UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Monitoração de Protocolos de Alto Nível Através da Implementação de um Agente RMON2

por
LUCIO ROSEIRA BRAGA

Dissertação submetida à avaliação, como requisito parcial para
a obtenção do grau de Mestre em
Ciência da Computação

Prof. Liane Rockenbach Tarouco
Orientador

Porto Alegre, dezembro de 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Braga, Lucio Roseira

Monitoração de protocolos de alto nível através da implementação de um Agente RMON2 / por Lucio Roseira Braga. – Porto Alegre: PPGC da UFRGS, 2001.

113f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2001. Orientadora: Tarouco, Liane Rockenbach.

1. MIB RMON2. 2. Gerenciamento. 3. Agente RMON2. 4. Protocolos de alto nível. 5. TCP/IP. I. Tarouco, Liane Rockenbach. II. Título.

Agradecimentos

Quero agradecer, em especial, ao Luciano Paschoal Gaspary pela enorme ajuda que me prestou durante todo o período de desenvolvimento da dissertação.

Ao Edgar e ao Luis pelo apoio técnico durante a fase de desenvolvimento do agente.

Não poderia deixar de prestar uma homenagem a minha família, pois ela é o alicerce da minha vida e de onde vem o incentivo para buscar meus objetos.

Aos meus amigos e a Carla por representarem sentidos da vida.

Por fim, a professora Liane Margarida Rockenbach Tarouco, minha orientadora.

Sumário

Lista de Abreviaturas **6**
Lista de Figuras **7**
Lista de Tabelas **9**
Resumo **10**
Abstract **11**
1 Introdução **12**
1.1 Trabalhos Relacionados **12**
1.2 Objetivos da Dissertação **13**
1.3 Organização do Trabalho **14**
2 A MIB RMON2 ..... **15**
2.1 Grupo *Protocol Directory* **16**
2.1.1 Identificadores de protocolos 17
2.1.2 Parâmetros dos protocolos 19
2.1.3 Os objetos da tabela *protocolDirTable* 20
2.2 O grupo *Protocol Distribution* **21**
2.3 O grupo *Address Map* **23**
2.4 O grupo *Network-Layer Host* **25**
2.5 O grupo *Application-Layer Host* **28**
2.6 O grupo *Network-Layer Matrix* **30**
2.6.1 Estatísticas do tipo origem/destino de protocolos do nível de rede 30
2.6.2 Estatísticas do tipo *topN* de protocolos do nível de rede 33
2.7 O grupo *Application-Layer Matrix* **35**
2.7.1 Estatísticas do tipo origem/destino de protocolos do nível de aplicação 36
2.7.2 Estatísticas do tipo *topN* de protocolos do nível de aplicação 37
2.8 O grupo *User History Collection* **38**
2.9 O grupo *Probe Configuration* **41**
2.9.1 Seqüências de Controle 42
2.9.2 A tabela *Serial Configuration* 42
2.9.3 A tabela *Network Configuration* 43
2.9.4 A tabela *Trap Destination* 43
2.9.5 A tabela *Serial Connection* 44
3 O que se pode obter com a MIB RMON2 ..... **45**
3.1 Administração e estatísticas das atividades do usuário **45**
3.1.1 Volume de acessos 45
3.1.2 Aplicações e protocolos usados 46
3.1.3 Comunicações estabelecidas 48
3.2 Perfil global do uso da rede **49**
3.3 Otimização de usuários e distribuição de recursos **52**
3.4 Gerenciamento de segurança **54**
4 O Agente RMON2 ..... **56**
4.1 Módulo Capturador **57**
4.2 Base de Dados dos Pacotes Capturados **59**
4.3 Base de Dados da MIB RMON2 ..... **61**
4.3.1 Grupo *Protocol Directory* 62
4.3.2 Grupo *Protocol Distribution* 63

4.3.3 Grupo Network-Layer Host ... 65
4.3.4 Grupo Application-Layer Host ... 66
4.3.5 Grupo Network-Layer Matrix ... 67
4.3.6 Grupo Application-Layer Matrix ... 70
4.4 Módulo Conversor ... 71
4.4.1 Populando as tabelas da MIB RMON2 ... 72
4.4.2 Populando as tabelas matrixTopN ... 78
4.5 Extensão do Agente SNMP para a MIB RMON2 ... 79
4.5.1 Agente RMON2 ... 80
5 Exemplos de aplicação do Agente ... 85
5.1 Exemplo 1 ... 85
5.1.1 Visão geral ... 86
5.1.2 Estatísticas de hosts ... 88
5.1.3 Comunicações entre pares de hosts ... 89
5.2 Exemplo 2 ... 91
5.2.1 Primeiro teste ... 92
5.2.2 Segundo teste ... 93
6 Conclusões ... 96
Anexo 1 Estendendo o agente Net SNMP ... 99
Anexo 2 Instalação do Agente RMON2 ... 109
Bibliografia ... 111

Lista de Abreviaturas

API	Application Program Interface
ASC II	American Standard Code for Information Interchange
CCITT	Consultative Committee for International Telegraph and Telephone
CPU	Central Processing Unit
DNS	Domain Name Server
DOD	Department of Defense
HTTP	Hypertext Transport Protocol
HTTPS	Hypertext Transport Protocol Secure
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IP	Internet Protocol
ISO	International Standards Organization
LLC	Logical Link Control
MAC	Medium Access Control
MIB	Management Information Base
OID	Object Identifier
PDU	Protocol Data Unit
POP3	Post Office Protocol
RAM	Random Access Memory
RFC	Request for Comments
RMON	Remote Network Monitoring
SMTP	Simple Mail Transport Protocol
SNAP	Subnetwork Access Protocol
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TFTP	Trivial File Transport Protocol
UDP	User Datagram Protocol
VSNAP	Virtual Subnetwork Access Protocol
WWW	World Wide Web

Lista de Figuras

FIGURA 2.1 – Estrutura da MIB RMON 15
FIGURA 2.2 – Estrutura do grupo *Protocol Directory* 16
FIGURA 2.3 – Formatos de valores dos índices da tabela *protocolDir* 18
FIGURA 2.4 – *ProtocolDirID* da pilha *ether2.ip.tcp.www-http* 18
FIGURA 2.5 – Exemplo de índice da tabela *protocolDir* 19
FIGURA 2.6 – Objetos da tabela *protocolDir* 21
FIGURA 2.7 – Estrutura do grupo *Protocol Distribution* 21
FIGURA 2.8 – Objetos da tabela *protocolDistControl* 22
FIGURA 2.9 – Objetos da tabela *protocolDistStats* 23
FIGURA 2.10 – Estrutura do grupo *Address Map* 23
FIGURA 2.11 – Objetos da tabela *addressMapControl* 24
FIGURA 2.12 – Objetos da tabela *addressMap* 25
FIGURA 2.13 – Estrutura do grupo *Network-Layer Host* 25
FIGURA 2.14 – Objetos da tabela *hlhostcontrol* 27
FIGURA 2.15 – Objetos da tabela *nlhost* 28
FIGURA 2.16 – Estrutura do grupo *Application-Layer Host* 28
FIGURA 2.17 – Objetos da tabela *alhost* 29
FIGURA 2.18 – Estrutura do grupo *Network-Layer Matrix* 30
FIGURA 2.19 – Objetos da tabela *hlmatrixcontrol* 31
FIGURA 2.20 – Objetos da tabela *nlMatrixSD* 32
FIGURA 2.21 – Objetos da tabela *nlMatrixDS* 32
FIGURA 2.22 – Objetos da tabela *nlMatrixTopNControl* 33
FIGURA 2.23 – Objetos da tabela *nlMatrixTopN* 35
FIGURA 2.24 – Estrutura do grupo *Application-Layer Matrix* 36
FIGURA 2.25 – Objetos da tabela *alMatrixSD* 36
FIGURA 2.26 – Objetos da tabela *almatrixDS* 37
FIGURA 2.27 – Objetos da tabela *alMatrixTopNControl* 38
FIGURA 2.28 – Objetos da tabela *alMatrixTopN* 38
FIGURA 2.29 – Estrutura do grupo *User History Collection* 39
FIGURA 2.30 – Objetos da tabela *usrHistoryControl* 40
FIGURA 2.31 – Objetos da tabela *usrHistoryObject* 40
FIGURA 2.32 – Objetos da tabela *usrHistory* 41
FIGURA 2.33 – Estrutura do grupo *Probe Configuration* 42
FIGURA 2.34 – Objetos da tabela *serialConfig* 43
FIGURA 2.35 – Objetos da tabela *netConfig* 43
FIGURA 2.36 – Objetos da tabela *TrapDest* 44
FIGURA 2.37 – Objetos da tabela *serialConnect* 44
FIGURA 3.1 – Exemplo de gráfico retratando a taxa de utilização da rede por dois *hosts* 46
FIGURA 3.2 – Exemplo de gráfico mostrando a taxa de uso dos protocolos baseado no volume total de pacotes observados de um determinado *host* 47
FIGURA 3.3 – Exemplo de gráfico representando a taxa de uso da rede por protocolos para um determinado *host* 48
FIGURA 3.4 – Conexões estabelecidas entre os *hosts* e identificação das aplicações utilizadas 49
FIGURA 3.5 – Taxa de utilização da banda pelos protocolos de aplicação 50
FIGURA 3.6 – Distribuição de protocolos ao longo do tempo 51

FIGURA 3.7 – Taxa de uso da rede por agrupamentos para uma específica aplicação. 51
FIGURA 3.8 – Usuários que mais consomem recursos da rede 52
FIGURA 3.9 – Algoritmo para otimização da localização de usuários e recursos 53
FIGURA 3.10 – Atividades observadas em um *host* servidor de rede..... 54
FIGURA 3.11 – Usuários e seus fluxos de tráfego para um determinado servidor 55
FIGURA 4.1 – Visão esquemática, em camadas, da estrutura do agente 56
FIGURA 4.2 – Desenho esquemático do agente..... 57
FIGURA 4.3 – Fluxograma do módulo capturador..... 59
FIGURA 4.4 – Fluxograma representando o processo para insersão de dados na tabela "protocoldiststats" 74
FIGURA 4.5 – Fluxograma representando o processo para insersão de dados na tabela "nlhost"..... 74
FIGURA 4.6 – Fluxograma representando o processo para insersão de dados nas tabelas "nlmatrixsd" e "nlmatrixds" 75
FIGURA 4.7 – Fluxograma do processo de inserção de dados nas tabelas da MIB RMON2, exceto para as tabelas *TopN*..... 77
FIGURA 4.8 – Fluxograma demonstrando os passos da função manipuladora..... 82
FIGURA 4.9 – Exemplo da tabela "nlhost" com algumas entradas..... 83
FIGURA 5.1 – Cenário da organização monitorada 86
FIGURA 5.2 – Taxa de utilização da rede por protocolos de aplicação 87
FIGURA 5.3 – Taxa de utilização da rede por dois *hosts* 89
FIGURA 5.4 – Protocolos de aplicação e seus respectivos tráfegos durante um certo período para um *host* 89
FIGURA 5.5 – Estações que acessaram o Servidor Proxy..... 90
FIGURA 5.6 – Comunicações entre pares de máquinas juntamente com o protocolo de aplicação utilizado 90
FIGURA 5.7 – Cenário para o teste de desempenho..... 92

Lista de Tabelas

TABELA 3.1 – Exemplo de informação recuperada da tabela *nlHost* 46
TABELA 3.2 – Exemplo de informação recuperada do grupo *alHost* 47
TABELA 3.3 – Exemplo de informações recuperadas da tabela *alMatrixSD* 49
TABELA 3.4 – Exemplo de informação recuperado do grupo *protocolDist*..... 50
TABELA 3.5 – Exemplo de informações recuperadas da tabela *nlMatrixTopN* 52
TABELA 3.6 – Atividades da rede em um certo *host*..... 54
TABELA 4.1 – Detalhamento da tabela "conexao"..... 60
TABELA 4.2 – Tabela "lastchange" 62
TABELA 4.3 – Tabela "protocoldir" 62
TABELA 4.4 – Tabela "protocoldistcontrol"..... 64
TABELA 4.5 – Tabela "protocoldiststats"..... 64
TABELA 4.6 – Tabela "hlhostcontrol"..... 65
TABELA 4.7 – Tabela "nlhost" 66
TABELA 4.8 – Tabela "alhost" 67
TABELA 4.9 – Tabela "nlmatrixsd"..... 67
TABELA 4.10 – Tabela "nlmatrixds"..... 68
TABELA 4.11 – Tabela "nlmatrixtopncontrol"..... 69
TABELA 4.12 – Tabela "nlmatrixtopn" 70
TABELA 4.13 – Tabela "almatrixsd" 70
TABELA 4.14 – Tabela "almatrixds" 71
TABELA 4.15 – Tabela "almatrixtopn" 71
TABELA 4.16 – Exemplo da estrutura que contém os encapsulamentos de protocolo ativos da tabela "protocoldir"..... 73
TABELA 4.17 – Exemplo de entradas na tabela "nlmatrixsd"..... 78
TABELA 4.18 – Tabela "nlmatrixtopn" gerada a partir da tabela "nlmatrixsd" ordenada pelo número de pacotes..... 79
TABELA 4.19 – Estrutura que possui todos os objetos pertencentes a MIB RMON2.. 80
TABELA 4.20 – Respostas enviadas para o gerente após uma consulta a tabela "nlhost" da figura 4.9 84
TABELA 5.1 – Taxa de utilização da rede por protocolos de aplicação 87
TABELA 5.2 – Taxa de uso da rede por agrupamentos..... 88
TABELA 5.3 – Amostra de informação retirada da tabela "nlhost"..... 88
TABELA 5.4 – Pares de estações que mais se comunicam com a respectivo protocolo de aplicação 91
TABELA 5.5 – Primeiro teste com tempo de tráfego de 10 segundos 92
TABELA 5.6 – Primeiro teste com tempo de tráfego de 1 minuto..... 93
TABELA 5.7 – Primeiro teste com tempo de tráfego de 10 minutos 93
TABELA 5.8 – Segundo teste com tempo de tráfego de 10 segundos 93
TABELA 5.9 – Segundo teste com tempo de tráfego de 1 minuto..... 94
TABELA 5.10 – Segundo teste com tempo de tráfego de 10 minutos 94

Resumo

A área de gerência de rede cresce à medida que redes mais seguras e menos vulneráveis são necessárias, e as aplicações que concorrem pelo seu uso, necessitam de alta disponibilidade e qualidade de serviço.

Quando estamos focando a gerência da infra-estrutura física das redes de computadores, por exemplo, a taxa de uso de um segmento de rede, podemos dizer que esse tipo de gerenciamento encontra-se em um patamar bastante sedimentado e testado. Por outro lado, há ainda lacunas para pesquisar na área de gerenciamento de protocolos de alto nível. Entender o comportamento da rede como um todo, conhecer quais *hosts* mais se comunicam, quais aplicações geram mais tráfego e, baseado nessas estatísticas, gerar uma política para a distribuição de recursos levando em consideração as aplicações críticas é um dever nas redes atuais.

O grupo de trabalho IETF RMON padronizou, em 1997, a MIB RMON2. Ela foi criada para permitir a monitoração de protocolos das camadas superiores (rede, transporte e aplicação), a qual é uma boa alternativa para realizar as tarefas de gerenciamento recém mencionadas.

Outro problema para os gerentes de rede é a proliferação dos protocolos de alto nível e aplicações corporativas distribuídas. Devido a crescente quantidade de protocolos e aplicações sendo executados sobre as redes de computadores, os *softwares* de gerenciamento necessitam ser adaptados para serem capazes de gerenciá-los. Isso, atualmente, não é fácil porque é necessário usar linguagens de programação de baixo nível ou atualizar o *firmware* dos equipamentos de monitoração, como ocorre com os *probes* RMON2.

Considerando este contexto, esse trabalho propõe o desenvolvimento de um agente RMON2 que contemple alguns grupos dessa MIB. O agente baseia-se na monitoração protocolos de alto nível e aplicações que são executados sobre o IP (*Internet Protocol*) e *Ethernet* (na camada de enlace).

Além da implementação do agente, o trabalho apresenta um estudo de como obter estatísticas do agente RMON2 e usá-las efetivamente para gerenciar protocolos de alto nível e aplicações.

Palavras-chave: MIB RMON2, gerenciamento, agente RMON2, protocolos de alto nível, TCP/IP

TITLE: "MONITORING OF HIGH-LEVEL PROTOCOLS THROUGH THE IMPLEMENTATION OF A RMON2 AGENT"

Abstract

The network management area grows as safer and less vulnerable networks are needed and applications, that compete to use the network, need higher availability and quality of service.

When we focus on the management of the physical infrastructure of computer networks, for instance the network segment usage rate, we can say that this type of management is already very disseminated and tested. On the other hand, there is still room for research in the high-layer protocol management area. Understanding the behavior of the network as a whole, knowing which hosts communicates more, which applications generate more traffic and, based on these statistics, generate a policy for the distribution of resources taking the critical applications into consideration is a must in the current networks.

The IETF RMON work group standardized, in 1997, the RMON2 MIB. It was created to allow high-layer protocol monitoring (network, transport, application), which is a good alternative to accomplish the management tasks just mentioned.

Other problem to network managers is the proliferation of high-layer protocols and distributed corporate applications. Due to the increasing amount of protocols and applications running over computer networks, the management software need to adapt to be able to manage them. This is not that easy nowadays because it is necessary to use low level programming languages or update the firmware of the monitoring devices, as happens with RMON2 probes.

Considering this context, this work proposes the development of an RMON2 agent that is able to handle some groups of this MIB. The agent is based on the monitoring of high-layer protocols and applications that run over IP (*Internet Protocol*) and Ethernet (in the link-layer).

Besides the implementation of the agent, the work presents how one can get statistics from an RMON2 agent and use them effectively to manage high-layer protocols and applications.

Keywords: RMON2 MIB, management, RMON2 Agent, high-layer protocols, TCP/IP

1 Introdução

Com as tecnologias de transmissão de dados atuais e sua constante evolução no que tange a velocidade da transmissão e a perda de pacotes, a qual se encontra em um patamar muito próximo de zero, possibilitam aos projetistas de softwares desenvolver aplicações distribuídas que utilizem mais recursos da rede.

Todas as novas aplicações somadas as já existentes concorrem pelo uso da banda e, por conseguinte, dependendo do quanto de recursos da rede certas aplicações utilizam, podem acarretar perdas de desempenho daquelas aplicações críticas a uma organização, as quais precisam funcionar dentro de limiares de garantias de recursos da rede. Como a maioria das empresas está, paulatinamente, usufruindo a rede de computadores (Internet) para fechar negócios (correio eletrônico), vender produtos (Servidor Web aliado à banco de dados), realizar reuniões entre as sedes espalhadas pelo mundo (vídeo conferência), entre outros, esses serviços devem possuir uma alta disponibilidade e qualidade de serviço.

Através desse cenário apresentado, podemos entender o quanto é importante estar gerenciando e manutenindo essas redes de computadores. O desafio para os administradores e técnicos se encontra na crescente complexidade e o surgimento de novas tecnologias. O que antigamente era simples, por serem, as redes, bastante homogêneas (a maioria dos equipamentos vinham de um certo fabricante), hoje em dia isso não ocorre. Pelo fato dessa complexidade, os padrões de gerenciamento foram criados e, conseqüentemente, com a difusão do uso desses padrões tanto pelos fabricantes como pelos gerentes, acarretou em um gradual desinteresse pelas aplicações proprietárias.

A gerencia da infra-estrutura física das redes se encontra hoje bem consistentes e testadas, ao passo que o gerenciamento das aplicações e protocolos que são executados sobre ela necessitam ser, ainda, pesquisados.

1.1 Trabalhos Relacionados

Muitas abordagens têm sido propostas tanto no que se refere a gerenciamento de protocolos de alto nível quanto a gerenciamento distribuído. O principal tópico trabalhado em pesquisas sobre gerenciamento de protocolos de alto nível é monitoração. Algumas alternativas simples para a monitoração são os *sniffers* de rede como `tcpdump` [JAC2001] e `snoop` [SUN2001]. Essas ferramentas realizam a captura de pacotes da rede e requerem ferramentas que analisem e correlacionem, de forma *offline*, os dados capturados. *Sniffers* fornecem detalhes sobre a atividade da rede, mas de forma individualizada (por segmento). Analisadores de protocolo, como o `Ethereal` [COM2001], concentram-se no conteúdo de cada pacote e não nas atividades globais da rede. Seguindo essa linha de ferramentas é preciso listar `ntop` [DER99], que possui interface web e destina-se à monitoração e medição de tráfego, incluindo funções que viabilizam a caracterização de tráfego, o cálculo de utilização da rede e da taxa de utilização da rede por protocolo, além da detecção de congestionamento.

Outras soluções, como o `Tivoli Enterprise` [COO99], são intrusivas, pois requerem que as aplicações, ao serem desenvolvidas, façam chamadas especiais a procedimentos de monitoração. Essa abordagem é aplicável para a monitoração das

aplicações desenvolvidas na própria empresa, mas não pode ser usada no gerenciamento de protocolos oriundos de aplicações proprietárias (ex. navegadores e servidores web, clientes e servidores de e-mail). Além disso, é preciso investir na capacitação dos desenvolvedores para a utilização das APIs de monitoração.

Tarouco, Carrilho e Silveira descreveram em [TAR94, CAR94, SIL95] MIBs para a monitoração de protocolos. Tarouco apresentou uma proposta para gerenciamento de correio eletrônico. Carrilho introduziu um modelo para o gerenciamento do protocolo FTP. Silveira apresentou uma MIB genérica para a monitoração de alguns protocolos de alto nível. Essas propostas foram concebidas quando ainda não havia nenhuma iniciativa padronizada nesse sentido. O padrão RMON [WAL95], mais especificamente o RMON2 (*Remote Networking Monitoring Management Information Base version* 2) [WAL97], definido em 1997, agrega grande parte das funcionalidades previstas nos projetos recém citados. Através do RMON2 pode-se monitorar o tráfego com base em endereços do nível de rede, possibilitando conhecer os *hosts* origem e destino reais destes pacotes, mesmo que estes cheguem ou deixem o segmento de rede local via uma interface de um roteador. Além disso, ele pode decodificar e monitorar o tráfego no nível de aplicação e determinar a evolução e padrões de tráfego considerando estes protocolos. Uma das vantagens dessas propostas com relação aos *sniffers* é o desacoplamento entre os equipamentos de monitoração e as ferramentas de visualização (gerentes). Esta característica permite que vários *probes* (anexados em várias sub-redes) possam ser monitorados a partir de um único ponto, ao contrário das ferramentas apresentadas em [JAC2001, SUN2001 e COM2001].

Uma exigência provocada pela rápida proliferação de protocolos e aplicações que são executadas sobre as redes de computadores é que as ferramentas destinadas à monitoração sejam adaptáveis. Infelizmente, boa parte das ferramentas existentes não estão completamente preparadas para permitir a monitoração de novos protocolos e aplicações e operam com base em um conjunto pré-determinado deles. A capacidade para poder observar novos protocolos depende de atualização do *firmware* do equipamento de monitoração, como os probes RMON2, ou da programação em linguagens de baixo nível como a arquitetura proposta por Malan e Jahanian [MAL98] e a ferramenta `ntop`. Nesse último caso, grande parte dos gerentes de rede acaba ignorando a possibilidade de personalizar o que deva ser monitorado devido à complexidade da tarefa.

1.2 Objetivos da Dissertação

Através da necessidade de gerenciar aplicações e protocolos de alto nível, seguir os padrões mundiais de gerenciamento (conjunto de padrões SNMP), ser um software adaptável no sentido de possibilitar a gerência de novas aplicações, esse trabalho visa desenvolver um agente RMON2 que contemple alguns grupos da respectiva MIB.

O agente baseia-se na monitoração de aplicações e protocolos de alto nível que utilizem como protocolo de rede o IP (*Internet Protocol*) e como o protocolo de enlace o *Ethernet*.

Complementando as características do agente anteriormente descritas, a implementação tem, também, como finalidade oferecer a comunidade científica um agente RMON2 *free* e de código aberto para que possa ser usado como base para novas implementações.

Além do desenvolvimento do agente, como um segundo objetivo, mas não menos importante, é a realização de um estudo do que se pode obter na MIB RMON2 para tarefas de gerenciamento da rede no âmbito dos protocolos de alto nível.

1.3 Organização do Trabalho

O capítulo 2 apresenta a MIB RMON2 através de um detalhamento da sua estrutura desde a definição até as funções referentes a cada grupo, tabela e objeto. No capítulo 3 busca-se descrever o que pode ser obtido da MIB RMON2 para fins de gerencia da rede. O capítulo 4 tem como meta à descrição do agente RMON2 implementado. O capítulo 5 descreve alguns exemplos de aplicação do agente elaborados com o intuito de validar o agente RMON2 proposto. O capítulo 6 encerra o trabalho com as conclusões obtidas da dissertação e lista alguns trabalhos futuros que podem ser feitos a partir da implementação, assim como melhorias ao mesmo.

2 A MIB RMON2

A iniciativa de gerenciar redes de computadores vem desde o final da década de 80 quando as redes de computadores tiveram um enorme crescimento. No entanto, a partir das especificações do padrão SNMP [SCH90] e da MIB-II [McC91] ocorridos no inicio da década de 90, o gerenciamento das redes começou a se popularizar. Os padrões se tornaram o mecanismo dominante para o gerenciamento de equipamentos de rede.

Esses equipamentos possuem anexados a si agentes de gerenciamento que tem como função coletar informações sobre o tráfego de rede e estatísticas como o número de pacotes recebidos. Através dessas informações, uma estação de gerenciamento tem uma visão do volume de tráfego tanto de entrada como de saída do dispositivo monitorado, mas desconhece o comportamento do tráfego na rede local como um todo.

Através das especificações dos padrões RMON [WAL95] e RMON2 [WAL97] a visão do tráfego da rede como um todo é solucionado, pois possuem a capacidade de prover um meio efetivo e eficiente para monitorar o comportamento de sub-redes enquanto reduz a carga ambos em outros agentes e em estações de gerenciamento. Os padrões são primariamente uma definição de uma MIB.

Enquanto a RMON tradicional busca a identificação de problemas físicos na rede, visualizando o fluxo de quadros de roteador a roteador, a RMON2 monitora padrões de uso da rede, observando o conteúdo dos pacotes e obtendo informações necessárias para a monitoração de aplicações cliente-servidor e comunicações fim-a-fim.

O padrão RMON2, o qual representa a MIB RMON2, compreende por uma extensão da MIB RMON original, através da adição de novos grupos. A figura 2.1 ilustra a estrutura da MIB RMON juntamente com a MIB RMON2.

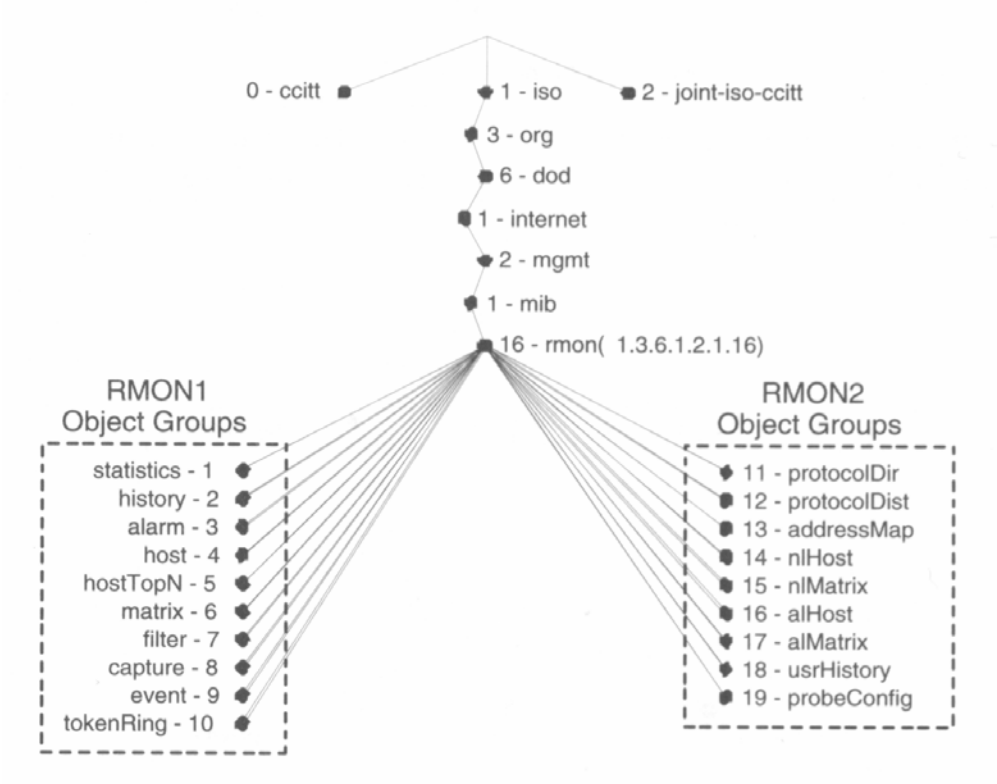


FIGURA 2.1 – Estrutura da MIB RMON

No lado direito da figura estão representados os grupos pertencentes a MIB RMON2 e podem ser resumidos assim:

- *protocol directory (protocolDir)*: é um repositório que indica todos os protocolos que o *probe* é capaz de interpretar;
- *protocol distribution (protocolDist)*: agrega estatísticas sobre o volume de tráfego gerado por cada protocolo, por segmento de rede local;
- *address map (addressMap)*: associa cada endereço de rede ao respectivo endereço MAC, armazenando-os em uma tabela;
- *network-layer host (nlHost)*: coleciona estatísticas sobre o volume de tráfego de entrada e saída das estações com base no endereço do nível de rede;
- *network-layer matrix (nlMatrix)*: provê estatísticas sobre o volume de tráfego entre pares de estações com base no endereço do nível de rede;
- *application-layer host (alHost)*: agrega estatísticas sobre o volume de tráfego de entrada e saída das estações com base em endereços do nível de rede e encapsulamento de protocolo de aplicação (transporte e aplicação);
- *application-layer matrix (alMatrix):* coleciona estatísticas sobre o volume de tráfego entre pares de estações com base no endereço do nível de rede e encapsulamento de protocolo de aplicação (transporte e aplicação);
- *user history collection (usrHistory):* amostra periodicamente objetos especificados pelo usuário (gerente) e armazena as informações coletadas de acordo com parâmetros definidos também pelo usuário;
- *probe configuration (probeConfig):* define parâmetros de configuração padrões para *probes* RMON;

2.1 Grupo *Protocol Directory*

O grupo *protocolDir* é constituído de um objeto escalar simples e uma tabela. Sua estrutura pode ser vista na figura 2.2.

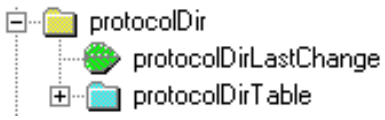


FIGURA 2.2 – Estrutura do grupo *Protocol Directory*

A tabela *protocolDirTable* é um repositório dos protocolos que são conhecidos e podem ser decodificados pelo *probe*. Os *probes* de diferentes fabricantes podem suportar diferentes conjuntos de protocolos.

Um encapsulamento de protocolo para o RMON é uma lista ordenada de protocolos que iniciam com o protocolo da camada MAC. Por exemplo, um protocolo IP sobre um protocolo MAC *Ethernet* é diferente que um protocolo IP sobre um protocolo MAC *Token Ring*. Além disso, um protocolo IP sobre qualquer MAC é um encapsulamento válido, mas não existe um protocolo SNMP sobre qualquer protocolo de transporte. Um encapsulamento é uma lista informal de nomes de protocolos separados com pontos e começando pelo protocolo da camada de enlace. Por exemplo, *ether2.ip.udp* é um informal nome para o protocolo UDP sobre o protocolo IP que está sobre o protocolo MAC *Ethernet*. A especificação formal para os nomes é uma codificada seqüência de octetos. Cada protocolo da respectiva camada requer quatro octetos.

A identificação de um encapsulamento de protocolo é chamado *protocolDirID*.

Cada entrada na tabela *protocolDirTable* é indexada pelos objetos *protocolDirID* e *protocolDirParameters*.

2.1.1 Identificadores de protocolos

O objeto *protocolDirID* contém uma seqüência de octetos que identifica univocamente um protocolo específico. Os identificadores dos protocolos são organizados hierarquicamente, através de uma árvore, de forma similar à hierarquia de objetos da MIB. A raiz da árvore é o identificador de um protocolo do nível de enlace.

Cada nível de protocolo é identificado por uma ou mais seqüências de 32 bits, sendo que cada uma delas é codificada através de quatro sub-identificadores [a.b.c.d], onde cada sub-identificador é representado por um octeto. Por exemplo, o identificador para *Ethernet* é o valor 1 (hexadecimal), codificado como [0.0.0.1] e referenciado simbolicamente como *ether2*. A seguir, são apresentados outros identificadores já atribuídos a protocolos do nível de enlace:

- ether2 =1 [0.0.0.1]
- llc =2 [0.0.0.2]
- snap =3 [0.0.0.3]
- vsnap =4 [0.0.0.4]
- ianaAssigned =5 [0.0.0.5]

Abaixo de cada um destes nodos na árvore de identificadores, encontram-se protocolos que são diretamente encapsulados pelo protocolo do nível de enlace. Esta estrutura se estende por tantos níveis quantos forem necessários. Por exemplo, o protocolo IP pode ser executado diretamente sobre o protocolo de enlace *Ethernet*. Simbolicamente, o identificador para IP executando sobre *Ethernet* é *ether2.ip*. De forma análoga, o protocolo TCP executa sobre o protocolo IP e é encapsulado por ele. Assim, TCP executando sobre IP em um segmento de rede local *Ethernet* é representado por *ether2.ip.tcp*. Vale ressaltar ainda que diversos protocolos do nível de aplicação executam sobre TCP, sendo o HTTP um deles. Neste caso, considerando o mesmo ambiente do exemplo anterior, tem-se *ether2.ip.tcp.www-http* como identificador.

A estrutura do objeto *protocolDirID* é ilustrada na figura 2.3b (o objeto *protocolDirParameters* é tratado na próxima seção). Os valores que este objeto pode assumir consistem de um contador (um octeto), que indica o número de octetos do restante do identificador, seguido por uma seqüência de 4*x*N octetos, onde N é o número de identificadores de protocolos necessários para representar o encapsulamento completo do protocolo em questão. De forma geral, cada identificador é formado por quatro octetos. Uma exceção, contudo, é o protocolo *vsnap*, representado em oito octetos.

Embora na figura 2.3b seja ilustrado um identificador com quatro protocolos encapsulados, na prática podem ser representados quaisquer combinações válidas de encapsulamento.

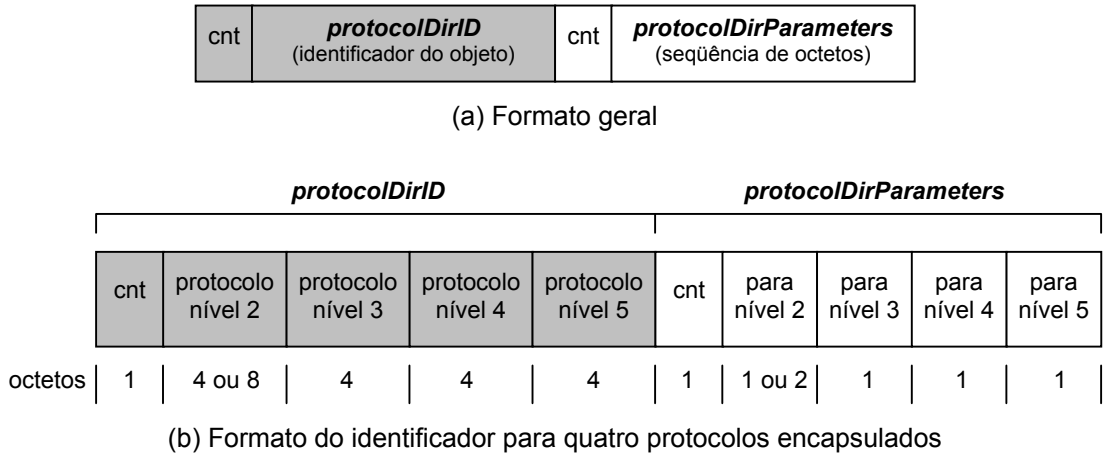


FIGURA 2.3 – Formatos de valores dos índices da tabela *protocolDir*

Já foi salientado que o identificador para o *Ethernet* é [0.0.0.1]. Cada protocolo encapsulado pelo quadro *Ethernet* é identificado por um número de 32 bits no formato [0.0.a.b], onde a e b contêm um valor de 16 bits que coincide com o do campo *Type* do quadro *Ethernet*. Este campo é usado para identificar o protocolo encapsulado no quadro. No caso do IP, este número é [0.0.8.0].

De maneira análoga, o datagrama IP contém o campo *Protocol*, que identifica o protocolo usuário do IP. Este valor é codificado no formato [0.0.0.a]. Na definição do padrão, o valor atribuído ao TCP é 6. Assim, o identificador para o protocolo TCP executando sobre o IP é [0.0.0.6].

Por fim, a PDU do protocolo TCP possui o campo *Port*, de 16 bits, que identifica o protocolo usuário do TCP. O valor atribuído ao protocolo HTTP é 80. Novamente, cada valor de 16 bits é codificado na forma [0.0.a.b] e, portanto, HTTP sobre TCP é codificado como [0.0.0.80]. Agregando os valores apresentados, o *protocolDirID* que identifica univocamente HTTP executando sobre TCP/IP em um segmento de rede local *Ethernet* é ilustrado na figura 2.4.

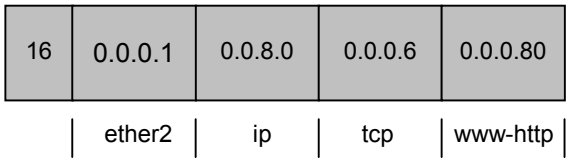


FIGURA 2.4 – *ProtocolDirID* da pilha *ether2.ip.tcp.www-http*

É válido salientar que é necessária uma entrada na tabela para cada protocolo a ser decodificado pelo *probe*. Por exemplo, supondo que ele seja capaz de:

- Interpretar todos os quadros MAC *Ethernet*;
- Desencapsular um datagrama IP do quadro *Ethernet* e interpretá-lo;
- Desencapsular uma PDU TCP do datagrama IP e interpretá-la;
- Desencapsular uma PDU HTTP da PDU do TCP e interpretá-la.

Neste caso, são necessárias quatro entradas em *protocolDirEntry*. Os valores (*protocolDirID*) correspondentes são:

- ether2 (4.0.0.0.1)
- ether2.ip (8.0.0.0.1.0.0.8.0)
- ether2.ip.tcp (12.0.0.0.10.0.8.0.0.0.0.6)

- ether2.ip.tcp.www-http (16.0.0.0.1.0.0.8.0.0.0.0.6.0.0.0.80)

2.1.2 Parâmetros dos protocolos

O objeto *protocolDirParameters*, juntamente com o protocolo *protocolDirID*, permite indexar elementos da tabela *protocolDir*. Ele contém informações sobre a capacidade do *probe* com relação a um protocolo particular. Essas informações são codificadas em um valor, estruturado da seguinte forma: um octeto para o campo *cnt*, que indica o número de octetos restantes, seguido de outros N octetos, um para cada nível de protocolo.

A figura 2.3a mostra a estrutura geral dos índices da tabela *protocolDir*: *protocolDirID* e *protocolDirParams*. A figura 2.3b, por sua vez, ilustra esta mesma estrutura considerando a identificação de um protocolo no quarto nível de encapsulamento. É válido observar que a correspondência entre níveis de protocolos e parâmetros é um para um.

Cada um dos bits nos octetos de parâmetros define uma capacidade particular. Os quatro bits menos significativos possuem valores reservados, que se aplicam a todos os protocolos:

- *countsFragments* (bit 0): protocolos de níveis superiores encapsulados no protocolo em questão serão contabilizados corretamente mesmo se este protocolo fragmentar as PDUs de níveis superiores;
- *trackSessions* (bit 1): contabiliza corretamente todos os pacotes de protocolos que iniciam uma seção em uma porta previamente conhecida e então a transferem dinamicamente para uma outra pela duração da seção (protocolo TFTP, por exemplo).

Continuando o mesmo exemplo da seção anterior, a codificação dos dois objetos: *protocolDirID* e *protocolDirParameters*, ilustrada na figura 2.5, forma um índice para a tabela *protocolDir*. No exemplo, o índice referencia o protocolo HTTP executando sobre TCP/IP/*Ethernet*, com fragmentos contabilizados corretamente para o protocolo IP e os de níveis superiores.

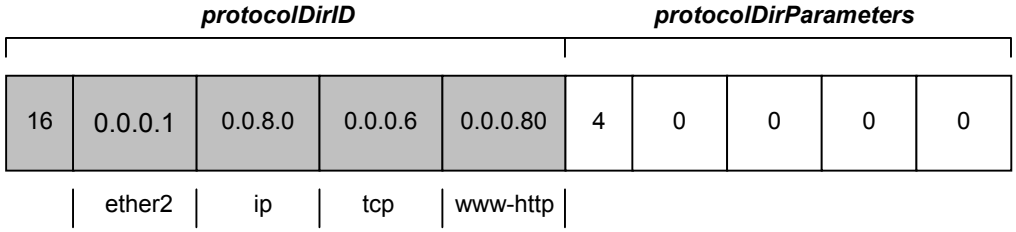


FIGURA 2.5 – Exemplo de índice da tabela *protocolDir*

A seguir, são apresentados outros exemplos de índices para esta tabela:

- ether2.ip.udp.snmp (16.0.0.0.1.0.0.8.0.0.0.0.17.0.0.0.161.4.0.0.0.0)
- snap.ip.tcp.telnet (16.0.0.0.3.0.0.8.0.0.0.0.6.0.0.0.23.4.0.0.0.0)

2.1.3 Os objetos da tabela *protocolDirTable*

Na figura 2.6 pode ser vista a tabela *protocolDir* expandida, mostrando os objetos que fazem parte da mesma e eles podem ser detalhados assim:

- *ProtocolDirID*: é a identificação de um encapsulamento de protocolo sendo do tipo OCTET STRING;
- *ProtocolDirParameters*: especifica parâmetros para o encapsulamento de protocolo. Seu tipo é OCTET STRING;
- *ProtocolDirLocalIndex*: é um inteiro positivo que identifica o protocolo em outras tabelas da MIB;
- *ProtocolDirDescr*: especifica uma descrição textual do encapsulamento de protocolo. Esse objeto não pode ser modificado se o associado objeto *protocolDirStatus* estiver no estado *active* (valor igual a 1);
- *ProtocolDirType*: especifica dois atributos do encapsulamento. Os atributos são – *extensible* (0) que identifica que o encapsulamento pode ser estendido, isto é, um novo protocolo baseado no protocolo em questão pode ser criado pela estação de gerenciamento. *AddressRecognitionCapable* (1) especifica que o encapsulamento é um protocolo da camada de rede e que o *probe* pode determinar endereços da camada de rede;
- *ProtocolDirAddressMapConfig*: especifica se o mapeamento de endereços é suportado por esse encapsulamento. Os valores permitidos são – *notSupported* (1), o encapsulamento não é um protocolo da camada de rede ou o *probe* não suporta o mapeamento de endereços para o encapsulamento. *SupportedOff* (2), o mapeamento de endereços é suportado, mas não está ativo. *SupportedOn* (3), o endereçamento de endereços é suportado e o mesmo está ativo. Caso o valor do objeto seja *notSupported* (1) o *probe* não pode permitir que o valor do objeto seja modificado. Quando o valor do objeto mudar de *SupportedOn* (3) para *SupportedOff* (2), todas as entradas na tabela de mapeamento de endereços relacionadas à entrada desse objeto devem ser deletadas;
- *ProtocolDirHostConfig*: especifica se as tabelas *host* do RMON2 são suportadas por esse encapsulamento. Os valores permitidos são – *notSupported* (1), o *probe* não suporta as tabelas *host* do RMON2 para o encapsulamento. *SupportedOff* (2), as tabelas *host* do RMON2 são suportadas, mas não estão ativas. *SupportedOn* (3), as tabelas *host* do RMON2 são suportadas e as mesmas estão ativas. Se o objeto estiver com o valor *notSupported* (1), o *probe* não pode permitir que o valor seja modificado. Quando o valor do objeto mudar de s*upportedOn* (3) para s*upportedOff* (2), todas as entradas nas tabelas *nlHost* e *alHost* relacionadas à entrada desse objeto devem ser deletadas. Uma observação diz respeito a cada entrada da tabela *alhost*. Como cada entrada referencia duas entradas da tabela *protocolDir* (uma para o encapsulamento da camada de rede e outra para o encapsulamento de alto nível), a mesma só será criada se o valor do objeto for *supportedOn* (3) para ambas;
- ProtocolDirMatrixConfig: especifica se as tabelas *matrix* do RMON2 são suportadas por esse encapsulamento. Os valores permitidos são – *notSupported* (1), o *probe* não suporta as tabelas *matrix* do RMON2 para o encapsulamento. *SupportedOff* (2), as tabelas *matrix* do RMON2 são suportadas, mas não estão ativas. *SupportedOn* (3), as tabelas *matrix* do RMON2 são suportadas e as mesmas estão ativas. Se o objeto estiver com o valor *notSupported* (1), o *probe* não pode permitir que o seu valor seja modificado. Quando o valor do objeto

mudar de s*upportedOn* (3) para s*upportedOff* (2), todas as entradas nas tabelas *nlMatrix* e *alMatrix* relacionadas à entrada desse objeto devem ser deletadas. Uma observação diz respeito a cada entrada da tabela *alMatrix*. Como cada entrada referencia duas entradas da tabela *protocolDir* (uma para o encapsulamento da camada de rede e outra para o encapsulamento de alto nível), a mesma só será criada se o valor do objeto for *supportedOn* (3) para ambas;

- *ProtocolDirOwner*: entidade que configurou essa entrada. Esse objeto pode ser alterado;
- *ProtocolDirStatus*: estado dessa entrada. Uma entrada não pode estar com o estado *active* (1) ao menos que todos os campos da entrada tenham um valor coerente. Se o valor do objeto for diferente de *active*, então todas as associadas entradas das tabelas *nlHost*, *nlMatrixSD*, *nlMatrixDS*, *alHost*, *alMatrixSD* e *alMatrixDS* devem ser excluídas. Os possíveis estados para esse objeto, assim como para qualquer objeto status das demais tabelas podem ser: *active* (1), *notInService* (2), *notReady* (3), *createAndGo* (4), *createAndWait* (5) ou *destroy* (6).

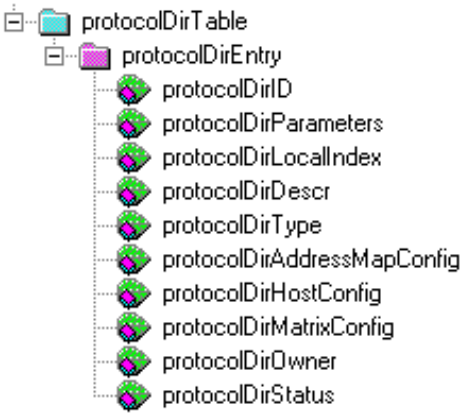


FIGURA 2.6 – Objetos da tabela *protocolDir*

Além da tabela *protocolDirTable* o grupo *protocolDir* contém o objeto escalar *protocolDirLastChange*. O valor desse objeto é um *timestamp* que indica quando uma entrada da tabela *protocolDir* foi adicionada ou deletada pela última vez, ou quando os valores dos objetos *protocolDirAddressMapConfig*, *protocolDirHostConfig*, *protocolDirMatrixConfig* foram modificados.

2.2 O grupo *Protocol Distribution*

O grupo *Protocol Distribution (protocolDist)* é formado por uma tabela de controle e uma tabela de dados que determinam os encapsulamentos de protocolo usados em uma rede monitorada. A representação visual do grupo pode ser visto na figura 2.7.

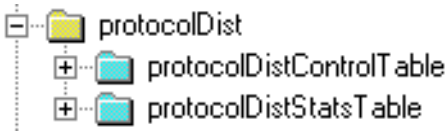


FIGURA 2.7 – Estrutura do grupo *Protocol Distribution*

A tabela de controle possui uma entrada para cada segmento de rede que se está analisando. Quando se fala em segmento de rede está intrínseco uma interface de rede única. A tabela é indexada por um objeto, o *protocolDistControlIndex*. Para fins de exemplificação, um objeto indexado dessa tabela tem o seguinte formato:

- protocolDistControlDataSource.1

A tabela de controle expandida, isto é, mostrando os objetos pertencentes ao mesmo pode ser vista na figura 2.8. Já os objetos da tabela são caracterizados conforme segue:

- *ProtocolDistControlIndex*: Valor inteiro que permite identificar de forma unívoca uma entrada nesta tabela. Este mesmo valor é utilizado para indexar as entradas correspondentes na tabela *protocolDistStats*.
- *ProtocolDistControlDataSource*: Identifica a interface e, portanto, a sub-rede que é a fonte dos dados para esta entrada. Seu valor não pode ser modificado se o objeto *protocolDistContrtolStatus* da respectiva entrada estiver com o valor *active* (1);
- *ProtocolDistControlDroppedFrames*: Número total de quadros recebidos pela interface em questão que o *probe* opta por não contabilizar. Isto ocorre quando lhe faltam recursos (memória, capacidade de processamento).
- *ProtocolDistControlCreateTime*: Possui o valor do objeto s*ysUpTime* quando esta entrada foi ativada. O objeto pode ser usado pela estação de gerenciamento para assegurar que a tabela não foi deletada e recriada entre requisições.
- *ProtocolDistControlOwner*: entidade que configurou essa entrada. Esse objeto pode ser alterado;
- *ProtocolDistControlStatus*: o estado dessa entrada. Uma entrada não pode existir no estado *active* (1) ao menos que todos os objetos da entrada contenham um valor correto. Caso esse objeto tenha o valor diferente de *active*, então todas as entradas associadas a essa entrada na tabela *protocolDistStats* devem ser excluídas.

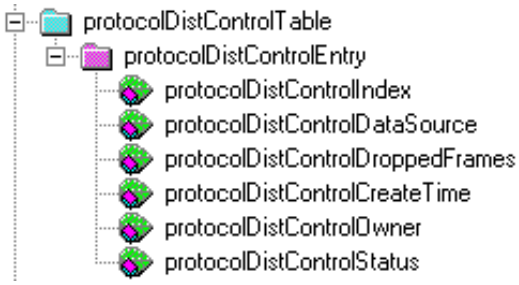


FIGURA 2.8 – Objetos da tabela *protocolDistControl*

Já a tabela de dados (vide figura 2.9) contém a quantidade de tráfego (número de pacotes e octetos) observado na monitoração por cada um dos encapsulamentos de protocolos do grupo *protocolDir* suportados e para o qual pelo menos um pacote tenha sido observado.

As estatísticas são geradas a partir dos pacotes sem erros e dos octetos que os mesmos contém. Nota-se que as contabilizações são feitas para encapsulamentos e não para protocolos. Para explicar melhor, um simples pacote pode gerar o incremento de vários contadores. Por exemplo, um pacote HTTP sobre o TCP sobre o IP sobre o *Ethernet* poderá resultar nos incrementos dos contadores de pacotes e octetos para os encapsulamentos de protocolos *ether2.ip.tcp.www-http*, *ether2.ip.tcp* e *ether2.ip*.

FIGURA 2.9 – Objetos da tabela *protocolDistStats*

As entradas na tabela de dados são indexadas por uma entrada da tabela de controle (*protocolDistControlIndex*) e pelo objeto *protocolDirLocalIndex* da tabela *protocolDir*. O objeto *protocolDirLocalIndex* deve ser usado para distinguir qual o encapsulamento de protocolo que está sendo monitorado.

A tabela de dados possui os seguintes objetos:

- *ProtocolDistStatsPkts*: número de pacotes recebidos sem erros para o tipo de protocolo;
- *ProtocolDistStatsOctets*: número total de octetos dos pacotes recebidos sem erros para o tipo de protocolo.

2.3 O grupo *Address Map*

O grupo *Address Map* associa cada endereço de rede ao endereço correspondente da sub-camada MAC. Este grupo é formado por três objetos escalares, uma tabela de controle (*addressMapControlTable*) e uma tabela de dados (*addressMapTable*), conforme ilustrado na figura 2.10.

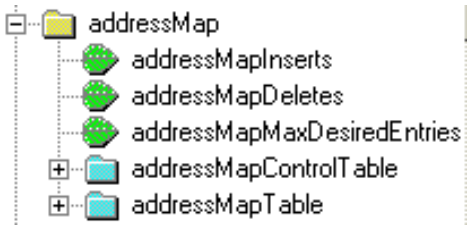


FIGURA 2.10 – Estrutura do grupo *Address Map*

Os três objetos escalares são:

- *addressMapInserts*: número de vezes em que foram acrescentadas entradas de mapeamento de endereços na tabela de dados. Se uma entrada é inserida, depois deletada e novamente inserida, o valor do objeto é incrementado 2 vezes;
- *addressMapDeletes*: número de vezes em que foram removidas entradas de mapeamento de endereços da tabela de dados. Se uma entrada é deletada, depois inserida e novamente deletada, o valor do objeto é incrementado 2 vezes;
- *addressMapMaxDesiredEntries*: número máximo admissível de entradas na tabela de dados. Se o valor for igual a -1, o *probe* pode criar qualquer número de entradas.

Para se ter o tamanho da tabela de dados em um certo instante qualquer, realiza-se uma subtração do valor do objeto *addressMapDeletes* do valor do objeto *addressMapInserts*.

A maioria das tabelas de controle dos grupos RMON2 possuem um objeto *MaxDesiredEntries*. Como o volume de dados manipulados pelos agentes RMON2 é muito grande, esta característica é importante, pois permite ao gerente controlar como

os recursos são alocados, configurando os valores associados a estes objetos da forma mais apropriada.

A estrutura das tabelas de controle e de dados no grupo *Address Map* difere da utilizada nos outros grupos, onde cada entrada na tabela de controle cria e controla um conjunto de entradas na tabela de dados. Ao contrário, no grupo *Address Map* há uma única tabela de dados, cujas entradas provêem mapeamentos entre endereços do nível de rede e endereços MAC. Na tabela de controle, existe uma entrada para cada sub-rede associada ao *probe*. Cada entrada habilita tanto a descoberta de endereços em uma nova sub-rede como a adição de mapeamentos de endereços na tabela de dados. Portanto, a tabela de dados não é indexada por uma entrada da tabela de controle.

Esta forma de organização simplifica o trabalho a ser realizado pela aplicação de gerenciamento. Por exemplo, esta aplicação é capaz de observar duplicações diretamente na tabela central, ao invés de ter que procurá-las em tabelas separadas.

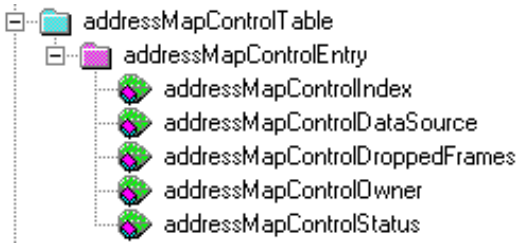


FIGURA 2.11 – Objetos da tabela *addressMapControl*

Os objetos da tabela *addressMapControlTable* são compreendidos conforme segue:

- *AdressMapControlIndex*: Valor inteiro que identifica de forma única uma entrada na tabela *addressMapControl*;
- *AddressMapControlDataSource*: Identifica a interface e, portanto, a sub-rede que é a fonte dos dados para esta entrada. Seu valor não pode ser modificado se o objeto a*ddressMapControlStatus* da respectiva entrada estiver com o valor *active* (1);
- *AddressMapControlDroppedFrames*: Número total de quadros recebidos para esta interface que o *probe* opta por não contabilizar. Isto ocorre quando lhe faltam recursos (memória, capacidade de processamento);
- *AddressMapControlOwner*: entidade que configurou essa entrada. O dado do objeto pode ser modificado;
- *AddressMapControlStatus*: o estado dessa entrada. Uma entrada não pode existir no estado *active* (1) ao menos que todos os objetos da entrada tenham um valor correto. Caso esse objeto tenha o valor diferente de *active*, então todas as entradas associadas a essa entrada na tabela *addressMap* devem ser excluídas.

A tabela *addressMapTable* armazena os mapeamentos de endereços descobertos a partir da análise de quadros MAC livres de erros. São criadas entradas para todos os protocolos, cujos valores do objeto *protocolDirAddressConfig* na tabela *protocolDir* sejam igual a *supportedOn*(3). Quando o valor do objeto *protocolDirAddressMapConfig* for configurado para *supportedOff* (2) ou a entrada do protocolo é deletada da tabela *protocolDir*, todas as correspondentes entradas da tabela de dados ao qual refere-se o encapsulamento de protocolo devem ser deletadas.

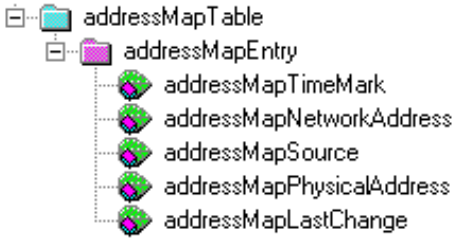


FIGURA 2.12 – Objetos da tabela *addressMap*

Os objetos da tabela *addressMap* são apresentados abaixo.

- *AdressMapTimeMark*: Contém o valor em *TimeTicks* (contabiliza em centenas de segundos) e que representa quando a entrada foi atualizada pela última vez. Seu valor não pode ser recuperado, sendo apenas usado para indexação;
- *AddressMapNetworkAddress*: Endereço do nível de rede para esta entrada;
- *AddressMapSource*: Última interface ou porta do repetidor na qual o endereço de rede foi observado;
- *AddressMapPhysicalAddress*: Último endereço MAC origem para o qual o endereço de rede foi observado;
- *AddressMapLastChange*: Possui o valor do objeto s*ysUpTime*, correspondente ao instante da última atualização desta entrada.

Esta tabela é indexada por quatro objetos: o filtro de tempo, a identificação do protocolo, o endereço de rede e a interface onde o foi observado (*addressMapTimeMark*, *protocolDirLocalIndex*, *addressMapNetworkAddress* e *addressMapSource*). Assim, dado um endereço de rede de um protocolo particular observado em uma determinada interface em um intervalo de tempo, é possível obter o endereço MAC correspondente.

É esperada a existência de um único endereço MAC (apenas uma entrada na tabela) para um endereço de rede qualquer. Por outro lado, para um endereço MAC podem estar associados diversos endereços de rede.

2.4 O grupo *Network-Layer Host*

O grupo *Network-Layer Host* (*nlHost*) permite decodificar pacotes com base em seus endereços de rede. Como conseqüência, gerentes podem observar além dos roteadores que interligam as sub-redes e identificar as reais estações que estão se comunicando. Este grupo coleta estatísticas similares às do grupo *host* da MIB RMON1. A diferença é que o grupo *nlHost* faz esta coleta com base no endereço de rede e não no endereço MAC.

Duas tabelas formam o grupo, sendo uma de controle (*hlHostControlTable*) e uma de dados (*nlHostTable*) (vide figura 2.13).

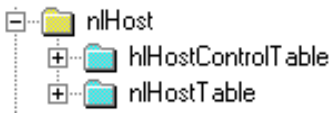


FIGURA 2.13 – Estrutura do grupo *Network-Layer Host*

A tabela de controle é composta pelos objetos apresentados abaixo, e os mesmos podem ser vistos na figura 2.14.

- *HlHostControlIndex*: Valor inteiro que identifica de forma única uma entrada na tabela *hlHostControl* (as entradas nesta tabela referem-se a uma única interface do monitor). Este índice é também utilizado para identificar entradas correspondentes nas tabelas *nlHost* e *alHost*;
- *HlHostControlDataSource*: Identifica a interface e, portanto, a sub-rede que é fonte de informação para as tabelas de dados (*nlHost* e *alHost*). O valor do objeto não pode ser modificado se o objeto *hlHostControlStatus* da respectiva entrada estiver com o valor *active*(1);
- *HlHostControlNlDroppedFrames*: Número total de quadros recebidos pela interface identificada no objeto *hlHostControlDataSource*, que o *probe* opta por não contabilizar na tabela *nlHost*;
- *HlHostControlNlInserts*: Número de vezes em que foram acrescentadas entradas na tabela de dados *nlHost*. Caso uma entrada seja inserida, depois deletada e novamente inserida, o valor do campo é incrementado 2 vezes;
- *HlHostControlNlDeletes*: Número de vezes em que foram removidas entradas da tabela de dados *nlHost.* Caso uma entrada seja deletada, depois inserida e novamente deletada, o valor do campo é incrementado 2 vezes;
- *HlHostControlNlMaxDesiredEntries*: Número máximo admissível de entradas na tabela *nlHost*. Se o valor for configurado para –1, o *probe* pode adicionar quantas entradas quanto forem necessárias. O valor do objeto não pode ser modificado se o objeto *hlHostControlStatus* da respectiva entrada estiver com o valor *active*(1);
- *HlHostControlAlDroppedFrames*: Número total de quadros recebidos pela interface identificada no objeto *hlHostControlDataSource*, que o *probe* opta por não contabilizar na tabela *alHost*;
- *HlHostControlAlInserts*: Número de vezes em que foram acrescentadas entradas na tabela de dados *alHost*. Assim como no objeto hlHostControlNlInserts, se uma entrada seja inserida, depois deletada e novamente inserida, o valor do campo é incrementado 2 vezes;
- *HlHostControlAlDeletes*: Número de vezes em que foram removidas entradas da tabela de dados *alHost*. Assim como no objeto *hlHostcontrolNlDeletes*, se uma entrada for deletada, depois inserida e novamente deletada, o valor do campo é incrementado 2 vezes;
- *HlHostControlAlMaxDesiredEntries*: Número máximo admissível de entradas na tabela *alHost*. Se o valor for configurado para –1, o *probe* pode adicionar quantas entradas quanto forem necessárias. O valor do objeto não pode ser modificado se o objeto *hlHostControlStatus* da respectiva entrada estiver com o valor *active*(1);
- *HlHostControlOwner*: entidade que configurou essa entrada. O dado desse objeto pode ser modificado;
- *HlHostControlStatus*: o estado dessa entrada. Uma entrada não pode existir no estado *active* (1) ao menos que todos os objetos da entrada tenham um valor coerente. Caso esse objeto tenha o valor diferente de *active*, então todas as entradas associadas a essa entrada nas tabelas *nlHost* e *alHost* devem ser excluídas.

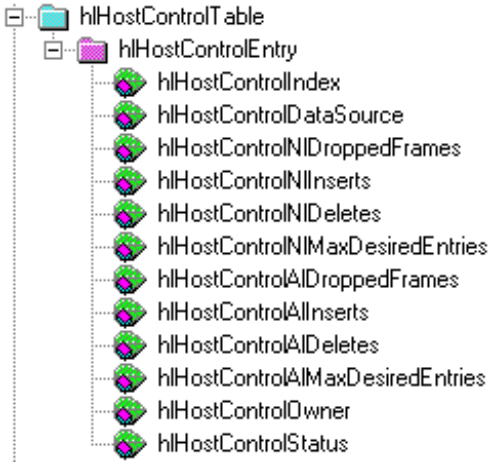


FIGURA 2.14 – Objetos da tabela *hlhostcontrol*

É válido observar que são mantidas contabilizações separadas sobre o número de inserções e remoções para as tabelas *nlHost* e *alHost*. O mesmo ocorre com o objeto *MaxDesiredEntries*, que aparece duas vezes na tabela *hlHostControl*. Isto ocorre porque não há um relacionamento um para um entre as entradas das mesmas. Cada entrada na tabela nlHost referencia um endereço de rede de forma unívoca para um dado equipamento (*host*). Por outro lado, cada entrada na tabela *alHost* referencia, de forma única, um protocolo de aplicação sendo observado. Podem existir múltiplos protocolos de aplicação ativos no mesmo *host* (diversas aplicações executando sobre o protocolo IP em uma mesma estação de trabalho, por exemplo).

A tabela *nlHost* (vide figura 2.15) criará entradas para todos os encapsulamentos de protocolo do nível de rede presentes na tabela de diretórios que possuam o objeto *protocolDirHostConfig* com valor igual a *supportedOn(3)*. Caso o valor do objeto *protocolDirHostConfig* for configurado para *supportedOff* (2) ou a entrada do encapsulamento de protocolo é deletada da tabela *protocolDir*, todas as correspondentes entradas da tabela ao qual refere-se o encapsulamento de protocolo devem ser deletadas. O *probe* adicionará entradas nesta tabela para todos os endereços de rede origem e destino observados em pacotes livres de erros. Os objetos presentes em cada entrada são:

- *NlHostTimeMark*: Contém um valor em *TimeTicks* e representa quando a entrada foi atualizada pela última vez;
- *NlHostAddress*: Endereço de rede para esta entrada;
- *NlHostInPackets*: Número de pacotes livres de erros que foram transmitidos para este endereço desde que foi adicionado à tabela;
- *NlHostOutPackets*: Número de pacotes livres de erros que foram enviados por este endereço desde que foi adicionado à tabela;
- *NlHostInOctets*: Número de octetos que foram transmitidos para este endereço desde que foi adicionado à tabela. Octetos de pacotes com erros não são contabilizados;
- *NlHostOutOctets*: Número de octetos livres de erros que foram enviados por este endereço desde que foi adicionado à tabela. Octetos de pacotes com erros não são contabilizados;
- *NlHostCreateTime*: Possui o valor do objeto s*ysUpTime* quando esta entrada foi ativada;

- *NlHostOutMacNonUnicastPkts*: Número de pacotes transmitidos por este endereço que foram direcionados para um endereço MAC de *broadcast* ou *multicast*, desde que esta entrada foi adicionada à tabela.

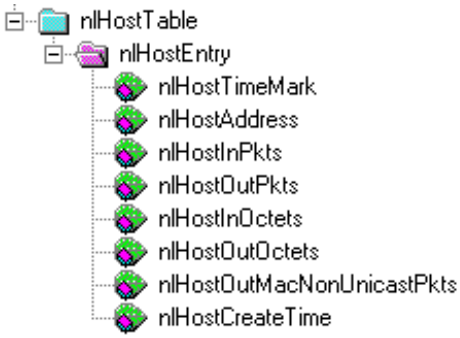


FIGURA 2.15 – Objetos da tabela *nlhost*

A função da tabela *nlHost* é coletar estatísticas básicas sobre o tráfego de entrada e saída de cada equipamento descoberto, considerando endereços do nível de rede. Logo que uma nova entrada é adicionada à tabela *hlHostControl*, o monitor começa a observar e coletar endereços de rede na interface correspondente. A cada novo endereço descoberto, uma entrada é adicionada à tabela *nlHost* e o valor do objeto *hlHostControlNlInserts* é incrementado em uma unidade.

A tabela *nlHost* é indexada por quatro objetos:

- *hlHostControlIndex*: define a interface;
- *nlHostTimeMark:* filtro de tempo;
- *protocolDirLocalIndex:* identifica o encapsulamento de protocolo do nível de rede;
- *nlHostAdress:*o endereço de rede.

Assim, dado um endereço de rede para um protocolo particular observado em uma determinada interface dentro de um intervalo de tempo, é possível obter estatísticas de tráfego para aquele endereço.

A tabela *nlHost* contém uma entrada para cada endereço de rede descoberto. Como um equipamento (roteador, por exemplo) pode ter mais de um endereço de rede associado, é possível que nesta tabela exista um número maior de entradas do que o de equipamentos efetivamente descobertos.

2.5 O grupo *Application-Layer Host*

O grupo *Application-Layer Host* é composto por uma tabela chamada *alHost*, sendo a mesma, controlada pela tabela *hlHostControl*. A tabela *alHost* contém uma entrada para cada encapsulamento de protocolo do nível de aplicação descoberto. É válido ressaltar que o termo nível de aplicação refere-se a todos os protocolos acima do nível de rede. Os objetos da tabela *alHost* são apresentados na figura 2.16.

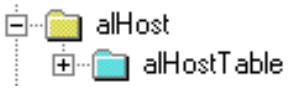


FIGURA 2.16 – Estrutura do grupo *Application-Layer Host*

A tabela *alHost* (vide figura 2.17) criará entradas para todos os encapsulamentos de protocolo do nível de aplicação presentes na tabela de diretórios que possuam o objeto *protocolDirHostConfig* com valor igual a *supportedOn(3)* e a entrada que corresponde ao encapsulamento de protocolo de rede estiver com o valor do objeto *protocolDirHostConfig* igual a *supportedOn* (3). Assim como a tabela *nlHost*, se o objeto *protocolDirHostConfig* for configurado para *supportedOff* (2) ou a entrada do protocolo é deletada da tabela *protocolDir*, todas as correspondentes entradas da tabela ao qual refere-se o encapsulamento de protocolo devem ser deletadas. O *probe* adicionará entradas nesta tabela para todos os endereços de rede origem e destino observados em pacotes livres de erros. Os objetos presentes em cada entrada são os seguintes:

- *AlHostTimeMark*: Contém um valor em *TimeTicks* e representa quando a entrada foi atualizada pela última vez;
- *AlHostInPackets*: Número de pacotes livres de erros, do protocolo em questão, que foram transmitidos para este endereço desde que foi adicionado à tabela;
- *AlHostOutPackets*: Número de pacotes livres de erros, do protocolo em questão, que foram enviados por este endereço desde que foi adicionado à tabela;
- *AlHostInOctets*: Número de octetos, do protocolo em questão, que foram transmitidos para este endereço desde que foi adicionado à tabela. Octetos de pacotes com erros não são contabilizados;
- *AlHostOutOctets*: Número de octetos livres de erros, do protocolo em questão, que foram enviados por este endereço desde que foi adicionado à tabela. Octetos de pacotes com erros não são contabilizados;
- *AlHostCreateTime*: Possui o valor do objeto s*ysUpTime* quando esta entrada foi ativada. Pode ainda ser usado pela estação de gerenciamento para assegurar que a entrada não foi deletada e recriada entre requisições.

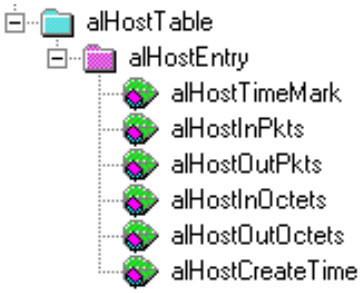


FIGURA 2.17 – Objetos da tabela *alhost*

A tabela *alHost* é indexada por cinco objetos:

- *hlHostControlIndex*: define a interface;
- *alHostTimeMark*: filtro de tempo;
- *protocolDirLocalIndex*: identidade do encapsulamento de protocolo do nível de rede;
- *nlHostAddress*: o endereço de rede;
- *protocolDirLocalIndex*: identidade do protocolo do nível de aplicação.

Observe que o objeto *protocolDirLocalIndex* é utilizado duas vezes para indexar a tabela. Tal uso se deve ao fato por ser duas instâncias distintas: uma para o encapsulamento de protocolo do nível de rede e outra para o encapsulamento de protocolo do nível de aplicação.

2.6 O grupo *Network-Layer Matrix*

O grupo *Network-Layer Matrix*, assim como o *Application-Layer Matrix* apresentado adiante, opera com a coleta de estatísticas em pares de *hosts*. Estas estatísticas são realizadas com base no endereço do nível de rede. Ele consiste de cinco tabelas: duas de controle e três de dados. Uma das tabelas de controle e duas das de dados são responsáveis por coletar e armazenar estatísticas para a matriz de *hosts*. As tabelas restantes coletam informações para as estatísticas *topN*.

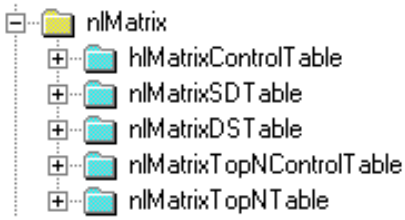


FIGURA 2.18 – Estrutura do grupo *Network-Layer Matrix*

2.6.1 Estatísticas do tipo origem/destino de protocolos do nível de rede

A tabela *hlMatrixControl* é composta pelos objetos ilustrados na figura 2.19. A função de cada um destes objetos é detalhada abaixo:

- *HlMatrixControlIndex*: Valor inteiro que identifica de forma única uma entrada na tabela *hlMatrixControl* (cada entrada nesta tabela refere-se a uma interface do *probe* – uma sub-rede);
- *HlMatrixControlDataSource*: Identifica a interface e, portanto, a sub-rede, que é fonte de informação para as entradas na tabela de dados. Esse objeto não pode ser modificado se o objeto *hlMatrixControlStatus* desta entrada estiver com o seu valor igual a *active* (1);
- *HlMatrixControlNlDroppedFrames*: Número total de pacotes recebidos pela interface que o *probe* opta por não contabilizar nas tabelas de dados do grupo *nlMatrix*;
- *HlMatrixControlNlInserts*: Número de vezes em que foram acrescentadas entradas nas tabelas de dados do grupo *nlMatrix*. Caso uma entrada seja inserida, depois deletada e novamente inserida, o valor do campo é incrementado 2 vezes;
- *HlMatrixControlNlDeletes*: Número de vezes em que foram removidas entradas das tabelas de dados do grupo *nlMatrix*. Caso uma entrada seja deletada, depois inserida e novamente deletada, o valor do campo é incrementado 2 vezes;
- *HlMatrixControlNlMaxDesiredEntries*: Número máximo admissível de entradas nas tabelas de dados do grupo *nlMatrix*. Se o valor do objeto for igual a –1 então o probe pode criar qualquer número de entradas. E se o associado objeto hlMatrixControlStatus tiver com o valor igual a active (1) o objeto não pode ser modificado;
- *HlMatrixControlAlDroppedFrames*: Número total de pacotes recebidos pela interface que o *probe* opta por não contabilizar nas tabelas de dados do grupo *alMatrix*;
- *HlMatrixControlAlInserts*: Número de vezes em que foram acrescentadas entradas nas tabelas de dados do grupo *alMatrix*. Caso uma entrada seja

inserida, depois deletada e novamente inserida, o valor do campo é incrementado 2 vezes;

- *HlMatrixControlAlDeletes*: Número de vezes em que foram removidas entradas das tabelas de dados do grupo *alMatrix*. Caso uma entrada seja deletada, depois inserida e novamente deletada, o valor do campo é incrementado 2 vezes;
- *HlMatrixControlAlMaxDesiredEntries*: Número máximo admissível de entradas para cada tabela das tabelas de dados do grupo *alMatrix*. Se o valor do objeto for igual a –1 então o probe pode criar qualquer número de entradas. Outro ponto é com respeito ao associado objeto *hlMatrixControlStatus* da entrada. Se o mesmo tiver com o valor igual a *active* (1) esse objeto não pode ser modificado;
- *HlMatrixControlOwner*: entidade que configurou essa entrada. O dado desse objeto pode ser modificado;
- *HlMatrixControlStatus*: o estado dessa entrada. Uma entrada não pode existir no estado *active* (1) ao menos que todos os objetos da entrada contenham um valor coerente. Caso esse objeto tenha o valor diferente de *active*, então todas as entradas associadas a essa entrada nas tabelas *nlMatrixSD*, *nlMatrixDS*, *alMatrixSD* e *alMatrixDS* devem ser excluídas.

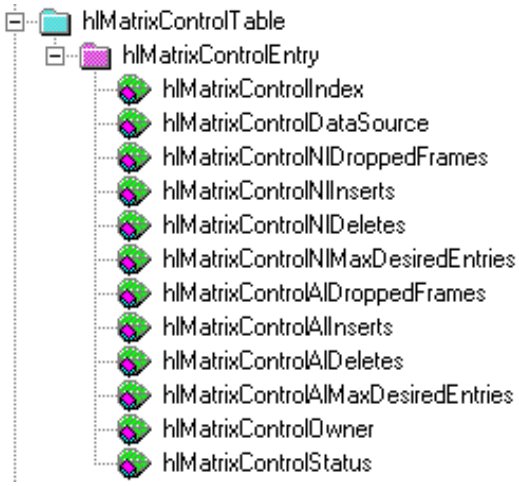


FIGURA 2.19 – Objetos da tabela *hlmatrixcontrol*

A tabela *nlMatrixSD*, a qual pode ser vista na figura 2.20, é utilizada para armazenar estatísticas sobre o tráfego de um endereço do nível de rede para um número de destinos. Esta tabela criará entradas para todos os protocolos do nível de rede presentes na tabela *protocol directory*, cujo valor do objeto *protocolDirMatrixConfig* seja igual a *SupportedOn*(3). Esta tabela possui os seguintes campos:

- *NlMatrixSDTimeMark*: Contém um valor em *TimeTicks* e representa quando a entrada foi atualizada pela última vez;
- *NlMatrixSDSourceAddress*: Endereço de rede origem para esta entrada;
- *NlMatrixSDDestAddress*: Endereço de rede destino para esta entrada;
- *NlMatrixSDPkts*: Número de pacotes livres de erros transmitidos desta origem para este destino;
- *NlMatrixSDOctests*: Número de octetos livres de erros transmitidos desta origem para este destino (pacotes com erros não são contabilizados);
- *NlMatrixSDCreateTime*: Valor do objeto s*ysUpTime* quando esta entrada foi adicionada a esta tabela.

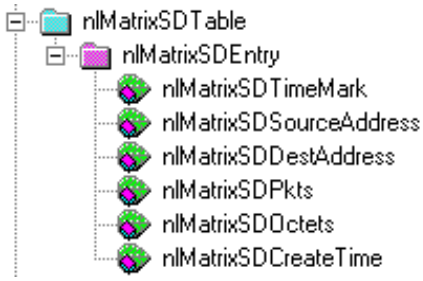


FIGURA 2.20 – Objetos da tabela *nlMatrixSD*

A tabela *nlMatrixSD* é indexada por cinco objetos:

- Uma entrada da tabela *nlMatrixControl*;
- Um filtro de tempo (*nlMatrixSDTimeMark*);
- Protocolo do nível de rede (*protocolDirLocalIndex*);
- Endereço do nível de rede do *host* origem (*nlMatrixSDSourceAddress*);
- Endereço do nível de rede do *host* destino (*nlMatrixSDDestAddress*).

Um exemplo de um objeto indexado pode ser:

- NlMatrixSDPkts.1.787867.1.4.192.168.0.1.4.192.168.0.2.

A tabela *nlMatrixDS* (figura 2.21) contém os mesmos objetos que a tabela *nlMatrixSD*, mas é indexada primeiro pelo endereço destino e, depois, pelo origem. A interpretação para estas tabelas é a seguinte: cada entrada na tabela *hlMatrixControl* identifica uma interface distinta do *probe* (cada uma delas estará provavelmente ligada a uma sub-rede). A tabela *nlMatrixSD* contém duas entradas para cada par de *hosts* naquela sub-rede que trocaram informações recentemente. Uma das entradas agrega informações sobre o tráfego em um sentido e a outra reporta sobre o tráfego no sentido inverso. Estas mesmas entradas estão presentes na tabela *nlMatrixDS*. Desta forma, a estação de gerenciamento pode facilmente recuperar informações sobre o tráfego de uma estação para todas as outras (através da tabela *nlMatrixSD*) ou sobre o tráfego total de todas as estações para uma estação particular (através da tabela *nlMatrixDS*).

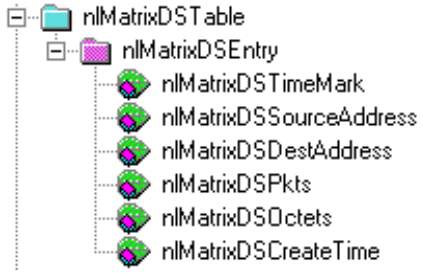


FIGURA 2.21 – Objetos da tabela *nlMatrixDS*

Tão logo o *probe* detecte uma troca que envolva uma comunicação entre um novo par de estações, ele cria uma nova entrada em ambas as tabelas de dados. Se o limite especificado para o objeto *hlMatrixControlNlMaxDesired* for alcançado, então o *probe* remove a primeira entrada da tabela *nlmatrixSD* e da *nlmatrixDS*.

2.6.2 Estatísticas do tipo *topN* de protocolos do nível de rede

O grupo *Network-Layer Matrix* contém uma tabela de controle e uma de dados. As estatísticas *topN* presentes na MIB RMON2 diferem das existentes na RMON tradicional. Na RMON1, o grupo *hostTopN* agrega estatísticas que permitem ordenar os *hosts* em uma sub-rede de acordo com algum parâmetro. Na MIB RMON2, por outro lado, esta ordenação é realizada com base no tráfego entre pares de *hosts*. A tabela *nlMatrixTopNControl* inclui os objetos ilustrados na figura 2.22.

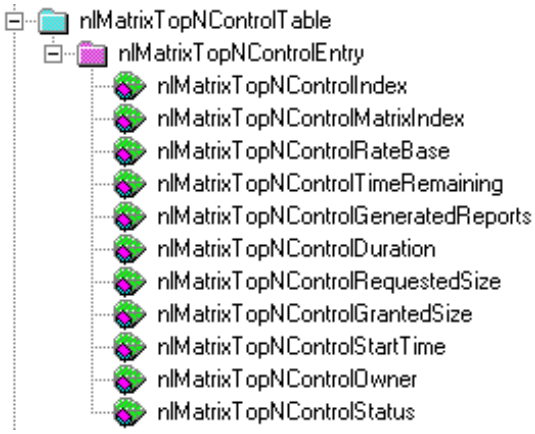


FIGURA 2.22 – Objetos da tabela *nlMatrixTopNControl*

A função de cada um dos objetos é:

- *NlMatrixTopNControlIndex*: Valor inteiro que identifica, de forma única, uma entrada na tabela *nlMatrixTopNControl*. Cada entrada nesta tabela define um conjunto de estatísticas do tipo *topN* para uma determinada interface do *probe* (cada sub-rede ligada a ele);
- *NlMatrixTopNControlMatrixIndex*: Especifica uma sub-rede particular; este objeto é importante porque as estatísticas *topN* são geradas utilizando entradas das tabelas *nlMatrixSD* ou *nlMatrixDS*, que são indexadas pela informação da sub-rede a qual pertencem. Esse objeto não pode ser modificado se o associado objeto *nlMatrixTopNControlStatus* estiver com o valor igual a *active* (1);
- *NlMatrixTopNControlRateBase*: Especifica o critério de classificação para a criação das estatísticas. O objeto pode assumir o valor *nlMatrixTopPkts* (1) que contabiliza e ordena as entradas da tabela pelo número de pacotes ou *nlMatrixTopNOctets* (2) que contabiliza e ordena as entradas da tabela pelo número de octetos. Esse objeto não pode ser modificado se o associado objeto *nlMatrixTopNControlStatus* estiver com o valor igual a *active* (1);
- *NlMatrixTopNControlTimeRemaining*: Quantidade de segundos restantes no intervalo de amostragem para a atualização da tabela de dados *nlMatrixTopN,* realizada com base nas informações coletadas neste intervalo de tempo. Caso o valor seja alterado pelo gerente, as entradas associadas a essa entrada na tabela *nlMatrixTopN* devem ser deletadas;
- *NlMatrixTopNControlGeneratedReports*: Número de relatórios que foram gerados por essa entrada;
- *NlMatrixTopNControlDuration*: Intervalo de amostragem, em segundos, para a coleta de informações. O dado do objeto é repassado para o objeto *NlMatrixTopNControlTimeRemaining* assim que termina um relatório e inicia a próxima atualização das estatísticas *topN* na tabela de dados *nlMatrixTopN*;

- *NlMatrixTopNControlRequestedSize*: Número máximo de entradas da tabela *matrix* requisitadas para esse relatório;
- *NlMatrixTopNControlGrantedSize*: Número máximo de entradas para a tabela *topN*;
- *NlMatrixTopNControlStartTime*: Valor do objeto *sysUpTime* quando o relatório descrito nesta entrada foi iniciado pela última vez;
- *NlMatrixTopNControlOwner*: entidade que configurou essa entrada. O dado desse objeto pode ser modificado;
- *NlMatrixTopNControlStatus*: o estado dessa entrada. Uma entrada não pode existir no estado *active* (1) ao menos que todos os objetos da entrada contenham um valor coerente. Caso esse objeto tenha o valor diferente de *active*, então todas as entradas associadas a essa entrada nas tabela *nlMatrixTopN* devem ser excluídas.

A tabela de dados *nlMatrixTopN* é composta pelos objetos apresentados abaixo e os mesmos podem ser vistos na figura 2.23:

- *NlMatrixTopNIndex*: Valor inteiro que identifica univocamente uma entrada entre todas as outras de um determinado relatório. Cada entrada representa um par de *hosts* origem-destino, que não se repete no mesmo relatório. A forma de atribuição do valor às entradas se dá em ordem crescente de índice e decrescente quanto ao valor *PktRate* ou *OctetRate*. Por exemplo, a entrada com o maior valor (*PktRate* ou *OctetRate*) recebe o valor 1, o segundo de maior valor recebe o valor 2 e assim por diante até o índice N (número máximo de entradas para o relatório) ser atribuído ou não ter mais entradas na tabela;
- *NlMatrixTopNProtocolDirLocalIndex*: Identifica, de forma única, um encapsulamento de protocolo do nível de rede;
- *NlMatrixTopNInSourceAddress*: Endereço do nível de rede do *host* origem desta entrada;
- *NlMatrixTopNInDestAddress*: Endereço do nível de rede do *host* destino desta entrada;
- *NlMatrixTopNPktRate*: Número de pacotes observados do *host* origem para o destino durante o intervalo de amostragem;
- *NlMatrixTopNReversePktRate*: Número de pacotes observados do *host* destino para o origem durante o intervalo de amostragem. Se o valor do objeto *nlMatrixTopNControlRateBase* for igual a *nlmatrixTopNPkts*, a ordenação das entradas é baseada inteiramente no objeto *nlmatrixTopNPktRate* e não no valor desse objeto;
- *NlMatrixTopNOctetRate*: Número de octetos observados do *host* origem para o destino durante o intervalo de amostragem;
- *NlMatrixTopNReverseOctetRate*: Número de octetos observados do *host* destino para o origem durante o intervalo de amostragem. Se o valor do objeto *nlMatrixTopNControlRateBase* for igual a *nlmatrixTopNOctets*, a ordenação das entradas é baseada inteiramente no objeto *nlmatrixTopNOctetRate* e não no valor desse objeto.

FIGURA 2.23 – Objetos da tabela *nlMatrixTopN*

A tabela *nlMatrixTopN* é indexada inicialmente pelo objeto *nlMatrixTopNControlIndex* e então por *nlMatrixTopNIndex*. O primeiro deles define um conjunto de entradas que constituem um relatório. Este relatório terá seus dados coletados da sub-rede a qual a interface (*nlMatrixTopNMatrixIndex*) em questão do *probe* está associada e será construído de acordo com um dos critérios definidos pelo objeto *nlMatrixTopNControlRateBase*. O segundo índice é utilizado para referenciar os pares origem-destino classificados na tabela *nlMatrixTopN* de acordo com o volume de tráfego observado entre eles.

O processo de preparação do relatório inicia quando uma estação de gerenciamento cria uma entrada na tabela de controle (*nlMatrixTopNControl*) para solicitar a criação de um novo relatório. Esta entrada instrui o *probe* a medir a diferença entre os valores iniciais e finais de algumas variáveis do grupo *nlMatrix* (tabelas *nlMatrixSD* e *nlMatrixDS*) durante um período determinado de tempo. Este período é armazenado em dois objetos: *nlMatrixTopNControlDuration* e *nlMatrixTopNControlTimeRemaining*. O primeiro deles possui um valor estático; o segundo, por outro lado, apresenta um valor que é decrementado a cada segundo até atingir o valor 0. Neste instante, o *probe* calcula os resultados finais e cria um conjunto de N entradas na tabela de dados, indexadas por *nlMatrixTopNIndex*, com os N pares de estações que mais se comunicaram neste período de tempo em ordem decrescente de volume de tráfego.

Ao contrário do que ocorre na MIB RMON1, na RMON2 o processo de geração de novas tabela *topN* é disparado novamente tão logo um ciclo se complete. Desta forma, o relatório é atualizado automaticamente a cada *x* segundos, onde *x* é o valor associado ao objeto *nlMatrixTopNControlDuration*. Uma vez criado o relatório, os dados ficam disponíveis somente para leitura para a estação central de gerenciamento, até que sejam sobrescritos por outro relatório mais atual; o gerente pode determinar quando o próximo relatório estará disponível consultando o valor do objeto *nlMatrixTopNControlTimeRemaining*.

2.7 O grupo *Application-Layer Matrix*

O grupo *Application-Layer Matrix* é formado por três tabelas de dados e uma de controle (vide figura 2.24). Duas das tabelas de dados manipulam estatísticas associadas às matrizes, que armazenam informações sobre o volume de tráfego entre pares de estações. A terceira tabela de dados é responsável pelas estatísticas do tipo *topN*.

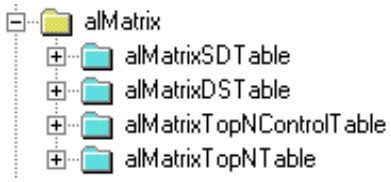


FIGURA 2.24 – Estrutura do grupo *Application-Layer Matrix*

2.7.1 Estatísticas do tipo origem/destino de protocolos do nível de aplicação

As duas tabelas de dados que manipulam estatísticas sobre a comunicação entre pares de estações através de protocolos do nível de aplicação são controladas pela tabela *hlMatrixControl*. A tabela *alMatrixSD* é utilizada para armazenar estatísticas sobre o tráfego de saída de uma estação utilizando um determinado encapsulamento de protocolo do nível de aplicação. Ela criará entradas para todos os encapsulamentos de protocolo do nível de aplicação que estejam presentes na tabela *protocolDir* cujo valor do objeto *protocolDirMatrixConfig* seja igual a *SupportedOn*(3) e seu encapsulamento de protocolo de rede tenha também o valor *SupportedOn* (3) para o objeto *protocolDirMatrixConfig*. Os objetos da tabela *alMatrixControl* (vide figura 2.25) são detalhados abaixo.

- *AlMatrixSDTimeMark*: Contém um valor em *TimeTicks* e representa quando a entrada foi atualizada pela última vez. Seu valor não pode ser recuperado, sendo apenas usado para indexação;
- *AlMatrixSDPackets*: Número de pacotes, sem erros, do protocolo do nível de aplicação transmitidos do endereço origem para o destino desde que esta entrada foi adicionada à tabela;
- *AlMatrixSDOctets*: Número de octetos, sem erros, do protocolo do nível de aplicação transmitidos do endereço origem para o destino desde que esta entrada foi adicionada à tabela;
- *AlMatrixSDCreateTime*: Valor do objeto s*ysUpTime* quando esta entrada foi adicionada a esta tabela.

FIGURA 2.25 – Objetos da tabela *alMatrixSD*

A forma de indexação desta tabela é um bom exemplo para ilustrar que na MIB RMON2 é possível indexar algumas tabelas utilizando objetos que não sejam necessariamente parte das mesmas. Os índices da tabela *alMatrixSD* são, nesta ordem:

- *hlMatrixControlIndex*: identifica, de forma única, uma sub-rede e uma entrada na tabela *hlMatrixControl*; com este primeiro índice é possível selecionar todas as entradas da tabela *alMatrixSD* observadas na sub-rede em questão;
- *alMatrixSDTimeMark*: filtro de tempo. Diminui o conjunto de entradas selecionadas, ao utilizar às que possuírem valor igual ou maior ao utilizado para fazer a seleção;

- *protocolDirLocalIndex*: identifica de forma unívoca um protocolo do nível de rede;
- *nlMatrixSDSourceAddress*: identifica o *host* que originou os pacotes;
- *nlMatrixSDDestAddress*: identifica o *host* que recebeu os pacotes;
- *protocolDirLocalIndex*: identifica, de forma única, o protocolo de aplicação que opera sobre o protocolo do nível de rede selecionado acima. Assim, para o par de *hosts* origem-destino selecionados nos cinco índices anteriores, este índice seleciona uma entrada única na tabela *alMatrixSD*, que agrega estatísticas sobre o protocolo de aplicação que esteja sendo utilizado entre este par de *hosts*.

Como foi observado, a indexação desta tabela envolve a utilização de seis objetos de três tabelas distintas. O documento que define a MIB RMON2 inclui o seguinte exemplo para a seleção de uma única instância do objeto *alMatrixSDPkts* em uma única entrada da tabela *alMatrixSD*:

- alMatrixSDPkts.1.783495.18.4.128.2.6.6.4.128.2.6.7.34

O sub-identificador [1] especifica a primeira entrada da tabela *nlMatrixControl*. O sub-identificador [783495] especifica o tempo, em *time ticks*, para esta entrada, indicando que ele só irá selecionar objetos que foram mais recentes a este instante de tempo. O sub-identificador [18] especifica o protocolo do nível de rede definida pela entrada 18 da tabela *protocolDir*. A seqüência [4.128.2.6.6] especifica o endereço do nível de rede origem; neste caso, é um endereço IP consistindo de quatro octetos, cujo valor é 128.2.6.6. Da mesma forma, a seqüência [4.128.2.6.7] especifica o endereço IP destino. Por fim, o sub-identificador [34] identifica o protocolo do nível de aplicação definido na entrada 34 da tabela *protocolDir*.

A tabela *alMatrixDS* contém a mesma informação que a tabela *alMatrixSD,* mas é indexada primeiramente pelo endereço destino e então pelo origem. Veja os objetos da tabela na figura 2.26.

FIGURA 2.26 – Objetos da tabela *almatrixDS*

2.7.2 Estatísticas do tipo *topN* de protocolos do nível de aplicação

As tabelas de controle e dados que armazenam estatísticas do tipo *topN* para os protocolos do nível de aplicação possuem a mesma estrutura e funcionalidade das coletadas para protocolos do nível de rede.

Com relação à tabela de controle (vide figura 2.27) existe uma diferença no objeto *alMatrixTopNControlRateBase* que pode assumir quatro valores:

- *alMatrixTopNTerminalPkts*(1) que representa uma contabilização e ordenação das entradas através do número de pacotes dos protocolos terminais (protocolos de aplicação);
- *alMatrixTopNTerminalOctets*(2) que representa uma contabilização e ordenação das entradas através do número de octetos dos protocolos terminais;

- *alMatrixTopNAllPkts*(3) que representa uma contabilização e ordenação das entradas através do número de pacotes de todos os protocolos de alto nível;
- *alMatrixTopNAllOctets*(4) que representa uma contabilização e ordenação das entradas através do número de octetos de todos os protocolos de alto nível.

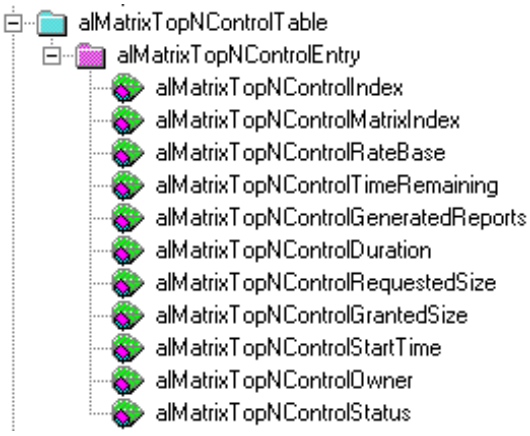


FIGURA 2.27 – Objetos da tabela *alMatrixTopNControl*

A tabela de dados, *alMatrixTopN* (vide figura 2.28), realiza as mesmas funções da tabela *nlMatrixTopN*, porém com encapsulamentos de protocolo do nível de aplicação. Os objetos das tabelas são os mesmos. A diferença está na inclusão de um novo índice, *AppProtocolDirLocalIndex*, que especifica o encapsulamento de protocolo de aplicação a ser contabilizado.

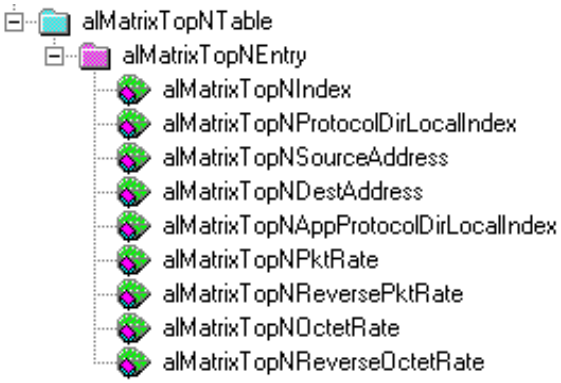


FIGURA 2.28 – Objetos da tabela *alMatrixTopN*

2.8 O grupo *User History Collection*

O grupo *User History Collection* consulta periodicamente o valor de objetos e armazena os dados obtidos em um *log*, considerando alguns parâmetros definidos pelo gerente. Com esta característica, ele pode configurar estudos baseado na história de qualquer objeto presente nas MIBs que o *probe* suporta. No padrão RMON1, esta funcionalidade é oferecida para um conjunto pré-definido de objetos.

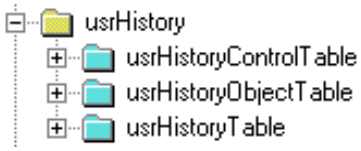


FIGURA 2.29 – Estrutura do grupo *User History Collection*

A estrutura deste grupo é a mais complexa de todos os grupos do RMON2. Ele é composto por tabelas organizadas em três níveis de hierarquia (vide figura 2.29). A primeira tabela nesta hierarquia é *usrHistoryControl*. Nela, são definidos os detalhes das funções de amostragem. Subordinada a ela, existe uma ou mais instâncias da tabela *usrHistoryObject*. Cada uma destas instâncias determina os objetos que serão amostrados. Por fim, para cada instância da tabela *usrHistoryObject* existe uma ou mais instâncias da tabela *usrHistory*, que armazenam os dados coletados.

A funcionalidade dos objetos da tabela *usrHistoryControl* (vide figura 2.30) podem ser assim descritos:

- *UsrHistoryControlIndex*: Valor inteiro que identifica, de forma única, uma entrada na tabela *usrHistoryControl*. Cada entrada configura um conjunto de amostragens, em um intervalo particular, de um ou mais objetos específicos de alguma MIB;
- *UsrHistoryControlObjects*: Número de objetos da MIB para os quais serão coletados dados. Esse objeto não pode ser modificado se o associado objeto *usrHistoryControlStatus* estiver com o valor *active* (1);
- *UsrHistoryControlBucketsRequested*: Número que indica o número solicitado de intervalos de amostragem nos quais os dados serão coletados e armazenados na porção da tabela *usrHistory* associada a esta entrada;
- *UsrHistoryControlBucketsGranted*: Número que indica o número real de intervalos de amostragem nos quais os dados serão coletados e armazenados na porção da tabela *usrHistory* associada a esta entrada. Quando o valor de *usrHistoryControlBucketsRequested* é modificado, o *probe* deve atualizar este objeto para o valor mais próximo do solicitado quanto possível;
- *UsrHistoryControlInterval*: Intervalo, em segundos, no qual os dados serão amostrados para cada objeto definido na parte da tabela *UsrHistory* associada a esta entrada. O intervalo pode ser configurado para um valor entre 1 e $2^{31} - 1$. Esse objeto não pode ser modificado se o associado objeto *usrHistoryControlStatus* estiver com o valor *active* (1);
- *UsrHistoryControlOwner*: entidade que configurou essa entrada. Esse objeto pode ser alterado;
- *UsrHistoryControlStatus*: o estado dessa entrada. Uma entrada não pode existir no estado *active* (1) ao menos que todos os objetos da entrada tenham um valor coerente. Caso esse objeto tenha o valor diferente de *active*, então todas as entradas associadas a essa entrada na tabela *usrHistory* devem ser excluídas.

FIGURA 2.30 – Objetos da tabela *usrHistoryControl*

O esquema de amostragem é ditado pelos objetos *usrHistoryControlBucketsGranted* e *usrHistoryControlInterval*. Por exemplo, utilizando os valores padrões, o *probe* deveria amostrar cada um dos objetos (número de objetos = *usrHisotryControlObjects*) uma vez a cada 30 minutos; cada amostra de cada objeto é armazenada em uma entrada da tabela *usrHistory*, onde as 50 entradas mais recentes são resguardadas.

A outra tabela de controle, conforme já mencionado, é a *usrHistoryObject* (vide figura 2.31). Os objetos associados a ela são:

- *UsrHistoryObjectIndex*: Valor inteiro que identifica univocamente uma entrada nesta tabela. Cada entrada define um objeto da MIB a ser amostrado periodicamente;
- *UsrHistoryObjectVariable*: O valor deste objeto é um identificador de objeto que especifica uma instância particular de objeto da MIB a ser amostrado. Só podem ser amostrados objetos cujos valores sejam do tipo *Integer32*(*Integer32*, *Counter*, *Gauge* e *TimeTicks*). Esse objeto não pode ser modificado se o associado objeto *usrHistoryControlStatus* estiver com o valor *active* (1);
- *UsrHistoryObjectSampleType*: Define se o valor consultado será armazenado na forma absoluta ou como um delta em relação à amostra anterior. Se o valor deste objeto for *absoluteValue*(1), então o valor do objeto selecionado será copiado diretamente para a tabela *usrHistory*. Se o valor for *deltaValue*(2), então o valor do objeto selecionado na última amostragem é subtraído do valor atual e a diferença obtida é armazenada como uma entrada na tabela *usrHistory*. Esse objeto não pode ser modificado se o associado objeto *usrHistoryControlStatus* estiver com o valor *active* (1).

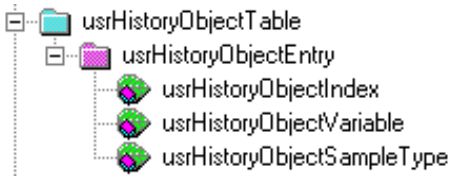


FIGURA 2.31 – Objetos da tabela *usrHistoryObject*

Por fim, os objetos da tabela de dados *usrHistory* (vide figura 2.32) são apresentados abaixo:

- *UsrHistorySampleIndex*: Índice que identifica, de forma única, a amostra particular que esta entrada representa entre todas as outras associadas à mesma entrada da tabela *usrHistoryControl*. O primeiro valor para este índice é 1.
- *UsrHistoryIntervalStart*: Valor do objeto *sysUpTime* no início do intervalo dentro do qual esta amostra foi obtida;

- *UsrHistoryIntervalEnd*: Valor do objeto *sysUpTime* no final do intervalo dentro do qual esta amostra foi obtida;
- *UsrHistoryAbsValue*: Valor absoluto do objeto. Este valor pode variar de 0 a 2^{32}-1;
- *UsrHistoryValStatus*: Contém um dos três valores: *valueNotAvailable*(1), *valuePositive*(2) ou *valueNegative*(3). Caracteriza o valor obtido anteriormente como não disponível, positivo ou negativo.

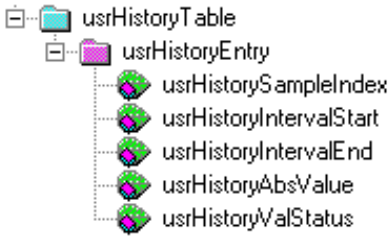


FIGURA 2.32 – Objetos da tabela *usrHistory*

2.9 O grupo *Probe Configuration*

Este grupo foi projetado para melhorar a interoperabilidade entre *probes* RMON e ferramentas de gerenciamento de diferentes fabricantes. Ele define um conjunto padrão de parâmetros para a configuração dos *probes*. Torna-se mais simples, desta forma, que, por exemplo, uma aplicação de gerenciamento de um fabricante seja capaz de configurar remotamente o *probe* de um outro.

O grupo consiste de um conjunto de objetos escalares e quatro tabelas (vide figura 2.33). Os objetos escalares são apresentados abaixo:

- *ProbeCapabilities*: Indica os grupos RMON suportados;
- *ProbeSoftwareRev*: Revisão de software para este dispositivo;
- *ProbeHardwareRev*: Revisão de hardware para este dispositivo;
- *ProbeDateTime*: Data e hora atuais do *probe*;
- *ProbeResetControl*: Pode conter os valores *running*(1), *warmBoot*(2) ou *coldBoot*(3);
- *ProbeDownloadFile*: Nome do arquivo que contém a imagem do *boot* a ser carregada do servidor TFTP;
- *ProbeDownloadTFTPServer*: endereço IP do servidor TFTP que contém a imagem do *boot* a ser carregada;
- *ProbeDownloadAction*: Pode conter os valores *imageValid*(1), *downloadtoPROM*(2) ou *downloadtoRAM*(3). Quando o dispositivo é atualizado para 2 ou 3, o dispositivo interrompe a operação normal, carrega a imagem de *boot* e realiza um *warm boot* para iniciar a nova aplicação carregada;
- ProbeOutDownloadStatus: Pode conter um dos seguintes valores:
    - downloadSucess(1);
    - downloadStatusUnknown(2);
    - downloadGeneralError(3);
    - downloadNoResponseFromServer(4);
    - downloadCheckSumError(5);
    - downloadIncompatibleImage(6);
    - tftpFileNotFound(7);

- tftpAcessViolation(8).

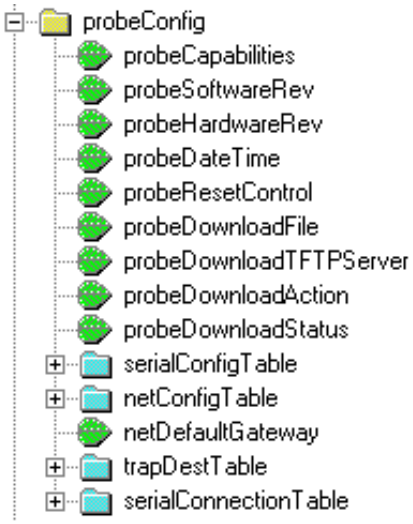


FIGURA 2.33 – Estrutura do grupo *Probe Configuration*

2.9.1 Seqüências de Controle

São utilizadas em diversas tabelas do grupo *Probe Configuration*. Uma seqüência de controle é utilizada para a comunicação com um modem ou outro dispositivo serial. Na verdade, as seqüências representam uma série de comandos que irão controlar como o dispositivo irá interagir com o dispositivo remoto através da interface serial. Os comandos são representados por seqüências de dois caracteres, sendo o primeiro, o caracter "^". A seguir, são apresentados alguns comandos pré-definidos:

^s Envia a seqüência que segue, até o próximo comando ou seu fim;
^c Espere pelo número de segundos que segue;
^t Configure o *timeout* com o valor que segue;
^w Aguarde pela seqüência de retorno, até o próximo comando ou seu final;
^! Envie o caractere "^";
^d Aguarde pelo número de segundos representado pelos dígitos decimais que seguem;
^b Envie o comando *break* durante o número de milissegundos especificado pelos dígitos decimais que seguem.

2.9.2 A tabela *Serial Configuration*

Esta tabela contém uma entrada para cada interface do *probe*. Seus objetos são assim detalhados:

- *SerialMode*: Conexão associada pode ser direta(1) ou via modem(2);
- *SerialProtocol*: Protocolo de enlace utilizado pode ser *slip*(2) ou outro(1);
- *SerialTimeout*: Número de segundos de inatividade permitido antes de encerrar a conexão nesta interface;
- *SerialModemInitString*: Seqüência de controle que especifica inicialização para o modem associado a esta interface;

- *SerialModemHangUpString*: Seqüência de controle que especifica como desconectar uma conexão via modem nesta interface;
- *SerialModemConnectResp*: Seqüência de caracteres ASCII que descrevem o código de resposta do modem e a taxa de transferência associada;
- *SerialModemNoConnectResp*: Seqüência de caracteres ASCII que descrevem códigos de resposta gerados por um modem para reportar sobre falhas na tentativa de conexão;
- *SerialDialoutTimeout*: Valor do *timeout* para a discagem.

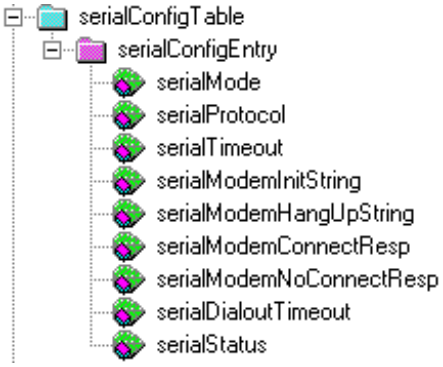


FIGURA 2.34 – Objetos da tabela *serialConfig*

2.9.3 A tabela *Network Configuration*

Possui uma entrada para cada interface presente no *probe*. As entradas possuem os objetos ilustrados abaixo:

- *NetConfigIpAddress*: Endereço IP desta interface;
- *NetConfigSubnetMask*: Máscara IP da sub-rede para esta interface.

FIGURA 2.35 – Objetos da tabela *netConfig*

Além disso, a tabela apresenta um objeto escalar denominado *netDefaultGateway*, que indica o endereço IP do *default gateway* ou roteador.

2.9.4 A tabela *Trap Destination*

Esta tabela define os endereços destino para as *traps* geradas pelo *probe*. A tabela mapeia uma comunidade para uma ou mais entradas, que definem a destinação das *traps*. A mesma *trap* será enviada para todas as destinações especificadas nas entradas cujo objeto *trapDestCommunity* tenha o mesmo valor do objeto *eventCommunity* do grupo *event do padrão* RMON1. A tabela inclui os campos abaixo:

- *TrapDestIndex*: Índice único para esta entrada;
- *TrapDestCommunity*: Comunidade a qual este endereço IP de destino pertence;

- *TrapDestProtocol*: Protocolo a ser utilizado para enviar a *trap*;
- *TrapDestAddress*: Endereço IP para o qual a *trap* será enviada.

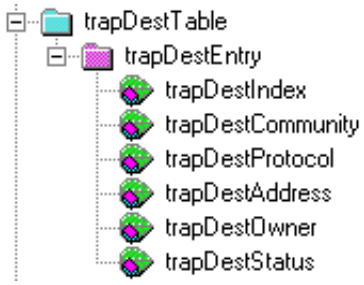


FIGURA 2.36 – Objetos da tabela *TrapDest*

Esta tabela é utilizada conjuntamente com a tabela *event* do grupo *Event* do padrão RMON1. Em essência, a tabela *event* define *traps* que serão enviadas para comunidades específicas e a tabela *trapDest* especifica os endereços IP para enviar *traps* para os membros daquela comunidade.

2.9.5 A tabela *Serial Connection*

Esta tabela armazena parâmetros necessários para iniciar uma conexão SLIP com uma estação de gerenciamento. Seus objetos podem ser assim apresentados:

- *SerialConnectIndex*: Índice único para esta entrada;
- *SerialConnectDestIpAddress*: Endereço IP que pode ser alcançado do outro lado desta conexão SLIP;
- *SerialConnectType*: Pode conter um dos seguintes valores: *direct*(1), *modem*(2), *switch*(3) e *modemSwitch*(4);
- *SerialConnectDialString*: Seqüência de controle que especifica o número telefônico para estabelecer uma conexão discada;
- *SerialConnectSwitchConnectSeq*: Seqüência de controle que especifica como estabelecer uma conexão para troca de dados;
- *SerialConnectDisconnectSeq*: Seqüência de controle que especifica como encerrar uma conexão de troca de dados;
- *SerialConnectSwitchResetSeq*: Seqüência de controle que especifica como reiniciar uma conexão de troca de dados após um *timeout*.

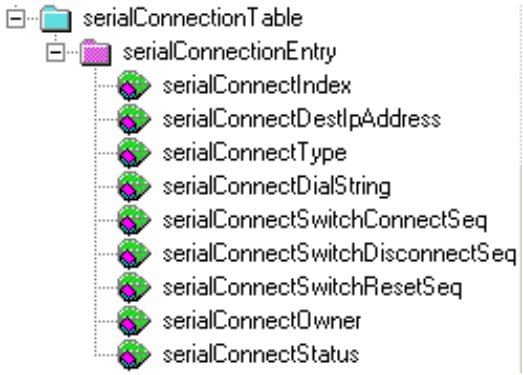


FIGURA 2.37 – Objetos da tabela *serialConnect*

3 O que se pode obter com a MIB RMON2

Como a MIB RMON2 monitora protocolos de alto nível, isto é, pode monitorar protocolos da camada de rede, transporte e aplicação, uma visão mais detalhada do comportamento da rede pode ser obtida. As informações referentes ao comportamento da rede podem ser obtidas diretamente das variáveis da MIB ou então indiretamente, através da derivação das mesmas.

Tais informações podem ser classificadas, segundo certos perfis de gerenciamento de rede, em quatro categorias segundo Gaspary em [GAS99b]. Para tanto, essas categorias tem o seu foco direcionado para:

- Administração e estatísticas das atividades do usuário;
- Perfil de uso global da rede;
- Otimização de usuários e distribuição de recursos;
- Gerenciamento de segurança.

Nas subseções a seguir, serão descritos as categorias e quais grupos, tabelas e objetos da MIB RMON2 devem ser consultados para alcançar as estatísticas referentes à categoria. Cabe ressaltar que as tabelas e figuras expostas no capítulo são meramente ilustrativas, isto é, pode não haver correlação entre ambas.

3.1 Administração e estatísticas das atividades do usuário

Quando um administrador está interessado em traçar o perfil de uso da rede, é muito importante para ele obter informações sobre como um certo usuário ou um grupo a utiliza. A partir dessas informações, o gerente pode responder a questões como:

- Quais os usuários que mais utilizam a rede?
- Que períodos um certo usuário costuma utilizar a rede?
- Com quem esses usuários se comunicam e em quais períodos?
- Quais as aplicações e protocolos que eles executam?

Com a resolução das perguntas o administrador tem a possibilidade de observar as atividades do usuário e verificar se elas atendem os interesses da organização.

3.1.1 Volume de acessos

Informações que relatam o volume de acessos à rede de um certo usuário podem ser obtidas através de requisições ao grupo *network-layer host* da RMON2. Esse grupo decodifica os pacotes baseando-se em seu endereço de rede. Como está baseado no endereço de rede, pode-se observar além dos equipamentos de interconexão das sub-redes e, portanto consegue-se identificar os *hosts* reais que estão se comunicando [PER99].

A tabela *nlHost* provê estatísticas de rede básicas para cada endereço da camada de rede visto em um segmento de rede monitorado (vide tabela 3.1). Quando uma nova entrada é adicionada à tabela *hlHostControl*, o *probe* começa a observar pacotes e a coletar os endereços da camada de rede na respectiva interface. Cada novo endereço de

rede descoberto no segmento de rede resulta a criação de uma nova entrada na tabela *nlHost*.

TABELA 3.1 – Exemplo de informação recuperada da tabela *nlHost*

HostAddress	*InPkts*	*OutPkts*	*InOctets*	*OutOctets*	*OutMacNonUnicastPkts*
10.67.4.125	35939	41989	14433978	18650480	0
10.25.146.137	29920	21640	16845056	2728123	0
200.20.120.11	1242	1174	150069	1395316	0
200.226.126.17	550	426	156286	393357	0

Requisições a esse grupo podem ajudar os administradores a descobrir quais usuários mais usam a rede e em que períodos. Nesse contexto, o grupo *user history* pode ser usado para armazenar valores de objetos em diferentes intervalos de tempo ou o gerente pode fazer requisições ao agente de tempos em tempos.

Uma simples fórmula para calcular a taxa de uso da rede (t.u.r), em porcentagem, por um certo usuário durante um intervalo de tempo entre t_1 e t_2 é mostrado abaixo. A variável *ifSpeed* representa a velocidade da tecnologia de rede do segmento em que o *probe* está anexado.

$$t.u.r = \frac{\left(\frac{\left[\left(nlHostInOctets_{t_2} + nlHostOutOctets_{t_2} \right) - \left(nlHostInOctets_{t_1} + nlHostOutOctets_{t_1} \right) \right] \bullet 8}{ifSpeed} \right)}{t_2 - t_1} \bullet 100 \qquad (1)$$

A figura 3.1 mostra um exemplo de um gráfico, o qual pode ser gerado por requisições ao grupo *nlHost* e aplicando a fórmula acima.

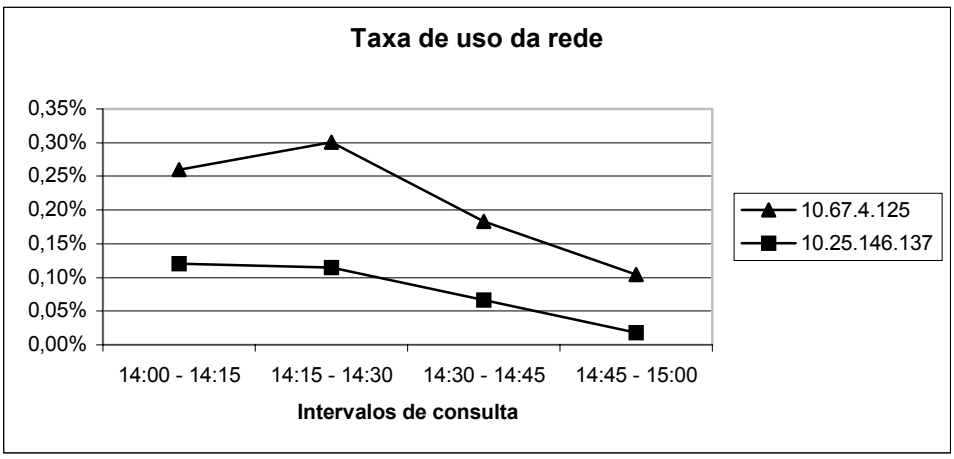


FIGURA 3.1 – Exemplo de gráfico retratando a taxa de utilização da rede por dois *hosts*

3.1.2 Aplicações e protocolos usados

Após ter determinado o quanto uma estação (usuário) utiliza da rede, o gerente busca descobrir quais os protocolos e aplicações que cada usuário executa, assim como em que períodos eles são monitorados. Tais informações podem ser obtidas através de requisições periódicas ao grupo *application-layer host* ou configurando relatórios remotos para os objetos do grupo *alHost* em diferentes instantes de tempo.

A tabela *alHost* fornece para o administrador estatísticas do tráfego de entrada e saída dos *hosts* considerando os protocolos da camada de aplicação. O termo camada de aplicação, como mencionado na descrição da respectiva tabela no capítulo da MIB RMON2, refere-se a todos os protocolos acima da camada de rede.

Há uma ou mais entradas na tabela *alHost* para cada protocolo da camada de aplicação descoberto. Essas entradas são também organizadas de acordo com seus endereços da camada de rede, logo, informações como o tráfego HTTP gerado de ou endereçado a um certo *host* pode ser facilmente obtido.

A tabela 3.2 ilustra um exemplo de dados recuperados do grupo *alHost*.

TABELA 3.2 – Exemplo de informação recuperada do grupo *alHost*

HostAddress	*Protocol*	*InPkts*	*OutPkts*	*InOctets*	*OutOctets*
10.25.152.3	HTTP	830	342	45.311	42.543
10.25.152.3	SMTP	250	24	17.900	2.406
10.25.152.3	FTP	567	158	32.193	19.765
10.25.152.45	HTTP	2037	411	209.312	56.927

A informação provida pelo grupo *alHost* pode ser usada para desenhar gráficos que indiquem ambos os protocolos e aplicações em uso por um certo *host* e a quantidade de tráfego gerado por cada um deles. A figura 3.2 mostra um exemplo de um gráfico que pode ser obtido por requisições ao *probe* em dois intervalos de tempo (t_1 e t_2). Ele representa a taxa de uso de cada protocolo pelo *host* 10.25.152.3 baseado no volume total de pacotes observados vindo de e endereçados a ele durante o intervalo de tempo t_2-t_1. A fórmula para calcular essa taxa de uso é a seguinte:

$$t.u.a = \frac{(alHostInOctets_{t2\,prot.app} + alHostOutOctets_{t2\,prot.app}) - (alHostInOctets_{t1\,prot.app} + alHostOutOctets_{t1\,prot.app})}{nlHostInOctets_{t2} + nlHostOutOctets_{t2}) - (nlHostInOctets_{t1} + nlHostOutOctets_{t1})} \bullet 100 \quad (2)$$

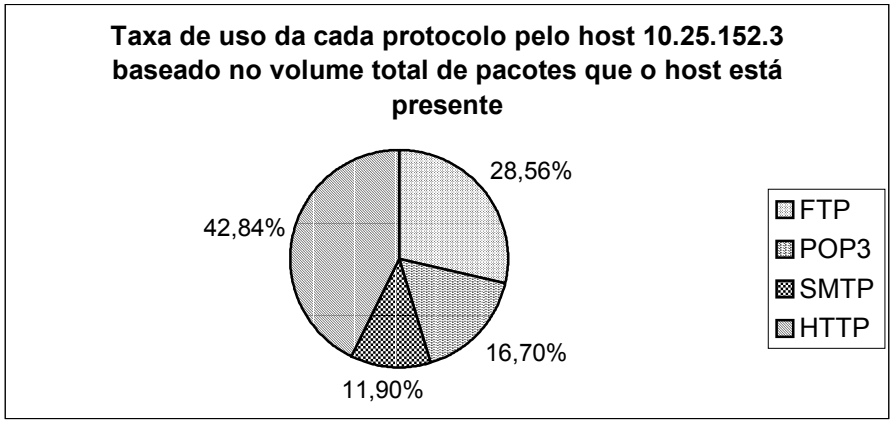


FIGURA 3.2 – Exemplo de gráfico mostrando a taxa de uso dos protocolos baseado no volume total de pacotes observados de um determinado *host*

A figura 3.3 mostra a taxa de uso do protocolo considerando quatro intervalos consecutivos de tempo. Esse tipo de gráfico histórico ajuda o administrador a determinar os perfis de acesso dos usuários. Nesse gráfico, a taxa de uso do protocolo foi calculada baseado na velocidade do segmento monitorado. Então, é possível observar o impacto que um protocolo em uso por um certo *host* está causando à rede. A fórmula para obter essa informação é:

$$t.u.a = \frac{\left(\frac{\left[(alhostInOctets_{t2} + alHostOutOctets_{t2}) - (alhostInOctets_{t1} + alHostOutOctets_{t1})\right] \bullet 8}{IfSpeed}\right)}{t2 - t1} \bullet 100 \quad (3)$$

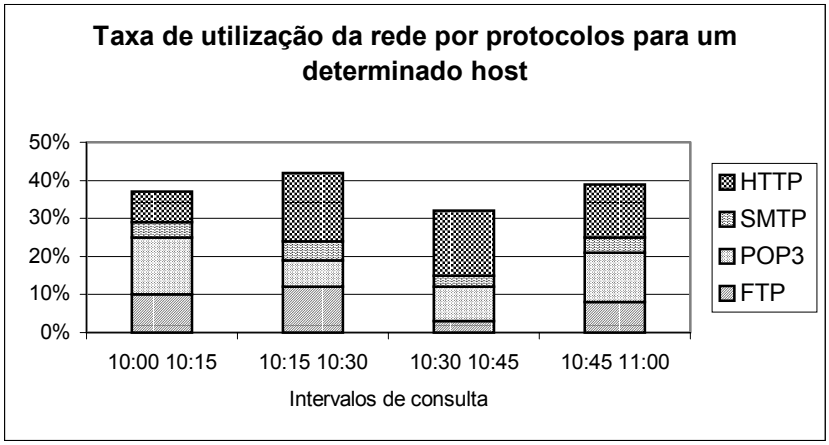


FIGURA 3.3 – Exemplo de gráfico representando a taxa de uso da rede por protocolos para um determinado *host*

3.1.3 Comunicações estabelecidas

Outro fator importante para melhor entender o comportamento de usuários da rede é identificar quais são os pares fonte/destino de cada comunicação estabelecida. O grupo *network-layer matrix* tem um importante papel nesse processo. Ele coleta estatísticas de tráfego entre comunicações de pares de *host* baseado em seus endereços da camada de rede. Através desse grupo pode-se obter os pares de *hosts* que se comunicam e através da derivação dessas informações chegar a respostas tais como: quais clientes acessam um servidor, com que estações uma determinada estação estabelece comunicação, o quanto um par de *hosts* utiliza da rede e em que períodos essa utilização é maior.

No entanto, o grupo *network-layer matrix* mostra as comunicações para encapsulamentos de protocolo da camada de rede. Para determinar, além dos pares de máquina, os protocolos de aplicação que essas comunicações utilizam juntamente com o tráfego gerado por eles o grupo *application-layer matrix* deve ser o escolhido.

Pode-se observar na tabela 3.3 a contabilização do tráfego da origem ao destino e vice-versa em duas diferentes entradas. Através de requisições periódicas ao *probe* RMON2 ou por meio de relatórios configurados no grupo *user history*, é possível obter as informações mencionadas.

TABELA 3.3 – Exemplo de informações recuperadas da tabela *alMatrixSD*

SDSourceAddress	*SDDestAddress*	*Protocol*	*SDPkts*	*SDOctets*
10.25.111.200	google.com	16 (ether2/ip/tcp/www-http)	15	578
google.com	10.25.111.200	16 (ether2/ip/tcp/www-http)	237	17.900
10.25.111.200	ftp.redhat.com	17 (ethe2/ip/tcp/ftp)	29	2.193
ftp.redhat.com	10.25.111.200	17 (ethe2/ip/tcp/ftp)	12.033	409.312
10.25.111.140	10.25.111.40	18 (ether2/ip/tcp/pop3)	307	36840
10.25.111.40	10.25.111.140	18 (ether2/ip/tcp/pop3)	700	90.582
10.25.111.200	10.25.111.1	16 (ether2/ip/tcp/www-http)	49	5.971
10.25.111.1	10.25.111.200	16 (ether2/ip/tcp/www-http)	14.987	1.000.329
10.25.111.200	10.25.111.40	19 (ether2/ip/tcp/smtp)	20	1.986

A partir dessa tabela pode-se gerar diversos gráficos. Como exemplificação, a figura 3.4 mostra as conexões feitas entre pares de *hosts* diferenciando-as pelo protocolo de aplicação em uso.

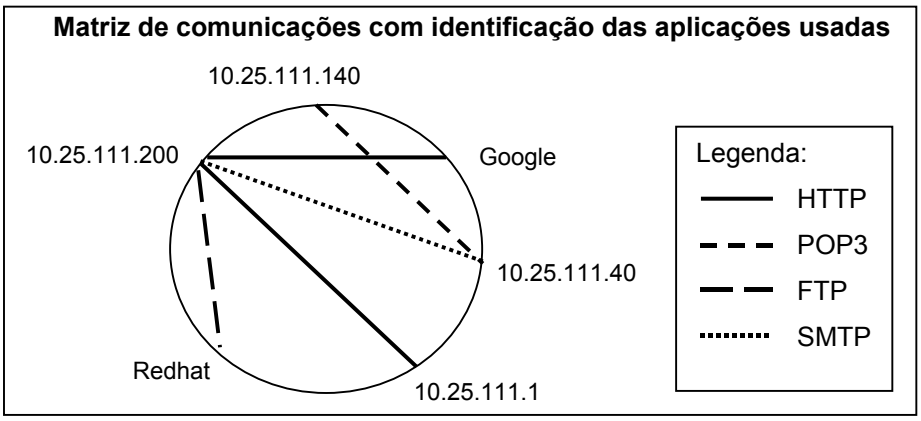


FIGURA 3.4 – Conexões estabelecidas entre os *hosts* e identificação das aplicações utilizadas

3.2 Perfil global do uso da rede

Nas seções anteriores, mecanismos para controlar e monitorar as atividades da rede baseados nos *hosts* foram apresentados. Entretanto, em muitas situações o administrador deve investigar o perfil de uso da rede levando em consideração a organização ou uma aplicação como um todo ou alguns conjuntos de *hosts*. Por exemplo, tal informação é essencial se o administrador quer descobrir quais as divisões que mais consomem recursos da rede, quando a rede está sobrecarregada e quais usuários/grupos contribuem para intensificar este problema.

O grupo *protocol distribution* contabiliza o número de octetos e pacotes monitorados pelo *probe* para cada encapsulamento de protocolo suportado, sem considerar os *hosts* envolvidos.

A tabela 3.4 mostra um exemplo de informação recuperada do grupo *protocolDist.* A partir dessa tabela pode-se obter informações como:

- O quanto um encapsulamento de protocolo utiliza da largura de banda do segmento de rede durante um certo período de tempo;
- A variação da taxa de uso da rede durante um certo período de tempo;
- Do volume total de pacotes observados no segmento para um determinado protocolo de rede, qual a porcentagem que as aplicações que rodam sobre o

respectivo protocolo de rede (observadas no segmento) utilizam desse volume total.

TABELA 3.4 – Exemplo de informação recuperado do grupo *protocolDist*

Protocolo	*protocolDistStatsPkts*	*protocolDistStatsOctets*
15 (ether2/ip)	9873	3103761
16 (ether2/ip/tcp/www-http)	1644	459100
23 (ether2/ip/tcp/smtp)	1290	345923
24 (ether2/ip/tcp/ftp)	483	229564
3333 (ether2/ip/tcp/Arca)	6456	2069174

A figura 3.5 mostra a taxa de utilização da banda pelos protocolos de aplicação observados pelo *probe* em um certo período de tempo, utilizando para a geração do gráfico os dados da tabela sobre a fórmula abaixo descrita.

$$t.u.p = \frac{\left(\frac{\left[(noctetos_{t2} - noctetos_{t1}) \bullet 8\right]}{ifSpeed}\right)}{t2 - t1} \bullet 100 \qquad (4)$$

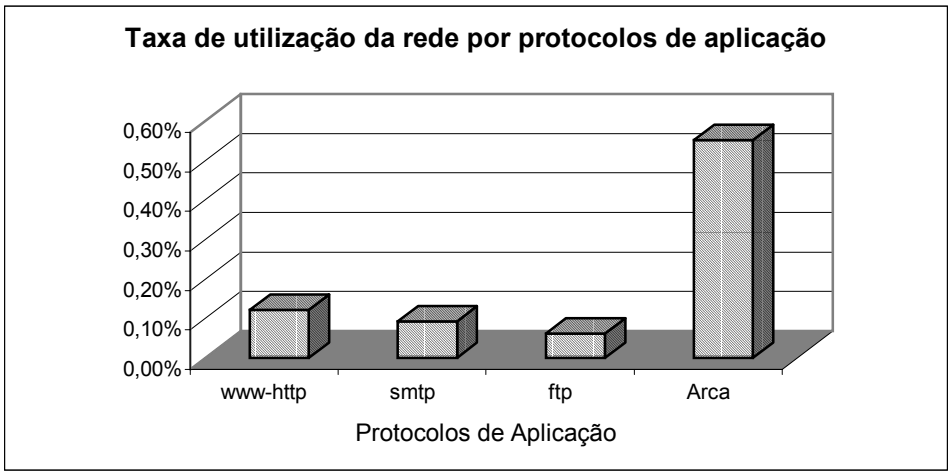


FIGURA 3.5 – Taxa de utilização da banda pelos protocolos de aplicação

A realização de requisições periódicas a esse grupo torna possível, também, a observação na variação da taxa de uso da rede durante um certo período de tempo (vide figura 3.6).

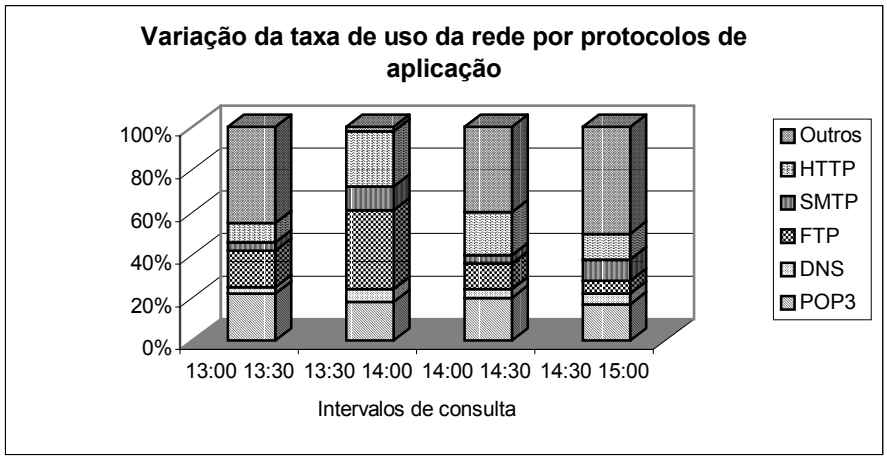


FIGURA 3.6 – Distribuição de protocolos ao longo do tempo

Para determinar a taxa de uso da rede de cada agrupamento, o administrador deve saber quais *hosts* pertencem a cada agrupamento. Nesse caso, o grupo a ser pesquisado é o *network-layer host*. A metodologia a ser usada é a seguinte: o *probe* contabiliza pacotes e octetos de entrada e saída para cada *host* identificado. No fim de um intervalo de contabilização, a taxa de uso da rede é calculada para cada *host* usando a fórmula apresentada em (1). Depois, *hosts* de cada agrupamento são agrupados e suas taxas de uso da rede são adicionadas.

No entanto, se deseja saber o quanto um grupo utiliza da rede para uma determinada aplicação, a tabela a ser consultada é a *alhost*.

O gráfico na figura 3.7 foi plotado baseado nos valores dos objetos coletados em dois diferentes instantes de tempo. Nesse caso, a informação resultante reflete a situação da rede neste intervalo específico para a aplicação "Arca".

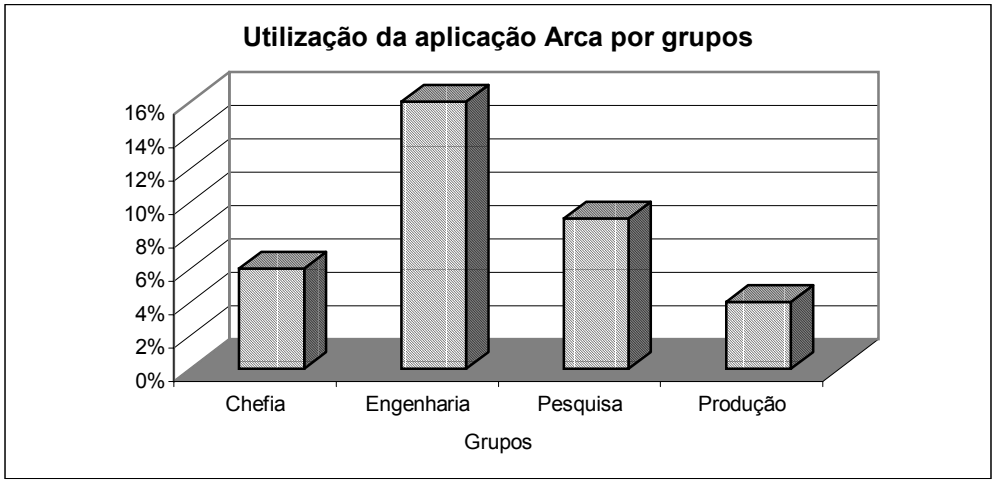


FIGURA 3.7 – Taxa de uso da rede por agrupamentos para uma específica aplicação

Outra possibilidade é realizar tais medições em vários momentos do dia, por vários dias e calcular a média das taxas obtidas. Essa metodologia provê ao administrador uma informação mais precisa sobre a utilização da rede em cada departamento no caso de uma organização ou agrupamento que utiliza uma aplicação.

O tipo de informação comentado acima pode ser decisivo na alocação de custos, uma vez que ele ajuda o administrador a decidir onde investir ou serve como referência

para deliberar privilégios de uma certa aplicação quanto ao uso da rede. Nesse contexto, a identificação dos *hosts* que realizam muitos dos acessos é também importante. Tal informação pode ser encontrada nas tabelas *topN* do grupo *network-layer matrix*.

As tabelas *topN* provem um eficiente meio para uma estação de gerenciamento obter os pares de *hosts* que mais se comunicam através de uma ordenação decrescente das entradas da tabela *matrix* baseadas em uma escolha de estatística de rede (número de octetos transmitidos de um *host* origem para um *host* destino) [WAL97][PER99].

A tabela 3.5 apresenta os dados recuperados da tabela *nlMatrixTopN*. Esses dados tornam possível a criação de gráficos como na figura 3.8, os quais mostram os usuários que mais consomem recursos da rede.

TABELA 3.5 – Exemplo de informações recuperadas da tabela *nlMatrixTopN*

Protocolo	Endereço origem	Endereço destino	PktRate	Reserve PktRate	OctetRate	Reverse OctetRate
15(ether2/ip)	172.16.108.12	172.16.108.1	213	32	40065	6023
15(ether2/ip)	172.16.108.45	172.16.108.23	156	17	23913	2194
15(ether2/ip)	172.16.106.25	200.248.252.1	89	29	12882	6745
15(ether2/ip)	172.16.109.7	172.16.108.1	67	13	5294	968

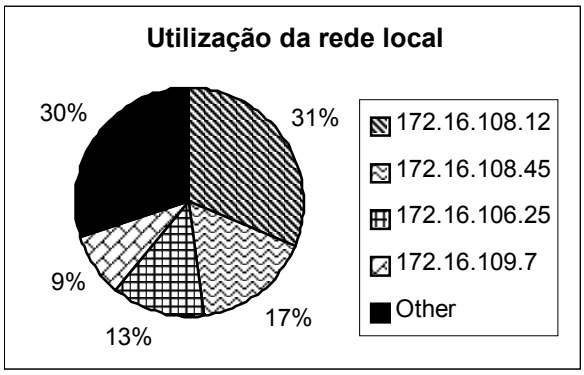


FIGURA 3.8 – Usuários que mais consomem recursos da rede

3.3 Otimização de usuários e distribuição de recursos

Uma importante contribuição do RMON2 é a possibilidade de verificar se usuários e recursos estão adequadamente posicionados na rede de maneira a aumentar ao máximo o isolamento do tráfego em cada departamento da companhia [GAS99b]. O grupo que fornece essa informação é o *application-layer matrix*, comentado na seção 3.1.3. O procedimento na figura 3.9 pode ser aplicado para otimizar a localização de usuários e recursos.

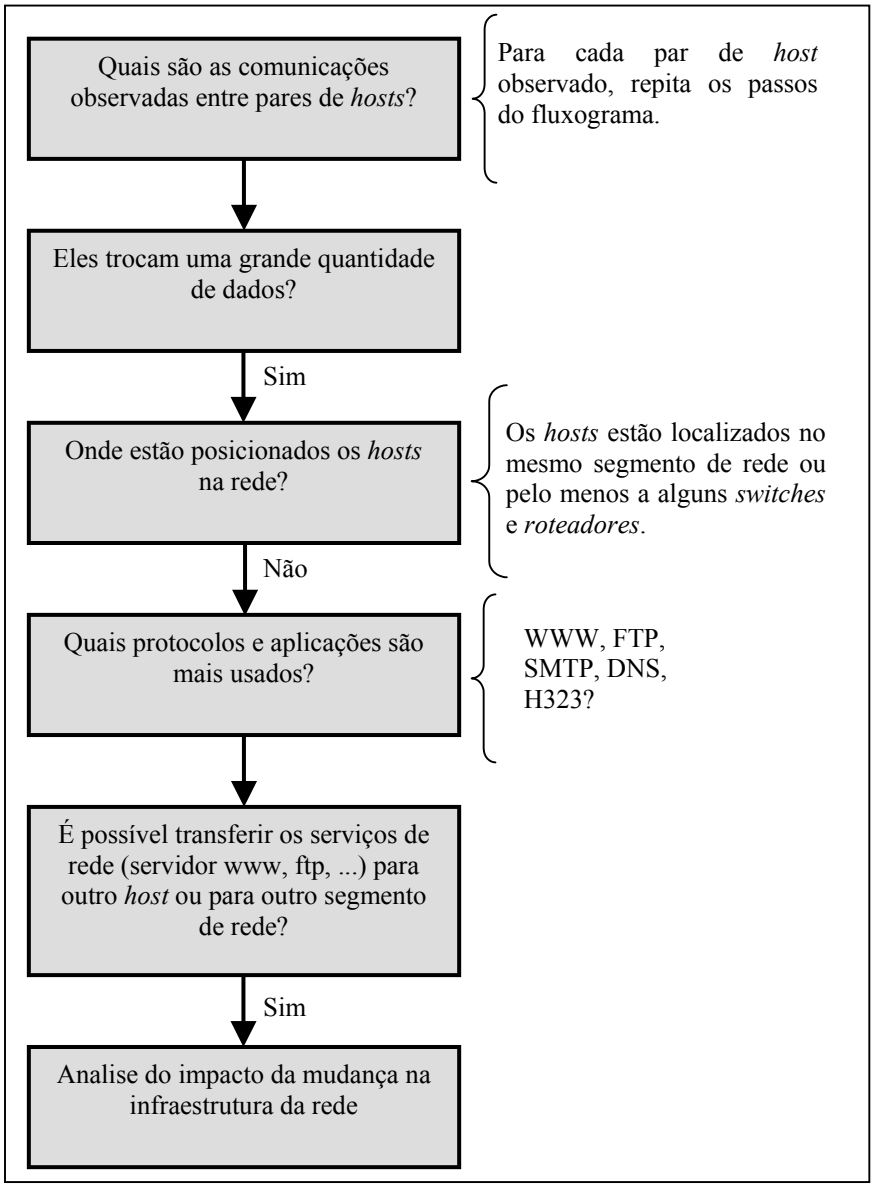


FIGURA 3.9 – Algoritmo para otimização da localização de usuários e recursos

Em alguns casos, usuários e recursos estão apropriadamente posicionados, minimizando o tráfego entre diferentes segmentos de rede. Entretanto, alguns recursos podem estar sobrecarregados, os quais afetam os tempos de resposta de protocolos e aplicações. Nestes casos balanceamento de carga é necessário. Para fazer isso, o administrador deve medir a taxa de uso de um certo recurso. Essa informação é fornecida pelo grupo *application-layer matrix*.

A metodologia usada para determinar as atuais atividades sobre um certo recurso é a seguinte. O *probe* contabiliza pacotes e octetos de entrada e saída para cada par de *hosts* identificados. O monitoramento pode ser realizado em dois intervalos de tempo (instantes t_1 e t_2) ou periodicamente. No fim de um intervalo de contabilização, todas as entradas cujo endereço fonte ou destino é o mesmo do recurso monitorado são selecionadas. As entradas selecionadas são então agrupadas de acordo com o protocolo da camada de aplicação. Depois, para cada grupo, os octetos de entrada e saída são adicionados.

A tabela 3.6 ilustra a metodologia recém apresentada. Na monitoração, pode-se observar o tráfego HTTP, SMTP e Oracle. Então, as entradas podem ser arranjadas em três grupos. No grupo HTTP, o número de octetos de entrada (34.567) e de saída (125.954) é adicionado no instante t_1(160.521). O mesmo cálculo é repetido no instante t_2 (1.060.521). A quantidade de tráfego HTTP observado no recurso é obtido subtraindo o resultado da soma dos octetos de entrada e saída nos instantes t_1 e t_2. O mesmo cálculo deve ser realizado nos outros grupos.

TABELA 3.6 – Atividades da rede em um certo *host*

Source Address	*Dest. Address*	*Protocol*	*Octets (t_1)*		*Octets (t_2)*		*t_2-t_1*
172.16.108.12	172.16.108.1	HTTP	34.567	160.521	192.224	1.060.521	900.000
172.16.108.1	172.16.108.12	HTTP	125.954		864.297		
172.16.109.5	172.16.108.1	SMTP	37.234	58.123	220.211	314.123	256.000
172.16.108.1	172.16.109.5	SMTP	20.889		93.912		
172.16.108.12	172.16.108.1	Oracle	45.082	68.863	341.090	534.834	465.971
172.16.108.1	172.16.108.12	Oracle	23.781		193.744		

A figura 3.10 mostra um exemplo de um gráfico, mostrando as atividades de rede de um *host* servidor ao longo do dia.

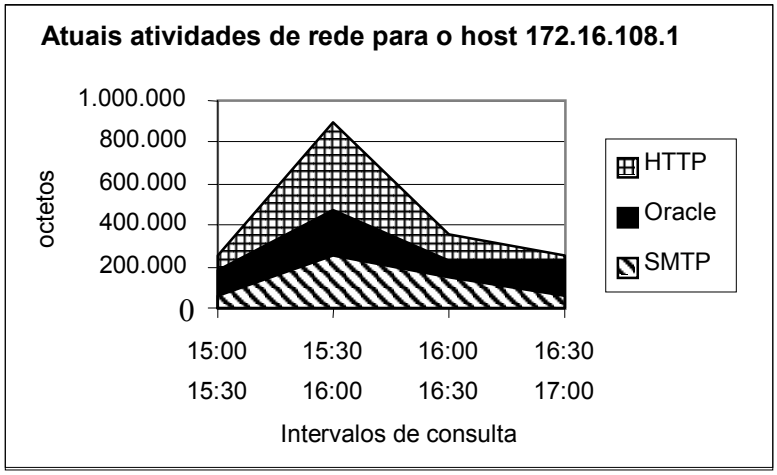


FIGURA 3.10 – Atividades observadas em um *host* servidor de rede

3.4 Gerenciamento de segurança

Segurança é uma questão essencial às redes corporativas. Atualmente, mais e mais mecanismos tais como *firewalls* têm sido incorporados nas redes para manter os intrusos longe dos dados estratégicos da companhia [PER99].

A RMON2 pode ser usada como uma ferramenta para detectar a presença de intrusos na rede. Como já mencionado, o grupo *application-layer matrix* mostra os pares de *host* que estão se comunicando. Portanto, se o administrador periodicamente monitorar esse grupo, ele pode identificar usuários não autorizados tentando estabelecer comunicações com *hosts* da rede. É também importante observar o protocolo que está

sendo usado. Dependendo das políticas de segurança da companhia, um *telnet* pode representar uma tentativa de invasão.

Quando um endereço de rede suspeito é observado, alguém pode começar a capturar os pacotes gerados pelo *host* e analisar as operações que ele está executando. Essa operação é suportada pelo grupo *capture* da MIB RMON.

O gráfico na figura 3.11 mostra os usuários que estão acessando certos recursos de um determinado servidor. É adequado monitorar *hosts* estratégicos onde residem servidores de base de dados, WWW e correio eletrônico. O Gráfico é gerado com informações retiradas do grupo *application-layer matrix*.

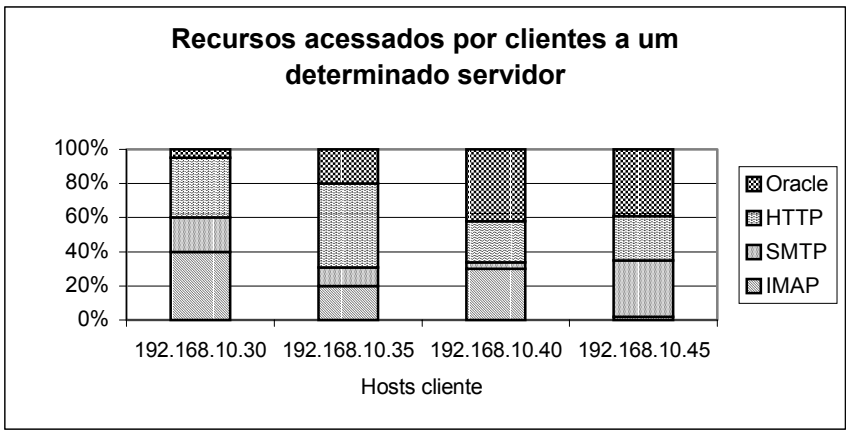


FIGURA 3.11 – Usuários e seus fluxos de tráfego para um determinado servidor

Em alguns casos, pode ser útil identificar quais protocolos cada usuário observado executa. Essa informação pode ser obtida no grupo *alHost*, mencionado na seção 3.1.2.

Outra possibilidade de tentativa de invasão que pode ser constatada através da tabela *alHost* diz respeito à verificação de um comportamento anômalo. Por exemplo, se está se monitorando um servidor Web que tem como comportamento padrão enviar mais octetos do que receber e constata-se que está ocorrendo o inverso. Logo, tal procedimento pode representar alguma tentativa de ataque.

4 O Agente RMON2

Esse capítulo tem por finalidade descrever a estrutura que foi desenvolvida para atender a proposta da criação do agente RMON2. O capítulo aborda a especificação geral do agente (plataforma no qual ele foi desenvolvido, o sistema operacional utilizado, a linguagem de programação usada), os programas, pacotes, bibliotecas que foram necessários para a composição da sua estrutura básica e os módulos que compõem o agente detalhadamente.

O agente foi projetado para ser usado sobre o Sistema Operacional Linux. Tal escolha se deve ao fato do Linux ser um sistema operacional robusto, estável, possuir um ótimo desempenho, ser escalável (permitindo o uso de máquinas com poucos recursos), amplamente utilizado pela comunidade científica, seguir o princípio de código aberto e *free.*

Através da escolha do Linux, a plataforma de *hardware* que pode ser utilizada é qualquer uma que seja compatível com o sistema operacional.

Além da escolha do Linux como sendo a base do agente, outros *softwares* foram necessários para solucionar requisitos como a captura de pacotes da rede, armazenamento de dados e agente SNMP que pudesse ser estendido.

Para solucionar o problema da captura dos pacotes, a biblioteca de captura `Libpcap` [MAC2001] desenvolvida pelo grupo de redes do Laboratório Nacional de *Lawrence Berkeley* foi à escolhida por ser amplamente utilizada para esse propósito pela comunidade científica e além do mais, ferramentas como, por exemplo, `ntop` [DER99], `Ethereal` [COM2001], `tcpdump` [JAC2001] a utilizam para tal propósito.

Quanto ao armazenamento dos dados, a solução encontrada foi o uso do banco de dados `MySql` [MYS2001a], pois comparando-o com outros banco de dados, como `PostgreSQL` e `mSQL`, apresenta melhor desempenho, sendo esse fator o que mais pesou na escolha. O software é leve e ao mesmo tempo oferece um rico e altamente útil conjunto de funções, é muito bem aceito pela comunidade científica, é *multi-threaded* e também é *free*.

O pacote `Net SNMP` [NET2001] que é o novo nome do projeto `UCD SNMP` foi quem solucionou o problema de um agente SNMP que pudesse ser estendido. O pacote é um agente SNMP que possibilita a criação de agentes estendidos, possui ferramentas para requisitar ou configurar informações de agentes SNMP, entre outras utilidades.

A figura 4.1 dá uma visão da estrutura do agente em uma visão de camadas.

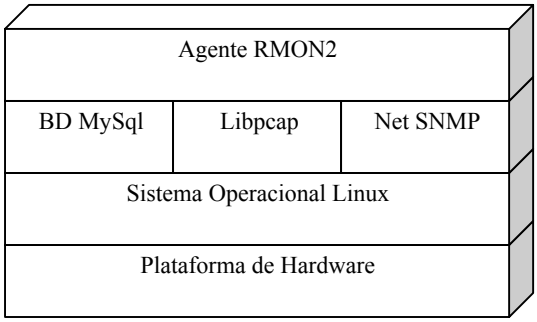


FIGURA 4.1 – Visão esquemática, em camadas, da estrutura do agente

A implementação do agente foi realizada através do uso da linguagem de progamação C. Tal escolha deve-se a portabilidade que os *softwares* escolhidos para

compor o agente tem para com a linguagem, assim como a comunicação com o banco de dados, a qual é realizada através de uma API desenvolvida em C.

Após ter detalhado os recursos usados na implementação do agente, passa-se a referenciar o que o agente, ora proposto, implementa da MIB RMON2 padronizada. Ele implementa seis grupos da respectiva MIB, sendo que tais grupos não foram escolhidos de forma aleatória, mas sim através de uma pesquisa do que se pode obter da MIB RMON2 especificamente no monitoramento de aplicações e protocolos de alto-nível. O que cada grupo pode fornecer de estatísticas para o gerente pôde ser visto no capítulo 3.

Partindo para a descrição dos grupos implementados, temos:

- *Protocol Directory*;
- *Protocol Distribution*;
- *Network-Layer Host*;
- *Network-Layer Matrix*;
- *Application-Layer Host*;
- *Application-Layer Matrix*.

Como pode ser observado, os grupos *AddressMap*, *UsrHistory* e *ProbeConfig* ficaram de fora, pois esses grupos não oferecem informações sobre protocolos de alto nível. Uma resalva pode ser feita ao grupo *UsrHistory* que pode obter essas informações desde que o gerente o configure.

O agente foi dividido em módulos e bases de dados. Cada qual tem funções específicas que quando sincronizadas formam o agente. Nas subseções seguintes do capítulo far-se-á um detalhamento dos componentes do agente.

A figura 4.2 apresenta uma visão geral da estrutura do agente, como os módulos e bases de dados, além de propiciar a visualização do caminho percorrido pelos dados desde a captura do pacote até a obtenção das estatísticas do comportamento da rede pelo gerente.

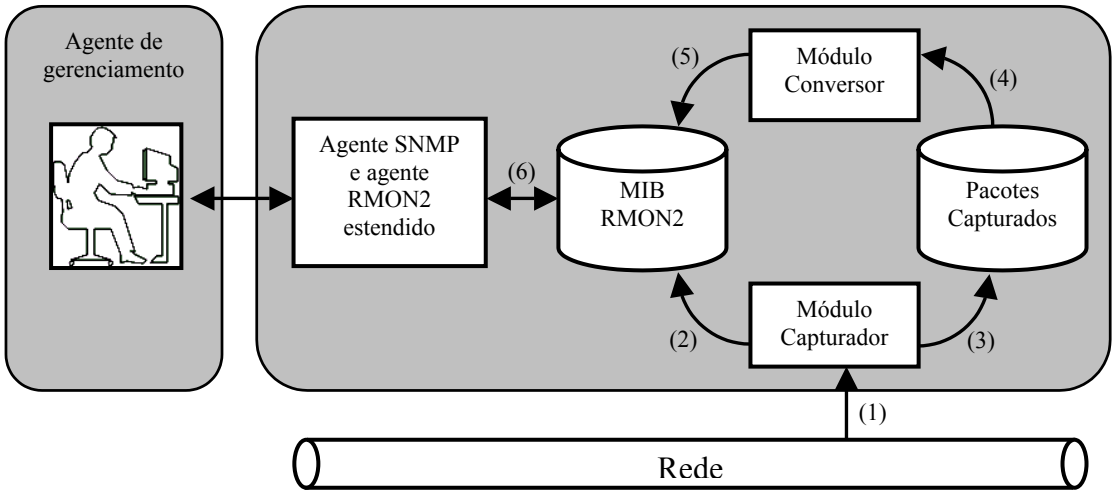


FIGURA 4.2 – Desenho esquemático do agente

4.1 Módulo Capturador

O módulo "capturador" é um programa executável bastante simples. Ele é o responsável pela captura dos pacotes que trafegam pelo segmento de rede ao qual o agente está conectado. Para capturar os pacotes, o módulo utiliza a biblioteca de captura

`Libpcap` e, para armazenar os dados obtidos pela captura, utiliza a API para a conexão com o banco de dados `MySql`.

Preliminarmente, o módulo obtém a interface que capturará os pacotes de um arquivo de configuração e, em seguida, uma conexão com a base de dados que armazenará os pacotes capturados é aberta (vide fluxo 3 da figura 4.2). Além da abertura da conexão com a base de dados que conterá os pacotes capturados, uma conexão com a base de dados RMON2 é estabelecida (vide fluxo 2 da figura 4.2) apenas para inserir nas tabelas de controle (*protocoldistcontrol*, *hlhostcontrol*, *hlmatrixcontrol*) uma entrada que representa a interface na qual se está fazendo a captura.

A partir de então, um processo é disparado com a função de configurar a interface de rede para operar em modo promíscuo e, posteriormente, este processo fica aguardando a chegada dos pacotes para então decodificá-los.

Assim que um pacote é capturado, o mesmo passa pela fase de decodificação, que o analisará e obterá certos parâmetros úteis para servir como base aos passos subseqüentes do agente.

O primeiro passo a ser feito na decodificação é analisar o cabeçalho da camada de enlace. Cabe ressaltar que o agente proposto se preocupa apenas com o protocolo de encapsulamento *Ethernet*. Os campos do cabeçalho utilizados são o *type* e o *destination address*. Através do campo *type*, verifica-se o seu respectivo valor no intuito de descobrir se o protocolo da camada de rede é o IP. Caso a condição falhe, o pacote é descartado. Já o campo *destination address* é usado para a verificação da ocorrência de um pacote broadcast (endereço MAC igual a FF-FF-FF-FF-FF-FF) ou multicast (endereço MAC iniciando por 01-00-5E).

Passando com sucesso no primeiro passo da decodificação, o cabeçalho IP do pacote é observado. Através do campo *version* busca-se descobrir a versão do protocolo IP. Se a versão for a quatro (utilizada atualmente na rede Internet), o pacote não é descartado e, em seguida descobre-se o protocolo da camada de transporte através do campo *protocol* que pode ter o valor seis (6), representando o protocolo TCP, ou dezessete (17), representando o protocolo UDP. Dentro do cabeçalho desses protocolos de transporte obtém-se as portas origem e destino do pacote e, a partir delas, pode ser obtido o protocolo de aplicação. Os endereços IP origem e destino são obtidos através do campo *source* e *dest* do cabeçalho do protocolo de rede IP.

Com o término da fase da análise, caso o pacote não tenha sido descartado, os parâmetros obtidos do pacote são gravados na base de dados dos pacotes capturados. Para fins de recapitulação, os parâmetros salvos na base são:

- Endereço IP do *host* origem do pacote;
- Endereço IP do *host* destino do pacote;
- Porta origem do pacote;
- Porta destino do pacote;
- Instante de tempo em que o pacote foi capturado;
- Número de octetos do pacote;
- Protocolo da camada de enlace;
- Protocolo da camada de rede;
- Protocolo da camada de transporte;
- Interface em que foi capturado;
- Pacote é não unicast (broadcast ou multicast).

Um fluxograma demonstrando todos os passos no qual o módulo capturador executa desde a espera por um pacote até a inclusão do mesmo na base de dados dos pacotes capturados pode ser visto na figura 4.3.

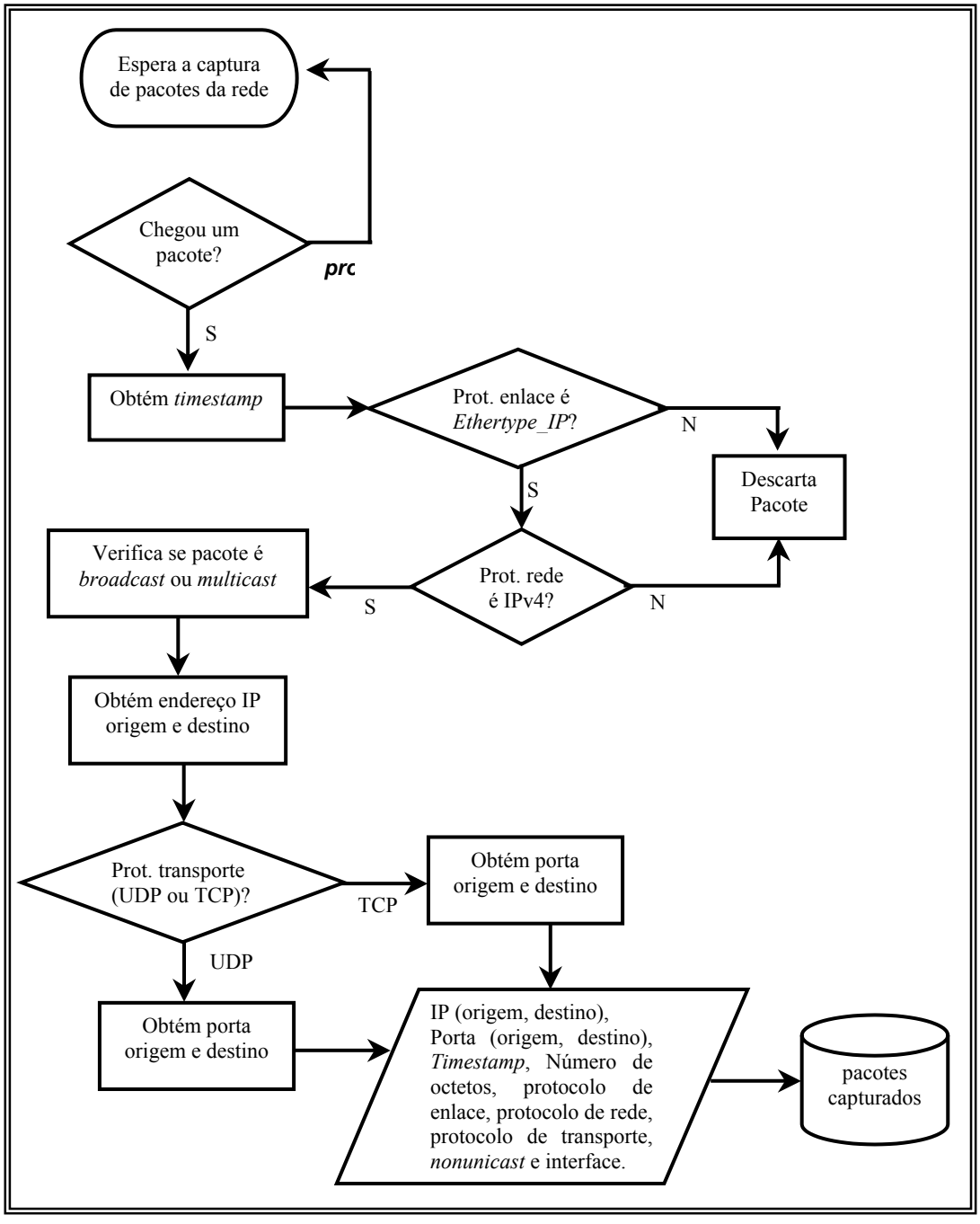


FIGURA 4.3 – Fluxograma do módulo capturador

4.2 Base de Dados dos Pacotes Capturados

Através da necessidade de se ter armazenado em algum repositório todos os pacotes capturados pelo módulo de captura e, além disso, utilizá-lo para popular as tabelas da MIB RMON2, fez-se necessária a criação de uma base de dados. Essa base criada

possui uma única tabela, a qual é a responsável pelo armazenamento dos pacotes. Cada tupla da tabela corresponde a um pacote capturado.

Seguindo a escolha do banco de dados `MySql` como o responsável pelo armazenamento dos dados do agente, a base de dados criada para atender tal necessidade recebeu o nome "cap_pac". Já a tabela pertencente à base, atende pelo nome "conexao" e possui doze campos, tendo os mesmos sido mencionados na seção anterior (módulo capturador).

A estrutura da tabela com seus respectivos campos e parâmetros pode ser visualizada na tabela 4.1.

TABELA 4.1 – Detalhamento da tabela "conexao"

Nome			Tipo		Chave Primária
Indice	bigint(20)	unsigned	NOT NULL	Auto_increment	*
Iporigem	int(11)		NULL		
Ipdestino	int(11)		NULL		
Porigem	smallint(5)	unsigned	NULL		
Pdestino	smallint(5)	unsigned	NULL		
Time	bigint(20)	unsigned	NULL		
Numoctets	smallint(5)	unsigned	NULL		
Protlink	smallint(5)	unsigned	NULL		
Protnet	smallint(5)	unsigned	NULL		
Prottrans	tinyint(3)	unsigned	NULL		
Indexcontrol	tinyint(3)	unsigned	NULL		
NonUnicast	tinyint(3)	unsigned	NULL		

Cada campo armazena os seguintes valores:

- Indice – esse campo indexa cada entrada da tabela. Ele é a chave primária da mesma e foi criado para ser usado pelo módulo de conversão. O tipo do campo é um *bigint* sem sinal, logo podendo o seu valor variar de 0 até 18.446.744.073.709.551.615 conforme descrito no manual de referência do `MySql` [MYS2001b];
- Iporigem – armazena o endereço IP do *host* origem num formato decimal. O campo é do tipo *int* e seu valor pode variar de –2.147.483.648 até 2.147.483.647;
- Ipdestino – armazena o endereço IP do *host* destino num formato decimal. O campo possui os mesmos requisitos do "Iporigem";
- Porigem – armazena o valor da porta origem do pacote, podendo ser a porta de um protocolo de aplicação ou uma porta alocada dinamicamente acima da porta 1024 (no caso de ser um cliente requisitando uma aplicação ou até mesmo uma aplicação proprietária que utiliza uma porta específica). Seu valor é o mesmo encontrado no campo *source port number* tanto do cabeçalho do protocolo UDP quanto do TCP. Como o campo é do tipo *smallint* sem sinal a variação do valor pode ser de 0 até 65.535;
- Pdestino – armazena o valor da porta destino do pacote. Os valores para esse campo são os mesmos que podem ser encontrados no campo "Porigem". O valor para o campo é encontrado no campo *destination port number* tanto do cabeçalho do protocolo UDP quanto do TCP. Por ser do mesmo tipo do campo Porigem, seu valor segue os mesmos limites;
- Time – armazena o instante do tempo em que o pacote foi capturado. O valor do tempo é expresso em milissegundos, representa a quantidade de tempo desde a inicialização do agente até o momento da captura e é obtido através de uma

função que utiliza o arquivo "*/proc/time*" do Sistema Operacional Linux. Este arquivo contém um dado, expresso em milisegundos, que representa a quantidade de tempo que a máquina está ligada. Seu campo tem o tipo *bigint* sem sinal e sua variação segue a descrita do campo "Indice";

- Numoctets – armazena o número de bytes que o pacote possui. Seu tipo é *smallint* sem sinal;
- Protlink – armazena o valor que representa o protocolo da camada de enlace em notação decimal. Como o agente monitorará, apenas as aplicações que são executadas sobre o protocolo IP e esse sobre o *Ethernet*, logo se tem um protocolo de enlace. Portanto, todos os pacotes terão o valor um (1) para esse campo. A escolha do tipo *smallint* sem sinal é para se ter uma maior maleabilidade na inclusão de novos protocolos;
- Protnet – armazena o valor que representa o protocolo da camada de rede em um formato decimal. Como se está analisando apenas aplicações que executam sobre o Ipv4, o valor para esse campo será sempre 2048 (valor do campo *type* do cabeçalho *Ethernet* convertido para decimal). O tipo atribuído é o mesmo do campo "Protlink";
- Prottrans – armazena o valor que representa o protocolo da camada de transporte. Nesse caso, podemos ter dois protocolos, o UDP e o TCP, cujos valores são respectivamente 17 e 6. Esses valores são os mesmos encontrados no campo *protocol* do cabeçalho IP. O seu tipo é *tinyint* sem sinal e tem a possibilidade de variar de 0 até 255;
- Indexcontrol – armazena um valor inteiro que representa a interface na qual este pacote foi capturado. Como a captura ocorre apenas em uma interface para esse agente, então o valor do campo será sempre um (1). O tipo do campo é o mesmo do campo "Prottrans". Embora o campo tem a possibilidade de ter uma variação de 255, se for implementada a captura de mais do que uma interface o máximo que poderá chegar será em torno de seis interfaces de captura por motivos de limitação de hardware e sistema operacional;
- Nonunicast – especifica se o pacote é um pacote *broadcast* ou *multicast*. Caso o pacote seja *broadcast* ou *multicast* o valor do campo é um (1), caso contrário o valor é zero (0).

4.3 Base de Dados da MIB RMON2

Através da necessidade de se ter a MIB RMON2 especificada para poder armazenar as informações oriundas da análise da rede, buscou-se a construção de uma base de dados, sobre o banco `Mysql`, que a represente. Cada tabela da MIB pertencente aos grupos implementados pelo agente teve, conseqüentemente, uma tabela criada na base de dados. Tal base criada recebeu o nome de "rmon2".

A fonte de inserção de dados às tabelas da base de dados é o módulo "conversor" em sua grande maioria. No entanto, algumas tabelas podem receber dados diretamente do módulo "capturador", como mencionado na descrição do módulo.

As subseções seguintes representam os grupos da MIB que o agente implementou. Dentro de cada grupo, suas respectivas tabelas são detalhadas. O detalhamento mostra as estruturas das tabelas com seus respectivos campos, o tipo dos campos, chaves primárias, chaves estrangeiras, entre outros.

4.3.1 Grupo *Protocol Directory*

Para atender o grupo *protocolDir*, duas tabelas foram criadas, uma para o objeto escalar *protocolDirLastChange* e a outra para a tabela *protocolDirTable*.

A tabela "lastchange" é a responsável pelo armazenamento do dado referente ao objeto escalar. Tal tabela não possui chave primária, pois tem apenas uma entrada. O único campo da tabela é o "Time" que armazena um valor *timestamp* em milissegundos e o seu valor pode variar de zero até 18.446.744.073.709.551.615 pelo fato do campo ser do tipo *bigint* sem sinal. A especificação da tabela pode ser vista na tabela 4.2.

TABELA 4.2 – Tabela "lastchange"

Nome	Tipo		Chave primária
Time	bigint(20)	unsigned	NOT NULL

Já a tabela *protocolDirTable* é representada pela tabela "protocoldir". Suas entradas correspondem a encapsulamentos de protocolo que podem ser interpretados ou não pelo *probe* e cada campo da tabela está associado a um objeto ou ao índice do objeto na MIB.

A tabela 4.3 mostra uma visão dos campos pertencentes à tabela.

TABELA 4.3 – Tabela "protocoldir"

Nome	Tipo			Chave primária
Idlink	smallint(5)	unsigned	NULL	
Idnet	smallint(5)	unsigned	NULL	
Idtrans	tinyint(3)	unsigned	NULL	
Idapp	smallint(5)	unsigned	NULL	
Par1	tinyint(3)	unsigned	NULL	
Par2	tinyint(3)	unsigned	NULL	
Par3	tinyint(3)	unsigned	NULL	
Par4	tinyint(3)	unsigned	NULL	
LocalIndex	smallint(5)	unsigned	NOT NULL	*
Descr	varchar(35)		NULL	
Type	tinyint(3)	unsigned	NULL	
AddressMapConfig	tinyint(3)	unsigned	NULL	
HostConfig	tinyint(3)	unsigned	NULL	
MatrixConfig	tinyint(3)	unsigned	NULL	
Owner	varchar(30)		NULL	
Status	tinyint(3)	unsigned	NULL	

Cada campo da tabela possui especificações distintas e eles representam um certo objeto da MIB RMON2. Os campos são assim especificados:

- Idlink – corresponde à identificação do protocolo da camada de enlace para a respectiva entrada. Esse campo é usado para fins de indexação e faz parte do objeto *protocolDirID*. Utiliza um tipo *smallint*;
- Idnet – corresponde à identificação do protocolo da camada de rede para a respectiva entrada. Usado para fins de indexação e pertence ao objeto *protocolDirID*;
- Idtrans – corresponde à identificação do protocolo da camada de transporte para a respectiva entrada. Caso o encapsulamento de protocolo seja um protocolo de rede, esse campo terá o valor zero. Mas se for um encapsulamento de transporte ou aplicação o valor do campo pode variar entre o valor seis (6) que representa o

protocolo UDP e dezessete (17) para o protocolo TCP. O campo é usado para fins de indexação e pertence ao objeto *protocolDirID*;

- Idapp – corresponde à identificação do protocolo da camada de aplicação para a respectiva entrada. Caso o encapsulamento de protocolo seja um protocolo de transporte ou de rede, esse campo terá o valor zero. No entanto, sendo um protocolo de aplicação o valor desse campo será o valor da porta que o protocolo usa na comunicação. Para entender melhor, se a entrada corresponde ao protocolo HTTP que utiliza a porta 80, então o campo terá o valor 80 armazenado. O campo é usado para fins de indexação e pertence ao objeto *protocolDirID*;
- Par1, Par2, Par3, Par4 – correspondem ao objeto *protocolDirParameters*. O valor do campo pode ser zero se for um *countsFragments* ou um se for *trackSessions* conforme especificado no subseção 2.1.2 do capítulo da MIB RMON2. Os campos são usados para fins de indexação;
- LocalIndex – esse campo identifica cada entrada da tabela de forma distinta, representando o objeto *protocolDirLocalIndex*. Seu valor é arbitrário e o campo é a chave primária da tabela;
- Descr – corresponde à descrição textual do encapsulamento de protocolo. Esse campo é o que possui o dado para o objeto *protocolDirDescr*. O seu tipo é um *varchar(35)* o qual significa que pode ser armazenado no campo até 35 caracteres;
- Type – corresponde ao objeto *protocolDirType* e seu valor varia entre zero se o tipo for extensível ou um caso o tipo do encapsulamento seja capaz de reconhecer endereços;
- AddressMapConfig – corresponde ao objeto *protocolDirAddressMapConfig* e portanto pode assumir um dos três valores possíveis que são o *notSupported* (1), *supportedOff* (2) ou *supportedOn* (3);
- HostConfig – corresponde ao objeto *protocolDirHostConfig* e pode assumir os mesmos valores do objeto "AddressMapConfig";
- MatrixConfig – corresponde ao objeto *protocolDirMatrixConfig* e pode assumir os mesmos valores do objeto "AddressMapConfig";
- Owner – corresponde ao objeto *protocolDirOwner*. Por ser um campo com o tipo *varchar(30)*, o nome do proprietário da tabela poderá ter um tamanho máximo de 30 caracteres;
- Status – corresponde ao objeto *protocolDirStatus* e pode ter seis valores diferentes. Os valores possíveis são: *active* (1), *notInService* (2), *notReady* (3), *createAndGo* (4), *createAndWait* (5) ou *destroy* (6).

4.3.2 Grupo *Protocol Distribution*

O grupo *protocol distribution*, como já é de conhecimento, possui duas tabelas. Uma de controle e a outra com as informações estatísticas. Para armazenar os valores dos objetos pertencentes a essas tabelas foram criadas duas tabelas. As tabelas "protocoldistcontrol" e "protocoldiststats" representam respectivamente as tabelas *protocolDistControlTable* e *protocolDistStatsTable* da MIB RMON2.

A tabela "protocoldistcontrol" tem a seguinte estrutura (vide tabela 4.4).

TABELA 4.4 – Tabela “protocoldistcontrol”

Nome	Tipo			Chave primária
Indice	tinyint(3)	unsigned	NOT NULL	*
DataSource	varchar(10)		NULL	
DroppedFrames	int(10)	unsigned	NULL	
CreateTime	bigint(20)	unsigned	NULL	
Owner	varchar(30)		NULL	
Status	tinyint(3)	unsigned	NULL	

Seus campos representam os objetos da tabela e as características de cada campo são:

- Indice – corresponde a um valor arbitrário que identifica cada entrada na tabela e, portanto, representa a chave primária da tabela. O campo é usado para identificar a interface que está capturando pacotes, representa o objeto *protocolDistControlIndex* e sua função é indexar as entradas;
- DataSource – armazena o nome da interface ao qual a entrada pertence. Por exemplo, para a interface ethernet 0 é armazenado o valor “eth0”. Representa o objeto *protocolDistControlDataSource*;
- DroppedFrames – representa o objeto *protocolDistControlDroppedFrames*;
- CreateTime – corresponde ao instante de tempo em que a entrada foi criada. Seu valor é expresso em milissegundos e representa o tempo desde que o agente foi inicializado até o momento da criação da entrada. Tal campo representa o objeto *protocolDistControlCreateTime*;
- Owner – corresponde ao objeto *protocolDistControlOwner*;
- Status – corresponde ao objeto *protocolDistControlStauts* e pode ter os mesmos valores do objeto “Status” da tabela “protocoldir”.

A outra tabela, “protocoldiststats”, está estruturada conforme a tabela 4.5.

TABELA 4.5 – Tabela “protocoldiststats”

Nome	Tipo			Chave primária
Indicecontrol	tinyint(3)	unsigned	NOT NULL	*
LocalIndex	Smallint(5)	unsigned	NOT NULL	*
Pkts	Int(10)	unsigned	NULL	
Octets	Bigint(20)	unsigned	NULL	

Seus campos têm as seguintes características:

- Indicecontrol – esse campo é uma chave estrangeira para a tabela “protocoldistcontrol” além de ser chave primária da tabela. Seu valor está relacionado com o campo “Indice” da tabela “protocoldistcontrol” e é usado para fins de indexação da tabela;
- LocalIndex – também corresponde a uma chave estrangeira e ao mesmo tempo faz parte da chave primária da tabela. Seu valor está relacionado ao campo “LocalIndex” da tabela “protocoldir” e é usado como índice da tabela;
- Pkts – corresponde o objeto *protocolDistStatsPkts* e guarda a quantidade de pacotes capturados que possuem o encapsulamento de protocolo ao qual a entrada representa.
- Octets – corresponde o objeto *protocolDistStatsOctets* e guarda a quantidade de octetos capturados que possuem o encapsulamento de protocolo ao qual a entrada representa.

4.3.3 Grupo *Network-Layer Host*

Para atender as tabelas pertencentes ao grupo *network layer host*, foram criadas duas tabelas. Uma delas é a tabela "hlhostcontrol" que representa a tabela *hlHostControlTable* e a outra é a tabela "nlhost" que referencia a tabela *nlHostTable*.

Os campos pertencentes à tabela "hlhostcontrol" com seus respectivos atributos podem ser vistos na tabela 4.6.

TABELA 4.6 – Tabela "hlhostcontrol"

Nome	Tipo			Chave primária
Indice	tinyint(3)	unsigned	NOT NULL	*
DataSource	varchar(10)		NULL	
NlDroppedFrames	int(10)	unsigned	NULL	
NlInserts	int(10)	unsigned	NULL	
NlDeletes	int(10)	unsigned	NULL	
NlMaxDesiredEntries	smallint(5)	unsigned	NULL	
AlDroppedFrames	int(10)	unsigned	NULL	
AlInserts	int(10)	unsigned	NULL	
AlDeletes	int(10)	unsigned	NULL	
AlMaxDesiredEntries	smallint(5)	unsigned	NULL	
Owner	varchar(30)		NULL	
Status	tinyint(3)	unsigned	NULL	

Cada campo da tabela tem as seguintes características:

- Indice – assim como na tabela "protocoldistcontrol", o campo é usado como chave primária e ao mesmo tempo chave estrangeira. Seu valor também segue o mesmo princípio: diferenciar cada interface que está capturando pacotes. O campo corresponde ao objeto *hlhostControlIndex*;
- DataSource – segue as características do campo "DataSource" da tabela "protocoldistcontrol". Representa o objeto *hlhostControlDataSource*;
- NlDroppedFrames – representa o objeto *hlhostControlNlDroppedFrames* e armazena o mesmo valor do objeto *protocolDistControlDroppedFrames*;
- NlInserts – número de inserções feitas na tabela "nlhost". Representa o objeto *hlhostControlNlInserts*;
- NlDeletes – número de entradas excluídas na tabela "nlhost". Representa o objeto *hlhostControlNlDeletes*;
- NlMaxDesiredEntries – número máximo de entradas na tabela "nlhost". Representa o objeto *hlhostControlNlMaxDesiredEntries*;
- AlDroppedFrames – representa o objeto *hlhostControlAlDroppedFrames* e armazena o mesmo valor que o objeto *protocolDistControlDroppedFrames*;
- AlInserts – número de inserções feitas na tabela "alhost". Representa o objeto *hlhostControlAlInserts*;
- AlDeletes – número de entradas excluídas na tabela "alhost". Representa o objeto *hlhostControlAlDeletes*;
- AlMaxDesiredEntries – número máximo de entradas na tabela "alhost". Representa o objeto *hlhostControlAlMaxDesiredEntries*;
- Owner – corresponde ao objeto *hlhostControlOwner*;
- Status – corresponde ao objeto *hlhostControlStauts* e pode ter os mesmos valores do objeto "Status" da tabela "protocoldir".

A estrutura da tabela “nlhost” pode ser visualizada na tabela 4.7.

TABELA 4.7 – Tabela “nlhost”

Nome	Tipo				Chave Primária
TimeMark	bigint(20)	unsigned	NOT NULL		
Address	int(10)		NOT NULL		
InPkts	int(10)	unsigned	NULL		
OutPkts	int(10)	unsigned	NULL		
InOctets	bigint(20)	unsigned	NULL		
OutOctets	bigint(20)	unsigned	NULL		
OutMacNonUnicastPkts	int(10)	unsigned	NULL		
CreateTime	bigint(20)	unsigned	NULL		
LocalIndex	smallint(5)	unsigned	NOT NULL		
ControlIndex	tinyint(3)	unsigned	NOT NULL		
Sequence	int(10)	unsigned	NOT NULL	Auto_increment	*

Os campos da tabela têm as seguintes características:

- TimeMark – corresponde ao instante de tempo em que a entrada foi criada ou pela última vez atualizada. O campo representa o objeto *nlhostTimeMark* e faz parte do índice das entradas;
- Address – identifica o endereço IP de uma estação sendo que o mesmo é expresso em notação decimal. O campo representa o objeto *nlhostAddress* e é usado para indexar as entradas;
- InPkts – quantidade de pacotes que a estação recebeu. Representa o objeto *nlhostInPkts*;
- OutPkts – quantidade de pacotes que a estação transmitiu. Representa o objeto *nlhostOutPkts*;
- InOctets – quantidade de octetos que a estação recebeu. Representa o objeto *nlhostInOctets*;
- OutOctets – quantidade de octetos que a estação transmitiu. Representa o objeto *nlhostOutOctets*;
- OutMacNonUnicastPkts – número de pacotes não *unicast* transmitidos pela estação. Representa o objeto *nlhostOutMacNonUnicastPkts*;
- CreateTime – corresponde ao instante de tempo em que a entrada foi criada. Esse campo representa o objeto *nlhostCreateTime* e é semelhante ao campo “CreateTime” da tabela “protocoldistcontrol”;
- LocalIndex – esse campo é uma chave estrangeira do campo “LocalIndex” da tabela “protocoldir” e o mesmo faz parte do índice para as entradas da tabela;
- ControlIndex – representa o objeto *nlhostControlIndex* e constitui também uma chave estrangeira do campo “Indice” da tabela “hlhostcontrol”. O campo compõe o índice para as entradas;
- Sequence – campo destinado para uso interno. Sua função é ordenar as tuplas da tabela conforme elas são inseridas. O campo constitui a chave primária da tabela.

4.3.4 Grupo *Application-Layer Host*

Esse grupo possui apenas a tabela *alHostTable*. Para representar a tabela, criou-se a tabela “alhost”. A especificação da tabela pode ser vista na tabela 4.8.

A tabela possui apenas uma chave primária compreendida pelo campo “Sequence” e suas características são as mesmas do campo “Sequence” da tabela “nlhost”. Os campos “LocalIndexNet”, “LocalIndexApp” e “ControlIndex” são chaves estrangeiras, sendo que os dois primeiros referenciam tuplas da tabela “protocoldir” (“LocalIndexNet” referencia encapsulamentos de protocolo de rede e “LocalIndexApp” referencia encapsulamentos de protocolo de aplicação) e a última referencia entradas da tabela “hlhostcontrol”. Os demais campos seguem os mesmos princípios da tabela “nlhost”, no entanto, é claro que esses campos representam os objetos da tabela *alHostTable*.

TABELA 4.8 – Tabela “alhost”

Nome	Tipo				Chave Primária
TimeMark	bigint(20)	unsigned	NOT NULL		
InPkts	int(11)	unsigned	NULL		
OutPkts	int(11)	unsigned	NULL		
InOctets	bigint(20)	unsigned	NULL		
OutOctets	bigint(20)	unsigned	NULL		
CreateTime	bigint(20)	unsigned	NULL		
LocalIndexNet	smallint(5)	unsigned	NOT NULL		
LocalIndexApp	smallint(5)	unsigned	NOT NULL		
ControlIndex	tinyint(3)	unsigned	NOT NULL		
Address	bigint(20)		NOT NULL		
Sequence	int(10)	unsigned	NOT NULL	Auto_increment	*

4.3.5 Grupo *Network-Layer Matrix*

Para atender o grupo *network layer matrix*, foram criadas cinco tabelas. Uma chamada “hlmatrixcontrol” que representa a tabela *hlMatrixControlTable*, outras duas chamadas “nlmatrixsd” e “nlmatrixds” que representam respectivamente as tabelas *nlMatrixSDTable* e *nlMatrixDSTable*, finalizando com as tabelas “nlmatrixtopncontrol” e “nlmatrixtopn” que representam as tabelas *nlMatrixTopNControlTable* e *nlMatrixTopNTable*.

A tabela “hlmatrixcontrol” é idêntica a tabela “hlhostcontrol” (vide tabela 4.6). Já as tabelas “nlmatrixsd” e “nlmatrixds” possuem os mesmos campos e características, no entanto, a única diferença está na ordem dos campos *Saddress* e *Daddress*. Na tabela “nlmatrixsd” (vide tabela 4.9), *Saddress* antecede o campo *Daddress*, já na tabela “nlmatrixds” (vide tabela 4.10) a ordem é inversa.

TABELA 4.9 – Tabela “nlmatrixsd”

Nome	Tipo				Chave Primária
MatrixControl	tinyint(3)	unsigned	NOT NULL		
LocalIndex	smallint(5)	unsigned	NOT NULL		
TimeMark	bigint(20)	unsigned	NOT NULL		
Saddress	bigint(20)		NOT NULL		
Daddress	bigint(20)		NOT NULL		
Pkts	int(10)	unsigned	NULL		
Octets	bigint(20)	unsigned	NULL		
CreateTime	bigint(20)	unsigned	NULL		
Sequence	int(10)	unsigned	NOT NULL	Auto_increment	*

Os campos da tabela “nlmatrixsd” têm suas especificações conforme expostas:

- MatrixControl – representa o objeto *nlmatrixControlIndex* e constitui também uma chave estrangeira do campo “Indice” da tabela “hlmatrixcontrol”. O campo é usado para indexar as entradas;
- LocalIndex – esse campo é uma chave estrangeira do campo “LocalIndex” da tabela “protocoldir” e faz parte do índice das entradas da tabela;
- TimeMark – corresponde ao instante de tempo em que a entrada foi criada ou pela última vez atualizada. O campo representa o objeto *nlmatrixsdTimeMark* e é usado para indexar as entradas;
- Saddress – identifica o endereço IP da estação origem do pacote em notação decimal. O campo representa o objeto *nlmatrixsdSourceAddress* e é usado para indexar as entradas;
- Daddress – identifica o endereço IP da estação destino do pacote em notação decimal. O campo representa o objeto *nlmatrixsdDestAddress* e é usado para indexar as entradas;
- Pkts – quantidade de pacotes contabilizados para o par de estações. Representa o objeto *nlmatrixsdPkts*;
- Octets – quantidade de octetos contabilizados para o par de estações. Representa o objeto *nlmatrixsdOctets*;
- CreateTime – corresponde ao instante de tempo em que a entrada foi criada. Esse campo representa o objeto *nlmatrixsdCreateTime* e é semelhante ao campo “CreateTime” da tabela “nlhost”;
- Sequence – campo destinado para uso interno. Sua função é ordenar as tuplas da tabela conforme elas são inseridas. O campo constitui a chave primária da tabela.

TABELA 4.10 – Tabela “nlmatrixds”

Nome	Tipo				Chave Primária
MatrixControl	tinyint(3)	unsigned	NOT NULL		
LocalIndex	smallint(5)	unsigned	NOT NULL		
TimeMark	bigint(20)	unsigned	NOT NULL		
Daddress	int(11)		NOT NULL		
Saddress	int(11)		NOT NULL		
Pkts	int(10)	unsigned	NULL		
Octets	bigint(20)	unsigned	NULL		
CreateTime	bigint(20)	unsigned	NULL		
Sequence	int(10)	unsigned	NOT NULL	Auto_increment	*

A tabela “nlmatrixtopncontrol” (vide tabela 4.11) tem poucas semelhanças com as tabelas anteriormente mencionadas. Quando se fala em poucas semelhanças, quer dizer, muitos dos campos presentes na tabela são distintos para essas tabelas.

TABELA 4.11 – Tabela “nlmatrixtopncontrol”

Nome	Tipo			Chave Primária
Indice	int(10)	unsigned	NOT NULL	*
MatrixIndex	tinyint(3)	unsigned	NULL	
RateBase	tinyint(3)	unsigned	NULL	
TimeRemaining	int(10)	unsigned	NULL	
GeneratedReports	int(10)	unsigned	NULL	
Duration	int(10)	unsigned	NULL	
RequestedSize	smallint(5)	unsigned	NULL	
GrantedSize	smallint(5)	unsigned	NULL	
StartTime	bigint(20)	unsigned	NULL	
Owner	varchar(30)		NULL	
Status	tinyint(3)	unsigned	NULL	

As especificações dos campos da tabela podem ser assim descritas:

- Indice – representa o objeto *nlMatrixTopNControlIndex*. Esse objeto identifica cada entrada da tabela, portanto é a chave primária da mesma. Cada entrada, como descrito na definição da MIB RMON2 é uma forma de ordenação das entradas da tabela *TopN* atrelada a essa tabela de controle;
- MatrixIndex – representa o objeto *nlMatrixTopNControlMatrixIndex*. Esse campo é uma chave estrangeira para o objeto “Indice” da tabela “hlmatrixcontrol”;
- RateBase – representa o objeto *nlmatrixTopNControlRateBase*. Os valores possíveis para esse campo são: um (1) quando o tipo de ordenação é feito pelo número de pacotes ou dois (2) quando o tipo de ordenação é feito pelo número de octetos;
- TimeRemaining – representa o objeto *nlmatrixTopNControlTimeRemaining*;
- GeneratedReports – representa o objeto *nlmatrixTopNControlGeneratedReports*
- Duraction – representa o objeto *nlmatrixTopNControlDuraction;*
- RequestedSize – representa o objeto *nlmatrixTopNControlRequestedSize;*
- GrantedSize – representa o objeto *nlmatrixTopNControlGrantedSize;*
- StartTime – representa o objeto *nlmatrixTopNControlStartTime;*
- Owner – representa o objeto *nlmatrixTopNControlOwner;*
- Status – representa o objeto *nlmatrixTopNControlStatus* e pode ter seis valores diferentes. Os valores possíveis são os mesmos do objeto “Status” da tabela “protocoldir”.

A estrutura da tabela “nlmatrixtopn” pode ser vista na tabela 4.12. Já os seus campos possuem as seguintes especificações:

- TopNControlIndex – representa o objeto *nlmatrixTopNControlIndex*. Esse campo além de ser uma das chaves primárias da tabela é um chave estrangeira do campo “Indice” da tabela “nlmatrixTopNControl”;
- Indice – representa o objeto *nlmatrixTopNIndex* e constitui também a chave primária da tabela;
- LocalIndexNet – representa o objeto *nlmatrixTopNProtocolDirLocalIndex*. O campo é uma chave estrangeira do campo *LocalIndex* da tabela “protocolDir”;
- SAddress – representa o objeto *nlmatrixTopNSourceAddress*;
- DAddress – representa o objeto *nlmatrixTopNDestAddress*;
- PktRate – representa o objeto *nlmatrixTopNPktRate*;
- ReversePktRate – representa o objeto *nlmatrixTopNReversePktRate*;

- OctetRate – representa o objeto *nlmatrixTopNOctetRate*;
- ReverseOctetRate – representa o objeto *almatrixTopNReverseOctetRate*.

TABELA 4.12 – Tabela “nlmatrixtopn”

Nome	Tipo			Chave Primária
TopNControlIndex	smallint(5)	unsigned	NOT NULL	*
Indice	smallint(5)	unsigned	NOT NULL	*
LocalIndexNet	smallint(5)	unsigned	NULL	
Saddress	bigint(20)		NULL	
Daddress	bigint(20)		NULL	
PktRate	int(10)	unsigned	NULL	
ReversePktRate	int(10)	unsigned	NULL	
OctetRate	bigint(20)	unsigned	NULL	
ReverseOctetRate	bigint(20)	unsigned	NULL	

4.3.6 Grupo *Application-Layer Matrix*

Para atender as quatro tabelas do grupo *application-layer matrix*, foram criadas respectivamente quatro tabelas. A tabela *alMatrixSDTable* é representada pela tabela “almatrixsd”, a *alMatrixDSTable* pela “almatrixds”, a *alMatrixTopNControlTable* pela “almatrixtopncontrol” e finalizando, a tabela *alMatrixTopNTable* é representada pela tabela “almatrixtopn”.

As tabelas “almatrixsd” e “almatrixds” possuem os mesmos campos e características, no entanto, a única diferença está na ordem dos campos *Saddress* e *Daddress*. Na tabela “almatrixsd” (vide tabela 4.13), *Saddress* antecede o campo *Daddress*, já na tabela “almatrixds” (vide tabela 4.14) a ordem é inversa.

TABELA 4.13 – Tabela “almatrixsd”

Nome	Tipo				Chave Primária
MatrixControl	tinyint(3)	unsigned	NOT NULL		
LocalIndexNet	smallint(5)	unsigned	NOT NULL		
LocalIndexApp	smallint(5)	unsigned	NOT NULL		
TimeMark	bigint(20)	unsigned	NOT NULL		
SAddress	bigint(20)		NOT NULL		
DAddress	bigint(20)		NOT NULL		
Pkts	int(10)	unsigned	NULL		
Octets	bigint(20)	unsigned	NULL		
CreateTime	bigint(20)	unsigned	NULL		
Sequence	int(10)	unsigned	NOT NULL	Auto_increment	*

Os campos das tabelas “almatrixsd” e “almatrixds” são semelhantes às tabelas “nlmatrixsd” e “nlmatrixds”. A diferença está na existência de um campo a mais nas tabelas “almatrixsd” e “almatrixds” que simboliza o encapsulamento de protocolo da camada superior (*LocalIndexApp*). Tal campo é uma chave estrangeira do campo *LocalIndex* da tabela “protocoldir” e o mesmo é usado para fins de indexação das entradas das tabelas.

TABELA 4.14 – Tabela “almatrixds”

Nome	Tipo				Chave Primária
MatrixControl	tinyint(3)	unsigned	NOT NULL		
LocalIndexNet	smallint(5)	unsigned	NOT NULL		
LocalIndexApp	smallint(5)	unsigned	NOT NULL		
TimeMark	bigint(20)	unsigned	NOT NULL		
DAddress	bigint(20)		NOT NULL		
SAddress	bigint(20)		NOT NULL		
Pkts	int(10)	unsigned	NULL		
Octets	bigint(20)	unsigned	NULL		
CreateTime	bigint(20)	unsigned	NULL		
Sequence	int(10)	unsigned	NOT NULL	Auto_increment	*

A tabela “almatrixtopncontrol” é idêntica a tabela “nlmatrixtopncontrol”. As diferenças estão nos possíveis valores para o campo *Ratebase* (podendo variar de 1 até 4). Já a tabela “almatrixtopn” (vide figura 4.15) apresenta o campo “LocalIndexApp” a mais quando comparada a tabela com a tabela “nlmatrixtopn”. Esse campo serve para referenciar um encapsulamento de protocolo de aplicação.

TABELA 4.15 – Tabela “almatrixtopn”

Nome	Tipo			Chave Primária
TopNControlIndex	tinyint(3)	unsigned	NOT NULL	*
Indice	smallint(5)	unsigned	NOT NULL	*
LocalIndexNet	smallint(5)	unsigned	NULL	
LocalIndexApp	smallint(5)	unsigned	NULL	
SAddress	bigint(20)	unsigned	NULL	
DAddress	bigint(20)	unsigned	NULL	
PktRate	int(10)	unsigned	NULL	
ReversePktRate	int(10)	unsigned	NULL	
OctetRate	bigint(20)	unsigned	NULL	
ReverseOctetRate	bigint(20)	unsigned	NULL	

4.4 Módulo Conversor

O módulo “conversor” é um programa executável mais complexo quando comparado com o módulo “capturador”. Nele são feitas todas as consistências necessárias para se popular as tabelas da base de dados da MIB RMON2. Para a implementação do módulo foi utilizada a API para a conexão com o banco de dados `MySql`.

Ao inicializar o módulo, uma conexão com a base de dados da MIB RMON2 (vide fluxo 5 da figura 4.2) é aberta com o intuito de eliminar todos os registros de todas as tabelas que conterão as estatísticas (*protocoldiststats*, *nlhost*, *alhost*, *nlmatrixsd*, *nlmatrixds*, *almatrixsd*, *almatrixds*, *nlmatrixtopn* e *almatrixtopn*). Após esse passo preliminar, várias *threads* são disparadas sendo que cada uma é responsável por uma tarefa. Sendo mais preciso, o número máximo de *threads* disparadas pode chegar a sete.

Uma das *threads* é disparada automaticamente sendo sua responsabilidade obter informações da base de dados dos pacotes capturados e, a partir desses dados gerar entradas para as tabelas da base de dados da MIB RMON2. Já as tabelas *TopN*, as quais não são populadas pela *thread* acima relatada, receberão dados de outras *threads*. As

tabelas *TopN* do grupo *nlmatrix* poderão ser populadas por, no máximo, duas *threads* e as tabelas *TopN* do grupo *almatrix* poderão ser populadas pelas quatro *threads* restantes.

O motivo de ter duas *threads* para atender o grupo *nlmatrix* é que como o agente captura apenas de uma interface e as tabelas *TopN* do grupo em questão podem ser ordenadas pelo número de pacotes e/ou pelo número de octetos, logo cada *thread* fica responsável por uma das formas de ordenação.

Já o grupo *almatrix*, pode ser ordenado pelo número de pacotes terminais (apenas protocolos de aplicação), número de pacotes, número de octetos terminais e/ou número de octetos, conforme descrito na subseção 2.7.2 do capítulo da MIB RMON2. Por isso podem ser disparadas até quatro *threads*, uma para cada forma de ordenação.

Nas subseções seguintes, serão descritas a fase da conversão dos pacotes capturados para as tabelas da MIB RMON2 e a forma como as tabelas *TopN* são populadas.

4.4.1 Populando as tabelas da MIB RMON2

Toda a tarefa de popular as tabelas de dados da MIB RMON2, exceto as *topN*, a partir dos pacotes capturados é feita por uma rotina que, a cada um minuto, inicia um ciclo de processamento e essa, por sua vez, está anexada à *thread* dedicada a este fim.

Ao iniciar um ciclo de processamento dos pacotes capturados, faz-se a abertura de uma conexão com a base de dados dos pacotes capturados (vide fluxo 4 da figura 4.2) e outra com a base de dados da MIB RMON2 (vide fluxo 5 da figura 4.2). Caso uma ou as duas conexões falhem o ciclo não fará a contabilização dos pacotes.

Após a fase de estabelecimento da conexão com as bases de dados, uma varredura na tabela “protocoldir”, pertencente a base de dados da MIB RMON2, é feita para se descobrir quais os protocolos que estão ativos, isto é, o campo “status” do encapsulamento de protocolo estiver no estado *active* (1). Os encapsulamentos de protocolo que passarem no teste serão armazenados em um vetor com uma estrutura que guarda para cada encapsulamento os seguintes parâmetros:

- Protocolo de enlace;
- Protocolo de rede;
- Protocolo de transporte;
- Protocolo de aplicação;
- Valor do campo *protocolDirAddressMapConfig;*
- Valor do campo *protocolDirHostConfig*
- Valor do campo *protocolDirMatrixConfig;*
- Valor aleatório atribuído ao protocolo dentro da tabela “protocoldir” que é obtido no campo *LocalIndex*.

Os encapsulamentos de protocolo presentes nessa estrutura representam os protocolos que o agente está monitorando atualmente e, para tanto, poderão ser geradas contabilizações nas tabelas da MIB caso algum pacote capturado o utilize.

Para exemplificar esta fase, a tabela 4.16 mostra quatro entradas com os respectivos valores dos campos para um encapsulamento de protocolo da camada de enlace (*Ethernet*), outro para a camada de rede (IP), outro para a de transporte (UDP) e por fim uma para a de aplicação (SNMP).

TABELA 4.16 – Exemplo da estrutura que contém os encapsulamentos de protocolo ativos da tabela “protocoldir”

Prot. enlace	Prot. rede	Prot. transporte	Prot. aplicação	AddressMap Config	HostConfig	MatrixConfig	LocalIndex
1	0	0	0	1	1	1	1
1	2048	0	0	1	3	3	2
1	2048	17	0	1	3	3	3
1	2048	17	161	1	3	3	4

Como podem ser observados na tabela 4.16, os campos que representam os protocolos das camadas superiores ao encapsulamento de protocolo da entrada recebem o valor zero (0). Por exemplo, a terceira entrada da estrutura representa o protocolo de transporte UDP que usa o protocolo de enlace *Ethernet* (1), o protocolo de rede IP (2048). O protocolo da camada superior (aplicação), por não existir, recebe o valor zero (0).

A próxima verificação diz respeito à entrada que corresponde à interface de captura nas tabelas de controle “protocoldistcontrol”, “hlhostcontrol” e “hlmatrixcontrol”. O valor do campo “status” de cada entrada dessas tabelas devem estar no estado *active* (1) para que as tabelas de dados relacionadas à tabela de controle possam receber dados da base dos pacotes capturados. Para fins de exemplificação, caso a entrada da tabela de controle “protocoldistcontrol”, a qual representa a interface de captura de pacotes, estiver com o campo status igual a *active* (1), conseqüentemente, a tabela de dados “protocoldiststats” poderá receber dados da base de dados dos pacotes capturados.

Posteriormente adquire-se uma amostra dos pacotes capturados da tabela “conexao” pertencente à base dos pacotes capturados. Essa amostra compreende todas as entradas da tabela até o instante da requisição. Para cada pacote todos os campos são obtidos (endereços IP origem e destino, portas origem de destino, instante de tempo em que o pacote foi capturado, tamanho do pacote em octetos, protocolo de enlace, protocolo de rede, protocolo de transporte e a interface em que o pacote foi capturado).

Ao analisar o pacote, os protocolos de suas camadas são comparados com as entradas da estrutura que possui os encapsulamentos de protocolos ativos com a finalidade de verificar se os protocolos do pacote estão presentes, logo podendo ser geradas estatísticas sobre eles.

Analisando do ponto de vista do protocolo de rede, se o mesmo for encontrado no vetor dos ativos (ter uma entrada que representa o encapsulamento de protocolo de rede que possua como protocolo de enlace o mesmo do pacote), o pacote poderá ser contabilizado nas tabelas “protocoldiststats”, “nlhost”, “nlmatrixsd” e “nlmatrixds” caso passe por algumas verificações.

A tabela “protocoldiststats” poderá receber dados se a entrada da tabela de controle (“protocoldistcontrol”) que representa a interface em que o pacote foi capturado ter o valor do campo status igual a *active* (1). Para visualizar os passos seguidos pelo processo, veja o fluxograma da figura 4.4.

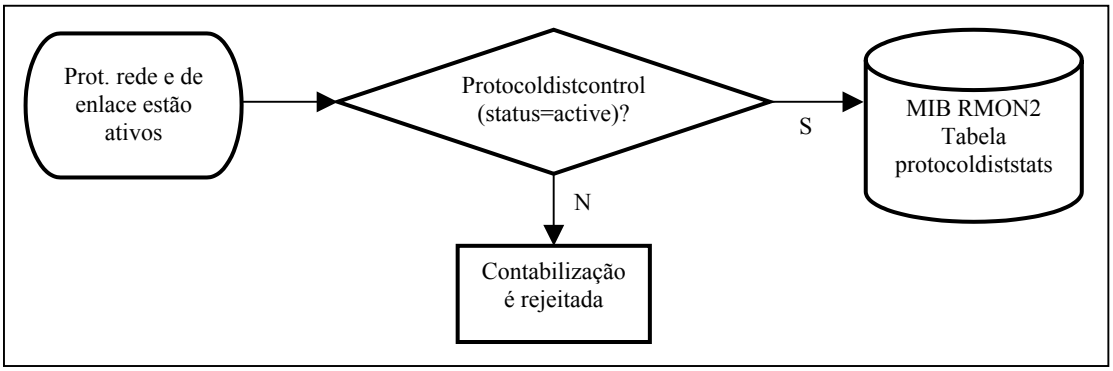


FIGURA 4.4 – Fluxograma representando o processo para insersão de dados na tabela "protocoldiststats"

Já a tabela "nlhost" poderá receber a contabilização do pacote se o campo "HostConfig" da entrada que representa o encapsulamento de protocolo de rede da estrutura dos encapsulamentos ativos tiver o valor *supportedOn* (3) e a entrada da tabela de controle ("hlhostcontrol") que referencia a interface em que o pacote foi capturado ter o valor do campo status igual a *active* (1). O fluxograma representando tal processo pode ser visto na figura 4.5.

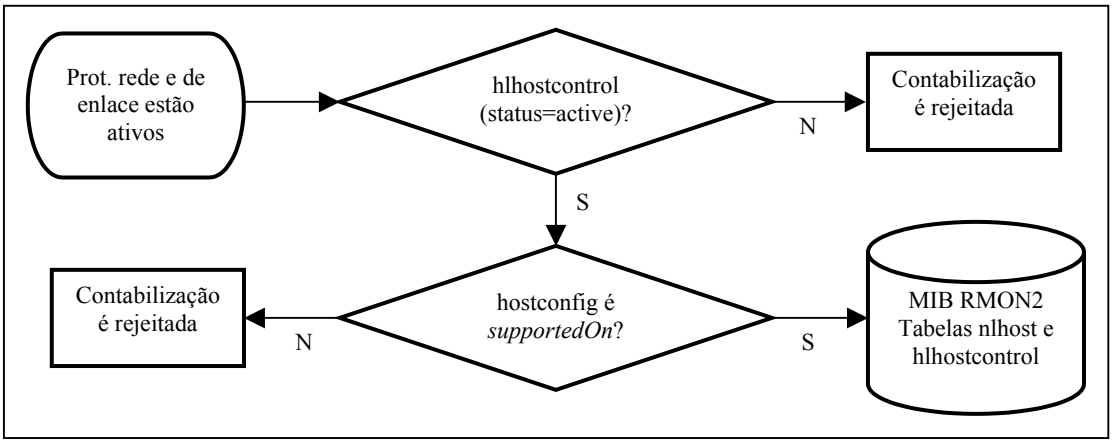


FIGURA 4.5 – Fluxograma representando o processo para insersão de dados na tabela "nlhost"

As tabelas "nlmatrixsd" e "nlmatrixds", por sua vez, receberão a contabilização caso o campo "MatrixConfig" da entrada que representa o encapsulamento de protocolo de rede da estrutura dos encapsulamentos ativos tiver o valor *supportedOn* (3) e a entrada da tabela de controle ("hlmatrixcontrol") que referencia a interface que o pacote foi capturado tiver o valor do campo status igual a *active* (1). O fluxograma da figura 4.6 demostra os passos do processo.

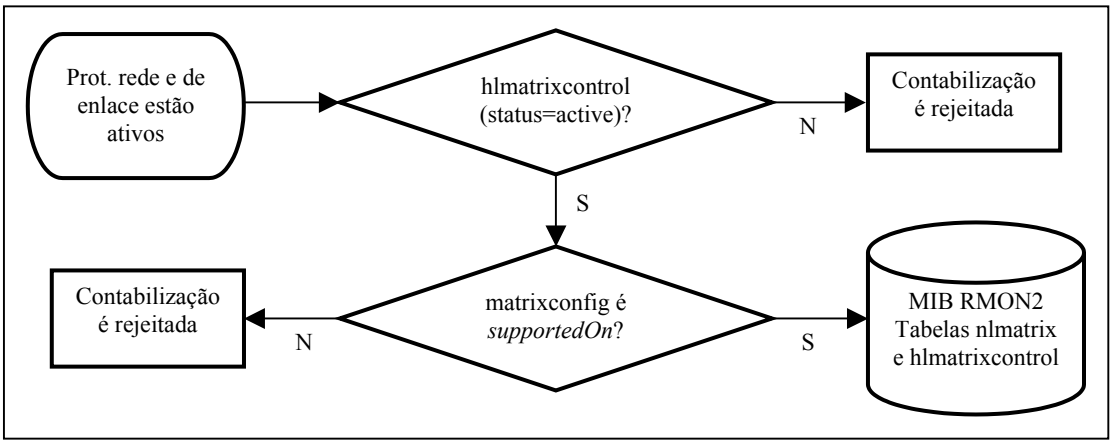


FIGURA 4.6 – Fluxograma representando o processo para insersão de dados nas tabelas "nlmatrixsd" e "nlmatrixds"

Passando pelas condições anteriores, no momento de popular as tabelas, mais alguns detalhes devem ser observados. Para a tabela "protocoldiststats", se existir uma entrada para o encapsulamento de protocolo de rede e interface de captura em questão, os campos "Pkts" e "Octets" serão atualizados, senão a nova entrada será inserida na tabela.

A tabela "nlhost" é, no entanto, um pouco mais complicada. Para cada pacote verifica-se a existência de uma entrada para o endereço IP origem, encapsulamento de protocolo de rede e interface em que foi capturado. Existindo, a entrada é atualizada (campos "OutPackets", "OutOctets" e "TimeMark"), caso contrário, a entrada é inserida contabilizando os mesmos campos da atualização. Os passos seguidos para o endereço origem são realizados para o endereço destino. No entanto, para o endereço destino os campos que recebem alterações são: "InPackets", "InOctets" e "TimeMark". Uma vez que o pacote não é *unicast*, a contabilização só será feita para o endereço origem e o campo "MacNonUnicast" dessa entrada é incrementado em um. Além desses procedimentos, se a entrada foi inserida na tabela "nlhost" então se deve incrementar o valor do campo "NlInserts" da entrada da tabela de controle associada a inserida na tabela "nlhost". Caso a quantidade de entradas da tabela for igual ao valor do campo "NlMaxDesiredEntries" então exclui-se a primeira entrada da tabela "nlhost" e incrementa o valor do campos "NlDeletes" (pelo motivo da exclusão) e "NlInserts" (pelo motivo da nova inserção).

Finalizando, as tabelas do grupo *nlmatrix* ("nlmatrixsd" e "nlmatrixds") passam pelos seguintes passos para a contabilização se o pacote for *unicast*: verifica a existência de uma entrada na tabela "nlmatrixsd" que contenha o endereço origem e destino do pacote, o encapsulamento de protocolo de rede e a interface na qual o pacote foi capturado. Se existir, atualiza-se a entrada através da contabilização dos campos "Pkts", "Octets" e "TimeMark". Falhando a pesquisa, a entrada deve ser inserida. Para a tabela "nlmatrixds" os mesmos passos são feitos mudando apenas a precedência dos campos de endereço IP origem e destino.

Quando uma inserção for feita na tabela "nlmatrixsd", certamente, será inserida uma na "nlmatrixds", logo o campo "NlInserts" da tabela de controle associada às entradas das tabelas recebe um incremento de dois. Caso a quantidade de entradas das tabelas "nlmatrixsd" e "nlmatrixds" (as duas tabelas sempre terão o mesmo número de entradas) for igual ao valor do campo *NlMaxDesiredEntries*, exclui-se a primeira

entrada de cada tabela (“nlmatrixsd” e “nlmatrixds”) e se incrementa, respectivamente, o valor do campo “NlDeletes” e “NlInserts” em dois.

Mudando o enfoque, o modo que as tabelas de aplicação (“alhost”, “almatrixsd” e “almatrixds”) e a tabela “protocoldiststats” são populadas para os protocolos de alto nível (protocolos acima da camada de rede) será descrito.

Para inserir dados nessas tabelas, obrigatoriamente, o encapsulamento de protocolo de rede deve estar habilitado. Além disso, o encapsulamento de protocolo de aplicação para ser contabilizado deve ter o encapsulamento de protocolo de transporte ativo.

A inserção de dados na tabela “alhost” é bastante semelhante à forma de como é feita na tabela “nlhost”. A diferença está no momento de verificar se os dados de um pacote já foram inseridos na tabela. Nesse caso, acrescenta-se ao índice o encapsulamento de protocolo de alto nível.

As tabelas “almatrixsd” e “almatrixds” também seguem o mesmo procedimento das tabelas *nlmatrix* diferenciando apenas no momento da inserção que, neste caso, verifica a existência de uma entrada que contenha os mesmos campos usados para verificar a existência de uma entrada nas tabelas *nlmatrix* acrescido do encapsulamento de protocolo de alto nível.

O procedimento para popular as tabelas da MIB RMON2 não é muito simples. Para facilitar a compreensão e/ou ter uma visão mais geral do processo, o fluxograma da figura 4.7 é apresentado.

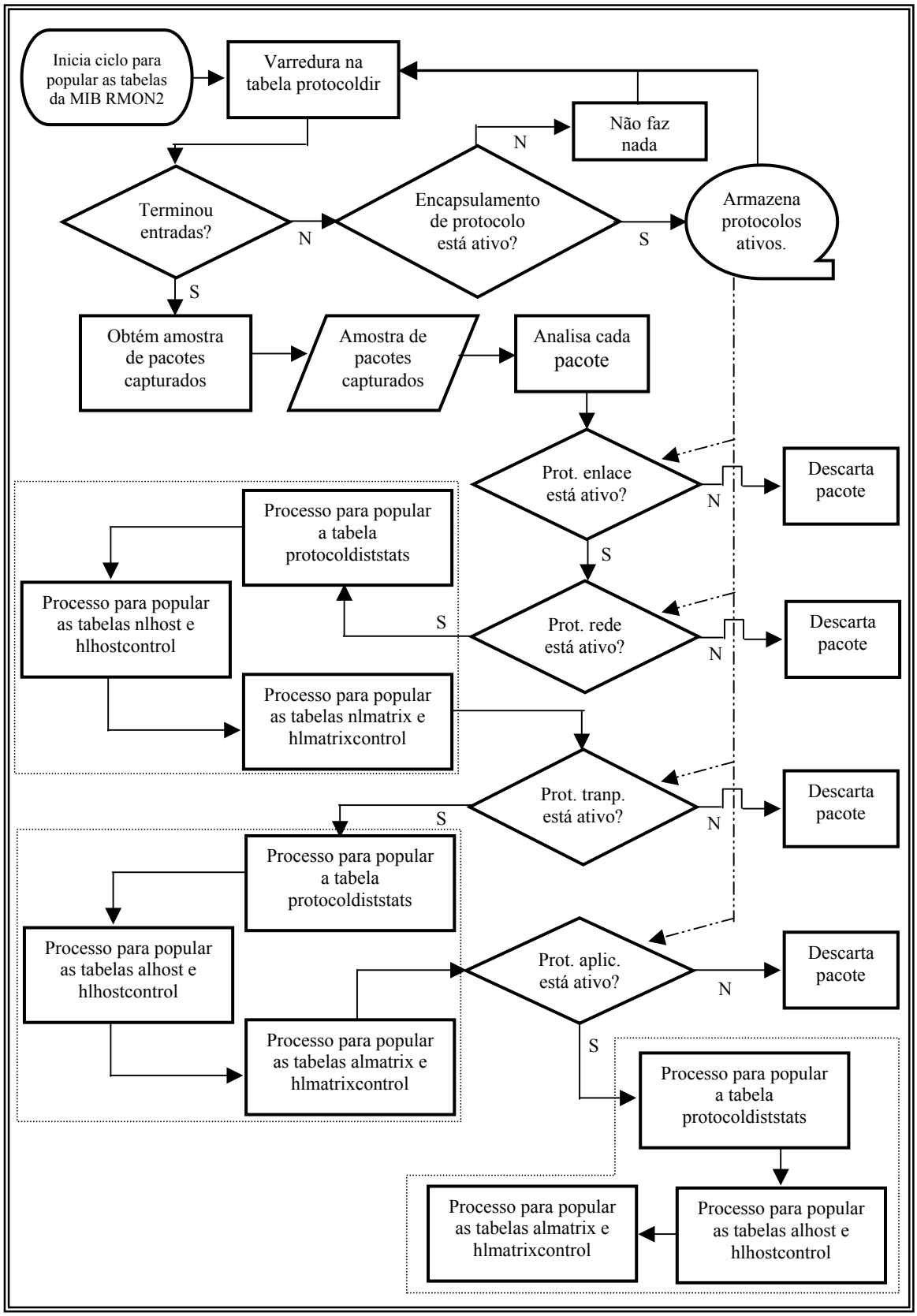


FIGURA 4.7 – Fluxograma do processo de inserção de dados nas tabelas da MIB RMON2, exceto para as tabelas *TopN*

4.4.2 Populando as tabelas *matrixTopN*

As tabelas *TopN* têm uma forma peculiar de serem populadas quando comparadas com as demais tabelas da MIB RMON2. A tabela "nlmatrixtopn" recebe dados da tabela "nlmatrixsd". Já a tabela "almatrixtopn", tem como fonte de dados a tabela "almatrixsd".

O primeiro passo antes de começar a popular as tabelas *TopN* é descobrir quantas entradas suas tabelas de controle possuem. Recordando, cada entrada das tabelas representam uma forma de contabilização para a respectiva tabela de dados. Por exemplo, caso a tabela de controle "nlmatrixtopncontrol" possua duas entradas, conseqüentemente são disparadas duas *threads* com a função de popular a tabela "nlmatrixtopn". Cada *thread* referencia uma das entradas da tabela de controle. O procedimento realizado para a insersão de dados na tabela "nlmatrixtopn" é, respectivamente, realizado para a tabela "almatrixtopn".

A *thread*, ao ser inicializada, abre uma conexão com a base de dados "rmon2" e requisita à tabela de controle o tipo de ordenação, o tempo entre contabilizações e a quantidade de entradas possíveis para tal na tabela de dados *TopN*.

Após a *thread* obter os parâmetros básicos para desempenhar a sua função e o intervalo de tempo oriundo do parâmetro tempo entre contabilizações ter transcorrido, começa a fase de inserção de dados. A primeira tarefa a ser realizada é eliminar todas as entradas da tabela *TopN* que referenciam a entrada da tabela de controle ao qual a *thread* está vinculada e, só após esse procedimento, a tabela começa a ser populada.

Para popular a tabela "nlmatrixtopn", uma varredura é feita na tabela "nlmatrixsd" e obtém-se para cada entrada da tabela o endereço origem e destino, o protocolo de rede e o campo referente ao tipo de ordenação (número de pacotes ou número de octetos).

A tabela "nlmatrixsd" pode possuir uma única entrada para um par de máquinas que realizaram ou estão realizando uma comunicação em um sentido (origem – destino), e da mesma forma, só pode existir uma entrada na tabela para o respectivo par de máquinas no sentido contrário (destino – origem), conforme pode ser visto na tabela 4.17 (entradas 1 e 2 ou 3 e 4). Por esse fator, verifica-se primeiramente a existência na tabela "nlmatrixtopn" de uma entrada que tenha como endereço fonte o endereço destino da entrada e como endereço destino o endereço fonte da entrada. A resposta da pesquisa pode ser a ausência da entrada e, neste caso, a entrada que se está analisando é inserida à tabela, caso contrário, a entrada descoberta receberá a contabilização para o campo *Reverse* da respectiva ordenação (vide figura 4.18).

TABELA 4.17 – Exemplo de entradas na tabela "nlmatrixsd"

Matrix Control	Local Index	TimeMark	Saddress	DAddress	Pkts	Octets	Create Time
1	1	2345	10.25.110.3	10.25.110.11	9	7132	1000
1	1	2347	10.25.110.11	10.25.110.3	10	13350	1002
1	1	4968	10.25.110.3	200.234.54.10	50	25787	302
1	1	5070	200.234.54.10	10.25.111.3	55	54231	307
...	…	…	…	…	…	…	…

TABELA 4.18 – Tabela “nlmatrixtopn” gerada a partir da tabela “nlmatrixsd” ordenada pelo número de pacotes

TopN Control Index	Indice	Local Index	SAddress	Daddress	PktRate	Reverse PktRate	Octet Rate	Reverse OctetRate
1	1	1	10.25.110.3	10.25.110.11	9	10	0	0
1	2	1	10.25.110.3	200.234.54.10	50	60	0	0
1	3	1	200.234.54.10	10.25.111.3	55	52	0	0
...	...	...	...	...	...	...	0	0

Como mencionado anteriormente, se uma entrada não foi encontrada na tabela *TopN*, então ela seria inserida na mesma. Essa afirmação vale apenas se a quantidade máxima de entradas da tabela não tiver sido alcançada. Caso contrário, a nova entrada é comparada com a entrada que possui o menor valor e porventura a nova entrada tenha o valor em questão maior do que a de menor valor, a nova entrada substituirá a de menor valor.

Para a tabela “almatrixtopn” a concepção da estrutura de inserção de dados segue os mesmos moldes da tabela *TopN* para o encapsulamento de protocolo de rede. No entanto há duas diferenças:

- Os parâmetros obtidos da tabela fonte (“almatrixsd”) têm a inclusão do parâmetro referente ao encapsulamento de protocolo de alto nível;
- Quando o processo está usando as formas de ordenação número de pacotes terminais ou número de octetos terminais, apenas os protocolos de aplicação são contabilizados.

4.5 Extensão do Agente SNMP para a MIB RMON2

Quando se fala do agente SNMP está mencionando-se o agente que faz parte do pacote `Net SNMP`. Ao instalar o pacote o mesmo cria um agente SNMP que fica esperando requisições por informações de MIBs ao qual ele é capaz de resolver. O resolver uma requisição entende-se por delegar a solicitação ao “subagente” que atende a MIB. Exemplificando, a MIB-II é uma das MIBs padrões que o pacote original possui, e portanto responde as requisições feitas a ela.

No entanto, a adição de alguma MIB, tanto particular quanto padronizada que não faz parte do pacote, requer o desenvolvimento de um agente estendido que vai, a partir de então, responder as requisições feitas aos objetos pertencentes a MIB. Para anexar o agente desenvolvido ao agente SNMP existe duas maneiras: recompilar o agente desenvolvido juntamente com o pacote ou desenvolver o agente como um módulo dinâmico. A escolha feita foi o desenvolvimento de um módulo (biblioteca) dinâmico por ser uma forma mais simplificada e clara.

No anexo 1 encontra-se um tutorial de como estender o agente SNMP através das ferramentas inclusas no pacote Net SNMP e quais os procedimentos que devem ser feitos para unir o agente estendido ao agente SNMP.

O agente RMON2 desenvolvido pode ser entendido como uma extensão do agente SNMP que atende as requisições feitas pelo gerente com respeito aos objetos pertencentes a MIB RMON2.

Toda a requisição (por exemplo, um GET de um objeto da MIB RMON2) direcionada para a estação que possui o agente SNMP juntamente com as extensões, é atendida pelo agente SNMP, o qual delega a requisição para o agente estendido

RMON2 que, por sua vez, busca a informação referente ao objeto na base de dados da MIB RMON2 e a retorna para o gerente.

Para a criação do agente estendido, além do uso do pacote Net SNMP, foi usada a API que faz a comunicação com o banco de dados `MySql`.

Nas subseções a seguir, será feito o detalhamento da estrutura do agente RMON2, o modo como os dados da base de dados são recuperados, a forma de identificação de um objeto, a montagem da resposta, a configuração do valor de um objeto, entre outros detalhes ligados à operação do agente.

4.5.1 Agente RMON2

A estrutura geral do agente, isto é, a concepção de como o agente RMON2 estendido foi modelado, é formada por uma estrutura que contém todos os objetos ao qual o agente atende com seus respectivos parâmetros e por diversas rotinas. Essas rotinas podem ser classificadas da seguinte forma:

- Rotinas que atendem a determinados objetos;
- Rotinas que configuram objetos que podem ser escritos;
- Rotinas que buscam e/ou atualizam dados na base de dados RMON2 MIB.

O OID (*object identifier*) básico do agente, também referenciado como o OID raiz, para que requisições e ou configurações de objetos pertencentes a MIB RMON2 possam ser atendidos pelo agente criado é o *1.3.6.1.2.1.16*.

Para cada objeto pertencente à MIB foi atribuído um valor OID que complementa o OID raiz com o intuito de distinguí-los. Esse valor está implícito na estrutura que discrimina os objetos do agente, logo representando um dos parâmetros de cada objeto. Além desse parâmetro, cada entrada (objeto) da estrutura tem configurado o nome do objeto conforme especificado na MIB, um valor seqüencial que é usado para fins de identificar o objeto dentro da implementação (forma utilizada pelo projeto `Net SNMP`), a definição do seu tipo, um rótulo que informa se o mesmo pode ou não ser configurado, a função que o manipula e por fim a definição do comprimento do OID sufixo.

Um esboço dessa estrutura pode ser vista na tabela 4.19.

TABELA 4.19 – Estrutura que possui todos os objetos pertencentes a MIB RMON2

Nome e Nº Mágico	Tipo do objeto	RO/ RW	Função manipuladora	Comprimento do OID sufixo	OID sufixo
protocoldirlastchange (1)	Asn_Timeticks	Ronly	var_rmon2	2	11.1
protocolDirLocalIndex (6)	Asn_Integer	Ronly	var_protocolDirTable	4	11.2.1.3
protocolDirDescr (7)	Asn_Octet_Str	Rwrite	var_protocolDirTable	4	11.2.1.4
protocolDirType (8)	Asn_Integer	Ronly	var_protocolDirTable	4	11.2.1.5
…	…	…	…	…	…
nlhostInPkts (61)	Asn_Gauge	Ronly	var_nlHostTable	4	14.2.1.3
nlhostOutPkts (62)	Asn_Gauge	Ronly	var_nlHostTable	4	14.2.1.3
…	…	…	…	…	…

As rotinas manipuladoras estão separadas por tabelas da MIB RMON2, isto é, uma rotina manipula todos os objetos pertencentes àquela tabela. O motivo para se usar uma rotina por tabela é porque cada tabela tem uma forma de indexação diferente, como mostrado na definição da MIB (capítulo 2).

Apesar de possuírem indexações diferentes, todas as rotinas seguem o mesmo padrão de funcionamento. Os passos seguidos por essas rotinas podem ser assim descritos:

- Fase de verificação da existência de entradas na tabela;
- Fase de montagem do índice referente ao objeto;
- Fase de validação da requisição;
- Fase de busca do valor referente ao objeto (recuperação do dado) ou configuração de um certo valor ao objeto. A obtenção do dado como a configuração de um objeto é feita por rotinas específicas que acessam as tabelas da base de dados "rmon2".

Todas as rotinas manipuladoras possuem seis variáveis passadas por parâmetros, dos quais quatro são usadas para passar informações sobre a requisição. Esses parâmetros compreendem;

- Um apontador para o objeto que está sendo manipulado (entrada da estrutura contendo os objetos da MIB RMON2);
- O OID da requisição;
- O comprimento do OID requisitado;
- Uma variável que indica se a requisição é exata (GET ou SET) ou inexata (GET NEXT, GET WALK).

Apesar de sobrarem dois parâmetros para o retorno da informação de uma resposta, são usados quatro parâmetros. Os parâmetros OID da requisição e comprimento do OID requisitado são, também, usados para retornar dados. O que cada parâmetro retorna, pode ser assim descrito:

- O OID que está sendo retornado;
- O comprimento deste OID;
- O comprimento do tipo do dado da resposta;
- Um ponteiro para a rotina que configura o objeto (caso o objeto não possa ser configurado ou se está buscando seu valor, esse ponteiro terá o valor zero).

A figura 4.8 mostra um fluxograma demonstrando os passos seguidos por uma requisição dentro da função manipuladora. Uma ressalva que deve ser feita é quanto à presença de duas bases de dados da MIB RMON2. Na verdade, essas bases representam uma só, foram postas duas representações para deixar o fluxograma mais claro.

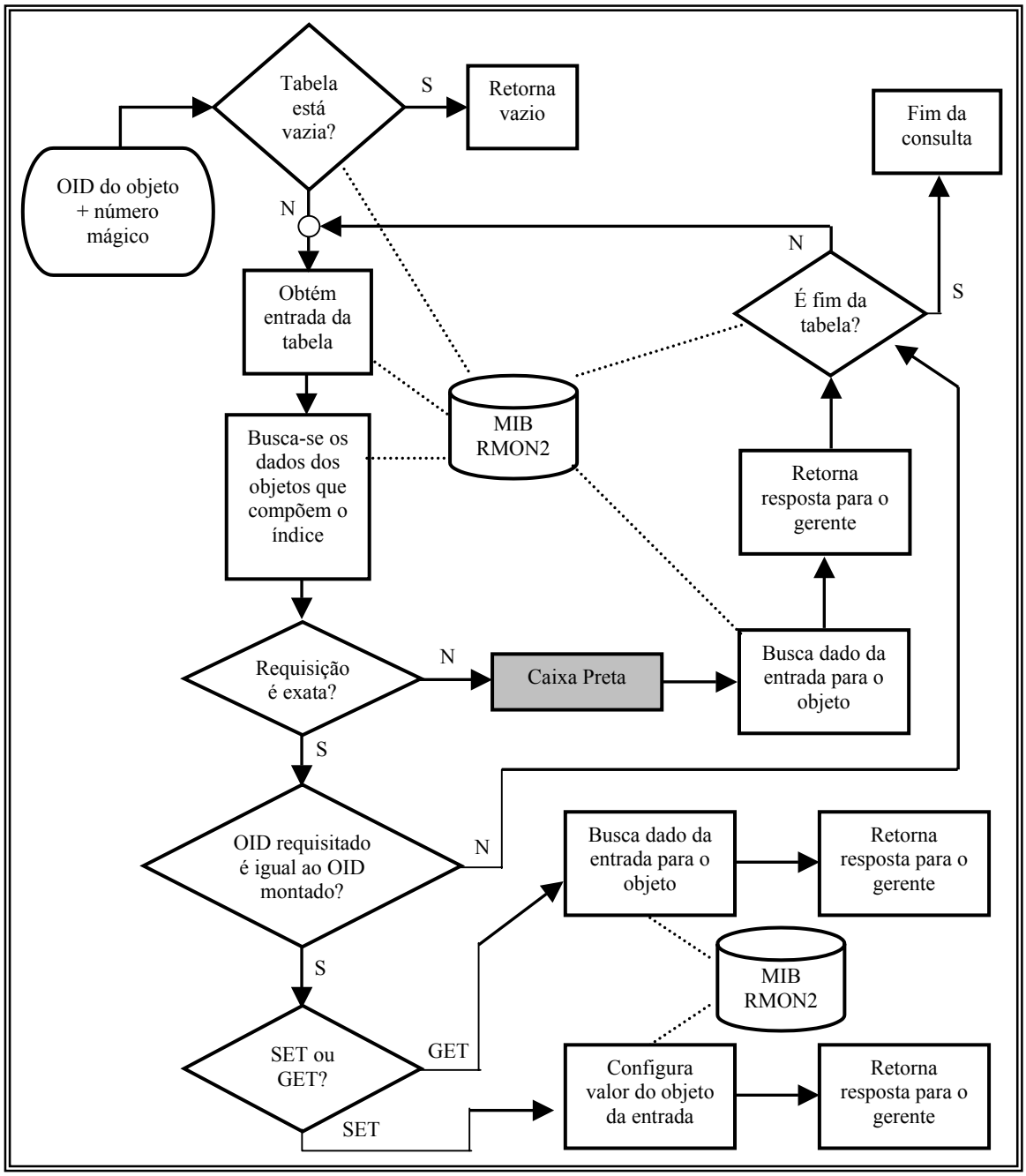


FIGURA 4.8 – Fluxograma demonstrando os passos da função manipuladora

Para entender melhor os passos seguidos desde uma requisição do gerente até o retorno da informação, um exemplo será demonstrado.

Pressupõe-se que foi feita uma requisição pelo gerente por todas as entradas da tabela "nlhost" para o objeto "nlhostInPkts" (GETWALK *1.3.6.1.2.1.16.14.2.1.3*). O agente, ao receber a requisição, verifica dentro da estrutura dos objetos que fazem parte do mesmo qual OID de objeto (OID raiz mais o OID sufixo) coincide com o OID requisitado. Descobrindo o objeto, a função que o manipula é disparada, recebendo os correspondentes parâmetros como descritos anteriormente.

Em um primeiro momento, a função manipuladora verifica se a tabela requisitada possui entradas. Caso a resposta da consulta seja nula, a função é cancelada e informa-se a ausência de entradas ao gerente. Por outro lado, se a resposta for diferente de nulo,

o próximo passo diz respeito à atribuição do índice. A obtenção dos índices se dá através de rotinas que buscam na base de dados os seus referentes dados. Esses dados são acrescentados ao OID que se está montando (OID raiz + OID do sufixo do objeto). Para visualizar um índice acrescentado ao OID requisitado, veja as entradas da tabela 4.20.

Com o OID montado acrescido do índice, o passo seguinte é verificar o tipo de requisição (exato ou inexato) feito pelo gerente.

Se o tipo de requisição for exato, o que não é o caso do exemplo, compara-se o OID requisitado com o OID montado. Obtendo como resposta a igualdade e a requisição for um GET, o valor do objeto é obtido através de uma rotina que busca a informação na base de dados. No entanto se a requisição for um SET, o valor do objeto é configurado através de uma rotina específica para o objeto.

Porém, se o tipo de requisição é inexato (exemplo), o procedimento para se obter os valores do objeto para as entradas pode ser dividido em 4 passos:

1. para a primeira entrada da tabela que contém o objeto requisitado, verifica-se se o OID montado acrescido do índice é maior do que o OID requisitado. Caso for, uma variável auxiliar guarda o OID da entrada. Exemplificando através do exemplo temos: OID da primeira entrada, o qual representa o OID montado é igual a *1.3.6.1.2.1.16.14.2.1.3.1.56946.1.10.2.2.2*, já o OID requisitado é igual a *1.3.6.1.2.1.16.14.2.1.3*. Portanto, a variável auxiliar receberá o valor do OID montado;
2. para as demais entradas, verifica-se se o OID montado (próxima entrada, respectiva, da tabela) acrescido do índice é maior do que o OID requisitado e o OID montado acrescido do índice é menor do que o OID armazenado na variável auxiliar. Caso a condição for verdadeira a variável auxiliar recebe o OID montado da entrada;
3. repete-se o segundo procedimento até ter passado por todas as entradas da tabela;
4. após a varredura por todas as entradas, a entrada menor (valor dos índices ordenados de forma crescente) requisitará seu valor, através de uma rotina específica, na base de dados "rmon2" e retornará o seu valor para o gerente;
5. repete-se os passos 1 ao 4 até todas as entradas terem sido respondidas.

Os procedimentos 1, 2, 3 compreendem a "Caixa Preta" da figura 4.8.

A tabela usada para a demonstração da requisição do objeto "nlhostInPkts" de todas as entradas da tabela "nlhost" assim como a separação dos campos da tabela entre índice e dados pode ser visualizada na figura 4.9.

Control Index	TimeMark	Local Index	Address	InPkts	OutPkts	InOctets	OutOctets	OutMac NonUnicast Pkts	Create Time
1	56946	1	10.2.2.2	5	10	…	…	0	5462
1	70202	1	10.2.2.6	100	90	…	…	8	6355
1	60340	1	10.2.2.9	43	40	…	…	5	8129
1	60323	1	10.2.2.3	80	85	…	…	0	8395

Índice — Dados dos objetos

FIGURA 4.9 – Exemplo da tabela "nlhost" com algumas entradas

Já o resultado da requisição feita para o agente está descrito na tabela 4.20. Pode-se observar também nessa tabela a ordenação feita dos índices. Portanto, a seqüência em que estão as entradas representa a ordem de retorno das mesmas para o gerente.

TABELA 4.20 – Respostas enviadas para o gerente após uma consulta a tabela "nlhost" da figura 4.9

Objeto mais índice	Resultado da entrada
NlhostInPkts.1.56946.1.4.10.2.2.2	5
NlhostInPkts.1.60323.1.4.10.2.2.3	80
NlhostInPkts.1.60340.1.4.10.2.2.9	43
NlhostInPkts.1.70202.1.4.10.2.2.6	100

As rotinas que são disparadas quando há uma requisição de configuração (SET) possuem algumas peculiaridades. Cabe ressaltar que cada objeto pertencente à MIB RMON2 que pode ser configurado possui uma rotina própria. Apesar de existir uma rotina distinta para cada objeto, elas possuem três fases básicas que devem ser percorridas antes de se fazer a alteração do valor do campo. Caso uma das fases gere um erro ou uma condição falhar, o valor do objeto não será alterado.

As fases são assim detalhadas:

- RESERVE: usada para verificar a sintaxe das variáveis (tipo da variável é o mesmo do objeto), a consistência do valor que se deseja atribuir ao objeto (se um objeto pode apenas receber os valores 1, 2 ou 3 então deve-se verificar se o valor repassado pelo SET coincide com os mesmos) e alocar quaisquer recursos requeridos para a execução do SET. Ocorrendo o sucesso dessa fase, a rotina de escrita é chamada com a ação ACTION. No entanto pode ocorrer de uma das condições falharem, neste caso a rotina de escrita é chamada com o intuito de liberar os recursos que tenham sido alocados;
- ACTION: nessa fase é onde ocorre a implementação da operação SET. Mas o detalhe desta fase é que a operação SET é realizada sobre uma variável temporária. O motivo é para prevenir algum erro que possa ocorrer como, por exemplo, um valor que aparentemente é válido, mas, no entanto é um valor inaceitável. Passando por essa fase, a rotina de escrita é chamada novamente com a ação COMMIT. E se ocorrer um erro chama-se a rotina com a ação UNDO para liberar todos os recursos alocados por ela;
- COMMIT: fase final da rotina de configuração de um objeto. Essa fase tem como função gravar o valor atribuído à variável temporária na fase ACTION a um repositório confiável (a base de dados da MIB RMON2) e liberar os recursos alocados pela rotina.

5 Exemplos de aplicação do Agente

Para validar o agente RMON2 criado, foram feitos dois exemplos: o primeiro mostra o agente monitorando a rede de computadores de uma organização real, já o segundo tem como meta avaliar o desempenho. Cada exemplo criado foi realizado em um cenário específico.

Nas subseções a seguir os exemplos serão descritos assim como os resultados obtidos dos mesmos.

5.1 Exemplo 1

O primeiro exemplo foi realizado em uma organização e busca analisar o tráfego de dados dos *hosts* da Intranet, Extranet e dos acessos remotos discados além de uma visão geral do tráfego.

As estações da Intranet compreendem as redes:

- 10.25.108.0 com máscara 255.255.255.0;
- 10.25.111.0 com máscara 255.255.255.0;
- 10.25.144.0 com máscara 255.255.240.0.

As estações da Extranet possuem, também, IPs não válidos para a Internet. Todos os IPs que não fazem parte da rede da Intranet representam estações da Extranet.

As estações que fazem acesso remoto podem receber dezesseis IPs diferentes, dependendo do modem no qual a conexão é estabelecida. Esses IPs vão de 10.25.110.101 até 10.25.110.116.

O *host*, que possui o agente, foi conectado em uma porta do *Switch* que tem conectado a si uma interface do *Firewall* e servidores da Intranet. O *Switch* recebeu a configuração de espelhar todos os pacotes que passam pela porta onde está conectado o *Firewall* para a porta onde está anexado o agente.

A plataforma de hardware do agente foi um microcomputador Pentium 200 MMX® com 64Mb de memória RAM.

A velocidade de transmissão do segmento de rede onde o agente está conectado é de 10 Mbits/s.

Para acesso a Internet, os usuários da Intranet utilizam um *Proxy* que está instalado no *Firewall*, sendo esse protocolo de aplicação um exemplo da monitoração de quaisquer protocolos/aplicações, uma vez que a porta de comunicação utilizada é diferente do padrão usado.

O período de análise feito foi de uma hora, sendo que o período de obtenção dos dados foi de quinze em quinze minutos. Como o período de captura foi pequeno, o objeto *NlMaxDesiredEntries* e *AlMaxDesiredEntries* da tabela *hlHostControl* receberam o valor "–1" e como foi comentado no capítulo que descreve a MIB RMON2, quando esses objetos estiverem com o valor "–1" a quantidade de entradas em suas respectivas tabelas de dados é ilimitado.

O agente foi configurado para receber contabilizações apenas pelo número de octetos para a tabela *nlMatrixTopN* e a tabela *alMatrixTopN* recebeu contabilizações pelo número de octetos terminais. A quantidade máxima de entradas para as tabelas *topN* foi de dez.

A representação esquemática do cenário do exemplo pode ser vista na figura 5.1.

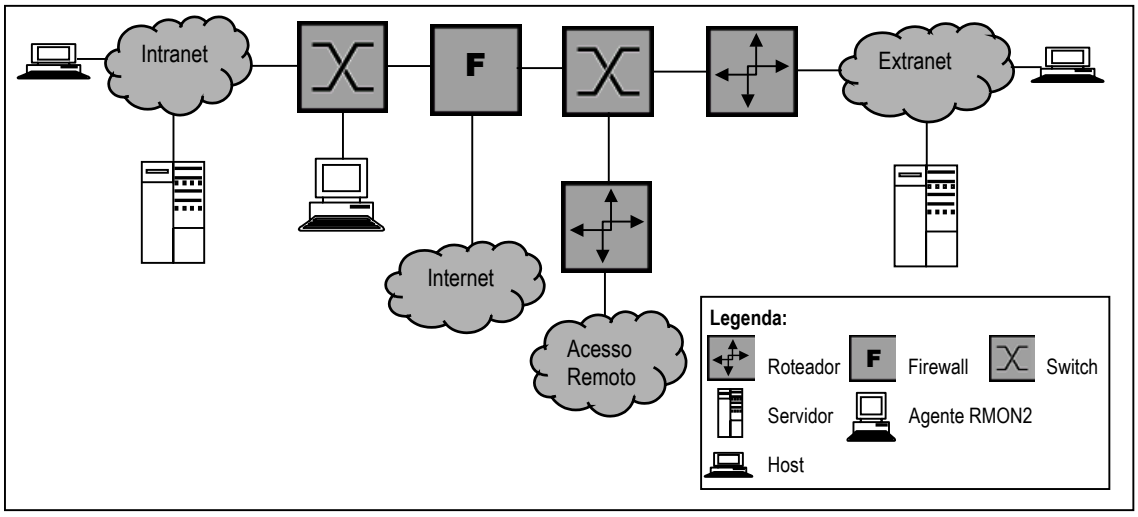


FIGURA 5.1 – Cenário da organização monitorada

De posse das informações oriundas da MIB RMON2, as estatísticas do comportamento da rede numa visão geral, assim como específico por *hosts* ou por pares de *hosts* foi alcançado.

5.1.1 Visão geral

Com respeito à visão geral do tráfego, observou-se que a utilização da banda para o protocolo IP foi em média de 0,95% durante o período de uma hora. Isso mostra que houve pouco consumo de banda, uma vez que o tráfego é totalmente feito sobre o protocolo de rede IP. Para chegar ao resultado, utilizou-se a fórmula (4) descrita na seção 3.2, e os dados para popular a mesma foram os objetos *InOctets* e *OutOctets* da entrada da tabela *protocoldiststats* que referencia o protocolo IP.

Dos aplicativos configurados na tabela *protocoldir* do agente e que estavam ativos, os que receberam contabilização, isto é, aqueles aplicativos que se constatou pacotes trafegando pelo segmento monitorado foram:

- Proxy;
- DNS;
- SMTP
- HTTP;
- POP3;
- Telnet;
- IMAP;
- SNMP;
- FTP;
- HTTPS.

Cada aplicativo gerou uma porcentagem de utilização da rede durante o período de monitoramento. A tabela 5.1 representa um intervalo de amostragem, a qual expõe a taxa de utilização da banda para os aplicativos acima descritos.

TABELA 5.1 – Taxa de utilização da rede por protocolos de aplicação

Protocolo de Aplicação	Utilização
Proxy	0,92%
DNS	0,0052%
SMTP	0,0002%
HTTP	0,079%
POP3	0,004%
Telnet	0%
IMAP	0%
SNMP	0,0003%
FTP	0,00002%
HTTPS	0,008%

O protocolo Proxy, que representa o acesso a Internet pelos usuários da Intranet, compreende quase o total da utilização da banda nesse segmento.

O gráfico com a variação da utilização da rede pelos aplicativos durante o período de uma hora pode ser visto na figura 5.2.

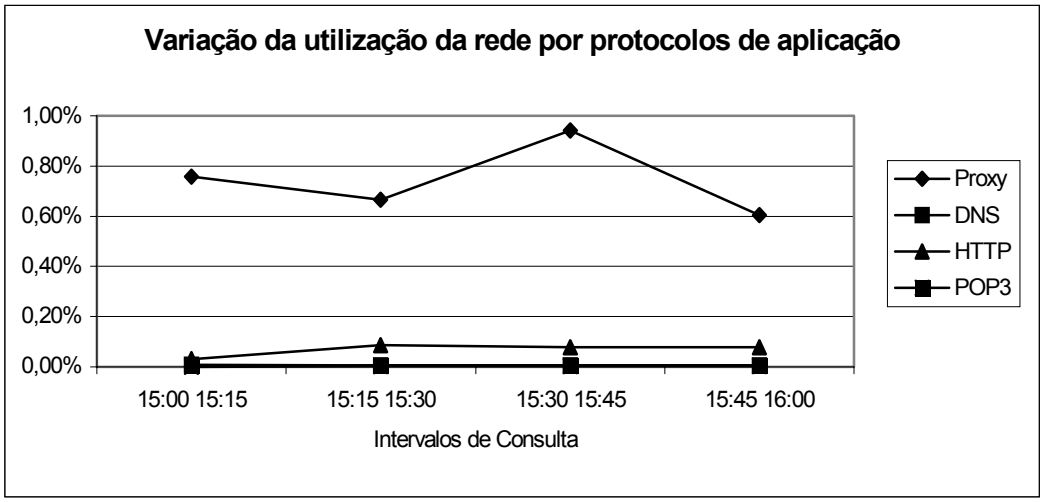


FIGURA 5.2 – Taxa de utilização da rede por protocolos de aplicação

Ainda verificando o perfil global de utilização da rede, a taxa de utilização da rede por grupos baseando-se no volume total de tráfego pode ser obtida através da tabela *nlhost* e aplicando a fórmula (2), descrita na seção 3.1.2, com a variante que é para *hosts* e não para os protocolos. Obtém-se para cada *host* a sua taxa de utilização e após soma-se os resultados dos *hosts* que pertencem a um determinado grupo. Os grupos são: acesso discado, departamento V, X, Y, Z, W pertencentes a Intranet. A tabela 5.2 mostra a porcentagem de uso da rede de cada grupo.

TABELA 5.2 – Taxa de uso da rede por agrupamentos

Grupo	Taxa de Utilização
Acesso Discado	1,12%
Departamento V	0,006%
Departamento X	6,55%
Departamento Y	29,30%
Departamento Z	1,25%
Departamento W	2,76%

5.1.2 Estatísticas de hosts

O total de *hosts* distintos que transmitiram e/ou receberam pacotes durante o período de monitoração foi de 214. Esse dado foi obtido através da quantidade de entradas presentes na tabela *nlhost*.

Assim como dito no capítulo 3, a partir dos dados da tabela *nlhost* pode-se descobrir o quanto uma estação utiliza da rede, o quanto de pacotes e octetos foram transmitidos e quanto foram recebidos pela estação, pode-se, também, ter uma idéia da quantidade de pacotes *broadcast* ou *multicast* transmitidos pela estação. A tabela 5.3 mostra alguns *hosts* escolhidos aleatoriamente que mostram as contabilizações retiradas diretamente da tabela *nlhost*.

TABELA 5.3 – Amostra de informação retirada da tabela "nlhost"

Endereço IP	Pacotes recebidos	Pacotes transmitidos	Octetos recebidos	Octetos transmitidos	Pacotes não unicast
10.25.108.21	42.488	48.919	6.428.354	2.812.508	0
10.25.146.65	1.486	1.421	961.970	300.012	0
10.25.108.25	6.773	5.501	805.700	643.244	0
10.25.108.42	1.167	1.844	168.723	229.995	5
10.67.61.7	370	489	51.567	550.763	0
10.25.108.2	924	1.343	139.650	768.987	3

A partir da tabela acima e com os dados colhidos dos demais intervalos, o gráfico da figura 5.3 mostra a taxa de utilização da rede pelo servidores 10.25.108.21 (Servidor Proxy) e 10.25.108.25 (Servidor DNS, SMTP, POP3).

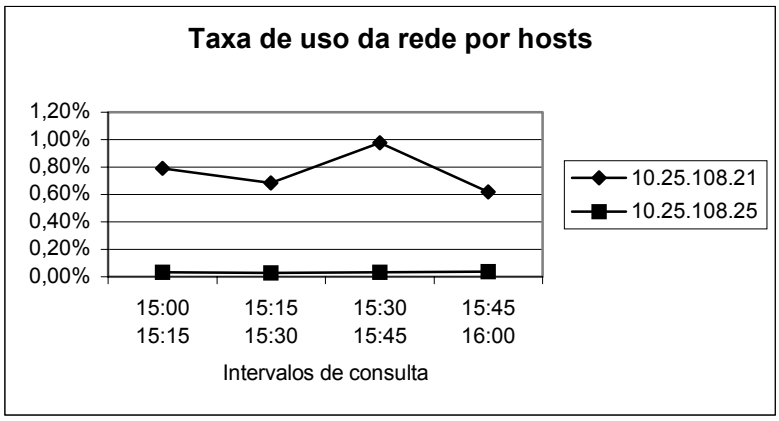
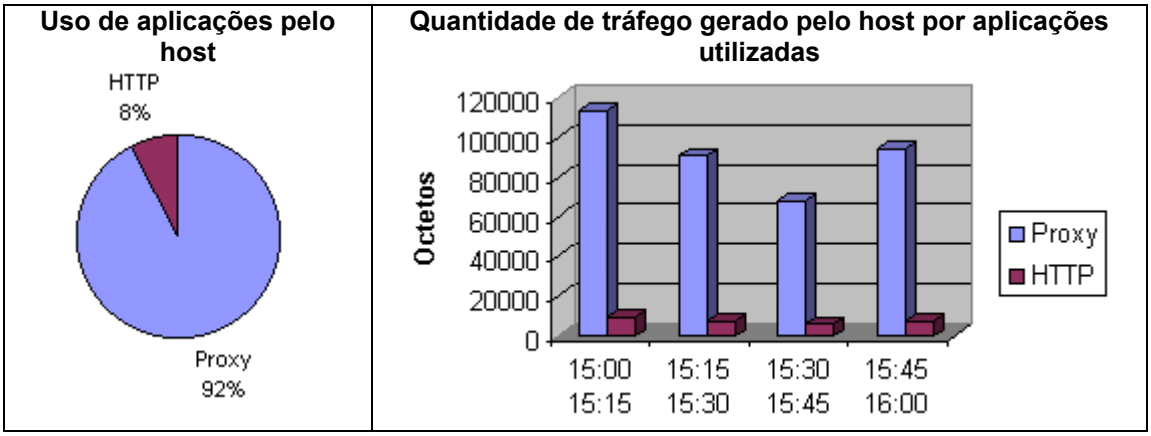


FIGURA 5.3 – Taxa de utilização da rede por dois *hosts*

A máquina 10.25.108.42 é uma estação de trabalho que acessa recursos da rede. Esses recursos, ou melhor, aplicações que essa estação utiliza podem ser obtidas através de consultas à tabela *alhost*. Além de descobrí-las, pode-se obter a porcentagem de uso de cada aplicação pela estação.



(a) protocolos de aplicação (b) Tráfego por aplicações

FIGURA 5.4 – Protocolos de aplicação e seus respectivos tráfegos durante um certo período para um *host*

Conhecendo o tráfego de uma estação (vide figura 5.4b) assim como os protocolos de aplicação que elas utilizam (vide figura 5.4a), o próximo passo é descobrir os pares de estações que estabelecem uma comunicação.

5.1.3 Comunicações entre pares de *hosts*

Como pôde ser visto na visão geral, o tráfego que mais foi observado durante o período de monitoração foi o acesso a Internet pelas estações da Intranet.

Através da tabela *nlmatrixds* é possível gerar gráficos que representem as estações que estabeleceram uma comunicação com o servidor Proxy. A figura 5.5 mostra algumas estações que conectaram a Internet.

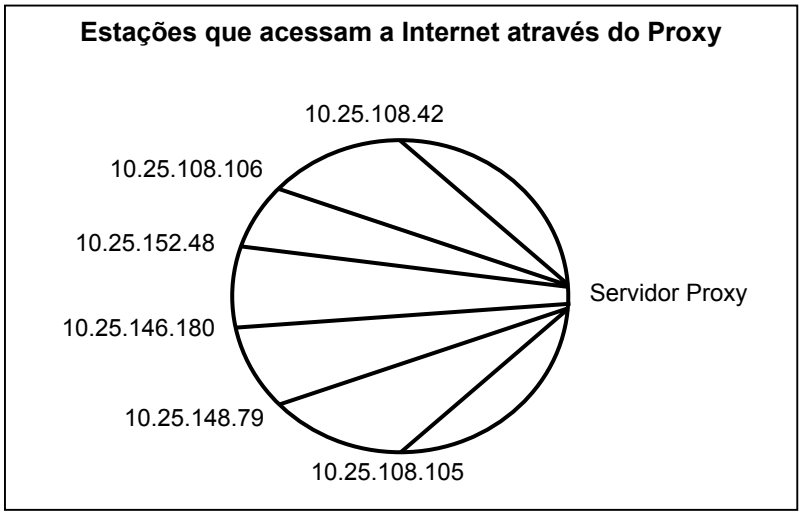


FIGURA 5.5 – Estações que acessaram o Servidor Proxy

As estações remotas estabeleceram conexões com o servidor de DNS e Correio Eletrônico (10.25.108.25) e com o Servidor Web da corporação (10.25.108.2). Para visualizar as conexões veja a figura 5.6. Neste gráfico nota-se a presença dos pares de máquinas que realizaram uma comunicação e ao mesmo tempo distingue o tipo de protocolo de aplicação executado. Os dados, para conseguir o resultado gráfico da figura, foram obtidos da tabela *almatrixsd*.

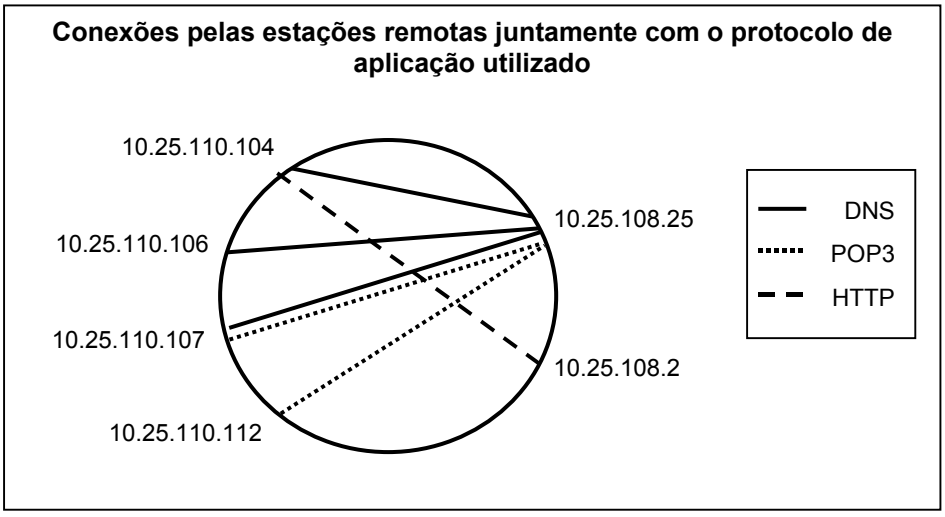


FIGURA 5.6 – Comunicações entre pares de máquinas juntamente com o protocolo de aplicação utilizado

Os pares de *hosts* que mais se comunicaram durante o período de monitoração e juntamente com o protocolo de aplicação utilizado pode ser visto na tabela 5.4. A tabela que foi requisitada para alcançar o objetivo foi a *alMatrixTopN*.

TABELA 5.4 – Pares de estações que mais se comunicam com a respectivo protocolo de aplicação

Estação origem	Estação destino	Aplicativo	N. Octetos (origem – destino)	N. Octetos (destino – origem)
10.25.108.106	10.25.108.21	Proxy	341.782	3.321.699
10.25.152.56	10.25.108.21	Proxy	227.777	1.025.467
10.25.108.42	10.25.108.21	Proxy	198.268	168.723
10.25.111.10	10.25.108.25	DNS	100.569	56.740
10.25.108.101	10.25.108.21	Proxy	61.540	47.146
10.25.108.123	10.25.108.21	Proxy	60.516	42.640
10.25.108.25	10.67.4.34	DNS	50.887	38.973
10.25.108.42	209.73.225.17	HTTP	30.720	0
10.25.108.110	10.25.108.21	Proxy	16.220	17.880
10.25.146.65	164.109.144.201	HTTP	3.680	0

Quanto às questões de segurança, não houve indícios de tentativa de invasão. As tabelas *alhost* e *almatrixsd* foram verificadas e nem os *hosts* nem os pares de *hosts* apresentavam uma aplicação que pudesse representar um ataque. Explicando o que foi feito, na tabela *alhost* observou-se os protocolos de aplicação presentes nos pacotes destinados a ou enviados para um determinado *host* que estava sob análise. Caso algum protocolo atípico (não deveria ter sido descoberto) foi encontrado, isso poderia representar uma invasão.

No entanto observou-se que as estações cliente 10.25.108.42 e 10.25.146.65 apresentam uma equivocada configuração do navegador com respeito ao Proxy ou possui algum software que tenta estabelecer uma conexão com um *host* da Internet. Caso a segunda hipótese seja a verdadeira, pode representar um indício de ataque.

5.2 Exemplo 2

O segundo exemplo tem por finalidade validar o desempenho do agente. A idéia é monitorar o tráfego de dados entre dois *hosts* (cliente e servidor) variando o período de transmissão de pacotes e a taxa de utilização da rede. Para a geração do tráfego foi utilizado um *software* específico chamado `Iperf` [TIR2001], o qual gera tráfego TCP ou UDP conforme as configurações especificadas.

Para a realização do exemplo foi criada uma rede isolada com três máquinas, sendo que a interconexão das estações foi feita através de um *Hub* com velocidade de transmissão de 10 Mbits/s. A primeira e a segunda máquina são, respectivamente, as estações cliente e servidor que possuem o *software* `Iperf` instalado. Já a última máquina possui o agente RMON2 instalado e sua configuração de *hardware* era um K6-III 400® com 64Mbytes de memória RAM. O cenário desse exemplo pode ser visualizado na figura 5.7.

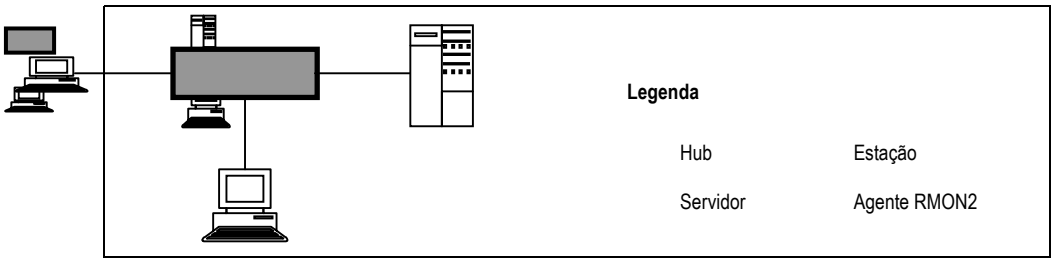


FIGURA 5.7 – Cenário para o teste de desempenho

O tamanho do datagrama para os testes foi de 403 bytes. Esse valor foi escolhido por ser o tamanho médio dos pacotes que trafegam pela Internet. O dado foi obtido através de um estudo realizado pela equipe de gerenciamento e análise de operações da *National Library for Applied Network Research* (NLANR), a qual observou uma amostra de 342 milhões de pacotes que trafegaram pela Internet e obteve como um de seus resultados o tamanho médio dos pacotes [AGI2001].

Nas seções seguintes serão descritos os testes realizados para a verificação do desempenho do agente, sendo que as diretivas observadas foram o consumo do processador e memória pelos módulos e pela base de dados que compõem o agente, assim como a quantidade de pacotes que o agente conseguiu capturar.

5.2.1 Primeiro teste

O primeiro teste visou verificar principalmente a perda de pacotes do módulo "capturador". Para esse teste foi inicializado o banco de dados `MySql` e o módulo "capturador". As monitorações basearam nos seguintes parâmetros:

- Taxa de transmissão de 100kbits/s em períodos de tempo de 10 segundos, 1 minuto e 10 minutos;
- Taxa de transmissão de 1Mbit/s em períodos de tempo de 10 segundos, 1 minuto e 10 minutos;
- Taxa de transmissão de 8Mbits/s em períodos de tempo de 10 segundos, 1 minuto e 10 minutos.

Os resultados obtidos do teste para o tempo de tráfego de 10 segundos podem ser observados na tabela 5.5. Para o tempo de tráfego de 1 minuto, a tabela 5.6 mostra os resultados e a tabela 5.7 apresenta os dados obtidos para o tempo de tráfego de 10 minutos.

TABELA 5.5 – Primeiro teste com tempo de tráfego de 10 segundos

Taxa de transmissão	Pacotes enviados	Pacotes obtidos	Pacotes capturados (%)
100 Kbits/s	319	319	100%
1 Mbit/s	3.178	3.178	100%
8 Mbits/s	25.447	12.726	50%

TABELA 5.6 – Primeiro teste com tempo de tráfego de 1 minuto

Taxa de transmissão	Pacotes enviados	Pacotes obtidos	Pacotes capturados (%)
100 Kbits/s	1.907	1.907	100%
1 Mbit/s	19.061	19.061	100%
8 Mbits/s	152.674	72.932	48%

TABELA 5.7 – Primeiro teste com tempo de tráfego de 10 minutos

Taxa de transmissão	Pacotes enviados	Pacotes obtidos	Pacotes capturados (%)
100 Kbits/s	19.059	19.059	100%
1 Mbit/s	190.599	190.599	100%
8 Mbits/s	1.522.326	717.115	47,2%

O que pôde ser constatado nesse teste foi à quantidade de perda de pacotes ocorrida quando a taxa de utilização aumenta, conforme visto nas tabelas 5.5, 5.6 e 5.7. Tal fato pode ser explicado pela incapacidade de inserção dos pacotes na tabela do banco de dados. No caso da taxa de transmissão de 8 Mbits/s, o banco de dados deveria fazer 2.481,4 inserções por segundo.

A constatação de que o banco de dados era o gargalo veio a partir de um teste onde se verificou que a quantidade de pacotes capturados pelo módulo "capturador" era a mesma da quantidade de pacotes enviados ou numa pior hipótese tinha uma porcentagem de perda de 3%, mas esse valor não retratava o número de entradas da tabela "conexão" pertencente à base de dados "cap_pac".

Uma solução para esse problema seria o estudo de um meio para o armazenamento dos pacotes que fosse mais rápido ou então utilizar uma estação mais robusta.

5.2.2 Segundo teste

O segundo teste visou verificar o consumo de memória e CPU, além da perda de pacotes do agente. Para esse teste, o agente RMON2 foi totalmente inicializado. Quando se fala em totalmente inicializado, quer dizer que o módulo "capturador", o "conversor" (com todas as formas de ordenação das tabelas *TopN* habilitadas), o banco de dados `MySql` e o agente SNMP com a extensão para o agente RMON2 foram ativados.

As monitorações, para esse segundo teste, foram realizadas com os mesmos parâmetros do primeiro teste, sendo que os resultados obtidos para o período de tráfego de 10 segundos podem ser visualizados na tabela 5.8. Para o período de tráfego de 1 minuto, a tabela 5.9 apresenta os dados alcançados e por fim a tabela 5.10 mostra as estatísticas para o período de tráfego de 10 minutos.

TABELA 5.8 – Segundo teste com tempo de tráfego de 10 segundos

Taxa de	Pacotes	MySql		Capturador		Conversor	
transmissão	capturados (%)	CPU	Memória	CPU	Memória	CPU	Memória
100 Kbits/s	100%	78%	3%	7%	2%	15%	6%
1 Mbit/s	100%	78%	3%	7%	3%	15%	17%
8 Mbits/s	48,6%	78%	3%	7%	5%	15%	28%

Analisando os resultados da tabela 5.8, podemos considerar:

- O `MySql` tem suas taxas de utilização de CPU e memória constantes nas variações da taxa de transmissão;
- O módulo "capturador" tem seu consumo de CPU em torno de 7% durante o período de tráfego e após, passa a não utilizar a CPU. Já o consumo de memória tem uma variação de 2% a 5% dependendo da quantidade de pacotes transmitidos por segundo;
- O módulo "conversor" passa a utilizar os recursos de CPU e memória depois de transcorridos 60 segundos, pois, como foi mencionado na descrição do módulo (seção 4.4.1), o agente atualiza as tabelas da MIB RMON2 de minuto em minuto. Quando o módulo é executado, ele passa a utilizar em torno de 15% da CPU, chegando a 20%. Já o consumo de memória está relacionado a quantidade de pacotes transmitidos por segundo e essa variação vai de 6% até 28%;
- A porcentagem de pacotes capturados não teve uma grande variação quando comparado ao primeiro teste, pois os módulos "capturador" e "conversor" foram executados em períodos diferentes. A pequena variação se deve ao maior número de processos inicializados.

TABELA 5.9 – Segundo teste com tempo de tráfego de 1 minuto

Taxa de transmissão	Pacotes capturados (%)	MySql		Capturador		Conversor	
		CPU	Memória	CPU	Memória	CPU	Memória
100 Kbits/s	100%	78%	3%	7%	2%	15%	6%
1 Mbit/s	100%	78%	3%	7%	3%	15%	17%
8 Mbits/s	46%	78%	3%	7%	5%	15%	28%

Os resultados da tabela 5.8, basicamente, são os mesmos do período de tráfego de 10 segundos. A única variação ocorreu quando a taxa de transmissão de dados foi de 8 Mbits/s, pois houve concorrência pelo uso do banco de dados entre os módulos, mas durante um pequeno período de tempo.

TABELA 5.10 – Segundo teste com tempo de tráfego de 10 minutos

Taxa de transmissão	Pacotes capturados (%)	MySql		Capturador		Conversor	
		CPU	Memória	CPU	Memória	CPU	Memória
100 Kbits/s	100 %	78 %	3 %	7 %	2 %	15 %	10 %
1 Mbit/s	99,9 %	78 %	3 %	7 %	4 %	15%	25 %
8 Mbits/s	38,1 %	78 %	3 %	7 %	10 %	15 %	50 %

A grande perda de desempenho pôde ser visualizada com o período de tráfego de 10 minutos. Durante o período de tempo, os módulos "capturador" e "conversor" (*thread* que insere dados nas tabelas da MIB RMON2 e as *threads* que populam as tabelas *TopN* que são executadas de 5 em 5 minutos) concorreram significantemente pelo uso do `MySql`.

O banco de dados `MySql` ficou permanentemente consumindo 78% a 80% da CPU. Com a taxa de transmissão de 8 Mbits/s, o desempenho do agente baixou significantemente. A taxa de pacotes capturados com sucesso caiu para 38,1%, o módulo "conversor" continuou utilizando de CPU em torno de 15% mas o consumo de memória teve um salto significante (50%).

Outro problema constatado foi a enorme demora na atualização das tabelas da MIB RMON2, dando a impressão de que a *thread* tinha parado, mesmo após o término do período de geração do tráfego.

Vale ressaltar que não houve requisições por parte de um gerente às tabelas da MIB RMON2, o que oneraria ainda mais o agente por também acessar as tabelas da MIB.

Partindo para as considerações finais do teste, a primeira conclusão que se pôde obter foi que o banco de dados `MySql` é o maior gargalo do agente. Pode-se dizer, também, que com a plataforma de *hardware* utilizada e com uma porcentagem de uso da rede elevada o agente teve um desempenho muito aquém das expectativas.

Para buscar um melhor desempenho para o agente, uma possibilidade seria a utilização de uma estação que possua um maior poder de processamento (talvez mais do que um processador, já que são disparadas várias *threads*), bastante memória RAM e um disco rígido bastante rápido.

6 Conclusões

Tendo em vista a busca de um gerenciamento das aplicações que trafegam sobre as redes de computadores, surgiu a necessidade do desenvolvimento de uma aplicação que oferecesse a um gerente de rede informações que propiciassem ao mesmo gerar uma análise estatística do comportamento tanto da aplicação num âmbito geral como mais especificamente dos *hosts* que a utilizam. A partir dessas informações, pode-se realizar mudanças na rede (lógica ou física) no intuito de oferecer a aplicação um desempenho satisfatório.

Como se sabe, o padrão RMON2 tem a capacidade de monitorar protocolos das camadas superiores do modelo TCP/IP (rede, transporte e aplicação), logo ele tem os requisitos necessários para a realização da monitoração de aplicações que são executadas sobre a rede.

Muitas empresas que atuam na área de gerencia de redes de computadores já desenvolveram *probes* RMON2. No entanto, esses *probes* são limitados no número de aplicações/protocolos capazes de monitorar e para conseguir monitorar mais protocolos/aplicações, os mesmos necessitam de uma atualização do software do agente pelo fabricante. Outro problema enfrentado por esses *probes* é a capacidade de monitorar aplicações proprietárias, uma vez que os protocolos que o mesmo atende são aqueles padronizados ou bastante conhecidos no mundo da Internet.

Seguindo esses princípios, o agente RMON2 desenvolvido buscou quebrar essa barreira existente no que tange a adaptabilidade do agente para gerenciar quaisquer protocolos/aplicações.

Além da gerência de quaisquer protocolos/aplicações, o agente tem uma segunda contribuição não menos importante, pois ele pode ser usado pela comunidade científica como base para novas implementações, uma vez que está voltado aos princípios de *software* livre e código aberto.

O trabalho de mestrado de Meneguetti [MEN2001] assim como a tese de doutorado de Gaspary [GAS2001b] podem ser listados como dois exemplos de uso desse agente RMON2 como software base para novas implementações.

Como o objetivo do agente é a monitoração de aplicações, o estudo realizado sobre como usar a MIB RMON2 para retornar informações ao gerente das atividades tanto de estações como das aplicações na rede mostrou a quantidade de estatísticas que podem ser obtidas da MIB. Quais grupos, tabelas, objetos da MIB devem ser consultados para a obtenção dessas estatísticas também foi parte do estudo o que propiciou eleger os grupos que contemplariam o agente.

As dificuldades encontradas durante o período de realização do trabalho começam com as pesquisas da escolha de como poderia ser feita a gerência das aplicações, se a implementação seguiria algum padrão ou seria algo particular, assim como quais estatísticas deveriam ser fornecidas.

Posteriormente, com a definição da forma de gerência, passou-se a buscar os softwares necessários para compor a base do agente e o conseqüente estudo de suas funcionalidades. Nesse período, foram analisados softwares que pudessem solucionar os requisitos necessários, fossem livres e altamente utilizados pela comunidade científica.

Na fase de implementação do agente, dificuldades foram encontradas na construção tanto dos módulos como da extensão do agente SNMP para suportar a MIB RMON2. No módulo “capturador”, a dificuldade foi na construção das rotinas necessárias para a captura dos pacotes, sendo solucionada com o estudo de exemplos de programas que

utilizavam a biblioteca de captura escolhida. No módulo “conversor” a dificuldade ocorreu nos inúmeros procedimentos e verificações que deveriam ser realizados para acrescentar um dado as tabelas da MIB RMON2. Por fim, as dificuldades para estender o agente SNMP foram solucionadas com um estudo da documentação disponível no *site* do projeto `Net SNMP` [NET2001], assim como a participação no grupo de discussão do mesmo. Desse estudo, gerou-se um tutorial de como estender o agente SNMP e o mesmo foi anexado ao trabalho para servir como auxílio para quem desejar estendê-lo.

Quanto ao desempenho do agente RMON2, pode-se dizer que para uma rede com pouco tráfego o agente se comporta bem. Já para uma rede que trafega uma grande quantidade de pacotes o desempenho deixa a desejar. Tal fato ocorre pela sobrecarga do banco de dados `MySql` e por sua enorme utilização da CPU da máquina, conforme pode ser visto nos dados das tabelas da seção 5.2. Para possibilitar um melhor desempenho nesses casos de grande volume de dados, uma solução seria a utilização de máquinas com um alto poder de processamento, bastante memória RAM e um disco rígido bastante rápido.

Dando continuidade ao projeto de desenvolvimento do agente RMON2, alguns trabalhos futuros, ligados ao aperfeiçoamento do agente poderiam ser considerados:

- Implementação dos demais grupos da MIB RMON2;
- Implementação do suporte a outros protocolos na camada de enlace, pois o agente suporta apenas o protocolo *Ethernet* para essa camada;
- Estudo mais aprofundado do desempenho do agente (possível otimização na forma de armazenamento dos dados). Apesar do `Mysql` ser um dos bancos de dados de melhor desempenho, ele gera muito consumo de CPU por estar constantemente sendo requisitado pelo agente;
- Desenvolver um estudo de que plataforma de hardware o agente teria um desempenho bastante satisfatório levando em consideração desempenho versus custo do hardware.

A n e x o s

Anexo 1 Estendendo o agente Net SNMP

Através desse anexo serão dados os procedimentos necessários para a escrita de um código que estenda as funcionalidades do agente de gerenciamento de rede `Net SNMP`.

O projeto do agente `Net SNMP` vem sendo projetado para capacitar a sua extensão através da adição de novos módulos. Como seu antecessor, o `CMU-SNMP` (*Carneige Mellon University*), a capacidade de chamar scripts externos é provavelmente o método mais simples de estender o agente.

No entanto, há situações onde tal tentativa se mostra inapropriada, por causa de certos parâmetros como velocidade, acesso a dados, confiabilidade, segurança, entre outros. Nestes casos, a solução mais óbvia é prover um código C que possa ser compilado dentro do agente para implementar o módulo desejado.

Para facilitar a geração deste código C, os pacotes `Net SNMP` mais recentes possuem uma ferramenta que a partir de uma MIB (arquivo contendo a especificação da MIB) gere o esqueleto do código C.

Para criar um novo módulo MIB, três arquivos são necessários:

- O arquivo com a definição da MIB;
- O arquivo *header* C;
- O arquivo de implementação C.

O arquivo MIB

Esse arquivo define o módulo MIB a ser implementado. Ele não é absolutamente necessário, pois o agente não faz qualquer uso direto das definições da MIB, mas é aconselhável usa-lo por 2 razões:

- Fornece uma inicial especificação do que é para ser implementado;
- A ferramenta `mib2c` usa essa descrição para produzir os arquivos C.

Para gerar os arquivos C a partir do arquivo MIB deve-se primeiramente copiar o arquivo com a MIB para o diretório de repositórios das MIBs (vide quadro A1.1).

```
# cp <arquivo_mib>.txt /usr/local/share/snmp/mibs
ou
# mkdir $HOME/.snmp
# mkdir $HOME/.snmp/mibs
# cp <arquivo_mib>.txt $HOME/.snmp/mibs
```

QUADRO A1.1 – Formas de copiar o arquivo MIB para o repositório de MIBs

O passo subseqüente é o de atribuir a variável MIBS a nova MIB (vide quadro A1.2). Para fins de um melhor entendimento, NOVA-MIB é o nome do módulo da MIB e não o nome do arquivo. Esse nome pode ser visto dentro do arquivo MIB antes da palavra MODULE-IDENTITY (veja o quadro A1.3 que contém a parte inicial de um arquivo MIB).

```
# export MIBS=+NOVA-MIB
```

QUADRO A1.2 – Incluindo uma MIB à variável MIBS

```
…
ifMIB  MODULE-IDENTITY
       LAST-UPDATED "9611031355Z"
       ORGANIZATION "IETF Interfaces MIB Working Group"
       CONTACT-INFO
               "   Keith McCloghrie
                   Cisco Systems, Inc.
                   170 West Tasman Drive
                   San Jose, CA  95134-1706
                   US

                   408-526-5260
                   kzm@cisco.com"
       DESCRIPTION
               "The MIB module to describe generic objects for network
               interface sub-layers.  This MIB is an updated version of
               MIB-II's ifTable, and incorporates the extensions defined
               in RFC 1229."

       REVISION      "9602282155Z"
       DESCRIPTION
               "Revisions made by the Interfaces MIB WG."
       REVISION      "9311082155Z"
       DESCRIPTION
               "Initial revision, published as part of RFC 1573."
       ::= { mib-2 31 }
…
```

QUADRO A1.3 – Parte inicial da definição de uma MIB

Pode-se, também, carregar todas as MIBs (vide quadro A1.4) que estão armazenadas do diretório de repositórios de MIBs através da atribuição de ALL à variável MIBS.

```
# export MIBS=ALL
```

QUADRO A1.4 – Carrega todas as MIBs do repositório para a variável

Seguindo com a criação dos arquivos C, execute a ferramenta `mib2c` (vide quadro A1.5) passando como parâmetro o nome do módulo da MIB.

```
# mib2c NOVA-MIB
```

QUADRO A1.5 – Usando a ferramenta `mib2c` para gerar o esqueleto da MIB

A partir do comando especificado no quadro A1.5, será gerado o arquivo *header* (NOVA-MIB.h) e o arquivo C (NOVA-MIB.c).

O arquivo *header* C

Se o arquivo MIB é a definição do módulo para aplicações de gerenciamento de rede externas, então o arquivo *header* tem tradicionalmente servido com o mesmo propósito para o agente.

O arquivo simplesmente contém definições de rotinas visíveis publicamente e é completamente gerado pela ferramenta `mib2c`.

Cada arquivo *header* tem as seguintes funções protótipos:

- *extern void init_example (void);*
- *extern FindVarMethod var_example;*

Se o módulo incluir outras tabelas, ou outras coleções de variáveis que são implementadas em rotinas separadas, então essa segunda definição será repetida para cada uma delas.

Além disso, se algum dos objetos for READ/WRITE, isto é, pode ser configurado através de uma requisição SET, o arquivo *header* terá a definição da função protótipo para cada variável.

- *extern writeMethod write_varName;*

O arquivo C – estrutura principal da implementação

O trabalho principal da implementação do módulo está dentro do arquivo C. Muitos dos detalhes deste arquivo dependerão do módulo que está sendo implementado e isso pode apenas ser descrito pelo programador.

Entretanto, existe uma estrutura, bastante clara, que a implementação necessitará seguir, embora possa haver variações dependendo do estilo do módulo que está sendo implementado (se possuir tabelas ou ser uma coleção de valores individuais).

Apesar de apresentar variações, a modelagem como um todo da estrutura e os elementos que são comuns em todos os estilos seguem um padrão. Esse padrão está voltado principalmente em rotinas obrigatórias, definição do cabeçalho e código de inicialização variável.

Assim como no arquivo *header*, muitos dos elementos são gerados automaticamente pela ferramenta `mib2c`.

Definição do módulo

A definição do módulo é obtida através do resultado da definição de *mib2cstruct variableN* (onde N é o comprimento do maior sufixo da tabela).

Assim tem-se a estrutura do quadro A1.6.

```
struct variableN example_variables[]= {
```

QUADRO A1.6 – Definição da estrutura dos objetos do módulo

Cada entrada corresponde a um objeto na árvore MIB (ou uma coluna no caso de entradas de tabelas) e poderão ser listadas em ordem crescente de OID. Uma entrada consiste dos seguintes campos:

- Um número mágico;
- Um indicador de tipo;
- Um indicador de acesso;
- O nome da rotina usada para manipular essa entrada;
- O comprimento do sufixo OID usado;
- Uma matriz de inteiros especificando o sufixo.

Portanto uma entrada típica nesta estrutura pode ser vista no quadro A1.7, assim como a visão da declaração dos objetos dentro da estrutura.

```
 struct variable4 rmon_variables[] = {
 #define PROTOCOLDIRLASTCHANGE  1
 {PROTOCOLDIRLASTCHANGE, ASN_TIMETICKS, RONLY, var_rmon, 2, {11,1}},
 #define PROTOCOLDIRLOCALINDEX  6
 {PROTOCOLDIRLOCALINDEX, ASN_INTEGER, RONLY, var_protocolDirTable, 4,
{11,2,1,3}},
 #define PROTOCOLDIRDESCR 7
 {PROTOCOLDIRDESCR, ASN_OCTET_STR, RWRITE, var_protocolDirTable, 4,
{11,2,1,4}},
 ...
 }
```

QUADRO A1.7 – Entradas representando objetos da MIB

O quadro A1.8 mostra a maneira como é declarada a localização raiz do módulo dentro da árvore MIB.

```
oid example_variables_oid[] = { 1,3,6,1,2,1,16 }
```

QUADRO A1.8 – Declaração do OID raiz do módulo

Inicialização do módulo

Muitos módulos necessitam ser inicializados antes que eles possam começar a prover informações. A inicialização é feita através da *rotina init_<name>*, onde *<name>* é o nome do módulo.

Essa rotina é teoricamente opcional, mas na prática é requerida para registrar o módulo no agente principal. A função que realiza o registro é a REGISTER_MIB (vide quadro A1.9), estando a mesma dentro da rotina de inicialização.

```
REGISTER_MIB ("example",example_variables,variableN,example_variables_oid);
```

QUADRO A1.9 – Função que registra o módulo desenvolvido dentro do agente SNMP

Os parâmetros da função correspondem:

- "example" é usado para propósito de identificação, sendo normalmente o nome do módulo;
- example_variables é a estrutura que define as variáveis que estão sendo implementadas;
- variableN é o tipo usado para esta estrutura;
- example_variables_oid é a localização da raiz do módulo.

Manipulando variáveis

Outra rotina obrigatória é a que manipula uma requisição a uma variável. Essa rotina é a que aparece na estrutura *variableN*. Ela possui seis parâmetros, dos quais quatro são usados para passar informações sobre uma requisição (*struct variable *vp, oid *name, int *length, int exact*) e quatro são usados para retornar informações sobre a resposta (*oid *name, int *length, int *var_len, WriteMethod **write_method*). A função

também retorna um ponteiro para o dado atual da variável requisitada ou *NULL* se esse dado não está disponível por alguma razão.

Como pode ser notado, os parâmetros *name* e *length* são usados tanto para a requisição quanto para a resposta.

A primeira coisa que essa rotina necessita fazer é validar a requisição, para assegurar que ela se encontra dentro da seqüência implementada pelo módulo. Ao mesmo tempo, é possível recuperar algumas das informações necessárias para responder a consulta.

Após a validação, a rotina usa o campo número mágico do parâmetro *vp* para determinar quais das possíveis variáveis que estão sendo implementadas é a requisitada. Isso é feito através da declaração *switch*, o qual terá tantos *cases* quanto o número de variáveis que usam essa rotina como sendo a sua função manipuladora.

Módulos que não possuem tabelas

Nos módulos que não possuem tabelas, os mesmos são representados por uma coleção de variáveis independentes. Neste caso as variáveis são manipuladas por uma função que contém uma declaração de *switch* e um *case* para cada variável.

Relembrando que a rotina que manipula a variável necessita cobrir dois propósitos distintos: validação da requisição e condição da resposta, os quais são manipulados separadamente.

A validação da requisição é feita usando uma função padrão (*header_generic*). Seus parâmetros são os mesmos da função principal e são simplesmente passados diretamente. O retorno é uma variável inteiro a qual indica se a validação teve sucesso ou não. Caso a retorno falhar, a rotina principal retornará um valor *NULL*.

Para fins de visualização do código usado na validação veja quadro A1.10.

```
unsigned char *
var_example(struct variable *vp, oid *name, size_t *length,
 int exact, size_t *var_len, WriteMethod **write_method)
 {
  …
  if (header_generic(vp,name,length,exact,var_len,write_method)
  == MATCH_FAILED )
    return NULL;
…
```

QUADRO A1.10 – Código usado para a validação da requisição

A função *header_generic* pode ser usada como uma caixa preta, mas para se ter idéia do que ela faz, a mesma pode ser dividida em duas (ou possivelmente três) sub-funções:

- Verifica se a requisição é válida;
- Seta o OID para o retorno;
- Opcionalmente, seta valores *default* para os outros parâmetros de retorno.

Para validar a requisição, a rotina header, primeiramente, necessita ter o OID em questão para subseqüentemente comparar com a originalmente requisitada. Esse OID, que é a combinação do prefixo do OID (raiz do módulo) mais o sufixo da especificada variável, foi montado antes de chamar a função que manipula a variável e está

disponível através do campo name do parâmetro vp. Para uma variável escalar, o OID ficará completo acrescentando o identificador de instância 0.

O OID completo é construído em um vetor local chamado *newname* o qual foi criado para esse propósito. Veja o quadro A1.11 para visualizar o fragmento de código que realiza essa primeira ação.

```
int
header_generic (vp, name, length, exact, var_len, write_method)
{
 oid newname[MAX_NAME_LEN];
        memcpy((char *) newname, (char *) vp->name, (int) vp->namelen
       * sizeof (oid));
 newname[ vp->namelen ] = 0;
 ...
}
```

QUADRO A1.11 – Fragmento de código da função *header_generic* utilizado para montar o OID que será usado posteriormente para comparar com o OID requisitado

Após ter formado o OID, o mesmo pode ser comparado contra a variável especificada na requisição original, que está disponível através do parâmetro *name*. Essa comparação é feita usando a função *snmp_oid_compare*, a qual pega os dois OIDs (juntamente com seus respectivos comprimentos) e retornam –1, 0 ou 1 dependendo se o primeiro OID precede o segundo, se são iguais ou o primeiro está posteriormente ao segundo.

No caso de uma requisição “exata”, (GET/SET), então a requisição é apenas válida se os dois OIDs forem idênticos (retornar 0 da função *snmp_oid_compare*). No caso de uma requisição GETNEXT ou GETBULK será válido se o OID que está sendo considerado (*newname*) vier depois da requisição original (*name*), logo a função *snmp_oid_compare* deverá retornar -1.

Esse procedimento pode ser visto no fragmento de código do quadro A1.12.

```
result =snmp_oid_compare(name,*length,newname,(int)vp->namelen +1);
if ((exact && (result != 0))      //requisição GET falha
   || (!exact && (result >= 0)))  //requisição GETNEXT falha
  return (MATCH_FAILED);
```

QUADRO A1.12 – Fragmento de código que difere o tipo de requisição

Finalmente, tendo determinado que a requisição é válida, deve-se atualizar os parâmetros *name* e *length* para retornar o OID que está sendo processado. Além desses parâmetros, atribui-se valores *default* para outros dois parâmetros de retorno (quadro A1.13).

```
memcpy ((char *)name, (char *)newname, ((int)vp->namelen + 1) * sizeof (oid));
*length = vp->namelen + 1;
*write_method = 0;  //variável READ_ONLY
*var_len = sizeof (long);   //default para resultados inteiros
return (MATCH_SUCCEEDED);
```

QUADRO A1.13 – Fragmento de código que atribui os valores dos parâmetros do OID que se está processando

A função *header_generic* pode ser vista dentro do arquivo "*util_funcs.c"*, sendo o mesmo pertencente ao pacote `Net SNMP`.

A condição de resposta é o retorno do dado correspondente a variável requisitada, sendo manipulado dentro do *case* que manipula a variável. Relembrando, a distinção entre as variáveis é feita através do campo número mágico do parâmetro *vp*.

Tabelas simples

A natureza tabular dessas tabelas é aparentemente visível dentro da definição da MIB. Essas tabelas têm quatro características:

- É indexada por um simples valor inteiro;
- Tais índices vão de um até um determinado máximo;
- Todos os índices da seqüência são válidos;
- O dado para um particular índice pode ser recuperado diretamente.

Se alguma destas condições falhar, então esta tabela não é uma tabela simples, logo esta técnica não poderá ser aplicada.

Caso se usou a ferramenta `mib2c` para gerar os arquivos C, todas as tabelas são assumidas por serem simples. Neste caso, aquelas tabelas que não forem simples deverão ser remodeladas.

Assim como as variáveis escalares, as rotinas que manipulam as variáveis das tabelas necessitam prover a validação da requisição e a recuperação do dado.

A validação da requisição é provida pela função *header_simple_table*. Essa função possui os mesmos parâmetros da função *header_generic* com a adição do valor do índice máximo. Esse valor deverá ser alterado caso foi usado à ferramenta `mib2c`, o qual apresenta a especificação "*dummy*" em seu lugar. Além disso, esse valor poderá variar durante a execução do módulo pelo fato de que o valor do índice máximo é obtido assim que uma requisição é solicitada. O retorno da função é da mesma forma da função *header_generic*.

A visualização da chamada para a validação pode ser visto no quadro A1.14.

```
unsigned char *
var_example_table(struct variable *vp, oid *name, size_t *length,
            int exact, size_t *var_len, WriteMethod **write_method)
{
 …
 if (header_simple_table(vp,name,length,exact,var_len,
     write_method,MaxIndex) == MATCH_FAILED )
return NULL;
…
```

QUADRO A1.14 – Fragmento de código mostrando a chamada para a validação da requisição

Com respeito à recuperação do dado, as tabelas simples apresentam a mesma estrutura das funções que manipulam as variáveis escalares, isto é, cada variável da tabela é um *case* dentro da declaração *switch*.

Tabelas gerais

Quando se fala de tabelas gerais, se refere às tabelas que são indexadas por valores diferentes de um simples inteiro (tais como um endereço IP), possuem um índice composto, o número máximo do índice não é facilmente determinável, nem todos os índices são válidos ou o necessário dado não está diretamente acessível.

Nestes casos, não existe uma rotina padrão, mas um algoritmo geral que deverá ser seguido. Esse algoritmo é o seguinte:

- Execute qualquer inicialização necessária;
- Percorra através das instâncias conhecidas recuperando o dado que pertence ao índice de cada uma, até que a desejada instância seja encontrada;
- Se nenhuma entrada é encontrada, retorna-se uma falha, caso contrário recupera-se o dado referente ao objeto.

Para uma requisição exata (GET e similar), identificar a desejada instância é simples. Necessita-se apenas construir o OID e verificar se ele corresponde ao OID requisitado. Construir o OID significa acrescentar ao OID passado como parâmetro *vp->name* (representa o OID raiz mais o sufixo do objeto) o índice da instância.

Já para o GETNEXT, a situação não é tão simples. Dependendo da ordenação dos campos índice da tabela, as entradas podem ser retornadas na mesma ordem que aparecem na tabela, no entanto isso não é garantido, acarretando em retornos de forma randômica. Para tanto, a tabela deve ser totalmente percorrida para cada requisição para determinar o correto sucessor.

O quadro A1.15 mostra a estrutura básica da função que manipula tabelas gerais.

```
//atribuição da variável vp->name para uma variável local
//Verifica a existência de entradas na tabela
//atribui os dados índice para a variável local;
if (exact) //requisição GET ou SET
 {
   //OID requisitado é idêntico ao montado a partir da entrada
   if (snmp_oid_compare(newname, 13, name, *length) == 0)
    {
     // salva o OID montado
     // obtém a entrada da tabela que possui os dados do objeto requisitado
     // suspende a procura na tabela
    }
 }
else // requisição é um GET NEXT
 {
   // se OID montado é maior do que o OID requisitado e, a variável
   // que armazena o OID mais correto não recebeu algum OID até o momento
   // ou se o OID montado precede a variável que contém o OID mais correto
  if ((snmp_oid_compare(newname, 13, name, *length) > 0) &&
     ((flag < 0) || (snmp_oid_compare(newname, 13, lowest, 13) < 0)))
   {
     // salva OID montado na variável que armazena o OID mais correto
     // obtém a entrada da tabela que possui os dados do objeto requisitado
   }
 }
```

QUADRO A1.15 – Estrutura básica da função que manipula tabelas gerais

Com respeito à recuperação do dado, as tabelas gerais apresentam a mesma estrutura das funções que manipulam as variáveis escalares, isto é, cada variável da tabela é um *case* dentro da declaração *switch*.

Como implementar uma variável configurável

O coração da manipulação de um SET é o parâmetro *write_method* da rotina que trata a variável. Esse parâmetro é um ponteiro para a rotina que manipula a variável em questão. Caso for usada a ferramenta `mib2c`, a mesma gerará uma rotina para cada variável configurável. Essa rotina pode ser declarada através do *template* visto no quadro A1.16.

```
int write_variable( int action,
                    u_char *var_val,
                    u_char var_val_type,
                    int var_val_len,
                    u_char *statP,
                    oid *name,
                    int name_len );
```

QUADRO A1.16 – *Template* da rotina que atende as variáveis configuráveis

O segundo, terceiro e quarto parâmetros provêem informações sobre o novo valor. São respectivamente o valor, o tipo e comprimento.

Os dois últimos parâmetros contêm o OID a ser configurado.

O valor que é retornado pela rotina é simplesmente uma indicação de que o corrente estágio do SET teve sucesso ou não. Essa rotina possui a responsabilidade de verificar se o OID provido é o OID que a rotina pode manipular, se o valor requisitado tem o tipo correto esperado pelo objeto e se o valor requisitado está em conformidade com o objeto.

Há dois parâmetros que restam a ser explicados. O quinto parâmetro *statP*, é o valor que poderá ser retornado de uma requisição GET para essa particular variável. Ele pode ser usado para verificar se o novo valor requisitado está consistente com o atual estado, mas o principal uso é para informar que uma nova entrada na tabela está sendo criada. No entanto, este parâmetro pode ser normalmente ignorado.

O último parâmetro é o *action*. Para entender este parâmetro, é necessário saber um pouco sobre como os SETs são implementados.

Um objeto ao receber uma requisição SET é manipulado em sucessão, isto é, se uma verificação falhar, a rotina deve ser capaz de retornar quaisquer mudanças feitas. Quando se fala em sucessão, quer dizer que a rotina é chamada mais de uma vez e a mesma deve ser capaz de executar uma determinada ação dependendo em que ponto do processo se está. Esse ponto é determinado pelo parâmetro *action*, o qual apresenta três fases básicas:

- RESERVE é usado para verificar a sintaxe de todas as variáveis fornecidas (os valores atribuídos são sensatos e coerentes) e alocar quaisquer recursos requeridos para executar o SET. Se algumas dessas chamadas falhar, as rotinas de escrita são chamadas novamente com a ação FREE, para liberar qualquer recurso que tenha sido alocado.
- Caso a fase RESERVE foi completada com sucesso, o próximo estágio é indicado pela ação ACTION, o qual é usado para implementar a operação de configuração. Se a fase ACTION falhar, então a lista de rotinas é chamada novamente com a ação UNDO. Essa ação reconfigura o valor mudado para o valor anterior a mudança e após libera quaisquer recursos que tenham sido alocados.
- Uma vez que a fase ACTION terminou com sucesso, a fase final, representada pela ação COMMIT, pode ser executada. Nessa fase, as variáveis temporárias

que foram manipuladas são definitivamente gravadas e, por conseguinte libera-se os recursos alocados.

Compilando e acrescentando o novo módulo ao agente

Para que o agente possa gerenciar o novo dispositivo, existem dois caminhos que podem ser escolhidos. Um deles é recompilar o agente `Net SNMP` novamente com a inclusão do novo módulo (código desenvolvido). Para recompilar o *snmpd*, deve-se executar novamente os passos seguidos na instalação do agente. A única diferença encontra-se na execução do script configure, que agora vira acompanhado de um parâmetro que indicara a inclusão de um novo modulo (veja quadro A1.17).

```
# ./configure --with-mib-mudules="NOME MODULO"
```

QUADRO A1.17 – Acrescentando um módulo ao agente SNMP para operar estaticamente

A outra possibilidade e pode ser considerada a mais flexível, é a utilização da declaração *dlmod* dentro do arquivo de configuração do agente ("snmpd.conf"). A partir dessa forma de inclusão de módulo, o agente é estendido dinamicamente por um objeto compartilhado. No entanto para operar corretamente, o módulo criado deve ser compilado com a extensão ".so". A única configuração no agente é o arquivo "snmpd.conf" como comentado anteriormente e essa configuração pode ser vista no quadro A1.18.

```
dlmod nome_do_módulo caminho/para/o/arquivo/nome_do_arquivo.so
```

QUADRO A1.18 – Acrescentando um módulo ao agente SNMP de para operar dinamicamente

Anexo 2 Instalação do Agente RMON2

Para a instalação do agente RMON2 desenvolvido, deve-se primeiramente intalar os software que compreendem a estrutura base do mesmo. Após a instalação da estrutura base, instala-se o agente propriamente dito.

O agente pode ser obtido por *download*, assim como os softwares que fazem parte do mesmo, por WWW em http://penta2.ufrgs.br/agent_rmon2/agent_rmon2-install.tar.gz.

De posse do arquivo, descompacte-o no diretório "/tmp" do Linux (vide quadro A2.1).

```
# cd /tmp
# tar -xzvf /caminho/do/arquivo/agent_rmon2.tar.gz
```

QUADRO A2.1 – Descompactando a implementação do Agente RMON2

Entre no diretório "agent_rmon2-install", criado pela descompactação, e execute os *scripts* conforme a ordem de numeração:

- *./1-libcap.sh* (para instalar a biblioteca `Libpcap`);
- *./2-ucd-snmp.sh* (para instalar o pacote `Net SNMP`);
- *./3-mysql.sh* (para instalar o banco de dados `MySql`);
- *./4-agent_rmon2.sh* (para instalar o Agente RMON2).

Caso algum dos softwares já esteja instalado na estação, o mesmo não precisa ser instalado novamente. Basta apenas verificar alguns parâmentros.

Se o `MySql` já estiver instalado na estação, o mesmo não precisa ser novamente instalado, basta alterar ou verificar o caminho de onde ele está instalado no *script* que instala o Agente RMON2 e alterar os parâmetros (HOST, USER e PASS) do arquivo "conf_mysql.h" presente no diretório "arquivos-conf" para ficar em conformidade com a instalação existente.

Já no pacote `Net SNMP`, se o mesmo estiver instalado, precisa apenas acrescentar no final do arquivo "snmpd.conf" a seguinte linha:

- *dlmod rmon2 /usr/local/agent_rmon2/lib/rmon2.so*

Se todos os passos forem seguidos e não ocorrer erro durante a instalação, será criado um diretório chamado "agent_rmon2" em "/usr/local".

Configurando o Agente

No diretório "/usr/local/agent_rmon2/conf" existe um arquivo chamado "agent.conf" que serve para configurar os parâmetros iniciais do agente. A descrição do arquivo pode ser visto no quadro A2.2.

```
1  #token que especifica a interface que capturara os pacotes da rede
2  INTERFACE_CAP eth0
3  #token com o valor do indice da tabela interfaces da mib-II
4  # para a interface em questão
5  INDEX_INTERFACE 2
6  #-------------------------------------------------------
7  #Configurações para as tabelas control
8  OWNER Lucio
9  NLMAX 100
10 ALMAX 200
11 DURACTION_TOPN 300
12 REQUESTEDSIZE_TOPN 20
13 #-------------------------------------------------------
14 #Escolha as formas de estatísticas TopN (0 - não e 1 - sim)
15 NLMATRIXTOPN_PKTS 0
16 NLMATRIXTOPN_OCTETS 1
17 ALMATRIXTOPN_PKTSTERM 0
18 ALMATRIXTOPN_OCTETSTERM 1
19 ALMATRIXTOPN_PKTS 0
20 ALMATRIXTOPN_OCTETS 1
```

QUADRO A2.2 – Arquivo de configuração do Agente RMON2

A tabela "protocoldir" possui os encapsulamentos de protocolo que o agente pode monitorar. Para acrescentar um novo protocolo de aplicação para o mesmo monitorar, deve-se fazer através de uma *query* ao banco de dados. Para exemplificação, veja o quadro A2.3.

```
INSERT INTO protocoldir VALUES (1,2048,6,80,0,0,0,0,10,'ether2.ip.tcp.www-
http',1,1,3,3,'Lucio',1);
```

QUADRO A2.3 – *Query* para a inserção de um encapsulamento de protocolo à tabela "protocoldir"

Executando o Agente

Para inicializar o agente execute o *script* "agent_rmon2" (vide quadro A2.4) presente no diretório bin em "/usr/local/agent_rmon2/bin".

```
# /usr/local/agent_rmon2/bin/agent_rmon2 start
```

QUADRO A2.4 – Comando para inicializar o Agente RMON2

Bibliografia

[AGI2001] AGILENT TECHNOLOGIES. **Mixed Packet Size Throughput**. Disponível em: <http://advanced.comms.agilent.com/routertester/member/journal/1MxdPktSzThroughput.html>. Acesso em: 05 nov. 2001.

[CAR94] CARRILHO. J. A. **Um modelo para o gerenciamento do protocolo ftp baseado em domínios**. 1994. Dissertação (Mestrado) – DCC da Unicamp, Campinas.

[COM2001] COMBS, G. ET AL. **The Ethereal Network Analyzer**. Disponível em: <http://ethereal.zing.org>. Acesso em: 14 fev. 2001.

[COO99] COOK, C. et al. **An Introduction to Tivoli Enterprise**. USA: International Technical Support Organization, 1999. Disponível em: <http://www.redbooks.ibm.com>. Acesso em: 10 fev. 2001.

[DER99] DERI, L.; SUIN, S. Ntop: Beyond Ping and Traceroute. In: INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 1999, Zurich, Switzerland. **Proceedings...** Berlin: Springer-Verlag, 1999. (Lecture Notes in Computer Science, 1700).

[ENG98] ENGEL, Fred. Application Behavior and Statistics through RMON2. In IEEE INTERNATIONAL WORKSHOP ON SYSTEMS MANAGEMENT, 3., 1998, New Port, Rhode Island. **Proceedings...** [S.l. : s.n], 1998.

[GAS98] GASPARY, Luciano P. **Estudo do padrão RMON2**, 1998. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[GAS99a] GASPARY, Luciano P.; TAROUCO, Liane M. R. Distributed Applications and High-level Protocols Traffic Monitoring Based on RMON2 Accounting Mechanisms. In: JORNADAS ARGENTINAS DE INFORMÁTICA E INVESTIGACÍON OPERATIVA, (REDES), 1999, Buenos Aires, Argentine. **Proceedings...** Buenos Aires: Sadio, 1999.

[GAS99b] GASPARY, Luciano P.; TAROUCO, Liane M. R. Characterization and Measurements of Enterprise Network Traffic with RMON2. In: INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 1999, Zurich, Switzerland. **Proceedings...** Berlin: Springer-Verlag, 1999. (Lecture Notes in Computer Science, 1700).

[GAS99c] GASPARY, Luciano P.; TAROUCO, Liane M. R. Managing Users, Applications and Resources with RMON2. In: GLOBAL TELECOMMUNICATIONS CONFERENCE, SYMPOSIUM ON ENTERPRISE APPLICATIONS AND SERVICES, GLOBECOM, 1999, Rio de Janeiro, Brazil. **Proceedings...** Piscataway, USA: IEEE Press, USA, 1999.

[GAS2001a] GASPARY, Luciano P.; BALBINOT, Luis F.; STORCH, R.; WENDT, F.; TAROUCO, Liane M. R. Management of High-layer Protocols and Network Services Through an Extensible Multiagent-based Architecture. In: IEEE INTERNATIONAL CONFERENCE ON NETWORKING, ICN, 2001, Colmar, France. **Proceedings...** [S.l. : s.n.], 2001.

[GAS2001b] GASPARY, Luciano P.; BALBINOT, Luis F.; TAROUCO, Liane M. R. Monitoring High-Layer Protocol Behavior Using the Trace Architecture. In: LATIN AMERICAN NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, LANOMS, 2., 2001, Belo Horizonte, Brazil. **Proceedings…** Belo Horizonte: UFMG, 2001.

[JAC2001] JAKOBSON, V.; LERES, C.; MCCANNE, S. **Tcpdump**. Lawrence Berkeley National Labs. Disponível em: <http://www-nrg.ee.lbl.gov>. Acesso em: 14 fev. 2001.

[LEV99] LEVI, D.; MEYER, P.; STEWART, B. **SNMP Applications**. [S.l.]: IETF, 1999. (Request for Comments 2573).

[MAC2001] MCCANNE, S.; LERES, C.; JAKOBSON, V. **Packet Capture Library (Libpcap)**. Lawrence Berkeley National Labs. Disponível em: <http://www-nrg.ee.lbl.gov>. Acesso em: 14 fev. 2001.

[McC90] McCLOGHRIE, K.; ROSE, M. **Structure and Identification of Management Information for TCP/IP-based Internets**. [S.l.]: IETF, 1990. (Request for Comments 1155).

[McC91] McCLOGHRIE, K.; ROSE, M. **Management Information Base for Network Management of TCP/IP-based Internets: MIB-II**. [S.l.]: IETF, 1991. (Request for Comments 1213).

[MAL98] MALAN, Robert; JAHANIAN, Farnam. An Extensible Architecture for Network Protocol Performance Measurement. **Computer Communication Review**, New York, v.28, n.4, 1998. Trabalho apresentado na ACM Sigcomm Conference on Applications, technologics, Architectures, and Protocols for Computer Communication, 1998.

[MEN2001] MENEGHETTI, Edgar; GASPARY, Luciano P.,TAROUCO, Liane M. R. Uma Ferramenta de Monitoração Programável Voltada à Detecção de Intrusão. In: Conferencia Latinoamericana de Informática, CLEI, 27., 2001, Merida, Venezuela. **Proceedings...** Merida: Universidad de los Andes, 2001.

[MIL97] MILLER, Mark. **Managing Internetworks with SNMP**. 2nd ed. USA: M&T Books, 1997.

[MYS2001a] MySQL AB COMPANY. **MySql Database Server**. Disponível em: <http://www.mysql.com>. Acesso em: 05 jan. 2001.

[MYS2001b] MySQL AB COMPANY. **MySql Reference Manual**. Disponível em: <http://www.mysql.com/Downloads/Manual/manual.pdf>. Acesso em: 05 jan. 2001.

[NET2001] NET-SNMP PROJECT. **NET-SNMP Home Page**. Disponível em: <http://www.net-snmp.org>. Acesso em: 24 fev. 2001.

[PER99] PERKINS, David T. **RMON**: Remote Monitoring of SNMP-Managed LANs. USA: Prentice Hall, 1999.

[SCH90] SCHOFFSTALL, M. Fredor. **A Simple Network Management Protocol (SNMP)**. [S.l.]: IETF, 1990. (Request for Comments 1157).

[SIL95] SILVEIRA, C. K.; MADEIRA, E. R. M. Um Esquema para o Gerenciamento do Tráfego de Aplicações em Redes TCP/IP. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 13., 1995, Belo Horizonte. **Anais...** Belo Horizonte: SBC, 1998.

[STA98] STALLINGS, William. **SNMP, SNMPv2, SNMPv3, RMON and RMON2 - Practical Network Management**. 3rd ed. USA: Addision-Wesley, 1998.

[STE94] STEVENS, Richard W. **TCP/IP Illustrated**. USA: Addision-Wesley, 1994. v. 1.

[SUN2001] SUN MICROSYSTEMS, INC. **Snoop UNIX Man Pages (SunOS 5.6)**.

[TAN97] TANENBAUM, Andrew. **Redes de Computadores**. Rio de Janeiro: Campus, 1997.

[TAR94] TAROUCO, Liane M. R.; SILVA, Ana C. B. Uma Proposta para Gerência de Correio Eletrônico. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 12., 1994, Curitiba. **Anais...** Curitiba: SBC, 1994.

[TIR2001] TIRUMALA, Ajay; FERGUSON, Jim. **Iperf**. Disponível em: <http://dast.nlanr.net/Projects/Iperf/>. Acesso em: 30 out. 2001.

[WAL95] WALDBUSSER, S. **Remote Network Monitoring Management Information Base**. [S.l.]: IETF, 1995. (Request for Comments 1757).

[WAL97] WALDBUSSER, S. **Remote Network Monitoring Management Information Base Version 2**. [S.l.]: IETF, 1997. (Request for Comments 2021).