José Messias Alves da Silva

Construção de Agentes SNMP em Ambientes Linux

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. DSc. Fernando Cortez Sica

Lavras
Minas Gerais - Brasil
2005

José Messias Alves da Silva

Construção de Agentes SNMP em Ambientes Linux

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em *17 de Abril de 2005*

Prof. Sandro Pereira Melo

Prof. Samuel Pereira Dias

Prof. DSc. Fernando Cortez Sica
(Orientador)

Lavras
Minas Gerais - Brasil

Agradecimentos

Talvez, tão difícil quanto fazer esta monografia é encontrar as palavras certas para agradecer a todos que tornaram mais simples e/ou mais produtiva a realização desta monografia. Espero que nesses agradecimentos eu possa retribuir, ao menos, um pouco da atenção que me foi dada.

Ao meu Orientador Prof. DSc Fernando Cortez Sica por seu apoio, paciência, sugestões, amizade e orientação segura, fator importante na realização deste trabalho.

Aos meus familiares. Com certeza, aqui faltam palavras para expressar o meu agradecimento.

Aos meus amigos João Almeida e Alexandre Jorge pelo incentivo e sugestões.

Aos meus amigos da turma ARL2003s2 que mesmo a distância, souberam me incentivar.

Aos meus amigos pela amizade e pelo incentivo.

Ao Professor DSc. Francisco Vieira de Souza e Professor MSc. Magno Alves dos Santos que muito me ajudaram na minha formação acadêmica, pessoas que eu posso certamente chamar de amigos.

A todos, a minha gratidão e muito obrigado.

Agradeço sobretudo a Deus por mais esta caminhada que estou concluindo com sucesso em minha vida.

Resumo

Este trabalho apresenta um estudo sobre protocolo de gerenciamento *Simple Network Management Protocol* (SNMP), através de informações que servem para entendimento dos conceitos básicos de gerenciamento de redes, expondo também as características e vantagens que este tipo de controle de rede pode oferecer, bem como a forma de implementar agentes SNMP para dispositivos de uma rede Local Área Network (LAN), utilizado várias ferramentas.

Abstract

This work presents a study about of the management protocol *Simple Network Management Protocol* (SNMP), through of informations that serves for understanding of basics concepts of management networks, shows too the characteristics and advantages what this type of network control can offers, as well the form of implementation agents SNMP for devices of a Local Area Network (LAN), using several tools.

Não basta ensinar ao homem uma especialidade,
porque se tornará assim uma máquina utilizável e
não uma personalidade.
É necessário que adquira um sentimento, um
senso prático daquilo que vale a pena ser empreendido,
daquilo que é belo, do que é moralmente correto.
(Albert Einstein)

Lista de Siglas e Abreviaturas

ACL – *Access Control List*
API – *Application Program Interface*
ASCII – *American Standard Code for Information Interchange*
ASN.1 – *Abstract Syntax Notation One*
CCITT – *Consultative Committe for International Telegraph and Telephone*
CMIP – *Common Management Information Protocol*
CMIS – *Common Management Information Service*
CMISE – *Common Management Information Service Element*
CPU – *Central Processor Unit*
DES – *Data Encryption Standard*
DoS – *Denial of Service*
EGP – *Exterior Gateway Protocol*
FDDI – *Fiber Distributed Data Interface*
FTP – *File Transfer Protocol*
GUI – *Graphic User Interface*
IEEE – *Institute of Electrical and Electronics Engineers*
IP – *Internet Protocol*
ISO – *International Organization for Standardization*
ITU-T – *International*
HTML – *Hipertext Markup Language*
HTTP – *Hipertext Transfer Protocol*
IAB – *Internet Activities Board*
IBM – *Industrial Business Machine*
IETF – *Internet Engineering Task Force*
IP – *Internet Protocol*
ITU-T – *International Telecommunication Union- Telecommunications Standardization Sector*
LAN – *Local Area Network*
MAN – *Metropolitan Area Network*
MD5 – *Message Digest Algorithm 5*
MIB – *Management Information Base*
MTU – *Maximum Transmission Unit*
NMS – *Network Management Station*
LAN – *Local Area Network*
OID – *Object Identifier*
OSI – *Open Systems Interconnection*

PC – *Personal Computer*
PDU – *Protocol Description Unit*
RMI – *Remote Method Invocation*
PARC – *Palo Alto Research Center*
RFC – *Request for Comments*
RMON – *Remote Network Monitoring MIB*
RPC – *Remote Procedure Call*
SHA – *Secure Hash Algorithm*
SMI – *Structure of Management Information*
SMTP – *Simple Mail Transfer Protocol*
SNA – *System Network Architecture*
SNMP – *Simple Network Management Protocol*
TCP – *Transmission Control Protocol*
UDP – *User Datagram Protocol*
WAN – *Wide Area Network*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As informações que circulam em uma rede de computadores devem ser transportadas de modo confiável e rápido. Para que isso aconteça é importante que os dados sejam monitorados de maneira que os problemas que porventura possam existir sejam resolvidos rapidamente. Uma rede sem mecanismos de gerência pode apresentar problemas como congestionamento do tráfego, recursos mal utilizados, recursos sobrecarregados, problemas com segurança e outros.

Assim, a necessidade de gerenciar falhas, configuração, contabilização, desempenho e segurança tem incentivado o desenvolvimento de várias ferramentas nos últimos anos. Uma categoria que se destaca é a de ferramentas com a finalidade de monitorar e fornecer estatísticas detalhadas sobre o tráfego nesses ambientes, para que se possa, por exemplo, identificar gargalos e caracterizar o tráfego. Essas ferramentas se baseiam na análise individual dos pacotes que trafegam na rede, e acabam se limitando a contabilizar o tráfego, classificando-o por remetente, destinatário, protocolo utilizado, entre outros critérios. Ao utilizar uma ferramenta como essa, torna-se difícil identificar problemas relacionados ao comportamento de um protocolo de alto nível, medir o tempo de resposta de determinada transação e detectar a presença de intrusos na rede.

Esta crescente necessidade de gerenciamento fez com que padrões para ferramentas fossem estabelecidos. Em resposta a esta necessidade surgiram dois padrões:

Família de Protocolos SNMP: o protocolo SNMP (*Simple Network Management Protocol*) refere-se a um conjunto de padrões para gerenciamento que inclui um protocolo, uma especificação de estrutura de dados, e um conjunto de objetos de dados. Este protocolo hoje já está na sua terceira versão oficial, chamada de SNMPv3. Este é o protocolo de gerencia adotado como padrão para redes TCP/IP, e que será tratado neste trabalho.

Sistemas de gerenciamento OSI (*Open Systems Interconnection*): este termo refere-se a um grande conjunto de padrões de grande complexidade, que definem aplicações de propósito geral para gerencia de redes, um serviço de gerenciamento e protocolo, uma especificação de estrutura de dados, e um conjunto de objetos de dados. Este conjunto de protocolos é conhecido como CMIP [Wil96]. Pela sua complexidade, e pela lentidão do processo de padronização, este sistema de gerenciamento não e muito popular.

1.1 Objetivos

1.1.1 Objetivo Geral

O trabalho desenvolvido tem como objetivo estudar e aplicar o protocolo de gerenciamento SNMP através da especificação e implementação de uma aplicação agente SNMP, para o gerenciamento de dispositivos de uma rede LAN. Assim, também, aprender os conceitos, protocolos, ferramentas e técnicas utilizados na gerência de uma rede de computadores.

1.1.2 Objetivos Específicos:

- Entender a necessidade da gerência de redes e as áreas nas quais a gerência de redes pode ser decomposta.
- Entender a arquitetura genérica empregada em soluções de gerência de redes de computadores.
- Entender a funcionalidade básica dos componentes utilizados na gerência de redes, incluindo plataformas e aplicações de gerência.
- Entender a solução SNMP de gerência de redes, a mais largamente utilizada no mercado, incluindo o modelo de informação, as MIBs mais importantes e o funcionamento do protocolo SNMP.
- Entender como agentes e gerentes são implementados na arquitetura SNMP
- Aprender a especificar uma solução de gerência de redes

1.2 Metodologia

O trabalho foi desenvolvido utilizando os materiais e os métodos descritos a seguir:

- Foi realizado um levantamento bibliográfico, na *internet* e em bibliotecas, de artigos científicos clássicos e atuais relacionados ao tema; em paralelo, foi realizado um estudo de como seria a gerência de redes, propriamente dita.
- Também foi realizado um estudo sobre os impactos do uso de agentes SNMP no mundo. As mudanças que estão ocorrendo, o que está sendo afetado;
- Ao término destas etapas, foi dado início a um estudo mais detalhado de como desenvolver e escrever MIBs e agentes.
- Ao final, foram utilizadas algumas ferramentas que facilitam a criação desses elementos de gerência, tais como Net-SNMP e MRTG.

1.3 Descrição dos Capítulos Posteriores

Este trabalho apresenta, no capítulo 2, os principais conceitos refentes ao gerenciamento de uma redes de computadores, aplicações, estrutura e objetivos.

O capítulo 3 apresenta a definição e as características principais das MIBs, um dos principais objetos de estudo desta monografia, direcionando-se para a sua construção, definição de tabelas e seus tipos. O capítulo 4 discorre de forma mais aprofundada a origem, as características e a utilização do protocolo de gerenciamento SNMP, agentes e gerentes, mensagens e seus tipos, encapsulamento e transmissão, sempre buscando enfatizar as suas qualidades e deficiências.

O capítulo 5 abrange o desenvolvimento de agentes SNMP na prática, enfocando a sua implementação em Ambientes Linux, instalação, configuração e uso de ferramenta, exemplos de utilização, monitoramento de obtenção e respostas de mensagens.

O capítulo 6 enfoca a utilização da ferramenta MRTG como auxilío ao monitoramento e exibição de dados através da visualização de relatórios dos dispositivos em que foram implementados SNMP.

O capítulo 7 é dedicado a alguma discussões que envolvem o desenvolvimento de SNMP atualmente, vantagens e desvantagens, limitações e sobretudo no que se refere à segurança deste protocolo. Também, este capítulo reserva-se a explanação de conclusões, relato de contribuições e o anseio de desenvolvimento de trabalhos futuros.

Capítulo 2

Gerenciamento de Redes

2.1 Introdução

Os sistemas de rede comerciais existentes atualmente estão se tornando extremamente complexos. Partindo de algumas poucas redes locais isoladas, esses sistemas se expandiram em LAN's que operam ao longo de corporações inteiras. Roteadores conectam LAN's à escritórios remotos e redes de alcance mundial se tornam cada vez mais presentes. Monitorar e fazer manutenção desses sistemas, para não mencionar problemas isolados, pode ser um desafio.

Entre as atividades básicas do gerenciamento de redes estão a detecção e correção de falhas, em um tempo mínimo, e no estabelecimento de procedimentos para a previsão de problemas futuros. Por exemplo, monitorando linhas cujo tráfego esteja aumentando ou roteadores que estejam se sobrecarregando, é possível tomar medidas que evitem o colapso da rede, como a reconfiguração das rotas ou troca do roteador por um modelo mais adequado.

2.2 Aplicações de Gerenciamento de Redes

Devido à intensa pressão colocada sobre os administradores de rede para que estabeleçam um grau de controle sobre a vasta quantidade de produtos LAN heterogêneos que florescem nos sistemas de computação atuais, os sistemas de gerenciamento de rede passaram a merecer uma consideração maior.

Sistemas de gerenciamento de rede empregam um *software* que controla e administra uma rede de uma simples estação de trabalho. Alguns sistemas são extremamente simples, usam um sistema básico e são fáceis de operar, dada sua interface amigável. O problema com a simplicidade é que, às vezes, determinados aspectos da rede podem ser sacrificados.

2.3 Estrutura de Gerenciamento

Hoje em dia existe uma grande diversidade de ambientes de rede podendo se encontrar tanto um *mainframe* IBM com rede SNA (*System Network Architecture*, arquitetura proprietária da IBM) como uma rede local com ambiente Linux. Diante de tal heterogeneidade, a dificuldade de se projetar um gerenciamento de rede pode ser grande, implicando o uso de várias ferramentas inseridas em uma estrutura possivelmente complexa e com limites de atuação definidos entre os componentes envolvidos.

Essa estrutura deve ser capaz de estabelecer a estratégia empregada no atendimento à chamadas dos usuários, assim como a atuação do pessoal envolvido nas tarefas de gerenciamento de rede, além de definir os supridores de serviços, inclusive externos e outros aspectos da rede. Portanto,é fundamental que haja organização, sendo que alguns aspectos como o atendimento aos usuários demonstram ser primordiais para o sucesso da estrutura. Também, é desejável que haja para o usuário um único ponto de contato para reportar problemas e mudanças.

Os limites de atuação da gerência devem considerar a amplitude desejada pelo modelo já instalado que, além de operar a rede, deve englobar, dentre outros, os seguintes itens :

- Controle de acesso à rede
- Disponibilidade e desempenho
- Documentação de configuração
- Gerência de mudanças
- Planejamento de capacidades
- Auxílio ao usuário
- Gerência de problemas
- Controle de inventário

A ênfase relativa atribuída em cada uma dessas categorias por uma instalação depende do tamaho e complexidade da rede.

De um ponto de vista técnico, observa-se que as redes de computadores estão em constante expansão, tanto fisicamente como em complexidade. Entretanto, para o usuário final a rede é vista como algo muito simples, ou seja, apenas supridor de ferramentas que facilitam suas atividades cotidianas.

Desta forma, para o usuário, os sistemas de computação devem ser capazes de auxiliá-lo a atingir vários objetivos tais como vendas, qualidade, eficiência, não sendo importante a forma como se obtém tais resultados. Entretanto, as dificuldades se concentram em como avaliar o impacto de uma eventual parada no computador central, se a paralisação for apenas parcial ou total ou apenas uma linha ou uma workstation. Este e outros aspectos devem ser abordados dentro do processo de elaboração de um modelo para gerenciamento de redes, visto que constituem o alicerce para a elaboração da estrutura.

2.4 Forma de Gerenciamento

De acordo com [DW], um sistema de gerenciamento é composto por entidades que participam do processo obtendo informações ou fornecendo informações. Em geral, centraliza-se o poder para coleta de informações a um gerente e agentes ficam responsáveis por enviarem informações a este gerente. Na figura 2.1, pode-se ter uma visão de como funciona um sistema de gerenciamento de redes, a comunicação entre aplicações e objetos gerenciados, que serão detalhados mais adiante.

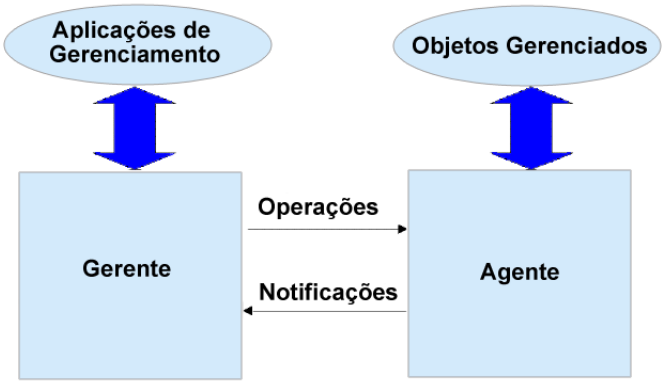

Figura 2.1: Modelo típico de um Sistema de Gerência de Redes

De uma maneira geral, as mensagens partem do gerente para os objetos gerenciados, obtendo destes as iformações necessárias ao gerenciamento da rede. Pode-se observar, na figura 2.2, como se dá a comunicação entre gerente e agente, os tipos de mensagens que fluem entre essas duas entidades.

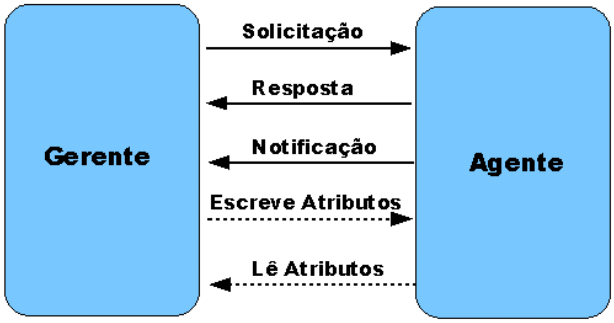

Figura 2.2: Mensagens em um Protocolo de Gerência de Redes

2.5 Considerações Finais

O conteúdo apresentado neste capítulo visa familiarizar o leitor com as principais idéias relacionadas ao gerencimento de redes. Apresentu-se também os conceitos de gerência de redes, estrutura de um gerenciamento de redes. É claro que o conhecimento literário descrito neste material é apenas uma introdução, sendo aconselhável a leitura de outros autores, para um maior embasamento sobre o conteúdo. Contudo, o conhecimento heurístico do especialista adquirido pela vivência em ambientes de redes também é de grande importância para a compreensão.

Nos próximos Capítulos serão apresentadas as informações relativas a definição da base de informações gerenciais, um estudo sobre o protocolo SNMP e sobre ferramentas de desenvolvimento de agentes SNMP e visualização de dados.

Capítulo 3

Base de informação gerencial (MIB)

3.1 Introdução e Definição de objetos na MIB

Um dos fatores mais importantes em sistema de gerenciamento é a forma como as informações sobre os elementos de rede estão armazenadas. Tais informações precisam estar disponíveis segundo um determinado padrão para que possam ser reconhecidas e utilizadas por qualquer aplicação.

Os objetos são imagens virtuais dos elementos básicos que se quer monitorar. Um objeto gerenciado é a visão abstrata de um recurso real do sistema. Assim, todos os recursos da rede que devem ser gerenciados são modelados, e as estruturas dos dados resultantes são os objetos gerenciados.

Os objetos gerenciados podem ter permissões para serem lidos ou alterados, sendo que cada leitura representará o estado real do recurso e cada alteração também será refletida no próprio recurso.

Os objetos são definidos usando a *Abstract Syntax Notation One* (ASN.1), que será melhor explicando na subseção 3.3.1 e estão localizados na MIB (*Management Information Base*).

Dessa forma, a MIB é o conjunto dos objetos gerenciados, que procura abranger todas as informações necessárias para a gerência da rede.

Até o presente momento, foram definidos basicamente quatro tipos de MIBs, a MIBI, a MIBII, a MIB experimental e a MIB privada.

- **MIB I** - As MIBs do tipo I fornecem informações gerais sobre os elementos da rede, sem levar em conta as características específicas dos equipamentos.
- **MIB II ou MIB Padrão** - é considerada uma evolução da MIB I, fornece

informações gerais de gerenciamento sobre um determinado equipamento gerenciado. É possível através das MIBs I e II obter informações como tipo e status da interface (*Ethernet, FDDI, ATM*), número de pacotes transmitidos, número de pacotes com erro, endereço IP das rotas, etc. Possui um conjunto de objetos bem definidos, conhecidos e aceitos pelos grupos e padrões Internet;

- **MIB Experimental** - aquela em que seus componentes (objetos) estão em fase de desenvolvimento e teste, em geral, eles fornecem características mais específicas sobre a tecnologia dos meios de transmissão e equipamentos empregados. Estas MIBs podem conter informações específicas sobre outros elementos da rede e gerenciamento de dispositivos que são considerados importantes.

- **MIB Privada** - As MIBs privadas ou proprietárias foram elaboradas com o objetivo de atuar sobre um equipamento em específico, possibilitando que detalhes característicos do mesmo possam ser obtidos. Desta forma, é possível obter informações sobre colisões, configuração, swap de portas, e muitas outras, de um roteador. Também é possível fazer testes, reinicializar ou desabilitar uma ou mais portas do roteador atravém das MIBs privadas.

De acordo com [RM01], no modelo SNMP, os recursos de uma rede são representados como objetos. Cada objeto é, essencialmente, uma variável que representa um aspecto do dispositivo gerenciado. Todos os objetos (variáveis) são armazenados na MIB.

3.1.1 Objetos MIB

Para usar o SNMP efetivamente, usuários precisam estar familiarizados com a Base de Informação do Gerenciador SNMP que define todos os valores de leitura e alteração que o SNMP é capaz. Para começar um administrador tem que conhecer a MIB do SNMP do equipamento ou da rede que estiver gerenciando. A MIB é organizada em uma estrutura de árvore, semelhante a uma estrutura de diretório de arquivos em disco. O topo do nível SNMP começa com o diretório que contém quatro níveis principais: O nível *mgmt*, que normalmente contém os objetos padrões do SNMP utilizados por todos os dispositivos da rede; o nível *private* que contém os objetos especializados definidos pelos fabricantes de equipamentos de rede; os níveis *extended* e *directory* que são normalmente destituídos de quaisquer dados ou objetos significantes [RM01].

3.1.2 Objetos Discretos da MIB

Muitos objetos de SNMP são discretos, o operador necessita apenas saber o nome do objeto da MIB e nenhuma outra informação. Objetos discretos representam freqüentemente valores sumários de um dispositivo, particularmente útil por apresentar informação da rede com a finalidade de comparar desempenho de dispositivos de rede. Se a extensão (número de instância) do objeto não é especificada, pode ser assumido como ".0" (ponto-zero). [Yor]

3.1.3 Tabelas de Objetos da MIB

Tabela de objetos contêm múltiplas partes de dados do gerenciador. Estes objetos são distintos dos itens *Discrete* adicionando uma extensão "." (ponto) para os nomes deles que distinguem o valor particular que é referenciado. A extensão "." (ponto) se refere ao número da "instância" de um objeto do SNMP. No caso do objeto *Discrete*, este número de instância será zero. No caso da "Tabela" de objetos, este número de instância será o índice na tabela do SNMP. Tabelas de SNMP são tipos especiais de objetos de SNMP que permitem aglutinar de forma estruturada um conjunto de informações. Por exemplo, SNMP define o objeto *ifDescr* como um objeto padrão do SNMP que indica a descrição de texto de cada interface apoiada por um dispositivo particular. Considerando que podem ser configurados dispositivos de rede com mais de uma interface, este objeto só poderia ser representado como um array [RM01].

Por convenção, objetos SNMP se agrupam sempre em um diretório de "Entrada", dentro de um objeto com uma "Tabela" (o objeto *ifDescr* objeto descrito acima, reside no diretório *ifEntry* que é armazenado no diretório *ifTable*. Há várias regras para desenvolver objetos SNMP, como segue [RM01]:

- ao criar no diretório de "Entrada" de uma tabela tem que conter o mesmo número de elementos como outros objetos no mesmo diretório, onde o número de uma instância é o mesmo de todas as entradas. Sempre são considerados objetos da tabela como ordens paralelas de dados;

- quando criando um novo objeto de "Entrada", o SNMP requer que um valor tenha sido associado com cada entrada da tabela em uma única mensagem de SNMP (único PDU). Isto significa que, para criar uma fila em uma tabela (pelo comando *set* no SNMP), um valor deve ser especificado para cada elemento na fila;

- se uma fila da tabela pode ser apagada, o SNMP requer pelo menos que para isso um objeto na entrada tenha um elemento de controle, que é documen-

tado para executar a exclusão da tabela. (Isto só se aplica se uma fila pode ser excluída, que necessariamente não é requerido de uma tabela do SNMP).

3.1.4 Tipos de Objetos de uma MIB

Segundo [Dav97], os objetos da MIB têm um tipo específico de valores. O SNMP define vários tipos primitivos, como se segue.

Tipo de Objetos de Texto

Define-se o tipo ***DisplayString*** que pode conter informação textual arbitrária (normalmente para um máximo de 256 caracteres). O texto deve conter somente caracteres de impressão. Exemplos deste tipo de objeto incluem valores *sysDescr* e *ifDescr*. Este tipo de objeto é bastante utilizado nas MIB.

Tipo de Objetos Contadores

Define-se o tipo *Counter* que é um valor numérico que só pode aumentar. Este é o tipo mais comum de objeto de SNMP na MIB padrão e inclui objetos como *ipInReceives*, *ipOutRequests*, e *snmpInPkts*. Contadores ultrapassam o valor de máximo da variável, mas nunca podem ser negativos.

Tipo de Objetos de Medida

Define-se o tipo *Gauge* que é um valor numérico que pode aumentar ou pode diminuir. Este tipo é encontrado em apenas algumas localizações dentro da MIB padrão. Exemplos deste tipo de objeto incluem o objeto *tcpCurrEstab*.

Tipo de Objetos Inteiros

Define-se o tipo *Integer* que pode conter valores positivos ou negativos. Este valor normalmente é fornecido por valores de tipo *Counter* ou *Gauge*, mas às vezes é expresso em MIB de equipamentos de fabricantes.

Tipo de Objetos Enumerados

Define-se um tipo *Enumerated Value* que associa um campo textual com um valor numérico. Este tipo é bastante comum na MIB padrão e inclui objetos como *ifAdminStatus*, cujo valores enumerados, são *up(1)*, *down(2)*, e *testing(3)*. Geralmente os valores enumerados são formatados da seguinte forma *nome(número)*.

Tipo de Objetos Tempo

Define-se um tipo *TimeTicks* que representa um tempo decorrido. Este tempo sempre tem uma resolução de um centésimo de segundo, até mesmo quando esta resolução não é usada. Administradores de rede freqüentemente formatam este valor de tempo como **HH:MM:SS:cc** para exibição. Por exemplo, o valor *sysUpTime* indica o tempo decorrido desde que um dispositivo foi reiniciado.

Tipo de Objetos Objeto

Define-se um tipo *Object* que pode conter o identificador para outro objeto SNMP. Se o objeto nomeado é compilado na MIB, o nome geralmente é apresentado com o mesmo nome no objeto SNMP. Um exemplo deste tipo de objeto é a variável *sysObjectID* da MIB.

Tipo de Objetos Endereço IP

Define-se um tipo *IP address*, que contém o Endereço IP de um dispositivo de rede. Este tipo de objeto é exibido freqüentemente como um endereço de IP convencional. Exemplos deste tipo de objeto incluem *ipRouteDest*, *ipRouteNextHop* e *ipRouteMask*.

Tipo de Objetos Endereço Físico

Define-se um tipo *Physical address*, que contém o endereço físico de um dispositivo de rede. Gerentes exibem freqüentemente este tipo de objeto como uma sucessão de valores hexadecimal, precedidos pela palavra-chave **hex:**, e separados através de dois pontos. Exemplos deste tipo de objeto incluem o objeto *ifPhysAddress*.

Tipo de Objetos String

Define-se um tipo *String*, que contém uma cadeia de caracteres arbitrários. Quando a cadeia de caracteres contém só caracteres ASCII, os gerentes exibem este valor como uma string de texto. Caso contrário, gerentes exibem este tipo como uma sucessão de valores hexadecimal, precedidos pela palavra-chave **hex:**, e separados através de dois pontos. Este tipo de valor é incomum, mas ocasionalmente encontrado em MIB proprietárias.

Tipo de Objetos Tabela

O tipo *Table* é um objeto que contém entradas da tabela. Este tipo de objeto sempre é um nome intermediário que contém um diretório de *Entrada* e que contém vários objetos da tabela.

Tipo de Objetos Auxiliares

O tipo *Branch* é um objeto que contém tipos adicionais, tabelas, ou qualquer tipo de objeto listado acima.

3.2 Acesso a Valores da MIB

Cada objeto SNMP é definido para ter um tipo de acesso **somente de leitura**, **leitura escrita** ou **apenas escrita**. Isso determina se o usuário pode ler o valor de objeto, pode ler e pode escrever o objeto (com um comando *set*) ou só escrever o objeto [Inc].

Antes que qualquer objeto possa ser lido ou possa ser escrito, o nome comunitário do SNMP deve ser conhecido. Estes nomes comunitários são configurados pelo administrador, e podem ser vistos como senhas necessárias para anexar dados ao SNMP. No sentido exato, nomes comunitários existem para permitir que partes da MIB no SNMP, e subconjuntos de objeto sejam referenciados. Como o termo **Comunitário** é requerido, espera-se que o verdadeiro propósito destes valores sejam identificar comunitariamente entre os objetos SNMP configurados. Porém, é prática comum fazer estes nomes comunitários limitarem o acesso da capacidade do SNMP para usuários sem permissão.

3.3 Construção de MIBs

As regras de construção das estruturas da MIB são descritas através da *Structure of Management Information - SMI*. Os objetos de uma MIB são especificados de acordo com a ASN.1 - *Abstract Syntax Notation One*, uma linguagem introduzida pela CCITT (hoje ITU-T) nas Recomendações X208 e X209, e escolhida pela ISO para descrever os tipos de dados usados em SNMP.

A notação sintática abstrata é uma forma de descrição abstrata dos dados com o objetivo de não se levar em consideração a estrutura e restrições do equipamento no qual está sendo implementada [Dav97].

A estrutura de informações de gerência SMI é um conjunto de documentos que definem:

- Forma de identificação e agrupamento das informações;
- Sintaxes permitidas;
- Tipos de dados permitidos.

As instâncias do objeto são chamadas de variáveis.

Os objetos são definidos segundo um determinado tipo (*Object Type*). Esses tipos possuem cinco campos:

- **Nome (*Name*)** - É um nome textual, normalmente curto, para o tipo de objeto denominado Descritor de Objeto, o qual corresponde à um identificador de objeto.
- **Identificador (*Object Identifier*)** - é o identificador do objeto, é formado por números que são separados por pontos.
- **Sintaxe (*Sintax*)** - É uma sintaxe abstrata para um tipo de objeto. Pode ser uma sintaxe construída reunindo tipos básicos ou uma sintaxe de aplicação tais como Network Adress, IpAddress ou Time Ticks. De uma maneira geral, pode ser:
  - uma sintaxe do tipo simples que pode ser um inteiro, uma string de octetos, um Object Identifier ou nulo;
  - pode ser também uma sintaxe de aplicação podendo ser um endereço de rede, um contador, uma medida, um intervalo de tempo ou incompreensível.
- **Definição** - É uma descrição textual da semântica de um tipo de objeto. Trata-se de um campo de grande importância para MIBs que pretendam ser usadas em um ambiente com fabricantes distintos.
- **Acesso** - É o tipo de controle que se pode ter sobre o objeto, podendo ser: somente leitura, leitura e escrita ou não acessível.
- ***Status*** - Pode ser obrigatório, opcional ou obsoleto.
- **DescrPart** - Descrição textual do objeto.
- **ReferPart** - Referência a outro objeto de outra MIB.
- **IndexPart** - Usado na definição de tabelas.
- **DefValPart** - Valor default quando o objeto é criado.

3.3.1 Tipos ASN.1

O Padrão ASN.1 que define a construção de MIBs possui alguns tipos básicos que são de enorme importância para a perfeita simbolização. Este tipos dividem-se em Primitivos, Definidos e Construtores:

Tipos Primitivos

- INTEGER - inteiro com sinal, como por exemplo, o número de interfaces em um sistema.
- OCTET STRING - uma *String* que é usada para representar um dado hexadecimal, como o endereço físico de uma interface. São *DisplayString*, *octetString*, *PhysAddress*.
- OBJECT IDENTIFIER - uma String de números derivados de uma árvore nomeada, utilizada para identificar um objeto.
- NULL - usado em *GetRequest PDU*, que será melhor detalhada mais adiante.
- ENUMERATED - um conjunto limitado de inteiros com um significado associado.
- BOOLEAN - um inteiro com valores *true (1)* ou *false (2)*.

Tipos Definidos

Alguns tipos de dados, específicos para aplicações SNMP, foram acrescentados para compor os tipos primitivos. Entre outros, estão incluídos neste conjunto:

- *NetworkAddress*
- *IpAddress*
- *Counter* (inteiro que incrementa até um valor máximo e em seguida retorna a zero)
- *Gauge* (inteiro que cresce e descresce)
- *TimeTicks*
- *Opaque*

No SNMPv2 foram adicionados alguns outros tipos de dados:

- *BIT STRING* - lista enumerada de *flags*

- *Integer32* - idêntico ao INTEGER, com intervalo de -2^{31} a $2^{31} - 1$
- *Counter32* - idêntico ao COUNTER, com intervalo de 0 a $2^{32} - 1$
- *Gauge32* - idêntico ao GAUGE, com intervalo de 0 a $2^{32} - 1$
- *NsapAddress* - para endereços OSI
- *Counter64* - intervalo de 0 a 2^{64}
- *Uinteger32* - inteiro sem sinal, com intervalo de 0 a $2^{32} - 1$

Tipos Construtores

Os tipos construtores permitem definir estruturas complexas a partir de tipos primitivos. Os construtores usados em SNMP são:

- SEQUENCE - Define as linhas de uma tabela. Cada entrada em SEQUENCE é uma coluna. É uma lista ordenada de tipos de dados.
- SEQUENCE OF - Também define as linhas de uma tabela. É uma lista ordenada de tipos de dados do mesmo tipo, como por exemplo, uma sequência de inteiros.

Definição de Tipos de Dados em ASN.1

A Definição de tipo de dados segundo ASN.1 são escritos da seguinte forma:

```
DatatypeName ::= DEFINITION
```

Por convenção, o primeiro caractere de um nome tipo de dado é uma letra maiúscula, como por exemplo:

```
RequestID ::= INTEGER
```

Definição de Tipos de Dados Construtores em ASN.1

Para exemplificar a definição de tipos de dados construtores, a figura **??** mostra que o tipo de dado *Message* é formado por três campos, um INTEGER, um OCTET STRING e um terceiro tipo chamado PDU.

```
Message ::=
        SEQUENCE {
                version         INTEGER { version -1(0) },
                community       OCTET STRING,
                pdu             PDU,
        }
```

Figura 3.1: Exemplo de Construção de Mensagem

3.3.2 Macros

Macros são mecanismos que auxiliam na definição dos objetos da *MIB*, além de permitirem a expansão do ASN.1. O SNMP utilizar esta característica da linguagem para fornecer informações completas sobre os objetos em uma *MIB*, como nome do objeto, que tipo será o objeto, restrições de valores, operações permitidas para o objeto como somente leitura (*read-only*), leitura e escrita (*read-write*), ou não-acessível (*not-accessible*), descrição de informações sobre o objeto. Para tanto, naa definição de *MIBs* tem-se o uso extensivo da macro *OBJECT-TYPE*.

Algumas dessas informações podem ser observadas na figura **??**

```
ipAdEntReasmMaxSize OBJECT–TYPE
        SYNTAX INTEGER  (0..65535)
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION
                " The size of the largest IP datagram which this
                entity can re-assemble from incoming IP fragmented
                datagrams received on this interface."
        ::= { ipAddrEntry 5}
```

Figura 3.2: Exemplo de construção de OBJECT-TYPE

De acordo com a definição mostrada na figura **??**, tem-se que:

- O *OBJECT IDENTIFIER* para *ipAdEntReasmMaxSize* é *{ipAddrEntry 5}*, que pode ser obtido percorrendo a árvore como **1.3.6.1.2.1.4.20.1.5**.
- A sintaxe *SYNTAX* , ou tipo de dado, para os possíveis valores de um *ipAdEntReasmMaxSize* é *INTEGER*.

- O inteiro está no intervalo de 0 a 65535. O acesso *ACCESS* informa que tipo de operação pode ser realizar sobre o objeto. Para este caso, a estação de gerenciamento pode somente ler o valor, não pode atualizá-lo.
- O estado *STATUS mandatory* informa ao implementador que esta variável deve ser suportada.
- A descrição *DESCRIPTION* informa que o valor dessa variável é o tamanho do maior datagrama que pode ser reconstruído a partir de fragmentos obtidos pela *interface*.

Pode-se ter também módulos, que agrupam objetos com funcionalidades correlatas, formando uma *MIB*. A construção *MODULEIDENTITY* permite que objetos relacionados entre si sejam agrupados. Além das definições *OBJECTTYPE*, a construção *MODULEIDENTITY* contém informações de contato com o autor do módulo, a data da última atualização, um histórico de revisões e uma descrição do módulo.

```
ipMIB MODULEIDENTITY
      LASTUPDATED
          "200503160000Z"
      ORGANIZATION "DM-UFPI"
      CONTACTINFO
          "Messias Alves, jmessias@ufpi.br "
      DESCRIPTION
           "The MIB module for managing IP
           and ICMP implementations, but
           excluding the management of
           IP routes."
      REVISION "200503040000Z"
      DESCRIPTION
          "The initial revision of this MIB
           module was part of MIBII."
::= {mib2 48}
```

Figura 3.3: Exemplo de construção MODULEIDENTITY

As palavras mais usadas na construção de *MIBs* são *BEGIN*, *DEFINITIONS*, *END*, *EXPORTS*, *IDENTIFIER*, *IMPORTS*, *INTEGER*, *NULL*, *OBJECT*, *OCTET*, *OF*, *SEQUENCE*, *STRING* e alguns símbolos especiais muito comuns são :, -, –, ::=, | .

3.4 Estrutura

A árvore hierárquica mostrada pela figura 3.4 foi definida pela ISO representa a estrutura lógica da MIB, mostra o identificador e o nome de cada objeto.

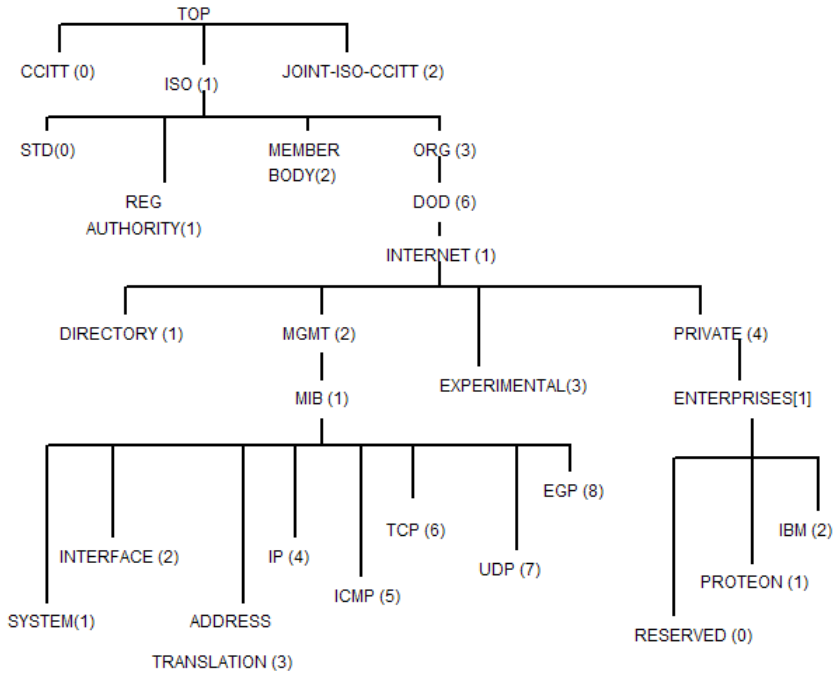

Figura 3.4: MIB

O nó raiz da árvore não possui rótulo mas possui pelo menos três subníveis, sendo eles: o nó 0 que é administrado pela *Consultative Committe for International Telegraph and Telephone - CCITT*, atualmente ITU-T ; o nó 1 que é administrado pela *International Organization for Standartization - ISO*; o nó 2 que é administrado em conjunto pela CCITT e pela ISO. Sob o nó ISO fica o nó que pode ser utilizado por outras instituições: o org (3), abaixo dele fica o dod (6) que pertence ao departamento de defesa dos EUA. O Departamento de Defesa dos EUA alocou

um sub-nó para a comunidade internet, que é administrado pela *International Activities Board* - IAB e abaixo deste nó temos, entre outros, os nós: management, experimental e private.

Sob o nó *management* ficam as informações de gerenciamento, é sob este nó que está o nó da MIB II.

Sob o nó *experimental* estão as MIBs experimentais.

Sob o nó *private* fica o nó *enterprises* e sob este nó ficam os nós das indústrias de equipamentos.

Como exemplo de um objeto, pode-se citar o *ipInReceives* pertencente ao grupo IP:

ipInReceives *Object Type*

Object Identifier: 1.3.6.1.2.1.4.3

Access: read-only

Syntax*: Counter32*

Description: O número total de datagramas que chegam nas interfaces, incluindo aqueles com erro.

3.4.1 MIB II

Conforme pode-se observar na figura 3.4, a MIB II é organizada a partir do nó mgmt (*management*). Abaixo da subárvore MIB II estão os objetos usados para obter informações específicas dos dispositivos da rede. Esses objetos estão divididos em 10 grupos, que estão presentes na tabela 3.1, obtida de [Dav].

Tabela 3.1: Grupos da MIB II

Grupo	Informação
system (1)	informações básicas do sistema
interfaces (2)	interfaces de rede
at (3)	tradução de endereços
ip (4)	protocolo ip
icmp (5)	protocolo icmp
tcp (6)	protocolo tcp
udp (7)	protocolo udp
egp (8)	protocolo egp
transmission (10)	meios de transmissão
snmp (11)	protocolo snmp

A planificação do nó da MIB II fica como está descrita na figura 3.5.

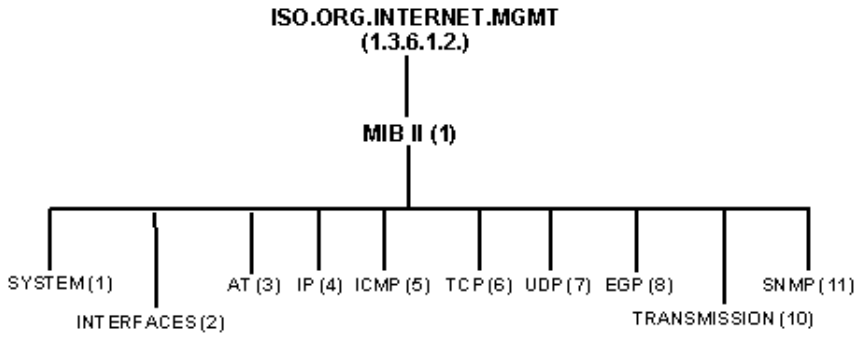

Figura 3.5: MIB II

3.4.2 Exemplos de Objetos da MIB II

Alguns dos objetos pertencentes aos grupos da MIB II são descritos a seguir:

Grupo *System*(1.3.6.1.2.1.1) - Define uma lista de objetos pertencentes à operação do sistema, como o tempo de funcionamento, contato e nome do sistema.

- *sysDescr* (1.3.6.1.2.1.1.1) : Descrição textual da unidade. Pode incluir o nome e a versão do hardware, sistema operacional e o programa de rede.
- *sysUpTime* (1.3.6.1.2.1.1.3): Tempo decorrido (em milhares de segundos) desde a última reinicialização do gerenciamento do sistema na rede.
- *sysContact* (1.3.6.1.2.1.1.4): Texto de identificação do gerente da máquina gerenciada e como contatá- lo.

Grupo *Interfaces* (1.3.6.1.2.1.2) - Rastreia o status de cada interface em uma entidade gerenciada. O grupo interfaces monitora as interfaces em funcionamento ou inativas e rastreia aspectos, como octetos enviados e recebidos, erros e eliminações.

- *ifNumber* (1.3.6.1.2.1.2.1): Número de interfaces de rede (não importando seu atual estado) presentes neste sistema.
- *ifOperStatus* (1.3.6.1.2.1.2.2.1.8): Estado atual da interface.
- *ifInOctets* (1.3.6.1.2.1.2.2.1.10): Número total de octetos recebidos pela interface.

Grupo IP (1.3.6.1.2.1.4) - Rastreia os diversos aspectos do IP, incluindo o roteamento do IP.

- *ipForwarding* (1.3.6.1.2.1.4.1): Indica se esta entidade é um gateway.
- *ipInReceives*(1.3.6.1.2.1.4.3): Número total de datagramas recebidos pelas interfaces, incluindo os recebidos com erro.
- *ipInHdrErrors* (1.3.6.1.2.1.4.4): Número de datagramas que foram recebidos e descartados devido a erros no cabeçalho IP.

Grupo ICMP (1.3.6.1.2.1.5) - Rastreia aspectos como erros do ICMP.

- *icmpInMsgs* (1.3.6.1.2.1.5.1): Número total de mensagens ICMP recebidas por esta entidade. Incluindo aquelas com erros.
- *icmpOutMsgs* (1.3.6.1.2.1.5.14): Número total de me nsagens ICMP enviadas por esta entidade. Incluindo aquelas com erros.

Grupo TCP (1.3.6.1.2.1.6) - Rastreia, entre outros aspectos, o estado das conexões TCP (como closed, listen, sysSent, etc).

- *tcpMaxConn* (1.3.6.2.1.6.4): Número máximo de conexões TCP que esta entidade pode suportar.
- *tcpCurrentEstab* (1.3.6.2.1.6.9): Número de conexões TCP que estão como estabelecidas ou a espera de fechamento.
- *tcpRetransSegs* (1.3.6.2.1.6.12): Número total de segmentos retransmitidos.

Grupo UDP (1.3.6.1.2.1.7) - Rastreia dados estatísticos do UDP.

- *udpInDatagrams*(1.3.6.1.2.1.7.1): Número total de datagramas UDP entregues aos usuários UDP.
- *udpNoPorts* (1.3.6.1.2.1.7.2): Número total de datagramas UDP recebidos para os quais não existia aplicação na referida porta.
- *udpLocalPort* (1.3.6.1.2.1.7.5.1.2): Número da porta do usuário UDP local.

Grupo SNMP (1.3.6.1.2.1.11) - Avalia o tráfego SNMP.

- *snmpInPkts* (1.3.6.1.2.1.11.1): Número total de mensagens recebidas pela entidade SNMP.

- *snmpOutPkts* (1.3.6.1.2.1.11.2): Número total de mensagens enviadas pela entidade SNMP.
- *snmpInTotalReqVars* (1.3.6.1.2.1.11.13): Número total de objetos da MIB que foram resgatados pela entidade SNMP.

3.5 A MIB RMON

A especificação RMON, *Remote MONitoring*, define funções padrão de monitoramento de rede e interfaces para comunicação entre dispositivos baseados em SNMP.

A RMON fornece às redes a capacidade de oferecer uma maneira eficiente e efetiva de monitorar o comportamento de subredes e ao mesmo tempo de reduzir a carga tanto em outros agentes como estações.

A MIB RMON utiliza um dispositivo do agente conectado à uma rede espalhada para coletar estatísticas de tráfico de rede. A MIB RMON também realiza cálculos diretamente no agente e não deixa a cargo do gerenciador para todas as suas funções. Tipicamente, um agente é somente responsável pela informação de gerenciamento que se relaciona com o seu próprio equipamento Sem um função de monitoramento remoto, se torna difícil, se não impossível, para um gerente construir um perfil de qualquer atividade em uma subrede individual.

A MIB RMON pode ser implementada diretamente nas aplicações hoje existentes e não requer que todo o SNMP v.2 seja usado. Para ser efetiva,deve-se prover uma estação dedicada de gerenciamento utilizando RMON e capacidade do agente deve estar ligada à LAN central. Os agentes RMON ficam residentes em dispositivos que monitoram cada subrede nas quais eles estão ligados, dessa forma dando ao gerente um monitoramento da camada de rede. Este monitoramento inclúi operação off-line, detecção de problemas e reportagem, dados de valores adicionados e suporte à múltiplos gerentes.

O RMON é empregado para estudar o tráfego em uma rede como um todo, superando as limitações da MIB-II onde um gerente SNMP pode obter apenas informações localizadas em um determinado equipamento. Tipicamente, estes monitores (também chamados de probes) operam em uma rede no modo promíscuo, visualizando todos os pacotes que passam através dela.

A gerência de redes remotas através de agentes remotos (por exemplo, agentes que implementam a MIB RMON) possui cinco funções:

- **Operações off-line**: são as operações nas quais uma estação de gerenciamento não necessita estar em contato direto com seus dispositivos de monitoração remotos. Essa função na MIB RMON permite que os agentes sejam

configurados para realizar diagnósticos e coletar estatísticas continuamente, mesmo que a comunicação entre a estação de gerenciamento não seja possível ou não seja eficiente;

- **Monitoração pró-ativa**: os recursos disponíveis nos monitores são potencialmente úteis para continuamente executar diagnósticos e manter logs do desempenho da rede. Essas informações são importantes para desenvolver a função baseline. A função baseline refere-se ao fato de manter um histórico da operação normal de uma rede por um tempo estendido, com o objetivo dessas informações posteriormente serem analisadas para identificar problemas potenciais numa rede;

- **Detecção e registro de problemas**: o monitor remoto pode fazer o reconhecimento de determinadas condições, realizando constantes averiguações;

- **Valorização dos dados coletados**: o monitor pode executar análises específicas nos dados coletados em suas subredes;

- **Múltiplos gerentes**: na configuração de uma rede pode haver mais de uma estação de gerenciamento como forma de oferecer maior nível de disponibilidade, para executarem diferentes funções ou ainda para gerenciar diferentes departamentos em uma empresa.

3.5.1 Grupos da MIB RMON

A MIB RMON está dividida em nove grupos:

- **Grupo *Statistics*** - esse grupo contém estatísticas medidas pelo monitor para cada interface monitorada no dispositivo.

- **Grupo *History*** - esse grupo registra amostras estatísticas periodicamente e as armazena para uma posterior recuperação.

- **Grupo *Alarm*** - esse grupo periodicamente obtém amostras estatísticas de variáveis do monitor e as compara com os limiares previamente configurados;

- **Grupo *Host*** - esse grupo contém estatísticas associadas a cada host descoberto na rede. Os endereços fonte e destino desses hosts são mantidos numa lista e obtidos dos pacotes bons que foram promiscuamente recebidos pela interface.

- **Grupo *HostTopN*** - esse grupo é usado para preparar relatórios que especificam os principais hosts de uma lista ordenada por uma de suas estatísticas;
- **Grupo *Matrix*** - esse grupo armazena a estatística do tráfego e número de erros entre pares de hosts.
- **Grupo *Filter*** - esse grupo permite que os pacotes sejam capturados com uma expressão de filtro arbitrária;
- **Grupo *Capture***: esse grupo permite que os pacotes sejam capturados sobre a correspondência de um filtro e que o sistema de gerenciamento crie múltiplos buffers de captura, controlando se o buffer de monitoração (trace buffer) continua ou interrompe a captura de pacotes quando estiver cheio;
- **Grupo *Event***: esse grupo controla a geração e notificação de eventos;

3.6 Compilador de MIB

Uma MIB pode ser compilada por um compilador de MIBs de forma que as informações presentes na MIB estejam disponíveis para aplicações como MIB *browsers* e *graphers*. São aplicações simples que obtém toda a sua capacidade de gerenciamento através da análise de uma MIB, sem intervenção humana.

Além de checar a sintaxe de uma MIB, um compilador de MIBs pode gerar automaticamente as estruturas de dados e o código necessários para que um agente implemente uma determinada MIB. Um compilador de MIBs também pode fazer com que as informações sobre os objetos gerenciados de MIBs proprietárias ou de novas MIBs que sejam padronizadas estejam disponíveis para uma aplicação de gerenciamento existente.

A entrada para um compilador de MIBs é uma coleção de módulos de MIBs escritos em um subconjunto da linguagem ASN.1. Estes módulos contêm definições de objetos gerenciados que correspondem às informações sobre os dispositivos da rede que podem ser manipulados através do protocolo SNMP. Os compiladores de MIBs podem gerar várias representações das definições dos objetos gerenciados contidos nas MIBs usadas como entrada. Estas representações podem ser processadas mais facilmente pelos agentes e aplicações de gerenciamento do que a representação ASN.1.

Algumas destas representações são declarações de estruturas de dados em linguagens de programação de alto nível, como C, que podem ser compiladas e ligadas em uma aplicação de gerenciamento ou agente. Outras são arquivos de dados contendo representações das definições dos objetos gerenciados que podem ser lidas para a memória por uma aplicação de gerenciamento ou agente em tempo de

execução. Em alguns casos, o compilador de MIBs gera um código de saída que auxilia na implementação das MIBs de entrada. Por exemplo, um compilador de MIBs pode gerar esqueletos de rotinas para a recuperação ou alteração do valor de um objeto gerenciado, ou rotinas para a geração de *Trap-PDUs* específicas.

A habilidade de reconhecer as descrições presentes em uma MIB mecanicamente é muito atraente, principalmente para os fabricantes de aplicações genéricas, pois estas podem cobrir uma grande variedade de agentes de MIBs. Com o grande número de MIBs padronizadas e MIBs proprietárias disponíveis atualmente, os compiladores de MIBs reduzem o esforço dos fornecedores para manterem suas aplicações atualizadas.

Atualmente os programas de gerenciamento apenas se limitam a coletar e exibir informações sobre os elementos da rede, sem entretanto analisá-las. Assim o papel de compreender o estado atual da rede e de encontrar soluções para os problemas cabe mesmo ao administrador. Esse aspecto dificulta muito a compreensão de MIBs desconhecidas relativas à um dispositivo desconhecido.

3.7 Considerações Finais

Neste capítulo foram apresentados alguns conceitos relacionados a construção de bases de informações gerenciais (MIBs), definição de objetos e acesso a valores, estrutura e tipos de MIBs. Também foram descritas as principais características da linguagem ASN.1 e realizado um detalhado estudo sobre os grupos pertencentes a algumas MIBs, como MIBII e RMON. No próximo capítulo são apresentadas algumas definições, conceitos e operações pertinentes ao protocolo de gerenciamento SNMP.

Capítulo 4

Protocolo de Gerenciamento SNMP

4.1 Introdução e Origens

A necessidade de mecanismos de gerenciamento nas redes baseadas em TCP/IP é atendida pelo SNMP (*Simple Network Management Protocol*) em associação com o esquema de MIB (Management Information Base). A *Internet Activities Board(IAB)*, o orgão que rege a política da Internet e o protocolo TCP/IP, requisitou um comitê para rever as opções de gerenciamento de redes. O comitê concluiu que o SNMP deveria ser adotado. Esse protocolo é baseado no *Simple Gateway Management Protocol*(SGMP) que havia sido desenvolvido para administrar redes regionais.

O comitê também começou a trabalhar em um protocolo futuro chamado de *Common Management Information Protocol* (CMIP). A idéia em torno do SNMP era de que ele seria um remédio rápido até que o CMIP estivesse pronto para uso. Isso foi em 1988.

Alguns dos objetivos e especificações no projeto do SNMP foram :

- **Gerenciamento de rede integrado** – A capacidade de gerenciar redes incorporando componentes que venham de uma variedade de fabricantes com uma simples aplicação.

- **Interoperabilidade** – A capacidade de que um equipamento de um vendedor seja gerenciado pelo equipamento de outro vendedor.

- **Padronização** – Padrões definem métodos de comunicação e estruturas de

dados de forma que redes não similares possam ser integradas com o gerenciamento de rede.

- **Custos** – Reduzir o custo da construção de um agente que suporte o protocolo;
- **Tráfego** – Reduzir o tráfego de mensagens de gerenciamento pela rede necessárias para gerenciar dos recursos da rede;
- **Restrições** – Reduzir o número de restrições impostas as ferramentas de gerenciamento da rede, devido ao uso de operações complexas e pouco flexíveis;
- **Simplicidade de Operações** – Apresentar operações simples de serem entendidas, sendo facilmente usadas pelos desenvolvedores de ferramentas de gerenciamento;
- **Atualização** – Permitir facilmente a introdução de novas características e novos objetos não previstos ao se definir o protocolo;
- **Independência** – Construir uma arquitetura que seja independente de detalhes relevantes à somente a algumas implementações particulares.

O Simple Network Management Protocol (SNMP) é um protocolo da camada de aplicação, como mostrado na Figura 4.1, desenvolvido para facilitar a troca de informações de gerenciamento entre dispositivos de rede. Estas informações transportadas pelo SNMP (como pacotes por segundo e taxa de erro na rede), permitem aos administradores da rede gerenciar o desempenho da rede de forma remota, encontrando e solucionando os problemas e planejar o crescimento da rede.

4.2 Características do protocolo SNMP

Normalmente, é utilizado uma aplicação na máquina do administrador chamado de cliente que se conecta a um ou mais servidores SNMP localizados em máquinas remotas, para executar operações sobre os objetos gerenciados.

O cliente não percebe quando as operações estão sendo executadas pelo agente sobre os objetos, o que fornece transparência ao protocolo, o que já não ocorre com outros protocolos de gerência como CMIP, por exemplo.

O SNMP também realiza um processo de autenticação para permitir ou não o acesso de clientes aos objetos gerenciados.

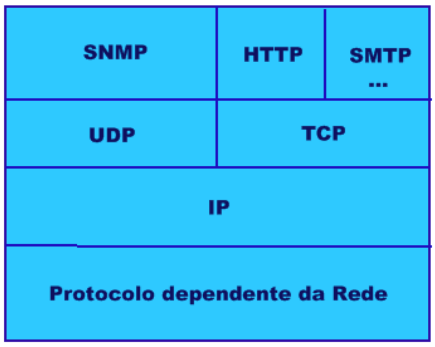

Figura 4.1: Localização do Protocolo SNMP no TCP/IP

A principal característica do SNMP é a simplicidade. Ao invés de apresentar muitos comandos como outros protocolos, ele possui apenas um pequeno conjunto de operações com funções básicas de busca/alteração.

Através do protocolo SNMP, o cliente enviará comandos com duas funções básicamente: Uma de obtenção dos valores dos objetos (função **GET**) e outra de alteração desses valores (função **SET**). É ainda previsto um mecanismo de notificação de alterações nos objetos da MIB (**TRAP**). Tal estrutura torna o protocolo simples,flexível e estável, pois mantém um formato básico fixo,mesmo que novos objetos sejam implementados ou mesmo que novas operações sejam definidas, o que poderá ser feito utilizando as operações básicas.

No envio e recepção de mensagens no protocolo, os nomes dos objetos não devem ser expressos na forma textual, mas sim na forma numérica que representa univocamente às diversas instâncias existentes, para que se torne o pacote da mensagem SNMP mais compacto. Assim, conforme foi explicado na seção 3.4, o objeto gerenciável **iso.org.dod.internet.mgmt.mib.ip.ipInReceives** será representado na mensagem SNMP como **1.3.6.1.2.1.4.3**. Quando a forma numérica do identificador terminar com zero, significa que o objeto é a única instância existente.

4.2.1 Estações de gerenciamento SNMP

A estação de gerenciamento SNMP é uma coleção de aplicações e banco de dados que controlam um grupo de agentes. Uma estação de gerenciamento de rede é composta por 5 componentes, sendo uma interface para o usuário, aplicações de gerenciamento, um banco de dados, um dispositivo SNMP e um canal de transporte/ligação.

A interface do usuário permite ao operador mandar comandos de gerencia-

mento e receber do agente respostas solicitadas ou não. Tal interface poderia ser em formato texto ou em algum tipo de interface gráfica para o usuário (*Graphic User Interface - GUI*).

As aplicações de gerenciamento operam na análise e processamento da informação de gerenciamento de rede obtida do agente.

O banco de dados, ou variáveis de interesse, contém todos os nomes, configurações, performances, topologia e dados examinados da rede. O banco de dados é separado em categorias que incluem a Management Information Base (MIB), o banco de dados do elemento de rede e o banco de dados da aplicação de gerenciamento.

4.3 Aplicações SNMP

Vários produtos têm surgido com a finalidade de gerenciar a rede, quase que em sua totalidade baseados no padrão SNMP e CMIP. O sucesso do SNMP se deve ao fato de ele ter sido o primeiro protocolo de gerenciamento acessível ao público, não proprietário e simples em sua implementação, o que possibilita o gerenciamento efetivo de ambientes com características não similares.

A implantação dos protocolos SNMP foi introduzida pelos fornecedores de *gateways*, *bridges* e roteadores. Normalmente, o fornecedor desenvolve o agente SNMP e, posteriormente, uma aplicação de gerenciamento para a estação gerente, sendo que tais produtos funcionam perfeitamente em ambientes GNU/Linux. Em sua realização, incorporam funções gráficas para o operador do centro de controle e incluem muitas vezes bibliotecas e utilitários que permitam a criação de aplicações de gerenciamento com características específicas para alguns componentes da rede.

As implementações básicas do SNMP permitem ao gerente monitorar e isolar falhas, já as aplicações mais sofisticadas permitem gerenciar o desempenho e a configuração da rede. Estas aplicações, normalmente, incorporam menus e alarmes para melhorar a interação com o profissional de gerência.

4.4 Elementos do SNMP

O SNMP possui uma característica cliente/servidor, onde o cliente seria o gerenciador da rede e o servidor o agente SNMP e a base que modela este agente é a MIB.

4.4.1 Agentes

No modelo de gerenciamento SNMP, *hosts*, *bridges*, *roteadores*, *hubs*, etc, devem ser equipados com agentes SNMP para que possam ser gerenciados pela estação de gerenciamento (*Network Management Station - NMS*) através do gerente SNMP. O agente responde a requisições da estação de gerenciamento, que pode ser o envio de informações de gerência ou ações sobre as variáveis do dispositivo onde está.

O funcionamento desta estrutura só é possível graças ao acesso direto à MIB que o agente possui, pois todas as informações de gerência encontram-se lá [Dav97]. Ao receber uma mensagem SNMP do gerente, o agente identifica que operação está sendo requisitada e qual(is) a(s) variável(is) relacionada, e a partir daí executa a operação sobre a MIB. Em seguida, monta uma nova mensagem de resposta, que será enviada ao gerente.

À primeira vista, a comunicação do agente com o gerente pode parecer injusta uma vez que o agente apenas responde ao que lhe é questionado. Mas há momentos em que o agente pode"falar" espontaneamente com o gerente sem que antes seja questionado. Isso ocorre quando o agente detecta, a partir da análise do contexto da MIB, alguma situação inesperada. Neste momento, o agente gera uma mensagem especial, o *Trap*, e a envia ao gerente, relatando sobre a situação.

Para o tratamento destas exceções e o envio de *Trap*, é concedido ao agente um certo poder de decisão, cabendo a ele, a partir da análise do contexto da MIB, decidir se é ou não necessário enviar o*Trap* ao gerente. Esse poder de decisão é concedido ao agente para que em certas situações, como quando da inicialização do sistema, *Trap* desnecessários não sejam trafegados pela rede, o que, em se tratando de dezenas de agentes, poderia interferir no desempenho global da rede.

Cabe ao agente um papel fundamental em todo o processo de gerenciamento da rede, acessando e disponibilizando informações de gerência contidas na MIB, além de indicar situações inesperadas de funcionamento do dispositivo que estiver gerenciando através do envio de *Trap* ao gerente.

4.4.2 Gerentes

A interface entre as aplicações de gerência correntes no NMS e os agentes espalhados pelos dispositivos da rede é o gerente. Cabe ao gerente enviar comandos aos agentes, solicitando informações sobre variáveis de um objeto gerenciado ou modificando o valor de determinada variável.

Os gerentes então processam estas informações colhidas pelos agentes e as repassam à aplicação que as requisitou. A comunicação entre o gerente e as aplicações é possível através da utilização das API do gerente SNMP pelo sistema.

A API (*Application Program Interface*) é um conjunto de funções que fazem o intermédio na execução de comandos entre um programa e outro, de forma a simplificar a um programa o acesso a funções de outro programa e que, no caso do SNMP, fazem o intermédio das execuções entre uma aplicação de gerência e o gerente SNMP [And04].

Quando uma solicitação da aplicação de gerência chega ao gerente, através da API, o gerente mapeia esta solicitação para um ou mais comandos SNMP que, através da troca de mensagens apropriadas, chegarão aos agentes correspondentes. Deve-se ressaltar que este comando SNMP montado pelo gerente pode ser enviado a um outro gerente, que pode ou não repassá-lo a um agente. Só que a comunicação gerente-gerente só é possível na versão estendida do SNMP, o **SNMPv2**, e não faz parte do SNMP padrão.

Cabe também ao gerente encaminhar à aplicação de gerência os *Trap* que porventura sejam enviados pelos agentes. Assim, o software de gerência terá conhecimento da presença de um novo equipamento na rede ou do mal funcionamento de algum dos dispositivos da rede.

Quando da inicialização do software de gerência, nada se sabe sobre a configuração ou funcionamento da rede. Essas informações vão sendo conhecidas através de *Trap* que são enviados pelos agentes. A partir daí o gerente realiza *polling*[1] [And04] a fim de manter a comunicação com os agentes, possibilitando ao software de gerência mapear, monitorar e controlar a rede.

4.5 Operações SNMP aplicáveis às Variávies da MIB

No gerenciamento SNMP existem várias operações para a comunicação entre os gerentes e agentes SNMP para obter informações dos dispositivos gerenciados.

Get

O gerente SNMP envia o comando *Get* a um determinado agente toda vez que necessita recuperar uma informação de gerenciamento específica do objeto gerenciado pelo agente. Estas informações encontram-se na forma básica de variáveis, que por sua vez estão na MIB do elemento de rede gerenciado.

[1] Forma de gerenciamento de redes em que o gerente requisita informações que se encontram nos agentes)

GetNext

O comando *GetNext* assemelha-se ao comando *Get*, no entanto, enquanto o comando *Get* solicita ao agente a leitura de determinada instância de uma variável, ao receber um comando *GetNext*, o agente deve ler a próxima instância disponível, na ordem especificada pela MIB, da(s) variável(is) associada(s).

Esta operação é especialmente utilizada na recuperação de uma tabela de instâncias da MIB cujo tamanho seja desconhecido. Para isto, é inicialmente especificado no comando *GetNext* o nome da tabela, o que retornará como resposta o primeiro item (neste caso, uma instância de variável) da tabela. Com o nome do primeiro item da tabela, o gerente pode enviar um outro *GetNext*, desta vez passando a resposta do *GetNext* anterior como parâmetro, a fim de recuperar o próximo item da tabela, e assim sucessivamente até recuperar todas as instâncias da variável na tabela. Para este caso, se o comando *Get* fosse utilizado ele falharia, pois o nome da tabela não corresponde a uma instância individual da variável.

Set

A operação *Set* requisita a um determinado agente a atribuição/alteração do valor de determinada(s) variável(is) de uma MIB. Alguns desenvolvedores, por exemplo, acreditam que este comando não deve retornar um Response. Já outros acham que a operação Set deve retornar alguma indicação de que a operação foi efetuada. Porém o mais correto seria que após cada operação Set sobre uma variável, uma operação Get fosse efetuada sobre a mesma variável a fim de assegurar que a operação Set foi efetuada.

Trap

A operação *Trap* difere de todas as outras. Ela é utilizada por um agente SNMP para notificar de forma assíncrona a um gerente algum evento extraordinário que tenha ocorrido no objeto gerenciado.

Diversos questionamentos são feitos quanto a esta operação. Talvez o maior deles seja sobre a escolha dos eventos que devem realmente ser notificados ao gerente. Embora seja quase unânime que o gerente deve ser informado de alguns eventos significativos, muitos fornecedores de produtos que implementam o SNMP trazem *Trap's* específicos, muitos deles desnecessários [Wil96].

Outro importante questionamento é como um agente pode ter certeza de que o gerente o recebeu, visto que a operação *Trap* não gera um *Response*, o que pode ocorrer quando, por exemplo, a máquina estiver perdendo pacotes. Isso não é fácil de solucionar. Uma possibilidade seria o agente gerar tantos *Trap's* quanto

necessário até que o gerente seja informado do evento. Essa solução, no entanto, aumentaria o tráfego na rede, afetando o seu desempenho.

Responses

Como já descrito, sempre que um agente recebe um comando *Get*, *GetNext* ou Set ele tenta executar a operação associada e, conseguindo ou não, constrói uma outra mensagem que é enviada ao emissor da requisição. Esta mensagem é a *GetResponse*. Das operações SNMP, apenas o *Trap* não gera um *Response*.

4.5.1 Características Extensíveis

As características extensíveis do SNMP permite ao usuário estender definições de novas MIB para o sistema. Estas definições normalmente são fornecidas por fabricantes de equipamento de rede, especialmente formatada usando a sintaxe ASN.1, que conforme vista na seção 3.3, é uma linguagem de declaração abstrata adotada pelo SNMP.

A MIB de um dispositivo SNMP é normalmente fixa. Ela é desenvolvida pelos fabricantes de equipamentos de rede (i.e. fabricante de roteadores, de hardware de computador, etc.) e não pode ser estendida ou modificada. Esta extensão de SNMP se refere estritamente ao software gerenciador SNMP [Inc].

4.6 Funcionamento do Protocolo SNMP

O SNMP foi projetado para ser o mais simples possível, sendo baseado apenas em estações de gerenciamento de rede e elementos da rede. As estações de gerenciamento da rede são responsáveis por rodar aplicações de gerenciamento que monitorem e controlem os elementos da rede. Os elementos da rede são *hubs* inteligentes, roteadores e pontes possuem agentes que estão localizados dentro do limites dos elementos, que são responsáveis por realizar as funções que são requisitadas pelas estações de gerenciamento.

Cada máquina gerenciada é vista como um conjunto de variáveis que representam informações referentes ao seu estado atual, estas informações ficam disponíveis ao gerente através de consulta e podem ser alteradas por ele. Cada máquina gerenciada pelo SNMP deve possuir um agente e uma base de informações MIB.

O SNMP é o meio pelo qual a estação de gerenciamento e os elementos de rede se comunicam. É um protocolo simples que permite à um administrador inspecionar ou alterar variáveis em um elemento de rede a partir de uma estação de gerenciamento remota.

Todo o monitoramento SNMP é realizado pelo sistema de gerenciamento de rede. As estações de gerenciamento acessam os elementos de rede para obter a informação desejada ou para mudar uma variável. Nesse caso estará sendo realizada uma operação de *polling*.

O agente é um componente que pode ser implementado em hardware ou em software. O que ele realiza na verdade é coletar dados de um dispositivo e armazená-los na MIB.

4.6.1 SNMP e o Protocolo TCP/IP

O protocolo SNMP foi desenvolvido para rodar sobre a camada de transporte, na camada de aplicação da pilha de protocolo TCP/IP. A maioria das implementações do SNMP utilizam o *User Datagram Protocol -UDP* como protocolo de transporte. O UDP é um protocolo não-confiável, não garantindo a entrega, a ordem ou a proteção contra duplicação das mensagens [Dav97].

O SNMP utiliza o UDP, pois foi desenvolvido para funcionar sobre um serviço de transporte sem conexão. A maior razão para isto é o desempenho. Utilizando um serviço orientado à conexão, como o TCP, a execução de cada operação necessitaria de uma prévia conexão, gerando um *overhead* significativo, o que é inaceitável na atividade de gerência onde as informações devem ser trocadas entre as entidades da forma mais rápida possível, sem afetar o desempenho da transmissão dos demais dados. Segmentos UDP são transmitidos em datagramas IP. O cabeçalho UDP inclui os campos de origem e destino, um *checksum* opcional que protege o cabeçalho UDP e os dados de usuário (em caso do *checksum* ser violado, a *Protocol Description Unit (PDU)* é descartada) [Tel] .

Duas portas UDP são associadas às operações SNMP. As operações Get, *GetNext*, *GetBulk*, *Inform* e *Set* utilizam a porta 161. Por ser acionada em casos de exceção, a operação *Trap* tem reservada para si a porta 162 [And04].

Como o UDP é um protocolo não-confiável, é possível que mensagens SNMP sejam perdidas. O SNMP por si só não fornece garantia sobre a entrega das mensagens. As ações a serem tomadas quando da perda de uma mensagem SNMP não são abordadas pelo padrão. No entanto, algumas considerações podem ser feitas sobre a perda de mensagens SNMP.

No caso de uma mensagem de *Get*, *GetNext* ou *GetBulk*, a estação de gerenciamento deve assumir que esta mensagem (ou o *GetResponse* correspondente) tenha sido perdida se não receber resposta num certo período de tempo. A estação de gerenciamento então pode repetir a mensagem uma ou mais vezes até que obtenha a resposta. Desde que um único *request-id*(que identifica a mensagem) seja utilizado para cada operação distinta, não haverá dificuldade em identificar mensagens duplicadas.

No caso de uma mensagem de *Set*, a recuperação deve provavelmente envolver o teste no objeto associado à operação através de uma Get sobre este mesmo objeto a fim de determinar se a operação Set foi ou não efetivada.

Conforme [Inc], no SNMP o gerente executa o gerenciamento através da utilização do protocolo SNMP, que é implementado no topo das camadas UDP, IP e do protocolo dependente do tipo de rede (*Ethernet*, FDDI, X.25, ATM, entre outras).

A figura 4.2 ilustra o contexto do protocolo SNMP na pilha de protocolo TCP/IP, utilizando o UDP como protocolo de conexão.

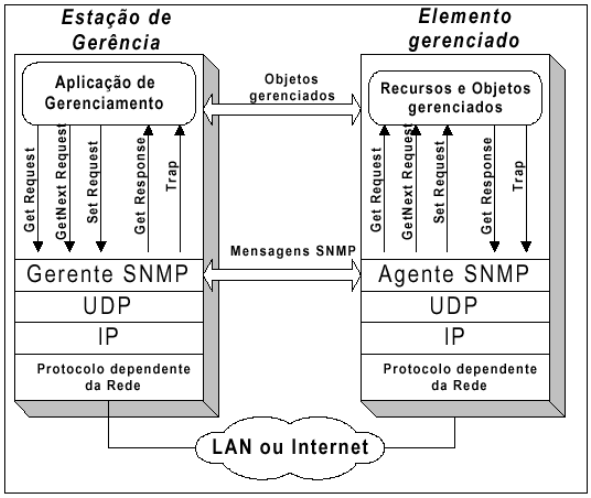

Figura 4.2: Protocolo SNMP sobre a Camada de Transporte

O SNMP requer alguns serviços da camada de transporte para que suas mensagens sejam transmitidas entre as entidades de gerenciamento. Isso independe da implementação da camada de transporte.

4.6.2 Serviço requeridos pelo SNMP

Rodando na camada de aplicação da arquitetura TCP/IP, como mostrado na Figura 4.2, o SNMP encontra-se acima das camadas de Transporte e Rede. Para que o SNMP possa funcionar adequadamente, estas camadas (Transporte e Rede) devem fornecer ao menos cinco serviços que são de vital importância à transmissão das mensagens SNMP através da rede [Inc]:

- **Roteamento** - A camada de Rede é quem fornece as funções de roteamento,

as quais aperfeiçoam toda a utilidade do gerenciamento da rede, dando a possibilidade de rotear várias vezes os pacotes em torno de áreas da rede com falhas. Isso permite que o gerenciamento da rede continue operando mesmo após falha em alguma(s) máquina(s) na rede.

- **Independência do meio** - Permite ao SNMP gerenciar diferentes tipos de elementos da rede, de forma independente. Caso o SNMP fosse construído sobre a camada de enlace, estaria preso a esta implementação de enlace específica, desse modo o SNMP não poderia gerenciar um outro dispositivo que implementasse outro protocolo de enlace, o que limitaria o gerenciamento da rede.

- **Fragmentação e Remontagem** - Este serviço está relacionado com a independência do meio. A mensagem SNMP pode ser fragmentada em pacotes, transmitida pelo meio e remontada no destino. O protocolo IP permite que os pacotes SNMP trafeguem pelo meio com diferentes tamanhos. A Fragmentação e a Remontagem reduzem a robustez geral do gerenciamento da rede, pois se qualquer fragmento for perdido pelo caminho, toda a operação irá falhar.

- ***Checksum* ponto-a-ponto** - O *checksum* ponto-a-ponto é um serviço de checagem de erros fornecido pela camada de transporte que permite a uma entidade conferir a integridade dos dados recebidos através da rede, aumentando a confiabilidade da transmissão;

- **Multiplexação/Demultiplexação** - Os serviços de Multiplexação e Demultiplexação são fornecidos pelo protocolo de transporte. Estes serviços facilitam em muito os possíveis relacionamentos de gerenciamento com o SNMP, permitindo que várias aplicações SNMP possam utilizar serviços da camada de transporte.

4.7 Formato das mensagens UDP

No SNMP, as informações são trocadas entre os gerentes e agentes na forma de mensagens. Cada mensagem possui duas partes, um cabeçalho e uma *Protocol Data Unit* (PDU). O formato geral de uma mensagem SNMP pode ser visualizada na Figura 4.3.

- **version** - Indica a versão do protocolo SNMP utilizado.

- **community** - O nome de comunidade atua como uma senha para autenticar a mensagem SNMP.

Figura 4.3: Formato de mensagem SNMP

- **SNMP PDU** - É a unidade de dados de protocolo, PDU, utilizada pelo SNMP. Contém os dados referentes a operação desejada (*Get*, *GetNext*, etc).

O campo PDU pode variar de acordo com o tipo de PDU utilizado em uma mensagem SNMP. As PDU *GetRequest PDU*, *GetNextRequest PDU* e *SetRequest PDU* são definidas no SNMP com o mesmo formato da *GetResponse PDU*, com os campos *error-status* e *error-index* com valor zero. Essas PDU possuem os seguintes campos [Dav97].

1. *PDU type* - indica o tipo de PDU, neste caso pode ser uma *GetRequest PDU*, uma *GetNextRequest PDU*, uma *SetRequest PDU* ou uma *GetResponse PDU*;

2. *request-id* - Usado para identificar o *request*. Essa mesma identificação será utilizada na resposta a esta mensagem;

3. *error-status* - É um sinalizador utilizado para indicar que uma situação inesperada ou erro ocorreu no processamento da mensagem; seus valores possíveis são:

   (a) *noError (0)* - Indica que não houve qualquer tipo de erro no processamento da mensagem;

   (b) *tooBig (1)* - Se o tamanho da mensagem *GetResponse* gerada exceder uma limitação local, então o campo error-status assume este valor. Neste caso, o valor do campo *error-index* deve ser zero;

   (c) *noSuchName (2)* - Indica que o valor de alguma variável da lista de variáveis (*variablebindings*) não está disponível na MIB em questão. Neste caso, o valor do campo *error-index* indica em qual variável da lista ocorreu o problema;

   (d) *badValue (3)* - Significa que o valor para uma determinada variável da lista não está de acordo com a linguagem ASN.1; ou que o tipo, tamanho ou valor é inconsistente. Assim como no caso anterior, o valor do campo *error-index* indica em qual variável da lista ocorreu o problema;

(e) *readOnly (4)* - Indica que determinada variável da lista só pode ser lida e não alterada. Neste caso, esse tipo de código de erro só é retornado após uma operação de *Set* sobre alguma variável. O valor do campo *error-index* indica em qual variável da lista ocorreu o problema;

(f) *genErr (5)* - Se o valor de uma determinada variável da lista não puder ser identificado ou alterado por outras razões que não as já citadas, então o campo *error-status* assume o valor genErr ou simplesmente 5. Neste caso, o valor do campo *error-index* indica em qual variável da lista ocorreu o problema;

(g) *error-index* - Quando o campo *error-status* é diferente de zero, este campo fornece uma informação adicional indicando qual variável da lista de variáveis causou a exceção (ou erro);

(h) *variablebindings* - Uma lista de nomes de variáveis e seus respectivos valores (em alguns casos, como no *GetRequest PDU*, esses valores são nulos). A estrutura deste campo é mostrada na Figura 4.4.

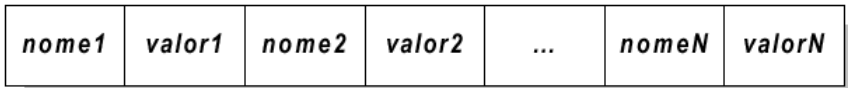

Figura 4.4: Estrutura do Campo *variablebindings*

Por se tratar de um caso particular de mensagem, indicando uma situação inesperada, a *Trap PDU* possui uma estrutura diferente das demais PDU utilizadas pelo SNMP.

Pode-se conferir na tabela 4.1, um apanhado dos principais campos de uma menagem SNMP.

4.7.1 *GetRequest PDU*

A *GetRequest PDU* é utilizada pela estação de gerência SNMP para executar a operação *Get*, requisitando ao agente SNMP a leitura de variáveis da MIB da máquina onde ele se encontra. A entidade emissora inclui os seguintes campos nesta PDU:

Percebe-se na Figura 4.5 que na *GetRequest PDU* os campos *error-status* e errorindex possuem o valor zero.

A entidade SNMP receptora responde a uma *GetRequest PDU* com uma *GetResponse PDU* contendo o mesmo valor do *request-id*, como está mostrando na figura 4.6. Caso a entidade receptora da *GetRequest PDU* possa fornecer o valor

Tabela 4.1: Campos de uma mensagem SNMP

Campo	Descrição
Version	Versão SNMP
Comunity	Comunidade a que pertence um agente SNMP com alguns conjuntos arbitrários de entidades de aplicação SNMP
Request-id	Usado para distinguir entre requests
Error-status	Usado para indicar que uma exceção ocorreu quanto processava um request
Error-index	Quando um *error-status* é não 0, *error-index* pode prover informações adicionais indicando qual variável na lista causou a exceção
Variable-bindings	Uma lista de nomes de variáveis e valores correspondentes
Enterprise	Tipo do objeto gerador do *Trap*
Generic-trap	Tipo genérico do *Trap*
Specific-trap	Código específico do *Trap*
Time-stamp	Tempo ocorrido entre última inicialização ou reinicialização da rede e a geração do *Trap*

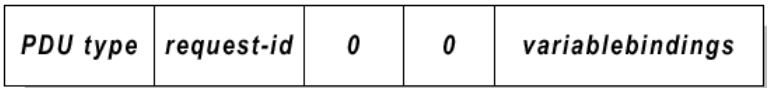

Figura 4.5: Formato da GetRequest PDU

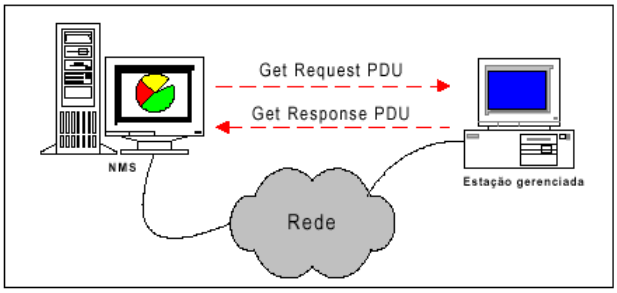

Figura 4.6: Passagem da *GetRequest PDU*

de todas as variáveis listadas no campo *variablebindings*, então é montada uma *GetResponse PDU* contendo este campo acrescentado do respectivo valor de cada variável. Se o valor de apenas uma variável não puder ser informado, então nenhum dos valores das outras variáveis são retornados [Dav97]. As condições de erro que podem acontecer são :

- para uma variável referenciada no campo *variablebindings* que não seja encontrada uma correspondente na MIB em questão; ou a variável referenciada pode ser um tipo agregado e desta forma não há uma valor agregado a esta instância. Neste caso, a entidade receptora retorna uma *GetResponse PDU* com o campo *error-status* indicando noSuchName e com o valor do campo *error-index* indicando que variável da lista de variáveis (*variablebindings*) causou o problema, ou seja, se a terceira variável da lista de variáveis não estiver disponível para a operação Get, então o campo *error-index* da *GetResponse PDU* possui o valor 3;

- a entidade receptora pode fornecer o valor de todas as variáveis da lista, mas o tamanho resultante da *GetResponse PDU* pode exceder a limitação local. Neste caso, a entidade receptora envia à entidade emissora uma *GetResponse PDU* com o campo errorstatus indicando tooBig;

- a entidade receptora pode não fornecer o valor de uma determinada variável por alguma outra razão. Neste caso, a entidade receptora retorna uma *GetResponse PDU* com o campo *error-status* indicando genErr e com o valor do campo *error-index* indicando que variável da lista de variáveis causou o erro. Se nenhum dos casos anterior se aplica, então a entidade receptora envia para a entidade da qual originou a *GetRequest PDU* uma *GetResponse PDU* com o campo errorstatus indicando noError e com o valor do campo *error-index* com valor zero.

4.7.2 *GetNextRequest PDU*

A *GetNextRequest PDU* é utilizada pela estação de gerência SNMP para executar a operação GetNext, requisitando ao agente SNMP a leitura da próxima variável na ordem lexicográfica da MIB da máquina onde ele se encontra. A *GetNextRequest PDU* é praticamente idêntica à *GetRequest PDU*, possuindo o mesmo mecanismo de troca e a mesma estrutura, como mostrado na Figura 4.7.

A única diferença é o seguinte: numa *GetRequest PDU* cada variável descrita na lista de variáveis (*variablebindings*) deve ter seu valor retornado. Na GetNextRequest PDU, para cada variável, é retornado o valor da instância que é a próxima na ordem segundo a definição da MIB. Assim como a *GetRequest PDU*,

PDU type	request-id	0	0	variablebindings

Figura 4.7: Formato da *GetNextRequest PDU*

a *GetNextRequest PDU* é chamada ou todos os valores de todas as variáveis da lista (*variablebindings*) são retornados ou nenhum deles é, conforme mostrado na figura 4.8.

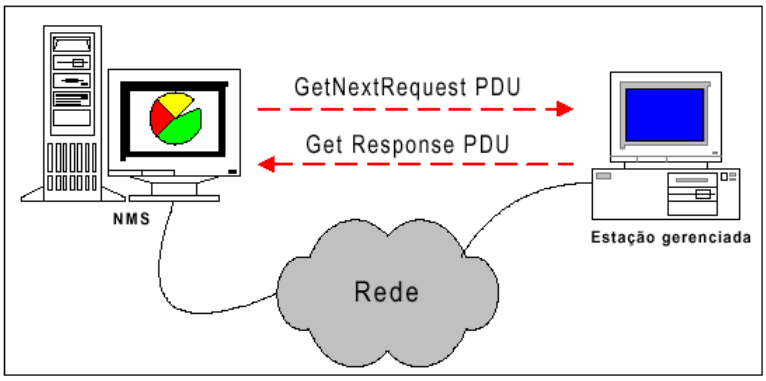

Figura 4.8: Passagem da *GetNextRequest PDU*

Apesar de parecer pouco significante, esta diferença entre a *GetRequest PDU* e a *GetNextRequest PDU* possui grandes implicações. A fim de entender essas implicações vamos analisar algumas possibilidades:

Supondo que uma estação de gerência (NMS) deseje recuperar os valores de todas as variáveis simples do grupo UDP de uma determinada MIB. A NMS então envia ao gerente responsável por gerenciar esta MIB uma *GetRequest PDU* na seguinte forma:

```
GetRequest (udpInDatagrams.0, udpNoPorts.0,
            udpInErrors.0, udpOutDatagrams.0)
```

Se a MIB em questão suportar todas as variáveis relacionadas na lista, então uma *GetResponse PDU* deverá ser retornada com os respectivos valores das variáveis referenciadas na *GetRequest PDU*:

```
GetResponse ((udpIndatagrams.0 = 100),(udpNoPorts.0 = 1),
             (udpInErrors.0 = 2) ,(udpOutDatagrams.0 = 200))
```

Onde 100, 1, 2 e 200 são os valores das respectivas variáveis requisitadas no GetRequest. Entretanto, se não for possível retornar o valor de qualquer das variáveis, então a *GetResponse PDU* é descartada e outra *GetResponse PDU* é construída e enviada ao gerente da NMS com o código de erro indicando noSuchName. Para assegurar que todos os valores disponíveis sejam informados, utilizando-se a *GetRequest PDU*, a estação de gerenciamento teria que enviar quatro destas PDU separadas, cada uma requisitando o valor de uma variável específica. Agora consideremos a utilização da *GetNextRequest PDU* na recuperação dos valores das variáveis:

```
GetNextRequest (udpIndatagrams.0, udpNoPorts.0,
                udpInErrors.0, udpOutDatagrams.0)
```

Neste caso, o agente irá retornar o valor da próxima instância do objeto (variável) para cada identificador da lista de variáveis. Suponha agora que todas as variáveis da lista sejam suportadas pela MIB em questão. O identificador (*Object Identifier*) para a variável ***udpInErrors***, por exemplo, é **1.3.6.1.2.1.7.3**. A próxima instância para esta mesma variável na ordem da MIB é identificada por ***udpInErrors***.0 ou **1.3.6.1.2.1.7.3.0**. Da mesma forma, a próxima instância de ***udpNoPorts*** **(1.3.6.1.2.1.7.2)** é **udpNoPorts.0 (1.3.6.1.2.1.7.2.0)**, e assim para as outras variáveis. Assim, se todos os valores estão disponíveis, o agente irá retornar uma *GetResponse PDU* na seguinte forma:

```
GetResponse ((udpIndatagrams.0 = 100),(udpNoPorts.0 = 1),
             (udpInErrors.0 = 2), (udpOutDatagrams.0 = 200))
```

A qual é a mesma retornada com a utilização da *GetRequest PDU*. Agora, suponhamos que o valor de *udpNoPorts* não esteja disponível e que a mesma *GetNextRequest PDU* seja utilizada. Neste caso, a resposta do agente viria na seguinte forma:

```
GetResponse ((udpIndatagrams.0 = 100),( udpInErrors.0 = 2),
             (udpInErrors.0 = 2) ,(udpOutDatagrams.0 = 200))
```

Neste caso, o identificador da variável *udpNoPorts.0* não é um identificador válido para a MIB em questão. Portanto, o agente retorna o valor da próxima instância na ordem, a qual neste caso é *udpInErrors.0*. Pode-se perceber que a utilização da

GetNextRequest PDU permite um melhor desempenho na recuperação de um conjunto de valores de variáveis quando há a possibilidade de algum destes valores não estarem disponíveis.

A utilização da operação *GetNext* é particularmente interessante na recuperação de uma seqüência de instâncias de variáveis. No entanto, esta operação ainda possui o inconveniente de se ter que enviar sucessivas *GetNextRequest PDU*, pois este tipo de PDU não permite recuperar um número grande de instâncias.

4.7.3 *GetBulkRequest PDU*

A *GetBulk PDU* é utilizada por um gerente SNMP para executar a operação *GetBulk*, uma das melhores características adicionadas ao SNMP padrão, que permite que um gerente SNMP resgate uma grande quantidade de informações de gerenciamento de uma determinada MIB. A *GetBulkRequest PDU* permite a um gerente SNMP requisitar que a resposta (*GetResponse PDU*) seja tão grande quanto possível dentro da restrição de tamanho.

A *GetBulkRequest PDU* segue o mesmo princípio da *GetNextRequest PDU*, recuperando sempre a próxima instância, na ordenação da MIB, de cada variável da lista de variáveis (*variablebindings*). A diferença é que, com a *GetBulkRequest PDU*, é possível especificar quantas próximas instâncias na ordem devem ser retornadas na mesma *GetResponse PDU*.

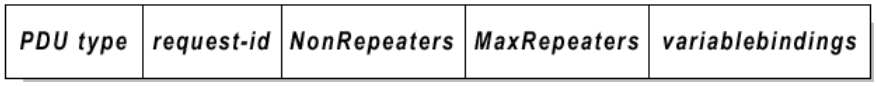

Figura 4.9: Formato da *GetBulkRequest PDU*

Como mostrado na Figura 4.9, a *GetBulkRequest PDU* utiliza um formato diferente das demais PDU, com os seguintes campos:

- *PDU type, request-id e variablesbindings* - Estes campos possuem na *GetBulkRequest PDU* a mesma função que possuem nas demais PDU (*GetRequest PDU*, *GetNextRequest PDU*, *SetRequest PDU*, *Response PDU*, *InformRequest PDU* e *Trap* PDU);

- *Nonrepeaters* - Especifica o total das primeiras variáveis da lista de variáveis (*variablebindings*) das quais apenas a próxima instância na ordem deve ser retornada;

- *Max-repetitions* - Especifica o número de sucessores, também na ordem, que devem ser retornados para o resto (`total_de_variáveis_da_lista - Nonrepeaters`) das variáveis da lista de variáveis.

Após executar a operação *GetBulk* o agente deve construir uma *GetResponse* PDU, contendo as instâncias das variáveis requisitadas e seus respectivos valores. No entanto, pode acontecer de a *GetResponse PDU* não retornar todos os pares (instância, valor) requisitados na lista (podendo até retornar nenhum dos pares). Isso pode ocorrer por três razões:

1. se o tamanho da mensagem que encapsula a *GetResponse PDU* exceder a limitação local de tamanho ou exceder o tamanho máximo de uma mensagem de request (definido pelo protocolo), então a GetResponse é gerada com um número menor de pares no campo *variablebindings*. Ou seja, a *GetResponse PDU* é gerada inserindo-se os pares até que o tamanho máximo seja alcançado;

2. se todos os pares subseqüentes descritos na lista de variáveis possuírem o valor *endOfMibView* (significando que não existe um sucessor da variável na ordem da MIB), o campo *variablebindings* pode ser truncado neste ponto, retornando nenhum dos pares requisitados;

3. se o processamento de uma *GetBulkRequest PDU* necessitar de uma grande quantidade de tempo de processamento pelo agente, então o agente pode particionar o processamento, retornando resultados parciais. A figura 4.10 ilustra o funcionamento deste tipo de PDU.

Se o processamento de alguma variável falhar, por qualquer motivo que não seja *endOfMibView*, então nenhum dos pares requisitados é retornado. Em vez disso, o agente envia ao gerente uma *GetResponse PDU* com o campo *error-status* indicando *genErr* e o valor do campo *error-index* apontando para a variável da lista que gerou o problema. O código de erro tooBig nunca é retornado por um agente SNMP pois, como já foi dito, quando a mensagem de resposta é muito grande nem todas as variáveis requisitadas são retornadas na mesma PDU, diminuindo seu tamanho.

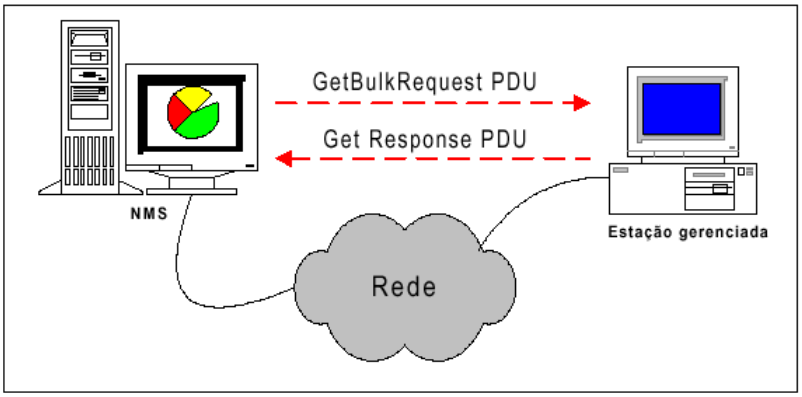

Figura 4.10: Passagem da *GetBulkRequest PDU*

4.7.4 *SetRequest PDU*

A *SetRequest PDU* é utilizada por um gerente SNMP na realização da operação Set, indicando ao agente que o valor de determinada(s) variável(is) deve(m) ser alterado(s). Este tipo de PDU possui o mesmo formato da *GetRequest PDU*. No entanto, é utilizada para escrever uma variável na MIB, ao invés de lê-la. Diferente da *GetRequest PDU* e da *GetNextRequest PDU*, cujo valor das variáveis referenciadas na lista (*variablebindings*) têm valor zero, neste tipo de PDU o valor de cada instância da lista representa o valor a ser modificado na MIB em questão. Na figura 4.11 pode-se observar os campos de uma mensagem *SetRequest PDU*.

PDU type	*request-id*	*0*	*0*	*variablebindings*

Figura 4.11: Estrutura da *SetRequest PDU*

Quando um agente recebe uma *SetRequest PDU* ele efetua, quando possível, a operação Set, modificando o valor das variáveis correspondentes, e retorna ao gerente da NMS emissora uma *GetResponse PDU* contendo o mesmo *request-id*. Assim como a operação Get, a operação Set é solicitada ou todas as variáveis são modificadas/atualizadas ou nenhuma delas o é.

Se um agente que recebeu uma *SetRequest PDU* puder alterar o valor de todas as variáveis especificadas na lista (*variablebindings*), então ele constrói uma *GetResponse* PDU incluindo a mesma lista de variáveis, com seus respectivos valores

atualizados, enviando-a ao gerente da NMS emissora. Se ao menos um dos valores não puder ser modificado, então nenhum valor é retornado e nem escrito na MIB em questão. As mesmas condições de erro usadas na *GetRequest PDU* podem ser retornadas (*noSuchName, tooBig, genErr*).

Outro erro que pode ocorrer durante a operação Set é o badValue. Esta condição de erro ocorre quando algum dos pares (variável, valor) contidos na *SetRequest PDU* estiver inconsistente no que diz respeito ao tipo, tamanho ou valor. Quando algum erro ocorre o agente descarta a *GetResponse PDU* e constrói uma nova com o campo statuserror indicando o tipo de erro que ocorreu durante a operação e com o campo *error-index* apontando para a variável da lista que ocasionou o problema. A figura 4.12 ilustra o funcionamento deste tipo de PDU.

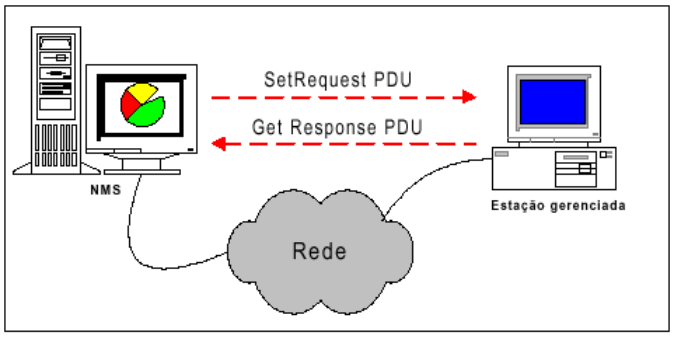

Figura 4.12: Passagem da *SetRequest PDU*

Um curioso código de erro que pode ser retornado por um agente durante a execução da operação Set é o *read-only*, que praticamente não é implementado. O problema é que o *read-only* significa que uma determinada variável na MIB em questão cujo valor deveria ser modificado possui status de apenas leitura (*read-only*), não podendo portanto ser alterado. No entanto, quando um gerente pede a um agente para modificar uma variável que seja do tipo read only, o agente não podendo executar esta operação, constrói e envia uma *GetResponse PDU* com o código de erro indicando noSuchName e não readOnly.

O grande problema desta confusão é o fato de que quando um gerente recebe uma *GetResponse PDU* de uma operação de Set com o código de erro indicando noSuchName, ele não tem como saber se a instância da variável realmente não foi encontrada na MIB (o que justifica o noSuchName) ou se o problema, na verdade, é que a instância da variável possui status read only na MIB em questão.

Neste caso, para ter certeza da origem do problema, o gerente teria que enviar uma *GetRequest PDU* para a variável que causou o problema a fim de verificar

se realmente a variável não existe na MIB ou se ela apenas possui o status de *read-only*.

4.7.5 *Trap PDU*

A *Trap PDU* é utilizada pelo agente SNMP na execução da operação *Trap*, notificando de forma assíncrona eventos extraordinários que tenham ocorrido no objeto gerenciado. A *Trap PDU* possui uma estrutura diferente das outras PDU como mostrada na Figura 4.13.

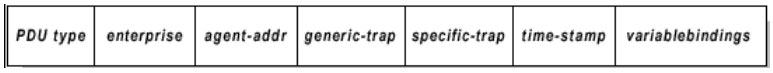

Figura 4.13: Estrutura da *Trap PDU* - SNMP

- *PDU type* - indica o tipo de PDU, neste caso indica que trata-se de uma *Trap* PDU;
- *enterprise* - indica o tipo do objeto que gerou o *Trap*; este campo é preenchido a partir do *sysObjectID*;
- *agent-addr* - endereço do objeto que gerou o *Trap*;
- *generic-trap* - indica o tipo do *Trap*; os valores possíveis para este campo são:
  - *coldStart (0)* - Significa que a entidade emissora do *Trap* está sendo inicializada, de tal modo que a configuração do agente ou da entidade de protocolo podem ser alteradas;
  - *warmStart (1)* - Significa que a entidade emissora do *Trap* está sendo reinicializada, de tal modo que nenhuma configuração do agente nem da entidade é alterada;
  - *linkDown (2)* - Significa que a entidade emissora do *Trap* reconheceu que o estado de alguma de suas linhas de comunicação representadas na configuração do agente está disponível;
  - *linkUp (3)* - Significa que a entidade emissora do *Trap* reconheceu que o estado de uma de suas linhas de comunicação representadas na configuração do agente está disponível;

– *authentication-Failure (4)* - Significa que a entidade emissora do *Trap* foi o destinatário de uma mensagem de protocolo que não estava corretamente autenticada. Quando da implementação do gerente, esta propriedade de enviar este tipo de *Trap* deve ser configurável, podendo em determinados casos ser desativada;
– *egpNeighborLoss (5)* - Significa que um vizinho EGP (Exterior Gateway Protocol) da entidade emissora do *Trap* foi o mesmo EGP de quem a entidade emissora marcou a queda e cujo relacionamento foi perdido;
– *enterprise-Specific (6)* - Significa que a entidade emissora do *Trap* reconhece que ocorreu algum evento avançado específico ocorreu;
– *specific-trap* - possui o código específico do *Trap*;
– *time-stamp* - armazena o tempo decorrido entre a última reinicialização da entidade que gerou o *Trap* e a geração do *Trap*; contém o valor de sysUpTime;
– *variablebindings* - Uma lista de nomes de variáveis e seus respectivos valores.

Toda vez que uma máquina da rede é inicializada, o agente responsável deve enviar ao gerente um *Trap* do tipo coldStart (0), indicando que a máquina "entrou"na rede. Outra característica que difere a *Trap PDU* das outras PDU é o fato de que ela não necessita de uma resposta, como mostra a Figura 4.14.

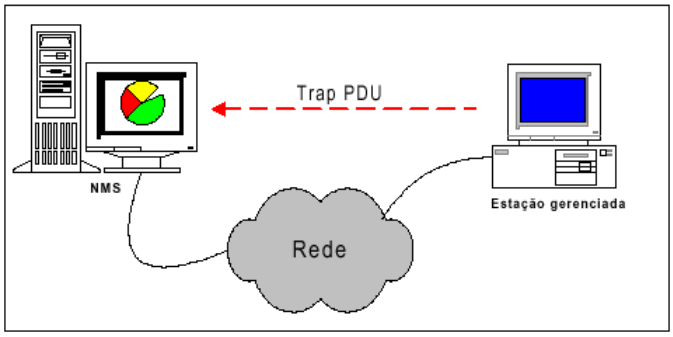

Figura 4.14: Passagem da *Trap PDU*

4.7.6 *Trap PDU*-v2

A *Trap PDU*-v2 segue as mesmas regras da *Trap PDU* utilizada na versão básica do SNMP, mas com um formato diferente. Conforme a Figura 4.15, este

formato é o mesmo de todas as outras PDU utilizadas no SNMP, exceto a *Get-BulkRequest PDU*, o que facilita o processamento das mensagens pelo agente SNMP.

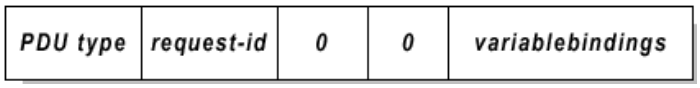

Figura 4.15: Estrutura da *Trap PDU* - SNMPv2

4.7.7 InformRequest PDU

A InformRequest PDU é enviada por um agente SNMP a outro agente SNMP, fornecendo informações de gerenciamento ao último agente. Esta PDU inclui o campo *variablebindings* com os mesmos elementos da *Trap PDU*-v2, como mostra a Figura 4.16.

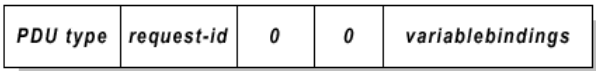

Figura 4.16: Estrutura da InformRequest PDU

Ao receber uma InformRequest PDU, o agente determina o tamanho da mensagem de resposta que encapsula uma *GetResponse PDU* com os mesmos valores dos campos *request-id*, *error-status*, *error-index* e *variablebindings* da InformRequest PDU recebida. Se este tamanho exceder a limitação local de tamanho ou exceder o tamanho máximo da mensagem de resposta, então uma *GetResponse PDU* é construída com o código de erro (*error-status*) indicando tooBig, o índice de erro (*error-index*) com valor zero e com a lista vazia de variáveis (*variablebindings*). A figura 4.17 ilustra o funcionamento deste tipo de PDU.

Caso o tamanho da mensagem seja aceitável, seu conteúdo é passado para a aplicação destino e é construída e enviada para o agente origem uma *GetResponse PDU* com o código de erro indicando *noError* e o índice de erro com valor zero.

4.7.8 *GetResponse PDU*

O formato da *GetResponse PDU* é idêntico a *GetRequest PDU* a não ser pela indicação do tipo (*PDU Type*). Uma *GetResponse PDU* é gerada por uma entidade SNMP sempre que esta recebe uma *GetRequest PDU*, *GetNextRequest PDU* ou *SetRequestPDU*, como descrito anteriormente.

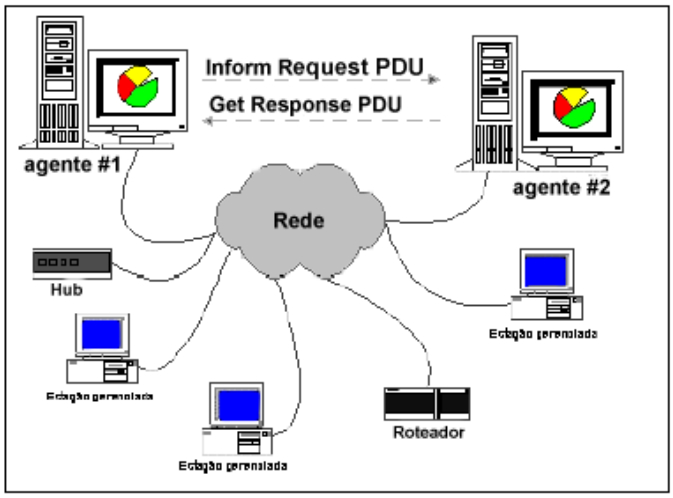

Figura 4.17: Passagem da InformRequest PDU

4.8 Considerações Finais

Este capítulo teve como objetivo realizar uma explanação sobre a origem, características e o funcionamento do protocolo SNMP. Também realizou-se um estudo detalhado sobre os elementos e as operações que podem ser executadas sobre as variáveis MIBs, exibindo o formato para cada tipo de mensagem e o seu funcionamento, sempre procurando fornecer ao administrador de redes informações que possibilite obter maturidade para a implementação de agentes SNMP em sua rede.

Capítulo 5

Desenvolvimento de Agentes SNMP

5.1 Introdução

Neste capítulo será tratado as formas e técnicas necessárias ao desenvolvimento de agentes SNMP. O Desenvolvimento de agentes SNMP em Ambientes Linux é possível sob duas óticas de implementação. A implementação através de agentes extensíveis e estendidos.

- **Extensíveis** - Normalmente já implementam a MIB-II. Usam o SNMP diretamente. Possuem APIs de programação para outros agentes.

- **Estendidos** - Implementam normalmente complementos da MIB-II (objetos específicos). Usam o SNMP indiretamente do agente extensível. Interage com um agente extensível através de APIs de programação.

Em geral, o agente extensível deve implementar o SNMP, para que os agentes estendidos utilizem. A interação com os recursos ainda deve ser feita de forma proprietária.

Um agente extensível pode possuir vários agentes estendidos, interagindo com recursos diferentes. Normalmente um agente extensível é fornecido pelo sistema operacional ou por bibliotecas específicas, que é o caso do Linux. Além do agente, tem-se também bibliotecas auxiliares para manipulação de estruturas do protocolo (SnmpAPI)

Para a construção de agentes, pode-se utilizar basicamente três técnicas principais :

- Construção direta via sockets
- Proxy SNMP para extensão indireta de agente
- Construção de agentes estendidos via API dos agentes extensíveis

Sockets

Aloca-se a porta UDP 161 e recebe-se mensagens remotamente. A aplicação deve ser capaz de entender as mensagens que chegar, isto é, deve implementar o protocolo SNMP. Depois das requisições, as respostas devem ser geradas também de acordo com o protocolo.

A aplicação deve saber enviar traps diretamente utilizando a porta 160 remota do gerente associado. Normalmente, deve-se implementar a MIB-II, além dos objetos de gerenciamento extras

Proxy

Sobrepõe-se o agente local através de um novo agente. O novo agente implementa objetos extras, e chama o agente original para os objetos anteriores. Assim, neste caso, evita a implementação da MIB-II. Porém apresenta alguns problemas, tais como duas portas são usadas para se acessar alguns recursos, possiblidade de cascateamento de *proxy's* e duplo mapeamento em plataformas de uma mesma máquina.

Agente Estendido

Neste caso, apenas uma porta usa SNMP no sistema, que é a porta 161 do agente extensível. Assim, agentes estendidos são chamados apenas quando necessário, tendo-se vários agentes estendidos associados a um mesmo agente extensível.

5.2 Desenvolvimento de agentes via NET-SNMP

O NET-SNMP é um conjunto de softwares que inclui:

- Um agente SNMP extensível ;
- Um daemon receptor de *traps*;
- Ferramentas de gerência, como por exemplo, *tkmib*;

- Ferramentas de desenvolvimento, como a mais comumente utilizada, ***mib2c***;
- Bibliotecas de desenvolvimento, em especial, *snmpwalk*;

Para a instalação do Net-SNMP deve-se seguinte os seguintes passos:

1. Logar-se se como root no diretório padrão;
2. Fazer o download do pacote # `net-snmp-5.2.1.tar.gz` , que se pode obter http://www.net-snmp.org.
3. Descompactar o pacote # `tar xvfz net-snmp-5.2.1.tar.gz`
4. Deslocar-se para o diretório descompactado # `cd net-snmp-5.2.1`
5. Verificar as opções de configuração # `./configure --help`
6. Utilzando apenas #`./configure` se utlizará, por padrão, SNMPv1.
7. Compilar o pacote # `make`
8. Instalar o pacote # `make install`

Os comandos do Net-SNMP são copiados para os diretórios `/usr/local/bin` e `/usr/local/sbin`. Caso estes diretórios não estejam presentes na variável de ambiente `PATH`, é necessários acrescentá-los. Os arquivos MIB utilizados pelo gerente SNMP ficam localizados no diretório `/usr/local/share/snmp/mibs`. Enfim, ao final dos passos anteriores, observa-se que a estrutura de diretórios listada na figura 5.1 estará criada.

5.2.1 Configuração do Agente SNMP

O agente Net-SNMP é o executável `snmpd` que se encontra no diretório `/usr/local/sbin`. O arquivo de configuração utilizado pelo Net-SNMP é chamado `snmpd.conf` e deve estar presente no diretório `/usr/local/share/snmp`. A princípio, os fontes do pacote vem acompanhado de um arquivo de exemplo (`EXAMPLE.conf`) que deve ser adaptado de acordo com as necessidades do ambiente de rede.

Inicialmente, pode-se copiar este arquivo de exemplo de configuração através de:

```
#cp /usr/loca/share/snmp/docs/EXAMPLE.conf /usr/local/share/snmp/snmpd.conf
```

Para mais informações sobre arquivo de configuração, sugere-se consultar os manuais do arquivo

```
/root
   /net-snmp-5.2.1
        /agent  # fontes para o daemon SNMP
             /mibgroup # fontes para as MIBs
                  /mibII # fontes da MIB-II
                      /examples # fontes de MIBs de exemplo
/usr
   /bin # ferramentas snmp
   /sbin # daemons snmpd e snmptrapd
   /lib # bibliotecas snmp
   /share/snmp
           /mibs # arquivos de MIB em .txt
           /snmpconf # arquivos de configuracao
```

Figura 5.1: Estrutura de diretório criada após instalação Net-SNMP

```
# man snmpd.conf
```

Dentro desse arquivo, os campos mais significantes são a declaração da máquina e o IP da rede que o agente SNMP terá acesso, `local` e a rede utilizada, para este caso será simplesmente chamada `minharede`. Para efeito de exemplificação, a comunidade (`community`) é `tst`:

```
# sec.name community source
com2sec local localhost tst
com2sec minharede 192.9.201.0/24 tst
```

Em seguida, a declaração de grupos para acesso aos objetos SNMP gerenciados pelo agente SNMP, para `local` e `minharede`:

```
#
# Segundo passo, mapear os nomes seguros nos nomes dos grupos:
# sec.model sec.name
group local MeuGrupoRW v1
group local MeuGrupoRW v2c
group MeuGrupoRO v1 minharede
group MeuGrupoRO v2c minharede
```

Acesso de ler e esrever objetos MIB a partir de uma requisição local SNMP e acesso de somente leitura para requisições remotas:

```
#
# Aqui 2 grupos possuem acesso de forma diferente
# Permissoes de Escrita:
# context sec.model sec.level match read notif Write
access exact MyROGroup "" any noauth all none none
access exact MyRWGroup "" any noauth all all none
```

Em seguida, validação de `traps` SNMP TRAPs em direção ao gerente SNMP que está rodando na máquina local com a comunidade `tst`:

```
#
# Traps v1 e v2 habilitas e perceptiveis na maquina local
#
# command host to manage community
trapsink localhost tst
trap2sink localhost tst
```

Pode-se também declarar os objetos *sysLocation* e *sysContact* para um sistema vazio:

```
syslocation Teresina, Piaui, Brasil
syscontact JMessias <jmessias@ufpi.br>
```

Com a ferramenta `snmpconf` é possivel gerar um arquivo `snmpd.conf` de uma forma interativa e fácil sem saber a sua estrutura. Para tanto, pode-se utilizá-lo de duas formas. Para uma criação rápida e simples de um arquivo `snmpd.conf`, utiliza-se:

```
# snmpconf -g basic_setup
```

Ou, então, através de perguntas mais detalhistas, utilizando:

```
# snmpconf
```

Após a criação do arquivo `snmpd.conf`, este deve ser copiado para o diretório `/usr/local/share/snmp`. Enfim, pode-se iniciar o agente SNMP através de:

```
# /etc/init.d/snmpd start
```

Contactando o agente pela primeira vez.

```
# snmpwalk localhost public system
```

Pode-se verificar a comunicação entre o agente e o gerente SNMP utilizando um *sniffer* de rede[1], como **tcpdump**[2], em uma outra janela ou terminal.

```
# tcpdump -vv -i lo
```

Para o correto funcionamento do Net-SNMP é importante configurar as variaveis de ambiente do Linux. Para acessar todos os objetos MIB gerenciados pelo agente SNMP de uma forma simbolica e não numérica (OID), é necessário ler os arquivos MIB que estão contidos no diretório `/usr/local/share/snmp/mibs`.

```
#
# PATH
#
PATH=$PATH:/usr/local/bin:/usr/local/sbin

# MIBS: forca ler todos os arquivos MIB presentes
# em /usr/local/share/snmp/mibs
#
MIBS=ALL

# exportar todas as variaveis
#
export PATH MIBS
```

[1]Um sniffer é um programa que consegue capturar todo o tráfego que passa em um segmento de uma rede

[2]TCPDump é um poderoso analisador de pacotes de rede baseado em modo texto. Está disponível em: http://www.tcpdump.org

Para a utilização de todas as operações existentes no protocolo SNMP, na tabela 5.1 tem-se um conjunto de comandos que se pode utilizar para realizar tais operações.

Tabela 5.1: Comandos do Net-SNMP

COMANDO	DESCRIÇÃO
snmpget	envia uma requisição *SNMP Get* para obter o valor atual contido em um objeto MIB gerenciado por um agente SNMP remoto.
snmpset	envia uma requisição *SNMP Set* para atualizar o valor atual contido em um objeto MIB gerenciado por um agente SNMP remoto.
snmpgetnext	envia uma requisição *SNMP GetNext* e obtém o valor do próximo objeto MIB gerenciado gerenciado por um agente SNMP remoto, se disponível.
snmpwalk	este comando é semelhante ao comando snmpgetnext, porém permiti obter todos os valores de uma MIB.
snmptranslate	permiti converter um objeto MIB representado sob a forma simbólica para a forma decimal (OID) e vice-versa.

A seguir, alguns exemplos de utilização dos comandos listados na tabela 5.1.
Correspondência entre uma OID e sua forma simbólica

```
# snmptranslate 1.3.6.1.2.1.1.3.0
SNMPv2-MIB::sysUpTime.0
```

Representação gráfica de um sistema por meio da análise dos arquivos MIB contidos no diretório `/usr/local/share/snmp/mibs`:

```
# snmptranslate -Tp -IR system
+-- system(1)
|
+---R-- String sysDescr(1)
|Textual Convention: DisplayString
|Size: 0..255
...
```

Requisição em SNMPv1 para obter o valor atual do objeto *sysUpTime* gerenciado pelo agente SNMP executado na máquina local:

```
# snmpget -v 1 -C tst localhost system.sysUpTime.0
SNMPv2-MIB::sysUpTime.0 = Timeticks: (12908) 0:02:09.08
```

Requisição em SNMPv2c para obter o valor atual do objeto *sysUpTime*:

```
# snmpget -v 2c -C tst localhost system.sysUpTime.0
SNMPv2-MIB::sysUpTime.0 = Timeticks: (13966) 0:02:19.66
```

Requisição em SNMPv1 para obter o valor atual dos objetos *sysLocation* e *sysContact* (definidos no arquivo `snmpd.conf`):

```
# snmpget -v 1 -C tst localhost system.sysLocation.0
SNMPv2-MIB::sysLocation.0 = STRING: Teresina, íPiau, Brasil
# snmpget -v 1 -C tst localhost system.sysContact.0
SNMPv2-MIB::sysContact.0 = STRING: JMessias <jmessias@ufpi.br>
```

Perguntando ao sistema por objetos MIB gerenciados pelo agente SNMP executado na máquina local:

```
# snmpwalk -v 1 -C tst localhost system
SNMPv2-MIB::sysDescr.0 = STRING: Linux jordan 2.4.18-3
                         #1 Mon Apr 11 07:31:07 EDT 2005 i586
SNMPv2-MIB::sysObjectID.0 = OID: Net-SNMP-MIB::netSnmpAgentOIDs
SNMPv2-MIB::sysUpTime.0 = Timeticks: (28119) 0:07:21.19
SNMPv2-MIB::sysContact.0 = STRING: JMessias <jmessias@ufpi.br>
SNMPv2-MIB::sysName.0 = STRING: jordan
SNMPv2-MIB::sysLocation.0 = STRING: Teresina, íPiau, Brasil
. . .
```

Colocando um novo valor ao objeto *sysLocation* usando o SNMPv1:

```
# snmpset -v 1 -C tst localhost system.sysLocation.0 s "gauss"
Error in packet.
Reason: (noSuchName) There is variable No such name in this MIB.
Failed object: SNMPv2-MIB::sysLocation.0
```

Colocando um novo valor ao objeto *sysLocation* usando o SNMPv2c:

```
# snmpset -v 2c -C tst localhost system.sysLocation.0 S "gauss"
Error in packet.
Reason: notWritable (that object does not support modification)
Failed object: SNMPv2-MIB::sysLocation.0
```

Comparando as saídas dos dois últimos comandos *snmpset*, pode-se perceber que o SNMPv1 não retorna um código de erro, o que já acontece com o SNMPv2c. Neste caso, está-se tentando modificar um objeto MIB com acesso de somente leitura.

5.3 Escrevendo e Instalando uma nova MIB

Nesta seção, mostrar-se-á como escrever uma MIB exemplo e instala-lá junto a um novo daemon `snmpd`.

A MIB que será utilizada de exemplo será a exibida no anexo A1.

Copiando a MIB para o diretório de MIBS.

```
# cp JM-TESTE-1-MIB.txt /usr/local/share/snmp/mibs
```

Fazendo com que as ferramentas SNMP reconheça esta MIB.

```
# echo "mibs +JM-TESTE-1-MIB" >> /usr/local/share/snmp/snmp.conf
```

Para verificar se a MIB está carregada utiliza-se o comando snmptranslate:

```
# snmptranslate -IR -Tp experimental
```

5.3.1 Gerando um esqueleto de código C

Pode-se utilizar a ferramenta ***mib2c*** para gerar um esqueleto de código em C para suporte a uma nova MIB. O esqueleto de código deve ser instrumentado pelo desenvolvedor, e compilado junto ao agente SNMP, gerando um novo daemon snmpd. A ferramenta ***mib2c*** é encontrada em `/usr/bin/mib2c`

Para executar, entretanto, ela necessita de suporte SNMP via Perl

Para instalar suporte SNMP via Perl:

```
# cd /root/net-snmp-5.2.1/perl/SNMP
# perl Makefile.PL -NET-SNMP-PATH=/usr
# make
# make test
# make install
```

Figura 5.2: Passos para instalar no Net-SNMP suporte a Perl

Supondo um arquivo de MIB chamado `JM-TESTE-1-MIB.txt` que internamente define um módulo `jmteste`

1. Copiar o arquivo de MIB para o diretório de arquivos de MIB

   ```
   # cp JM-TESTE-1-MIB.txt  /usr usr/share/snmp/mibs
   # chmod 755 JM-TESTE-1-MIB.txt
   ```

É necessário que as ferramentas do Net-SNMP reconheça esta nova MIB. Para tanto, deve-se executar o seguinte comando:

```
# echo "mibs +JM-TESTE-1-MIB" >> /usr/local/share/snmp/snmp.conf
```

2. Exportar todas as MIBs as - `# export MIBS=ALL`
3. Ir para o diretório de desenvolvimento de agente - `# cd /root/net-snmp-5.2.1/agent/mibgroup/examples`
4. Compilar o arquivo de MIB via ***mib2c*** - `# /usr/bin/mib2c JM-TESTE-1-MIB`

5.3.2 Configurando a compilação de um novo daemon snmpd

Após usar o ***mib2c***, dois arquivos devem ter sido no seu diretório criados, `JM-TESTE-1-MIB.c` e `JM-TESTE-1-MIB.h` .

Para não misturar com o exemplo anterior, iremos renomeá-los simplemente para `exemplo.c` e `exemplo.h`.

Depois de instrumentar o arquivo exemplo.c, é necessário configurar a compilação de um novo daemon snmpd com suporte a `exemplo`, conforme os passos a seguir.

1. Ir para o diretório de instalação do NET-SNMP

```
# cd /root/net-snmp-5.2.1/perl/SNMP
```

2. Gerar nova configuração incluindo `exemplo`

```
# ./configure --prefix=/usr --with-mib modules="examples/exemplo"
```

5.3.3 Compilando e instalando o suporte a MIB exemplo

1. Interromper o daemon snmpd (se em execução) -

```
# /etc/init.d/snmpd stop
```

2. Ir para o diretório de compilação do daemon snmpd -

```
# cd /root/net-snmp-5.2.1/agent
```

3. Compilar o novo daemon -

```
# make
```

4. Instalar o novo daemon -

```
# make install
```

5. Inicializar o novo daemon -

```
# /etc/init.d/snmpd start
```

6. Testar se o agente responde à MIB `exemplo`

```
# snmpwalk localhost public exemplo !
```

Na figura 5.3 tem-se os diretórios utilizados com Net-SNMP e a descrição de cada um.

```
# Diretorio para colocacao de novas MIBs
  /usr/share/snmp/mibs

# Diretorio para configuracao do novo agente snmp
 /root/net-snmp-5.2.1

# Diretorio para os fontes do agente a ser construido
 /root/net-snmp-5.2.1/agent/mibgroup/examples/

# Diretorio para compilacao e instalacao do novo daemon snmp
 /root/net-snmp-5.2.1/agent

# Diretorio de controle do daemon snmpd
  /etc/init.d/snmpd
```

Figura 5.3: Resumo dos diretórios utilizados com Net-SNMP

Sugeri-se sempre utilizar três shells no processo de desenvolvimento de suporte a MIBs, como a MIB JM-TESTE-1-MIB:

1. Shell para edição do fonte exemplo.c
   `# cd /root/net-snmp-5.2.1/agent/mibgroup/examples`

2. Shell para edição da MIB JM-TESTE-1-MIB.txt
   `# cd /usr/share/snmp/mibs`

3. Shell para compilação, instalação e execução do daemon snmp

```
# cd /root/net-snmp-5.2.1/agent
# ln -s /etc/init.d/snmpd snmpdd
```

Instrumentação do código

O código gerado por ***mib2c*** não é livre de erros. Geralmente, apresenta dois principais problemas , a princípio, devem ser resolvidos:

- Correção das rotinas de escrita em Strings (e derivados, por exemplo endereços IP)
- Definição do número de linhas de uma tabela através da constante `TABLE_SIZE`

Correção das rotinas de em Strings (e derivados)
O código original abaixo abaixo:

```
case RESERVE2:
     size = var_ val_len;
     string = (char char *) var_val;
```

Deve ser substituído por:

```
case RESERVE2:
     size = var_ val_len;
     strncpy (string, (char char *) var_val val, size);
     string[size] = 0;
```

Definição do número de linhas de uma tabela através da constante `TABLE_SIZE`
O código :

```
if (header_simple_table(vp, name, length, exact, var_len ,
    write_method, TABLE_SIZE) == MATCH_FAILED )
```

Deve ser substituído por:

```
if (header_simple_table (vp, name, length, exact, var_len,
    write_method, 3) == MATCH_FAILED )
```

Onde 3 é o número de linhas da tabela

5.3.4 Traps em NET-SNMP

Há três funções que são utilizadas para o envio de traps de dentro de um módulo snmpd. São elas:

- `send_easy_trap`
- `send_trap_vars`
- `send_v2trap`

As traps são enviadas para todos os gerentes cadastrado no arquivo de configuração `snmpd.conf` A função `send_easy_trap` é usada para enviar traps rapidamente, sem a passagem de objetos extras (além do *sysUpTime* convencional de uma *trap*). A função `send_var_traps` é usada para enviar traps, mas com a possibilidade de se ter objetos extras (além do *sysUpTime*). A função `send_v2trap` é usada para enviar traps SNMPv2.

5.4 Considerações Finais

A ferramenta descrita neste capítulo possui várias características que possibilitam a real implementação de agentes SNMP em ambientes Linux. Explanou-se sobre a configuração e instalação dessa ferramenta, a estrutura de diretórios utilizadas por ela, os comandos que realizam as operanções do protocolo SNMP e utilização da ***mib2c***, para a criação de esqueletos de código C a partir de MIBs. No Capítulo 6 é apresentado ferramentas para a visualização dos dados existentes nas MIBs instaladas, através de gráficos, sua instalação e configuração.

Capítulo 6

Monitoramento e Visualização através do MRTG

6.1 Introdução

MRTG (*Multi Router Traffic Grapher*) é uma ferramenta de monitoramento, um pacote escrito em Perl usado para controlar o tráfego e os recursos de rede.

Com o MRTG é possível realizar a monitoração de qualquer equipamento que ofereça suporte ao protocolo SNMP. Seu relatório é relado em formato HTML, sendo assim, podendo ser publicado através de um servidor de HTTP (Web).

6.2 Download e Instalação

6.2.1 Download

Pode-se baixar o MRTG a partir da URL:
http://people.ee.ethz.ch/~oetiker/webtools/mrtg/pub/mrtg.tar.gz.
Recomenda-se a verificação de novas versões no site oficial, periodicamente.

6.2.2 Compilação e Instalação

Acessando um diretório temporário de produção.

```
# cd /usr/local/src
# tar xvfz mrtg-2.10.11.tar.gz
# cd mrtg-2.10.11
# ./configure --prefix=/usr/local/mrtg-2
# make
# make install
```

Figura 6.1: Passos para compilação e instalação do MRTG

6.2.3 Configurando o MRTG

Após a instalação dos pacotes, partira-se-á para a criação dos arquivos de configuração dos *hosts* que o MRTG irá monitorar.

Para configurar o MRTG, é preciso inicialmente duas ações:

- Configurar um arquivo com informações sobre o dispositivo;
- Configurar um Agendador de Tarefa para reconhecer as informações do dispositivo.

O MRTG possui um utilitário que auxilia na criação dos arquivos configuração. Trata-se do `cfgmaker`, que possui a seguinte sintaxe:

```
# cfgmaker --output /etc/mrtg/disp_abc_xyz.cfg community@disp.abc.xyz
```

Onde:

- o parâmetro *–output* define o arquivo de configuração que será gerado pelo **cfgmaker**;
- community é a comunidade SNMP do *host*, que é `router.abc.xyz`.

Convém observar que para que o nome do *host* seja resolvido, o equipamento com o MRTG deve estar configurado com o(s) nome(s) do(s) servidor(es) DNS (arquivo `/etc/resolv.conf`); além disso, é necessário registro no servidor DNS para o *host*, associando-o com o respectivo endereço IP. Também pode ser

utilizada uma entrada para o mesmo no arquivo `/etc/hosts` ou, ainda, o endereço IP no lugar do nome.

Para exemplificar, supõe-se que se tem uma placa de rede ***Ethernet* 3COM 3C905C TX/TX-M *Tornado*** com endereço **IP 192.168.1.2** e comunidade *SNMP public*.

A sintaxe do comando cfgmaker seria:

```
# cfgmaker --output /etc/mrtg/eth0_3c905c.cfg public@192.168.1.2
```

Será gerado o arquivo `eth0_3c905c.cfg` como saída (em `/etc/mrtg`). Este arquivo contém as informações necessárias para que sejam gerados os gráficos.

Porém, existem alguns parâmetros que podem ser configurados no arquivo para melhor visualização. Estes parâmetros são definidos nas seções *Global Config Options* e *Global Defaults* e alguns dele estão presente na tabela 6.1.

Tabela 6.1: Opções de Configuração do MRTG

OPÇÃO	DESCRIÇÃO
***WorkDir**:`var/www/mrtg`*	Define qual será a pasta de trabalho do MRTG, ou seja, a pasta onde serão salvos os arquivos gerados pelo MRTG (logs, arquivos html e png, etc). É recomendável criar uma sub-pasta para cada *host*.
***Options [_]:** growright, bits*	São duas opções em uma (mas podem ser configuradas separadamente). O *growright* faz com que o gráfico desloque da direita para a esquerda, fazendo com que o horário atual fique à direita no gráfico; já o parâmetro *bits* define que o gráfico trará as informações em *bits* (por padrão, as informações são expressas em *bytes*).
***Refresh:** 600*	É o tempo, em segundos, em que o navegador irá atualizar a página. Por padrão, 300 segundos (5 minutos).
***Interval:** 10*	É o tempo, em minutos, em que o MRTG irá buscar novas informações estatísticas junto ao*host*. Por padrão, 5 minutos.
***Language:** brazilian*	Linguagem que será utilizada nos arquivos HTML que o MRTG gera.
***RunAsDaemon:** Yes*	Para rodar o MRTG como daemon processo. Ou seja, o MRTG ficará carregado, e vai buscar os dados do *host* conforme o parâmetro *Interval* (ou nos 5 minutos padrão).

6.2.4 Agendando Tarefa para o MRTG

Pode-se programar o agendador de tarefa para executar o MRTG de tempos em tempos. Atualmente o padrão é de 5 minutos, mas como este processo está local, você pode diminuir esse tempo para 3 min ou até mesmo 1 min.

```
# crontab -e
  */5 * * * * /usr/bin/mrtg /home/mrtg/mrtg.conf
```

6.2.5 Execução

Com o arquivo (ou arquivos) de configuração pronto, é hora de colocar o MRTG para monitorar os seus *hosts*. Isso é relativamente fácil. No prompt, basta entrar com o comando:

```
# mrtg /etc/mrtg/eth0_3c905c.cfg
```

Essa linha de comando iniciará o MRTG como root. Se utilizou-se a opção RunAsDaemon, não é necessário acrescentar mais nada.

Porém, será necessário digitar esta linha de comando sempre que a máquina for reinicializada. Além disso, será necessário digitar uma linha de comando para cada arquivo de configuração (caso tenha optado por criar um arquivo por *host*, com eu faço). Para automatizar a tarefa, pode-se criar um script de inicialização, adicioná-lo ao `/etc/init.d` e criar os links simbólicos para os runlevels que serão utilizados. Também é recomendável a criação de um usuário para rodar o MRTG.

Na figura 6.2 pode observar um *script* para inicialização do MRTG.

```
#! /bin/sh
/usr/bin/mrtg --user=mrtg-user /etc/mrtg/eth0_3c905c.cfg
```

Figura 6.2: Exemplo de *script* para iniciar MRTG

Convém lembrar que deve ser adicionada uma linha para cada arquivo CFG, se forem arquivos separados por *hosts*. Uma vez criado o script em `/etc/init.d` (que se sugere nomear `mrtgd`), também é necessário criar os links para os *run-levels* apropriados:

```
# ln -s /etc/init.d/mrtgd /etc/rc3.d/S99mrtgd
```

Se forem exibidas mensagens de erro na criação de alguns arquivos, deve-se verificar as permissões nos diretórios `/etc/mrtg` (caso tenha sido criado) e `/var/lock/mrtg`.

Recomenda-se, ainda, que seja atribuída propriedade destes diretórios ao usuário *mrtg-user*.

6.3 Visualização de Relatórios

O MRTG cria um arquivo HTML referente ao dispositivo em questão. O relatório pode ser visto através de qualquer navegador (cliente HTTP). Normalmente, o mesmo é publicado através de um servidor HTTP, com acesso público ou privado.

Pode-se observar nas figuras 6.3 e 6.4, exemplos de relatórios gerados pelo MRTG para interface de rede.

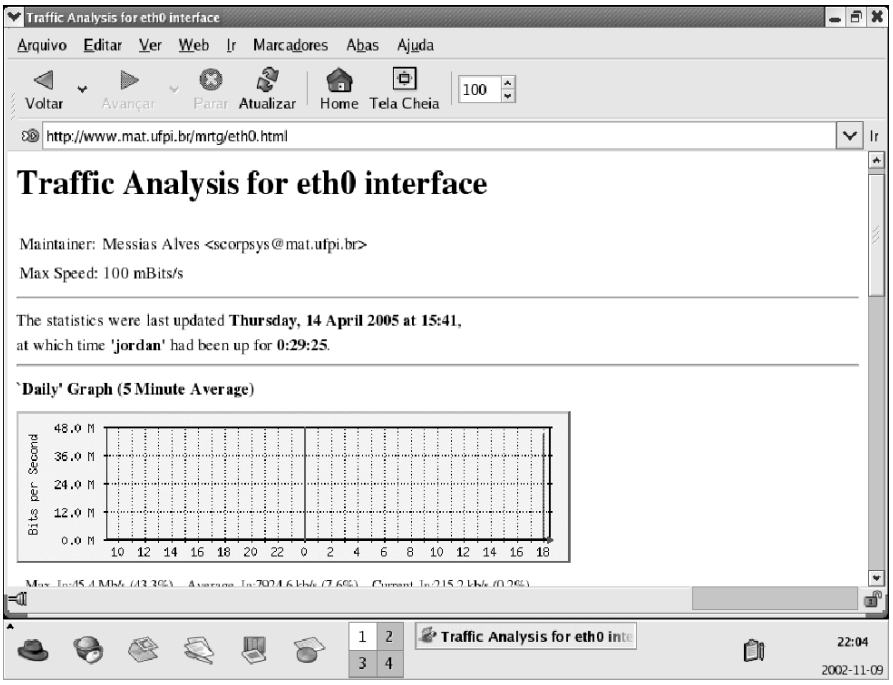

Figura 6.3: Exemplo de relatório diário gerado pela interface eth0 - 3com

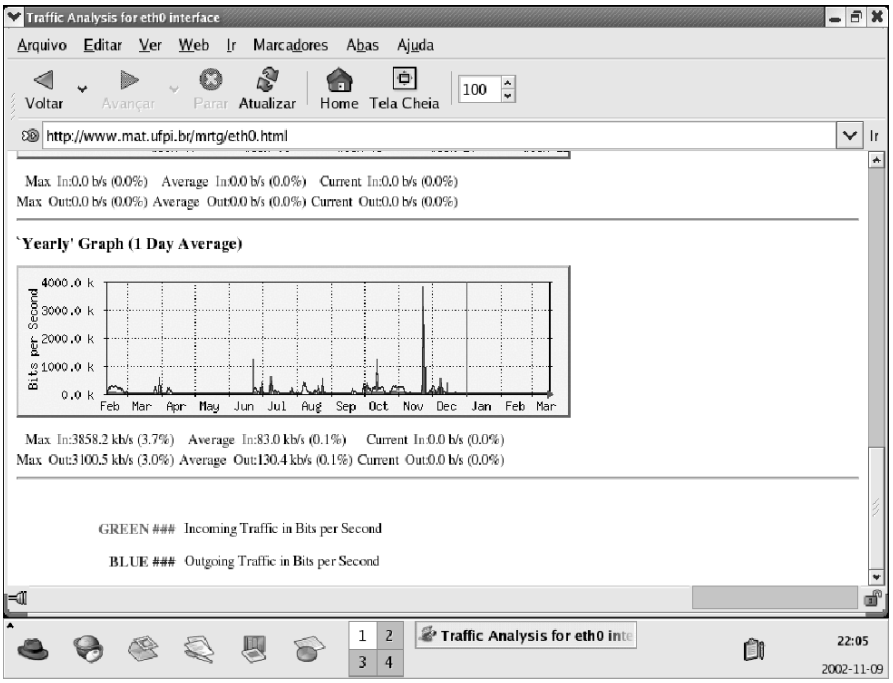

Figura 6.4: Exemplo de relatório anual gerado pela interface eth0 - 3com

6.4 Outras Ferramentas

Há várias outras ferramentas que se destinam a visualização de dados sememlhantes ao MRTG. Entre elas se destaca o **RRDTOOL** é uma ferramenta GPL escrita por *Tobias Oetiker*, o mesmo desenvolvedor do MRTG. Na realidade, o **RRDtool** [1] é uma reimplementação do MRTG, desenvolvida para sobrepor uma série de limitações que o MRTG possui, permitindo se criar gráficos e *loggings* de maneira rápida e flexível. O grande segredo do RRDtool está em sua base de dados, chamada ***RRD (Round Robin Database)*** que permite criar gráficos com qualquer tipo de dados.

Dentre as características, ou melhor, vantagens que o RRDTool oferece, destaca-se:

- Banco de Dados Imutável (tamanho)
- Possui Módulo Perl para integração com interfaces *Web*

[1] Disponível em http://www.rrdtool.org, ou diretamente em http://people.ee.ethz.ch/ oetiker/webtools/rrdtool/pub/?M=D

- Gráficos totalmente personalizaveis
- Pode-se trabalhar com condicionais (IF, ELSE, LT, GT, EQ, entre outros)
- Permite utilização de operadores (+, -, *, /, %)
- Pode-se utilizar funções matemáticas(COS, EXP, LOG, entre outros)

Não foi possível utilizar esta ferramnta neste trabalho, o que se fará em trabalhos futuros. No entanto, na figura 6.5 pode observar um gráfico gerado pelo RRDTool

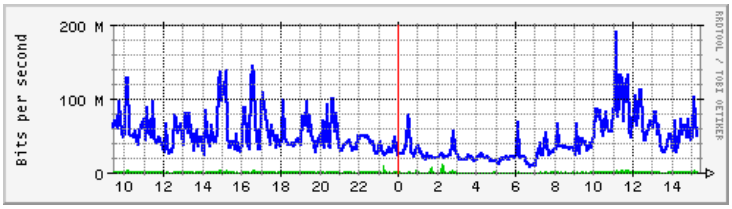

Figura 6.5: Exemplo de Gráfico gerado pelo RRDTool a partir de Roteador

6.5 Considerações Finais

A ferramentas descrita neste capítulo oferece um boa visualização que possibilita tirar conclusões acerca do estado dos equipamentos presentes na rede. Pode-se gerar vários tipos de gráficos e preparar páginas de acordo com a necessidade de cada ambiente.

Capítulo 7

Conclusões

O gerenciamento de rede é um requerimento para qualquer administrador de redes que deseja controlar suas LANs e WANs. Esse novo e vasto império de produtos, projetados para atuar como sistemas cohesivos e bem organizados, pode rapidamente se tornar uma massa desorganizada de dispositivos operacionais independentes.

Para aliviar esses problemas, aplicações de monitoramento de redes baseadas em SNMP devem ser empregadas. O gerenciamento de rede por meio de um sistema baseado em SNMP oferece diversas visões da rede, incluindo uma perspectiva global, uma perspectiva de segmento e uma perspectiva do dispositivo. Também, o gerenciador de rede fornece uma rápida visão de problemas o que possibilita ao administrador a habilidade de ver problemas com uma simples verificação nos estados dos equipamentos através de gráficos gerados por ferramentas.

A necessidade de gerenciamento de redes é evidente e tende a crescer à medida que as redes se tornam maiores e mais complexas. Como essas redes estão cada vez mais heterogêneas, uma padronização do protocolo de gerência garante que estas sejam gerenciadas de maneira uniforme.

De maneira semelhante, com o crescimento na utilização de equipamentos móveis, redes sem infra-estrutura e equipamentos domésticos em rede os protocolos de gerencia precisaram sofrer simplificações em função da carência de processamento e memória RAM disponíveis nesses equipamentos. Quando os equipamentos domésticos estiverem sendo usados será necessária uma preocupação com a segurança no que tange: acesso, autenticação e ações de controle nesses equipamentos.

7.1 Pontos Positivos e Negativos do SNMP

O SNMP tem vários pontos positivos. Um deles é sua popularidade para a gerência de redes TCP/IP. Agentes SNMP estão disponíveis para vários dispositivos de rede, desde computadores até *bridges*, *modems* ou impressoras.

Adicionalmente, o SNMP é um protocolo de gerenciamento flexível e extensível. Pode-se estender os agentes SNMP para cobrir dados específicos de dispositivos, pelo uso de arquivos ASN.1. O SNMP pode assumir numerosos trabalhos específicos para classes de dispositivos fornecendo um mecanismo padrão de controle de rede e monitoramento.

Apesar de seu nome, (***Simple*** *Network Management Protocol*), o SNMP é um protocolo relativamente complexo para implementar. Também, o SNMP não é um protocolo tão eficiente assim. Os modos nos quais são identificadas as variáveis SNMP (como strings de byte onde cada byte corresponde a um nodo particular no banco de dados da MIB) conduz desnecessariamente a grandes pacotes de dados PDU, que consomem partes significativas de cada mensagem de SNMP, sobrecarregando a rede de transmissão de dados.

Enquanto codificações complicadas frustram programadores, a utilização por parte dos usuários finais não dependem da complexidade dos algoritmos que disponibilizam os dados a eles.

7.2 Vantagens ao se utilizar SNMP

Uma das principais vantagens do protocolo SNMP é sem dúvida a sua simplicidade, podendo ser facilmente implementado numa rede de computadores. Muitos dispositivos já vem de fábrica preparados para o uso do SNMP, sendo fácil a configuração dos agentes na rede.

Além disso, é possível também que um gerente de redes programe as variáveis que deseja monitorar, tornando a rede flexível à administração do gerente. Também devido à simplicidade, a maioria dos dispositivos existentes, mesmo os com baixo poder de processamento, podem utilizar o SNMP já que a concentração do processamento ocorre na máquina do gerente.

A rapidez é uma característica do SNMP, pois o gerente não precisa fazer um login no agente e estabelecer uma conexão TP/IP para receber seus dados.

Outra vantagem do protocolo é a sua popularidade. Isso pode ser identificado pelo fato de que a maioria dos fabricantes de dispositivos para redes os projetam para suportar SNMP, inclusive projetando mibs privadas com objetos particulares de cada dispositivo.

Outro benefício do SNMP é a expansibilidade. Por causa de seu projeto simples, é fácil para o protocolo ser atualizado de forma que ele possa ser adequado às necessidades de usuários no futuro.

7.3 Algumas limitações do SNMPv1

O SNMP possui falhas no quesito segurança. Intrusos na rede podem ter acesso a informações dos dispositivos, além de podem modificar as variáveis dos mesmos alterando o funcionamento da rede. Portanto, o SNMP:

- Não é apropriado para o gerenciamento de redes muito grandes, devido a limitação de performance de pooling;
- Traps SNMP não são reconhecidos;
- O padrão SMNP básico provê somente autenticação trivial;
- O modelo SNMP MIB é limitado e não suporta aplicações que questionam o gerenciamento, baseadas em valores ou tipos de objetos;
- Não suporta comunicação *manager-to-manager*.

7.4 O Futuro do SNMP (SNMPv3)

O Grupo de Trabalho do SNMP versão 3 está encarregado de preparar as recomendações para a próxima geração do SNMP. O objetivo deste grupo de trabalho é a de produzir documentos necessários para prover um padrão para a próxima geração de funções SNMP.

Ao longo dos últimos anos, verificou-se um grande número de atividades objetivando incorporar segurança e outros melhoramentos ao SNMP. Infelizmente não houve uma padronização da evolução da versão 2 devido a falta de incorporação dos melhoramentos criando-se assim duas versões que foram chamadas de **V2u** e **V2***.

O grupo de trabalho da versão 3 está trabalhando em novos conceitos baseados nestas duas versões divergentes, para que passe a existir uma documentação única com elementos técnicos incorporado das duas versões.

O SNMPv3 que tem como objetivo principal alcançar a segurança, sem esquecer-se da simplicidade do protocolo, através de novas funcionalidades:

- autenticação e privacidade

- autorização e controle de acesso
- nomes de entidades
- pessoas e políticas
- usernames e gerência de chaves
- destinos de notificações
- relacionamentos proxy
- configuração remota

O SNMPv3 trouxe como principais vantagens aspectos ligados à segurança. Esta segurança busca evitar a alteração das mensagens enviadas. Além disto, barra-se o acesso a elementos estranhos à execução de operações de controle, que são realizadas através da primitiva *SetRequest*. Evita-se também a leitura das mensagens por parte de estranhos, além de se garantir ao gerente o direito de alteração da senha dos agentes. A segurança é conseguida através da introdução de mecanismos de criptografia com o DES (*Data Encryption Standard*) e de algoritmos de autenticação que podem ser tanto o MD5 (*Message Digest Algorithm 5*) quanto o SHA (*Secure Hash Algorithm*).

Contudo, o SNMPv3 ainda não prevê a proteção contra ***Denial of Service***, que impede a troca de informações entre agente e gerente e contra **Análise de tráfego**, que permite a observação do padrão geral do tráfego entre agente e gerente.

7.5 Segurança no Protocolo SNMP

O protocolo SNMP pode realizar operações de reconfiguração na rede alterando características de equipamentos ou até desligando máquinas, sendo que não existe qualquer mecanismo de segurança aplicado ao conteúdo das mensagens.

O controle é feito através da verificação do conteúdo de um campo especial no pacote do SNMP denominada comunidade. A comunidade é definida como sendo o relacionamento entre duas entidades do SNMP. Ela é definida como um conjunto de bytes formando caracteres ASCII que serão utilizados para efetuar este relacionamento.

Dessa forma, quando é realizada a comunicação entre duas entidades do SNMP, a entidade destinatária da mensagem realiza a verificação do conteúdo da comunidade para averiguar se esta informação é proveniente do remetente indicado.

Através da comunidade é possível que o agente realize uma verificação da integridade do agente realizando autenticação e políticas de acesso (agente controla que diferentes gerentes podem obter diferentes variáveis da MIB) através deste campo. O mais importante no entendimento de comunidade, é que sem o conhecimento prévio da comunidade de um determinado equipamento gerenciável, será impossível a qualquer aplicação de gerência acessar as informações da MIB.

Outra consideração importante é que um equipamento pode ter mais de uma comunidade configurada, como uma com direitos de leitura, escrita e *trap*. Portanto, um mesmo equipamento poderá contar com três strings de comunidade diferentes. Isto tem o objetivo de se aumentar a segurança no acesso aos equipamentos.

Por exemplo, o arquivo `snmpd.conf` é onde se pode configurar o acesso que se quer fornecer às estações gerente quando realizam a requisição SNMP. Pode-se limitar o acesso com *read-only* e *read-write*. Enfim, com esse tipo de autenticação, o acesso a MIB se torna pouco seguro, devido principalmente a:

- **Identificação da origem** - a comunidade é transmitida sem qualquer proteção
- **Integridade da mensagem** - ao ser interceptada a mensagem não garante qualquer proteção referente ao conteúdo
- **Tempo-limite** - período de tempo que a mensagem pode ficar presa por algum serviço
- **Privacidade** - qualquer serviço pode monitorar uma comunicação entre entidades SNMP
- **Autorização** - não há controle de autorização de acesso aos dados da MIB

7.6 Resultados Alcançados e Contribuições

Dentro dos objetivos pretendidos por este trabalho, os estudos realizados possibilitaram obter uma grande comprensão acerca de gerenciamento de redes, componentes, soluções, entre outras coisas. Também, mostrou como é possível e fácil escrever MIBs e desenvolver agentes SNMP com o uso de *software* livre, que estão a cada dia mais eficientes. Mostrou-se como o gerenciamento de redes com uso de SNMP em redes TCP/IP torna o trabalho do Administrador de Redes menos fatigante e mais agradável. Pretendia-se experimentar mais outras ferramentas que auxiliam a construção de agentes e o monitoramento de redes. Contudo, adiou-se esta atividade para trabalhos posteriores.

7.7 Trabalhos Futuros

Como futuros estudos, pretende-se aprofundar o estudo das ferramentas utilzadas neste Trabalho e experimentar mais ferramentas, tais como *RRDTool*, *Nagios* entre outras e realizar estudo comparativo de desempenho e funcionalidade entre elas. Deseja-se também fazer um estudo detalhado sobre CMIP, buscando observar semelhanças e diferenças com este outro protocolo. Por fim, elaborar um artigo sobre Gerenciamento de Redes com Softwares Livres e Desenvolvimento de uma ferramenta para interpretação de dados e gráficos.

Referências Bibliográficas

[And04] TANENBAUM Andrew. *Redes de Computadores*. Editora Campus, Santa Clara, 4a. edition, 2004.

[Dav] GUERRERO David. *Network Management and Monitoring with Linux*. Disponível em http://www.develnet.es/ david/papers/snmp/. Último Acesso: 25 de Janeiro de 2005.

[Dav97] PERKINS David. *Understanding SNMP MIBs*. California Evan McGinnis, Santa Clara, 1997.

[DW] STEVENSON Douglas W. *Net Management: what it is and what it isn't*. Disponível em http://www.sce.carleton.ca/netmanage/NetMngmnt/NetMngmnt.html. Último Acesso: 25 de Janeiro de 2005.

[Inc] Anixter Inc. *An Introduction to SNMP*. Disponível em http://www.anixter.com/techlib/buying/network/snmp1.htm. Último Acesso: 25 de Janeiro de 2005.

[RM01] SCHMIDT R. Mauro, DOUGLAS e Kevin J. *SNMP Essencial*. Campus, 2001.

[Tel] DPS Telecom. *SNMP Tutorial Series: 5 Quick Steps to Understanding SNMP and its Role in Network Alarm Monitoring*. Disponível em http://www.dpstele.com/layers/l2/snmp-tutorials.html. Último Acesso: 25 de Janeiro de 2005.

[Wil96] STALLINGS Willian. *SNMP, SNMPv2 and RMON*. Addison Wesley, 2a. edition, 1996.

[Yor] COHEN Yoran. *SNMP simple network management protocol*. Atualização 02/2000. Disponível em

http://wwwdos.uniinc.msk.ru/tech1/1995/snmp/snmp.htm. Último Acesso: 12 de Janeiro de 2005.

Apêndice A

Anexos

A.1 Modelo de MIB para Utilização

Nesta MIB, denominada JM-TESTE-1-MIB, tem-se a utilização de módulos para agrupar objetos. No entanto, para simplificar o exemplo, criou-se apenas o objeto *firstKey*.

```
JM-TESTE-1-MIB DEFINITIONS ::= BEGIN

IMPORTS
     MODULE-IDENTITY,
OBJECT-TYPE,
INTEGER
FROM SNMPv2-SMI;

jmteste MODULE-IDENTITY
    LAST-UPDATED "200503040000Z"
    ORGANIZATION "DM-UFPI"
    CONTACT-INFO
"None yet."
    DESCRIPTION
"AgentX testing MIB"
    REVISION  "200503040000Z"
    DESCRIPTION
"None yet."
    ::= { experimental 72}

firstKey OBJECT-TYPE
    SYNTAX INTEGER (0..100)
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION
"Value initialized to 0 and on each
 access:
 - Return current val.
         - increment"
```

```
    ::= { jmteste 1 }

END
```

A.2 Modelo de MIB em C e o seu Cabeçalho

Tem-se aqui o código de um arquivo em **C** gerado a partir do exemplo inicial deste apêndice. Assim, também pode-se construir MIBs a partir código em linguagem **C**.

```
/*
 *  Template MIB group implementation generated by mib2c - exemplo.c
 *
 */

/* include important headers */
#include <net-snmp/net-snmp-config.h>
#if HAVE_STDLIB_H
#include <stdlib.h>
#endif
#if HAVE_STRING_H
#include <string.h>
#else
#include <strings.h>
#endif

/* needed by util_funcs.h */
#if TIME_WITH_SYS_TIME
# ifdef WIN32
#  include <sys/timeb.h>
# else
#  include <sys/time.h>
# endif
# include <time.h>
#else
# if HAVE_SYS_TIME_H
#  include <sys/time.h>
# else
#  include <time.h>
# endif
#endif

#if HAVE_WINSOCK_H
#include <winsock.h>
#endif
#if HAVE_NETINET_IN_H
#include <netinet/in.h>
#endif

#include <net-snmp/net-snmp-includes.h>
#include <net-snmp/agent/net-snmp-agent-includes.h>
```

```
/* header_generic() comes from here */
#include "util_funcs.h"

/* include our .h file */
#include "example1.h"


#include<net-snmp/library/snmp_debug.h>

long firstKey = 0;

/*********************
 *
 *  Initialisation & common implementation functions
 *
 *********************/

    /*
     * This array structure defines a representation of the
     *  MIB being implemented.
     *
     * The type of the array is 'struct variableN', where N is
     *  large enough to contain the longest OID sub-component
     *  being loaded.  This will normally be the maximum value
     *  of the fifth field in each line.  In this case, the second
     *  and third entries are both of size 2, so we're using
     *  'struct variable2'
     *
     * The supported values for N are listed in <agent/var_struct.h>
     *  If the value you need is not listed there, simply use the
     *  next largest that is.
     *
     * The format of each line is as follows
     *  (using the first entry as an example):
     *     1: EXAMPLESTRING:
     * The magic number defined in the example header file.
     * This is passed to the callback routine and is used
     *   to determine which object is being queried.
     *     2: ASN_OCTET_STR:
     * The type of the object.
     * Valid types are listed in <snmp_impl.h>
     *     3: RONLY (or RWRITE):
     * Whether this object can be SET or not.
     *     4: var_example:
     * The callback routine, used when the object is queried.
     * This will usually be the same for all objects in a module
     *   and is typically defined later in this file.
     *     5: 1:
     * The length of the OID sub-component (the next field)
     *     6: {1}:
     * The OID sub-components of this entry.
     * In other words, the bits of the full OID that differ
     *   between the various entries of this array.
     * This value is appended to the common prefix (defined later)
     *   to obtain the full OID of each entry.
```

```
     */
struct variable2 example_variables[] = {
    { EXAMPLEINTEGER,   ASN_INTEGER,   RWRITE, var_example, 1, {1}}
  };

    /*
     * This array defines the OID of the top of the mib tree that we're
     *  registering underneath.
     * Note that this needs to be the correct size for the OID being
     *  registered, so that the length of the OID can be calculated.
     *  The format given here is the simplest way to achieve this.
     */
  oid example_variables_oid[] = { 1,3,6,1,3,72 };



    /*
     * This function is called at the time the agent starts up
     *  to do any initializations that might be required.
     *
     * In theory it is optional and can be omitted if no
     *  initialization is needed.  In practise, every module
     *  will need to register itself (or the objects being
     *  implemented will not appear in the MIB tree), and this
     *  registration is typically done here.
     *
     * If this function is added or removed, you must re-run
     *  the configure script, to detect this change.
     */
void init_example2(void)
{
    /*
     * Register ourselves with the agent to handle our mib tree.
     * The arguments are:
     *    descr:   A short description of the mib group being loaded.
     *    var:     The variable structure to load.
     * (the name of the variable structure defined above)
     *    vartype: The type of this variable structure
     *    theoid:  The OID pointer this MIB is being registered underneath.
     */
  REGISTER_MIB("example2", example_variables, variable2, example_variables_oid);

}



    /*
     * Define the callback function used in the example_variables structure.
     * This is called whenever an incoming request refers to an object
     *  within this sub-tree.
     *
     * Four of the parameters are used to pass information in.
     * These are:
     *   vp   The entry from the 'example_variables' array for the
     *    object being queried.
     *   name   The OID from the request.
```

```
     *   length  The length of this OID.
     *   exact   A flag to indicate whether this is an 'exact' request
     *    (GET/SET) or an 'inexact' one (GETNEXT/GETBULK).
     *
     * Four of the parameters are used to pass information back out.
     * These are:
     *   name    The OID being returned.
     *   length   The length of this OID.
     *   var_len  The length of the answer being returned.
     *    write_method   A pointer to the SET function for this object.
     *
     * Note that name & length serve a dual purpose in both roles.
     */

u_char *
var_example(struct variable *vp,
    oid *name,
    size_t *length,
    int exact,
    size_t *var_len,
    WriteMethod **write_method)
{
    /*
     *  The result returned from this function needs to be a pointer to
     *    static data (so that it can be accessed from outside).
     *  Define suitable variables for any type of data we may return.
     */
  static unsigned int long_ret;    /* for everything else */



    /*
     * Before returning an answer, we need to check that the request
     *  refers to a valid instance of this object.  The utility routine
     *  'header_generic' can be used to do this for scalar objects.
     *
     * This routine 'header_simple_table' does the same thing for "simple"
     *  tables. (See the AGENT.txt file for the definition of a simple table).
     *
     * Both these utility routines also set up default values for the
     *  return arguments (assuming the check succeeded).
     * The name and length are set suitably for the current object,
     *  var_len assumes that the result is an integer of some form,
     *  and write_method assumes that the object cannot be set.
     *
     * If these assumptions are correct, this callback routine simply
     * needs to return a pointer to the appropriate value (using 'long_ret').
     * Otherwise, 'var_len' and/or 'write_method' should be set suitably.
     */
     //DEBUGMSDGTL(("example","var_example entered\n"));
  if (header_generic(vp, name, length, exact, var_len, write_method) == MATCH_FAILED)
    return NULL;


    /*
     * Many object will need to obtain data from the operating system in
```

```
     *  order to return the appropriate value.  Typically, this is done
     *  here - immediately following the 'header' call, and before the
     *  switch statement. This is particularly appropriate if a single
     *  interface call can return data for all the objects supported.
     *
     * This example module does not rely on external data, so no such
     *  calls are needed in this case.
     */

    /*
     * Now use the magic number from the variable pointer 'vp' to
     *  select the particular object being queried.
     * In each case, one of the static objects is set up with the
     *  appropriate information, and returned mapped to a 'u_char *'
     */
  switch (vp->magic)
    {
    case EXAMPLEINTEGER:
      /*
       * Here the length assumption is correct, but the
       *  object is writeable, so we need to set the
       *  write_method pointer as well as the current value.
       */
      long_ret = firstKey++;
      *var_len = sizeof(long);
      *write_method = write_exampleint;
      return (u_char *) &long_ret;
    default:
      /*
       *  This will only be triggered if there's a problem with
       *   the coding of the module.  SNMP requests that reference
       *   a non-existant OID will be directed elsewhere.
       *  If this branch is reached, log an error, so that
       *   the problem can be investigated.
       */
      //DEBUGMSDGTL(("snmpd", "unknown sub-id %d in examples/var_example\n",
      //vp->magic));
    }
  /*
   * If we fall through to here, fail by returning NULL.
   * This is essentially a continuation of the 'default' case above.
   */
  return NULL;
}

/*********************
 *
 *  Writeable object SET handling routines
 *
 *********************/
int
write_exampleint(int action,
 u_char *var_val,
 u_char var_val_type,
 size_t var_val_len,
 u_char *statP,
```

```
 oid *name,
 size_t name_len)
{
  /* Define an arbitrary maximum permissible value */
#define MAX_EXAMPLE_INT 100
    static long intval;
    static long old_intval;

    switch ( action ) {
case RESERVE1:
  /*
   *  Check that the value being set is acceptable
   */
  if (var_val_type != ASN_INTEGER ) {
    //DEBUGMSDGTL(("example", "%x not integer type", var_val_type));
    return SNMP_ERR_WRONGTYPE;
  }
  if (var_val_len > sizeof(long)) {
    //DEBUGMSDGTL(("example", "wrong length %x", var_val_len));
    return SNMP_ERR_WRONGLENGTH;
  }

  intval = *((long *) var_val);
  if (intval > MAX_EXAMPLE_INT) {
    //DEBUGMSDGTL(("example", "wrong value %x", intval));
    return SNMP_ERR_WRONGVALUE;
  }
  break;

    case RESERVE2:
      /*
       *  This is conventially where any necesary
       *   resources are allocated (e.g. calls to malloc)
       *  Here, we are using static variables
       *   so don't need to worry about this.
       */
      break;

    case FREE:
      /*
       *  This is where any of the above resources
       *   are freed again (because one of the other
       *   values being SET failed for some reason).
       *  Again, since we are using static variables
       *   we don't need to worry about this either.
       */
      break;

    case ACTION:
      /*
       *  Set the variable as requested.
       *   Note that this may need to be reversed,
       *   so save any information needed to do this.
       */
      old_intval = firstKey;
      firstKey          = intval;
```

```
      break;

    case UNDO:
      /*
       *  Something failed, so re-set the
       *   variable to its previous value
       *  (and free any allocated resources).
       */
      firstKey   = old_intval;
      break;

    case COMMIT:
      /*
       *  Everything worked, so we can discard any
       *   saved information, and make the change
       *   permanent (e.g. write to the config file).
       *  We also free any allocated resources.
       *
       *  In this case, there's nothing to do.
       */
      break;

    }
    return SNMP_ERR_NOERROR;
}
```

Tem-se aqui o código do arquivo de cabeçalho gerado a partir do exemplo inicial deste apêndice, juntamente com o arquivo anterior.

```
/*
 *  Template MIB group interface generated by mib2c - example.h
 *
 */

/* Don't include ourselves twice */
#ifndef _MIBGROUP_EXAMPLE_H
#define _MIBGROUP_EXAMPLE_H

    /*
     * We use 'header_generic' from the util_funcs module,
     *  so make sure this module is included in the agent.
     */
config_require(util_funcs)

    /*
     * Declare our publically-visible functions.
     * Typically, these will include the initialization and shutdown functions,
     *  the main request callback routine and any writeable object methods.
     *
     * Function prototypes are provided for the callback routine ('FindVarMethod')
     *  and writeable object methods ('WriteMethod').
     */
extern void init_example(void);
```

```
extern FindVarMethod var_example;
extern WriteMethod write_exampleint;
extern WriteMethod write_exampletrap;
extern WriteMethod write_exampletrap2;

    /*
     * Magic number definitions.
     * These must be unique for each object implemented within a
     *  single mib module callback routine.
     *
     * Typically, these will be the last OID sub-component for
     *  each entry, or integers incrementing from 1.
     *  (which may well result in the same values anyway).
     *
     * Here, the second and third objects are form a 'sub-table' and
     *   the magic numbers are chosen to match these OID sub-components.
     * This is purely for programmer convenience.
     * All that really matters is that the numbers are unique.
     */

#define EXAMPLESTRING 1
#define EXAMPLEINTEGER 21
#define EXAMPLEOBJECTID         22
#define EXAMPLETIMETICKS 3
#define EXAMPLEIPADDRESS        4
#define EXAMPLECOUNTER 5
#define EXAMPLEGAUGE            6
#define EXAMPLETRIGGERTRAP      7
#define EXAMPLETRIGGERTRAP2     8

#endif /* _MIBGROUP_EXAMPLE_H */
```