摘 要

在当今世界,网络已经成为人们工作生活中必不可少的元素。它是提高工作效率,完善生活质量的重要工具。如何提高网络系统的可靠性,减少不必要的故障损失,最大限度地发挥网络的效益,则成为当今网络管理系统的主要课题。目前，世界上使用最广泛的网管协议是基于 TCP/IP 的简单网络管理协议 SNMP(Simple Network Management Protocol)，该协议简单、易于实现且具有良好的可扩充性，是工业界事实上的网管协议标准。SNMP(简单网络管理协议)是一种网络设备之间客户机/服务器模式的简单通信协议。路由器、交换机、打印机、HUB 等等都可以成为 SNMP 系统中的服务器方。而 SNMP 系统中的客户机往往是单独的一台计算机，轮询网络设备并记录它们所返回的数据。这里允许一台服务器多个客户机的情形。SNMP 允许用很少的网络带宽和内存收集很多有用的系统、网络数据。SNMP 提供了一种统一的、跨平台的网管办法。支持 SNMP 的网络管理系统,在网络上成为很流行的网络管理解决方案。

本论文首先介绍了 SNMP 和网络管理的相关内容，国内外 SNMP 的发展现状和发展趋势。接着系统介绍了 SNMP AGENT 的管理体系结构，通过分析网络管理者和被管理对象之间的功能需求，结合 AGENT 平台，提出了 SNMP 在 AGENT 中应用的各个要点和难点。针对 AGENT 不同命令的不同传输数据量设计了异步和非异步的处理过程；针对 AGENT 的网络通讯管理机制和内存使用机制设计了更符合 AGENT 特色的 SNMP 实现方式；针对 AGENT 中关于回退的数据结构设计了回退处理模式；针对 AGENT 中性能、告警命令的特殊性以及 AGENT 数据库的特点等具体问题设计了代码更优化、代码量更少的 SNMP 模块。本论文更加详细的描述了 SNMP AGENT 中的模块结构、网络管理体系、内部处理方式、信息流程以及在 PSOS 下实现的要点。

最后，根据 SNMP 和 AGENT 特点以及发展的需要，总结了当前的工作，并提出 SNMP 在 AGENT 中应用的优点和不足。

关键词：SNMP，AGENT，MIB，异步，非异步

ABSTRACT

In current society, the network has already become the essential element in people's work and life. Network system brings us not only the convenience of work and life but also the amusement. How to enhance network system's reliability, reduce the nonessential breakdown and loss, and maximum the benefits of network, becomes the network management system's main topic now. At present, the world's most widely used network management protocol is based on TCP/IP simple network management protocol SNMP, which is simple, easy to realize and to have the good expansion. It's the industrial world's network management protocol standard in fact. SNMP (simple network management protocol) is a simple communication protocol between network equipments by client/server pattern. The router, the switch, the printer, the HUB and so on may become server side in the SNMP systems. Client side is often an independent computer in the SNMP system, and its work is to poll network equipment and record the data they return. The situation that a server to many clients is also permitted. The SNMP is able to collect many useful systems, the network data with the very few network band width and the memory. SNMP provides a unified, cross-platform network management, and the network management system which supports SNMP has become a very popular network management solution in the network.

The paper firstly introduces the SNMP, the content related the network management, and the SNMP development in present situation and development tendency in both domestic and foreign. Then it introduces the SNMP AGENT management architecture systematically, and through analyzing the functional requirements between network managers and managed objects with AGENT platform, it proposes elements and difficulties in the application of SNMP to the AGENT. The paper researches asynchronous and non-asynchronous treating processes in view of the different AGETN order with different transmission data quantity; designs SNMP realization way which is more comfortable to AGENT in view of the AGENT network communication management mechanism and the memory use mechanism; think out the retroversion processing pattern in view of the retroversion construction of data in

AGENT; explores a SNMP module with more optimized and less quantity code in view of the special features of the performance, warning orders and the datebase and so on. This paper describes more detailed the SNMP AGENT modular structure, the network management system, the internal treatment way, the information flow and the main point which realize under PSOS.

Finally, according to SNMP and the AGENT characteristic and the need of development, the paper summarizes the current work, and proposes merit and the insufficiency in the application of SNMP to the AGENT.

Keywords: SNMP, AGENT, MIB, asynchronous, non-asynchronous

独 创 性 声 明

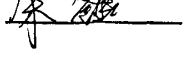
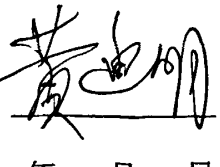
本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：  日期：　　年　月　日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：  导师签名： 

日期：　　年　月　日

第一章 引言

1.1 课题来源和背景

随着计算机和通信技术的飞速发展，网络管理技术已成为重要的前沿技术。目前还没有对网络管理的精确定义。例如，对公用交换网，网络管理通常指实时网络监控，以便在不利的条件下（如过载、故障）使网络的性能仍能达到最佳。又如，狭义的网络管理仅仅指网络的通信量管理，而广义的网络管理指网络的系统管理[1]。网络管理功能可概括为 OAM & P，即网络的运行（Operation）、处理（Administration）、维护（Maintenance）、服务提供（Provisioning）等所需要的各种活动。有时也只考虑前三种，即把网络管理功能归结为 OAM。

所有网络的存在都是为了实现信息的传播和共享，这些信息需要有高效、正确和可靠的手段实现通信[2]。网络管理是非常重要的，因为它能使这种有用的互联网工作能够适当地启动并持续下去。网络管理能监视网络活动，控制设备的运行，并且能够承担起所有的其他相关任务[3]。

简单网络管理协议 SNMP 首先是 IETF 的研究小组为了解决在 Internet 上的路由器管理问题提出的，现在它已经用于管理所有的网络设备，包括路由器、UNIX 工作站和 PC 机等，只要它能支持代理进程的处理能力。而且 SNMP 能在当今各种网络协议上运行，如 IPX、OSI、AppleTalk 以及其它传输协议等[4]。SNMP 简单网络管理协议广泛应用在基于 TCP/IP 网络的分布式系统中，用来实现获取和设置分布式环境中代理端的信息，以实现对分布式系统的有效管理[5]。被管理的实体可以通过代理进程来将自身的状态发送给特殊的 SNMP 管理站软件，也可以由管理端程序通过 SNMP，在网络上任何地方获取所有支持简单网络管理协议的网络节点的信息，并且可对其进行远程配置。

1.2 国内外研究现状和发展态势

简单网络管理协议（SNMP）是目前 TCP/IP 网络中应用最为广泛的网络管理协议。1990 年 5 月，RFC1157 定义了 SNMP（simple network management protocol）的第一个版本 SNMPv1。RFC1157 和另一个关于管理信息的文件 RFC1155 一起，提

供了一种监控和管理计算机网络的系统方法[6]。因此，SNMP 得到了广泛应用，并成为网络管理的事实上的标准。

SNMP 在 90 年代初得到了迅猛发展，同时也暴露出了明显的不足，如难以实现大量的数据传输，缺少身份验证（Authentication）和加密（Privacy）机制。因此，1993 年发布了 SNMPv2，具有以下特点：

- 支持分布式网络管理
- 扩展了数据类型
- 可以实现大量数据的同时传输，提高了效率和性能
- 丰富了故障处理能力
- 增加了集合处理功能
- 加强了数据定义语言

但是，SNMPv2 并没有完全实现预期的目标，尤其是安全性能没有得到提高，如：身份验证（如用户初始接入时的身份验证、信息完整性的分析、重复操作的预防）、加密、授权和访问控制、适当的远程安全配置和管理能力等都没有实现[7]。1996 年发布的 SNMPv2c 是 SNMPv2 的修改版本，功能增强了，但是安全性能仍没有得到改善，继续使用 SNMPv1 的基于明文密钥的身份验证方式。IETF SNMPv3 工作组于 1998 年元月提出了互联网建议 RFC2271-2275，正式形成 SNMPv3。这一系列文件定义了包含 SNMPv1、SNMPv2 所有功能在内的体系框架和包含验证服务和加密服务在内的全新的安全机制，同时还规定了一套专门的网络安全和访问控制规则[7]。可以说，SNMPv3 是在 SNMPv2 基础之上增加了安全和管理机制。

SNMP 最重要的指导思想就是要尽可能简单，以便缩短研制周期。SNMP 的基本功能包括监视网络性能、检测分析网络差错和配置网络设备等。在网络正常工作时，SNMP 可实现统计、配置和测试等功能。当网络出故障时，可实现各种差错检测和恢复功能[8]。虽然 SNMP 是在 TCP/IP 基础上的网络管理协议，但也可扩展到其他类型的网络设备上。

Internet 还有一个远期的网络管理标准 CMOT（Common Management information service and protocol Over TCP/IP），意思是“在 TCP/IP 上的公共管理信息服务与协议”。虽然 CMOT 使用了 OSI 的网络管理标准 CMIS/CMIP，但现在还未达到实用阶段[9]。

1.3 本论文的主要工作内容

传统光网络的管理系统的主要目标是完成基本的设备管理和网络管理功能。由于传输网管采用传统的方式不方便直接对所有的单板进行有效的管理，所以在这里提出了传输网管代理（AGENT）的概念，AGENT程序在传输网络管理组织结构中处于网元管理层的位置，响应管理者的命令完成对网元的配置管理、告警管理、性能管理、维护管理、安全管理等功能。而我实习期间所在的AGENT小组为了项目需要和产品的稳定性在原来的基础上实现AGENT平台为最大程度的跨设备、跨系统平台，增强代码的复用性，引入模块化的思想，将整个Agent软件开发平台分为几个模块。在这样的基础上加入了SNMP模块，以实现OADM SNMP AGENT。让SNMP更能符合AGNET的特点，是本文的工作重心。

中兴传输网管在使用SNMP协议之前一直是使用公司内部定义的私有协议。该协议只能对中兴生产的设备和网管进行传输通信。为了产品需要和市场需求，为了使不同厂家的设备和网管可以方便的互相互控管理，因此将SNMP这一标准化的协议应用到AGENT当中。

本文基于工业实践的基础探讨网元层SNMP AGENT的设计流程，侧重研究网元层AGENT平台中SNMP功能模块的设计与实现。对AGENT中SNMP部分实现过程中的各个步骤进行了详细的描述，结合AGENT和SNMP的特点，在各个细节和重点上，提出了SNMP的优缺点以及解决办法。以下是本论文的组织结构。

第一章是全文的概述，介绍了论文的撰写背景和意义，以及课题解决的主要问题和论文结构。

第二章介绍了SNMP AGENT的体系结构，包括AGENT中与SNMP相关的功能模块、任务、消息队列、事件定义。

第三章简要介绍了SNMP的定义。

第四章在AGENT的功能需求基础上，提出了更符合AGETN特色的SNMP的详细设计和实现。

最后对论文工作进行了总结，并提出了论文存在的不足之处以及以后的研究方向。

第二章 AGENT 软件体系结构

2.1 AGENT 软件平台简介

图 2-1 是 Unitrans ZXONM 网管系统结构层次图：

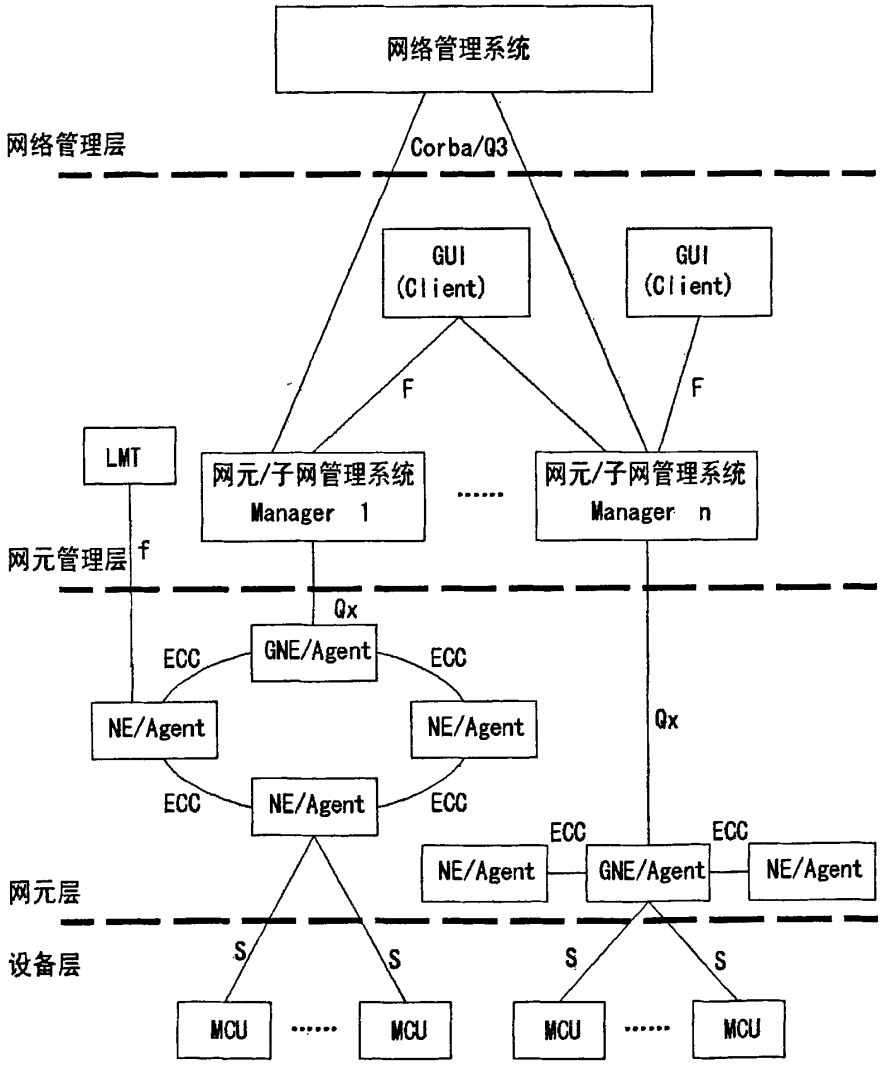


图 2-1 网管系统结构

其中，网络管理层对所辖区域内的所有网元进行管理，主要的功能包括：从全网的观点协调与控制所有网元的活动；提供修改或终止网络服务；就网络性能，可用性等事项与上层的服务管理层进行交互。

网元管理层直接行使对个别网元的管理职能，主要的功能包括：控制与协调一系列网络单元；为网络层的管理与网络单元进行通信提供协调功能；维护与网络单元有关的统计等数据。

网元层是电信网络中的网络元素，实现电信网的基本功能，是网络管理的被管理者角色，提供网络管理的原始数据。

网管接口：

网元管理层可以向上层（网络管理层）提供 Corba 或 Q3 接口。内部各层接口的定义如下：

Qx 接口：网元与网元/子网管理系统之间的接口；

S 接口：网元层与设备层之间的接口，S 口采用基于 HDLC 通讯机制进行一点对多点的通讯（NCP-MCU 通讯通路）；

F 口：界面 GUI 与 Manager 之间的接口，基于 TCP/IP；

f 口：NCP 与 LMT 之间的接口，本地网管系统中，f 接口与 Qx 接口相同（LCT 本地维护和管理终端）[10]；

网元间接口：网元与网元之间通过 ECC 协议或 TCP/IP 协议进行通讯；NE 与 NE 之间互连通过基于 DCC 通道的 Qecc；DCC 通道的 D1~D3 字节用于再生段数据通信，D4~D12 字节用于复用段数据通信；

网管代理器 Agent 在网元层上，Agent 应用程序软件应能完成配置管理、告警管理、性能管理、维护管理、安全管理等功能[11]。

Agent 主要作用：①接收网络管理层 manager 下发的指令，经过分析处理再转发到单板；保存关键配置，能够在脱离 manager 的情况下，对单板进行配置管理 ②接收单板上报命令，经过分析处理转发给网管，使单板和网管能够及时通信，保证数据的一致性[12]。

2.2 AGENT 中的功能模块

为了便于描述和区分 AGENT 中信息的流向，在 AGENT 中设置了三个功能模块，并约定了各个模块之间的层次关系，如图 2-2 所示：

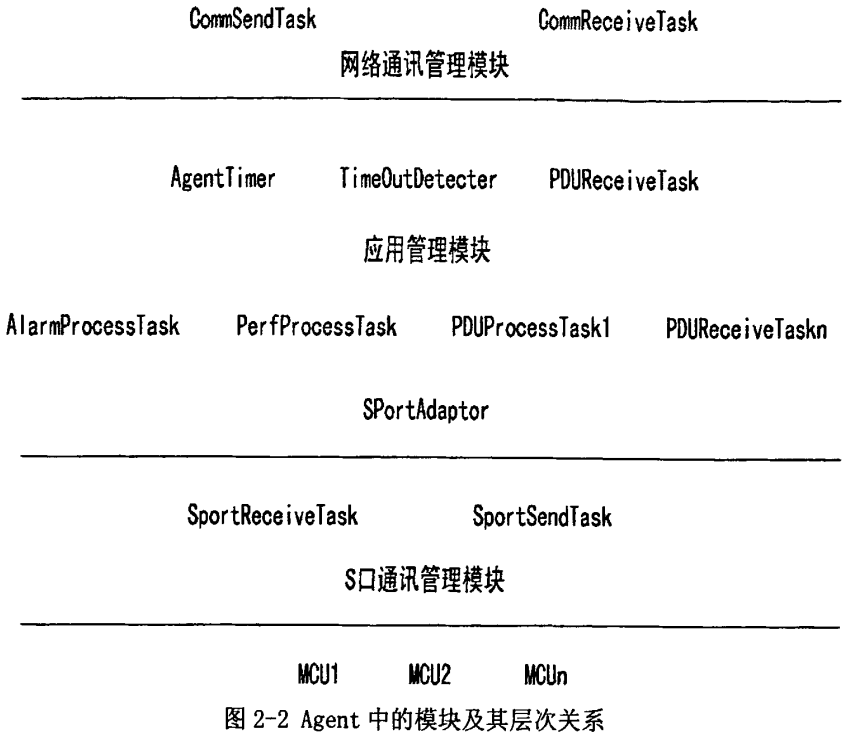


图 2-2 Agent 中的模块及其层次关系

有关三个功能模块中函数的相互调用关系和完成的功能会在以后进行阐述。

2.2.1 网络通讯管理模块

描述：主要功能是向应用管理任务提供透明的网络信息接收和发送功能。对 AGENT 中的网络通讯（含 ECC 和以太网）进行管理（包括连接管理、数据的发送和接收，提供多种通讯协议—UDP、TCP 及以后的 TP4 支持，提高程序的可扩充性、可维护性），使应用管理任务与具体的通讯协议和通讯管理无关；同时进行网络通讯状况的统计，这样在通讯状况比较差时，应用管理模块抑制 trap 的发送[13]。它包括网络通讯接收和网络通讯发送两个任务。

2.2.2 应用管理模块

描述：这是整个 AGENT 的核心，其主要功能是通过与网络通讯管理模块和 S 口通讯管理模块交互，执行 SNMP PDU 的请求命令，根据设备的当前状态和 AGENT 的运行状况，向 MANAGER 发送相关的 trap 信息，并完成私有配置数据的上载和下载操作，从而达到实现配置管理、故障管理、性能管理安全管理的目的[14]。

应用管理模块包括 PDU 接收与分发任务、应用超时检测任务、PDU 处理任务、告警管理任务、性能管理任务、S 口适配任务和定时管理任务。

2.2.3 S 口通讯管理模块

描述：它完成与 MCU 的交互功能，从而使应用管理模块独立于 S 口通讯的具体实现。它由 S 口接收任务和 S 口发送任务组成。

2.2.4 AGENT 中的功能组成

为了便于描述和区分 AGENT 中信息的流向，在 AGENT 中设置了十一个功能类，并约定了各个功能类之间的层次关系，如图 2-3 所示，各功能类之间通过消息队列进行信息的传递。其中网管通过 Qx 口与 AGENT 进行消息传递，命令到达 AGENT 后由本地调度类判定后发给告警管理、性能管理、配置管理等各个管理类，由其中的管理类处理命令后（或根据需要处理数据库）上报网管或者下发单板。单板的命令通过 S 口发给 AGENT。

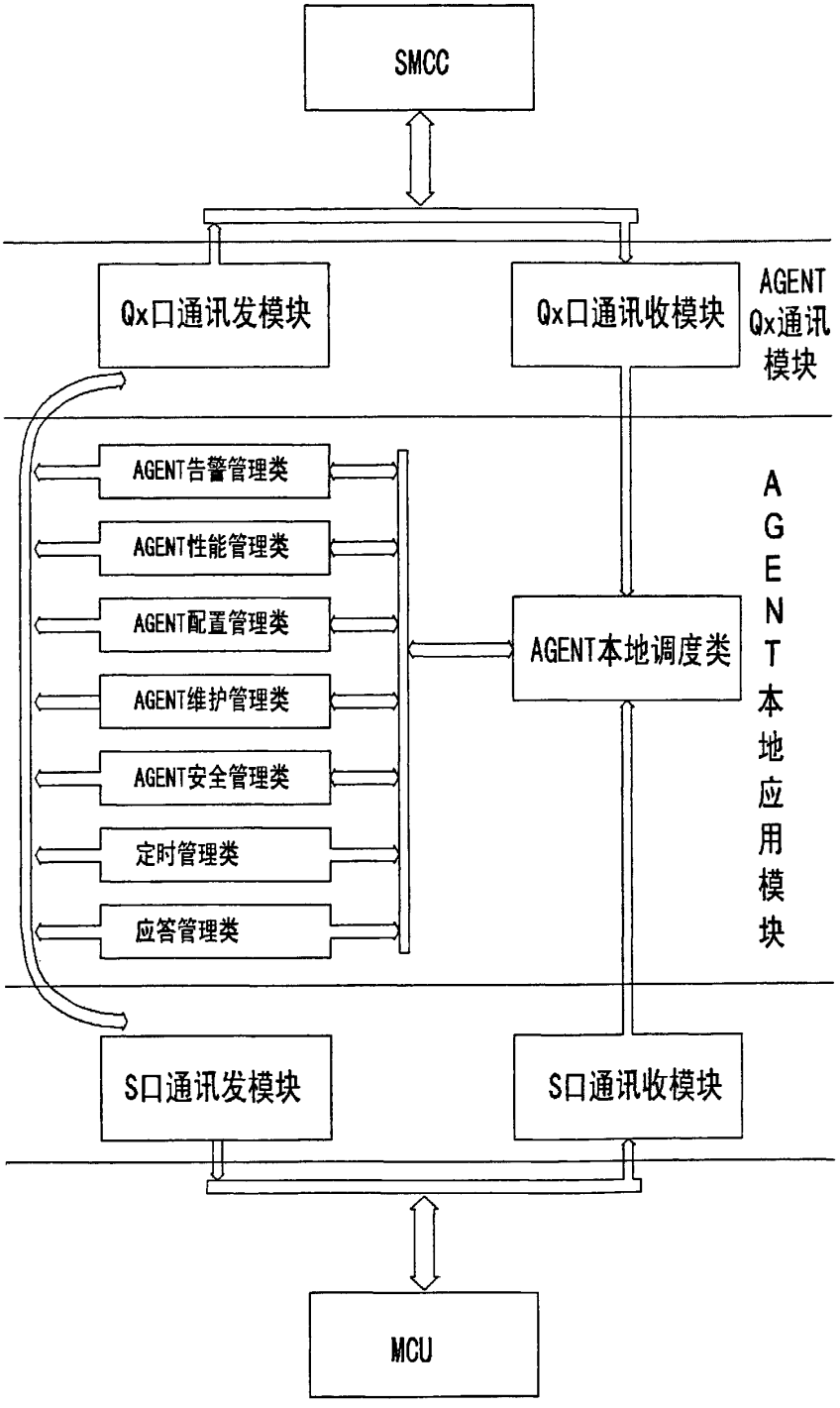


图 2-3 Agent 体系结构

2.3 AGENT 中的任务

2.3.1 任务体：root – 根任务

描述：这是整个 AGENT 应用的入口，主要是进行一些基本的初始化，包括创建和启动其它的任务、创建消息队列和信号灯、初始化 PSOS SNMP 工具包等。其中 SNMP 的初始化如图 2-4 所示：

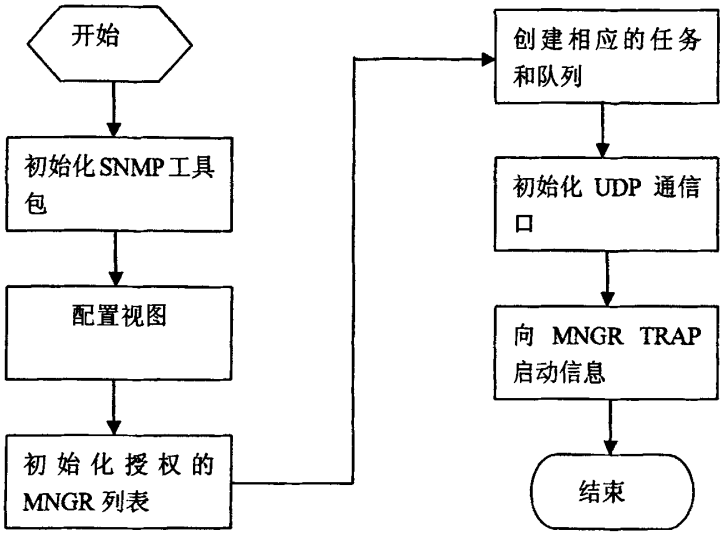


图 2-4 root 中 SNMP 初始化

2.3.2 任务体：PDUReceiveTask – PDU 接收与分发任务

描述：在 AGENT 中只创建一个这种任务。完成 SNMP PDU 的接收和分发以及上载、下载私有配置数据的处理，包括从 CommReceiveTask 接收从 MANAGER 发送来的 GetRequest、GetNextRequest、GetBulkRequest、SetRequest PDU 请求，并根据负载平衡算法将它们发送给负载最轻的 PDUProcessTask 任务；同时还进行私有上下载配置数据的处理。

2.3.3 任务体：TimeOutDetecter - 应用超时检测任务

描述：在 AGENT 中只创建一个这种任务。完成应用层异步处理超时的判别以及出现超时时的处理（包括 SNMP PDU 异步超时和私有配置数据下载的超时处理）。

2.3.4 任务体：PDUProcessTask – PDU 处理任务

描述：在 AGENT 中总共创建了 SERVER_NUM 个相同的 PDU 处理任务（此类任务的个数根据实际运行时的负载情况进行调节）。其功能是接收 PDUReceiveTask 分发来的 PDU，完成 PDU 包所请求的各种操作，包括 Get、GetNext、GetBulk 和 Set，并完成与 S 口的信息交互。对于不进行后台异步处理的 PDU，在该 PDU 全部处理完毕后产生响应 PDU，经 SnmpIoComplete 编码后发送给 CommSendTask，由 CommSendTask 完成响应 PDU 的发送工作；对于需要进行异步处理的 PDU，PDUProcessTask 完成所有的 MCU 命令的组织并发送给 SPortAdaptor，调用 RegisterAsyncEntry 完成异步处理的登记，然后挂起该 PDU。AGENT 中所有与 MANAGER 的 SNMP 信息交互都在这里完成，包括配置管理、安全管理、性能门限管理、告警严重等级管理、视图管理等。

2.3.5 任务体：AlarmProcessTask – 告警管理任务

描述：在 AGENT 中只创建一个这种任务。完成所有与告警、故障有关的控制和管理，包括从 SPortAdaptor 接收 MCU 上报的当前告警信息、向 MANAGER 发送告警发生或消失 trap、根据 trapPermitted 的设置控制 trap 的发送、实现 NE 级告警的过滤功能；完成历史告警表的维护；根据当前告警信息控制 NCP 和机架告警灯及告警响铃的控制；与 MCU 进行当前告警的一致性维护；根据获得的当前告警信息参与保护倒换、激光器关断等设备控制操作。

2.3.6 任务体：PerfProcessTask – 性能管理任务

描述：在 AGENT 中只创建一个这种任务。完成所有与性能有关的控制和管理，包括从 SPortAdaptor 接收 MCU 上报（或本任务主动查询而获得）的性能信息，刷新和维护当前和历史性能（含 15 分钟和 24 小时）库，处理性能的零抑制（主要是针对数字量）。在 AGENT 中包含一个当前 15 分钟和 HSTY_PERF_NUM 个历史 15 分钟性能寄存器以及一个当前 24 小时和一个历史 24 小时性能寄存器。

2.3.7 任务体：AgentTimer – 定时管理任务

描述：在 AGENT 中只创建一个这种任务。这是 AGENT 中具有最高优先级的任务，主要是为其它的任务提供定时功能。

2.3.8 任务体：SPortAdaptor - S 口适配任务

描述：在 AGENT 中只创建一个这种任务。它向应用管理任务屏蔽 S 口报文接收和发送的具体实现，在应用管理任务与 SPortSendTask 和 SPortReceiveTask 之间充当桥梁作用，完成应用管理任务发给 MCU 的命令和 S 口上报（或应该管理任务主动查询的响应结果）信息的转发；执行 SNMP PDU 及 MANAGER 向 AGENT 下载的私有配置数据的异步处理，并将处理的结果发送给 CommSendTask，由其发送给 MANAGER。

2.3.9 任务体：SPortReceiveTask - S 口接收任务

描述：在 AGENT 中只创建一个这种任务。接收 MCU 发送来的 S 口命令响应或主动上报数据，将其发送给 SPortAdaptor 进行分发或处理，与 SPortSendTask 一起完成 MCU 通讯中断、恢复的检测（每接收到一个 S 口数据，表示 NCP 与 MCU 的通讯是正常的，由此刷新由 SPortSendTask 维护的 MCU 通讯状况表）。

2.3.10 任务体：SPortSendTask - S 口发送任务

描述：在 AGENT 中只创建一个这种任务。接收 SPortAdaptor 任务发送来的所有 S 口命令报文（包括配置、控制、告警查询、性能查询等命令），根据任务间消息交换队列或报文中的目标地址信息，将报文发送给相应的 MCU；同时定期检测板是否在位，提供与 MCU 通讯中断、恢复的管理，将这些信息以通讯状况指示发送给应用管理任务。

2.3.11 任务体：CommReceiveTask - 网络通讯接收

描述：向应用管理模块提供透明的网络信息接收功能，并与 CommSendTask 一起完成通讯链路的管理。

2.3.12 任务体：CommSendTask - 网络通讯发送

描述：向应用管理模块提供透明的网络信息发送功能，并向应用管理模块提供发送失效指示。

2.4 AGENT 中消息队列的定义

AGENT 中定义的消息队列以及它与任务之间的关系如图 2-5 所示：

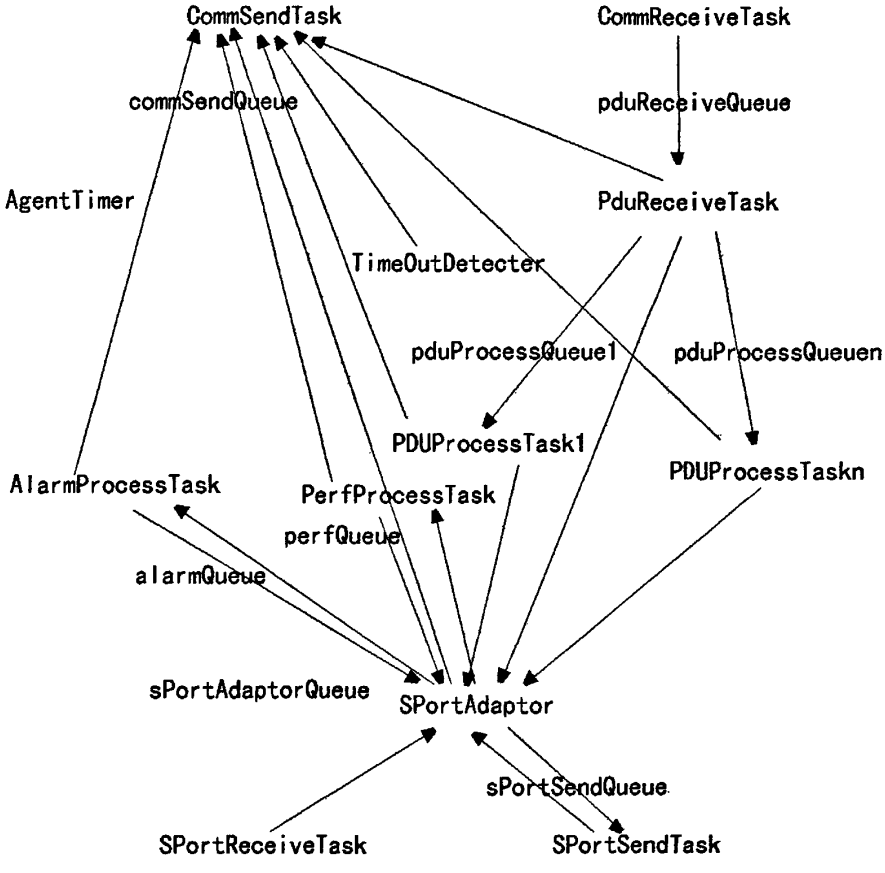


图 2-5 AGENT 中任务与队列的关系

队列 PduReceiveQueue

描述：在 AGENT 中只创建一个该消息队列。由 PDUReceiveTask 使用，接收从 MANAGER 发送来的 SNMP PDU 和上载、下载私有配置信息等命令和数据。

队列 PduProcessQueue

描述：AGENT 中共有 SERVER_NUM 个该消息队列，由 PDUProcessTask 使用，每个 PDUProcessTask 使用其中一个，pduProcessTask[x]与 pduProcessQueue[x]之间是一一对应的，具有相同的下标，用于接收从 PDUReceiveTask 发送来的 SNMP PDU。

队列 SPortAdaptorQueue

描述：在 AGENT 中只创建一个该消息队列。由 SPortAdaptor 使用，它接收：

A. PDUProcessTask 发送来的与 MCU 进行交互的信息，如配置命令、控制命令、查询命令等；

B. AlarmProcessTask 发送来的告警查询命令、以及对系统中发生的极其严重告警的反应（比如控制激光器的关断、与保护倒换有关的控制等）；

C. PerfProcessTask 发送来的性能查询命令；

D. PDUReceiveTask 发送来的 MANAGER 下载私有配置数据后的配置与控制命令；

E. SPortReceiveTask 发送来的主动上报数据和命令响应；

F. SPortSendTask 发送来的单板是否在位、MCU 通讯中断与恢复等信息。

队列 AlarmQueue

描述：在 AGENT 中只创建一个该消息队列。由 AlarmProcessTask 使用，接收从 SPortAdaptor 发送来的告警查询结果和主动上报的告警信息，包括与单板是否在位、MCU 通讯中断与恢复有关的信息。

队列 PerfQueue

描述：在 AGENT 中只创建一个该消息队列。由 PerfProcessTask 使用，接收从 SPortAdaptor 发送来的性能查询结果。对于历史 15 分钟性能，建议不采用 15 分钟时刻主动上报的方式，而是在 15 分钟时刻到来后由 NCP 主动采集。

队列 SPortSendQueue

描述：在 AGENT 中只创建一个该消息队列。由 SPortSendTask 使用，接收 SPortAdaptor 发送来的配置、控制和查询命令。

队列 CommSendQueue

描述：在 AGENT 中只创建一个该消息队列。由 CommSendTask 使用，它接收：

A. AlarmProcessTask 发送来的告警发生、消失 trap（包括 15 分钟性能越限告警、MCU 通讯中断与恢复 trap 以及特定的事件，如保护倒换、激光器关断）；

B. PerfProcessTask 发送来的与性能有关的 trap(如 24 小时性能越限告警)；

C. SPortAdaptor 发送来的 SNMP PDU 异步处理的响应、MANAGER 下载私有配置数据的处理响应（指需要与 MCU 进行交互的情况）；

D. PDUProcessTask 发送来的 SNMP PDU 处理响应（指不进行异步处理的情况）以及权限检查失败 trap 信息；

E. PDUReceiveTask 发送来的 MANAGER 上载私有配置数据的响应、MANAGER 下

载私有数据的响应（指不进行异步处理的情况）；

F. TimeOutDetector 发送来的 SNMP PDU 异步处理超时的响应及下载私有数据超时的响应（指需要进行异步处理的情况）；

2.5 AGENT 中的事件定义

AGENT 中定义的事件以及它与任务之间的关系如图 2-6 所示：

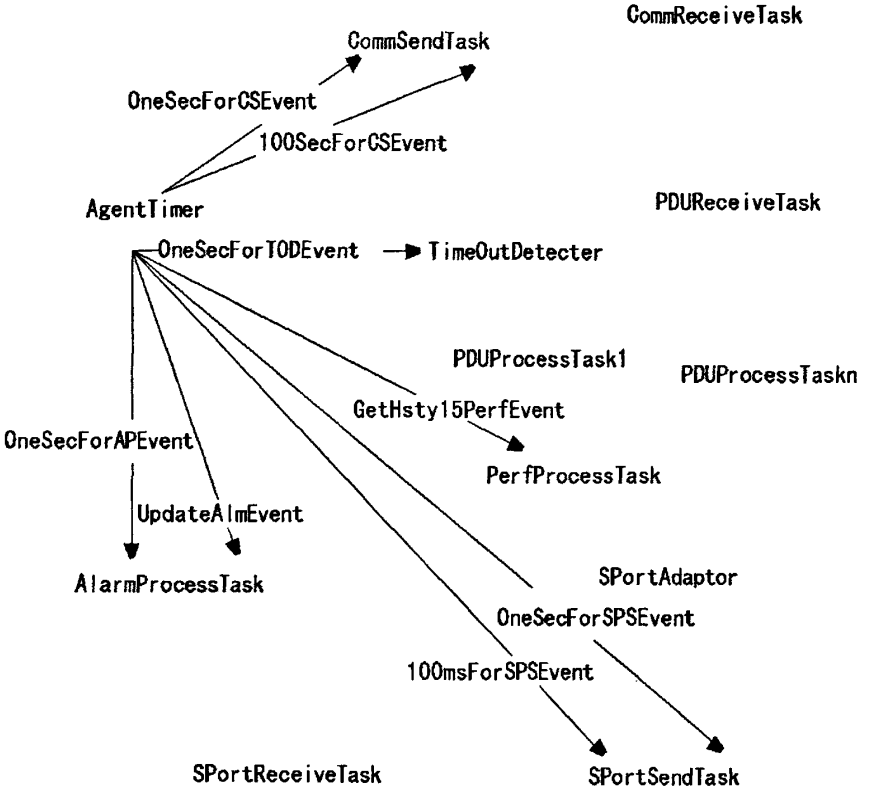


图 2-6 AGENT 中任务与事件的关系

事件 OneSecForCSEvent

描述：由 AgentTimer 向 CommSendTask 发送的“一秒定时到”事件，用于 CommSendTask 进行与网络发送（包括连接建立）超时有关的管理。该事件占用编码事件字的第 0 位（左边为最高位--第 31 位，右边为最低位--第 0 位），编码值为 0x01。

事件 100msForCSEvent

描述：由 AgentTimer 向 CommSendTask 发送的“100 毫秒定时到”事件，用于

CommSendTask 进行与网络发送（包括连接建立）的重新尝试。该事件占用编码事件字的第 1 位，编码值为 0x02。

事件 OneSecForTODEvent

描述：由 AgentTimer 向 TimeOutDetecter 发送的“一秒定时到”事件，用于 TimeOutDetecter 进行 SNMP PDU 异步处理及 MANAGER 下载私有配置数据异步处理有关的超时管理。该事件占用编码事件字的第 2 位，编码值为 0x04。

事件 OneSecForAPEvent

描述：由 AgentTimer 向 AlarmProcessTask 发送的“一秒定时到”事件，用于 AlarmProcessTask 进行告警灯（包含 NCP 和机架灯）和告警声音的控制。该事件占用编码事件字的第 3 位，编码值为 0x08。

事件 UpdateAlmEvent

描述：由 AgentTimer 向 AlarmProcessTask 发送的“刷新告警”事件，用于 AlarmProcessTask 定期从 MCU 刷新当前告警。该事件占用编码事件字的第 4 位，编码值为 0x10。

事件 GetHsty15PerfEvent

描述：由 AgentTimer 向 PerfProcessTask 发送的“15 分钟时刻到”事件，用于通知单板 15 分钟时刻到。该事件占用编码事件字的第 5 位，编码值为 0x20。

事件 OneSecForSPSEvent

描述：由 AgentTimer 向 SPortSendTask 发送的“一秒定时到”事件，用于 SPortSendTask 进行 S 口命令发送的超时管理。该事件占用编码事件字的第 6 位，编码值为 0x40。

事件 100msForSPSEvent

描述：由 AgentTimer 向 SPortSendTask 发送的“100 毫秒定时到”事件，用于 SPortSendTask 进行 S 口命令发送重新尝试。该事件占用编码事件字的第 7 位，编码值为 0x80。

第三章 SNMP 协议简介

3.1 SNMP 协议简介

3.1.1 SNMP 的配置

图 3-1 是使用 SNMP 的典型配置。整个系统必须有一个管理站（management station），它实际上是网控中心。在管理站内运行管理进程。在每个被管对象中一定要有代理进程。管理进程和代理继承利用 SNMP 报文进行通信，而 SNMP 报文又使用 UDP 来传送。图中有两个主机和一个路由器。这些协议栈中带有阴影的部分是原来这些主机和路由器所具有的，而没有阴影的部分是为实现网络管理而增加的。

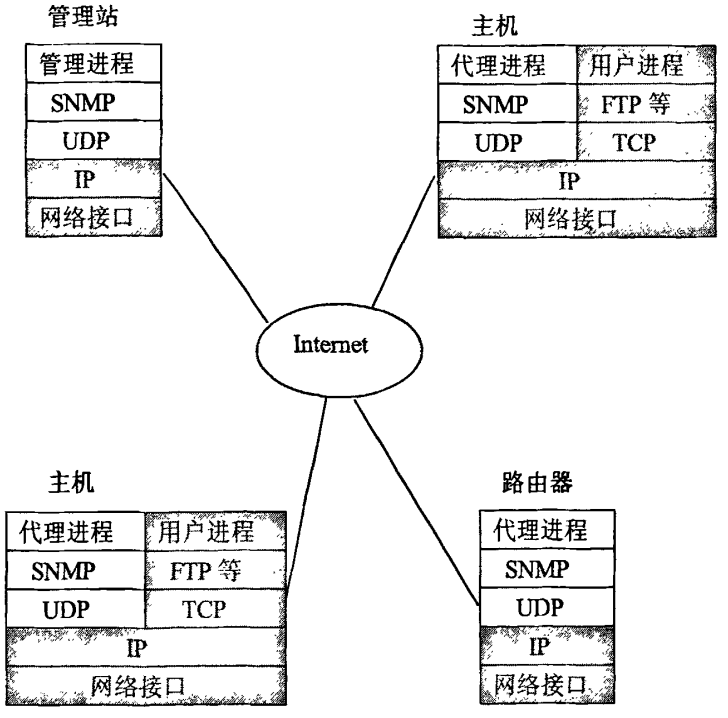


图 3-1 SNMP 配置图

有时网络管理协议无法控制某些网络元素，例如该网络元素使用的是另一种网络管理协议。这是可使用委托代理（proxy agent）。委托代理能提供如协议转换和过滤操作的汇集功能[15]。然后委托代理来对管理对象进行管理。图 3-2 表示委托管理的配置情况。

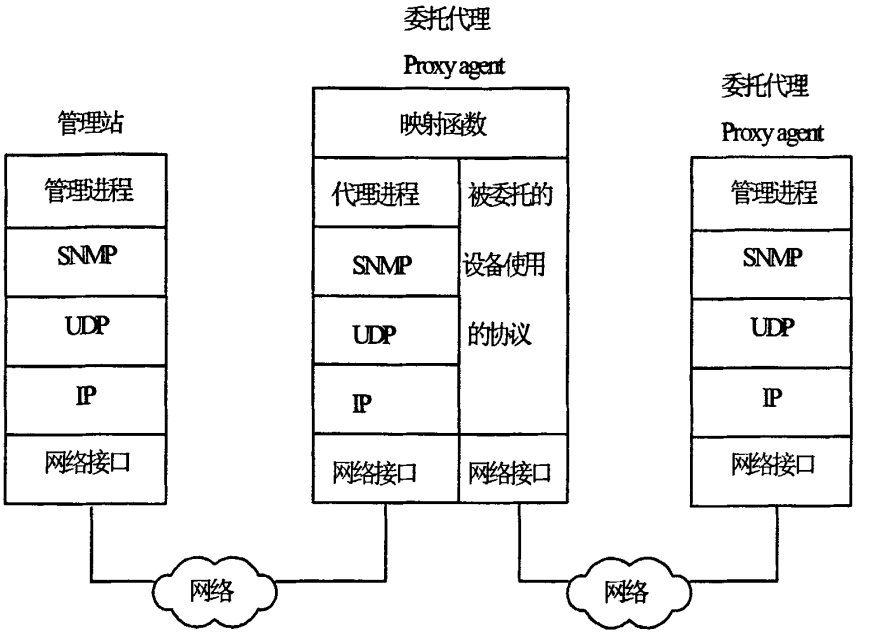


图 3-2 委托管理的配置

SNMP 的网络管理由三部分组成，即管理信息库 MIB、管理信息结构 SMI 以及 SNMP 本身。下面简要介绍。

3.1.2 管理信息库 MIB

管理信息库 MIB 指明了网络元素所维持的变量（即能够被管理进程查询和设置的信息）。MIB 给出了一个网络中所有可能的被管理对象的集合的数据结构。SNMP 的管理信息库采用和域名系统 DNS 相似的树型结构，它的根在最上面，根没有名字[16]。图 3-3 画的是管理信息库的一部分，它又称为对象命名（object naming tree）。

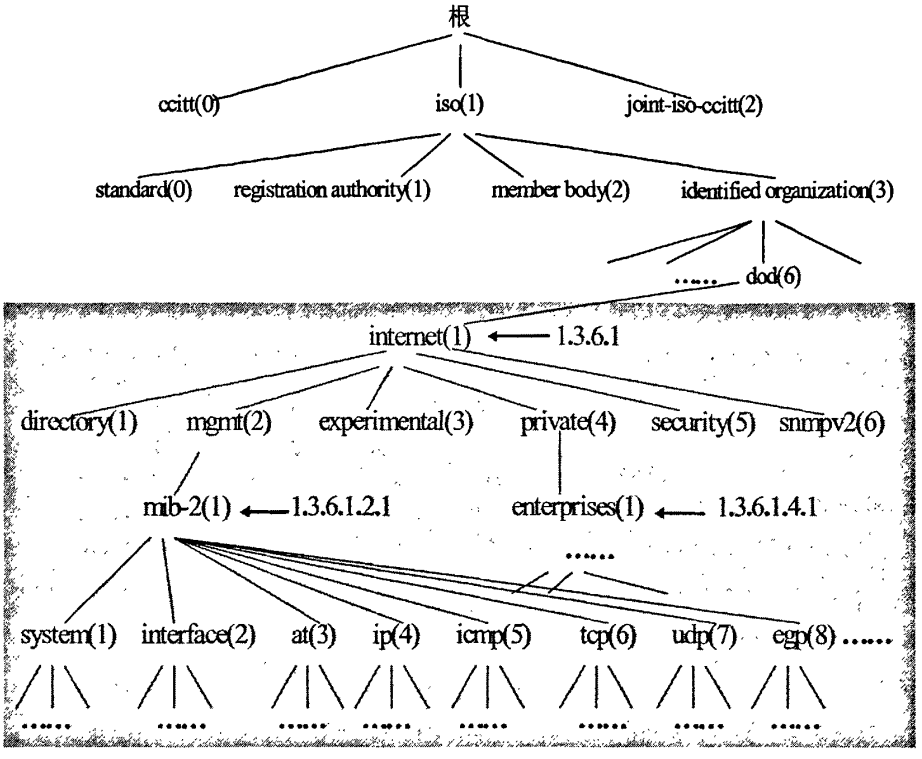


图 3-3 管理信息库的对象命名举例

对象命名树的顶级对象有三个，即 ISO、ITU-T 和这两个组织的联合体。在 ISO 的下面有 4 个结点，其中的一个（标号 3）是被标识的组织。在其下面有一个美国国防部（Department of Defense）的子树（标号是 6），再下面就是 Internet（标号是 1）。在只讨论 Internet 中的对象时，可只画出 Internet 以下的子树（图中带阴影的虚线方框），并在 Internet 结点旁边标注上{1.3.6.1}即可。

在 Internet 结点下面的第二个结点是 mgmt（管理），标号是 2。再下面是管理信息库，原先的结点名是 mib。1991 年定义了新的版本 MIB-II，故结点名现改为 mib-2，其标识为{1.3.6.1.2.1}，或{Internet(1).2.1}。这种标识为对象标识符。

最初的结点 mib 将其所管理的信息分为 8 个类别，见表 3-1。现在的 mib-2 所包含的信息类别已超过 40 个[17]。

表 3-1 最初的结点 mib 管理的信息类别

类别	标号	所包含的信息
System	(1)	主机或路由器的操作系统
interfaces	(2)	各种网络接口及它们的测定通信量
address translation	(3)	地址转换（例如 ARP 映射）
ip	(4)	Internet 软件（IP 分组统计）
icmp	(5)	ICMP 软件（已收到 ICMP 消息的统计）
tcp	(6)	TCP 软件（算法、参数和统计）
udp	(7)	UDP 软件（UDP 通信量统计）
egp	(8)	EGP 软件（外部网关协议通信量统计）

应当指出，MIB 的定义与具体的网络管理协议无关，这对于厂商和用户都有利。厂商可以在产品（如路由器）中包含 SNMP 代理软件，并保证在定义新的 MIB 项目后该软件仍遵守标准。用户可以使用同一网络管理客户软件来管理具有不同版本的 MIB 的多个路由器。当然，一个没有新的 MIB 项目的路由器不能提供这些项目的信息。

这里要提一下 MIB 中的对象{1.3.6.1.4.1}，即 enterprises（企业），其所属结点数已超过 3000。例如 IBM 为{1.3.6.1.4.1.2}，Cisco 为{1.3.6.1.4.1.9}，Novell 为{1.3.6.1.4.1.23}等。世界上任何一个公司、学校只要用电子邮件发往 iana-mib@isi.edu 进行申请即可获得一个结点名。这样各厂家就可以定义自己的产品的被管理对象名，使它能用 SNMP 进行管理。

3.1.3 SNMP 的 5 种协议数据单元

SNMP 规定了 5 种协议数据单元 PDU（也就是 SNMP 报文），用来在管理进程和代理之间的交换。

- get-request 操作：从代理进程处提取一个或多个参数值。
- get-next-request 操作：从代理进程处提取紧跟当前参数值的下一个参数值。
- set-request 操作：设置代理进程的一个或多个参数值。

- get-response 操作：返回的一个或多个参数值。这个操作是由代理进程发出的，它是前面三种操作的响应操作。
- trap 操作：代理进程主动发出的报文，通知管理进程有某些事情发生[18]。

前面的 3 种操作是由管理进程向代理进程发出的，后面的 2 个操作是代理进程发给管理进程的，为了简化起见，前面 3 个操作今后叫做 get、get-next 和 set 操作。图 3-4 描述了 SNMP 的这 5 种报文操作。请注意，在代理进程端是用熟知端口 161 俩接收 get 或 set 报文，而在管理进程端是用熟知端口 162 来接收 trap 报文[19]。

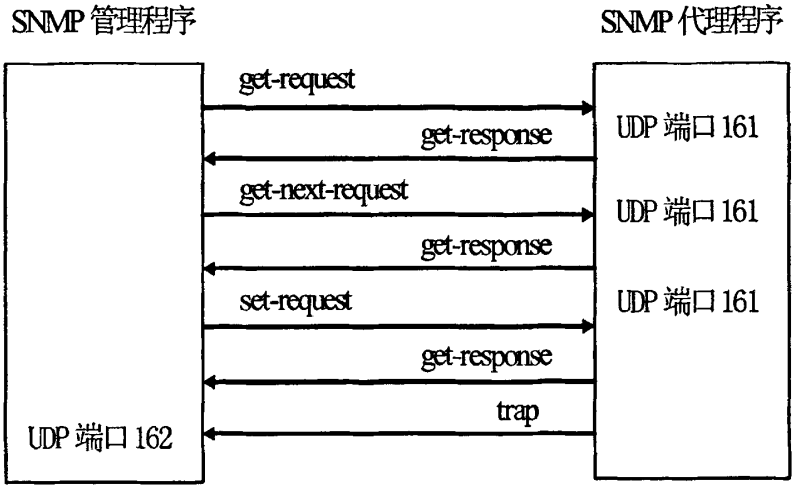


图 3-4 SNMP 的 5 种报文操作

图 3-5 是封装成 UDP 数据报的 5 种操作的 SNMP 报文格式。可见一个 SNMP 报文共有三个部分组成，即公共 SNMP 首部、get/set 首部 trap 首部、变量绑定。

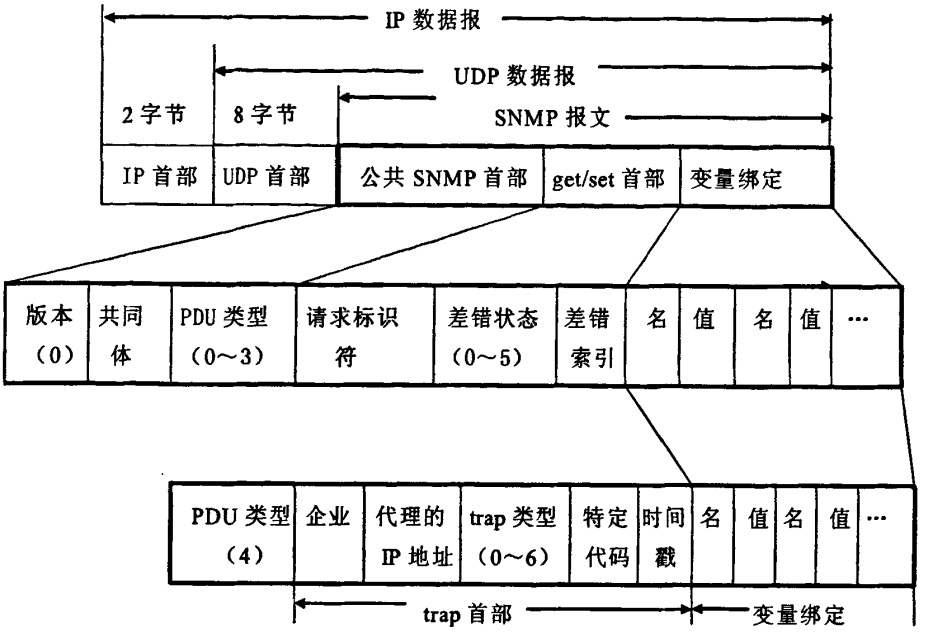


图 3-5 SNMP 报文格式

（1）公共 SNMP 首部

共三个字段：

- 版本

写入版本字段的是版本号减 1，对于 SNMP（即 SNMPV1）则应写入 0。

- 共同体（community）

共同体就是一个字符串，作为管理进程和代理进程之间的明文口令，常用的是 6 个字符“public”。

- PDU 类型

根据 PDU 的类型，填入 0～4 中的一个数字，其对应关系如表 3-2 所示意图。

表 3-2 PDU 类型

PDU 类型	名称
0	get-request
1	get-next-request
2	get-response
3	set-request
4	trap

（2）get/set 首部

- 请求标识符(request ID)

这是由管理进程设置的一个整数值。代理进程在发送 get-response 报文时也要返回此请求标识符。管理进程可同时向许多代理发出 get 报文，这些报文都使用 UDP 传送，先发送的有可能后到达。设置了请求标识符可使管理进程能够识别返回的响应报文对于哪一个请求报文。

- 差错状态（error status）

由代理进程回答时填入 0～5 中的一个数字，见表 3-3 的描述。

表 3-3 差错状态描述

差错状态	名字	说明
0	noError	一切正常
1	tooBig	代理无法将回答装入到一个 SNMP 报文之中
2	noSuchName	操作指明了一个不存在的变量
3	badValue	一个 set 操作指明了一个无效值或无效语法
4	readOnly	管理进程试图修改一个只读变量
5	genErr	某些其他的差错

- 差错索引(error index)

当出现 noSuchName、badValue 或 readOnly 的差错时，由代理进程在回答时设置的一个整数，它指明有差错的变量在变量列表中的偏移。

（3）trap 首部

- 企业（enterprise）

填入 trap 报文的网络设备的对象标识符。此对象标识符肯定是在图 3 的对象命名树上的 enterprise 结点{1.3.6.1.4.1}下面的一棵子树上。

- trap 类型

此字段正式的名称是 generic-trap，共分为表 3-4 中的 7 种。

表 3-4 trap 类型描述

trap 类型	名字	说明
0	coldStart	代理进行了初始化
1	warmStart	代理进行了重新初始化
2	linkDown	一个接口从工作状态变为故障状态
3	linkUp	一个接口从故障状态变为工作状态
4	authenticationFailure	从 SNMP 管理进程接收到具有一个无效共同体的报文
5	egpNeighborLoss	一个 EGP 相邻路由器变为故障状态
6	enterpriseSpecific	代理自定义的事件，需要用后面的“特定代码”来指明

当使用上述类型 2、3、5 时，在报文后面变量部分的第一个变量应标识响应的接口。

- 特定代码(specific-code)

指明代理自定义的事件编号（若 trap 类型为 6），否则为 0。

- 时间戳(timestamp)

指明自代理进程初始化到 trap 报告的事件发生所经历的时间，单位为 10ms。例如时间戳为 1908 表明在代理初始化后 1908ms 发生了该事件。

（4）变量绑定(variable-bindings)

指明一个或多个变量的名和对应的值。在 get 或 get-next 报文中，变量的值应忽略[20]。

3.1.4 管理信息结构 SMI

SNMP 中，数据类型并不多。这里我们就讨论这些数据类型，而不关心这些数据类型在实际中是如何编码的。

- INTEGER

一个变量虽然定义为整型，但也有多种形式。有些整型变量没有范围限制，有些整型变量定义为特定的数值（例如，IP 的转发标志就只有允许转发时的或者不允许转发时的这两种），有些整型变量定义一个特定的范围（例如，UDP 和 TCP 的端口号就从 0 到 65535）。

- OCTER STRING

0 或多个 8 bit 字节，每个字节值在 0～255 之间。对于这种数据类型和下一种数据类型的 BER 编码，字符串的字节个数要超过字符串本身的长度。这些字符串不是以 NULL 结尾的字符串。

- DisplayString

0或多个8bit字节，但是每个字节必须是ASCII码。在MIB-II中，所有该类型的变量不能超过255个字符（0个字符是可以的）。

- OBJECT IDENTIFIER
- NULL

代表相关的变量没有值。例如，在get或get-next操作中，变量的值就是NULL，因为这些值还有待到代理进程处去取。

- IpAddress

4字节长度的OCTER STRING，以网络序表示的IP地址。每个字节代表IP地址的一个字段。

- PhysAddress

OCTER STRING类型，代表物理地址（例如以太网物理地址为6个字节长度）。

- Counter

非负的整数，可从0递增到232—1（4294976295）。达到最大值后归0。

- Gauge

非负的整数，取值范围为从0到4294976295(或增或减)。达到最大值后锁定直到复位。例如，MIB中的tcpCurrEstab就是这种类型的变量的一个例子，它代表目前在ESTABLISHED或CLOSE_WAIT状态的TCP连接数。

- TimeTicks

时间计数器，以0.01秒为单位递增，但是不同的变量可以有不同的递增幅度。所以在定义这种类型的变量的时候，必须指定递增幅度。例如，MIB中的sysUpTime变量就是这种类型的变量，代表代理进程从启动开始的时间长度，以多少个百分之一秒的数目来表示。

- SEQUENCE

这一数据类型与C程序设计语言中的“structure”类似。一个SEQUENCE包括0个或多个元素，每一个元素又是另一个ASN.1数据类型。例如，MIB中的UdpEntry就是这种类型的变量。它代表在代理进程侧目前“激活”的UDP数量（“激活”表示目前被应用程序所用）。在这个变量中包含两个元素：

- IpAddress类型中的udpLocalAddress，表示IP地址。
- INTEGER类型中的udpLocalPort，从0到65535，表示端口号。

● SEQUENDEOF

这是一个向量的定义，其所有元素具有相同的类型。如果每一个元素都具有简单的数据类型，例如是整数类型，那么我们就得到一个简单的向量（一个一维向量）。但是我们将看到，SNMP在使用这个数据类型时，其向量中的每一个元素是一个SEQUENCE（结构）。因而可以将它看成为一个二维数组或表。

3.2 SNMPv2 协议

简单性是 SNMP 标准取得成功的主要原因。因为在大型的、多厂商产品构成的复杂网络中,管理协议的明晰是至关重要的，但同时这又是 SNMP 的缺陷所在——为了使协议简单易行,SNMP 简化了不少功能，如：

● 没有提供成批存取机制，对大块数据进行存取效率很低；

● 没有提供足够的安全机制，安全性很差；

● 只在 TCP/IP 协议上运行，不支持别的网络协议；

● 没有提供 manager 与 manager 之间通信的机制，只适合集中式管理，而不利于进行分布式管理；

● 只适于监测网络设备,不适于监测网络本身[21]。

针对这些问题,对它的改进工作一直在进行。如 1991 年 11 月,推出了 RMON(Remote Network Monitoring)MIB,加强 SNMP 对网络本身的管理能力。它使得 SNMP 不仅可管理网络设备，还能收集局域网和互联网上的数据流量等信息。1992 年 7 月,针对 SNMP 缺乏安全性的弱点,又公布了 S-SNMP(Secure SNMP)草案。

到 1993 年初,又推出了 SNMP Version2 即 SNMPv2(推出了 SNMPv2 以后,SNMP 就被称为 SNMPv1)。SNMPv2 包容了以前对 SNMP 所做的各项改进工作,并在保持了 SNMP 清晰性和易于实现 的特点以外,功能更强,安全性更好[22]。具体表现为：

● 提供了验证机制、加密机制、时间同步机制等，安全性大大提高，

● 提供了一次取回大量数据的能力，效率大大提高；

● 增加了 manager 和 manager 之间的信息交换机制,从而支持分布式管理结构。由中间(intermediate)manager 来分担主 manager 的任务,增加了远地站点的局部自主性。

● 可在多种网络协议上运行,如 OSI、Appletalk 和 IPX 等,适用多协议网络

环境(但它的缺省网络协议仍是UDP)。

根据Carnegie-Mellon大学(SNMPv2标准的制定者之一)的Steven Waldbusser测试结果，SNMPv2的处理能力明显强于SNMPv1,大约是SNMPv1的15倍[23]。

SNMPv2一共由12份协议文本组成(RFC1441-RFC1452),已被作为Internet的推荐标准予以公布[24]。

可看出它支持分布式管理。一些站点可以既充当manager又充当agent,同时扮演两个角色。作为agent,它们接受更高一级管理站的请求命令，这些请求命令中一部分与agent本地的数据有关,这时直接应答即可;另一部分则与远地agent上的数据有关[25]。这时agent就以manager的身份向远地agent请求数据,再将应答传给更高一级的管理站。在后一种情况下,它们起的是proxy(代理)的作用。

下面将SNMPv2标准加以详细介绍,包括SNMPv2标准的中的安全机制,SNMPv2标准中的Party实体,以及如何从通信协议操作、SMI、MIB三方面来看SNMPv2标准。

3.2.1 SNMPv2标准中的安全机制

SNMPv2对SNMPv1的一个大的改进,就是增强了安全机制。对管理系统安全的威胁主要有下面几种:

(1)信息篡改(modification)

SNMPv2标准中,允许管理站(manager)修改agent上的一些被管理对象的值。破坏者可能会将传输中的报文加以改变,改成非法值,进行破坏。因此,协议应该能够验证收到的报文是否在传输过程中被修改过。

(2)冒充(masquerade)

SNMPv2标准中虽然有访问控制能力,但这主要是从报文的发送者来判断的。那些没有访问权的用户可能会冒充别的合法用户进行破坏活动。因此，协议应该能够验证报文发送者的真实性,判断是否有人冒充。

(3)报文流的改变(message stream modification)

由于SNMPv2标准是基于无连接传输服务的,报文的延迟、重发以及报文流顺序的改变都是可能发生的。某些破坏者可能会故意将报文延迟、重发,或改变报文流的顺序,以达到破坏的目的。因此,协议应该能够防止报文的传输时间过长,以给破坏者留下机会。

(4)报文内容的窃取(disclosure)

破坏者可能会截获传输中的报文,窃取它的内容。特别在创建新的 SNMPv2 Party 时,必须保证它的内容不被窃取,因为以后关于这个 Party 的所有操作都依赖于它。因此,协议应该能够对报文的内容进行加密,保证它不被窃听者获取[26]。

针对上述安全性问题,SNMPv2 中增加了验证(Authentication)机制,加密(Privacy)机制,以及时间同步机制来保证通信的安全。

3.2.2 SNMPv2 协议操作

SNMPv2 标准的核心就是通信协议——它是一个请求/应答式的协议。

这个协议提供了在 manager 与 agent、manager 与 manager 之间交换管理信息的直观、基本的方法。

每条 SNMPv2 的报文都由一些域构成:

如果发送方、接收方的两个 Party 都采用了验证(authentication)机制,它就包含与验证有关的信息;否则它为空(取 NULL)。验证的过程如下:发送方和接收方的 Party 都分别有一个验证用的密钥(secret key)和一个验证用的算法。报文发送前,发送方先将密钥值填入 digest 域,作为报文的前缀。然后根据验证算法,对报文中 digest 域以后(包括 digest 域)的报文数据进行计算,计算出一个摘要值(digest),再用摘要值取代密钥,填入报文中的 digest 域。接收方收到报文后,先将报文中的摘要值取出来,暂存在一个位置,然后用发送方的密钥放入报文中的 digest。将这两个摘要值进行比较,如果一样,就证明发送方确实是 srcParty 域中所指明的那个 Party,报文是合法的;如果不一样,接收方断定发送方非法。验证机制可以防止非法用户"冒充"某个合法 Party 来进行破坏。

authInfo 域中还包含两个时间戳(time stamp),用于发送方与接收方之间的同步,以防止报文被截获和重发。

SNMPv2 的另一大改进是可以对通信报文进行加密,以防止监听者窃取报文内容。除了 privDst 域外,报文的其余部分可以被加密。发送方与接收方采用同样的加密算法(如 DES)。

通信报文可以不加任何安全保护,或只进行验证,也可以二者都进行。

第四章 SNMP 在 AGENT 中的应用

4.1 SNMP 协议设计思想

基于 SNMP 网络管理协议，实现 OADM AGENT 系统，完成 OADM 设备的配置管理、故障管理、性能管理、安全管理。

OADM 项目模型与以前的私有网管或者 Q3 网管比较起来基本相同，从上到下依次是 GUI，MANAGER，AGENT，单板。GUI 与 MANAGER 交互信息采用私有定义的 F 口报文格式，AGENT 与单板之间的信息交互采用私有定义的 S 口报文格式，两种报文格式与 II 型机等其他项目采用的基本相同。按照报文头和报文体划分，报文头的每个字节有具体含义，报文体是一重或者二重循环的结构体形式。

MANAGER 的数据库采用 SYBASE，AGENT 数据库采用我们私有的 1.0 数据库。

与其他项目唯一不同的是 MANAGER 和 AGETN 之间的通信协议不是以前的私有协议，而是采用 SNMP 统一的网络管理协议。仅仅由于这个协议的采用，我们的网管就可以管理其他厂家的同类设备，我们的设备也可以接受其他厂家的管理。当然前提是其他厂家也采用了 SNMP 协议，并且双方的 MIB 定义是相同的。

在与 SNMP 工具包的交互中，我们全部采用 SNMP 工具包所提供的异步处理方式，这样在我们的代码实现中处理起来会非常灵活。至于在与具体的对象或设备进行交互时是否使用异步，可以根据需要而定，比如当 MANAGER 向 AGENT 获取当前性能时，为了保证实时性，NCP 必须要到单板上去取，这种情况下最好采用异步方式，因为如果采用同步方式，则处理这个 SNMP PDU 的任务在单板返回性能信息之前必须等待，不能处理其它的 PDU，这样效率就比较低。如果使用异步方式，在异步处理完毕后，程序代码必须调用 SNMP_Continue()通知工具包继续对此 PDU 进行处理。对于其它信息的处理，可以直接从 NCP 中获取，因而没有必要采用异步方式。同时设计了异步处理注册队列数据结构来支持。

AGENT 中所有的任务都是在系统初始化时创建，而不是每收到一条消息创建一个任务、待任务完成后再删除该任务，这样做系统开销会比较大，而且处理时问题也比较多。为此设定 AgentTimer（AGENT 中的定时管理任务）与别的任务之间采用事件通讯方式，其它任务之间全部采用消息队列的方式进行通讯。

对于 MANAGER 从 AGENT 中取历史性能、历史告警的处理，这种操作一般都会

涉及到大量的数据传输，由于当前 UDP 实现的限制，处理大批量数据有许多不方便，因此考虑当 MANAGER 每次发送这条命令时，AGENT 只将最新的数据发给 MANAGER，老的数据则不进行传输，即只将 AGENT 在同一 MANAGER 两次发送这条命令之间新产生的数据发送给 MANAGER，这样可以大大缓解 ECC 信道的通讯状况，也可以减轻 MANAGER 处理这条命令的工作量。在讨论中，大家一致认为这部分工作由 MANAGER 来保证，AGENT 中不予考虑。

当前在 SDH 1.3 网管系统中出现了 S 口通讯紧张的现象，在 OADM AGENT 中，S 口通讯的信息量没有 SDH 中那么大，但是在每个 15 分钟结束时仍然可能出现丢数据的现象，而且当前 NCP 与 MCU 通讯时缺乏一种确认机制（特别是对于 MCU 主动上报的情况），因而可能出现频繁丢告警和性能的情况。为了减少这种情况的发生，在 15 分钟周期结束、NCP 通知 MCU“15 分钟时刻到”时，要求所有的 MCU 不主动上报历史性能，而是由 NCP 主动采集，从而缓解这一时刻 S 口的通讯状况。

当前 LMT 的实现，采用的是 UDP 和 SNMP，而不考虑采用串口和私有协议的方式，因此在 OADM AGENT 的实现中不需要对此提供特殊的支持。

对于历史性能、当前告警、历史告警信息，由于在实现时不需要保存到 FLASH 中，因此没有必要使用数据库，而是定义数据结构和链表，直接在内存中进行操作，这样可以提高系统的运行效率，操作起来也非常方便。因而在实现时需要定义相应的数据结构，当然对于历史告警，因为其索引是一个流水号，因此在结构中也要有这个东西。对于日志信息，由于它记录了 AGENT 中所发生的重要事件和相关的出错信息，需要保存在永久存储设备上，因此采用数据库来进行存储。

4.1.1 网络通讯管理机制

由于 SNMP 中采用的是 udp 通讯方式，通讯的可靠性得不到保证，在此基础上进行数据的可靠传输可能需要应用层做许多工作，而且可能得不偿失，最终还是达不到预期的效果。因此我们在 MIB 中添加 NE 和管理者链表对象（也可以在现有 MIB 中 system Group 的基础上扩充一些信息），在 NE 和 MANAGER 信息中都包含厂家信息。对于 MANAGER 和 NE 都是中兴通讯的情况下，采用 tcp（在不要求与其它厂家进行带内 ECC 互通的情况下，甚至可以采用 TP4，因为 TP4 提供一套标准的 ECC/以太网通讯协议，具有自动路由的功能，因而可以改善通讯的效果，提高系统的可用性、易用性和代码的质量，减少代码开发的工作量）方式通讯（这种情况下可以根据链路的状况判别通讯线路的好坏，并可以判别与 MANAGER 的连接状

况，从而可以优化系统的管理，比如在通讯状况不佳的情况下，减少 trap 的发送或发送尝试，或者只发送严重的 trap 信息，而将次要的 trap 进行抑制等），这样可以保证管理的可靠性；其余的情况（指 MANAGER 和 NE 有一个不是中兴通讯的）采用 udp 方式进行通讯（某些特定的情况也采用 UDP 方式，比如在初始条件下 NE 中没有配置管理者链表时）。与之相关设计了通讯地址结构用于不同的情况。

对于 pSOS SNMP 开发工具包来说，PDU 包（含 trap）的接收和发送全部都需要开发者自己进行处理，因而为上述的实现途径提供了可能。我们在具体实现时，在 tcp 和 udp 之上实现一个通讯管理层，使 AGENT 的应用管理功能与具体的网络通讯实现分离，这样系统的结构更加清晰，同时也提高了代码的可读性和可扩充性。通讯管理层的主要功能：接收从应用管理任务来的发送请求，根据请求中的目的地址，以及相应的厂家信息，确定是采用 tcp 还是 udp 通讯方式（也可能是 TP4），如果是 udp，则直接发送；如果是采用 tcp 方式，判别与目的地址的连接是否存在，如果存在则利用现有的链路，如果不存在，则新建一条链路，然后发送，如果发送不成功可以进行一定的尝试，这些操作对应用管理层都是透明的，当然要将发送不成功的信息反馈给应用管理层；同时监听是否有网络发送来的信息，如果有则接收，并将其发送给相应的应用管理任务。

对于 TCP 来说，由于在实现时有 Client 和 Server 之分，如果允许 AGENT 主动建链，MANAGER 中必须实现 Server 部分的代码，因而增加了工作量和实现的难度，因此在讨论中大家认为 AGENT 中暂时不实现主动建链的功能。

LMT 与 AGENT 之间的通讯，只采用 UDP 协议，而不考虑 TCP 和 TP4 协议。

4.1.2 内存使用机制

在 SNMP 应用中，大量使用的是 integer、oid 等小块内存空间，如果采用 pSOS 中现有的内存分配方式，这样就会出现即使当前只使用 4 个字节的空间，也要为它分配 128 或 256 甚至更大的内存空间（具体每块内存是多大与 sys_conf.h 中相关设置有关），这样就会造成极大的浪费，而且使用起来速度也会比较慢，因此可以考虑将 DSET 使用的内存分配机制借鉴过来（Vertel 中也使用类似的机制），在系统初始化时预先分配很多不同大小的内存块（如 8 字节、16 字节、32 字节、256 字节.....），在需要时直接从这些内存块中获取（每次获取一个最接近所需大小的内存块，比如程序中需要一个 6 字节的缓冲区，这时可以从 8 字节内存池中分配一个），使用完毕后再还原到相应的内存队列中。对于小于 256 字节的内存分配，

采用的是内存分区（partition）方式，而大于等于 256 字节的内存分配，则采用内存分块（region）的方式。如果所需分配的内存大小大于内存分配表中的最大值或者内存分配池不能满足应用需要时，直接使用 PSOS 的内存分配机制。

为了达到上述目的，我们修改了 SNMP 工具包的 SNMP_memory_alloc()和 SNMP_memory_free()函数，使其按上述的方式分配和释放内存空间。当然程序的其它地方必须全部使用这两个函数来进行内存的分配和释放。

在应用程序中我们不处理内存申请失败的情况，即在应用代码中不做内存申请失败的判别，而是在内存分配函数中进行尝试，如果尝试后还是无法分配空间，则进入异常处理函数 ExceptionProcess()，这样能够减少很多代码量。

AGENT 中内存使用原则：谁产生数据，则由谁申请；谁使用，则由谁释放。

在 AGENT 中，有许多状态信息是不需要存储在永久存储介质上的，比如当前 AGENT 中是否有可靠的系统时间这个状态量（用 haveRightTime 表示），如果 MANAGER 对 AGENT 进行了校时，只要 NCP 没有断电，则 NCP 上的时间应该是可靠的（当然为了保证 MANAGER 与 AGENT 之间时间的一致性，MANAGER 应定期向 AGENT 校时），而不管 NCP 是否被复位过；而且只有在 NCP 具有可靠的系统时间时，AGENT 才进行正常的性能、故障和日志记录，否则可能会带来许多不必要的麻烦。为了实现这种功能，我们从内存中开辟一片空间（使用固定的内存地址空间），这片空间不由操作系统进行管理，直接以内存地址和字节流的方式进行读写，我们称其为非操作系统控制区，在这片空间中保存 haveRightTime、freezeFault、freezePerfm 等信息。当然从这片空间读取数据时，必须进行校验和数据的合法性检查。

4.1.3 MIB 中接口命名

为了使接口更具有通用性，满足管理多种设备、多种技术、不同层次设备和网管的要求，本接口的定义采用了通用的对象管理信息模型，同时，本接口吸纳了国际标准中的接口扩展技术，为使得这些技术得以在 SNMP 环境下实施，本接口定义了本规则，用于约定信息的传递格式和方法。

本接口的对象名称和参数列表都采用名值对的方法传递，即对每一个对象名称或参数项，定义其名称和对应的值，形成名称－值对，用来表示一个对象的名称或参数项值，如：

```
NameAndStringValue-T ::= SEQUENCE{
name GraphicString
,value GraphicString
}
```

例子：{"slot","7"} 的名值对表示 7 号槽位。但是在实际网管管理的对象中，要定位更为准确，仅仅一个名值对是不够的。因此我们采用了多层名值对，就像写住宅地址那样，从大到小一个一个的写，逐渐缩小范围直到你要指定的那个对象为止。这样需要名值对列表来定义：

```
NVSList-T ::= SEQUENCE OF NameAndStringValue-T
NamingAttributes-T ::= NVSList-T
```

例子：{{"ME","1"},{"bay","1"},{"shelf","2"},{"slot","3"}} 表示一个具体网元下一个具体的槽位。也就是一个 NamingAttributesList-T 指定一个对象，在本接口中，用 NamingAttributesList-T 来表示一个对象，又叫做对象名。

所有的资源包括网元，设备容器，设备，Termination point 名字变量类型都是 NamingAttribute_T。NamingAttribute_T 是 Name 和 Value 对。Name 和 Value 都是字符串。

而由于 SNMP Agent 不提供条件查询的功能，所有的 MIBLEAF 没有逻辑关系，因此通过修改协议来实现 SNMP 条件查询，我们将所有 NamingAttribute_T 修改成 Object Iddentifier。所有的表中概念行的某个列定位如下：TableOID+COL+Index；在所有的表中，我们把各个资源的名字映射成 Index。所以我们需要作的实际上把 ANS.1 转化成 SNMP 中的 Object Identifier 格式的 Index，然后再把这些 Index 作为建立各个行的索引。

4.2 AGENT 中 SNMP PDU 和视图的处理过程

这部分设计了 SNMP 在 AGENT 中 SNMP PDU 和视图的处理过程。

4.2.1 AGENT 对一个 SNMP PDU 请求的处理过程：

A. 通过 CommReceiveTask 任务从网络接收一个 SNMP PDU 请求；

B. 调用 Process_Rcvd_SNMP_Packet_Async 对包进行处理；

C. SNMP 系统内核调用每个 VBP 的回调函数进行处理，每处理完一个 VBP 都要用 getproc_xx/setproc_xx 进行标记（标记该 VB 已经被处理了），同时设置该 VB

的值（对于 GetRequest）；

D. 对于进行异步处理的情况，用户显示地调用 SNMP_Continue，在这个调用中判别是否所有的 VB 都处理完了，如果是，则调用 SnmpIoComplete 进行处理，如果没有处理完则继续调用用户的回调进行处理；

E. 如果每个 VBP 都处理完毕，则调用 SNMP_Process_Finish 标记一个 SNMP 包已经处理完毕，并对响应 PDU 进行编码；

F. 通过 CommSendTask 将 SNMP 响应发送给 MANAGER，这样一个 SNMP PDU 请求正式处理完毕；

E. 如果在 C/D 的处理中出现了错误（由用户使用 getproc_error 进行标记），则直接跳到 E 进行处理，此时系统设置 PDU 中的 error-status 和 error-index。对于 getproc_nosuchins（对于 noSuchObject 的情况，此时因为没有找到相应的对象，因此也就不会调用用户的回调函数，而是由 SNMP 工具包直接将 VB 的值设置为 noSuchObject）的处理，SNMP 工具包的处理是只是将 VB 中的 value 设置为 noSuchInstance,其余的处理与 C 的处理一样，继续执行后续 VB 的 GET 操作，而不是直接跳转到 E 进行编码处理，即与正常处理完全一样，同样也将响应 PDU 中的 error-status 设置为 noError,将 error-index 设置为 0。当然 MANAGER 在接收到 GET 响应后，如果响应中 error-status 为 noError，则在获取 VB 的值时，首先要判别是否是 noSuchInstance 和 noSuchObject，只有不是这两种情况时，才能获取其正确的值。只有在出现 genErr（调用 getproc_error 进行标记）时才会设置 PDU 中的 error-status 和 error-index。

4.2.2 构造和发送一个 SNMP PDU 的过程

A. 调用 EBufferInitialize()初始化一个编码缓冲区。

B. 调用 SNMP_Create_Request2 构造一个 SNMPv2 的包结构,其入口参数包括：ptype 表示 SNMP PDU 的包类型，如 Get/Get-Next 等；version 表示所使用的 SNMP 的版本号；commlen 表示 SNMP 中所使用的群体名的长度；community 表示群体名；request_id 表示 SNMP PDU 中填写的请求号；num_vb 表示 SNMP PDU 中 VB 对的个数；nonreps 表示不重复的个数（用于 Get-Bulk）；maxreps 表示最大重复的个数（用于 Get-Bulk）。

C. 根据前面的 VB 个数填写相应的 VB 绑定对的对象 ID 和值（调用 SNMP_Bind_xxx）。

D. 调用 SNMP_Encode_Packet 对前面得到的包进行编码。

E. 将得到的编码后的包发送到目的地址。

F. 释放所使用的内存空间（包括编码前和编码后所使用的空间）。

H. 在调试 SNMP 功能的时候，每次上报的数据，如果数据的类型是字符串类型则每次上报的数据都是错的，是一些乱码数据，而对于简单类型如整数等则上报的数据是对的。通过跟踪程序发现协议的解包是对的，因为正确的调用了回调函数，在构造 RESPONSE 的最后一步调用了 PSOS 提供的 SNMP 的函数 getproc_got_string，该函数用来填写 VB 的值时，调用该函数是应用层处理数据的最后一步，这时发现应用传递的参数都是对的，也就是说我们应用层处理的逻辑是对的，错误可能出现的地方是我们对该函数的调用不对或者该函数有问题。但 PSOS 对 SNMP 软件包所提供的函数的说明很少，通过一些资料的研究和 SNMP 自动生成的代码的注释发现该函数的处理非常的特别，如下：

void getproc_got_string (SNMP_PKT_T *pktp, VB_T *vbp,
ALENGTH_T stringlen, OCTET_T string, int dynamic, OCTET_T type)

这个函数的功能是将字符串绑定到VB列表中，即对于字符串类型的VB取值操作时，用这个函数给VB赋值。这个函数中的字符指针的使用有特殊要求，当标志dynamic为1时，要求在使用时动态申请内存给字符串指针，PKTP包处理完之后，系统自动释放指针所对应的内存，用户不要自己释放指针，否则会出错。即当标志dynamic为1时，该函数内部会重新申请一块内存，然后把传递到函数内部的string的值拷贝到新申请的内存（深拷贝），再把string释放掉。当标志为0时，系统认为存放字符串的内存空间是静态的，不需要释放，他将字符串指针的值赋给VB，直到打包发送以后这段内存空间的值才允许改变，否则会出现取值错误。即当标志dynamic为0时，该函数直接把string的指针赋给VB，而不是string指向的内容（浅拷贝）。

但我们一般调用这个函数的地方是在回调函数内，指针是在回调函数内定义的（如getproc_got_string(pktp, vbp, string_length(data->tpName), data->tpName, 0, VT_STRING);），dynamic为0，跳出之后就释放了data指针，这样就导致了问题发生，因为getproc_got_string函数内部并没有真正的拷贝内容，而仅仅是拷贝指针，这样我们就在发送前把指针指向的内存给释放了，就出现了问题。

找到原因后，我们对该问题的解决办法如下：

每次调用该函数前，先申请一个内存pTmpData，然后把要发送的内容拷贝到该内存，再把dynamic设为1，由于getproc_got_string会自动释放该内存pTmpData，

所以在应用层不能释放pTmpData。如下：

```
pTmpData = (unsigned char *)SNMP_memory_alloc(TPNAMELEN);
    if(NULL != pTmpData)
    {
        memcpy(pTmpData, data->tpName, TPNAMELEN);
        pTmpData[TPNAMELEN-1] = '\0';
        getproc_got_string(pktp, vbp, string_length(pTmpData), pTmpData, 1,
        VT_STRING);
     }
```

4.2.3 对于 GetNextRequest 和 GetBulkRequest PDU 的处理

对于 Get-Next PDU 请求的处理，其含义是查找当前索引所表示的实例（如表中的一行）在字典顺序上的下一行，而不是物理上的下一行，比如对于表 Table 来说，假设在物理存储时，A、B、C 分别为表的第一、二、三行，而其字典顺序为 A、C、B，那么 MANAGER 对 A 发 Get-Next 命令时，返回的应该是 C 而不是 B。对于 SNMP 工具包来说，传递到表的 get-next-async 回调中的 compc 和 compl 分别表示表索引的长度和索引值，其中 compc 有可能为 0，这时表示取表在字典顺序上的第一行，参见 rfc1905(SNMPv2 的协议操作)中关于 Get-Next PDU 的操作部分。nextproc 必须调用 nextproc_next_instance，nextproc_next_instance_string、nextproc_no_next 或者 nextproc_error 表明该 VBP 已经被处理了。

对 Get-Bulk PDU 的处理，请参见 pSOS SNMP 使用手册 P48 和 rfc1905 中对 Get-Buld的描述。Get-Bulk请求PDU中由两部分组成，第一部分主要表示获取标量变量的值，这些变量共有non-repeaters个，对于这一部分的处理与GetRequest的处理相同，调用的也是Get的回调函数；第二部分主要是对于概念表的处理，对于这一部分的处理，SNMP工具包对所有的VarBind list进行多次迭代，每次迭代都相当于一次Get-Next的处理，每次迭代时都将前一次迭代的结果作为本次迭代的入口（当然第一次迭代时可能没有指定所要获取的表的索引值，此时表示获取表在字典顺序上的第一行），当响应包达到最大值、或者迭代的次数等于 max-repetitions、或者已经没有后续的数据、或者处理时出现了错误时，整个迭代操作才停止，这时SNMP工具包调用SnmpIoComplete将响应编码并发送给 MANAGER。

4.2.4 对于 SetRequest PDU 的处理

A. SNMP 工具包调用 validate_set_pdu()（每个 AGENT 中只有一个此函数）对一个 Set PDU 进行检查，这是一个包一级的回调函数，与后面的 testproc 和 setproc 不同，它们是 VB 一级的回调。程序中使用这个回调可以在一开始就有机会判别 SET 的值是否合法，特别是当多个变量之间存在某些逻辑关系时，这样就可以避免出现发现这种不合法时程序已经对某些变量进行了设置操作，在这种情况下，程序要做许多回退操作（将已经设置了的变量恢复成其原来的值）；使用这个回调函数，可以减少 AGENT 中回调函数的数量，使程序更容易维护，但需要实现一个好的 OID 匹配函数（实现的好坏与最终系统运行的效率有很大的关系），这样才能具体定位某个变量。对于使用 testproc/setproc 函数的情况，OID 匹配功能由工具包自己完成，因而不需要用户程序关心。这个回调可以有三种返回方式：

a. 返回-1，表示这个 Set PDU 包有错误，此时工具包终止后续的处理（不会再调用后面的与 SET 有关的回调），产生一个 set-reponse 包，并将包的 error-status 设置 为 GEN_ERR,然后调用 SnmpIoComplete 将响应发送给 MANAGER。

b. 返回 0：表示此 SET-PDU 没有问题，返回后 SNMP 工具包会调用后续的回调（如 testproc/setproc）进行处理，当然用户也可以将某些 VB_T 的 vb_flags 设置为 VFLAG_ALREADY_TEST，表明这些 VB_T 已经检查过了，这样 SNMP 工具包就不调用这些 VB 的 testproc 回调了。

c. 返回 1：表示此 SET-PDU 没有问题，而且用户将所有的 Set 工作在此全部完成，该回调返回后，工具包终止后续的处理，产生一个 set-reponse 包，并标记这个 PDU 处理没有任何问题，然后调用 SnmpIoComplete 将响应发送给 MANAGER。

B. SNMP 工具包为 SET PDU 中的每个 VB_T 调用其 testproc_xxx（）函数（如果存在的话），当然在此回调中，用户也可以将该 Set PDU 中公用此回调的所有 VB 一次性处理（通过调用 group_by_getproc_and_instance 函数来完成），测试完成后用户程序要调用 testproc_good 或 testproc_error 表明测试是否正确，也可以调用 setproc_good()表明对这个 VB 的 SET 操作已经完成，此后工具包就不会调用其 setproc_xxx（）。此回调返回后，对于同组处理过的 VB，工具包不会再调用此回调对其进行检测。此回调的具体的函数名要在 MIB 中用“DEFAULT test-function-async xxxxxxx”进行指定（具体参见 SNMP 手册第八章）。如果 B 中没有错误,工具包调用 user_pre_set(),其功能和返回值与 validate_set_pdu()类似，如果返回出错，工具包会调用 user_set_failed()并产生一个错误的响应包

给 MANAGER。

C. SNMP 工具包为 Set PDU 中的每个 VB_T 调用其 setproc_xxx（）回调函数（如果存在而且前面的回调中没有对其进行 SET 的话）。在这个函数中，用户程序要调用 setproc_good()或 setproc_error（）表明已经对此 VB 进行了处理。这样在明确(当用户异步处理完成后，由用户程序进行调用)或隐含调用 SNMP_Continue（）时，都要判别相应的处理是否完成（主要是看 VB 的 vb_flags 域的相应标记是否设置），如果在此回调中没有调用 setproc_good()或 setproc_error（）设置相应的标记，则工具包就会再次进入这个函数，因而形成一个无限循环。

D. 如果前面处理都没有出错的话，工具包将调用与 SET 有关的最后一个回调函数 user_post_set()，主要是给用户程序提供一个 SET 善后处理的场合，比如清除以前所分配的空间或者对后台设备执行某些操作等。

E. 如果在 setproc 时出错，SNMP 工具包去查找那些已经进行了 SET 操作的 VBP 的 undoproc 域，从而调用这些 undoproc 回调函数恢复前面所进行的所有 SET，VBP 的 undoproc 域可以在 VBP 的 testproc 和 setproc 回调中进行设置。在 undoprocs 中也可以进行异步处理，每次异步处理完成后必须调用 SNMP_Continue 让工具包进行后续的处理。当所有的 undoprocs 处理完成后，工具包调用 user_set_failed()并产生一个错误的响应包给 MANAGER。

F. 在进行 SET 操作时，可以使用 VBP 的 vb_priv 和 vb_free_priv 来保存一些自己要使用的内部数据。vb_priv 用来保存自己的数据（比如存储被 SET 对象实例的老数据，这样在 SET 失败时，undoproc 就可以使用这个数据进行恢复操作），而 vb_free_priv 则表示释放此数据的子程序，主要是当 SET 操作失败时，为用户提供一种释放资源的手段。用户一般在 testproc 回调中设置这两个域，在 setproc 中，如果一切都操作正常，则要清除这两个域，此时要自己释放资源。

4.2.5 视图的处理过程

我们只使用 rfc1445 所定义的视图规范。视图是指 SNMP 实体可以访问的管理对象资源。在 SNMPv2 中，我们必须创建和安装相应的视图，才能进行 MIB 对象的管理操作。在 OADM 中我们只使用最基本的视图操作。

A. 视图的创建：可以用 SNMP_View_Create 创建一个视图，这是所有视图操作的基础。其入口参数是所创建视图的对象标识符，返回一个 VIEWLEAF_T 类型的视图结构。其功能是创建一个 VIEWLEAF_T 类型的视图结构并设置其缺省状态。只

有对其进行安装后，才能使用这个视图。

B. 调用 SNMP_View_Set_xxx 设置 A 中所创建的视图的相关信息，主要是设置视图的类型（指是 included 还是 excluded）、状态（激活、非服务状态等）、存储类型（挥发或非挥发等）、掩码（掩码的表示形式为“xx:xx:xx”，其中 xx 为 16 进制的数字，按位来表示某个对象标识符是否进行比较，为 1 表示需要进行比较，为 0 表示不比较，假设子树为“1.3.6”，掩码设置为“e0”，e 的二进制表示为 1110，这样表示在对 MIB 进行访问时，对于每个目标对象标识符，都要比较它的前三个标识符是否为“1.3.6”，如果不是，则不允许访问，如果是，则可以进行访问，同时由于掩码后面的全为 0，表示不进行比较，这样某个 SNMP 实体就可以访问“1.3.6”这棵子树下所有的对象。如果掩码为“e1”，其二进制标识为 11100001，则表示除了比较前面 3 个标识符组件外，还要比较第八个标识符组件。）等信息。

C. 调用 SNMP_View_Install 对视图进行安装，该函数的入口参数为所安装视图的索引号和前面 A、B 中创建的视图。到现在为止，这个视图才可以使用。

D. 当一个视图没有使用价值后，必须用 SNMP_View_Delete 将其删除，在删除视图之前，必须调用 SNMP_View_Deinstall()将这个视图从视图表中卸载下来。

4.3 AGENT 中的信息流向

这部分设计了 OADM AGENT 中典型的信息流向和处理过程。

4.3.1 非异步 Get/SetRequest 操作的信息流向

非异步 Get/SetRequest 操作的信息流向(GetNextRequest 和 GetBulkRequest 的信息流向与下面一致)如图 4-1 所示：

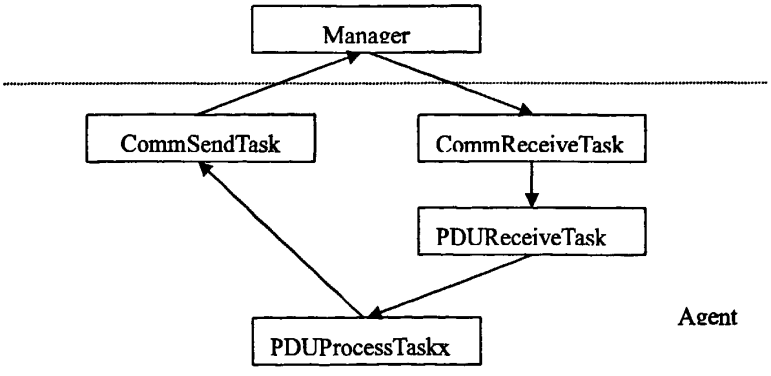


图 4-1 非异步 Get/SetRequest 操作的信息流向

具体的流程如下：

1. MANAGER 将 SNMP PDU 发送给 AGENT，通讯的协议可以是 UDP、TCP（必须使用与 161 不同的端口，此处用宏 TCP_PORT 进行标识）或者是 TP4；

2. CommReceiveTask 接收 MANAGER 发送给 AGENT 的 SNMP PDU，并将其发送给 PDUReceiveTask；

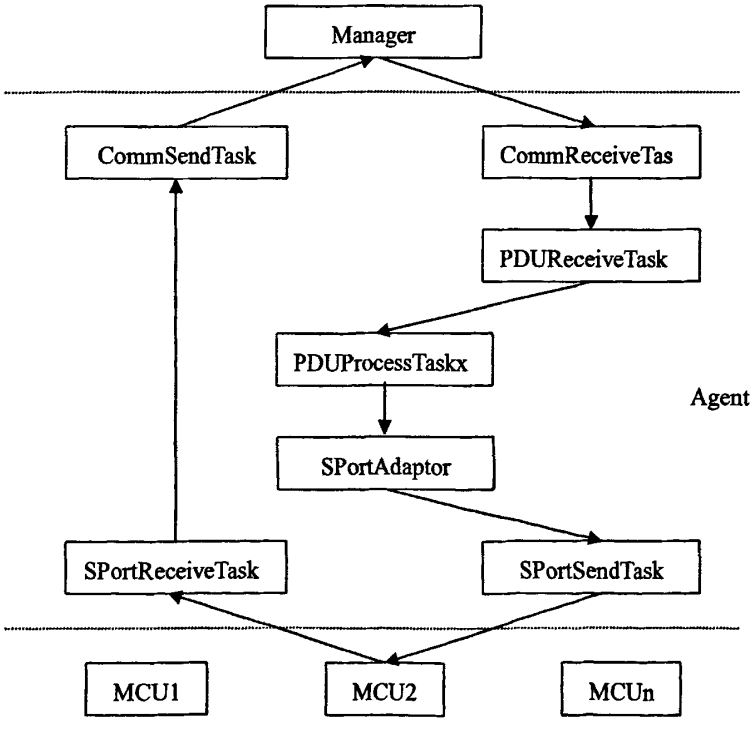
3. PDUReceiveTask 接收到该 PDU 后，根据负载平衡算法将其发送给选定的 PDUProcessTask；

4. PDUProcessTask 调用 Process_Rcvd_SNMP_Packet_Async 对该 PDU 进行处理，在这个函数处理中，首先调用 validate_SNMP_community（）进行访问确认，所有的 VB 都处理完毕后调用 SnmpIoComplete 对 PDU 响应进行编码，然后发送给 CommSendTask；

5. CommSendTask 将编码后的响应 PDU 通过合适的通讯协议发送给 MANAGER。

4.3.2 异步 Get/SetRequest 操作的正常信息流向

异步 Get/SetRequest 操作的正常信息流向如图 4-2 所示：



4-2 异步 Get/SetRequest 操作的正常信息流向

具体的流程如下：

1. MANAGER 将 SNMP PDU 发送给 AGENT，通讯的协议可以是 UDP、TCP 或者是 TP4；

2. CommReceiveTask 接收 MANAGER 发送给 AGENT 的 SNMP PDU，并将其发送给 PDUReceiveTask；

3. PDUReceiveTask 接收到该 PDU 后，根据负载平衡算法将其发送给选定的 PDUProcessTask；

4. PDUProcessTask 调用 Process_Rcvd_SNMP_Packet_Async 对该 PDU 进行处理。此时可能需要与 MCU 进行信息交互，因而需要采用异步处理方式，在调用 RegisterAsyncEntry 注册一个异步处理 ENTRY 后将 MCU 命令发送给 SPortAdaptor，并将正在处理的 PDU 挂起；

5. SPortAdaptor 将 MCU 命令发送给 SPortSendTask；

6. SPortSendTask 将命令发送给 MCU；

7. MCU 将命令的响应发送给 SPortReceiveTask；

8. SPortReceiveTask 将响应发送给 SPortAdaptor；

9. SPortAdaptor 接收到 MCU 响应后，根据响应中的 PDU ID（放在报文的 CALLID 中）和 S 口请求 ID（放在报文的 IVH 中），查找到对应的异步处理 ENTRY，如果所有的 MCU 命令都处理完毕，则调用 SNMP_Continue()继续 PDU 的处理，此时如果所有的 VB 都处理完毕，调用 SnmpIoComplete 对 PDU 响应进行编码，然后发送给 CommSendTask；

10. CommSendTask 将编码后的响应 PDU 通过合适的通讯协议发送给 MANAGER。

4.3.3 异步 Get/SetRequest 操作超时时的信息流向

异步 Get/SetRequest 操作操作超时时的信息流向如图 4-3 所示：

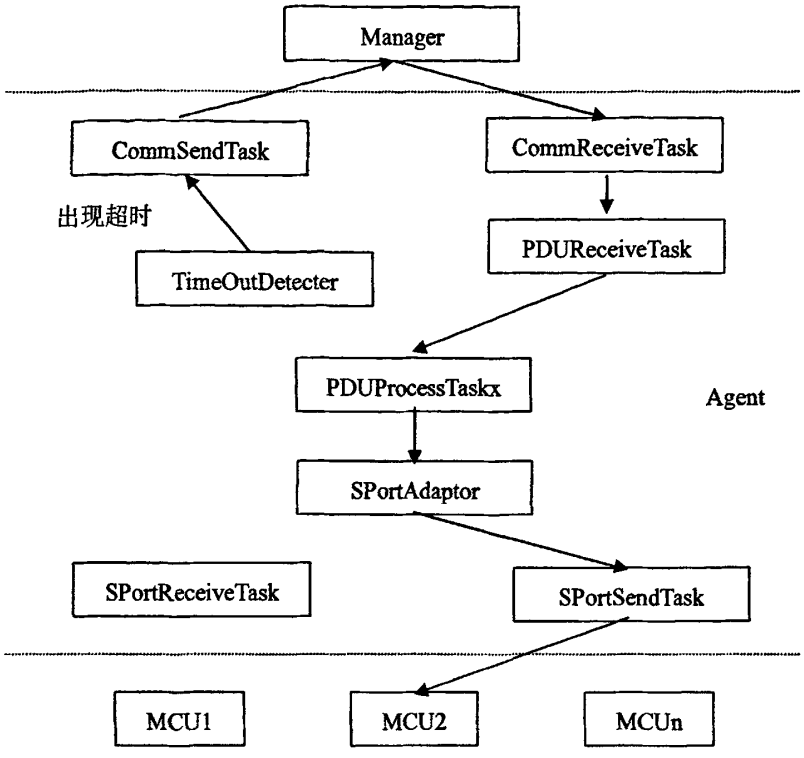


图 4-3 异步 Get/SetRequest 操作超时时的信息流向

具体的流程如下：

1. MANAGER 将 SNMP PDU 发送给 AGENT，通讯的协议可以是 UDP、TCP 或者是 TP4；

2. CommReceiveTask 接收 MANAGER 发送给 AGENT 的 SNMP PDU，并将其发送给 PDUReceiveTask；

3. PDUReceiveTask 接收到该 PDU 后，根据负载平衡算法将其发送给选定的 PDUProcessTask；

4. PDUProcessTask 调用 Process_Rcvd_SNMP_Packet_Async 对该 PDU 进行处理。此时可能需要与 MCU 进行信息交互，因而需要采用异步处理方式，在调用 RegisterAsyncEntry 注册一个异步处理 ENTRY 后将 MCU 命令发送给 SPortAdaptor，并将正在处理的 PDU 挂起；

5. SPortAdaptor 将 MCU 命令发送给 SPortSendTask；

6. SPortSendTask 将命令发送给 MCU；

7. 在等待 MCU 响应时出现超时，TimeOutDetecter 调用 SNMP_Continue 对进

行异步操作的 PDU 进行处理，并将处理的结果（编码后的响应 PDU，标识为 genErr）发送给 CommSendTask；

8. CommSendTask 将编码后的响应 PDU 通过合适的通讯协议发送给 MANAGER。

4.3.4 下载私有配置的正常信息流向

下载私有配置的正常信息流向如图 4-4 所示：

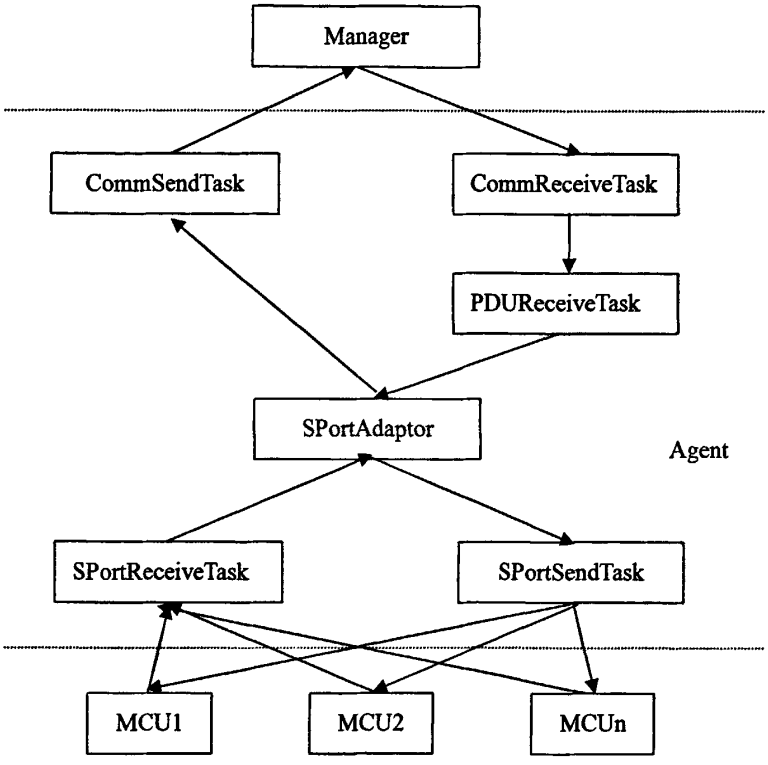


图 4-4 下载私有配置的正常信息流向

具体的流程如下：

1. MANAGER 将私有配置数据发送给 AGENT（使用 TCP 或 TP4 通讯协议，如果使用 TCP 的话，必须使用特定的端口，否则应用管理任务无法对信息进行区分，此处用 LOAD_DATA_PORT 进行标识）；

2. CommReceiveTask 接收 MANAGER 通过 TCP（或 TP4）发送给 AGENT 的私有配置数据，并将其发送给 PDUReceiveTask；

3. PDUReceiveTask 接收到后，进行私有配置数据的处理，此时需要与 MCU 进行信息的交互，因此需要进行异步处理的登记，然后将 MCU 命令发送给

SPortAdaptor；

4. SPortAdaptor 将接收到的 MCU 命令发送给 SPortSendTask；

5. SPortSendTask 将命令发送给 MCU；

6. MCU 将命令的响应发送给 SPortReceiveTask；

7. SPortAdaptor 根据接收到的 MCU 响应，检查是否执行成功，如果全部执行成功，此时可以存储私有配置数据，向 MANAGER 返回成功；否则不存储数据，返回失败；

8. 将 AGENT 对私有配置数据处理的结果发送给 CommSendTask；

9. CommSendTask 将私有配置数据的响应发送给 MANAGER。

4.3.5 下载私有配置数据超时时的信息流向

下载私有配置数据超时时的信息流向如图 4-5 所示：

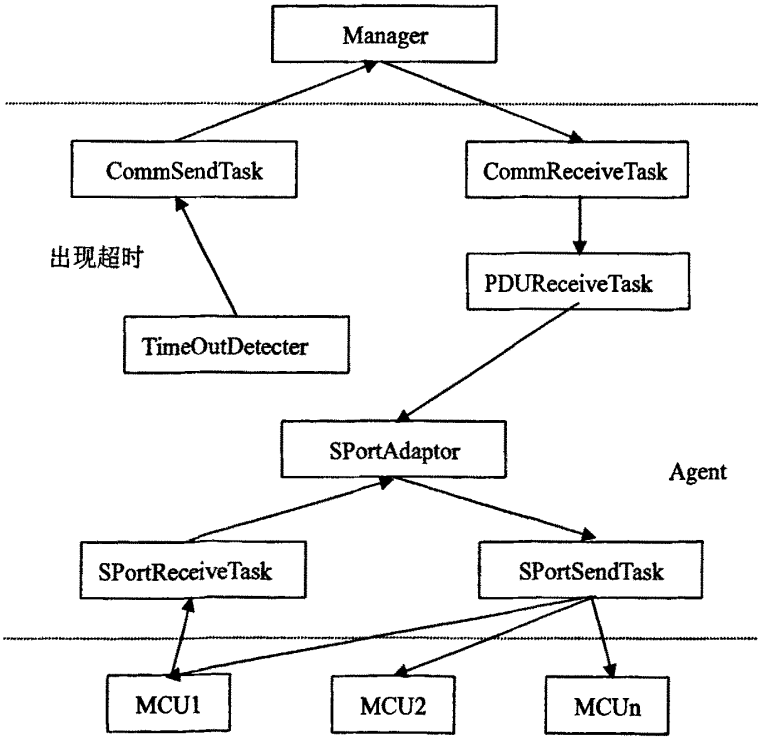


图 4-5 下载私有配置数据超时时的信息流向

具体的流程如下：

1. MANAGER 将私有配置数据发送给 AGENT(使用 TCP 或 TP4 通讯协议)；

2. CommReceiveTask 接收 MANAGER 通过 TCP（或 TP4）发送给 AGENT 的私有配置数据；

3. PDUReceiveTask 接收到后，进行私有配置数据的处理，此时需要与 MCU 进行信息的交互，因此需要进行异步处理的登记，然后将 MCU 命令发送给 SPortAdaptor；

4. SPortAdaptor 将接收到的 MCU 命令发送给 SPortSendTask；

5. SPortSendTask 将命令发送给 MCU；

6. MCU 将命令的响应发送给 SPortReceiveTask；

7. SPortAdaptor 根据接收到的 MCU 响应，检查是否执行成功，此时因为只接收到一个 MCU 响应，所以继续等待其它两个 MCU 响应；

8. 在等待 MCU 响应时出现超时，TimeOutDetecter 判断 MANAGER 下载私有配置数据失败，并将响应结果发送给 CommSendTask；

9. CommSendTask 将响应结果通过合适的通讯协议发送给 MANAGER。

4.3.6 上载私有配置数据的信息流向

上载私有配置数据的信息流向如图 4-6 所示：

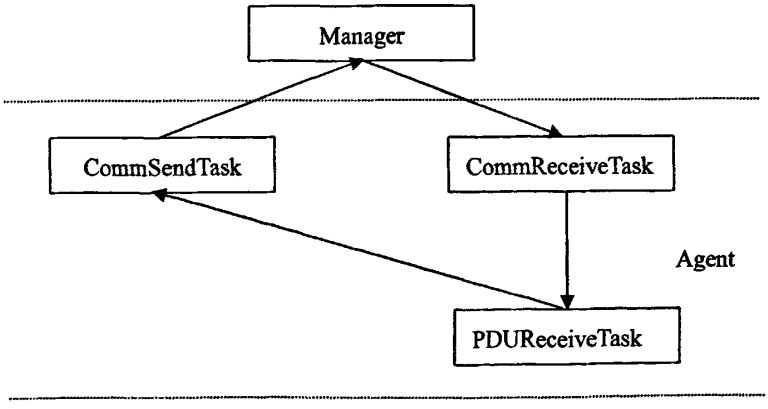


图 4-6 上载私有配置数据的信息流向

具体的流程如下：

1. MANAGER 将上载私有配置数据命令发送给 AGENT(使用 TCP 或 TP4 通讯协议)；

2. CommReceiveTask 接收 MANAGER 通过 TCP（或 TP4）发送给 AGENT 的上载命令，并将其发送给 PDUReceiveTask；

3. PDUReceiveTask 接收到后，进行上载数据的处理，并将处理的结果发送给 CommSendTask；

4. CommSendTask 将响应结果通过合适的通讯协议发送给 MANAGER。

4.3.7 上报权限检查失败 trap 的信息流向

上报权限检查失败 trap 的信息流向如图 4-7 所示：

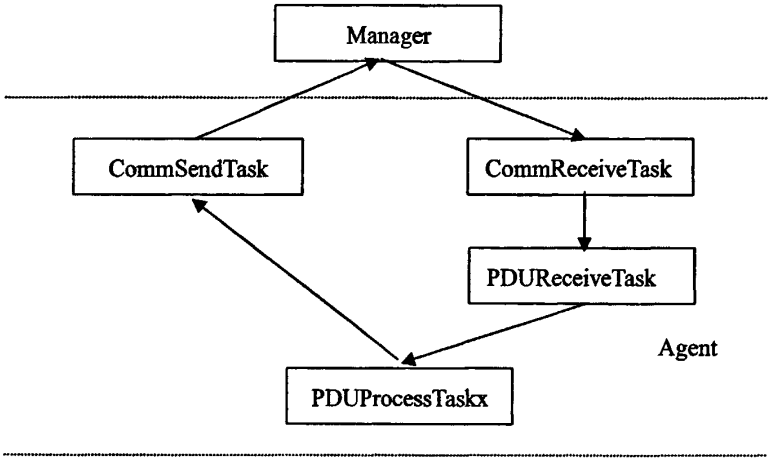


图 4-7 上报权限检查失败 trap 的信息流向

具体的流程如下：

1. MANAGER 将 SNMP PDU 发送给 AGENT，通讯的协议可以是 UDP、TCP 或者是 TP4；

2. CommReceiveTask 接收 MANAGER 发送给 AGENT 的 SNMP PDU，并将其发送给 PDUReceiveTask；

3. PDUReceiveTask 接收到该 PDU 后，根据负载平衡算法将其发送给选定的 PDUProcessTask；

4. PDUProcessTask 调用 Process_Rcvd_SNMP_Packet_Async 对该 PDU 进行处理，在 PDU 的处理中，首先调用 validate_SNMP_community 进行权限检查，检查不成功，则向 MANAGER 发送权限检查失败 trap，并终止 PDU 的处理；

5. CommSendTask 将接收到的权限检查失败 trap 通过合适的通讯协议发送给 MANAGER。

4.3.8 上报告警 trap 的信息流向

上报告警 trap 的信息流向如图 4-8 所示：

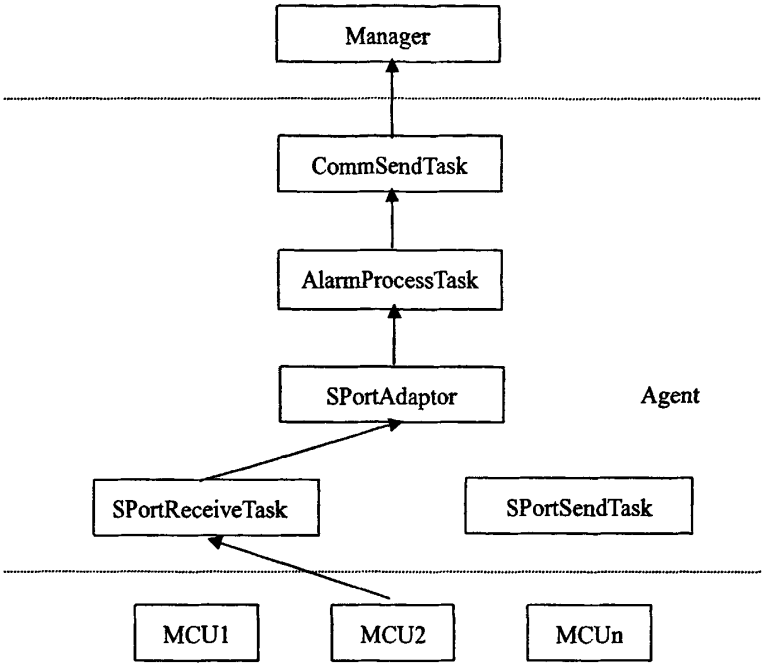


图 4-8 上报告警 trap 的信息流向

具体的流程如下：

1. MCU 检测到单板上的告警发生或消失，并将其报告给 SPortReceiveTask；

2. SPortReceiveTask 将告警发送给 SPortAdaptor；

3. SPortAdaptor 完成 MCU 响应或主动上报信息的分发，因而将告警信息发送给 AlarmProcessTask；

4. AlarmProcessTask 接收到该告警后，刷新当前告警库，并进行 NCP 和机架告警灯及声音告警的控制，然后根据 trapPermitted 和告警过滤的状况产生告警 trap，并发送给 CommSendTask；

5. CommSendTask 将接收到的告警 trap 通过合适的通讯协议发送给 MANAGER。

6. 由于 SNMP 协议的特殊性，在告警处理上，与其他的网管有一些不同。告警分为查询，刷新和主动上报几种。

对于查询操作，由界面直接向 MANAGER 发查询命令，而不是向 AGENT 发送命令。

对于刷新操作，MANAGER 清除自己的数据，并向 AGENT 发送刷新告警的命令。

AGENT 清除自己保留的数据，采取异步操作的方式向单板取告警。

对于单板主动产生的告警，采取了非标准的 TRAP 的方式，一般的标准 TRAP 只绑定一条记录，这样一次只能上报一条告警，我们的设备如果采取这种方式势必很慢。所以采取了私有的方式，对于一次绑定多条记录的 TRAP 的验证结果如下：

关于 TRAP 绑定 VB 的个数：

按照 SMI（SNMP 的管理信息结构）的规定，TRAP 只能绑定一个 VB，但通过实验，自定义的 TRAP 也可以绑定多条 TRAP。绑定的个数只受发送长度的限制。

关于 TRAP 的长度：

按照正常方式，TRAP 采用 UDP 方式传输，最大长度不超过 1400 个字节。但通过实验，发现可以突破这个长度，只要对 UDP 的 SOCKET 调用 SETSOCKETOPT（）函数进行发送长度的设置。最大长度可以达到 8000 个字节。并且通过协议分析软件的分析，发现采取 UDP 方式在 PPP 链路上传输时，8000 个字节的分包以及组包过程全部在 IP 层完成。但数据经过分包之后，传递的可靠性受到影响。尤其在多个 AGNET 同时上报 TRAP 的时候。

在测试中比较明显地可以看到 SNMP 报文中冗余信息比较多，不太适用于大数据量通讯的场合，比如在查询历史 15 分钟性能时，使用 SNMP 处理就可以明显地看到速度比较慢，需要很长时间才能处理完毕一个 15 分钟内产生的历史性能数据。对于数据量不是很大的一些管理网元的操作，使用 SNMP 模块有实现简单、便于扩展等优点，但是由于协议本身的限制，使得在应用 SNMP 模块处理大数据量时，不得不采用分包的策略，导致处理效率会明显降低。

4.4 SNMP AGENT 协议基于 PSOS 的实现要点

SNMP 协议的 MANAGER 侧在工作站上实现，AGENT 在 PSOS 上实现。PSOS 上对于 SNMP 协议大概提供了三个方面的支持：

(1)在 PSOS 中有一个支持 SNMP 协议的库和一些相关的头文件。

(2)因为 MIB 库是用描述性语言写的，文件名称是*.MIB，所以还需要提供一个专门用来编译 MIB 的编译器。这个编译器根据特定的 MIB 库的格式生成一个.C 的源文件以及一些头文件。

(3)另外还提供了一个软件包，软件包提供了进行 SNMP 处理的基本框架和封装了一些处理函数。比如编码解码处理函数。

在 PSOS 上实现 SNMP 协议最主要的就是根据基本框架填加一些具体的处理。

下面大概介绍一下 PSOS 上实现 SNMP AGENT 的软件框架结构。如图 4-9 所示。

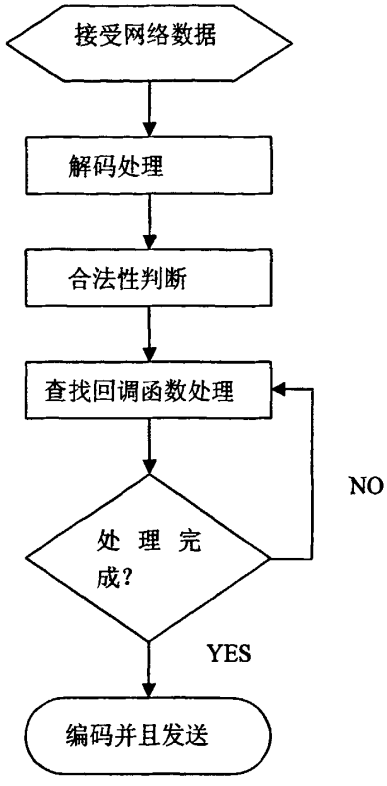


图 4-9 SNMP AGENT 处理流程

(1)通过 UDP 方式接收 MANAGER 发送过来的请求，可以是 SNMPGET，SNMPSET，SNMPNEXT 等几种请求。这时接收到的数据是经过编码的，要正确识别数据内容，首先调用解码处理。解码的处理是由 PSOS 工具包提供的。

(2)解码之后要进行数据合法性的检查，这里主要有两项检查，一是看发送命令的 MANAGER 是否有操作权限。比如有的 MANAGER 只可以进行查询而不能进行设置，有的只能对一棵 MIB 树上的一个分支进行操作。这些都要经过检查。另外一项检查主要是针对 SET 操作时的数据合法性检查，比如检查设置的值是否在许可范围之内，类型是否相符合。

(3)之后系统自动调用相应的回调函数对具体的操作进行处理。在 MIB 树上有很多节点，节点可以是简单类型，也可以是表格。在 MIB 库中，每一个节点可以对应一个处理函数。比如其中一个节点代表当前告警表。那么我们在 *.mib 文件中对表格定义的地方用一个特殊的语句 DEFAULT 来定义对应当前告警表的处理函数。如下面例子所示。每一个 SNMP 命令对应一个回调函数。当 MANAGER 发送 SNMPGET

命令时，系统调用告警表中对 SNMPGET 操作的 otnCurrentAlarmTable_get_async 函数进行处理。如果 MANAGER 发送 SNMPSET 命令时，系统调用告警表中对 SNMPSET 操作的 otnCurrentAlarmTable_set_async 函数进行处理。

不同的叶子可以采用不同的回调函数，也可以采用相同的回调函数。如果某个叶子节点没有定义回调函数。则系统去查找他的父亲节点有没有定义回调函数。如果定义了，则采用父亲节点的回调函数进行处理。如果父亲节点没有定义，再向上查找。一直到找到回调函数为止。

在 PSOS 上实现 AGENT 时，最主要的工作在于为 MIB 库中定义的不同节点编写不同的回调函数。比如当前告警表，告警属性表，当前性能表，历史性能表等。

对于 MANAGER 下发的每一个 VB，AGENT 都需要标志他是否已经被处理，否则会出现死循环。

(4)如果所有 VB 已经被处理完毕，系统就可以进行编码并且向网络发送回应数据。至此，处理过程就完成了。

4.5 ENVOY 对 SNMP 的支持

Epilogue 公司的产品 ENVOY v9.1 为我们开发 SNMP 提供了支撑框架，它能支持 SNMP V1、SNMP V2、SNMP V3，而且通过 MIBCompiler 能编译生成自己的 MIB 库，其源代码是 C 文件，并提供了与操作系统相关的配置文件，可在多个操作系统上运行。该产品的主要特性如下：

1. 支持 SNMP V1、SNMP V2、SNMP V3 的组合配置选项。
2. 支持用户定义的管理参数。
3. 支持用户定义的 TRAP 和 INFORMS.
4. 可以作为 Agent 运行在被管理设备或管理工作站上。
5. 可以在操作系统内核、应用程序内、智能网络前端设备上运行。
6. 支持被管理设备上在一个线程内同时处理多个 SNMP 请求。
7. 占用内存少。
8. 支持多个 MIB 视图，从而使不同用户看到不同的 MIB 变量。
9. 为 Agent 和 Manager 的开发提供了辅助函数库，包括生成行集。
10. 提供了简易的配置系统。通过编辑单个配置文件，用户可以指定编译选项和编译器。用户可以轻松地加入新的编译器。
11. 支持多个 MIB 树，你可以为每个数据包或者社区（community）选择相应

的 MIB 树，这提供了一种改变社区或上下文相关的数据或例程的手段。缺省是一个 MIB 树，如果你不使用该特性的话，就不用做额外的工作。

12. 提供对一般性代理机制的支持，也提供针对 SNMPv1 和 SNMPv2 的代理支持。

13. 允许动态增删 MIB 树节点。

14. 支持 AgentX 协议，从而子 Agent 能动态地在主 Agent 上注册、注销。

15. 支持 AgentX 协议，从而允许用户选择在主 Agent 和子 Agent 之间通信的方法。

Envoy SNMP 提供了五个功能单元：

Decoder：完成解码功能。它的输入是一个 ASN.1 格式编码的 SNMP packet，输出是一个 C 的数据结构，该结构代表的内容是这个 packet。并且，它还完成简单的安全检查功能，包括检查用户的团体字符串和 MIB 视域。

Encoder：它的功能和 Decoder 相反，是完成编码。其输入是一个 C 的数据结构，输出是一个适合于承载在 UDP 之上传输的 ASN.1 格式的包。

MIB Interface：它的输入是一个代表 Get/GET NEXT/SET/GET BULK 的请求包(Request Packet)的 C 数据结构，MIB interface 处理这个请求包，并产生一个应答包(Response Packet)。

Received Packet Manager：守候任务收到 SNMP 的 packet 之后，调用本功能实体，本功能实体依次调用 Decoder、MIB Interface 和 Encoder 功能实体来处理这个数据包。

Packet Generator：这个功能实体主要是用来完成告警数据的上报的。它接收告警管理功能模块的调用，并调用 Encoder，将系统内部告警数据封装成 SNMP 的 Trap/Inform 包，发送到 SNMP Manager。

下面介绍与 SNMP 包处理流程相关的几个重要的入口函数：

Void IO_COMPLETE_T (SNMPADDR_T *for_addr, SNMPADDR_T *loc_addr, PTR_T pktp, ALENGTH_T need, PTR_T cookie)

该类型的函数被作为参数传给 Process_Rcvd_SNMP_Packet_Async()，并在包处理完毕需要发送响应包时被调用，这个函数会调用 SNMP_Process_Finish () 来将要发送的包 pktp 编码放入一个缓冲区。

Void ERROR_COMPLETE_T (SNMPADDR_T *for_addr, SNMPADDR_T *loc_addr, int error_code, PTR_T cookie)

该类型的函数被作为参数传给 Process_Rcvd_SNMP_Packet_Async( )，并在 ENVOY 处理完 SNMP 包又没有响应包的情况下被调用，

void Process_Rcvd_SNMP_Packet_Async(size_t pktl, bits8_t *pktp, SNMPADDR_T *for_addr, SNMPADDR_T *loc_addr, ALENGTH_T pktsize, IO_COMPLETE_T *io_complete, ERR_COMPLETE_T *error_complete, PTR_T cookie)

该函数是处理 SNMP 包的入口函数，其中 pktl 指明了包的长度，pktp 指向包的起始地址，for_addr 和 loc_addr 是收到的 SNMP 包的源地址和目标地址，pktsize 指明可以处理的包的最大尺寸，它和 SNMP_MAX_PACKET_SIZE 当中较小的一个被用来控制包的最大尺寸。若有响应包要发送，则调用函数 io_complete 来发送响应包给 manager，否则调用 error_complete 来释放相关资源。

Void SNMP_Continue(SNMP_PKT_T *pktp)

该函数用来判断 SNMP PDU 的中的每个 MIB 变量是否都已处理完，若已处理完，则发送响应包，否则继续处理直到完毕。

为满足项目的需要，需在以下几个方面对 ENVOY AGENT 源码进行二次开发：

1. 编写配置文件，满足项目功能需要。

2. 一般都要定义与项目有关的 MIB 库，因此在 MIB Interface 中需增加该内容的处理，包括：自定义 MIB 文件的创建，编译等。

3. 对标准 MIB 内容的裁减。

4. 需增加一个任务专门处理各种 TRAP 事件，接受其他模块发来的告警信息并转化成标准的 TRAP 结构，通过 UDP 端口 162 发给指定的 Manager

5. 各种 GET, GET NEXT, SET 函数的实现。

6. 修改 makefile.cfg，将需要添加的 MIB 文件、CTL 文件、源代码文件添加进去。

4.6 AGENT 中的数据结构定义

本节描述了关键的几个数据结构，并且在此基础上，设计了 SNMP 的网络连接、异步处理、信息交换、回退处理等几个功能。

4.6.1 AGENT 中消息队列结构

表示 AGENT 中消息队列的基本信息，在创建消息队列时使用。

```
typedef  struct  MessageQueueStruct {
        char     queueName[5];          /* 消息队列的名称 */
```

```
        ULONG    queueId;                /* 消息队列的 ID */
} MessageQueue;
```

4.6.2 管理者链表信息结构

在 AGENT 中必须保留有 MANAGER 的相关信息，这样 AGENT 就可以主动发送 trap 或者请求重要的配置或数据信息（如系统时间等）。

```
typedef  struct  ManagerListInfoStruct {
        UINT        managerId;     /* 管理者的 ID 信息 */
        CommAddr    addr;          /* 管理者的地址 */
        UCHAR       version[3];    /* 管理者的软件版本，主版本号、
次版本号和分发号各占用一个字节 */
        UCHAR       vendor[40];    /* 管理者的生产厂家 */
} ManagerListInfo;
```

4.6.3 通讯地址结构

因为 AGENT 中采用通讯管理模块屏蔽网络通讯协议的差异，因此我们设计一种通用的地址表示方式。

```
typedef  struct  CommAddrStruct {
        char   addrType;                              /* 表示地址的
类型 */
        unsigned char  addrValue[COMM_ADDR_LEN];      /* 表示地址值
*/
} CommAddr;
```

具体的地址表示方式：

A. addrType 为 0 表示与 MANAGER 交互的广播地址，主要用于向 MANAGER 发送 trap，此时 addrValue 没有意义，如果发送 trap 时使用这种地址，CommSendTask 通过一个函数获取所有 MANAGER 的地址，然后向这些 MANAGER 进行发送；如果只是向某个特定的 MANAGER 发送，则使用后面的地址表示方式，设置明确的 MANAGER 地址，在这种情况下，如果采用 TCP 或 UDP 通讯协议，则 addrValue 的 4~5 字节必须设置为端口号，即设置为 UDP_TRAP_PORT 或 TCP_TRAP_PORT。当前不支持广播方式（原因是大家认为 CommSendTask 与应该管理模块交互获取 MANAGER 的通讯地

址，违背了模块化的宗旨，因此现在应用程序要一一构造 trap)。

B. addrType 为 1 表示 TCP/IP 的 IP 地址和端口号，addrValue 的 0~3 字节表示 IP 地址，4~5 字节表示端口号。对于 192.192.1.1:162，addrValue 中的表示为 C0C0010100A2，即此处表示的是 IP 地址和端口号未使用 htonl 和 htons 进行字节顺序转换的形式。

C. addrType 为 2 表示 OSI 的 NSAP 地址（即采用 TP4 通讯方式），addrValue 的 0 字节表示 NSAP 地址的长度，1~addrValue[0]字节表示 NSAP 地址。由于 SNMP 工具包中的 SNMPADDR_T 结构中地址的长度有限，只能有 16 个字节，因此 COMM_ADDR_LEN 的定义是有限制的。

D. addrType 为 3 表示 S 口通讯地址，OADM 系统中槽位号只是在子架内统一编码，因此 addrValue 的 0 字节为子架号，1 字节为槽位号。如果 addrValue[0] 为 0xFF，表示该命令发给所有的单板。

E. addrType 为其它值表示该地址无效。

其中 A、B、C 三种地址表示方式用于 MANAGER 与 AGENT 之间的信息交互，D 地址表示方式用于 NCP 与 MCU 之间的信息交互。

IP 地址与 CommAddr 转换由 GetIPAddrFromCommAddr 和 GetCommAddrFromIPAddr 两个函数来完成。

4.6.4 异步处理注册队列

为了对 SNMP PDU 请求和 MANAGER 下载私有配置数据进行异步处理，我们设计了 AGENT 中必须保存与异步处理有关的信息，同时对于一个异步处理，可能需要发送多条 MCU 命令，因此需要记录两级 requestId，AsyncProcessEntry 中的 cmdRequestId 从 pktp->lcl_ident 获取（实质上是由全局变量 cmdRequestId 循环获取，为了保证异步注册队列中不会出现重复的请求 ID，因此对于 MANAGER 下载私有配置数据并进行异步处理时，也由全局变量 cmdRequestId 来产生 AsyncProcessEntry 中的 cmdRequestId)，sPortRequestIdInfo 中的 requestId 由 sPortRequestId 循环产生；同时对于异步处理，需要加入超时的管理。在命令发出后，每接收到一条响应，根据 cmdRequestId 找到对应的 AsyncProcessEntry，再根据 MCU 响应的 requestId 在 sPortRequestIdInfo 进行查找，如果找到则清除这个单元，这样依次进行处理，当 sPortRequestIdInfo 变为 NULL 时，表示所有异步进行的操作都完成，这时就可以调用 SNMP_Continue 对 pktp 继续处理。如果

在规定的时间（timeOut 的初始值）间隔后 sPortRequestIdInfo 仍然不为空，则表示出现超时。

在生成 MCU 命令请求时，将 cmdRequestId 填写进 S 口命令报文的 CALLID 域，将 MCU 命令的 requestId 填写进 IVH 域，MCU 在返回响应时将这两个信息原样返回。

```
typedef  struct  AsyncProcessEntryStruct  {
        UINT    cmdRequestId;          /* PDU 或 MANAGER 下载私有配置数据的请求 ID */
        SPortRequestEntry   *sPortRequestIdInfo;  /* 记录 MCU 命令的请求 Id */
        USHORT      timeOut;          /* 异步处理的超时时间间隔，以秒为单位，缺省为 DEFAULT_TIMEOUT */
        UCHAR       cmdType;          /* 表示异步处理命令的类型 */
        VB_T        *vbp;             /* 需要进行异步处理的 VB，当处理的命令不是 SNMP PDU 时，设置为 NULL */
        SNMP_PKT_T      *pktp;        /* 需要进行异步处理的 PDU ，当处理的命令不是 SNMP PDU 时，设置为 NULL */
        CommAddr        netAddr;      /* 当下载私有配置数据并进行异步处理时，需要填写 MANAGER 的地址信息，以便处理完毕后将结果返回，对于 SNMP PDU 请求，该域没有意义，因为从 pktp 中可以获得此信息 */
        UCHAR       *privateInfo;     /* 用于存放每个异步处理所需的特定信息，信息的内容随异步处理命令的不同而变化，如果不需要特定信息，则设置为 NULL。针对每个异步处理，需要定义这个信息 */
        struct AsyncProcessEntryStruct *next;   /* 下一个异步处理单元 */
}  AsyncProcessEntry;
```

如果有多个 MANAGER 同时对一个 AGENT 进行管理，这样可能出现不同 PDU 的 request-id 是相同的，有两种方式解决这个问题：第一种方式是 MANAGER 在生成 requestId 时，保证不同 MANAGER 产生的 requestId 绝对不会重复，例如每个 MANAGER 根据其ＩＰ地址或ＭＡＣ地址（或 MANAGER ID）为依据来产生 requestId，此时 AGENT 中不做任何特殊的处理；第二种方式是各个 MANAGER 之间可以产生重复的请求 ID，AGENT 在填写 AsyncProcessEntry 的 cmdRequestId 时，将 MANAGER 的编号加入（例如将 MANAGER 编号填写在 cmdRequestId 的最高字节），MANAGER 的

编号根据 MANAGER 的通讯地址在 managerList 中进行查找（即该 MANAGER 在 managerList 中的顺序）。还有一种方式是采用 SNMP_PKT_T 结构中的 lcl_ident 来进行标识，这是一个提供给用户使用的特定信息，表示的是内部的 PDU 编号。

4.6.5 任务间的信息交换队列

在 AGENT 的实现中，任务之间的信息交换以消息队列的方式进行。在消息发送方，申请 DataBufBetweenTask 的内存并填写其信息内容，然后将 DataBufBetweenTask 的地址发送给接收方；在消息的接收方收到该消息后，完成信息的处理，然后释放 DataBufBetweenTask 所占用的空间。

```
typedef  struct  DataBufBetweenTaskStruct {
        UCHAR     flag;              /* 目前只使用最低位（使用时其它位设置为 0），由它来表示信息传递的方向，0 表示由上到下，1 表示由下到上，AGENT 中任务之间的层次关系约定见图 1 所示 */
        UCHAR     importantLevel;    /* 表示任务之间交换信息的重要程度 */
        CommAddr  addr;              /* 任务之间需要交互的地址信息 */
        UINT      dataExchangeId;    /* 由全局变量 dataExchangeId 循环产生，当网络或 MCU 通讯发送失败时，网络通讯管理模块或 S 口通讯管理模块向应用管理模块发送失败指示，指示中包含 dataExchangeId（就是网络通讯管理模块或 S 口通讯管理模块发送给它的值）和出错代码，这样应用管理模块就可以进行日志记录和回退处理（可以进行回退处理的前提条件是应用管理模块在将信息发送给网络通讯管理模块或 S 口通讯管理模块之前，必须登记与回退处理有关的信息，这些信息中同样也包含 dataExchangeId，dataExchangeId 用来唯一标识一个回退处理信息项）*/
        ULONG     length;            /* 数据的长度，以字节为单位 */
        UCHAR     *buffer;           /* 任务之间交换的信息缓冲区 */
} DataBufBetweenTask;
```

对于网络通讯管理模块与应用管理模块任务之间的信息交互，addr 表示 PDU 请求（或私有上下载命令）的来源地址或目的地址（对于 trap 而言）；对于 S 口通讯管理模块与应用管理模块任务之间的信息交互，表示该命令要发送(或接收到)的单板地址，当然这种情况下，buffer 中也包含了单板地址信息，所以填不填写

addr 都可以，但为了适应以后 S 口通讯协议的变化，最好还是填写和使用 addr 域，而不直接使用 buffer 中的单板地址。

importantLevel 用 LOW_LEVEL、MID_LEVEL 和 HIGH_LEVEL 来标识信息的三种不同重要程度，主要用于应用管理模块与网络通讯管理模块以及应用管理模块与 S 口通讯管理模块任务之间的信息交互，对于应用管理模块发送来的高等级的信息，网络通讯管理模块和 S 口通讯管理模块在信息发送时，应该进行更多次数的发送尝试，发送或建立连接的超时等待时间应该设置更长一些，如果条件允许，还可以与对端接收实体进行发送成功的确认。对于模块内部任务之间的信息交换，importantLevel 没有太大的意义。

4.6.6 视图管理信息结构

在 AGENT 中，需要实现对视图的管理，因此有必要定义一个视图信息结构，包含与视图有关的相关信息，视图使用的原则还是基于群组的管理方式，在使用视图之前，必须用下面的信息创建和安装视图表项。MANAGER 要配置这些信息，当然要定义相关的 MIB。

```
typedef  struct  ViewInfoStruct  {
         USHORT  index;              /* 视图表项的索引，从 1 开始 */
         int       status;              /* 视图表项的状态，取值为
RowStatus，参见 include/epilogue/envoy/h/view.h 中的定义 */
         int      storage;            /* 视图表项的存储状态 */
         int      type;             /* 该视图是 INCLUDED 还是 EXCLUDED */
         char     subtree[MAX_SUBTREE_LEN];  /* 该视图涉及的子树的
根的 OID */
         char     community[MAX_COMMUNITY_LEN];  /* 对应与该视图的
群组名 */
         char     mask[MAX_MASK_LEN];   /* 该视图的掩码，其信息是 16
进制串，字节间用：分割 */
         struct  ViewInfoStruct  *next;  /* 指向下一个视图表项 */
} ViewInfo;
```

4.6.7 回退信息注册结构

为了实现回退处理，当网络或 MCU 通讯发送失败后，网络通讯管理模块或 S 口通讯管理模块向应用管理模块发送失败指示，应用管理模块根据指示中的 dataExchangeId 查找到对应的回退注册信息，然后根据注册的回退信息进行相应的回退处理，并进行日志记录。当然应用管理模块在将信息发送给网络通讯管理模块或 S 口通讯管理模块之前，必须使用 RollBackInfo 登记与回退处理有关的信息。

```
typedef  struct  RollBackInfoStruct  {
        UINT  dataExchangeId;   /* 与 DataBufBetweenTask 中的定义相同 */
        UCHAR  commType;        /* 表示通讯类型，因为通讯分为网络通讯和 S 口通讯，具体的定义见前面全局变量定义中的通讯类型定义 */
        USHORT timeOut;         /* 该回退信息保留的时间 */
        UCHAR  *rollBackInfo;   /* 用于存放每个回退处理所需的特定信息，信息的内容随发送信息的不同而变化，如果不需要特定信息，则设置为 NULL。针对每个回退处理，需要定义这个信息，首先得有一个命令号。 */
        struct  RollBackInfoStruct *next; /* 下一个回退处理信息 */
 }  RollBackInfo;
```

对于回退信息的处理：应用管理模块在登记这个信息并将其发送给网络或 S 口通讯管理模块之后，定期去扫描回退信息注册队列，如果在一定的时间间隔内，没有收到发送失效指示，表示该数据已经成功发送，直接从队列中删除这个注册信息；如果在规定的时间间隔内，接收到了发送失效指示，则进行回退处理和日志记录，同样也要从队列中删除这个注册信息。需要注意一点：有可能出现在登记该注册信息之后，该发送信息在到达 CommSendTask 之前就处理失败了。当前的实现不考虑这种情况。

哪些数据的发送需要登记回退处理信息，在详细设计时根据需要确定，只有对那些重要数据或命令的发送，才进行这种处理，否则如果大量登记这种信息，必然会导致系统运行的效率非常低。向 S 口发送命令时，只对重要的配置和控制命令进行这种处理；网络通讯时，在 MANAGER 从 AGENT 获取日志信息、历史性能、历史告警等信息时，没有必要进行回退信息登记，因为 MANAGER 会定期向 AGENT 采集这些信息。

第五章 结论

SNMP 是实现统一网管的一个简单而实用的协议，其最大的优点就是可以方便的实现统一网管。RFC1901—RFC1908 详细规定了进行 SNMP 操作时应该遵循的准则，包含 MIB 库的对象应该怎样定义，报文格式是怎样的，不同的操作原语该怎样去响应。从而为不同的厂家实现 SNMP 协议提供了具体的实现方式。SNMP 协议由于他的通用性，使不同厂家的设备和网管可以方便的互相互控管理。由于他的操作原语比较简单，所以实现起来也很方便。任何一种设备，只要他提供 SNMP 标准接口而且我们又知道他的 MIB 库定义时，就可以很快的实现对这种设备的管理。

另外 ENVOY 成熟的 SNMP 软件开发包也给我们提供了方便的途径实现 SNMP 协议，软件包实现了协议的大部分内容，并且为用户提供了私有函数的入口，用户所需要做的工作就是实现各种操作原语的回调函数，具体的编码、解码，MIB 树的查找，系统的总体调度都由开发包实现，从而可以快速开发出用户所需的产品。

从理论上说，用 SNMP 来实现网管应该是一种很好的方式，现在市场上也有很多支持 SNMP 的产品。SNMP 协议 V3 版本的推出弥补了以往的 SNMP 协议对安全性考虑不足的缺点。从而使这种协议日趋完善。

但是从客观上来说，我认为 SNMP 协议不太适用我们这样有大量数据的传输网络管理。

SNMP 协议开始是从管理 INTERNET 网络发展起来的，其思路肯定是最适用与 INTERNET 网络的管理，在这种网络上，需要读取和设置的信息量都不是很大。

为了使操作与具体数据的具体含义没有关系，在 SNMP 中，每一个数据，无论是简单变量还是表格中的某个字段都可以单独进行操作，也就是说，在 SNMP 中，虽然存在表格的概念，但表格的数据是可以独立操作的，用户可以只读取或者设置表格中的某一个数据，但对于我们的系统，有时候表格的数据是不应该被拆离开的。拆开的数据对于设备来说是不完整的。

再有，每一个数据在发送时也是完全独立的，为了实现这种独立性，每一个数据的报文结构里都增加了很多附加信息，以指明这个变量在 MIB 中的具体位置，操作状态等信息，通过 UDP 方式发送时，由于 MTU 包的限制，一包中最大可以包含 60 个数据。对于我们这样有大量数据需要设置和读取的系统来说，这种格式对系统效率产生很大影响。执行速度肯定不如私有网管快。

通过对这个项目的开发，我觉得如果以后有类似的系统采用 SNMP 方式进行管理时，应该注意以下几点：

首先，SNMP 网管一般只对所有数据进行读取操作，而对其中很少部分的数据进行设置操作。而私有数据配置时具体应该采用什么样的协议没有任何限制，针对我们这种系统的具体情况，通过私有的方式下载数据是最方便的处理方式。所以可以完全使用私有的协议进行数据的设置，需要 SNMP 网络管理的部分再采用 SNMP 协议，这样做可以减少程序的复杂性，同时加快项目的进度。

另外如果希望完全采用 SNMP 进行程序的开发，则应该仔细分析公有 MIB 库和私有数据格式之间的差别，尽可能从上到下采用与 MIB 库相同的格式，减少转化操作，使程序得到简化。另外，最好不要出现 MIB 中的一个数据需要到多个表格中取信息的情况，这样处理起来很麻烦。网管也采用一套，通过团体字的不同来区分是私有操作还是统一网管的操作。

致谢

在论文的最后，我要向所有帮助过我的人致以诚挚的感谢！

首先向我的导师黄迪明教授致以最诚挚的谢意。他渊博的学识、严谨的治学态度、精益求精的工作作风和诲人不倦的为师风范，让我受益匪浅，并将激励我在今后的学习和工作中不断进取。在论文的写作过程中，他一直对我悉心指导、严格要求，并在百忙之中为我审阅论文。在此谨向黄迪明老师表示衷心的感谢和真诚的敬意。

感谢中兴通讯本部事业部传输业务软件开发部的所有同事，在和他们一起工作中我学到了很多知识，增长了见识。

感谢我的三位师傅：吴涛，刘健军和管冬根，与他们的讨论，使我解开了许多心中的疑惑。在论文的写作过程中，他们提出了许多有益的见解和宝贵的意见，并给予了许多帮助。

感谢我宿舍的所有舍友，与他们的联系，他们的帮助和支持，使得我顺利地完成了论文及毕业相关的所有手续，使我得以顺利地进行毕业答辩。

衷心感谢我的家人，他们对我多年来的关心与支持使我得以顺利地完成了学业。

最后，向所有帮助我的人表示衷心的感谢，并诚挚地感谢为评阅本论文而付出辛勤劳动的各位专家和学者！

参考文献

[1] 张宁，胡文雯. 网络管理的现状及面向业务的综合网络管理. 北京：现代电信科技,2003：26-28.

[2] 胡古雨. 现代通信网和计算机网管理. 北京：电子工业出版社，1996：13.

[3] 杨骏. 光传送网的管理. 北京：北京邮电大学出版社，2001：31.

[4] 张勐，魏笔凡. 计算机网络的组织与管理. 北京：人民邮电出版社，2000.11：545-550.

[5] Wisniewski S(詹文军 译). 高级网络管理. 北京：高等教育出版社，2006.6：175-184.

[6] Kurose J, Ross K(申震杰 译). 计算机网络-自顶向下方法与 Internet 特色. 北京：清华大学出版社，2003：547-556.

[7] 顾尚杰，薛质. 计算机通信网基础. 北京：电子工业出版社，2000：18-19.

[8] 岑贤道，常安青. 网络管理协议及应用开发. 北京：清华大学出版社,1998：26-27.

[9] 杨放春. 智能化现代通信网. 北京：北京邮电大学出版社，1999：17-18.

[10] 姚予疆. 通讯设备接口协议手册 北京：人民邮电出版社，2005.2：618-621.

[11] Nwana H. Software agent：an overview .Knowledge Engineering Review, 1996, 11(3):205-244.

[12] Wooldridge M J. Agent -Based software engineering . IEEE Transactions on Software Engineering，1997，144(1)：26-37

[13] 李智鹏，方昌始.基于 SNMP 的网络设备代理 Agent 的设计与实现.福建电脑，2007，1 ：127-128.

[14] 安晓嵘，李孝安.分布式网络管理系统中 SNMP Agent 的研究.科学技术与工程，2006，6(11)：1565-1568.

[15] Komer B(彭业飞 译). TCP/IP 网络管理 北京：电子工业出版社，1999.4：230-240.

[16] Stevens W R（范建华译）. TCP/IP 详解（卷 1）北京：机械工业出版社，2000：270-279.

[17] 王志文. 现代网络管理模型分析. 计算机工程与应用，1999，10(12)：46.

[18] Harnedy S. Total SNMP: Exploring the Simple Network Management Protocol (2nd Edition). New Jersey：Prentice Hall Inc,1998：24—26.

[19] Cardoso P F, Monteiro J L. SNMP and Industrial Networks[C]. Industrial Electronics Society, IECON '98, Proceedings of the 24th Annual Conference of the IEEE, 1998：242-246.

[20] 李雄伟，孙大跃. 基于 SNMP 网络管理系统的研究与开发. 网络安全技术与应用，2007，6：41-44.

[21] 姜飞，史浩山，徐志燕. 一种基于 SNMP 协议的主代理 / 子代理通信机制. 计算机工程，2007，33(21) ：81-83.

[22] Wijnen B，Harrington D，Presuhn R. An architecture for describing SNMP management frameworks. RFC2571，1999.4.

[23] Wijnen B，Presuhn R，McCloghrie K. View-based access control model (VACM) for the simple network management protocol(SNMP). RFC2575，1999.4.

[24] Harrington D, Presuhn R, Wijnen B. An Architecture for Describing SNMP Management Frameworks [EB/OL], 1999. 4.

[25] Koerner E. Design of a proxy for managing CMIP agents via SNMP[J]. Computer communications，1997.

[26] William. SNMP，SNMPv2，and RMON practical network management[second edition]. New York：Addison - wesley publishing company, 1996.

[27] ITU-T X.208：Specification of Abstract Syntax Notation One (ASN.1).

[28] IETF RFC1901：Introduction to Community-based SNMPv2.

[29] IETF RFC1902：Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2).

[30] IETF RFC1903：Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2).

[31] IETF RFC1904：Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2).

[32] IETF RFC1905：Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2).

[33] IETF RFC1906：Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2).

[34] IETF RFC1907：Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2).

[35] IETF RFC1908：Coexistence between Version 1 and Version 2 of the Internet- standard Network Management Framework.

[36] IETF RFC2558：Definitions of Managed Objects for the Sonet/SDH Interface Type.

[37] ITU-T G.872：Architecture of Optical Transport Networks.

[38] IETF RFC2493：Textual Conventions for MIB Modules Using Performance History Based on 15 Minute Intervals.

[39] ITU-T G.874：Management Aspects of Optical Transport Network Element.

[40] ITU-T G.875：Optical Transport Network (OTN) Management Information Model for the Network Element View.

攻硕期间取得的研究成果

发表论文：

[1] 嵌入式系统异常处理模块设计. 电子科技大学研究生学报计算机科学与技术增刊，2008.3，No.30