东省网络空间安全协会专版资料

B.5.1 自标识结构字段描述

unsigned long　length;

该字段指明这里所描述的自标识和自说明文档数据结构中的总字节数。

unsigned　num_net_vars;

该字段指明这个节点中声明的网络变量的数目。网络变量数组的每个元素都分别计算。在版本1格式中，这个字节是该节点声明的网络变量数目中最低有效字节，这里一个网络变量数组只算作一项。

unsigned　version;

该字段指明网络变量结构所遵循的格式，定义了版本0和版本1。

unsigned　msb_num_net_vars;

如果前一个字节中的版本号是0，则这个字节不出现。如果版本号是1，则这个字节是该节点声明的网络变量数目中最高有效字节，这里一个网络变量数组只算作一项。

unsigned　mtag_count;

该字段指明这个节点上的应用程序声明的可绑定的报文标签的数目。这些报文标签占据地址表中前面的表项(见B.3)。

B.5.2 网络变量描述符表字段描述

unsigned　ext_rec　: 1;

如果这个网络变量的节点自说明文档字符串后面紧随一个扩展记录，该比特位设置为1。

unsigned nv_sync :1;

如果这是一个同步网络变量，该比特位设置为1。

unsigned nv_polled :1;

如果这个网络变量是被轮询的，该比特位设置为1。

unsigned nv_offline :1;

如果网络管理工具应在这个网络变量被更新前将该节点离线，该比特位设置为1。

unsigned nv_service_type_config :1;

如果这个网络变量服务类型(ACKD，UNACKD，UNACKD_RPT)可以被网络管理报文修改，该比特位设置为1。

unsigned nv_priority_config :1;

如果这个网络变量的优先级可以被网络管理报文修改，该比特位设置为1。

unsigned nv_auth_config :1;

如果这个网络变量的鉴别可以被网络管理报文修改，该比特位设置为1。

广东省网络空间安全协会受控资料

unsigned nv_config_class :1;

如果这是一个配置网络变量，其值存放在非易失性存储器中，该比特位设置为1。

unsigned snvt_type_index;

如果这是一个标准类型网络变量(即SNVT)，该字段指明网络变量类型(1～250)。

B.5.3 SNVT表扩展记录

任何网络变量都可以有扩展记录。只要有一个扩展记录，SNVT描述符中ext_rec字段设置为1。扩展记录按应用程序中网络变量声明的顺序出现。每个扩展记录以一个字节的比特位掩码开始，该掩码定义后面所跟的字段。后面的字段按这里定义的比特位掩码的位顺序出现。这些比特位在掩码中从最高有效位开始(从高到低)。

unsigned mre :1;

如果这个比特位设置为1，则扩展记录包含对这个网络变量最大更新速率的估计值。该字段是一个无符号字节，取值范围0～127，代表速率范围为0～1878.0。速率是由公式$2^{(n/8)-5}$报文/秒给出的，舍入到每秒钟1/10个报文。

unsigned re :1;

如果这个比特位设置为1，则扩展记录包含对这个网络变量平均更新速率的估计值。该字段是一个无符号字节，取值范围0～127，代表速率范围为0～1878.0。速率是由公式$2^{(n/8)-5}$报文/秒给出的，舍入到每秒钟1/10个报文。

unsigned nm :1;

如果这个比特位设置为1，则扩展记录包含这个网络变量的名称。该名称可用多达17个字节的以

空终结的字符串来表达，如果它是网络变量数组元素，则可多达 22 个字节。每个数组元素有它自己的扩展记录，包含该数组名、左方括号、表示元素的索引号的 1，2 或 3 个十进制数字，还有右方括号。

unsigned sd :1;

如果这个比特位设置为 1，则扩展记录包含该网络变量的自说明文档字符串，这是一个多达 1023 个字节的以空终结的字符串。

unsigned nc :1;

如果这个比特位设置为 1，则扩展记录包含这个类型网络变量数目的 16 位计数（只对版本 1 格式）。这用于定义网络变量数组。如果这个比特位被清除，则扩展记录只定义了一个这种类型的网络变量。

B.5.4 SNVT 别名字段描述

unsigned binding_II :1;

如果节点使用新的绑定约束条件，这个比特位设置为 1。支持这些绑定约束条件的节点不要求在非轮询连接中给输出网络变量分配唯一的选择号。例如在支持这些绑定约束条件的节点中两个不同的输出网络变量可以被连接到任何类型的节点的同一个输入网络变量，只要这个输入网络变量不发起轮询。注意，由网络管理或监视节点对这种输出网络变量进行轮询必须使用 Fetch Network Variable 网络管理报文，而不用网络变量轮询报文。

unsigned query_stats :1;

如果 Read Memory 网络管理命令的查询统计寻址模式可被用来提取扩展的统计信息，则这个比特位设置为 1。

广东省网络空间安全协会受控资料

unsigned alias_count :6;

该字段指明被该节点使用的别名网络变量的数目（0～62）。取值 63 表明下面两个字节包含别名网络变量的实际数目。

unsigned long host_alias;

该字段指明被该节点使用的别名网络变量的数目（0～4095）。这个字段仅当 alias_count 是 63 时才出现。

B.6 配置结构

这个结构定义该节点的硬件和收发器属性。它存在非易失性存储器中，其一部分属于在节点制造期间写入的应用映像和在节点安装时写入的网络映像。

```
#define LOCATION_LEN   6
#define NUM_COMM_PARAMS 7

typedef struct {
    unsigned long    channel_id;                    // offset 0x00
    char             location[LOCATION_LEN];        // offset 0x02
    unsigned         comm_clock          : 5;       // offset 0x08
    unsigned         input_clock         : 3;
```

```
    unsigned        comm_type                   : 3;        // offset 0x09
    unsigned        comm_pin_dir                : 5;
    unsigned        preamble_length;                        // offset 0x0A
    unsigned        packet_cycle;                           // offset 0x0B
    unsigned        beta2_control;                          // offset 0x0C
    unsigned        xmit_interpacket;                       // offset 0x0D
    unsigned        recv_interpacket;                       // offset 0x0E
    unsigned        node_priority;                          // offset 0x0F
    unsigned        channel_priorities;                     // offset 0x10
    union {
        unsigned    xcvr_params[ NUM_COMM_PARAMS ];
        direct_param_struct     dir_params;                 // offset 0x11
    }
    unsigned        non_group_timer             : 4;        // offset 0x18
    unsigned        nm_auth                     : 1;
    unsigned        preemption_timeout          : 3;
} config_data_struct;

typedef struct {
    unsigned        collision_detect            : 1;        // offset 0x11
    unsigned        bit_sync_threshold          : 2;
    unsigned        filter                      : 2;
    unsigned        hysteresis                  : 3;
    unsigned        cd_to_end_packet            : 6;        // offset 0x12
    unsigned        cd_tail                     : 1;
    unsigned        cd_preamble                 : 1;
} direct_param_struct;
const config_data_struct config_data;
void update_config_data(const config_data_struct * config_data);
```

广东省网络空间安全协会受控资料

该结构为25字节长，它可以被Read Memory和Write Memory网络管理报文以address_mode=2经由网络读出和写入。在节点复位和初始化收发器时，媒体访问控制处理器从配置结构读取信道参数(偏移0x08～0x17)。因此，在信道参数作任何改变之后，该节点必须被复位以使改变的参数生效。

B.6.1 配置结构字段描述

unsigned long channel_id;

该字段指明这个节点被分配所在信道的ID。

char location[LOCATION_LEN];

该位置字段用于将描述节点物理位置的6字节ASCII字符串传递给网络管理工具。

unsigned node_priority;

该字段指明当节点在信道上发送优先级报文时使用的优先时间片(1～255)。它不应大于该信道上的优先时间片数目。如果节点没有分配优先时间片，这个字段为0。

unsigned channel_priorities;

该字段指明该信道上的优先时间片数目(0～255)。优先时间片数目从1开始。

unsigned non_group_timer : 4;

当节点接收到需要响应或确认的单播或广播(非多播)报文,接收定时器被设置为这个字段指定的时间间隔。如果在该时间间隔到时之前接收到一个报文带有相同的事务ID、优先级、源地址和目的地地址,则它被认为是前一个报文的重试。这个字段的编码见表B.3。当接收到使用Unique_Node_ID寻址的网络管理报文时,使用8.192 s接收定时器代替这个字段的值。

unsigned nm_auth : 1;

该字段指明对网络管理报文进行鉴别。设置该比特位防止节点被非授权网络管理工具所配置。但是,当节点未被配置时不能对接收到的网络管理报文进行鉴别。在设置节点状态为已配置状态之前,网络管理工具应确保网络管理鉴别比特位被设置为期望的状态。

unsigned preemption_timeout : 3;

该字段指明在占先模式下节点等待空闲缓冲区的最大时间。占先模式的超时作为一个差错记录下来并复位该节点。其编码见表B.4。

表B.4 缓冲区超时编码

0	永远
1	2s
2	4s
3	6s
4	8s
5	10s
6	12s
7	14s

广东省网络空间安全协会受控资料

B.7 统计相关结构

下面的结构定义应被用于带有与统计相关的寻址模式的读存储器(Read Memory)命令:

```
Typedef struct {
    unsigned long transmission_errors;
    unsigned long transmit_tx_failures;
    unsigned long receive_tx_full;
    unsigned long lost_messages;
    unsigned long missed_messages;
    unsigned long layer2_received;
    unsigned long layer3_received;
    unsigned long layer3_transmitted;
    unsigned long transmit_tx_retries;
    unsigned long backlog_overflows;
    unsigned long late_acknowledgements;
```

```
    unsigned long collisions;
    unsigned long eeprom_lock;
} stats_struct;
```

上述结构的字段定义如下：

unsigned long transmission_errors;

在数据包接收期间检测到的 CRC 差错的数目，这些差错可能是由于冲突或收发器输入端的噪声引起的。

unsigned long transmit_tx_failures;

节点在重试了配置的次数后未能接收到期望的确认或响应的次数。

unsigned long receive_tx_full;

由于在事务数据库中没有空间导致入向数据包被丢弃的次数。

unsigned long lost_messages;

由于没有可用的应用缓冲区导致入向数据包被丢弃的次数。如果入向报文对应用缓冲区来说太大导致被丢弃，则记录一个差错，但丢失报文的计数不增加。

unsigned long missed_messages;

由于没有可用的网络缓冲区导致入向数据包被丢失的次数。

广东省网络空间安全协会受控资料

unsigned long layer2_received;

这个节点接收到的第二层报文的数目。第二层报文是那些有正确的 CRC，并可以被寻址到任何一个节点的报文。

unsigned long layer3_received;

从协议处理器第三层传输的报文数目。这些报文可包括网络变量更新、显式报文、确认、重试、提示、服务管脚报文以及任何其他类型的报文。

unsigned long transmit_tx_retries;

由这个节点发送的重试的次数。这不包括用重复服务发送报文使用的重试。

unsigned long backlog_overflows;

信道使用统计达到其最大值 63 的次数。

unsigned long late_acknowledgements;

发送事务结束后到达这个节点的确认或响应的数目。

unsigned long collisions;

该字段指明冲突检测或冲突解决发生的次数，如果启动了这些特性。

unsigned long eeprom_lock;

该节点的 EEPROM 锁定状态。如果该字段值为 1，则该节点上经过校验和的 EEPROM 受到保护，防止存储器被写入。

附 录 C
（资料性附录）
行 为 特 性

C.1 信道容量和吞吐率

在定义关键性能参数时，采用下列符号：

bit/s	原始信道速率，比特/秒
Beta1	时间段
Beta2	时间片
w	MAC 随机窗口大小：16 个时间片
$D_{mean}=w/2$	平均繁忙信道数据包间隔：8 个时间片
$p=(1/2w+P_e)$	由于冲突和传输差错导致数据丢失的概率
P_e	由于传输差错导致丢失的概率
AvgPktSize	包括前置码在内的数据包长度
C_{cost}	冲突成本（2 个数据包）
group_size	组的大小
BL	信道使用统计

假设没有冲突且没有传输差错，在使用随机窗口 $w=2\times D_{mean}$ 时可以被传送的帧的数目见公式(C.1)：

$$\text{frames} = (\text{bit/s}) / (\text{Beta2} \times (D_{mean} + \#\text{prislots}) + \text{Beta1} + \text{AvgPktSize})\ [\text{kbit/s}] \cdots\cdots (\text{C.1})$$

式中相关参数见 C.1。

注：#prislots 为优先时间片数。

某些帧由于冲突丢失，某些由于传输差错丢失。因此，对 L3 网络服务可用的数据包带宽为 Net_L3 见公式(C.2)：

$$\text{Net\_L3} = \text{frames} \times (1-p)[\text{kbit/s}] \cdots\cdots (\text{C.2})$$

式中相关参数见 C.1。

当数据包由于冲突丢失时，第四层和第五层协议重传，因而进一步减少了有效带宽。当数据包丢失概率为 p 时，可用于第四层和第五层协议的净带宽为 Net_L4，见公式(C.3)：

$$\text{Net\_L4} = \text{Net\_L3} \times (1-C_{cost}\times p)[\text{TPDUs/s}] \cdots\cdots (\text{C.3})$$

式中相关参数见 C.1。

Net_L4 可以通过选择适当的 w 值而最大化。这个“净带宽”定义了最大事务处理率。除了鉴别事务是个例外，在一个大小为 N 的组中，一个事务要求传送 N 个 PDU。假定所有的组大小相同，则事务处理率见公式(C.4)：

$$\text{Trans\_Rate} = \text{Net\_L4} / \text{group\_size}\ [\text{trans/s}] \cdots\cdots (\text{C.4})$$

式中相关参数见 C.1。

表 C.1 显示对一个现在已实现的协议处理器在 10MHz 下运行的关键吞吐率参数。作下列假设：没有信道差错，信道使用统计估计正确，物理 NPDU 长度 120 比特位，信道上没有配置优先时间片。对于每秒事务处理数目，假定采用确认服务。对于 120 比特位的数据包长度，假定域 ID 长度为 1 字节，并且平均用户数据大小为 3～4 字节。

广东省网络空间安全协会受控资料

表 C.1 关键吞吐率参数

		信道速率	
		10 kbit/s	78 kbit/s
容量 (净 TPDU 数/s)		43 36	388 329
L4 或 L5 事务/s (组大小为 N)	N = 2 N = 4 N = 8 N = 16	~ 18 ~ 9 ~ 4 ~ 2	~ 164 ~ 82 ~ 41 ~ 20
(在 10 kbit/s 和在 78 kbit/s 的单个无差错信道)			

C.2 网络度量方式

busy_cycle 见公式(C.5)

$$\text{busy\_cycle} = \text{Beta2} \times (D_{\text{mean}} + \#\text{pri slots}) + \text{Beta1} + \text{AvgPktSize} \quad \cdots\cdots\cdots(\text{C.5})$$

式中相关参数见 C.1。

对信道使用统计为 BL 的单一信道,预期的平均网络延时为公式(C.6):

$$N_{\text{delay}} = (\text{BL}/2 + 1) \times \text{busy\_cycle} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots(\text{C.6})$$

式中 busy_cycle 见公式(C.5);其他相关参数见 C.1。

在一般情况下,经 k 次中继后的延时见公式(C.7):

$$N_{\text{delay}} = \text{SUM}\{(\text{BLi}/2 + 1) \times \text{busy\_cycle}\}\ i = 1,\cdots,k \quad \cdots\cdots\cdots\cdots\cdots\cdots(\text{C.7})$$

式中 busy_cycle 见公式(C.5);其他相关参数见 C.1。

对负载超过吞吐率的信道或网络(即 BL=1 永远不会为真),最坏情况下 N_{delay} 可能是无限的。这种情况发生的概率非常小,但有可能发生。当 BL = 1 时,起码有的时候,N_{delay} 是有限的。这时,其最坏情况下的值由到达率分布来确定,分布越均衡,最坏情况的延时越小。

假定传输差错概率为常数 P_e,且每经过 k 次中继后随机窗口为常数 w,则成功提交的概率见公式(C.8):

$$P(\text{L3}) = (1 - p)^k \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots(\text{C.8})$$

式中:

$p=(1/2w + P_e)$ ——数据包丢失的概率;

$1/2w$——由于冲突丢失的概率;

P_e——由于传输差错损坏数据包的概率。

图 C.1 绘出单个数据包经 k 次中继后成功提交的概率(P),其中 P_e 是 1%且 w 是 16。由此得出经单一信道的差错概率为 4.13%。提交失败的概率为 $1-P$。失败的概率用于计算发送该数据包最佳的尝试次数(attempts),因为失败的概率是由$(1-P)^{\text{attempts}}$给出的。

广东省网络空间安全协会受控资料

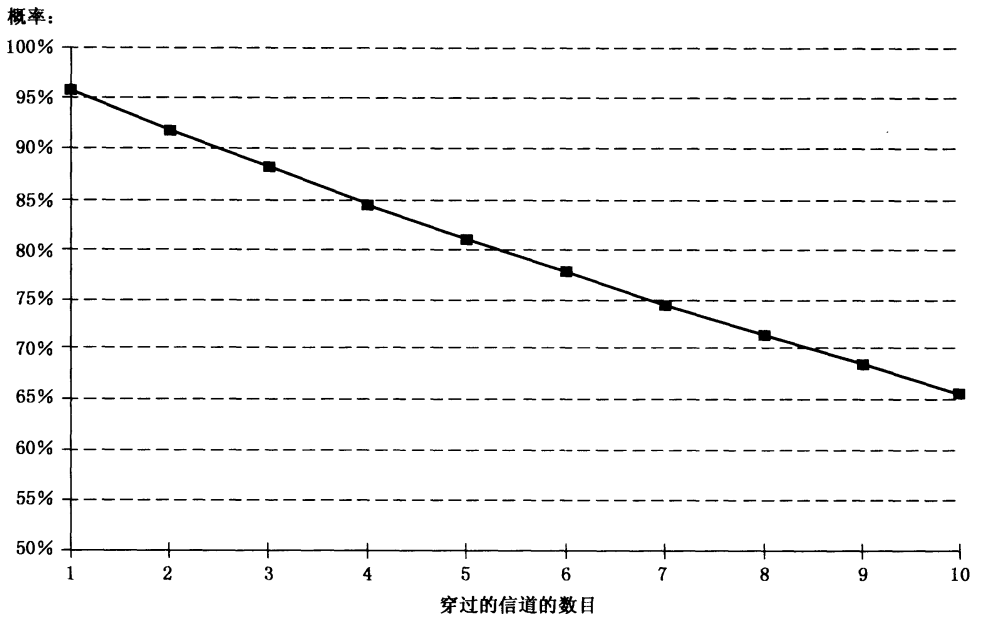


图 C.1　经 k 次中继后成功提交的概率

C.3　事务度量方式

在单一信道单一事务环境下事务完成时间见公式(C.9)：

$$T_{time} = group\_size \times busy\_cycle + T_{np} \qquad \cdots\cdots (C.9)$$

式中：

T_{np}——目标节点的事务处理时间，该时间对传输事务是可以忽略的，但对请求—响应(Request-Response)事务可能是非常重要的。

其他参数见 C.1。

上面忽略了冲突(或传输差错)。当考虑到冲突和多个事务时，该时间的增加见公式(C.10)：

$$T_{time} = x \times L4\_timer + y \times busy\_cycle + T_{np} \qquad \cdots\cdots (C.10)$$

式中：

$x = 0, 1, \ldots, L4\_retries$；

$y \leqslant \max(group\_size, BL)$。

其他参数见 C.1。

假定冲突和差错的组合概率为 p，在组大小为 n 时，在 k 次以内重试后事务完成的概率见公式(C.11)和(C.12)：

$$P\{no\ retry\} = (1-p)^n \qquad \cdots\cdots (C.11)$$

$$P\{\leqslant k\ retries\} = \sum_{i=n}^{k}\left(\frac{k+1}{i}\right)p^i(1-p)^{(k-i+1)}((1-p)^{(k-i+1)})^{(n-1)} \qquad \cdots\cdots (C.12)$$

$$k \geqslant 0, n \geqslant 2$$

式中：

p——数据包经信道丢失的概率。

$P\{\leqslant k\ retries\}$——由下列两个概率相乘计算出来的：

$P\{lose\ i\ messages\ in\ k+1\ attempts\}$

$P\{successfully\ receive\ all\ n-1\ ACKs\ given\ k-i+1\ successful\ msgs\}$

第一个概率是二项式概率，由等式中的前三项表示。第二个概率是由最后一项给出的。上述概率绘图见图 21。

广东省网络空间安全协会受控资料

C.4 边界条件——上电

上电对事务控制子层有影响:对复位后的第一个事务(事务编号为0)没有重复检测。由于这个原因,复位后的第一个应用操作应该是等幂的。

重启动一个自学习路由器也有影响:对发往某个(子网,节点)地址的报文是用发往所有地址的方式路由的,直至该路由器学习到了这个子网的位置,这通常发生在从那个子网送回第一个确认之时。

C.5 边界条件——重负载

负载超过信道容量将造成某些事务延时的增加。这可能导致超时和事务因失败或只有部分成功而被异常终止。

只要估计和实际信道使用统计相符,信道容量将保持在表C.1中定义的水平不变。信道使用统计不相符肯定会减少信道容量。低估的负面效果可能导致产生更多的冲突和吞吐率的损失。如果对信道使用统计的高估偏离不太大,相对而言,对信道吞吐率的影响不大。因此,在协议中对信道使用统计的计算宁愿估计偏高,也不要估计过低。

广东省网络空间安全协会受控资料

附 录 D
（规范性附录）
PDU 概要

本附录提供了所有协议数据单元(PDU)的概要描述。PDU 的语法图示描述见图 D.1。

所示的物理协议数据单元(PPDU)在 MPDU/LPDU 前面包含位同步和字节同步字段。参见对给定信道的相应位同步和字节同步字段的详细描述。

下列解释性注释说明图 D.1 的字段大小。每个字段上面的数字是该字段大小的比特数。

- 字段值：为方便语义描述，大部分的字段值定义为符号范围。一个符号值域(S_0，S_1，S_2，…，S_n)总是按定义的顺序映射到数字值域(0，1，2，…，n)。
- 位顺序：在一个字节内位的传输顺序总是“最高有效位在前”，即最高有效的位最先传送。在图 D.1 中，最高有效位是一个字节中最左边的位。
- 字节顺序：字节的传输顺序也总是“最高有效字节在前”，即最高有效的字节最先传送。在图 D.1 中，最高有效字节是一个字段中最左边的字节。

广东省网络空间安全协会受控资料

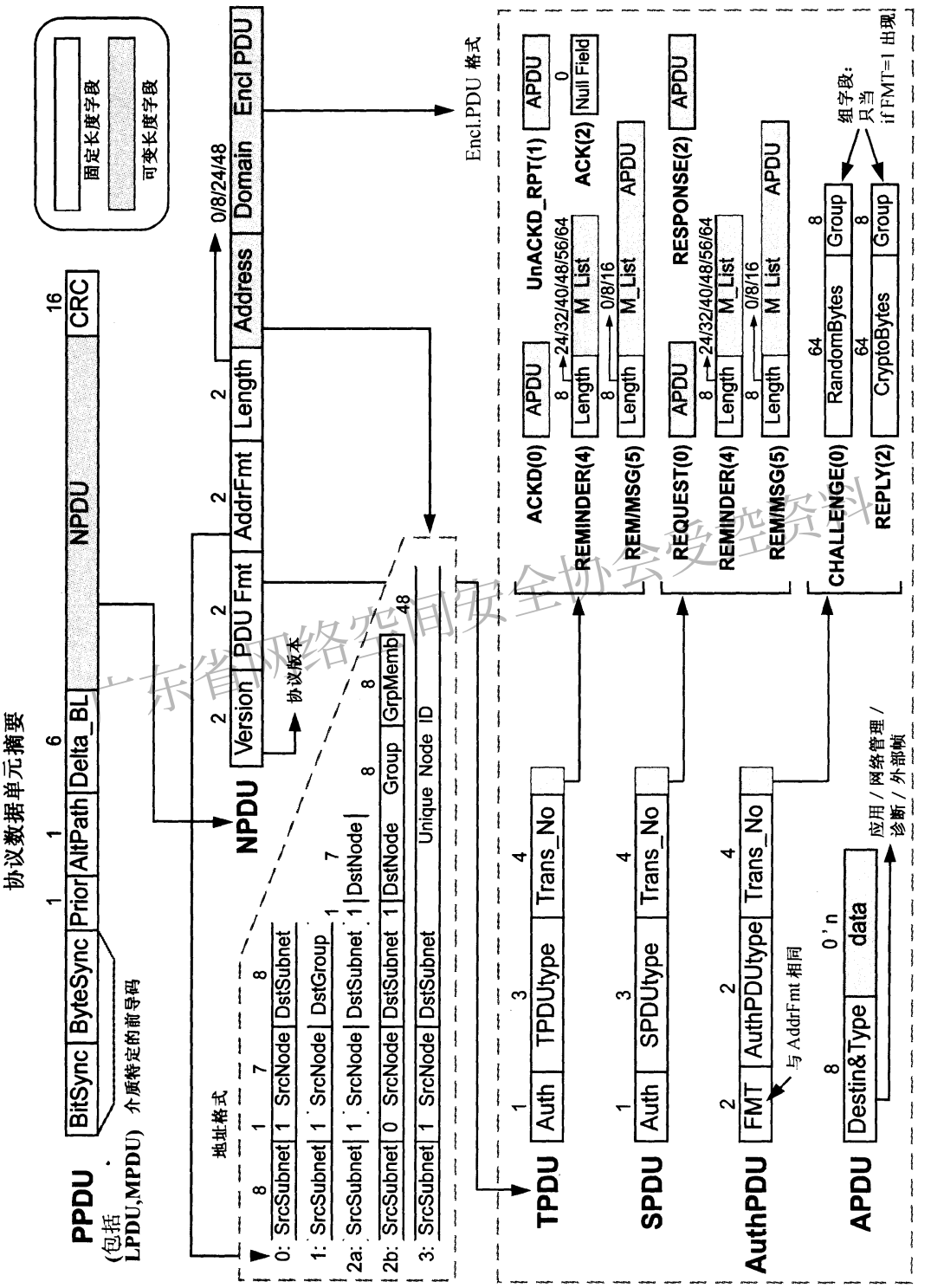


图 D.1 协议 PDU 概要

广东省网络空间安全协会受控资料

附 录 E
（规范性附录）
命名和编址

E.1 地址类型和格式

本文件地址是层次结构的。有三种基本地址类型，每个地址有三个部分，如下所示：

（域、子网、节点）

（域、子网、Unique_Node_ID）

（域、组、成员）

单个地址成分的语法和语义在 E.2～E.5 中描述。源地址和目的地地址是在每个 PDU 中传输的。为此，上面所示的基本编址格式会被进一步组合成为成对编址，在 E.6 中定义。

每个地址是组合的第 3 层和第 4 层地址，在第 2 层不提供编址。域和子网地址成分用于路由，可以称为网络地址。Unique_Node_ID 是名称而不是地址，因为节点移动时它并不改变。因此，用于路由（第 3 层）和用于命名（第 4 层）的地址部分被组合成本文件编址。

地址的长度通常是不同的，但在每个域内部是不变的。域字段的长度是变化的（0、1、3 或 6 字节）；子网和组字段为 8 比特位宽，允许每个域多达 256 个组和 255 个子网（子网 0 是保留值）。节点字段是 7 比特位（全为 0 的字段不用）。组成员字段是 6 比特位，取值范围 0～63。Unique_Node_ID 字段宽 48 比特位。由此可以得出每个域$(2^8-1)\times(2^7-1)$，即约 2^{15} 个节点。在一个系统中可以有多个域，以增加组成员的地址、节点等。

广东省网络空间安全协会受控资料

E.2 域

域标识符是编址层次结构中的第一部分。这个标识符在一定范围内对一个域给出了唯一的标记。域标识符的长短取决于这里所述的情况。48 比特位的域标识符用于提供全球的唯一性。当该情形在某些时候或某些地方受到另外一些限制时，（比如说，物理上，在一个建筑物内），可以用较短的域标识符。

域是一个虚拟网络，所有通信都限制在一个域内进行，其中一个原因是因为 NPDU/TPDU 的源地址和目的地地址必须从属于同一个域（见 E.6）。另一个原因是协议栈不支持相应的网际协议，在网际协议中需要处理域之间的通信，必须借助于应用层网关。

域也是一个管理的单元。特别是，组和子网地址是由负责这个域的网络管理员分配的，它们只有在这个域的范围内有意义。

E.3 子网和节点

子网标识符是编址层次结构中第二部分，它的值唯一标识域内的一个子网；子网地址为零表示该子网未定义或未知。

子网是域内的一个子集，其包含 0～127 个节点，且具有这样的特性，即子网内部没有路由。子网是信道路由的抽象，即子网是逻辑信道，无须对应物理信道拓扑结构。一个或多个子网可以对应于单一信道（或对应于二个或多个由存储转发中继器或网桥连接的信道）。

注：子网这一术语通常在涉及到网络中没有路由的一个子网内部的通信时使用。子网还有其他的特性，它最多包含 127 个节点。因此，二个或多个子网可能被包含在通常所谓的一个子网中，在 GB/Z 20177 中这叫做一个信道。亦即，信道是带宽的一个物理单元，而子网是协议中用于路由的一个逻辑结构。还要注意，信道是带宽的一个物理单元，因此一个信道由一个或多个网段组成。当一个信道由多个网段组成时，这些网段应通过物理层

中继器连接，以使该信道带宽保持一致，而不管它包含多少个网段。

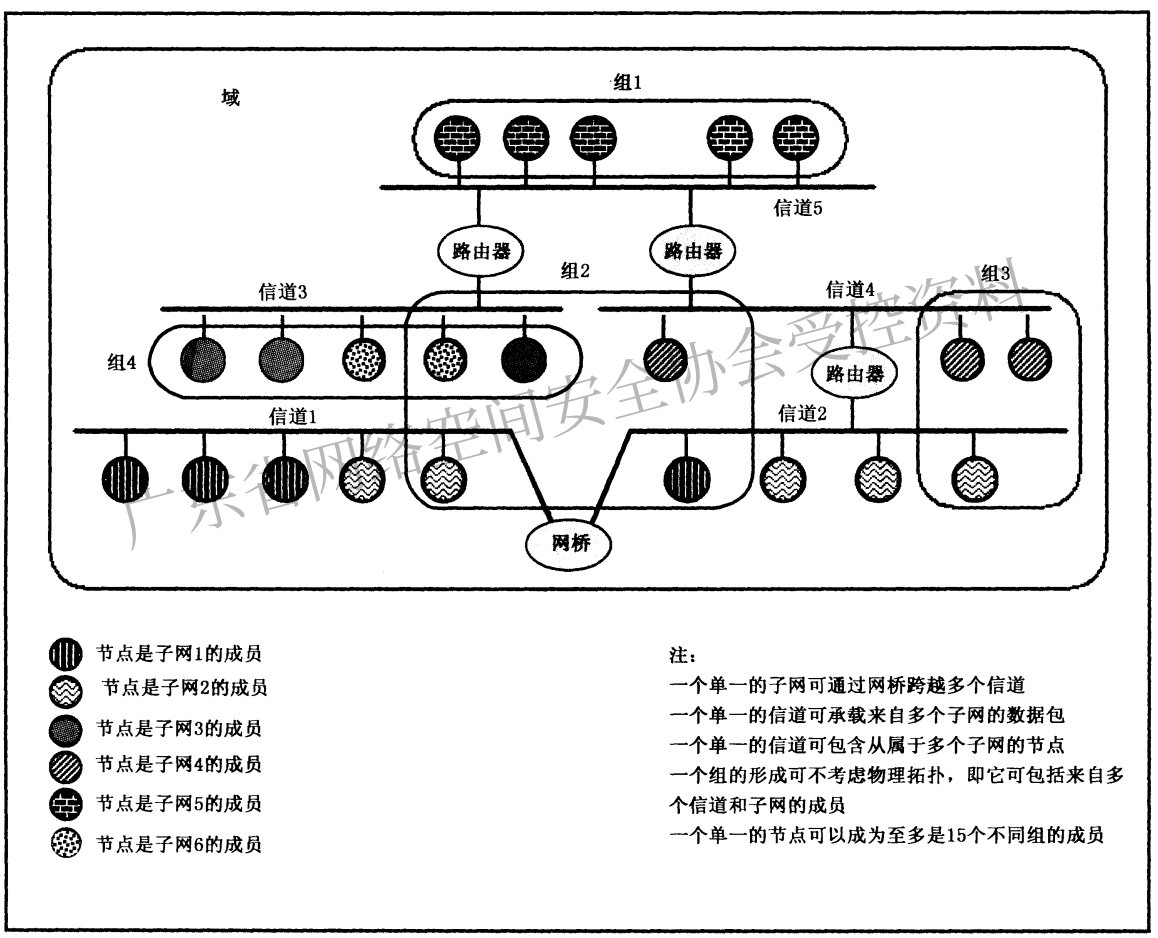
节点标识符标识子网内的一个(逻辑)节点。一个物理节点可从属于多于一个子网，这时，对于它所属的每一个子网都分配给这个物理节点一个(逻辑)节点号。一个物理节点最多可从属于二个子网，这二个子网必须属于不同的域。

E.4 组

组标识符是编址层次结构中的第二部分。它唯一标识了一个域内的一组节点。在这个组内，单个成员由编址结构的第三部分(即成员号)标识。

组编址方便了一对多的通信。组是为了支持功能寻址，特别是为了支持应用协议中使用的网络变量的概念。

物理拓扑和逻辑编址见图 E.1。



广东省网络空间安全协会受控资料

图 E.1 物理拓扑和逻辑编址(单个域)

E.5 Unique_Node_ID 和节点地址分配

每个符合 GB/Z 20177 的节点都被分配一个唯一的 48 比特位的标识符，称为 Unique_Node_ID。这个标识符是全世界唯一的，并且在制造时设定。这个标识符的值从节点制造完成后就不再改变。

一个未配置的节点除了这个 48 比特位的 Unique_Node_ID 外没有分配的地址。这些未配置的节点接收来自所有域的数据包，查找并响应所有包含该节点的 48 比特位 Unique_Node_ID 的任何数据包。

一个节点可以被分配多个地址。除 Unique_Node_ID 外，一个节点通常被分配一个(域、子网、节点)类型的地址和 0～15 个(域、组、成员)类型的地址。一个节点通常只是一个域的成员。但一个节点

最多可以是二个域的成员。一个属于两个域的节点有二个(域、子网、节点)地址,对每个域都有一个地址。

当使用 Unique_Node_ID 作为地址时,它只能用作目的地地址,并且它必须与域和源子网地址部分组合一起使用(见 E.6)。

E.6 NPDU 编址

对 NPDU 编址,是将在 E.1 中介绍的基本编址格式和表 E.1 及图 E.2 中定义的(源、目的地)地址配对组合在一起。一个 NPDU 以 5 种可能的编址格式之一携带源地址和目的地址。

表 E.1 NPDU/TPDU/SPDU 编址——逻辑地址格式

种类	逻辑地址格式	与 TPDU/SPDU 类型使用	
#0:	(Domain, sourceSub-Node,destSubnet)	广播:	域范围或单个子网内
#1:	(Domain, sourceSub-Node, destGroup)	多播:	报文或提示报文,
#2a:	(Domain, sourceSub-Node, destSub-Node)	单播:	提示或确认
#2b:	(Domain, sourceSub-Node, destSub-Node, Group, Mem)	多播:	确认
#3:	(Domain, sourceSub-Node, destSub, Unique_Node_ID)	单播:	报文或提示

在每种地址格式中,源子网 0 意为该节点不知道它的子网号。这是节点在用网络管理报文配置之前的情况。在图 E.2 中,注意每个地址格式包含一个 7 比特的源节点字段。源节点字段中的第 8 个比特是选择号字段,它用于提供编址格式的子变种。地址格式#2 是唯一具有这个能力的一种地址格式。在图 E.2 中,每个字段上面的数字代表这个字段宽度的比特数。NPDU 第一个字节包含 NPDU 首部,其中包含协议版本号、所含 PDU 的格式、编址格式和域字段的长度。NPDU 首部的下一部分指定 4 种基本编址格式之一种。NPDU 首部的最后一部分包含该域的长度。该地址字段紧跟在 NPDU 首部之后。

广东省网络空间安全协会受控资料

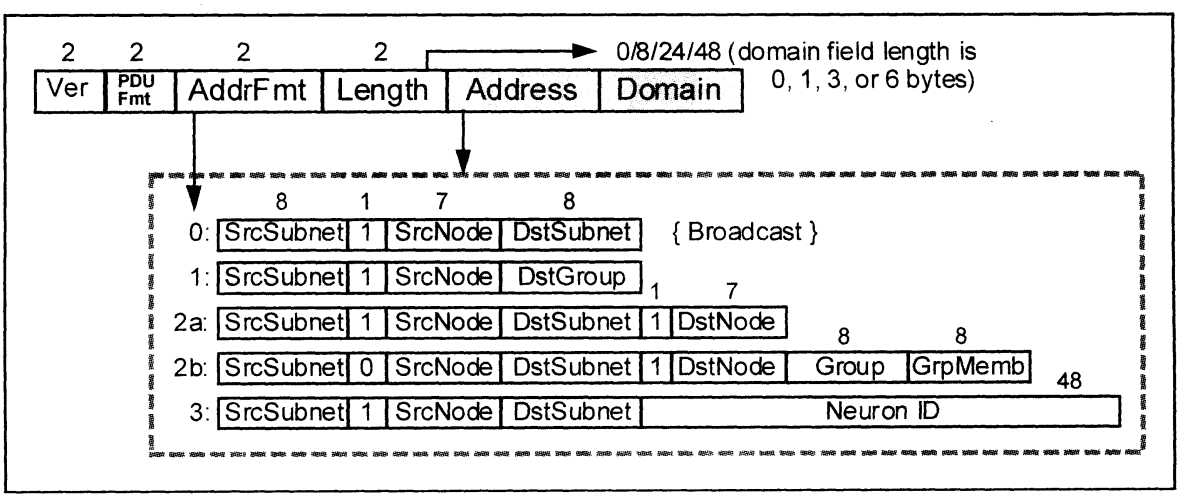


图 E.2 NPDU/TPDU/SPDU 编址——物理地址格式

该地址字段的第一部分是源子网字段,这个字段用于路由器学习拓扑结构和防止数据包的循环。源子网字段和源节点字段的组合用于确认、鉴别质询、回复鉴别质询、响应(当采用请求/响应机制时),以及拒收该节点发出的数据包。域长度字段在 NPDU 首部,跟在源地址和目的地地址对后面。最后是 PDU,其格式在 NPDU 首部规定。

地址格式#0 方便了域范围内的广播。NPDU 包含源节点的(子网、节点)地址和目的地子网。目

的地子网为 0 意味所有子网，而目的地子网在 1～255 范围内表示只广播到指明的目的地子网。

地址格式＃1 支持多播报文交付。它使用（子网、节点）格式的源地址，当目的地地址是（组）时，意味着将报文交付到整个组。

地址格式＃2 有二个形式。在＃2a 形式中，源地址和目的地地址均使用（子网、节点）形式。＃2a 形式用于单播报文交付和确认。＃2b 形式支持组确认，它的源地址部分是（子网、节点）。源地址和目的地地址字段与格式＃2a 相同以方便路由。然后，附在源地址和目的地地址字段后面的是确认节点的组和成员号。

地址格式＃3 支持按 Unique_Node_ID 寻址。因为这种寻址方式的主要意图是方便地址分配，Unique_Node_ID 只能用作目的地地址。该 ID 可以通过网络管理一章中描述的专门的网络管理报文从节点中得到，也可以通过发起一个手动服务请求从节点中获取（也在网络管理一章中描述）。在目的地子网未知的情况下，使目的地子网为 0，从而在整个网络中传播该报文。

广东省网络空间安全协会受控资料

GB/Z 20177.1—2006

广东省网络空间安全协会受控资料

中 华 人 民 共 和 国
国家标准化指导性技术文件
控制网络 LONWORKS 技术规范
第 1 部分:协议规范
GB/Z 20177.1—2006
*
中国标准出版社出版发行
北京复兴门外三里河北街 16 号
邮政编码:100045
网址 www.bzcbs.com
电话:68523946 68517548
中国标准出版社秦皇岛印刷厂印刷
各地新华书店经销
*
开本 880×1230 1/16 印张 33.25 字数 1 060 千字
2006 年 10 月第一版 2006 年 10 月第一次印刷
*
书号:155066·1-28053 定价 128.00 元

如有印装差错 由本社发行中心调换
版权专有 侵权必究
举报电话:(010)68533533

GB/Z 20177.1-2006