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

Gerenciamento Distribuído e Flexível de Protocolos de Alto Nível, Serviços e Aplicações em Redes de Computadores

por

LUCIANO PASCHOAL GASPARY

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

Profa. Dra. Liane Rockenbach Tarouco
Orientador

Porto Alegre, maio de 2002.

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Gaspary, Luciano Paschoal

Gerenciamento Distribuído e Flexível de Protocolos de Alto Nível, Serviços e Aplicações em Redes de Computadores / por Luciano Paschoal Gaspary. — Porto Alegre: PPGC da UFRGS, 2002.

139 f.: il.

Tese (doutorado) — Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2002. Orientador: Tarouco, Liane Rockenbach.

1. Gerenciamento de redes de computadores. 2. Internet. 3. Monitoração. I. Tarouco, Liane Rockenbach. II. Título.

A minha esposa,
Ana Cristina,
e aos meus pais,
João Carlos e Liane.

Agradecimentos

São muitas as pessoas que preciso agradecer no momento em que encerro a minha tese de doutorado e, ao mesmo tempo, completam-se dez anos desde o meu ingresso no Curso de Graduação em Ciência da Computação na Universidade Federal do Rio Grande do Sul.

Inicio com um agradecimento muito especial a minha orientadora, Profa. Dra. Liane Rockenbach Tarouco, por ter contribuído com idéias, ter me auxiliado a escolher as direções certas, ter incentivado publicações e, afetuosamente, ter festejado comigo os méritos alcançados.

Agradeço com a mesma intensidade a minha esposa, Ana Cristina, pelo amor, carinho e dedicação a mim dispensados nos oito anos de convivência. Agradeço, ainda, pela compreensão e paciência da Ana, que abdicou de muitos momentos de lazer para ficar ao meu lado.

Sem o suporte, a dedicação e o amor da minha família eu não teria chegado tão longe. Agradeço aos meus pais, João Carlos e Liane, por terem dedicado as suas vidas a minha educação integral e a de minha irmãs, Anelise e Cristine. A elas agradeço pelo carinho (e orgulho) que têm por mim. Merecem um abraço carinhoso minha avó, Maria Lygia, tios, primos, sogro, sogra e cunhados.

Muitas pessoas deram um pouco de si e contribuíram significativamente com esse trabalho. Agradeço, de coração, aos gigantes da Computação: Edgar Meneghetti, Eduardo Isaia Filho, Fabricio Wendt, José Roberto Gossler, Josué Klafke Sperb, Lucio Braga, Luis Felipe Balbinot, Marlom Alves Konrath, Roberto Storch e Rodrigo Trevisan.

E o que seria da tese (e da vida) se eu não tivesse contado com o apoio dos amigos? Felizmente, são muitos! Para não cometer a injustiça de esquecer alguém, agradeço a Lisandro Zambenedetti Granville em nome dos demais. Obrigado por ser, além de um grande amigo, um irmão.

Não menos importantes são os colegas que tenho na Universidade de Santa Cruz do Sul e na Universidade do Vale do Rio dos Sinos. Agradeço a todos pela convivência, pelos incentivos recebidos, pela compreensão nos momentos de ausência, pela curiosidade em saber se faltava muito para eu terminar a tese, ...

..., aí está ela!

"If I have seen farther than others,
it is because I stood on the shoulders of giants."
— Sir Isaac Newton

Sumário

[1] Esta seção é uma compilação do conteúdo dos artigos [GAS 99a, GAS 99b, GAS 99c], publicados pelo autor.

Lista de Abreviaturas

API	Application Programming Interface
APM	Application Performance Measurement
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
BNF	Bacchus-Naur Form
BPF	BSD Packet Filter
CCITT	Comité Consultatif International Téléphonique et Télégraphique
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
DHCP	Dynamic Host Configuration Protocol
DISMAN	Distributed Network Management
DLL	Dynamic Loadable Library
DNS	Domain Name System
FCAPS	Fault, Configuration, Accounting, Performance and Security
FTP	File Transfer Protocol
G-PTSL	Graphical PTSL
HMSC	High-Level Message Sequence Chart
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IIS	Internet Information Server
IP	Internet Protocol
IPX	Internetwork Packet Exchange
ISO	International Standards Organization
ITU	International Telecommunications Union
MAC	Medium Access Control
MIB	Management Information Base
MSC	Message Sequence Chart
NFS	Network File System

OID	Object Identifier
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PHP	PHP: Hypertext Preprocessor
POP	Post Office Protocol
POSIX	Portable Open System in Unix
PTSL	Protocol Trace Specification Language
QoS	Quality of Service
RFC	Request for Comments
RMON	Remote Network Monitoring
RSVP	Resource Reservation Protocol
RTFM	Realtime Traffic Flow Measurement
SDK	Software Development Kit
SMTP	Simple Mail Transport Protocol
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SRL	Simple Ruleset Language
T-PTSL	Textual PTSL
TCL	Tool Command Language
TCP	Transmission Control Protocol
ToS	Type of Service
TTCN	Tree & Tabular Combined Notation
UDP	User Datagram Protocol
URL	Unified Resource Locator
USM	User-based Security Model
VACM	View-based Access Control
WAN	Wide Area Network

Lista de Figuras

Lista de Tabelas

Resumo

As redes de computadores experimentam um grande crescimento não apenas em tamanho, mas também no número de serviços oferecidos e no número de protocolos de alto nível e aplicações que são executados sobre elas. Boa parte desses software (ex: ICQ e Napster), em geral, não está diretamente ligada a aplicações críticas, mas o seu uso não controlado pode degradar o desempenho da rede. Para que se possa medir o impacto dos mesmos sobre a infra-estrutura, mecanismos de gerência ligados à contabilização e caracterização de tráfego são desejáveis. Por outro lado, alguns protocolos, serviços e aplicações (ex: servidores DNS e Web) suportam aplicações críticas e precisam ser monitorados e gerenciados com maior atenção. Para essa classe de software de rede, a simples contabilização e caracterização de tráfego não é suficiente; tarefas de gerência como teste de serviços, detecção e manipulação de falhas, medição de desempenho e detecção de intrusão são importantes para garantir alta disponibilidade e eficiência da rede e aplicações. As ferramentas existentes para essa finalidade são, entre outros aspectos, (a) não integradas (necessidade de uma ferramenta para monitorar cada aplicação), (b) centralizadas (não oferecem suporte à distribuição de tarefas de gerenciamento) e (c) pouco flexíveis (dificuldade em gerenciar novos protocolos, serviços e aplicações). Nesse contexto, a tese propõe uma arquitetura, centrada na monitoração passiva em tempo real do tráfego de rede, para gerenciamento distribuído de protocolos de alto nível, serviços e aplicações em rede. Baseada na MIB (*Management Information Base*) Script do IETF (*Internet Engineering Task Force*), a arquitetura Trace oferece mecanismos para a delegação de tarefas de gerenciamento a gerentes intermediários, que interagem com agentes de monitoração e agentes de ação para executá-las. A tese propõe também PTSL (*Protocol Trace Specification Language*), uma linguagem gráfica/textual criada para permitir que gerentes de rede especifiquem as interações de protocolos (traços) que lhes interessam monitorar. As especificações são usadas pelos gerentes intermediários para programar os agentes de monitoração. Uma vez programados, esses agentes passam a monitorar a ocorrência dos traços. As informações obtidas são analisadas pelos gerentes intermediários, que podem requisitar a agentes de ação a execução de procedimentos (ex: *scripts* Perl), possibilitando a automação de diversas tarefas de gerenciamento. A arquitetura proposta é validada por um protótipo: a plataforma de gerenciamento Trace.

Palavras-chave: Gerenciamento de redes de computadores, Internet, monitoração.

TITLE: "DISTRIBUTED AND FLEXIBLE MANAGEMENT OF HIGH-LAYER PROTOCOLS, SERVICES AND NETWORKED APPLICATIONS"

Abstract

Computer networks experience a huge growth not only in size but also in the number of services offered, high-layer protocols and networked applications that flow over it. Some of these software (e.g. ICQ and Napster), in general, is not directly related to critical applications, but its uncontrolled usage may degrade network performance. To measure their impact on the infrastructure, management mechanisms related to traffic accounting and characterization are desirable. On the other side, some high-layer protocols, services and networked applications (e.g. DNS and Web) support critical applications and need to be carefully monitored and managed. To this type of network-dependent software traffic accounting and characterization is not enough; management tasks for service testing, fault detection and handling, performance measurement and intrusion detection are important to guarantee high availability and efficiency of the network and applications. Tools available for this purpose are (a) non-integrated (one need a different tool to monitor each application), (b) centralized (do not offer support to management task distribution) and (c) not very flexible (difficulty to manage new protocols, services and applications). In this context, this thesis proposes an architecture, based on passive, real-time network traffic monitoring, for distributed management of high-layer protocol, services and networked applications. Based on the IETF (*Internet Engineering Task Force*) Script MIB (*Management Information Base*), the Trace architecture provides mechanisms to allow a management station to delegate management tasks to mid-level managers that, in turn, interact with monitoring and action agents to execute these tasks. The thesis also proposes PTSL (*Protocol Trace Specification Language*), a graphical/textual language created to allow network managers to specify protocol interactions (traces) they want to be monitored. The specifications are used by mid-level managers to program monitoring agents. Once programmed, these agents start to monitor the occurrence of the trace. The information gathered are analyzed by the mid-level managers, which may delegate the execution of procedures (e.g. Perl scripts) to action agents, enabling the automation of several management tasks. The architecture proposed is validated by a prototype: the Trace management platform.

Keywords: Computer network management, Internet, monitoring.

1 Introdução

1.1 Motivação

A popularização das redes de computadores nos anos 90, impulsionada em grande parte pela Internet, provocou o surgimento de um variado e vasto conjunto de novas aplicações. Jogos para serem disputados em rede, como o Quake, aplicações para a realização de videoconferência (ex: NetMeeting e CUseeMe) e ferramentas para promover o intercâmbio de informações e arquivos, como o ICQ, são exemplos representativos desse novo cardápio de aplicações. Os protocolos clássicos que fazem parte da família TCP/IP, como o HTTP (*Hypertext Transfer Protocol*), o FTP (*File Transfer Protocol*), o SMTP (*Simple Mail Transport Protocol*) e o POP (*Post Office Protocol*), entre tantos outros, se disseminaram amplamente. As redes passaram a ser utilizadas como um meio alternativo para a realização de negócios por proporcionarem contato direto e eficiente das empresas com seus fornecedores e clientes, dando origem ao comércio eletrônico.

Os usuários de alguns desses protocolos, serviços e aplicações, dependendo do contexto em que os utilizam, não sofrem maiores prejuízos em caso de falhas que possam acontecer na infra-estrutura da rede. Por exemplo, o fato de um funcionário não conseguir momentaneamente acessar, via navegador *web*, um determinado *site*, dificilmente representará um problema para ele próprio ou para a organização onde trabalha. De forma similar, nada mais sério que uma *crise nervosa* ocorrerá se uma disputa em rede do jogo Quake for prejudicada pelo fato de os participantes precisarem aguardar um longo período de tempo para visualizar cada novo movimento realizado pelos rivais.

É inadmissível, por outro lado, imaginar aplicações críticas de uma organização, como as relacionadas ao comércio eletrônico, sendo alvo de falhas, de problemas associados a desempenho ou de ataques maliciosos. Dessas aplicações se espera (a) funcionamento 24 x 7 (vinte e quatro horas por dia, sete dias por semana), (b) tempo de resposta baixo – sob pena de se perder o cliente no meio de uma venda eletrônica por sua falta de paciência e (c) segurança, a ponto de garantir o sigilo dos dados informados pelo cliente para efetuar a compra (ex: número do cartão de crédito).

Os dois parágrafos anteriores retratam, de modo muito contrastante, que protocolos, serviços e aplicações possuem diferentes requisitos de funcionamento, dependendo do contexto em que se encontram. Para garantir esses requisitos *tarefas de gerenciamento* precisam ser permanentemente executadas. Embora aplicações como ICQ e Napster, em geral, não sejam críticas (não dependam de muitos requisitos para serem executadas), o seu uso não controlado pode acarretar em perda de desempenho da rede, prejudicando outros protocolos e aplicações. Para que se possa medir o impacto das mesmas sobre a infra-estrutura, tarefas de gerenciamento ligadas a contabilização e caracterização de tráfego são desejáveis. Por outro lado, alguns protocolos, serviços e aplicações (ex: servidores HTTP e DNS) podem suportar aplicações críticas e, assim, precisam ser monitorados e gerenciados com maior atenção [HEG 99]. Nesse caso, a simples contabilização e caracterização de tráfego não é suficiente; tarefas de gerenciamento como teste de serviços, detecção e manipulação de falhas, medição de desempenho e detecção de intrusão precisam ser, adicionalmente, executadas.

A seguir são relacionados alguns exemplos reais de tarefas de gerenciamento aplicáveis a protocolos de alto nível, serviços e aplicações em rede, que podem ser executados pelo gerente, com ou sem o auxílio de plataformas ou aplicações de gerenciamento específicas:

- *Monitoração da disponibilidade de serviços:* serviços como o DNS (*Domain Name System* e o DHCP *Dynamic Host Configuration Protocol*, essenciais ao funcionamento de uma rede TCP/IP, precisam estar permanentemente em execução; essa tarefa se resume em detectar quando algum desses serviços se encontra indisponível; a ação a ser realizada, imediatamente, é reiniciá-lo;

- *Contabilização dos acessos ao servidor HTTP:* consiste em medir o volume de acessos ao servidor *web*, não apenas os bem sucedidos, mas também os mal sucedidos e as tentativas não autorizadas; essas informações permitem ao gerente, por exemplo, (a) conhecer os horários mais críticos de acesso e providenciar a expansão ou reconfiguração do servidor HTTP para suportar mais acessos simultâneos, (b) contabilizar a ocorrência de problemas enfrentados por clientes HTTP no acesso a páginas mantidas no servidor e minimizar o problema revisando as páginas mantidas no *site* e (c) reconfigurar o servidor HTTP para não mais aceitar acessos da estação de onde partiram as tentativas de acesso não autorizadas;

- *Caracterização do tráfego de protocolos de alto nível:* constitui-se em determinar um perfil de utilização da rede, identificando quem e com que propósito, mais consome recursos da rede; essa tarefa pode ser fundamental, por exemplo, para a definição de uma política de uso da rede, para a descoberta de gargalos e para justificar a necessidade de expansão da infra-estrutura de comunicação.

- *Medição do tempo de resposta de uma aplicação em rede:* compreende a determinação do tempo de resposta percebido pelos usuários finais de uma aplicação; essa informação permite ao gerente de rede saber quando expandir o hardware da estação servidora (ex: com mais memória), quando o gargalo for a estação, ou reposicionar a estação para um ponto da rede em que não haja congestionamento, quando o alto tempo de resposta for ocasionado pelo atraso imposto pela própria rede;

- *Monitoração da segurança de serviços e aplicações em rede:* consiste em realizar a monitoração das estações que hospedam serviços e aplicações críticas para verificar se estão sendo vítimas de varredura de portas, de ataques de negação de serviço, entre outros; a ação associada, em caso afirmativo, pode ser a simples notificação do gerente, a reconfiguração do *firewall* para bloquear o acesso da estação de onde partiu o ataque ou, até mesmo, uma resposta imediata ao ataque sofrido [NOR 99].

1.2 Definição do problema

A definição, a programação, a execução e o acompanhamento de tarefas de gerenciamento como as supracitadas, em uma realidade onde o volume de protocolos, serviços e aplicações torna-se cada dia maior e os cenários passíveis de gerenciamento

evoluem cotidianamente, são o problema central abordado na tese. As dificuldades atuais em realizar gerenciamento de protocolos de alto nível, serviços e aplicações devem-se, principalmente, aos fatores comentados a seguir.

A maioria das plataformas ou aplicações de gerenciamento restringe-se a executar um conjunto específico de tarefas, ou seja, contempla apenas parcialmente as áreas funcionais de gerenciamento FCAPS (*Fault, Configuration, Accounting, Performance and Security*). No entanto, a rápida disseminação de protocolos, serviços e aplicações impede, pela falta de escalabilidade, que se utilize uma plataforma ou aplicação de gerenciamento diferente para realizar cada tarefa ou para monitorar cada uma das aplicações. Em organizações de pequeno e médio porte, é comum observar a utilização de um software de gerenciamento para monitorar o servidor HTTP, de outro para monitorar a disponibilidade dos principais serviços de rede, de um terceiro para monitorar a presença de intrusos na rede, apenas para mencionar alguns. Essa solução, porém, é precária para gerenciar ambientes maiores (ex: redes corporativas). Além de tornar custoso o processo de implantação e manutenção das tarefas de gerenciamento, a utilização de diferentes plataformas e aplicações de gerenciamento compromete a obtenção de uma visão única do estado dos protocolos, serviços e aplicações, e dificulta o intercâmbio e a correlação de informações obtidas por cada uma delas.

A segunda dificuldade em realizar gerenciamento de protocolos de alto nível, serviços e aplicações reside na incapacidade das plataformas ou aplicações de gerenciamento de operar eficientemente em grandes redes. Considerando que a quantidade de protocolos, serviços e aplicações passíveis de gerenciamento em uma rede corporativa tende a ser bastante elevada e que esses elementos, provavelmente, encontram-se dispersos pelos segmentos, torna-se indispensável a utilização de uma plataforma ou aplicação de gerenciamento que permita a distribuição de tarefas, fazendo com que boa parte do processamento seja realizado próximo aos elementos gerenciados.

A realidade, no entanto, é outra. A arquitetura de gerenciamento SNMP (*Simple Network Management Protocol*) ainda é centralizada, o que provoca (a) a utilização de uma boa parcela da banda da rede para trafegar informações entre a estação de gerenciamento e os nós gerenciados, (b) a sobrecarga da estação de gerenciamento com informações que ela, sozinha, deverá processar, entre outros aspectos amplamente discutidos [GAS 2000a, MAR 2000]. Embora as pesquisas para a criação de mecanismos que permitam a distribuição de tarefas de gerenciamento tenham iniciado em 1998, elas ainda não resultaram em padrões. Como conseqüência, as plataformas ou aplicações baseadas em SNMP conservam as características da arquitetura original. Algumas plataformas, como o HP Open View, implementaram soluções proprietárias para a distribuição de tarefas, mas sua utilização leva ao problema apontado anteriormente (a plataforma oferece condições para executar apenas um pequeno conjunto de tarefas de gerenciamento).

A terceira dificuldade encontra-se na pouca flexibilidade das plataformas e aplicações de gerenciamento existentes. A possibilidade de passar a gerenciar ou, simplesmente, monitorar um cenário não pensado originalmente obriga gerentes de rede a reprogramar manualmente aplicações de gerenciamento, a adquirir e implantar módulos-extra à plataforma em uso e, até mesmo, a aguardar atualizações de *firmware* distribuídas pelo fabricante – no caso de equipamentos de monitoração. Em um momento de efervescência de novos protocolos, serviços e aplicações, onde um conjunto cada vez maior de requisitos precisa ser atendido para satisfazer os

usuários, a limitação na agilidade com que esses elementos podem passar a ser gerenciados implica em ignorar, temporariamente, comportamentos significativos que podem ser resposta para problemas ligados a falhas, desempenho ou segurança. Dependendo do contexto, essa lacuna de tempo, em que o novo cenário deixou de ser gerenciado, pode representar prejuízos para a organização (ex: vulnerabilidade recém descoberta de um protocolo ou aplicação sendo explorada por um atacante).

1.3 Objetivos da tese

A pesquisa descrita na tese tem como objetivo principal *propor uma arquitetura de gerenciamento voltada a protocolos de alto nível, serviços e aplicações em rede*[1], que seja *integrada*, não se limitando a uma área de gerenciamento específica, *distribuída*, viabilizando a execução descentralizada de tarefas, e *rápida e facilmente adaptável* para monitorar cenários cada vez mais dinâmicos. São ainda objetivos da tese:

- *Implementar um protótipo para validar a arquitetura*: a implementação permite comprovar a exeqüibilidade da arquitetura proposta, indicando se a mesma pode ser utilizada em um ambiente real. O protótipo deve ser desenvolvido com base em software livre, propiciando (a) a sua utilização em diversos ambientes e (b) que outros trabalhos possam ser realizados e agregados a ele posteriormente;

- *Identificar cenários de aplicação da arquitetura*: enquanto no gerenciamento da infra-estrutura física da rede alguns tipos de anomalias são, hoje, facilmente identificadas por uma série de sintomas já bem conhecida e estudada [NUN 97], a área de pesquisa ligada a gerenciamento de protocolos, serviços e aplicações ainda não agrupou um conjunto significativo de situações que merecem monitoração ou gerenciamento. A identificação desses cenários e a definição de respectivas tarefas de gerenciamento, bem como a sua programação usando o protótipo, complementam a validação da arquitetura.

1.4 Organização da tese

O capítulo 2 posiciona a tese entre as diversas áreas de gerenciamento de redes existentes. Com base nesse referencial, algumas pesquisas relevantes sobre gerenciamento de protocolos de alto nível, serviços e aplicações em rede são apresentadas. O capítulo continua com uma análise das abordagens que norteiam essas pesquisas, procurando destacar suas potencialidades e limitações, e encerra com uma visão geral da abordagem usada na tese.

[1] Os termos protocolos de alto nível, serviços e aplicações em rede são usados ao longo da tese com as seguintes conotações: protocolos de alto nível são os protocolos padronizados que pertencem à família de protocolos TCP/IP como o IP, o TCP, o UDP e o HTTP; serviços são *software* que oferecem algum tipo de suporte ao funcionamento da rede (ex: DNS e DHCP); aplicações em rede, por fim, correspondem aos protocolos de alto nível proprietários, como o ICQ e o Napster, ou aos protocolos específicos de uma organização. É importante destacar que existe, em alguns momentos, sobreposição desses conceitos. Esse é o caso, por exemplo, do DNS que tanto pode ser enquadrado como protocolo de alto nível quanto como serviço.

No capítulo 3 é apresentada a linguagem PTSL (*Protocol Trace Specification Language*), proposta para permitir a representação de interações de protocolos de alto nível. O capítulo inicia apresentando características desejadas para a linguagem, passando, em seguida, para um rápido levantamento de formalismos considerados no projeto da mesma. O capítulo prossegue com a apresentação das notações gráfica e textual de PTSL, finalizando com exemplos de especificação e uma análise da linguagem proposta.

No capítulo 4 a arquitetura de gerenciamento Trace é detalhada. Num primeiro momento são abordadas as decisões de projeto consideradas na proposta da arquitetura. Em seguida, seus componentes e as relações estabelecidas entre eles são detalhados. Por fim são apresentados um exemplo de uso da arquitetura e uma análise da mesma.

O capítulo 5 apresenta o protótipo construído com base na arquitetura: a plataforma de gerenciamento Trace.

O capítulo 6 fornece um sumário do trabalho apresentado nessa tese, destacando suas contribuições. O capítulo inclui ainda algumas considerações finais e apresenta idéias de trabalhos futuros.

2 O Estado da Arte

Neste capítulo são apresentadas algumas pesquisas que representam o estado da arte em gerenciamento de protocolos de alto nível e serviços de rede. A seção 2.1 apresenta o padrão RMON2 (*Remote Network Monitoring Management Information Base Version 2*). A seção 2.2 aborda a aplicação de gerenciamento ntop. Na seção 2.3 a arquitetura RTFM (*Realtime Traffic Flow Measurement*) e a aplicação de gerenciamento NeTraMet são descritas. A seção 2.4 abre espaço para a MIB APM (*Application Performance Measurement Management Information Base*). Na seção 2.5 algumas plataformas e aplicações comerciais de gerenciamento são comentadas. A seção 2.6 sintetiza as pesquisas apresentadas e ressalta suas limitações técnicas, complementando a definição do problema discutida na seção 1.2. Por fim, a seção 2.7 introduz a abordagem proposta na tese. Para o leitor já familiarizado com os tópicos abordados nas seções 2.1 a 2.5, sugere-se avançar diretamente para a seção 2.6.

2.1 O padrão RMON2 do IETF[1]

Nos últimos anos o padrão SNMP e a especificação da MIB-II têm se apresentado como o mecanismo dominante para o gerenciamento de redes IP. Agentes de software presentes nesses equipamentos coletam informações sobre o tráfego de rede, incluindo estatísticas como o número de pacotes recebidos e enviados. Para determinar o volume e o comportamento do tráfego da rede como um todo, é preciso coletar e agregar as informações obtidas de cada dispositivo dessa rede. Esse método, além de ser muito oneroso, uma vez que gera elevado tráfego de gerenciamento, apresenta resultados bastante imprecisos [ENG 98].

Segundo Stallings [STA 96], a maior contribuição ao conjunto de padrões SNMP são as especificações das MIBs RMON [WAL 95] e RMON2 [WAL 97, GAS 98]. Enquanto a MIB RMON (*Remote Network Monitoring*) 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.

Os esforços para a padronização da MIB RMON iniciaram em 1990 com a criação do grupo de trabalho RMON do IETF. O padrão proposto através da RFC (*Request for Comments*) 1271 foi publicado em novembro de 1991. A primeira RFC estava voltada especificamente a redes Ethernet. Mais tarde, em 1993, o grupo de trabalho propôs extensões para redes Token Ring através da RFC 1513. Devido ao crescente interesse do mercado, logo diversos fabricantes passaram a implementar soluções considerando o futuro padrão. Em 1995, com a sedimentação e a interoperabilidade das implementações existentes até então, a MIB RMON foi padronizada (RFC 1757) [WAL 95].

As soluções RMON operam de acordo com o paradigma cliente-servidor. Cliente é a aplicação executada na estação de gerenciamento. Servidores, ou agentes, são os dispositivos utilizados para a monitoração. Distribuídos nas redes remotas, os

[1] Esta seção é uma compilação do conteúdo dos artigos [GAS 99a, GAS 99b, GAS 99c], publicados pelo autor.

agentes armazenam informações definidas na MIB RMON, que são obtidas a partir da análise de todos os pacotes que trafegam no segmento de rede onde se encontram. Desse modo, reduz-se para, no máximo, um o número de agentes, por segmento, que precisa ser consultado pela estação de gerenciamento para reunir informações sobre o tráfego de toda a rede. Os agentes RMON não residem apenas nos dispositivos dedicados à tarefa de monitoração, também conhecidos como *probes*, podendo ser encontrados em outros equipamentos de rede como *hubs*, *switches* e roteadores. A aplicação de gerenciamento e os agentes se comunicam através do protocolo SNMP.

Além de oferecer um rico conjunto de estatísticas ligado ao nível de enlace, como número de colisões e de pacotes *broadcast*, a MIB RMON inovou:

- ao possibilitar que os agentes criem e armazenem históricos de valores de objetos, contribuindo tanto para a redução de tráfego SNMP na rede quanto para a diminuição do processamento realizado pela estação de gerenciamento,
- ao permitir a configuração de alarmes e eventos que são disparados pelos agentes quando o valor de variáveis da MIB ultrapassa limiares pré-estabelecidos, proporcionando a realização de gerenciamento pró-ativo, e
- ao viabilizar a filtragem, segundo critérios determinados pelo gerente, e a captura de pacotes para análise.

Em 1994 iniciaram-se os trabalhos para estender a especificação da MIB RMON para incluir funcionalidade de monitoração de tráfego de protocolos de níveis superiores à sub-camada MAC (*Medium Access Control*) do nível de enlace. Esse trabalho, conhecido como RMON2, resultou na criação das RFCs 2021 [WAL 97] e 2024 [BIE 2000], em janeiro de 1997. Além de herdar as características e funcionalidades da RMON, a MIB RMON2 define novos grupos:

- *protocol directory*: é um repositório que indica todos os encapsulamentos de protocolos que o *probe* é capaz de interpretar;
- *protocol distribution*: agrega estatísticas sobre o volume de tráfego gerado por cada protocolo, por segmento de rede local;
- *address map*: associa cada endereço de rede ao respectivo endereço MAC, armazenando-os em uma tabela;
- *network-layer host*: coleciona estatísticas sobre o volume de tráfego de entrada e saída das estações com base no endereço do nível de rede;
- *network-layer matrix*: provê estatísticas sobre o volume de tráfego entre pares de estações com base no endereço do nível de rede;
- *application-layer host*: agrega estatísticas sobre o volume de tráfego de entrada e saída das estações com base em endereços do nível de aplicação;
- *application-layer matrix*: coleciona estatísticas sobre o volume de tráfego entre pares de estações com base no endereço do nível de aplicação;
- *user history collection*: amostra periodicamente objetos especificados pelo gerente e armazena as informações coletadas em uma tabela;

- *probe configuration*: define parâmetros de configuração para *probes* RMON;
- *rmon conformance*: descreve requisitos de conformidade para a MIB RMON2.

As informações disponibilizadas pela MIB RMON2 auxiliam a gerência de negócios de uma rede, possibilitando aos gerentes ter visão do comportamento das aplicações e protocolos de alto nível sendo utilizados, da taxa de utilização dos recursos envolvidos e dos usuários que mais os consomem. Assim, passam a ter condições de redefinir o fluxo de tráfego da rede, buscando uma melhor utilização desses recursos, e de observar quem se comunica com quem e quais aplicações estão sendo utilizadas, permitindo o estabelecimento de políticas que garantirão a utilização adequada da rede. O detalhamento das informações que podem ser obtidas a partir da MIB RMON2, e do que fazer com elas, é apresentado a seguir. Além disso, uma abordagem de fornecimento de QoS (*Quality of Service*) para aplicações legadas em redes TCP/IP baseada em RMON2 é apresentada em [GRA 2000].

2.1.1 Estatísticas sobre as atividades dos usuários

Obter informações sobre como um usuário, ou até mesmo um departamento, utiliza a rede é essencial se o gerente está interessado em traçar o perfil de uso desta e em conhecer quem são os usuários potenciais, com quem estes mais se comunicam, tanto local quanto remotamente, e que aplicações executam. De posse dessas informações, o gerente passa a ter subsídios que lhe auxiliarão a entender se a utilização que os usuários fazem da rede vai ou não ao encontro dos interesses da organização.

Volume de acessos

Informações relativas ao volume de acessos realizados por um determinado usuário podem ser obtidas através de consultas ao grupo *network-layer host* da MIB RMON2. Esse grupo permite decodificar pacotes com base em seus endereços de rede. Como conseqüência, gerentes de rede podem observar além dos roteadores que interligam as sub-redes e identificar as reais estações que estão se comunicando [PER 98]. O grupo é formado por uma tabela de controle (*hlHostControl*) e uma tabela de dados (*nlHost*), como ilustra a figura 2.1.

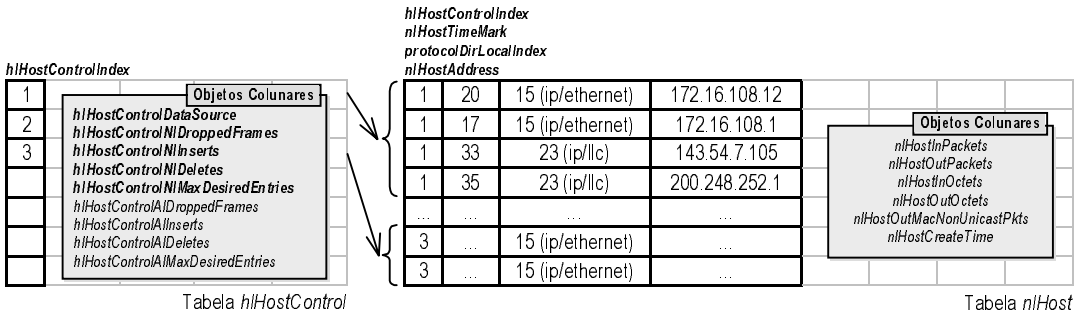


FIGURA 2.1 – Tabelas do grupo *network-layer host*

A tabela de controle possui uma entrada para cada interface (sub-rede) que está sendo monitorada. Cada entrada contabiliza informações como: número total de quadros recebidos pela interface que o *probe* opta por não contabilizar, número

de vezes que uma entrada foi adicionada ou removida da tabela de dados e número máximo admissível de entradas, para a interface em questão, na tabela de dados (*nlHost*) [WAL 97].

A função da tabela *nlHost*, por sua vez, é armazenar 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 identificado, uma entrada é adicionada à tabela *nlHost* [WAL 97].

Na figura 2.1 é possível verificar que a tabela de controle possui três entradas, indicando que o *probe* está analisando pacotes em três segmentos de rede distintos. No primeiro segmento foram identificados pacotes provenientes de quatro dispositivos, conforme pode ser observado na tabela *nlHost*. O segundo índice (*nlHostTimeMark*) dessa tabela indica o instante, em *timeticks*, em que a entrada foi criada ou atualizada pela última vez. A presença de um contador de tempo como índice da tabela de dados permite que o gerente de rede, no momento da consulta, informe que deseja receber apenas o valor dos objetos que sofreram modificação após um determinado instante. Como conseqüência, o tráfego de gerenciamento entre a estação central e os *probes* pode ser bastante reduzido. O terceiro índice, *protocolDirLocalIndex*, indica o tipo de encapsulamento observado. Os tipos de encapsulamento que o *probe* é capaz de analisar são catalogados no grupo *protocol directory*. Por fim, o quarto índice da tabela de dados indica o endereço, do nível de rede, do dispositivo identificado.

Um exemplo das informações obtidas com uma consulta ao grupo *network-layer host* é apresentado na tabela 2.1. Consultas periódicas a esse grupo permitem avaliar que usuários, e em que momentos do dia, mais acessam a rede. Abaixo, apresenta-se a fórmula (2.1) utilizada para calcular a taxa de utilização da rede (t.u.r), em porcentagem, por um determinado usuário (estação) num intervalo de tempo compreendido entre t_1 e t_2. *ifSpeed* indica a velocidade da tecnologia de rede onde o *probe* está atuando.

TABELA 2.1 – Consulta ao grupo *network-layer host*

HostAddress	*InPkts*	*OutPkts*	*InOctets*	*OutOctets*	*OutMacNonUnicastPkts*
172.16.108.12	1.000	345	80.345	25.367	33
172.16.108.1	2.350	733	97.334	33.292	125
143.54.7.105	5.930	299	112.445	5.293	0
200.248.252.1	100	30	49.238	3.777	12

$$\Sigma nlHostOctets_{t_i} = nlHostInOctets_{t_i} + nlHostOutOctets_{t_i}$$

$$t.u.r = \frac{(\Sigma nlHostOctets_{t_2} - \Sigma nlHostOctets_{t_1}) \times 8}{ifSpeed \times (t_2 - t_1)} \times 100 \qquad (2.1)$$

A figura 2.2 ilustra um exemplo de gráfico que pode ser gerado a partir de consultas ao grupo *network-layer host* e da aplicação da fórmula recém apresentada (2.1)[2].

[2]Os gráficos ilustrados ao longo desta seção são ilustrativos, baseados em dados hipotéticos.

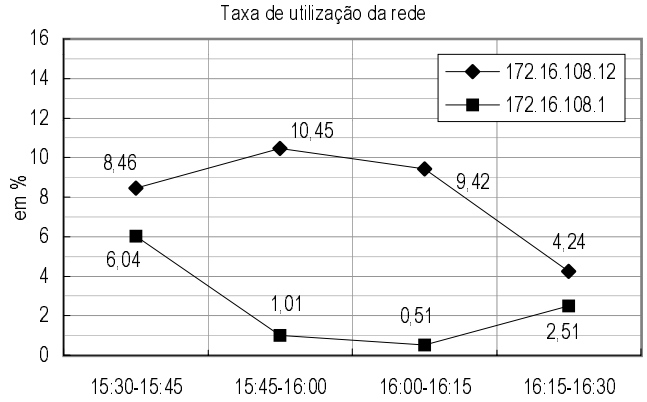


FIGURA 2.2 – Utilização da rede por 172.16.108.12 e 172.16.108.1

Aplicações e protocolos utilizados

Determinar padrões de utilização da rede requer do gerente um conhecimento pontual sobre os protocolos e aplicações que cada usuário executa, o instante em que isso ocorre e as estações com as quais esses usuários mais se comunicam, tanto local quanto remotamente [STA 96]. Essas informações podem ser obtidas através de consultas aos grupos *aplication-layer host* e *application-layer matrix*.

O grupo *application-layer host* é formado por uma tabela de controle (*hlHostControl*), que é a mesma do grupo *network-layer host*, e uma tabela de dados (*alHost*), conforme ilustra a figura 2.3. A tabela *alHost* permite ao gerente observar o tráfego de entrada e saída de um equipamento, considerando os protocolos de nível de aplicação. O termo nível de aplicação refere-se a todos os protocolos acima do nível de rede [WAL 97].

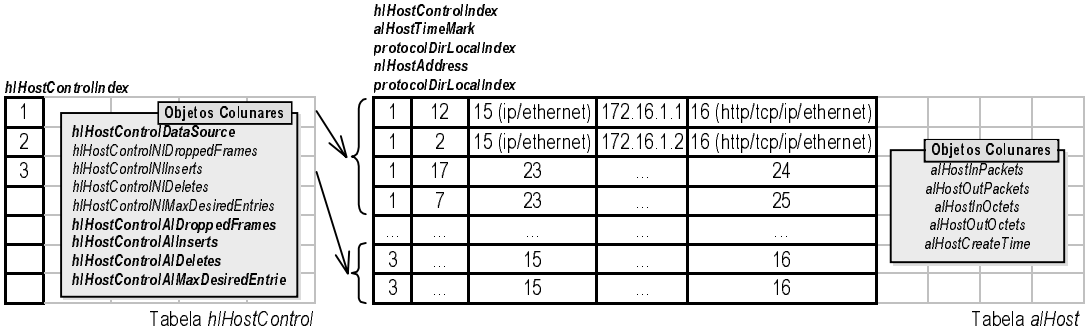


FIGURA 2.3 – Tabelas do grupo *application-layer host*

Há uma ou mais entradas na tabela *alHost* para cada protocolo do nível de aplicação descoberto. Essas entradas são organizadas também por endereços do nível de rede, de modo que é possível saber, por exemplo, o volume de tráfego HTTP proveniente ou destinado a uma determinada estação. A tabela é indexada por cinco objetos:

- *hlHostControlIndex*: segmento onde o *probe* observou o pacote;
- *alHostTimeMark*: filtro de tempo;
- *protocolDirLocalIndex*: identidade do protocolo do nível de rede;

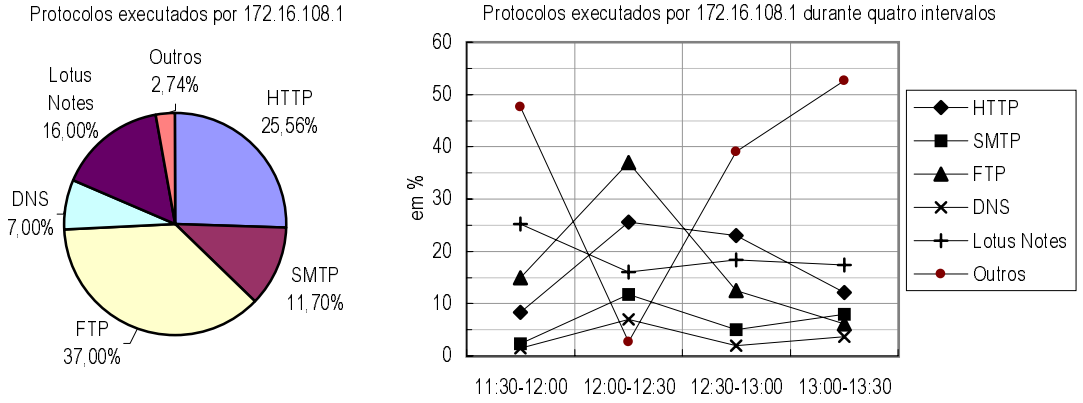
- *nlHostAddress*: endereço do nível de rede;
- *protocolDirLocalIndex*: identidade do protocolo do nível de aplicação.

A tabela 2.2 ilustra um exemplo de informações obtidas com uma consulta ao grupo *application-layer host*. Consultas periódicas a ele permitem determinar as aplicações e protocolos que os usuários mais utilizam, bem como os horários em que esses acessos ocorrem.

TABELA 2.2 – Consulta ao grupo *application-layer host*

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

De posse desse tipo de informações podem ser gerados gráficos que indiquem os protocolos e aplicações em uso por uma determinada estação e o volume de tráfego gerado por cada um deles. A figura 2.4a apresenta um exemplo de gráfico que pode ser obtido a partir de duas consultas ao *probe* (nos instantes t_1 e t_2). O gráfico apresenta a porcentagem de utilização de cada protocolo, pela estação 172.16.108.1, baseado no volume total de pacotes provenientes/destinados a ela. A fórmula para determinar a taxa de utilização de um protocolo do nível de aplicação (t.u.a) por uma estação, em um dado intervalo, é apresentada abaixo (2.2).



(a) (b)

FIGURA 2.4 – Protocolos utilizados por uma estação

$$\Sigma alHostOctets_{t_i prot_j} = alHostInOctets_{t_i prot_j} + alHostOutOctets_{t_i prot_j}$$

$$t.u.a = \frac{\Sigma alHostOctets_{t_2 prot} - \Sigma alHostOctets_{t_1 prot}}{\sum_{i=1}^{n} (\Sigma alHostOctets_{t_2 prot_i} - \Sigma alHostOctets_{t_1 prot_i})} * 100 \qquad (2.2)$$

O gráfico da figura 2.4b apresenta a taxa de utilização dos protocolos em quatro intervalos de tempo. Esse tipo de gráfico histórico auxilia o gerente a determinar

padrões de acesso dos usuários, uma vez que, a partir dele, tem condições de acompanhar os horários que esses usuários fazem mais uso da rede e quando eles utilizam determinados protocolos e aplicações. No gráfico, a taxa de utilização dos protocolos foi calculada com base na velocidade do segmento monitorado. Essa variante de gráfico permite visualizar o impacto que um determinado protocolo, utilizado por um determinado usuário, está causando na rede. A fórmula a ser utilizada, nesse caso, é a que segue (2.3).

$$\Sigma alHostOctets_{t_i} = alHostInOctets_{t_i} + alHostOutOctets_{t_i}$$

$$t.u.a = \frac{(\Sigma alHostOctets_{t_2} - \Sigma alHostOctets_{t_1}) \times 8}{ifSpeed \times (t_2 - t_1)} \times 100 \qquad (2.3)$$

Comunicações estabelecidas

Para ir além e conhecer com quem os usuários da rede se comunicam, e que protocolos estão envolvidos, pode-se recorrer ao grupo *aplication-layer matrix*. Esse grupo opera com a coleta de estatísticas em pares de estações, com base no endereço do nível de rede. É composto por várias tabelas; duas delas são ilustradas na figura 2.5.

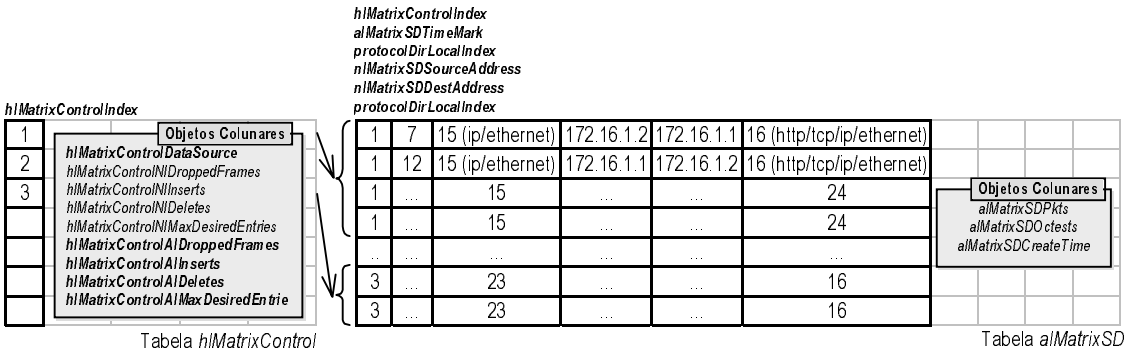


FIGURA 2.5 – Tabelas do grupo *application-layer matrix*

A tabela de controle (*hlMatrixControl*) é similar às tabelas de controle já apresentadas anteriormente. A tabela de dados (*alMatrixSD*) armazena informações sobre o volume de tráfego monitorado entre os pares de estações e é indexada pelos seguintes objetos [WAL 97]:

- *hlMatrixControlIndex*: segmento onde o *probe* observou o pacote;
- *alMatrixSDTimeMark*: filtro de tempo;
- *protocolDirLocalIndex*: identidade do protocolo do nível de rede;
- *nlMatrixSDSourceAddress*: endereço da estação que originou os pacotes;
- *nlMatrixSDDestAddress*: endereço da estação que recebeu os pacotes;
- *protocolDirLocalIndex*: identidade do protocolo do nível de aplicação.

A tabela 2.3 ilustra um exemplo de resultados que são obtidos ao se realizar uma consulta ao grupo *application-layer matrix*, mais precisamente à tabela *alMatrixSD*. Como é possível observar, ela contabiliza o tráfego da origem para o destino e do destino para a origem em duas entradas distintas.

TABELA 2.3 – Consulta ao grupo *application-layer matrix*

SDSourceAddress	*SDDestAddress*	*Protocol*	*SDPkts*	*SDOctets*
172.16.108.12	altavista.digital.com	16 (http/tcp/ip/ethernet)	15	578
altavista.digital.com	172.16.108.12	16 (http/tcp/ip/ethernet)	237	17.900
172.16.108.12	ftp.microsoft.com	17 (ftp/tcp/ip/ethernet)	29	2.193
ftp.microsoft.com	172.16.108.12	17 (ftp/tcp/ip/ethernet)	12.033	409.312
172.16.108.12	172.16.108.1	16 (http/tcp/ip/ethernet)	49	5.971
172.16.108.1	172.16.108.12	16 (http/tcp/ip/ethernet)	14.987	1.000.329

Através de consultas periódicas ao *probe* RMON2 é possível determinar com quem uma determinada estação está se comunicando, que protocolos estão sendo utilizados e o volume de tráfego gerado por eles.

A tabela 2.4 apresenta um exemplo de quadro demonstrativo das comunicações mantidas por uma estação de endereço 172.16.108.12 em um intervalo de tempo compreendido entre t_1 e t_2. As informações retornadas indicam que o usuário na máquina 172.16.108.12 está acessando, via um navegador Internet, o *site* de procura Alta Vista® e o servidor HTTP da própria empresa. Além disso, está recuperando, usando o protocolo FTP, um arquivo na Microsoft®.

TABELA 2.4 – Comunicações mantidas por 172.16.108.12

Endereço destino	Protocolo	Bytes
altavista.digital.com	HTTP	24.322
ftp.microsoft.com	FTP	34.967
172.16.108.1	HTTP	13.452

2.1.2 Perfil de utilização global da rede

Na seção anterior foram apresentados mecanismos para controlar e monitorar as atividades dos usuários de uma rede de forma individualizada. Em muitas situações, o gerente precisa ter acesso a informações que lhe indiquem o perfil de utilização da rede de forma globalizada, no âmbito de toda a organização ou a nível departamental. Esses dados são fundamentais para que o gerente possa, por exemplo, determinar que departamentos estão utilizando mais a rede, em quais horários a rede sofre com atrasos, e que aplicações e usuários mais contribuem para o agravamento desse problema.

A MIB RMON2 possui um grupo denominado *protocol distribution*, que contabiliza o número de octetos e pacotes monitorados pelo *probe* para cada um dos encapsulamentos de protocolos suportados por ele. É formado por duas tabelas: *protocolDistControl*, que controla a coleta de estatísticas básicas sobre todos os protocolos e *protocolDistStatus*, que armazena os dados coletados [WAL 97].

Cada entrada da tabela *protocolDistControl* refere-se a uma interface de rede única e controla um conjunto de entradas da tabela *protocolDistStats*, uma para

cada encapsulamento reconhecido na interface em questão. A tabela *protocolDistStats*, por sua vez, possui uma entrada para cada encapsulamento presente na tabela *protocolDir*, para a qual pelo menos um pacote tenha sido observado (vide figura 2.6).

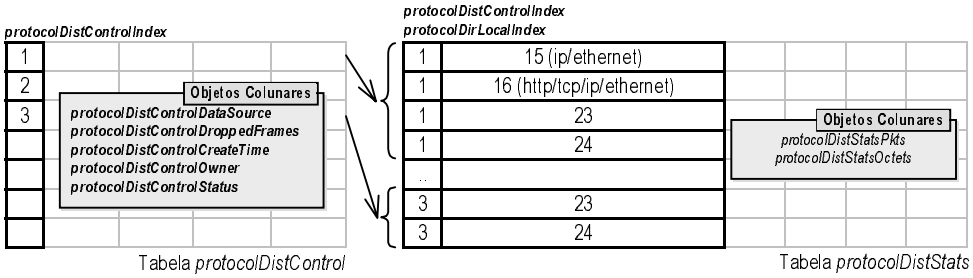


FIGURA 2.6 – Tabelas do grupo *protocol distribution*

A tabela 2.5 ilustra um exemplo de informações obtidas com uma consulta ao grupo *protocol distribution*. A realização de consultas periódicas a esse grupo viabiliza o acompanhamento da variância na taxa de utilização de protocolos no tempo, como pode ser observado na figura 2.7.

TABELA 2.5 – Consulta ao grupo *protocol distribution*

Protocol	*protocolDistStatsPkts*	*protocolDistStatsOctets*
15 (ip/ethernet)	3.417	1.034.587
16 (http/tcp/ip/ethernet)	1.644	459.100
23 (smtp/tcp/ip/ethernet)	1.290	345.923
24 (ftp/tcp/ip/ethernet)	483	229.564

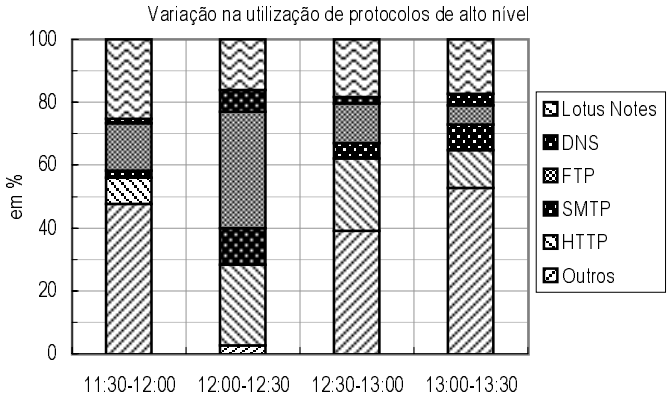


FIGURA 2.7 – Utilização global de protocolos e aplicações na rede

Determinar a taxa de utilização da rede por departamento requer que o gerente de rede conheça quais são as estações localizadas em cada um deles. O grupo a ser consultado é o *network-layer host*, já comentado anteriormente. A metodologia a ser empregada é a seguinte: o *probe* contabilizará o tráfego proveniente/oriundo a

qualquer estação identificada. Ao final de um intervalo de contabilização, calcula-se a taxa de utilização da rede por cada estação com a fórmula apresentada em 2.2 (página 25). Em seguida, seleciona-se as estações que fazem parte de cada departamento e, para cada um dos conjuntos, soma-se as taxas calculadas.

O requisito mínimo é que se utilize valores obtidos em dois instantes distintos para que se possa medir o volume de tráfego nesse intervalo. Nesse caso, a informação obtida refletirá a situação particular da rede naquele intervalo específico. Outra possibilidade é realizar as medições em vários momentos do dia, durante vários dias e contabilizar a média das taxas obtidas. Esse tipo de monitoração provê ao gerente uma posição mais precisa sobre a utilização da rede em cada departamento. A figura 2.8 ilustra um tipo de gráfico que pode ser gerado a partir dos dados obtidos.

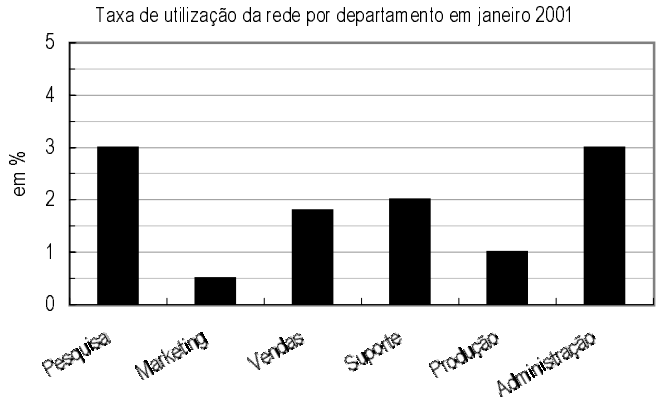


FIGURA 2.8 – Utilização da rede por departamento

O tipo de informação ilustrada acima pode ser determinante na alocação de custos, uma vez que pode indicar ao gerente onde os recursos de rede são mais utilizados e, por conseguinte, onde os investimentos com infra-estrutura devem ser prioritariamente realizados. A decisão de onde investir pode ser auxiliada, ainda, através da identificação precisa das estações que realizam a maior parte dos acessos. Esse grau de refinamento pode ser obtido através de consultas às tabelas *topN* do grupo *network-layer matrix*.

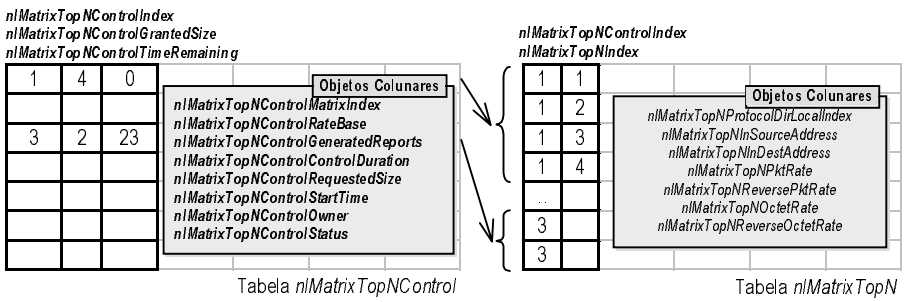


FIGURA 2.9 – Tabelas *topN* do grupo *network-layer matrix*

As tabelas *topN* agregam estatísticas e ordenam os pares de estações que estão se comunicando mais, segundo algum critério [WAL 97, PER 98]. Duas tabelas operacionalizam esse processo: a de controle (*nlMatrixTopNControl*) e a de dados (*nlMatrixTopN*). Cada entrada na tabela de controle referencia um relatório

em construção. Nela, são indicados: o critério de classificação para a criação das estatísticas (pacotes ou octetos), o intervalo de amostragem para a coleta de informações (em segundos), entre outros itens.

A figura 2.9 ilustra as duas tabelas mencionadas. É possível observar que há dois relatórios de estatísticas *topN* em andamento. O gerente, ao solicitar a geração de um relatório, indica, através da variável *nlMatrixTopNControlRequestedSize*, quantas entradas ele deseja que o relatório contenha. O *probe* avalia a condição de seus recursos como memória e processador e, se tiver condições, aceita a solicitação do usuário. Caso contrário, ele informa arbitrariamente quantas entradas ele admitirá (*nlMatrixTopNControlGrantedSize*). Na figura observa-se que o primeiro relatório possui quatro entradas.

É importante observar ainda que existe uma variável, *nlMatrixTopNControlTimeRemaining*, que indica o tempo que resta para o próximo relatório ficar pronto. Na figura acima o primeiro relatório acaba de ser atualizado (*TimeRemaining*=0). Tão logo o relatório seja finalizado, o *probe* automaticamente inicia um novo período de amostragem, indicado pela variável *nlMatrixTopNControlControlDuration*, para atualizar o relatório recém criado.

Na tabela 2.6 é apresentado um exemplo dos dados retornados a partir de uma consulta realizada à tabela *nlMatrixTopN*. Esses dados viabilizam a criação de gráficos, como o ilustrado na figura 2.10, que indicam quem são os usuários que mais consomem banda da rede.

TABELA 2.6 – Tabela *topN* ordenada pelo número de pacotes

Protocol	*SourceAddress*	*DestAddress*	*Pkt*	*ReversePkt*	*Octet*	*ReverseOctet*
15 (ip/ethernet)	172.16.108.12	172.16.108.1	213	32	40.065	6.023
15 (ip/ethernet)	172.16.108.45	172.16.108.23	156	17	23.913	2.194
15 (ip/ethernet)	172.16.106.25	200.248.252.1	89	29	12.882	6.745
15 (ip/ethernet)	172.16.109.7	172.16.108.1	67	13	5.294	968

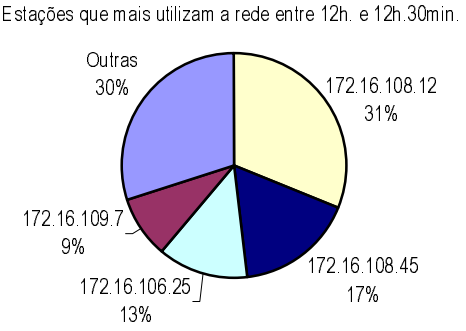


FIGURA 2.10 – Estações que mais utilizam a rede

2.1.3 Otimização da distribuição de usuários e recursos

Uma importante contribuição da MIB RMON2 é a possibilidade de verificar se usuários e recursos estão posicionados em segmentos apropriados com vistas ao confinamento de tráfego nos setores da organização [ENG 98]. O grupo que provê essas informações é o *application-layer matrix*, já apresentado na seção 2.1.1 deste

trabalho. As respostas obtidas com consultas a esse grupo permitem operacionalizar o procedimento ilustrado na figura 2.11.

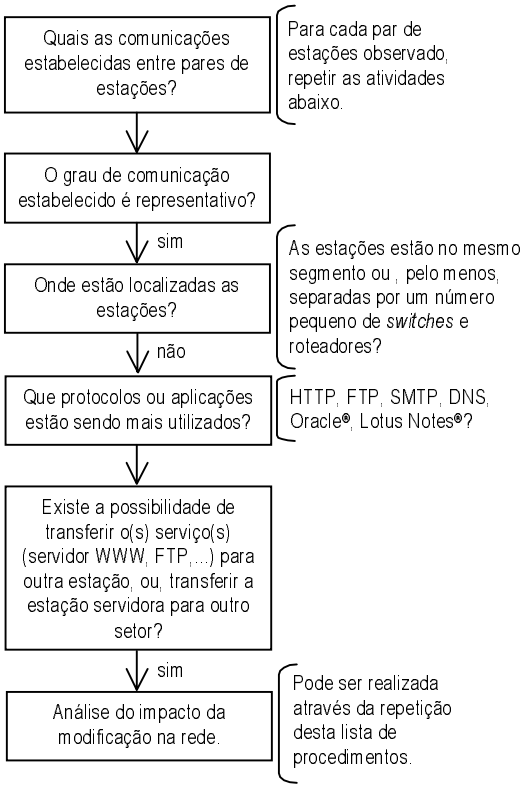


FIGURA 2.11 – Procedimento para realocação de usuários e recursos

Em alguns casos as estações estarão adequadamente posicionadas, minimizando o tráfego de dados em segmentos desnecessários, mas alguns dos recursos da rede estarão sobrecarregados, afetando os tempos de resposta de protocolos e aplicações. Quando isso ocorre, é importante a realização de balanceamento de carga. Nesse instante é importante que o gerente possa medir o grau de utilização de um determinado recurso. Para tal, ele pode utilizar mais uma vez as informações providas pelo grupo *application-layer matrix*.

A metodologia utilizada para determinar as atividades atuais em um determinado recurso é: o *probe* contabilizará o tráfego proveniente/oriundo a qualquer par de estações identificado. A monitoração pode ser realizada em dois intervalos de tempo, t_1 e t_2, ou periodicamente. Ao final de um intervalo de contabilização, seleciona-se todas as entradas que possuem como endereço fonte ou destino o recurso sendo inspecionado. Sobre as entradas selecionadas, realiza-se um novo agrupamento, agora, categorizando-as por protocolo. Em seguida, para cada agrupamento, faz-se o somatório dos octetos de entrada e de saída.

A tabela 2.7 ilustra esse cálculo: no recurso monitorado é observado tráfego HTTP, SMTP e Oracle, configurando a presença de três agrupamentos. No agrupamento HTTP é realizado o somatório do número de bytes de entrada (34.567) e saída (125.954) no instante t_1. Esse mesmo cálculo é repetido no instante t_2. O volume de utilização do protocolo HTTP no intervalo compreendido entre t_1 e t_2 é dado pela diferença dos somatórios nos dois instantes. O mesmo cálculo deve ser realizado com os outros agrupamentos.

TABELA 2.7 – Carga de atividades de um recurso

End. Origem	End. Destino	Protocolo	Num. bytes(t_1)	Num. bytes(t_2)	t_2-t_1
172.16.108.12	172.16.108.1	HTTP	34.567	192.224	
172.16.108.1	172.16.108.12	HTTP	125.954	864.297	
			160.521	1.060.521	900.000
172.16.109.5	172.16.108.1	SMTP	37.234	220.211	
172.16.108.1	172.16.109.5	SMTP	20.889	93.912	
			58.123	314.123	256.000
172.16.108.12	172.16.108.1	Oracle	45.082	341.090	
172.16.108.1	172.16.108.12	Oracle	23.781	193.744	
			68.863	534.834	465.971

Um exemplo de gráfico histórico sobre as atividades realizadas sobre um determinado recurso é apresentado na figura 2.12.

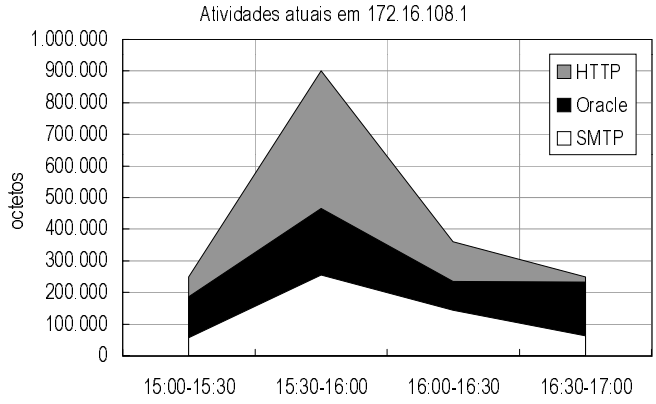


FIGURA 2.12 – Sobrecarga de protocolos e aplicações em uma estação

2.1.4 Gerenciamento de segurança

Segurança é um item essencial às redes corporativas. Atualmente, gerentes têm se cercado de mecanismos como *firewalls* para manter possíveis invasores longe dos dados estratégicos da empresa [PER 98].

A MIB RMON2 pode ser usada como uma ferramenta de auxílio à detecção de possíveis intrusos. Como já foi apresentado anteriormente, através de consultas permanentes ao grupo *application-layer matrix* é possível observar quais são os pares de estações que estão se comunicando (vide tabela 2.7). Já nesse ponto é possível que o gerente observe se há a presença de alguma estação desconhecida em contato com algum recurso (estação servidora, por exemplo) interno à organização. Se existe essa comunicação, é importante que o gerente verifique que protocolo está sendo utilizado. Dependendo da política de segurança da empresa, a tentativa de realização de telnet em uma estação, por exemplo, pode significar um ataque potencial.

O mesmo tipo de gráfico ilustrado na figura 2.12 pode ser criado destacando, neste caso, os diferentes usuários que estão acessando o recurso (vide figura 2.13).

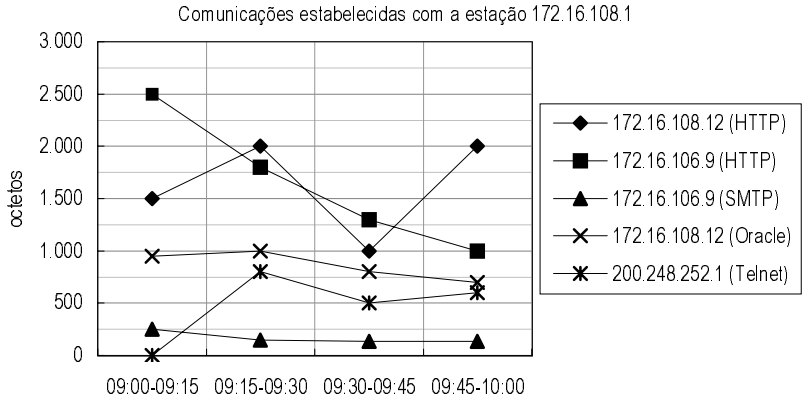


FIGURA 2.13 – Observação de usuários que acessam uma estação

O gráfico recém apresentado ilustra os acessos realizados a uma única estação. Esse tipo de informação é particularmente importante para estações estratégicas da empresa, onde residam servidores de banco de dados, de correio eletrônico, entre outros. Importante também é ter acesso a dados mais gerais que indiquem, para cada usuário observado na rede, os protocolos em uso por ele. Essas informações podem ser obtidas no grupo *application-layer host*, já apresentado na seção 2.1.1. A figura 2.14 ilustra um exemplo de gráfico que pode ser gerado com esses dados.

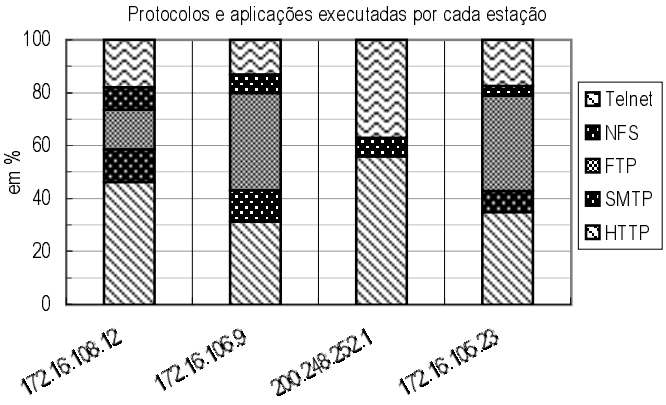


FIGURA 2.14 – Protocolos utilizados por cada estação da rede

2.2 A aplicação de gerenciamento ntop

Ntop, oficialmente apresentada em [DER 99], é uma aplicação de gerenciamento de código aberto, baseada na *web*, destinada à monitoração, medição e caracterização de tráfego de rede [NTO 2002]. Foi concebida para suprir necessidades de gerenciamento não atendidas pelas ferramentas que acompanham os sistemas operacionais, como *ping* e *traceroute*, ao mesmo tempo em que busca ser bem mais simples que as plataformas e aplicações de gerenciamento comerciais [DER 99]. Sua arquitetura está organizada em três módulos principais:

- *Capturador de pacotes*: baseado na biblioteca *libpcap*, o módulo de captura armazena temporariamente os pacotes capturados em uma fila; esse procedimento é realizado para que pacotes não sejam perdidos na ocorrência de rajadas de tráfego;

- *Analisador de pacotes capturados*: processa um pacote de cada vez. As informações sobre as estações são armazenadas em uma tabela *hash*; entradas ocupadas por estações que não enviam/recebem pacotes por um longo período de tempo são varridas e liberadas dessa tabela periodicamente para evitar a manipulação de tabelas muito grandes e o consumo de muita memória. O analisador utiliza um mecanismo de armazenamento semi-persistente para guardar informações como endereços IP (e respectivos endereços MAC) e informações sobre o sistema operacional em uso nas estações. O armazenamento das informações ligadas às estatísticas coletadas é realizado, de forma persistente, em um banco de dados SQL (*Structured Query Language*);

- *Visualizador de resultados*: existem duas formas de manipular a aplicação, o modo interativo, executado a partir de um terminal baseado em caracter, e o modo *web*, onde a ntop age como um servidor HTTP e permite que gerentes possam, remotamente, analisar estatísticas de tráfego através de um navegador Internet.

Uma característica interessante da aplicação é a possibilidade que ela oferece para a contabilização de fluxos definidos pelo gerente de rede. A filtragem de pacotes é baseada no BPF (*BSD Packet Filter*) [MCC 93], funcionalidade integrante da biblioteca *libpcap*. Os filtros BPF podem ser especificados através de expressões como as utilizadas pela ferramenta *tcpdump*, conforme ilustra a figura 2.15. Para um melhor desempenho, esses filtros são compilados e otimizados antes de seu armazenamento na aplicação ntop. Após ser iniciada, todos os filtros são aplicados a cada pacote capturado. Sempre que um pacote possui as características especificadas em um filtro, os respectivos contadores associados são atualizados. É importante ressaltar que o tempo de processamento de um pacote aumenta com o número e a complexidade dos filtros especificados.

```
tcpdump ip host ace and not helios
tcpdump 'tcp[13] & 3 != 0 and not src and dst net localnet'
tcpdump 'icmp[0] != 8 and icmp[0] != 0'
```

FIGURA 2.15 – Exemplos de filtros usados pela aplicação ntop

Para adicionar novas funcionalidades à aplicação é preciso desenvolver *plug-ins*, que devem ser implementados através de bibliotecas compartilhadas, ou DLLs (*Dynamic Loadable Libraries*) na terminologia Windows. Esses *plug-ins* precisam atender algumas condições pré-estabelecidas pela aplicação e ser armazenados em um diretório específico. Ao ser iniciada, ntop carrega os *plug-ins* seqüencialmente em ordem alfabética. Desenvolvedores podem usar esse mecanismo para estender o núcleo da aplicação ntop, definir visões alternativas para as informações de tráfego coletadas e implementar contadores de fluxos avançados que permitam realizar operações adicionais às de medição de tráfego.

A seguir são apresentados os cenários onde a aplicação ntop pode ser utilizada e um apanhado geral das informações que ela pode oferecer ao gerente de rede.

2.2.1 Medição de tráfego

Para Deri et. al. [DER 2000a], a medição de tráfego consiste em medir atividades de tráfego relevantes. A aplicação ntop realiza essas medições no segmento de rede onde se encontra, contabilizando cada pacote observado e associando estatísticas ao par remetente/destinatário. Assim, torna-se possível identificar todo o tráfego gerado por uma determinada estação. As informações armazenadas para cada uma são:

- *Dados enviados/recebidos*: tráfego total (volume e pacotes) gerado ou recebido, classificado de acordo com o protocolo de rede (ex: IP, IPX e AppleTalk) e, quando aplicável, com o protocolo do nível de aplicação (ex: FTP, HTTP e NFS);
- *Largura de banda utilizada*: utilização de banda atual, média e de pico;
- *IP Multicast*: total de tráfego *multicast* (volume e pacotes) gerado ou recebido;
- *Histórico das sessões TCP*: lista de sessões TCP ativas estabelecidas/aceitas e estatísticas de tráfego associadas;
- *Tráfego UDP*: quantidade total de tráfego UDP classificada por porta;
- *Serviços TCP/UDP usados*: lista de serviços baseados no protocolo IP oferecidos (ex: portas abertas e ativas) com a relação das últimas cinco estações que os utilizaram;
- *Sistema operacional*: sistema operacional utilizado, versão e atualizações instaladas;
- *Distribuição de tráfego*: tráfego local, de saída e de entrada na rede;
- *Distribuição de tráfego IP*: tráfego TCP *versus* UDP, distribuição relativa de protocolos IP de acordo com o nome da estação.

A aplicação ntop também fornece estatísticas globais de tráfego [DER 2000a], incluindo:

- *Distribuição de tráfego*: tráfego local, de saída e de entrada na rede;
- *Distribuição de pacotes*: número total de pacotes classificados por tamanho, tipo de endereço destino (ex: *unicast*, *broadcast* e *multicast*) e tráfego IP *versus* não IP;
- *Largura de banda utilizada*: utilização de banda atual, média e de pico;
- *Utilização e distribuição de protocolos*: distribuição do tráfego observado de acordo com o protocolo utilizado e origem/destino (local *versus* remoto);
- *Matriz de tráfego na sub-rede local*: tráfego monitorado entre cada par de estações na sub-rede;
- *Fluxos de rede*: estatísticas de tráfego de fluxos definidos pelo gerente.

2.2.2 Monitoração de tráfego

A habilidade de identificar situações onde o tráfego de rede não obedece a políticas especificadas ou excede determinados limiares constitui o que Deri et. al. denominam monitoração de tráfego. Em geral, gerentes especificam políticas a serem seguidas por todas as estações da rede, o que dificilmente acontece. Causas típicas desse mal comportamento estão relacionadas a, por exemplo, má configuração de sistemas operacionais, interfaces de rede e aplicações. Para auxiliar a identificação desses problemas, a aplicação ntop permite detectar:

- a utilização de endereços IP duplicados,
- interfaces de rede operando em modo *promíscuo*,
- a má configuração de aplicações, analisando o tráfego gerado por elas,
- o mal uso de serviços, através da identificação de estações que não utilizam *proxies* determinados,
- o mal uso de protocolos, através da identificação de estações que utilizam protocolos desnecessários,
- estações configuradas para agir como roteadores e
- a utilização excessiva de banda da rede.

2.2.3 Otimização e planejamento da rede

A não otimização da configuração das estações de uma rede pode influenciar negativamente o desempenho geral da mesma. Com as funcionalidades supracitadas ntop oferece ao gerente condições de identificar fontes potenciais de má utilização da banda da rede, provocada especialmente pelo uso de protocolos desnecessários e problemas de roteamento. Além disso, através da caracterização do tráfego e do entendimento de sua distribuição, é possível rever políticas que propiciem uma utilização mais inteligente dos recursos de rede.

2.2.4 Detecção de violações de segurança

A aplicação ntop oferece ainda mecanismos para rastrear ataques em andamento e identificar vulnerabilidades, incluindo *IP spoofing*, interfaces de rede em modo promíscuo, ataques de negação de serviço, cavalos de Tróia e varreduras de porta [DER 2000b].

Quando uma violação de segurança ou um aspecto ligado à má configuração da rede é identificado, ntop oferece mecanismos para gerar alarmes ao gerente (via e-mail, *traps* SNMP ou mensagens para telefones celulares) e executar ações, quando aplicáveis, para bloquear o ataque. Os registros mantidos no banco de dados da aplicação podem ser usados para entender o ataque e prevenir que outros semelhantes ocorram no futuro.

2.3 A aplicação de gerenciamento NeTraMet

NeTraMet é uma aplicação de código aberto destinada à medição de fluxos de tráfego [BRO 2001, NER 2002]. É a primeira implementação da arquitetura RTFM (*Realtime Traffic Flow Measurement*) [BRO 99a, BRO 99b, BRO 99c, BRO 99d, HAN 99], proposta pelo grupo de trabalho homônimo do IETF. NeTraMet possui três componentes:

- *Meters*: estações que medem o tráfego nos segmentos onde se encontram;
- *Meter readers*: estações que recuperam, dos *meters*, as informações coletadas usando o protocolo SNMP;
- *Managers*: estações a partir das quais se configuram (a) os *meters*, informando os fluxos que eles devem medir, e (b) os *meter readers*, determinando de que *meters* serão recuperadas as informações coletadas e com que freqüência.

Os *meters* são agentes SNMP que implementam a MIB *Meter* [BRO 99a], ou seja, qualquer gerente SNMP, conhecendo a comunidade de leitura ou escrita, pode interagir com eles. No entanto, a MIB *Meter* apresenta objetos relacionados à configuração do *meter* e ao armazenamento de informações sobre os fluxos observados, de modo que, para um bom aproveitamento das informações disponíveis na MIB, uma aplicação de gerenciamento específica é necessária.

NeMaC e nifty, ferramentas que acumulam as funções de *manager* e *meter reader*, são exemplos dessas aplicações. Outras ferramentas fazem parte da aplicação NeTraMet: (a) o *meter*, também chamado de NeTraMet, (b) o compilador srl, para auxiliar na criação de arquivos de configuração dos *meters* e (c) alguns utilitários (ex: fd_filter e fd_extract).

2.3.1 Regras de filtragem para contabilização de fluxos

Os fluxos a serem contabilizados por um *meter* são especificados através de regras de filtragem usando a linguagem SRL (*Simple Ruleset Language*) [BRO 99d]. A linguagem define um conjunto de aproximadamente quarenta atributos que podem ser usados para descrever essas regras. Os mais importantes são os atributos que especificam endereços de uma determinada camada de rede: *SourcePeerAddress* e *DestPeerAddress* (endereços IP origem e destino), *SourceTransAddress* e *DestTransAddress* (portas TCP/UDP local e remota) e *SourceTransType* (protocolo sobre o IP como ICMP, TCP e UDP). Esses cinco atributos são suficientes para identificar fluxos TCP individuais. Se tal granularidade não for desejada então menos atributos (ex: apenas *SourcePeerAddress* e *DestPeerAddress*) podem ser utilizados, produzindo um número menor de fluxos maiores.

Quando um *meter* observa um pacote, ele extrai desse pacote todos os atributos previstos pela arquitetura RTFM e, em seguida, executa as regras de filtragem definidas pelo gerente. As regras são executadas seqüencialmente. O comando *save* especifica os atributos (do pacote) que devem ser salvos. Nas regras do exemplo apresentado na figura 2.16 serão salvos os endereços IP origem e destino de cada pacote. O comando *count* faz com que o *meter* pare o processamento, criando uma chave para o fluxo. Essa chave é composta pelos valores dos atributos salvos (voltando ao exemplo, esses valores poderiam ser 192.168.1.1 e 192.168.1.2). A chave é

utilizada para localizar o fluxo na tabela de fluxos do *meter*. Se o fluxo já existir, os contadores de pacotes e bytes associados a ele são incrementados, caso contrário uma nova entrada deve ser adicionada à tabela.

```
save SourcePeerAddress;
save DestPeerAddress;
count;
```

FIGURA 2.16 – Regra para contabilizar tráfego entre pares de estações

A figura 2.17 ilustra uma regra de filtragem que permite contabilizar fluxos individuais originados em estações pertencentes a uma de seis sub-redes IP distintas. Os cinco comandos *save* produzirão entradas na tabela de fluxos do *meter* para cada 5-upla distinta de endereços IP, protocolo e números de porta. Vale observar que os números de porta só são salvos para fluxos TCP e UDP, já que os demais protocolos não possuem esse campo.

```
if SourcePeerAddress == (128.54/16, 132.239/16, 132.249/16,
   192.135.237/24, 192.135.238/24, 192.172.226/24) {
   save SourcePeerAddress;
   save DestPeerAddress;
   save SourceTransType;
   if SourceTransType == TCP || SourceTransType == UDP {
      save SourceTransAddress;
      save DestTransAddress;
   }
   count;
}
```

FIGURA 2.17 – Regra para contabilizar tráfego de fluxos individuais

2.3.2 Informações resultantes do processo de medição

Após a instalação dos *meters* (NeTraMet) e do *manager/meter reader* (NeMac), pode-se iniciar o processo de medição de fluxos. A ferramenta NeMac lê um arquivo de configuração onde estão especificados (a) as regras a serem executadas, (b) os *meters* que as executarão e (c) a freqüência com que os resultados parciais da medição devem ser lidos dos *meters*.

TABELA 2.8 – Informações oferecidas pela aplicação NeTraMet

Source Peer	*Dest Peer*	*Source TransType*	*Source Trans*	*Dest Trans*	*To PDUs*	*From PDUs*	*To Octets*	*From Octets*
128.54.1.1	200.248.252.1	TCP	1343	80	4	2	646	984
128.54.1.2	130.216.7.14	ICMP			2157		159618	0
132.239.12.48	10.10.128.72	TCP	2129	25	6	4	1110	1764

As informações obtidas pela ferramenta NeMac, via SNMP, junto aos *meters* são armazenadas em arquivo texto. A tabela 2.8 ilustra as principais informações que estariam presentes nesse arquivo caso a regra apresentada na figura 2.17 fosse utilizada. O arquivo pode, adicionalmente, ser processado pelas ferramentas fd_filter ou fd_extract, gerando saídas mais legíveis.

2.4 A MIB APM (*Application Performance Measurement*)

Os esforços de padronização de mecanismos que contribuam para o gerenciamento de protocolos de alto nível, serviços e aplicações em rede não encerraram com a especificação da MIB RMON2. Para complementar o conjunto de informações oferecido por ela, encontra-se em fase de estudo, no IETF, a MIB APM (*Application Performance Measurement*), proposta por Waldbusser em [WAL 2002]. Essa MIB define objetos que disponibilizam estatísticas sobre o desempenho (ex: tempo de resposta e taxa de transferência) que os usuários percebem nas aplicações que utilizam. Os métodos a serem empregados para construir essas estatísticas não são apresentados na MIB, de modo que cada fabricante pode optar pelos que julgar serem mais convenientes.

A MIB APM identifica métricas padrões que cada agente deve coletar. Praticamente todas as aplicações existentes para medição de desempenho utilizam tempo de resposta como uma delas. Waldbusser afirma, no entanto, que essa métrica é útil apenas para um conjunto de aplicações e que outras também precisam ser consideradas. Segundo ele, existem três tipos de aplicações, sendo que cada um requer uma forma diferente de medir desempenho:

- *Orientadas a transação*: possuem uma quantidade razoavelmente constante de dados a transferir. A métrica de desempenho é o tempo de resposta, dado pelo tempo decorrido entre a requisição de um usuário (ex: ao pressionar um botão) e o término da mesma (ex: apresentação dos resultados). *Sites* de comércio eletrônico são exemplos desse tipo de aplicação;

- *Orientadas a vazão*: possuem um volume maior e variado de dados a tranferir. A métrica usada, nesse caso, é Kb/s. Um exemplo de aplicação orientada a vazão é o FTP;

- *Orientadas a fluxo*: essas aplicações entregam dados a uma taxa constante, mesmo que exista banda extra disponível. Contudo, quando a infra-estrutura não pode entregar os dados nessa velocidade, há interrupção ou degradação do serviço prestado pela aplicação. A métrica usada para medir o desempenho dessas aplicações é dada pela razão entre o tempo em que o serviço é interrompido ou degradado pelo tempo total do serviço, e a medida usada é partes por milhão. A transmissão de um vídeo com duração de duas horas prejudicada por tela congelada durante dois segundos, por exemplo, representa um desempenho de 0,027% ou 277 ppm.

2.4.1 Relatórios gerados pela MIB

Na MIB APM cada transação é identificada pela 3-upla aplicação, servidor e cliente, e possui uma medida de disponibilidade e outra de desempenho associadas. A medida de desempenho é sensível ao tipo de aplicação (orientada a transação, vazão ou fluxo). Na tabela 2.9 são ilustradas as informações disponíveis para um conjunto hipotético de transações observado em um intervalo de tempo.

TABELA 2.9 – Transações armazenadas pela MIB APM

#	*Application*	*Client*	*Server*	*Successful*	*Responsiveness*
1	HTTP	Jim	CallCtr	No	-
2	HTTP	Jim	HR	Yes	12 ms
3	HTTP	Jim	Amazon	Yes	7 ms
4	HTTP	Jim	CallCtr	Yes	5 ms
5	Email	Jim	Pop3	Yes	12 ms
6	HTTP	Jane	CallCtr	Yes	3 ms
7	SAP/R3	Jane	SAP	Yes	19 ms
8	Email	Jane	Pop3	Yes	16 ms
9	HTTP	Joe	HR	Yes	18 ms
10	FTP	Jim	IETF	Yes	212 Kbps
11	RealVideo	Joe	CNN	Yes	100%

As transações observadas pelo agente APM podem ser agregadas de várias formas, gerando relatórios estatísticos de desempenho. A granularidade desses relatórios e a freqüência com que devem ser gerados podem ser especificadas pelo gerente da rede. É importante salientar, porém, que dados de diferentes aplicações não podem ser agrupados, porque (a) as características e parâmetros de desempenho de cada aplicação diferem muito e (b) suas métricas podem ser divergentes (ex: uma aplicação pode ser orientada a transação e a outra, orientada a vazão). As estatísticas oferecidas pela MIB após a agregação de um conjunto de transações são:

- *TransactionCount*: número de transações observado durante o período;
- *SuccessfulTransactions*: número de transações que completaram com sucesso;
- *ResponsivenessMean*: média aritmética do desempenho de todas as transações agregadas que completaram com sucesso;
- *ResponsivenessMin*: desempenho máximo observado, entre todas as transações agregadas que completaram com sucesso;
- *ResponsivenessMax*: desempenho mínimo observado, entre todas as transações agregadas que completaram com sucesso;
- $ResponsivenessB_x$: número de transações que completaram com sucesso, cujo desempenho se enquadra em uma das sete faixas especificadas. Como o desempenho de cada aplicação varia muito, o valor dessas faixas pode ser especificado separadamente para cada aplicação monitorada.

Quatro tipos distintos de agregação são suportados pela MIB APM:

- *Agregação de fluxos*: nesse tipo de agregação, um registro é criado para cada combinação distinta de aplicação, cliente e servidor observada pelo agente;
- *Agregação de clientes*: um registro é criado para cada combinação distinta de aplicação e cliente observada;
- *Agregação de servidores*: um registro é criado para cada combinação distinta de aplicação e servidor observada;

- *Agregação de aplicações*: um registro é criado para cada aplicação observada.

A tabela 2.10 apresenta o relatório que seria gerado caso a agregação de fluxos fosse aplicada às transações listadas na tabela 2.9. Vale destacar que o primeiro registro (HTTP/Jim/CallCtr) é a agregação das transações 1 e 4.

TABELA 2.10 – Relatório resultante da agregação de fluxos

Application	*Client*	*Server*	*Transaction Count*	*Sucessful Transactions*	*Rsp Mean*	*Rsp Min*	*Rsp Max*	*Rsp* B_0	*Rsp* B_1
HTTP	Jim	CallCtr	2	1	5	5	5	1	0
HTTP	Jim	HR	1	1	12	12	12	0	1
HTTP	Jim	Amazon	1	1	7	7	7	1	0
Email	Jim	Pop3	1	1	12	12	12	0	1
HTTP	Jane	CallCtr	1	1	3	3	3	1	0
SAP/R3	Jane	SAP	1	1	19	19	19	0	1
Email	Jane	Pop3	1	1	16	16	16	0	1
HTTP	Joe	HR	1	1	18	18	18	0	1
FTP	Jim	IETF	1	1	212	212	212	1	0
RealVideo	Joe	CNN	1	1	100	100	100	0	1

2.4.2 Identificação de transações

Uma aplicação pode possuir mais do que uma métrica de desempenho. Por exemplo, o processo de *login* a um servidor de e-mail (POP3) é orientado a transação, enquanto a recuperação de e-mails é orientada a vazão, uma vez que o desempenho é determinado pela taxa de transmissão obtida para recuperar as mensagens armazenadas. Muitas aplicações consistem de interações que possuem características de desempenho diferentes e merecem ser monitoradas separadamente.

A MIB APM permite monitorar a aplicação como um todo ou suas interações individualmente. Utilizar as duas abordagens possibilita que o gerente tenha, ao mesmo tempo, uma visão macro do comportamento da aplicação e uma visão segmentada. Esta última pode ser necessária para, em caso de mal funcionamento, identificar em que ponto da aplicação o desempenho está comprometido.

Algumas transações de protocolos bem conhecidos e disseminados foram selecionadas e já aparecem na própria definição da MIB. As aplicações mais importantes para as organizações, no entanto, são proprietárias e normalmente desenvolvidas dentro das próprias empresas. Essas aplicações nunca serão registradas como aplicações padrões e fabricantes de agentes APM dificilmente poderão incluir suporte a essas aplicações em seus produtos. Para resolver essa limitação, a MIB APM permite a configuração de interações relevantes de aplicações quaisquer através de uma tabela denominada *User-Defined Application*.

2.4.3 Estrutura da MIB

Após a apresentação do propósito e das funcionalidades oferecidas pela MIB APM, apresenta-se os grupos que a compõem. Nesse momento deverá ficar mais claro onde determinadas informações, já comentadas, encontram-se na MIB. Além disso, outras características (de menor importância para a tese) serão listadas. Os grupos pertencentes à MIB APM são:

- *Application Directory*: contém objetos que permitem a configuração de cada aplicação ou interação a ser monitorada; através desse grupo, o gerente pode selecionar a métrica de desempenho (dentre as suportadas pelo agente para a aplicação/interação) e determinar as faixas B_x a serem utilizadas;
- *User Defined Applications*: informa as aplicações específicas à organização que o agente está monitorando;
- *Report*: é usado para preparar relatórios que agrupam informações de desempenho por fluxo, cliente, servidor ou aplicação;
- *Current Transaction*: apresenta as transações que se encontram em andamento e as respectivas métricas de desempenho. Esse grupo foi criado para oferecer estatísticas para transações de longa duração (ex: transmissão de vídeo ou transferência de arquivos grandes), sem a necessidade de ter que aguardar o seu término;
- *Exception*: é usado para especificar limiares para as métricas de uma aplicação ou interação que, quando ultrapassados, geram notificações ao gerente de rede;
- *Notification*: nesse grupo são especificadas as notificações que serão enviadas ao gerente quando uma exceção ocorrer.

2.5 Plataformas e aplicações comerciais de gerenciamento

Há uma grande variedade de plataformas e aplicações comerciais com funcionalidades ligadas ao gerenciamento de protocolos de alto nível, serviços e aplicações em rede. Praticamente todos os fabricantes de plataformas de gerenciamento, como Hewlett Packard [HEW 2002], IBM [TIV 2002] e Computer Associates [COM 2002], possuem módulos destinados a essa tarefa. Competindo com as plataformas estão aplicações de gerenciamento mais específicas, comercializadas por empresas como Concord [CON 2002], NetIQ [NET 2002], Micromuse [MIC 2002], Aprisma [APR 2002], Lucent [LUC 2002], Candle [CAN 2002], NetScout [NEC 2002] e Progress Software [PRO 2002].

De forma geral, as plataformas e aplicações comerciais estão preocupadas em oferecer recursos que contribuam para garantir alta disponibilidade e bom desempenho de aplicações críticas. Entre as abordagens utilizadas para tal, destacam-se cinco. Na primeira delas agentes de software são instalados nas estações dos usuários finais. Esses agentes monitoram as transações executadas pelo usuário, calculam os tempos decorridos e, de tempos em tempos, informam os resultados obtidos a uma estação central. Uma variante dessa abordagem é a presença de agentes nas estações servidoras monitorando o comportamento das aplicações (por exemplo, através da observação permanente dos arquivos de *log*). A terceira abordagem trabalha com o acompanhamento da execução de transações artificiais, disparadas por estações posicionadas em segmentos a partir dos quais seja importante medir o desempenho percebido pelos usuários. A quarta (e mais poderosa) abordagem é baseada na instrumentação da aplicação a ser monitorada nos lados cliente e/ou servidor. Na quinta abordagem, por fim, as informações são obtidas através da monitoração passiva do tráfego de rede.

A utilização de agentes nas estações dos usuários finais é a técnica que oferece os resultados mais precisos com relação ao desempenho, uma vez que agrupa os atrasos da rede, da aplicação servidora e da própria aplicação cliente (ex: para apresentar os resultados). No entanto, pode não ser escalável dependendo do número de estações a serem monitoradas. Além disso, se o agente não for extremamente enxuto, implicará em consumo de recursos e perda de desempenho das estações onde for instalado. VitalSuite (Lucent), ETEWatch (Candle), Pegasus Application Monitor (NetIQ), Application Performance Management (Tivoli) e Spectrum (Aprisma) são exemplos de aplicações que fazem uso dessa técnica.

Para Boardman [BOA 99], a abordagem mais apropriada é aquela onde as aplicações são monitoradas por agentes de software localizados no lado servidor. Segundo ele, embora essa abordagem falhe ao não considerar atrasos gerados pela estação cliente e a própria rede, ela permite centralizar as medições no servidor, que é justamente onde ocorrem as maiores variações de desempenho. Ademais, essa abordagem, implementada nas aplicações eHealth (Concord) e Spectrum (Aprisma), oferece um conjunto grande de informações sobre a aplicação, facilitando, por exemplo, o diagnóstico de problemas e a configuração ideal da mesma.

A utilização de transações sintéticas para medir o desempenho de aplicações apresenta como desvantagem a utilização de recursos adicionais de rede. Este não chega a ser um problema em um ambiente de rede local, mas certamente é um agravante quando existem canais de longa distância envolvidos. As aplicações VitalSuite (Lucent), Pegasus Network Monitor (NetIQ), Application Performance Management (Tivoli) e Spectrum (Aprisma) oferecem esse tipo de funcionalidade.

A instrumentação de aplicações, técnica oferecida pela plataforma Tivoli, é muito útil para monitorar aplicações construídas na própria organização, mas não pode ser usada para monitorar protocolos e aplicações proprietárias. Além disso, são necessários investimentos consideráveis para treinar pessoal em como usar as APIs (*Application Programming Interface*) de programação.

Monitorar transações através da observação do tráfego de rede é a técnica que produz o menor impacto na infra-estrutura existente, pois não requer a instalação/configuração de software nas estações clientes e servidoras, não compromete a rede com tráfego de transações artificiais, nem exige que as aplicações sejam instrumentadas. No entanto, determinar onde os *probes* serão posicionados é uma tarefa complicada em ambientes segmentados. Além disso, o desempenho dos *probes* para capturar e processar pacotes em redes de alta velocidade pode ser um aspecto crítico. AppScout (NetScout) e VitalSuite (Lucent) são aplicações que implementam essa técnica.

As informações tipicamente coletas pelas plataformas e aplicações de gerenciamento comerciais são: disponibilidade, taxa de perda de pacotes, *round trip time* da rede, vazão da aplicação, atrasos observados na estação cliente, na rede e no servidor. Independente da abordagem utilizada, boa parte das plataformas e aplicações (ex: Patrol e eHealth) preocupa-se em reunir, adicionalmente, informações relacionadas ao desempenho do sistema operacional (como utilização da CPU, da memória e dos dispositivos de entrada e saída) e ao funcionamento da própria rede, para oferecer relatórios mais completos ao gerente. Assim, ao observar a degradação no desempenho de uma aplicação, torna-se mais fácil identificar as causas.

As plataformas e aplicações comentadas, embora bastante concentradas no gerenciamento de desempenho, oferecem algum suporte a falhas, configuração, con-

tabilização e segurança. Aplicações que utilizam agentes de software nas estações servidoras, como Tivoli e Spectrum, têm plenas condições de acompanhar a execução dos software que lá residem (ex: servidor HTTP, de e-mail e de banco de dados). Assim, falhas como indisponibilidade ou comportamento inesperado de um serviço podem ser naturalmente detectadas por essas aplicações de gerenciamento e, em muitos casos, ações corretivas podem ser automaticamente executadas. Quanto à configuração, algumas plataformas oferecem suporte a essa área de gerenciamento, permitindo que aplicações e serviços possam ser configurados remotamente. No que diz respeito à contabilização e segurança, as plataformas e aplicações comerciais limitam-se, em geral, a apresentar eventos gerados por elementos externos a elas (ex: agentes RMON2 e *firewalls*).

A diversidade de protocolos, serviços e aplicações passíveis de gerenciamento não varia muito entre as plataformas e aplicações existentes. Por via de regra, elas oferecem suporte ao gerenciamento de servidores HTTP (ex: Internet Information Server e Apache), servidores de e-mail (ex: Exchange), servidores de bancos de dados (ex: Oracle, DB2 e Sybase) e aplicações de *workflow* (ex: Lotus Notes), entre outros. Essas aplicações são gerenciadas por módulos específicos, comercializados individualmente, que aderem perfeitamente ao *framework* da plataforma ou aplicação de gerenciamento em uso. Apenas um pequeno número de plataformas e aplicações (ex: Tivoli e Spectrum) oferece mecanismos (ex: *toolkits* e SDKs – *Software Development Kits*) para a criação de módulos de gerenciamento específicos.

O último aspecto a ser destacado refere-se ao caráter proprietário das soluções mencionadas. Esse é um fator bastante perturbador por induzir as organizações a adquirirem software de gerenciamento de um único fabricante, mesmo que ele não atenda satisfatoriamente a boa parte das necessidades. Embora existam algumas iniciativas de integração entre plataformas e aplicações de diferentes fabricantes, os benefícios obtidos ficam longe dos proporcionados pelas soluções abertas.

2.6 Análise das soluções existentes

Antes de iniciar a análise das iniciativas apresentadas nas seções 2.1 a 2.5, retoma-se, de forma sucinta, o propósito de cada uma. A MIB RMON2 (*Remote Network Monitoring Management Information Base Version 2*) [WAL 97] provê ao gerente informações detalhadas sobre a utilização da rede, contabilizando estatísticas sobre protocolos de alto nível. A aplicação ntop [DER 99] oferece informações similares às disponibilizadas por RMON2. A arquitetura RTFM (*Realtime Traffic Flow Measurement*) [BRO 99c] presta-se à monitoração de fluxos. A MIB APM (*Application Performance Measurement*) provê informações sobre o desempenho de aplicações. Por fim, as plataformas e aplicações comerciais possuem um escopo de atuação mais abrangente, preocupando-se, sobretudo, com disponibilidade e desempenho.

2.6.1 Quanto às funcionalidades oferecidas

Três critérios foram utilizados na análise apresentada a seguir: a capacidade das iniciativas em oferecer suporte a diferentes áreas de gerenciamento e ao gerenciamento distribuído, bem como a flexibilidade oferecida por elas para monitorar novos cenários. Esses critérios coincidem diretamente com os aspectos que alicerçam a

definição do problema abordado na tese (apresentado na seção 1.2).

As iniciativas analisadas são limitadas no que tange ao suporte a diferentes áreas funcionais (falhas, configuração, contabilização, desempenho e segurança). A MIB RMON2 oferece recursos para contabilização de tráfego de protocolos de alto nível e aplicações, além de auxiliar, embora muito superficialmente, no gerenciamento de desempenho e segurança. A aplicação ntop também aparece forte no suporte à contabilização de tráfego, mas permite ainda resolver alguns (poucos) problemas ligados a configuração, desempenho e segurança. A aplicação NeTraMet, que implementa a arquitetura RTFM, está focada em contabilização, embora alguns parâmetros de desempenho possam ser obtidos. Desempenho é, também, a área de gerenciamento trabalhada pela MIB APM. Como diferencial aparecem as plataformas e aplicações comerciais que, além de se preocuparem com desempenho, provêm funcionalidades ligadas ao gerenciamento de falhas.

A incapacidade de descentralizar tarefas é o segundo aspecto em que peca a maioria das abordagens. As MIBs RMON2 e APM, por fazerem parte da arquitetura SNMP, estão naturalmente associadas ao gerenciamento centralizado. Esse problema pode ser contornado, no entanto, embutindo as MIBs em alguma arquitetura com suporte ao gerenciamento distribuído. Ntop, por sua vez, é uma aplicação *stand alone*. Embora o desacoplamento entre os módulos de monitoração e visualização de resultados seja possível (graças ao servidor HTTP implementado internamente ao monitor), não é viável coletar o tráfego de diferentes sub-redes e visualizá-lo a partir de um ponto central (a exemplo do que ocorre com o padrão RMON2). A aplicação NeTraMet oferece um certo grau de descentralização; os *meter readers* atuam como gerentes intermediários, reduzindo o conjunto de tarefas a ser realizado pelos *managers*. Nas plataformas e aplicações comerciais é comum a utilização de agentes de software com inteligência e autonomia para observar e reagir a eventos significativos. Nesse caso, boa parte do processamento passa a ser realizado por esses agentes. Problemas envolvendo uma aplicação monitorada podem ser resolvidos pelo agente responsável por ela. É importante destacar, porém, que esse tipo de descentralização é ineficaz para detectar problemas oriundos de mais de um ponto da infra-estrutura devido à dificuldade em correlacionar os eventos.

O terceiro aspecto tratado nesta análise é a flexibilidade das iniciativas. Boa parte delas não está completamente preparada para permitir a monitoração de novos protocolos e aplicações, e opera com base em um conjunto pré-determinado deles. A capacidade para poder gerenciar novos cenários depende de atualização do *firmware* do equipamento de monitoração, como ocorre com alguns *probes* RMON2, da configuração de filtros, como nas aplicações ntop e NeTraMet, ou da programação em linguagens de baixo nível, mecanismo presente em algumas plataformas e aplicações comerciais de gerenciamento (ex: Tivoli e Spectrum). A complexidade associada a essas ações, no entanto, faz com que gerentes de rede acabem ignorando a possibilidade de personalizar o que deva ser monitorado.

Vale ressaltar, ainda, que muitas iniciativas de gerenciamento, como ntop e NeTraMet, são limitadas à monitoração, cabendo ao gerente de rede tomar atitudes manualmente quando comportamentos inesperados desses protocolos são observados. Isto não é o que ocorre com plataformas e aplicações comerciais (ex: Spectrum Application Manager, Tivoli Web Services Manager e Tivoli Management Solution for Exchange), que oferecem ao gerente de rede mecanismos para configuração de limiares a atributos da rede a serem monitorados. Quando alcançado, um limiar po-

de disparar um *script* personalizado para resolver um problema potencial e notificar o gerente. A figura 2.18 sintetiza a análise apresentada.

	RMON2	**ntop**	**NeTraMet**	**APM**	**Comerciais**
Áreas de gerenciamento suportadas	contabilização e segurança	configuração, contabilização, desempenho e segurança	contabilização e desempenho	desempenho	falhas e desempenho
Suporte a gerenciamento distribuído	não	não	*meter readers*	não	agentes inteligentes
Especificação de um novo cenário	atualização de *firmware*	filtros BPF	linguagem SRL	configuração de objetos da MIB	programação usando SDKs
Suporte à associação de ações	não	não	não	não	sim

FIGURA 2.18 – Quadro resumo da análise das soluções investigadas

2.6.2 Quanto às tecnologias empregadas

Entre as soluções baseadas na monitoração passiva do tráfego de rede – RMON2, ntop e NeTraMet – observa-se diferenças na granularidade das informações coletadas, além de limitações impostas pelas aplicações. A MIB RMON2 e a aplicação ntop coletam estatísticas como o número de octetos e pacotes enviados/recebidos por uma estação (ou um par de estações), levando em consideração os protocolos utilizados. Uma das fraquezas de ambas as abordagens é a falta de informações relacionadas a desempenho e falhas. Essas dificuldades têm sido discutidas pelo grupo RMON do IETF através da MIB APM (*Application Performance Measurement*) [WAL 2002]. No caso da aplicação ntop é possível, ainda, reconhecer e contabilizar fluxos de pacotes, especificados por regras em baixo nível, que são usadas pelo filtro de pacotes BPF (*BSD Packet Filter*) [MCC 93]. Na arquitetura RTFM apenas campos pré-determinados (até a camada de transporte) dos pacotes capturados podem ser consultados. Informações de protocolos de níveis mais altos não podem ser consideradas devido a essa limitação. Além disso, a exemplo do que também ocorre com a aplicação ntop, o mesmo conjunto de regras é aplicado para cada pacote recebido, impossibilitando a correlação de pacotes dentro de um determinado fluxo.

As plataformas e aplicações comerciais, comparativamente, oferecem um conjunto bem maior de informações sobre as aplicações monitoradas, incluindo estatísticas sobre o estado do sistema (ex: utilização de CPU e memória) e tempos de resposta observados, além de dados significativos relacionados à execução das aplicações (normalmente oriundos dos arquivos de *log*). Para tal, precisam utilizar técnicas mais invasivas, como monitoração ativa (transações sintéticas, utilização de agentes em estações clientes ou servidoras) e instrumentação de código. O custo computacional e o impacto provocado por essas técnicas na infra-estrutura existente, em contrapartida, são bem maiores que os apresentados pelas aplicações baseadas na monitoração passiva do tráfego de rede.

2.7 Introdução à abordagem usada na tese

Entre as iniciativas apresentadas é possível identificar dois focos diferentes de gerenciamento. O primeiro foco, observado em iniciativas como RMON2, ntop, NeTraMet e APM, concentra-se, sobretudo, em oferecer recursos que permitam medir e caracterizar tráfego de rede, levando em consideração protocolos de alto nível, serviços e aplicações. O segundo foco, representado pelas plataformas e aplicações comerciais de gerenciamento, preocupa-se em proporcionar uma solução integral, incluindo não apenas monitoração, mas também ação (quando determinados comportamentos são observados).

A abordagem utilizada na tese, detalhada a partir do próximo capítulo, explora a técnica de monitoração passiva do tráfego de rede (usada pelas iniciativas ligadas à medição e caracterização tráfego) para observar eventos significativos ligados a falhas, contabilização, desempenho e segurança. A especificação desses eventos é realizada através de uma linguagem descritiva para representação de traços de protocolos denominada PTSL (*Protocol Trace Specification Language*) [GAS 2001a, GAS 2001b, GAS 2001c], apresentada no capítulo 3. A linguagem, baseada no conceito de máquinas de estados, permite ao gerente determinar seqüências de trocas de pacotes que, quando observadas na rede, indicam a ocorrência de eventos. Por exemplo, a observação de uma requisição DNS respondida por uma mensagem ICMP (*Internet Control Message Protocol*) do tipo *Port Unreachable* é um indicativo de que o *daemon* responsável pelo serviço de resolução de nomes não está sendo executado [POS 81]; portanto, a observação de uma troca de mensagens como essa no tráfego de rede representa um evento relacionado ao gerenciamento de falhas. A novidade da abordagem proposta com relação às iniciativas que também realizam monitoração passiva de tráfego reside na possibilidade de acompanhar e correlacionar pacotes observados (monitoração *stateful*). Com o conseqüente aumento na granularidade dos eventos observáveis, amplia-se consideravelmente o conjunto de tarefas de gerenciamento que podem ser executadas com a utilização dessa técnica.

Para viabilizar a abordagem proposta criou-se a arquitetura de gerenciamento Trace [GAS 2000b, GAS 2001a, GAS 2001b, GAS 2001c] (apresentada no capítulo 4). A arquitetura estende a infra-estrutura centralizada de gerenciamento SNMP para um modelo hierárquico ao introduzir gerentes intermediários, além dos tradicionais gerentes, e agentes de monitoração e ação. O gerente, a partir da estação de gerenciamento, define tarefas de gerenciamento, traços de protocolos (usando a linguagem PTSL) e ações (*scripts* em Tcl, Java ou Perl). Um exemplo de tarefa de gerenciamento ligado ao traço apresentado no parágrafo anterior seria: *"sempre que o traço for observado mais do que três vezes dentro de um intervalo de trinta segundos, disparar o reinício do serviço de resolução de nomes"*. As tarefas de gerenciamento são especificadas através de um *wizard* e oportunamente convertidas para um *script* Tcl (*Tool Command Language*). Um exemplo de ação seria um *script* Perl que reinicia o *daemon named*.

Para garantir a escalabilidade da arquitetura, a execução de tarefas de gerenciamento como a supracitada é delegada a gerentes intermediários. Ao executar uma tarefa de gerenciamento, o gerente intermediário inicialmente (a) configura um agente de monitoração informando o traço que ele deve passar a monitorar. O agente armazena as estatísticas sobre os traços em uma MIB similar à RMON2. Após a configuração do agente de monitoração, o gerente intermediário (b) consulta o agen-

te periodicamente para coletar estatísticas a respeito do traço associado. Sempre que o número observado de ocorrências do traço corresponder ao que a tarefa determina (no exemplo acima, três ocorrências em um intervalo de trinta segundos), (c) uma ação pode ser executada. Para tal, o gerente intermediário delega a execução do *script* de ação associado à tarefa (voltando ao exemplo, o *script* para reiniciar o *daemon named*) ao agente de ação que reside na estação onde o serviço monitorado se encontra. A arquitetura apresenta ainda mecanismos de notificação (*traps*) para que o gerente intermediário possa sinalizar a ocorrência de eventos significativos à estação de gerenciamento.

O que se propõe, portanto, é um ambiente de gerenciamento aberto e hierárquico que explora a observação passiva do tráfego para a execução de tarefas de gerenciamento ligadas a protocolos de alto nível, serviços e aplicações em rede. As funcionalidades obtidas com esse ambiente superam as oferecidas pelas iniciativas ligadas à medição e monitoração de tráfego, mas se assemelham às disponibilizadas pelas plataformas e aplicações comerciais. No entanto, ao contrário do que ocorre com a maioria das plataformas e aplicações, (a) tarefas de gerenciamento ligadas a quatro áreas funcionais de gerenciamento (falhas, contabilização, desempenho e segurança) podem ser especificadas e executadas, (b) o ambiente oferece suporte à descentralização de tarefas e, mais importante, (c) a especificação de novos cenários de gerenciamento pode ser realizada, a qualquer momento, usando mecanismos de alto nível.

3 Representação de Traços de Protocolos

Este capítulo descreve PTSL (*Protocol Trace Specification Language*), uma linguagem para a representação de interações (traços) de protocolos de alto nível, introduzida na seção 2.7 [GAS 2001a, GAS 2001b, GAS 2001c]. O capítulo está organizado da seguinte forma. Inicialmente, na seção 3.1, é esclarecido o que precisa ser especificado e, adicionalmente, são apresentados requisitos que a linguagem deve atender para permitir tais especificações. A seção 3.2 discute algumas notações e formalismos que foram analisados antes da elaboração da linguagem. Em seguida, as seções 3.3 e 3.4 detalham, respectivamente, as notações gráfica e textual de PTSL. Exemplos de especificações de traços de protocolos são apresentados na seção 3.5. O capítulo encerra, na seção 3.6, com uma análise da linguagem proposta.

3.1 Requisitos impostos à linguagem

A seção 1.1 apresentou exemplos de tarefas de gerenciamento ligados a protocolos de alto nível, serviços e aplicações em rede. Entre elas encontram-se a *monitoração da disponibilidade de serviços*, a *contabilização de acessos ao servidor HTTP* e a *monitoração da segurança de serviços e aplicações*. Segundo a abordagem proposta na tese, a implantação dessas tarefas requer, entre outras definições (citadas na seção 2.7 e detalhadas no próximo capítulo), a especificação de trocas de pacotes (mensagens[1]), cuja ocorrência no tráfego de rede deva ser observada para identificar, por exemplo, falhas, transações a serem contabilizadas (número de ocorrências e tempo de resposta) e problemas de segurança. A seguir são apresentados, em linguagem natural, exemplos de traços que podem ser utilizados para implantar tarefas de gerenciamento como as mencionadas:

1. *Indisponibilidade do serviço de resolução de nomes*: conforme apresentado na seção 2.7, a indisponibilidade do serviço de resolução de nomes pode ser detectada ao se observar uma requisição DNS padrão seguida de um pacote ICMP do tipo *Port Unreachable*;

2. *Requisições HTTP retornadas com erro*: para contabilizar o número de requisições simples para as quais são retornados códigos de erro é necessário identificar pacotes HTTP do tipo GET que são seguidos por pacotes HTTP que apresentem os códigos HTTP/1.1 400 a HTTP/1.1 417 [FIE 99];

3. *Sondagem de portas*: entre as diversas técnicas usadas por atacantes para sondar se a estação alvo oferece ou não determinado serviço TCP, uma delas é enviar pacotes TCP SYN para todas as portas da estação alvo; se ela não oferecer o serviço em uma determinada porta, envia um pacote TCP de retorno com o bit RST ligado em resposta à tentativa de conexão. Para detectar esse tipo de sondagem, portanto, é preciso observar um número significativo de pedidos de abertura de conexão (pacotes TCP SYN) seguidos de pacotes TCP RST.

[1] Os termos *mensagem* e *pacote* são usados como sinônimos ao longo de todo o trabalho.

Para poderem ser utilizados por uma ferramenta de monitoração os traços precisam ser especificados, de forma não ambígua, através de uma linguagem. Considerando (a) o objetivo de oferecer ao gerente de rede um mecanismo flexível para a especificação de novos cenários de gerenciamento (em oposição às soluções atuais que são bastante limitadas nesse sentido) e (b) diferentes tipos de interações de protocolos (incluindo os ilustrados acima), cujas descrições deveriam ser contempladas, foram identificados alguns requisitos que a linguagem deveria atender:

- A linguagem precisa ser de fácil utilização de modo que os esforços despendidos na especificação dos traços sejam os menores possíveis; gerentes de rede devem ser capazes de especificar traços conhecendo apenas a estrutura do protocolo ou aplicação em rede (formato dos pacotes);

- A linguagem deve permitir a especificação de traços que suportem a implantação de tarefas ligadas ao gerenciamento de falhas, contabilização, desempenho e segurança;

- Ao contrário de algumas abordagens apresentadas no capítulo 2, onde as regras são aplicadas a cada pacote individualmente, a linguagem deve permitir a correlação de pacotes, pertençam eles a um mesmo fluxo ou não (ex: *se um pacote X for observado de A para B e algum tempo depois um pacote Y for observado no sentido oposto, então incrementa um contador*); só assim é possível contabilizar as transações ilustradas nos exemplos supracitados (1, 2 e 3);

- A linguagem deve suportar a representação de traços de protocolos das camadas de rede, transporte (como o exemplo 3) e, principalmente, aplicação (como os exemplos 1 e 2), opção ausente na maioria das iniciativas apresentadas no capítulo 2;

- Os traços especificados usando a linguagem não devem ser limitados a uma troca de pacotes específica; cenários mais complexos, com mais de uma seqüência possível de interações, devem ser suportados (como o exemplo 2 ou a modelagem, em um único traço, de dois tipos de interações de protocolos que caracterizem sondagem de porta).

3.2 Em busca de um referencial

Determinados os requisitos que a linguagem precisaria atender, investigou-se formalismos e notações que poderiam ser aproveitados. Dois tipos de formalismos seriam necessários: um para representar a ordem em que as mensagens que compõem o traço devem ser observadas e outro para identificar cada mensagem (caracterização do pacote). Para a representação da ordenação das mensagens foram consideradas as notações de máquina de estados finitos, MSC (*Message Sequence Chart*) [ITU 96a] e TTCN (*Tree & Tabular Combined Notation*) [ISO 92]. Já para a caracterização de cada mensagem avaliou-se a possibilidade de utilizar ASN.1 (*Abstract Syntax Notation One*) [ITU 94] e BNF (*Bacchus-Naur Form*).

A notação de máquina de estados finitos, uma evolução da definição de autômatos, é simples e de fácil entendimento. Essa característica, somada à possibili-

dade que a notação oferece para representar a ordenação das mensagens (considerando diferentes combinações), faz dela uma alternativa adequada para a representação de traços de protocolos. O problema da explosão de estados, típico da notação, não ocorreria em virtude dos traços serem limitados a poucas interações.

MSC é uma linguagem padronizada pelo ITU-T [ITU 96a] (*International Telecommunications Union*), criada para permitir a descrição e a especificação de cenários de interação (trocas de mensagens) entre componentes de um sistema. Amplamente adotada por empresas de telecomunicações e organismos de padronização, a linguagem possui uma notação gráfica (MSC/GR) e uma textual (MSC/PR). É na notação gráfica, contudo, que reside a maior contribuição da linguagem: permitir a visualização clara e intuitiva das interações entre os componentes do sistema descrito, como ilustra o exemplo apresentado na figura 3.1. Ao analisar o uso de MSC para representar traços de protocolos (considerando os requisitos impostos na seção 3.1), identificou-se que a linguagem é restrita à especificação de traços em que uma única seqüência de mensagens pode ser representada. Essa limitação só foi resolvida recentemente, com a definição de HMSC (*High-Level Message Sequence Chart*) [MAU 97].

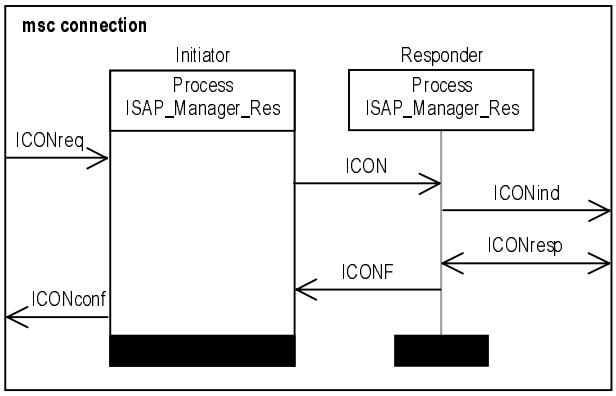


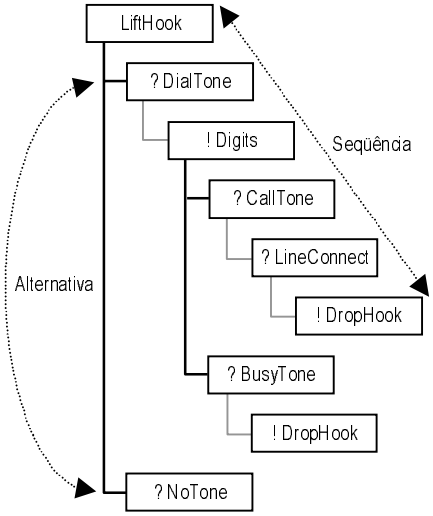
FIGURA 3.1 – Exemplo de especificação usando MSC/GR

Resgatando mais uma notação da área de telecomunicações, TTCN faz parte de um padrão proposto conjuntamente pela ISO e o extinto CCITT para teste de conformidade de protocolos OSI e CCITT [ISO 91]. A exemplo do que ocorre com a linguagem MSC, TTCN também possui uma notação gráfica (TTCN.GR), onde toda a especificação é descrita em tabelas, e uma notação textual (TTCN.MP). Sobre a notação gráfica, é importante destacar que ela não é tão clara e intuitiva quanto as máquinas de estado (como pode ser observado na figura 3.2b). A linguagem oferece suporte à ordenação das mensagens em traços que admitem mais de uma seqüência possível. Para tal, a descrição de todas as possíveis interações entre duas entidades é representada através de uma árvore (figura 3.2a). Como essa descrição é feita sobre tabelas, diferentes níveis de indentação são usados para indicar a progressão na árvore (vide figura 3.2b).

Das três notações consideradas, máquina de estados finitos e TTCN suportariam a especificação de traços de protocolos. No entanto, pela maior facilidade e clareza da notação gráfica (considerando o primeiro requisito imposto à linguagem), optou-se pelas máquinas de estado.

A decisão seguinte seria a definição de um mecanismo a ser adotado para

identificar as mensagens. Antes de comentar sobre os formalismos analisados, é importante frisar que essa identificação nada mais é do que a indicação dos campos do pacote que devem ser observados e os valores esperados em cada um deles. Por exemplo, uma requisição DNS padrão, mensagem que faz parte do traço mencionado na seção anterior, é identificada pelo campo QR, que precisa ter valor igual a 1 para indicar que se trata de uma consulta ao servidor, e pelo campo OPCODE com valor igual a 0, indicando que a consulta é padrão [MOC 87a, MOC 87b].



(a) Eventos e ordem de execução

Test Case Dynamic Behaviour

Test Case Name: Basic Connect
Group:
Purpose: Check that a normal Call connection can be established
Default:
Coments:

Nr	Label	Behaviour Description	CRef	V	Comments
1		L ! LiftHook			
2		L ? Dialtone			
3		L ! Digits	CallSubscr2		
4		L ? CallTone			
5		L ? LineConnect	ConnSubscr2		
6		L ! DropHook		P	
7		L ? BusyTone			
8		L ! DropHook		I	
9		L ? NoTone		F	

Detailed comments:

(b) Tabela de comportamento correspondente

FIGURA 3.2 – Exemplo de especificação usando TTCN

Mensagens de qualquer protocolo da família TCP/IP podem fazer parte de um traço, incluindo os do nível de rede, transporte e aplicação. Alguns desses protocolos são binários, possuindo campos de tamanho fixo (ex: DNS, IP, TCP e UDP). Outros são baseados em caracter e se caracterizam por possuir campos de tamanho variável separados por espaços em branco (ex: HTTP e SMTP) [LAR 99]. A notação usada para a caracterização das mensagens, portanto, deveria suportar a identificação de campos dos dois tipos de protocolos.

Conforme mencionado, ASN.1 e BNF foram os formalismos analisados para essa finalidade. ASN.1, proposta pelo ITU [ITU 94], é uma linguagem para descrição de informações estruturadas, amplamente utilizada na especificação de protocolos de comunicação. BNF, por sua vez, é uma notação de mais baixo nível tipicamente usada para definir a sintaxe de linguagens de programação. Enquanto ASN.1 aparece como uma alternativa adequada para definir protocolos cujos campos possuem uma localização pré-determinada (independente do protocolo ser baseado em caracter ou binário), BNF aparece como uma alternativa para representar protocolos baseados em caracter onde as linhas de texto que compõem a mensagem possam aparecer em qualquer ordem (ex: linhas do cabeçalho do protocolo HTTP).

Além do problema de serem disjuntas e complementares, ambas são de baixo nível e de difícil utilização. A maior desvantagem em optar por uma notação ou outra, contudo, é que o gerente de rede precisaria identificar todos os campos de uma mensagem para determinar os valores esperados em apenas alguns desses campos. No caso do exemplo acima, não seria suficiente identificar os campos QR e OPCODE; todos os campos da mensagem em questão precisariam ser especificados; os valores

esperados para QR e OPCODE, deveriam ser identificados e o valor dos demais campos seria indiferente. Essa obrigatoriedade tornaria o processo de especificação de traços mais complexo e pouco eficiente.

Pelas razões justificadas, nem ASN.1 tampouco BNF foram utilizados para a representação de mensagens. O que se fez foi criar uma notação simples (detalhada na seção 3.4), onde o gerente precisa identificar apenas os campos que lhe interessa observar em um determinado pacote, pertençam eles a protocolos baseados em caracter ou a protocolos binários.

3.3 Graphical PTSL

A linguagem PTSL (*Protocol Trace Specification Language*), baseada no conceito de máquina de estados finitos, possui componentes gráficos e textuais e, por essa razão, foi dividida em *Graphical* e *Textual* PTSL. As linguagens não são equivalentes. A linguagem textual permite a representação completa de um traço, incluindo a especificação da máquina de estados e as mensagens que provocam as transições. A linguagem gráfica, por sua vez, equivale a um sub-conjunto da textual, oferecendo a possibilidade de representar pictoricamente a máquina de estados e, em um alto grau de abstração, de identificar as transições (mensagens). É possível especificar um traço completo, de forma textual, sem fazer uso de *Graphical* PTSL. O contrário não é possível, uma vez que não há como mapear toda a sintaxe textual para a sintaxe gráfica. A seguir é apresentada *Graphical* PTSL (ou G-PTSL). Em seguida, na seção 3.4, é a vez da linguagem *Textual* PTSL ou simplesmente T-PTSL. No anexo A consta a especificação formal da gramática de *Textual* PTSL.

Conforme já mencionado, *Graphical* PTSL permite a definição parcial de um traço de protocolo. Através de uma máquina de estados, o gerente da rede pode criar uma especificação para monitorar todo ou apenas parte de um protocolo, ou ainda, interações abrangendo mais de um protocolo.

3.3.1 Representação de informações para catalogação e controle de versão

A notação gráfica oferece um construtor onde são incluídas informações sobre o traço, que são relevantes para a catalogação e o controle de versão das especificações (vide figura 3.3). As informações armazenadas para um traço são:

- *Version*: versão da especificação;
- *Description*: breve comentário a respeito do traço;
- *Key*: palavras-chave relacionadas ao traço;
- *Owner*: identificação do indivíduo que especificou o traço;
- *Last Update*: data e hora da última atualização da especificação.

  Além dessas informações, existe o campo *Port*, usado para indicar a porta TCP ou UDP do protocolo sendo monitorado; este só deve ser preenchido quando o traço for restrito a um único protocolo de aplicação.

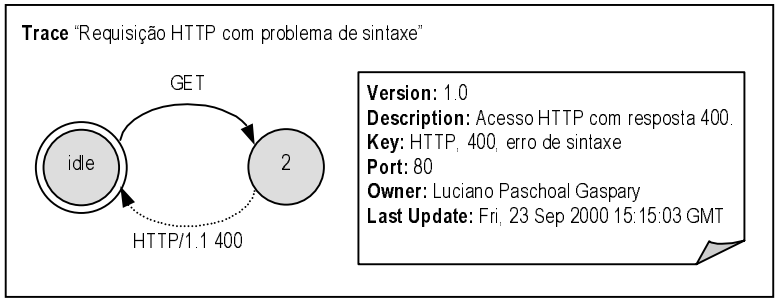


FIGURA 3.3 – Representação gráfica de um traço

3.3.2 Representação de estados, mensagens e agrupamentos

Os elementos básicos que fazem parte da especificação de um traço são:

- *Estado*: descreve o estado atual de um processo cliente;
- *Mensagem cliente*: responsável por fazer o processo cliente mudar de estado; essa mensagem é enviada pelo próprio processo cliente (ex: requisição ao processo servidor);
- *Mensagem servidor*: responsável por fazer o processo cliente mudar de estado; essa mensagem é enviada pelo processo servidor (ex: resposta a uma requisição);
- *Agrupamento*: quando duas ou mais mensagens distintas podem causar a transição de um mesmo estado para outro, elas são agrupadas. Nesse caso, a transição ocorrerá se qualquer uma das mensagens agrupadas for observada na rede.

Um traço inicia sempre a partir de um estado inicial denominado *idle*. A partir daí podem ser criados outros n estados, desde que os mesmos sejam sempre atingidos por alguma mensagem ou agrupamento (não é permitida a criação de estados inalcançáveis). O traço encerra no estado final.

A figura 3.3 ilustra um traço definido por dois estados (*idle* e 2), a mensagem do cliente (GET) e a mensagem do servidor (HTTP/1.1 400). Os estados são representados por círculos identificados; as mensagens do cliente são representadas por setas contínuas e as mensagens oriundas do servidor são representadas por setas pontilhadas. Nesse exemplo, os estados inicial e final são o mesmo (*idle*). Como é possível observar, o estado final é representado graficamente por dois círculos concêntricos. O traço permite monitorar requisições a um servidor HTTP que são respondidas com o código HTTP/1.1 400 (indicando que as mesmas apresentam problema sintático) [FIE 99].

Os textos que identificam as mensagens ou agrupamentos indicam, em um alto grau de abstração, os pacotes que precisam ser observados para que a máquina de estados evolua. Na representação gráfica, contudo, não é possível descrever em detalhe as propriedades que o pacote precisa ter para causar a transição.

Especificações PTSL alimentam agentes de monitoração que capturam pacotes do segmento de rede onde se encontram e observam a ocorrência ou não dos traços. Dessa forma, nas especificações não é necessário definir a origem e o destinatário

do traço. Todas as ocorrências do traço entre quaisquer pares de estações serão devidamente armazenadas e contabilizadas. Mais informações sobre esse processo são apresentadas no capítulo 4, que descreve a arquitetura Trace.

A especificação de um traço pode envolver mais do que um único protocolo. Esse é o caso do exemplo apresentado na figura 3.4. O traço *Indisponibilidade do serviço de resolução de nomes*, já mencionado na seção 3.1, é composto por uma requisição de resolução de nome (protocolo DNS), seguido de uma mensagem ICMP indicando *Port Unreachable*.

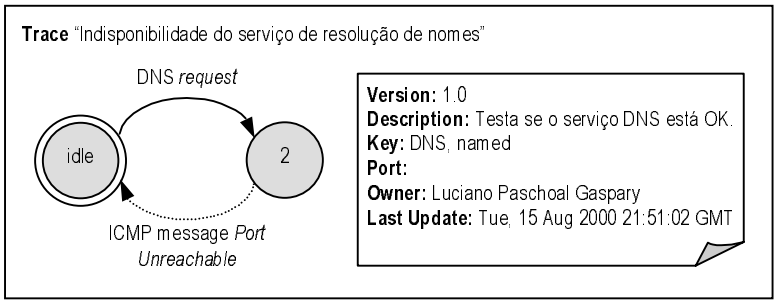


FIGURA 3.4 – Traço envolvendo mais de um protocolo

3.3.3 Representação de temporizadores

Da forma como foram apresentadas até o momento, as transições não possuem limite de tempo para ocorrer. Para determinar um *timeout* para uma transição, é preciso associar um valor (em milissegundos) a ela. No exemplo ilustrado na figura 3.5, abaixo, a transição do estado 2 para o estado inicial tem 5 segundos para ocorrer (após a observação na rede da primitiva GET). Se esse tempo for excedido, a monitoração do traço em questão é cancelada.

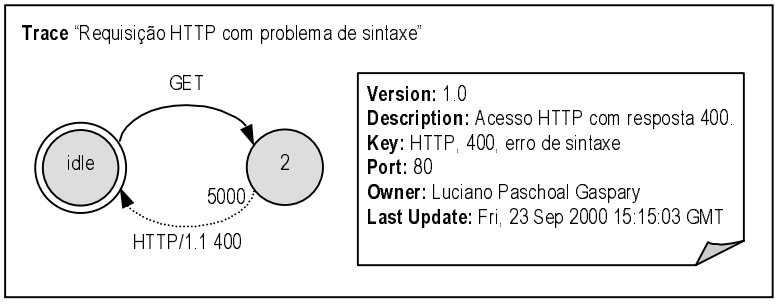


FIGURA 3.5 – Transições com temporizadores associados

3.4 Textual PTSL

A figura 3.6 ilustra o formato geral de uma especificação em T-PTSL. O exemplo usado é o mesmo que foi ilustrado na figura 3.3. A especificação de um traço inicia com a palavra-chave *Trace* e termina com *EndTrace* (linhas 1 e 21). De forma análoga à notação gráfica, num primeiro momento o gerente deve informar o nome do traço e, opcionalmente, os campos *Version*, *Description*, *Key*, *Port*, *Owner* e *Last*

Update (linhas 2 a 7). Em seguida, a especificação é dividida em três seções: *MessagesSection* (linhas 9 a 11), *GroupsSection* (linhas 13 a 15) e *StatesSection* (linhas 17 a 19).

```
1  Trace "Requisição HTTP com problema de sintaxe"
2  Version: 1.0
3  Description: Acesso HTTP com resposta 400.
4  Key: HTTP, 400, erro de sintaxe
5  Port: 80
6  Owner: Luciano Paschoal Gaspary
7  Last Update: Fri, 23 Sep 2000 15:15:03 GMT
8
9  MessagesSection
10 ...
11 EndMessagesSection
12
13 GroupsSection
14 ...
15 EndGroupsSection
16
17 StatesSection
18 ...
19 EndStatesSection
20
21 EndTrace
```

FIGURA 3.6 – Formato de uma especificação T-PTSL

Em *MessagesSection* são definidas as mensagens que causarão a evolução do traço. Quando duas ou mais mensagens causam a transição de um mesmo estado para outro, é possível agrupá-las, tornando a especificação mais clara. Esses agrupamentos são definidos na seção *GroupsSection*. Por fim, na seção *StatesSection* é definida a máquina de estados que representa o traço. É importante lembrar que no exemplo ilustrado na figura 3.6 ainda falta definir as mensagens, os agrupamentos e a máquina de estados.

3.4.1 Representação de mensagens cliente e servidor

A evolução da máquina de estados ocorre quando um pacote com campos idênticos aos especificados em uma mensagem cliente (*client*) ou servidor (*server*) é observado na rede. No exemplo apresentado na figura 3.3, a máquina de estados evolui do estado *idle* para o estado 2 ao observar uma mensagem GET. Nesse caso, é preciso identificar apenas um campo, o GET, que pode ser feito indicando a posição do mesmo (0) a partir do início do campo de dados do protocolo TCP.

A abordagem mencionada acima se aplica a uma boa parcela de protocolos: os baseados em caracter [LAR 99]. Estes são definidos como um conjunto de linhas de texto codificado em ASCII (ex: HTTP e SMTP). Os campos desses protocolos possuem tamanhos variáveis e, em geral, são separados por espaços em branco.

Não menos importantes, no entanto, são os protocolos binários (ex: IP e TCP), definidos como strings de octetos ou de bits com campos de tamanho fixo. A identificação de campos nesses protocolos não pode ser realizada da mesma forma, uma vez que as informações são coladas umas às outras.

Na linguagem PTSL campos de protocolos são representados por um sistema posicional. A diferença entre os protocolos baseados em caracter e binários requer a utilização de duas abordagens distintas para a identificação de campos. São elas:

- *FieldCounter*: usada em protocolos baseados em caracter, esta abordagem trata uma mensagem como um conjunto de primitivas separadas por espaços em branco; a identificação de um campo, nesse, caso, ocorre pela posição do mesmo na mensagem;
- *BitCounter*: usada em protocolos binários; a identificação de um campo é determinada por um *offset* em bits a partir do início do cabeçalho do protocolo em questão até o início do campo desejado; além da posição inicial do campo é preciso indicar ainda o número de bits que o campo ocupa.

A figura 3.7 apresenta a especificação da mensagem GET. Após a identificação da mensagem (linha 2), deve ser informado o tipo da mensagem (*client* ou *server*) (linha 3) e, opcionalmente, o *timeout* associado a ela (*MessageTimeout*). Em seguida são informados os campos a serem observados (linha 5). Nesse caso, o campo é do tipo *FieldCounter*, ou seja, é identificável pela sua posição na mensagem. As informações associadas a um campo *FieldCounter* são as seguintes:

- *Encapsulamento*: identifica onde o campo deve ser buscado; um encapsulamento do tipo Ethernet/IP/TCP indica que o campo deve ser buscado no campo de dados do protocolo TCP;
- *Posição*: indica a posição do campo na mensagem; a primeira posição é identificada por 0;
- *String de comparação*: uma vez identificado o campo no pacote, é feita a comparação entre ele e a string de comparação. A evolução da máquina de estados acontecerá quando, para um pacote capturado, as strings de comparação de todos os campos especificados coincidirem com os valores observados no pacote;
- *Descrição*: breve comentário a respeito do campo.

```
1 ...
2 Message "GET"
3 MessageType: client
4 // OffsetType Encapsulation FieldNumber Verb Description
5 FieldCounter Ethernet/IP/TCP 0 GET "Requisição de uma página Web"
6 EndMessage
7 ...
```

FIGURA 3.7 – Mensagem com campos de um protocolo baseado em caracter

Comentários podem ser adicionados a uma especificação textual. Para tal, utiliza-se barras duplas (//) para identificar o seu início (vide linha 4 na figura 3.7). Os comentários podem aparecer em uma linha dedicada a eles ou bem à direita em uma linha já usada pela especificação do traço.

Em oposição ao exemplo supracitado, o traço especificado na figura 3.4 é baseado em protocolos binários: o DNS e o ICMP. A mensagem que permitirá a evolução da máquina de estados do estado *idle* para o estado 2 é uma requisição DNS. Conforme já mencionado na página 52, para filtrar, entre os pacotes monitorados, uma requisição DNS é preciso observar dois campos: o QR que, quando possui valor 1, indica uma consulta ao servidor e o OPCODE que, quando possui valor 0, indica que é uma consulta padrão. A figura 3.8 ilustra parte inicial do cabeçalho do protocolo DNS.

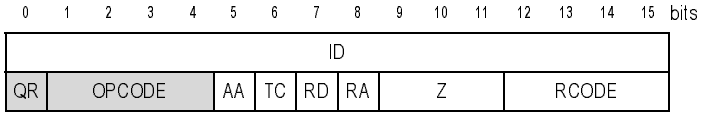


FIGURA 3.8 – Início do cabeçalho do protocolo DNS

A definição da mensagem mencionada acima é apresentada na figura 3.9. Por se tratar de um protocolo binário, todos os campos são do tipo *BitCounter*. As informações associadas a um campo *BitCounter* são as seguintes:

- *Encapsulamento*: identifica onde o campo deve ser buscado; no exemplo, o encapsulamento definido para ambos os campos é Ethernet/IP/UDP, indicando que o protocolo a ser monitorado será encontrado no campo de dados do protocolo UDP;
- *Posição inicial*: indica o *offset* em bits a partir do início do cabeçalho do protocolo em questão até o início do campo desejado; o primeiro bit é o 0;
- *Comprimento do campo*: indica o tamanho em bits do campo;
- *String de comparação*: string usada na comparação com o campo selecionado no pacote;
- *Descrição*: breve comentário a respeito do campo.

```
...
Message "DNS request"
MessageType: client
// OffsetType Encapsulation FirstBit NumberOfBits Verb Description
BitCounter Ethernet/IP/UDP 16 1 1 "Campo QR - 1 significa DNS request"
BitCounter Ethernet/IP/UDP 17 4 0000 "Campo OPCODE - 0 significa Standard query"
EndMessage
...
```

FIGURA 3.9 – Mensagem com campos de um protocolo binário

É possível ainda definir uma mensagem com campos de mais de um protocolo de um determinado encapsulamento. A figura 3.10 ilustra um encapsulamento IP/TCP.

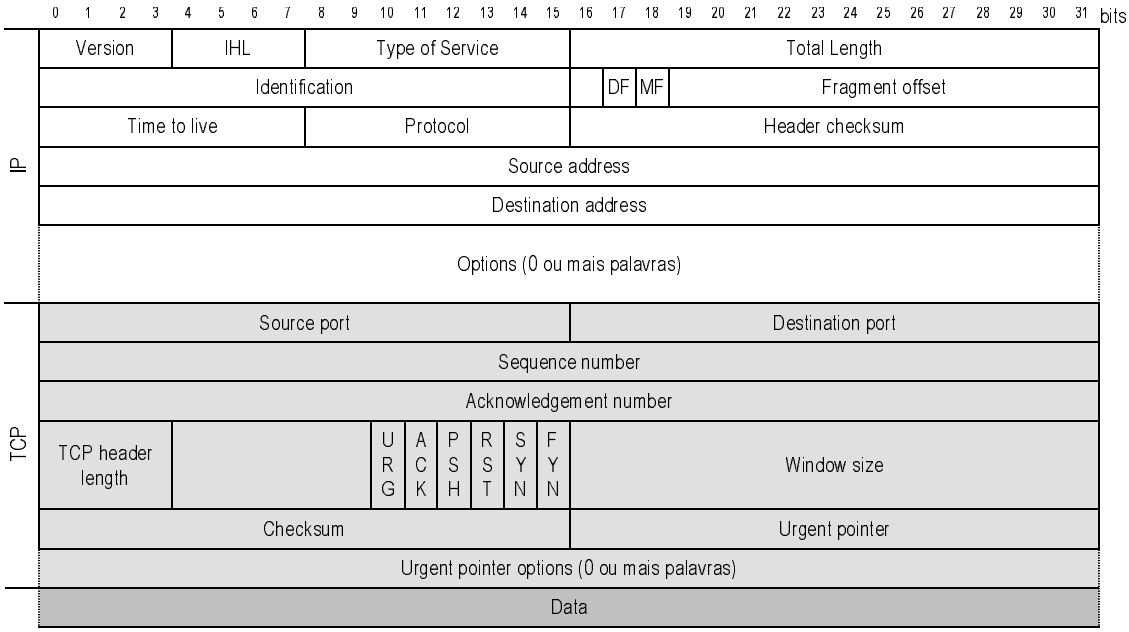


FIGURA 3.10 – Encapsulamento IP/TCP

O gerente da rede poderia estar interessado em contabilizar todas as requisições HTTP que trafegaram em datagramas com ToS (*Type of Service*) igual a 7 (máxima prioridade) do cliente para o servidor HTTP. Nesse caso, os campos a serem observados são o *Type of Service* (deve possuir o valor 7) e o *Destination Port* (deve possuir o valor 80), que pertencem a protocolos binários. O primeiro campo tem seu início no oitavo bit do datagrama IP e tem comprimento de 8 bits enquanto o segundo campo inicia no décimo sexto bit do cabeçalho TCP e ocupa 16 bits. A definição dessa mensagem é apresentada na figura 3.11.

```
...
Message "Requisições HTTP em datagramas de alta prioridade"
MessageType: client
// OffsetType Encapsulation FirstBit NumberOfBits Verb Description
BitCounter Ethernet 8 8 00000111 "Type of Service igual a 7"
BitCounter Ethernet/IP 16 16 0000000001010000 "Destination Port igual a 80"
EndMessage
...
```

FIGURA 3.11 – Mensagem com campos de dois protocolos distintos

Em alguns casos o gerente da rede pode estar interessado em observar a ocorrência de uma string ou seqüência de bits dentro do campo de dados de um determinado protocolo, não interessando a sua posição. Para tal, na definição da mensagem, acrescenta-se um campo do tipo *NoOffset*. Essa característica pode ser explorada, por exemplo, na observação de tentativas de intrusão. O exemplo apresentado na figura 3.12 define que o campo de dados de todos os pacotes TCP partindo de uma estação cliente será vasculhado em busca da string */etc/passwd*.

A linguagem admite também a utilização de coringas na especificação de campos de uma mensagem. Os coringas são representados por um asterisco (*) e podem ser usados para omitir a string de comparação.

```
...
Message "/etc/passwords"
MessageType: client
// OffsetType Encapsulation Verb Description
NoOffset Ethernet/IP/TCP /etc/passwords "Acesso ao arquivo /etc/passwords?"
EndMessage
...
```

FIGURA 3.12 – Campo de uma mensagem com *offset* indeterminado

3.4.2 Representação de agrupamentos de mensagens

É possível agrupar duas ou mais mensagens possíveis em uma única transição. O traço apresentado na figura 3.3 possibilita observar a ocorrência de requisições (a um servidor HTTP) que apresentam problemas de sintaxe (com retorno HTTP/1.1 400). Os acessos com retorno 401 a 417, que também denotam requisições mal sucedidas, podem ser incluídos nessa contabilização se forem especificadas (a) as mensagens que permitem identificá-los (linhas 2 a 24) e (b) um agrupamento, conforme ilustra a figura 3.13. A especificação resultante formaliza o traço *Requisições HTTP retornadas com erro*, mencionado no início deste capítulo. A descrição completa do mesmo é apresentada na seção 3.5.3.

```
1  ...
2  MessagesSection
3
4  Message "HTTP/1.1 400"
5  MessageType: server
6  FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Versão do protocolo"
7  FieldCounter Ethernet/IP/TCP 1 400 "Código de retorno"
8  EndMessage
9
10 Message "HTTP/1.1 401"
11 MessageType: server
12 FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Versão do protocolo"
13 FieldCounter Ethernet/IP/TCP 1 401 "Código de retorno"
14 EndMessage
15
16 ...
17
18 Message "HTTP/1.1 417"
19 MessageType: server
20 FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Versão do protocolo"
21 FieldCounter Ethernet/IP/TCP 1 402 "Código de retorno"
22 EndMessage
23
24 EndMessagesSection
25
26 GroupsSection
27
28 Group "HTTP/1.1 4XX"
29 Messages: "HTTP/1.1 400", "HTTP/1.1 401", "HTTP/1.1 402", "HTTP/1.1 403",
30           "HTTP/1.1 404", "HTTP/1.1 405", "HTTP/1.1 406", "HTTP/1.1 407",
31           "HTTP/1.1 408", "HTTP/1.1 409", "HTTP/1.1 410", "HTTP/1.1 411",
32           "HTTP/1.1 412", "HTTP/1.1 413", "HTTP/1.1 414",
```

```
33          "HTTP/1.1 415", "HTTP/1.1 416", "HTTP/1.1 417"
34 EndGroup
35
36 EndGroupsSection
37 ...
```

FIGURA 3.13 – Representação de um agrupamento de mensagens em T-PTSL

Nas linhas 29 a 33 são listadas as mensagens, já definidas na seção *MessagesSection*, que pertencem ao agrupamento. Na representação visual (figura 3.3), o rótulo associado à transição do estado 2 para *idle* deixa de ser HTTP/1.1 400 e passa a ser HTTP/1.1 4XX, título do agrupamento (linha 28).

3.4.3 Representação da máquina de estados

A seção *StatesSection* permite especificar, de forma textual, a máquina de estados que representa o traço. A máquina de estados do traço ilustrado na figura 3.3 pode ser mapeada para a seguinte especificação textual (vide figura 3.14). O estado final é identificado logo após o início da seção *StatesSection* (linha 3). Os estados *idle* e 2 são definidos, respectivamente, nas linhas 5 a 7 e 9 a 11. A especificação de um estado se resume a listar os eventos (mensagens e agrupamentos) que podem provocar uma transição e indicar, para cada um deles, o próximo estado (linhas 6 e 10).

```
1  ...
2  StatesSection
3  FinalState idle
4
5  State idle
6  "GET" GotoState 2
7  EndState
8
9  State 2
10 "HTTP/1.1 400" GotoState idle
11 EndState
12
13 EndStateSection
14 ...
```

FIGURA 3.14 – Representação de uma máquina de estados em T-PTSL

3.5 Exemplos de especificações

Para validar o poder de expressão da linguagem foram especificados traços que podem ser utilizados para implantar tarefas de gerenciamento ligadas a falhas, contabilização, desempenho e segurança. Alguns desses traços são apresentados a seguir. *Indisponibilidade do serviço de resolução de nomes* e *Requisições HTTP retornadas com erro*, citados na seção 3.1 e apresentados parcialmente ao longo do capítulo para ilustrar as funcionalidades oferecidas por PTSL, são agora listados por completo. Além deles, são descritos os traços *Sondagem de portas*, também mencionado informalmente na seção 3.1, *Inconformidade do protocolo POP3*, *Duração de conexão TCP* e *Ataque por inundação*.

3.5.1 Indisponibilidade do serviço de resolução de nomes

Indisponibilidade do serviço de resolução de nomes é um exemplo de traço que pode ser utilizado para implantar uma tarefa ligada ao gerenciamento de falhas. As especificações gráfica e textual desse traço são apresentadas nas figuras 3.4 e 3.15, respectivamente. Como uma mensagem ICMP do tipo *Port Unreachable* é sempre retornada por uma estação que não está "ouvindo" na porta para onde um pacote UDP foi enviado (ex: requisição DNS), traços semelhantes podem ser especificados para monitorar a disponibilidade de outros serviços baseados no protocolo UDP. Para tal, basta substituir a mensagem *DNS request* pela mensagem de requisição do serviço a ser monitorado.

```
Trace "Indisponibilidade do serviço de resolução de nomes"
Version: 1.0
Description: Testa se o serviço DNS está OK.
Key: DNS, named
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 15 Aug 2000 21:51:02 GMT

MessagesSection

Message "DNS request"
MessageType: client
// OffsetType Encapsulation FirstBit NumberOfBits Verb Description
BitCounter Ethernet/IP/UDP 16 1 1 "Campo QR - 1 significa DNS request"
BitCounter Ethernet/IP/UDP 17 4 0000 "Campo OPCODE - 0 significa Standard query"
EndMessage

Message "ICMP message Port Unreachable"
MessageType: server
BitCounter Ethernet/IP/TCP 0 8 00000011 "Campo Type - 3:Destination unreachable"
BitCounter Ethernet/IP/TCP 8 8 00000011 "Campo Code - 3: Port unreachable"
EndMessage
EndMessagesSection

StatesSection
FinalState idle

State idle
"DNS request" GotoState 2
EndState

State 2
"ICMP message Port Unreachable" GotoState idle
EndState

EndStatesSection

EndTrace
```

FIGURA 3.15 – Traço *Indisponibilidade do serviço de resolução de nomes*

3.5.2 Inconformidade do protocolo POP3

Freqüentemente são lançados no mercado novos protocolos para atender a um conjunto crescente de funcionalidades específicas (ex: ICQ, HTTP, Napster e RSVP). Alguns deles possuem duas ou mais versões sendo usadas em paralelo. Como conseqüência dessa rápida proliferação, protocolos pouco testados ou até mesmo com falhas são disseminados no mercado. Em muitos casos, essas anomalias são a causa da degradação do desempenho da rede e acabam passando despercebidas. Além disso, em função de diferenças entre versões, processos que implementam um determinado protocolo encontram dificuldades para estabelecer e manter comunicação entre si [GAS 2000a].

A linguagem PTSL pode ser usada para modelar, em um único traço, todas as interações admitidas por um protocolo. Esse traço pode ser usado para identificar problemas como os recém citados ou, até mesmo, para determinar se uma implementação específica está de acordo com o que estabelece o padrão (sem o detalhamento de um teste de conformidade).

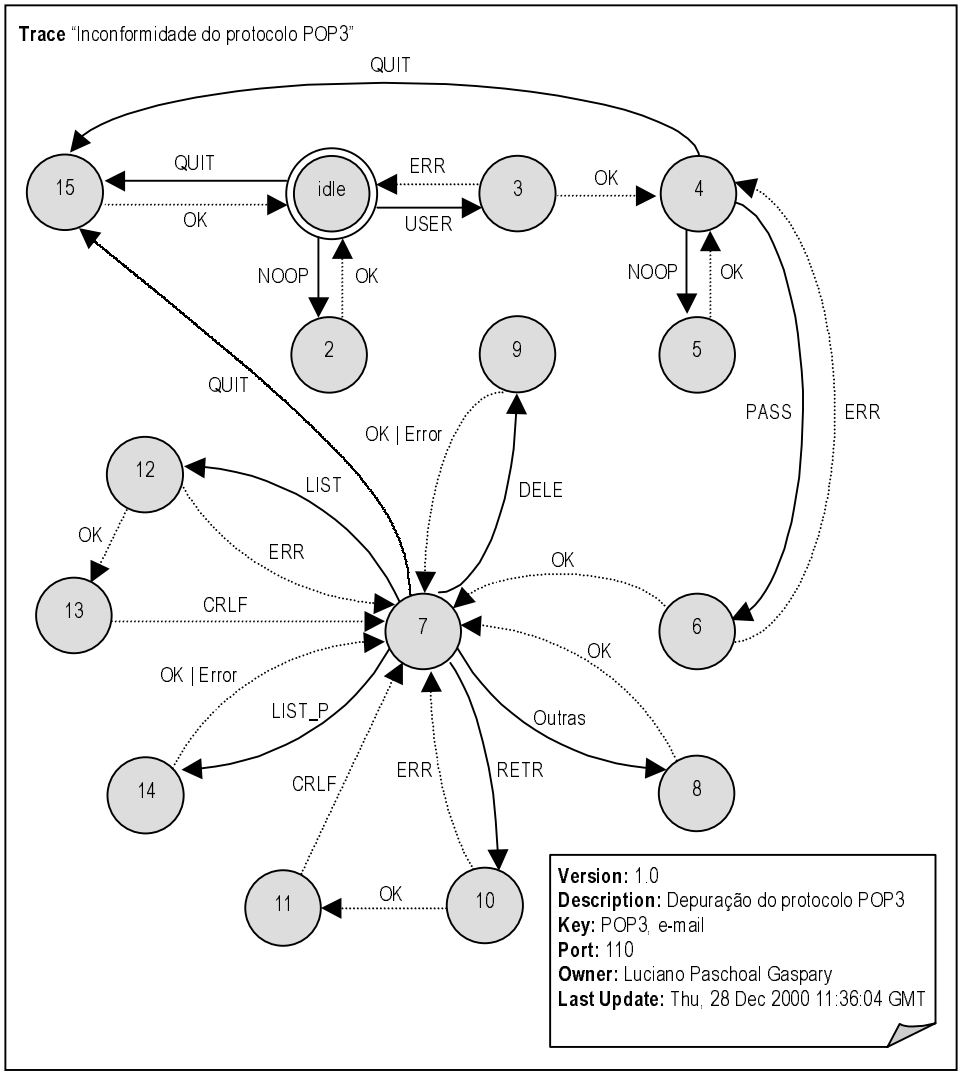


FIGURA 3.16 – Traço *Inconformidade do protocolo POP3* em G-PTSL

A figura 3.16 ilustra, graficamente, o traço *Inconformidade do protocolo POP3*, que descreve as interações do protocolo para recuperação de e-mails (conforme determina a RFC 1225 [ROS 91]). O procedimento básico de funcionamento do protocolo POP3 envolve duas etapas:

- o cliente tenta autenticar-se junto ao servidor, enviando-lhe seu nome de usuário e senha;
- caso a autenticação seja bem sucedida, as mensagens depositadas na conta são, uma a uma, recuperadas e apagadas, de acordo com as requisições do programa cliente.

A especificação desse traço permite observar o quão intuitiva, clara e poderosa é a notação gráfica proposta. Para representá-la usando MSC, que foi uma das hipóteses consideradas (vide seção 3.2), diversas especificações (uma para cada interação possível) precisariam ser descritas. Por outro lado, a especificação resultante seria menos legível caso TTCN fosse a linguagem adotada. A figura 3.17 apresenta a especificação do traço em T-PTSL.

```
Trace "Inconformidade do protocolo POP3"
Version: 1.0
Description: Depuração do protocolo POP3.
Key: POP3, e-mail
Port: 110
Owner: Luciano Paschoal Gaspary
Last Update: Thu, 28 Dec 2000 11:36:04 GMT

MessagesSection

Message "USER"
MessageType: client
// OffsetType Encapsulation FieldNumber Verb Description
FieldCounter Ethernet/IP/TCP 0 USER "Envia username do usuário"
EndMessage

Message "PASS"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 PASS "Envia senha do usuário"
EndMessage

Message "LIST"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 LIST "Solicita lista de mensagens"
EndMessage

Message "LIST_P"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 LIST "Solicita uma mensagem específica"
FieldCounter Ethernet/IP/TCP 1 *
EndMessage

Message "RETR"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 RETR "Solicita mensagem a ser lida"
EndMessage
```

```
Message "DELE"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 DELE "Solicita remoção de uma mensagem"
EndMessage

Message "STAT"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 STAT "Servidor POP está em funcionamento?"
EndMessage

Message "NOOP"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 NOOP "Sinalização do servidor"
EndMessage

Message "LAST"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 LAST "Solicita o número da última mensagem"
EndMessage

Message "RSET"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 RSET "Cancela a remoção de mensagens"
EndMessage

Message "QUIT"
MessageType: client
FieldCounter Ethernet/IP/TCP 0 QUIT "Encerra a conexão"
EndMessage

Message "CRLF"
MessageType: server
FieldCounter Ethernet/IP/TCP 0 .<CR><LF> "Fim da mensagem"
EndMessage

Message "OK"
MessageType: server
FieldCounter Ethernet/IP/TCP 0 OK "Operação realizada com sucesso"
EndMessage

Message "ERR"
MessageType: server
FieldCounter Ethernet/IP/TCP 0 ERR "Operação falhou"
EndMessage

EndMessagesSection

GroupsSection

Group "OK || Error"
Messages: "OK", "ERR"
EndGroup

Group "Outras"
Messages: "NOOP", "STAT", "LAST", "RSET"
EndGroup
```

```
EndGroupsSection

StatesSection
FinalState idle

State idle
"NOOP" GotoState 2
"USER" GotoState 3
"QUIT" GotoState 15
EndState

State 2
"OK" GotoState idle
EndState

State 3
"OK" GotoState 4
"ERR" GotoState idle
EndState

State 4
"NOOP" GotoState 5
"PASS" GotoState 6
"QUIT" GotoState 15
EndState

State 5
"OK" GotoState 4
EndState

State 6
"ERR" GotoState 4
"OK" GotoState 7
EndState

State 7
"Outros" GotoState 8
"DELE" GotoState 9
"RETR" GotoState 10
"LIST" GotoState 12
"LIST_P" GotoState 14
"QUIT" GotoState 15
EndState

State 8
"OK" GotoState 7
EndState

State 9
"OK | Error" GotoState 7
EndState

State 10
"ERR" GotoState 7
"OK" GotoState 11
EndState

State 11
```

```
"CRLF" GotoState 7
EndState

State 12
"ERR" GotoState 7
"OK" GotoState 13
EndState

State 13
"CRLF" GotoState 7
EndState

State 14
"OK | Error" GotoState 7
EndState

State 15
"OK" GotoState 1
EndState

EndStatesSection

EndTrace
```

FIGURA 3.17 – Traço *Inconformidade do protocolo POP3* em T-PTSL

3.5.3 Requisições HTTP retornadas com erro

A monitoração de transações executadas através do protocolo HTTP aparece como uma tarefa de gerenciamento essencial atualmente, dada a importância desse protocolo para a implementação de aplicações, sejam elas críticas ou não. Falhas, contabilização, desempenho e segurança são áreas de gerenciamento que precisam ser consideradas. Para um provedor de notícias, por exemplo, é importante contabilizar e caracterizar os acessos realizados ao *site*, identificando os acessos realizados com sucesso, os mal sucedidos e, até mesmo, as tentativas de acesso não autorizadas. As figuras 3.18 e 3.19 apresentam as especificações gráfica e textual do traço *Requisições HTTP retornadas com erro*, que permite contabilizar requisições a um servidor HTTP que apresentam problemas originados na estação cliente (ex: requisição com problema de sintaxe e solicitação de uma página inexistente). Muitas outras interações desse protocolo podem ser monitoradas usando especificações bastante similares à ilustrada.

Uma organização que utiliza o protocolo HTTP como suporte a aplicações críticas (ex: comércio eletrônico), além de se preocupar com contabilização das transações efetuadas, precisa implantar tarefas que permitam monitorar o desempenho e a segurança das mesmas. No caso do desempenho as transações mais significativas, cujo tempo de resposta mereça ser medido, precisam ser modeladas. Embora o exemplo abaixo não ilustre, é possível tornar mais granular as medições caso se complemente a identificação das requisições determinando a URL (*Unified Resource Locator*) da aplicação de interesse (GET /aplicação/*). Dessa forma, pode-se distinguir informações de desempenho de aplicações distintas. Com relação à segurança, servidores HTTP têm se mostrado vulneráveis a uma série de ações maliciosas. A seção 3.6 menciona uma amostra dessas vulnerabilidades e traços PTSL que podem

ser usados para detectar a exploração das mesmas. Mais informações relacionadas à segurança de servidores HTTP podem ser encontradas em [NOR 2001].

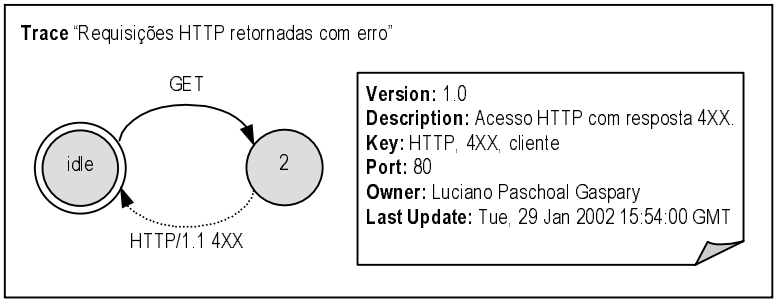


FIGURA 3.18 – Traço *Requisições HTTP retornadas com erro* em G-PTSL

```
Trace "Requisições HTTP retornadas com erro"
Version: 1.0
Description: Acesso HTTP com resposta 4XX.
Key: HTTP, 4XX, cliente
Port: 80
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 29 Jan 2002 15:54:00 GMT

MessagesSection

Message "GET"
MessageType: client
// OffsetType Encapsulation FieldNumber Verb Description
FieldCounter Ethernet/IP/TCP 0 GET "Requisição de uma página Web"
EndMessage

Message "HTTP/1.1 400"
MessageType: server
FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Versão do protocolo"
FieldCounter Ethernet/IP/TCP 1 400 "Código de retorno"
EndMessage

Message "HTTP/1.1 401"
MessageType: server
FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Versão do protocolo"
FieldCounter Ethernet/IP/TCP 1 401 "Código de retorno"
EndMessage

...

Message "HTTP/1.1 417"
MessageType: server
FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Versão do protocolo"
FieldCounter Ethernet/IP/TCP 1 402 "Código de retorno"
EndMessage

EndMessagesSection

GroupsSection
```

```
Group "HTTP/1.1 4XX"
Messages: "HTTP/1.1 400", "HTTP/1.1 401", "HTTP/1.1 402", "HTTP/1.1 403",
          "HTTP/1.1 404", "HTTP/1.1 405", "HTTP/1.1 406", "HTTP/1.1 407",
          "HTTP/1.1 408", "HTTP/1.1 409", "HTTP/1.1 410", "HTTP/1.1 411",
          "HTTP/1.1 412", "HTTP/1.1 413", "HTTP/1.1 414",
          "HTTP/1.1 415", "HTTP/1.1 416", "HTTP/1.1 417"
EndGroup

EndGroupsSection

StatesSection
FinalState idle

State idle
"GET" GotoState 2
EndState

State 2
"HTTP/1.1 4XX" GotoState idle
EndState

EndStatesSection

EndTrace
```

FIGURA 3.19 – Traço *Requisições HTTP retornadas com erro* em T-PTSL

3.5.4 Duração de conexão TCP

O traço *Duração de conexão TCP*, apresentado nas figuras 3.20 e 3.21 pode ser usado para implantar uma tarefa ligada ao gerenciamento de desempenho [GAS 2001d]. Mais abrangente que um traço baseado em protocolo do nível de aplicação (como o ilustrado na seção anterior), ele pode ser usado para medir o tempo que uma conexão TCP fica aberta. Assim, aplicações desenvolvidas sobre esse protocolo e que abrem uma nova conexão a cada transação (ex: HTTP) podem ter o tempo de resposta medido com a utilização desse traço.

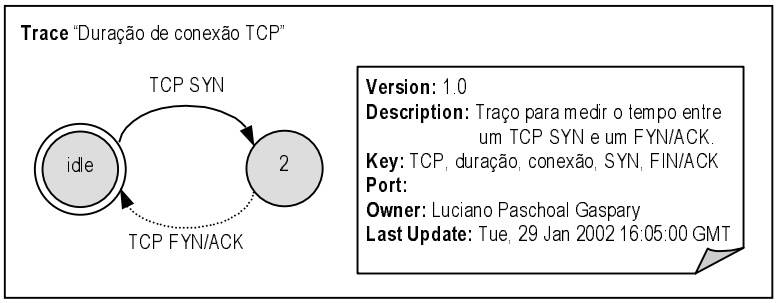


FIGURA 3.20 – Traço *Duração de conexão TCP* em G-PTSL

```
Trace "Duração de conexão TCP"
Version: 1.0
Description: Traço para medir o tempo entre um TCP SYN e um FYN/ACK.
Key: TCP, duração, conexão, SYN, FIN/ACK
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 29 Jan 2002 16:05:00 GMT

MessagesSection

Message "TCP SYN"
MessageType: client
BitCounter Ethernet/IP 110 1 1 "Flag SYN está ligado"
EndMessage

Message "TCP FYN/ACK"
MessageType: server
BitCounter Ethernet/IP 107 1 1 "Flag ACK está ligado"
BitCounter Ethernet/IP 111 1 1 "Flag FIN está ligado"
EndMessage

EndMessagesSection

StatesSection
FinalState idle

State idle
"GET" GotoState 2
EndState

State 2
"TCP FYN/ACK" GotoState idle
EndState

EndStatesSection

EndTrace
```

FIGURA 3.21 – Traço *Duração de conexão TCP* em T-PTSL

3.5.5 Sondagem de portas

Diversas técnicas são usadas por atacantes para sondar quais são os possíveis serviços TCP e UDP disponíveis em um equipamento alvo. Uma delas consiste no envio de pacotes para todas as portas de uma estação [NOR 2001]. No caso do protocolo TCP, ao enviar um pacote com o bit SYN ligado, a resposta esperada é um pacote com os bits SYN e ACK ligados caso aquela porta possua um serviço associado. Caso não haja nenhum serviço nessa porta, a resposta será um pacote TCP com o bit RST ligado. As figuras 3.22 e 3.23 ilustram um traço que permite detectar esse tipo de sondagem.

Outras técnicas de varredura de portas são facilmente implementadas através de PTSL [MEN 2002]. Por exemplo, a técnica na qual é enviado um pacote TCP com os bits SYN e ACK ligado e espera-se um pacote de resposta com o bit RST

ligado (caso a porta não esteja associada a algum serviço) pode ser especificada de forma similar.

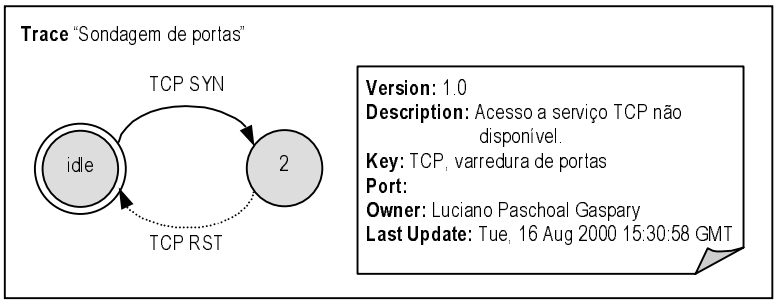


FIGURA 3.22 – Traço *Sondagem de portas* em G-PTSL

```
Trace "Sondagem de portas"
Version: 1.0
Description: Acesso a serviço TCP não disponível.
Key: TCP, varredura de portas
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 16 Aug 2000 15:30:58 GMT

MessagesSection

Message "TCP SYN"
MessageType: client
BitCounter Ethernet/IP 110 1 1 "Campo SYN  - 1 significa TCP Connect"
EndMessage

Message "TCP RST"
MessageType: server
BitCounter Ethernet/IP 109 1 1 "Campo RST"
EndMessage

EndMessagesSection

StatesSection
FinalState idle

State idle
"TCP SYN" GotoState 2
EndState

State 2
"TCP RST" GotoState idle
EndState

EndStatesSection

EndTrace
```

FIGURA 3.23 – Traço *Sondagem de portas* em T-PTSL

3.5.6 Ataque por inundação

O ataque por inundação consiste em enviar um grande número de pacotes de abertura de conexão (pacote com o flag SYN ligado) com um endereço de origem falso para uma determinada estação alvo. Esse endereço de origem falso deve ser inalcançável ou de uma estação inexistente (muitas vezes se usa um dos endereços reservados). A estação alvo, ao receber esses pacotes de abertura de conexão (SYN), cria uma entrada na fila de conexão e envia um pacote de resposta (SYN/ACK) para o endereço que solicitou a conexão. Após o envio do pacote de resposta, a estação alvo fica aguardando uma confirmação do solicitante da conexão. Como o endereço de origem dos pacotes é forjado, a estação alvo nunca receberá essa confirmação de conexão. Em um determinado momento, a fila de conexões da estação alvo fica lotada, e a partir daí todos os pedidos de abertura de conexão são descartados e o serviço indisponibilizado. Essa indisponibilização persiste durante alguns segundos, pois a estação alvo ao descobrir que a confirmação está demorando demais, remove a conexão aberta da lista.

A identificação desse ataque é realizada pelo traço ilustrado nas figuras 3.24 e 3.25. Ao contrário dos exemplos anteriores, o ataque é identificado pela observação de vários insucessos de ocorrência do traço.

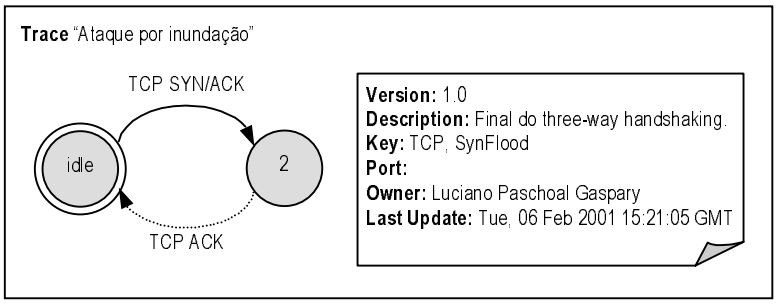


FIGURA 3.24 – Traço *Ataque por inundação* em G-PTSL

```
Trace "Ataque por inundação"
Version: 1.0
Description: Final do three-way handshaking.
Key: TCP, SynFlood
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 06 Feb 2001 15:21:05 GMT

MessagesSection

Message "TCP SYN/ACK"
MessageType: client
BitCounter Ethernet/IP 110 1 1 "Campo SYN"
BitCounter Ethernet/IP 107 1 1 "Campo ACK"
EndMessage

Message "TCP ACK"
MessageType: server
BitCounter Ethernet/IP 107 1 1 "Campo ACK"
EndMessage
```

```
EndMessagesSection

StatesSection
FinalState idle

State idle
"TCP SYN/ACK" GotoState 2
EndState

State 2
"TCP ACK" GotoState idle
EndState
EndStatesSection

EndTrace
```

FIGURA 3.25 – Traço *Ataque por inundação* em T-PTSL

3.6 Análise da linguagem

A proposta da linguagem PTSL é uma das importantes contribuições da tese. Os requisitos impostos à sua elaboração, apresentados na seção 3.1, foram plenamente atendidos. Em relação à facilidade de uso, PTSL mostrou-se uma linguagem simples e intuitiva. Alguns integrantes do grupo de pesquisa exercitaram a especificação de diversos traços [GAS 2001a, GAS 2001b, GAS 2001c, GAS 2001d, MEN 2001, MEN 2002], encontrando pouca ou nenhuma dificuldade nesse processo. A afirmação de que a linguagem é simples e intuitiva é feita também com base na observação das especificações resultantes, que são enxutas e bastante legíveis (devido à existência de uma notação gráfica e à natureza descritiva da notação textual).

As abordagens estudadas e listadas nas seções 2.1 a 2.3 se limitam a contabilizar pacotes enviados/recebidos entre determinados pares de estações classificando-os por protocolo [WAL 97, DER 99] e/ou por fluxo [BRO 2001, DER 99]. Nessas abordagens, as informações a que o gerente de rede tem acesso são, por exemplo, que entre as estações A e B foram observados n octetos/pacotes (a) do protocolo HTTP ou (b) de pacotes nos quais alguns de seus campos possuem valores pré-determinados. As inovações agregadas à PTSL aumentam a granularidade com que os protocolos podem ser monitorados, possibilitando analisar o comportamento de um protocolo ou parte dele ao propor a representação de traços de interesse. Isso instrumentaliza o gerente de rede com informações mais precisas, que lhe auxiliarão a operacionalizar o gerenciamento de falhas, contabilização, desempenho e segurança voltado a protocolos de alto nível e serviços de rede (conforme ilustram os traços apresentados na seção 3.5). Usando o mesmo exemplo anterior, o uso da linguagem permite observar e contabilizar, por exemplo, a ocorrência de acessos HTTP bem sucedidos, mal sucedidos e tentativas não autorizadas, bem como qualquer traço possível no protocolo HTTP. O poder de expressão de PTSL é outro ponto a seu favor. Enquanto boa parte das abordagens permite selecionar pacotes com base em alguns campos pré-determinados de protocolos até, no máximo, a camada de transporte [BRO 2001, BRO 99d], PTSL vai além ao possibilitar que a filtragem seja feita com base em quaisquer campos de quaisquer protocolos, incluindo os do nível de aplicação.

Ao modelar um conjunto de exemplos usando a linguagem, o grupo identificou algumas limitações [MEN 2001, MEN 2002]. Para resolvê-las propôs uma extensão à mesma, incluindo novos construtores e algumas funcionalidades. As mais importantes são detalhadas a seguir.

A falta de operadores de comparação é uma das limitações. A única operação de comparação admitida é o teste de igualdade. Contudo, para modelar alguns traços, em especial os ligados à detecção de intrusão, é importante comparar o valor encontrado em campos de um pacote com alguns limiares. Para resolver esse problema os operadores menor (<), menor ou igual (<=), maior (>), maior ou igual (>=) e diferente de (! =) foram acrescentados à linguagem. Quando nenhum operador é informado o teste de igualdade é realizado. Um exemplo de uso desses operadores pode ser observado na modelagem do traço que descreve o acesso a servidores HTTP com URLs longas, comportamento que caracteriza muitos ataques a servidores HTTP. A figura 3.26 apresenta a descrição, em PTSL, da mensagem que permite identificar requisições HTTP (GET) com URLs longas.

```
...
Message "Requisição HTTP com URL longa"
MessageType: client
BitCounter Ethernet 16 16 0000000111110100 > "Pacote maior do que 500"
BitCounter Ethernet/IP 16 16 0000000001010000 "Porta destino 80"
FieldCounter Ethernet/IP/TCP 0 GET "Operação GET"
EndMessage
...
```

FIGURA 3.26 – Exemplo de utilização de operadores na linguagem PTSL

A segunda funcionalidade não disponível na versão original da linguagem PTSL é a possibilidade de comparar o valor de um campo com o de outro campo (pertencentes a um mesmo pacote). A solução encontrada para a essa limitação foi a adoção do conceito de variáveis, com escopo limitado ao traço em andamento, conforme ilustra a figura 3.27. No exemplo, a variável a está sendo usada para comparar se os endereços IP origem e destino de um pacote são iguais. A observação dessa mensagem em um segmento de rede indica que uma estação (a) está sendo atacada através da técnica conhecida como *Land*.

```
...
Message "Mensagem com IP origem e destino iguais"
MessageType: client
BitCounter Ethernet 96 32 =a "Endereço IP origem é atribuído à variável a"
BitCounter Ethernet 128 32 $a "IP destino é igual ao valor de a?"
EndMessage
...
```

FIGURA 3.27 – Exemplo de utilização de variáveis na linguagem PTSL

Outra extensão está relacionada com a possibilidade de usar caracteres separadores diferentes do espaço, quando usada a abordagem *FieldCounter*. O exemplo apresentado na figura 3.28 ilustra uma requisição HTTP onde o atacante procura passar como argumento */c+dir+c:*, possivelmente a um servidor IIS (*Internet Information Server*). Uma URL como essa pode indicar que o atacante está querendo

executar algum *script* ou CGI (*Common Gateway Interface*) no servidor HTTP a fim de obter uma listagem dos arquivos existentes no servidor. A definição de uma mensagem que permita identificar a sub-string */c+dir+c:* poderia ser feita com a utilização da abordagem *NoOffset*. No entanto, essa abordagem, além de ser computacionalmente onerosa, pode resultar em alarmes falsos, uma vez que a máquina de estados evoluirá quando um pacote que contenha essa string, mesmo que em outra posição, for observada.

```
GET /scripts/..%C0%AF../winnt/system32/cmd.exe?/c+dir+c:\
```

FIGURA 3.28 – Uma requisição HTTP suspeita

Para resolver esse problema o grupo propôs a flexibilização do caracter a ser considerado separador. A figura 3.29 ilustra como isso pode ser feito. No exemplo, o caracter ? é definido como separador. Assim, ao processar a requisição acima dois campos seriam identificados: *GET /scripts/..%C0%AF../winnt/system32/cmd.exe* e */c+dir+c:*. A definição abaixo extrai o segundo campo e o compara com */c+dir+c:*.

```
FieldCounter Ethernet/IP/TCP 1 /c+dir+c:\ ?
```

FIGURA 3.29 – Utilização de outros separadores além do espaço

Por fim, é importante destacar uma limitação da linguagem que não foi resolvida. Trata-se da identificação de campos em protocolos baseados em caracter (normalmente do nível de aplicação) que apresentam mensagens em mais de uma linha, a exemplo do que ocorre com o protocolo HTTP (vide figura 3.30). Nesses casos, é possível identificar apenas a posição de campos que estejam na primeira linha da mensagem (no exemplo, *HTTP/1.0 200 OK*). Em alguns protocolos como o HTTP uma solução para essa limitação seria complexa, porque as linhas de texto que compõem o cabeçalho da mensagem podem aparecer em qualquer ordem. Uma forma de resolver o problema seria utilizar BNFs para descrever o cabeçalho e identificar os campos de interesse, mas essa alternativa foi descartada por sua complexidade. Uma possível solução de contorno para esses casos é a utilização da abordagem *NoOffset*.

```
HTTP/1.0 200 OK
Date: Tue, 15 Aug 2000 21:47:26 GMT
Server: Apache/1.3.2 (Unix)
Last-Modified: Wed, 26 May 1999 18:12:39 GMT
ETag: "8ce96-214-374c3997"
Accept-Ranges: bytes
Content-Length: 532
Content-Type: text/html
Age: 81
X-Cache: HIT from cache.dualnet.com.br
Connection: keep-alive
```

FIGURA 3.30 – Exemplo de cabeçalho do protocolo HTTP

4 A Arquitetura de Gerenciamento Trace

Conforme introduzido na seção 2.7, a arquitetura de gerenciamento Trace é uma extensão da infra-estrutura centralizada de gerenciamento SNMP para, através de um modelo em três camadas, suportar o gerenciamento hierárquico de protocolos de alto nível, serviços e aplicações em rede [GAS 2000b, GAS 2001a, GAS 2001b, GAS 2001c]. Implantada com base na MIB Script [LEV 2001], ela oferece mecanismos para que, a partir de uma estação de gerenciamento, seja possível a delegação de tarefas de gerenciamento (*scripts* em Tcl) a gerentes intermediários que, por sua vez, interagem com agentes de monitoração e agentes de ação para concretizá-las. Especificações PTSL são usadas pelos gerentes intermediários para programar os agentes de monitoração, que passam a observar a ocorrência dos traços. De posse de informações obtidas através da monitoração, os gerentes intermediários podem requisitar a agentes de ação a execução de procedimentos (*scripts* em Tcl, Java ou Perl), viabilizando a automação de diversas tarefas de gerenciamento. A arquitetura apresenta ainda mecanismos de notificação (*traps*) para que o gerente intermediário possa sinalizar a ocorrência de eventos significativos à estação de gerenciamento.

O capítulo está organizado da seguinte forma: inicialmente são apresentadas as decisões de projeto quanto ao paradigma de gerenciamento adotado e quanto à utilização da arquitetura SNMP. Em seguida são descritos os componentes da arquitetura Trace, incluindo o detalhamento de suas funções e as relações estabelecidas entre eles. O capítulo prossegue com a apresentação de um exemplo de uso da arquitetura e encerra com a análise da mesma.

4.1 Decisões de projeto

4.1.1 Paradigma de gerenciamento

Esta seção classifica os paradigmas de gerenciamento distribuído de rede. A classificação usada foi proposta por Martin-Flatin em [MAR 97a, MAR 97b] e adaptada por Schönwälder et al. em [SCH 2000]. O propósito da seção, além de definir os paradigmas de gerenciamento existentes, é apresentar as tecnologias disponíveis para implementá-los e posicionar a arquitetura proposta entre eles.

Uma aplicação típica de gerenciamento distribuído de rede consiste de gerentes, agentes e entidades capazes de realizar as tarefas de ambos, também conhecidas como gerentes intermediários. Tradicionalmente, o gerente é caracterizado por disparar procedimentos de gerenciamento e o agente, por realizar tarefas bastante simples como coletar dados e prover aos gerentes acesso a eles. Atualmente, entretanto, a distinção entre as tarefas associadas a gerentes e agentes começa a ficar menos nítida, uma vez que os agentes de gerenciamento têm assumido um conjunto crescente de responsabilidades. A caracterização dos gerentes intermediários, por sua vez, é dupla; enquanto desempenha seu papel de agente o gerente intermediário realiza tarefas, em geral simples, de forma autônoma, enquanto que ao se comportar como um gerente, as tarefas possuem um nível maior de abstração, e a sua execução requer a comunicação com outros agentes (vide figura 4.1, extraída de [SCH 2000]).

A figura 4.1 destaca a relação de hierarquia estabelecida entre gerentes, gerentes intermediários e agentes. No topo dessa hierarquia encontra-se um ou mais

gerentes; na base encontram-se os agentes e, no meio, os gerentes intermediários. Uma aplicação de gerenciamento não precisa ter instâncias de todas as entidades. É possível imaginar, por exemplo, uma aplicação centralizada sem a presença de gerentes intermediários.

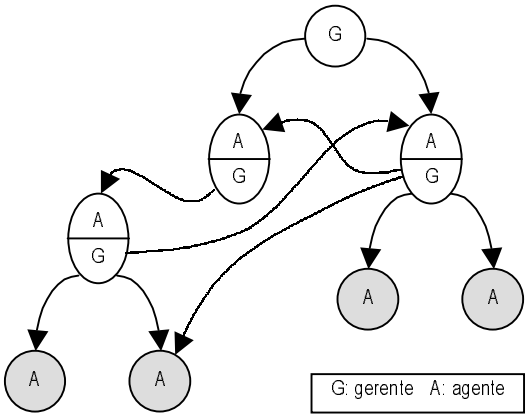


FIGURA 4.1 – Entidades de uma aplicação de gerenciamento e suas relações

É exatamente pelo número relativo de gerentes, gerentes intermediários e agentes que as aplicações de gerenciamento se diferenciam [SCH 2000]. Seja m o número total de gerentes e gerentes intermediários e n o número total de elementos da aplicação de gerenciamento, ou seja, a soma de m e o número de agentes. A partir dessas proposições, Schönwälder et al. distinguem quatro classes ou paradigmas de gerenciamento de rede:

$$
\begin{array}{lllll}
1 & = & \mathrm{m} & & \text{:gerenciamento centralizado} \\
1 & < & \mathrm{m} & \ll \; \mathrm{n} & \text{:gerenciamento fracamente distribuído} \\
1 & \ll & \mathrm{m} & < \; \mathrm{n} & \text{:gerenciamento fortemente distribuído} \\
 & & \mathrm{m} & \approx \; \mathrm{n} & \text{:gerenciamento cooperativo}
\end{array}
$$

Uma ilustração das relações estabelecidas entre gerentes, gerentes intermediários e agentes em cada um dos paradigmas supracitados é apresentada na figura 4.2, onde (a) denota uma arquitetura centralizada; (b), uma arquitetura fracamente distribuída; (c), uma arquitetura fortemente distribuída e (d), uma aplicação de gerenciamento cooperativo.

As tecnologias disponíveis para o desenvolvimento de aplicações de gerenciamento distribuídas ou cooperativas dividem-se entre aquelas que estendem funcionalidades de uma arquitetura existente e as que propõem arquiteturas novas (substituindo outras já em uso). Entre as soluções que se integram a arquiteturas de gerenciamento existentes destaca-se a MIB Script [LEV 2001], que é uma iniciativa do IETF. Por outro lado, agentes móveis, redes ativas e objetos distribuídos são as tecnologias usadas nas novas arquiteturas.

A MIB Script é uma solução de gerenciamento distribuído integrada à arquitetura SNMP. A MIB permite que *scripts*, que implementam procedimentos ligados a gerenciamento, sejam transferidos para o local onde devam ser executados (herdando o conceito de gerenciamento por delegação proposto por Goldszmidt em [GOL 96]). A execução desses *scripts* pode ser disparada remotamente, argumentos podem ser passados a eles e resultados podem ser retornados a quem solicitou sua execução.

A MIB Script é uma alternativa adequada para implementar soluções distribuídas, mas pouco recomendável para aplicações de gerenciamento cooperativas [SCH 2000] devido, por exemplo, à inexistência de suporte à transparência de localização de gerentes intermediários nos sistemas de execução dos *scripts*. Outra limitação da MIB é a sua incapacidade de migrar, além do código, o estado de execução de um *script*.

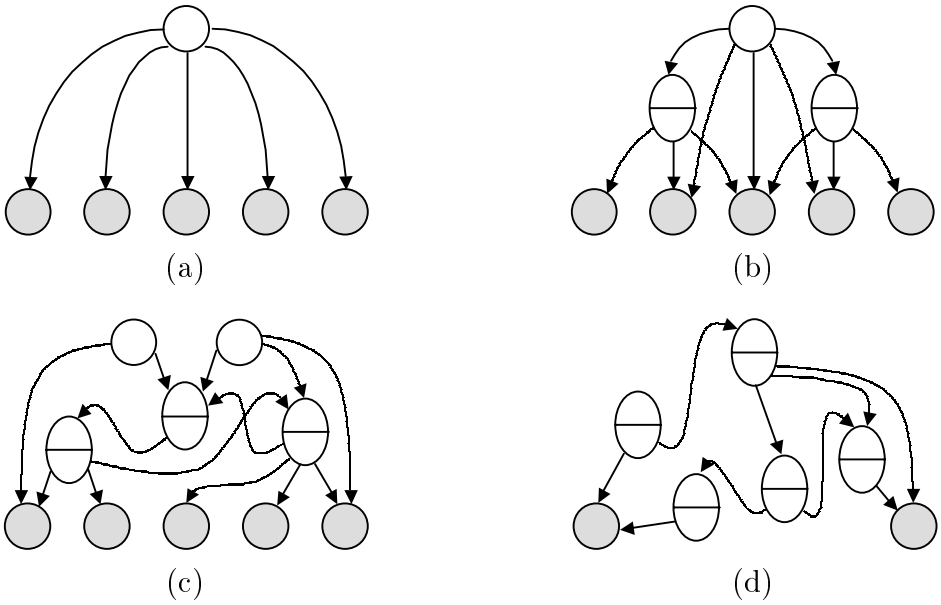


FIGURA 4.2 – Paradigmas de aplicações de gerenciamento

A tecnologia de agentes móveis tem sido utilizada para implementar aplicações de gerenciamento distribuídas e cooperativas [ALB 98, KAR 98, MAG 99] Apud [SCH 2000], embora pareça mais adequada para as cooperativas. A principal diferença entre agentes móveis e a MIB Script é que a primeira tecnologia suporta mobilidade de processos (código + estado de execução). Para Schönwälder, contudo, não são claros os benefícios que essa característica pode proporcionar na implantação de tarefas de gerenciamento. Uma vantagem em utilizar agentes móveis é que aplicações de gerenciamento complexas podem ser implementadas de forma mais estruturada se seus componentes forem projetados como agentes, cada um com uma tarefa ou objetivo específico a ser desempenhado. Esse princípio no projeto de aplicações leva a uma distribuição mais balanceada da complexidade das tarefas de gerenciamento.

Recentemente o foco da comunidade científica voltou-se às arquiteturas de redes ativas [TEN 97, CAL 98] Apud [SCH 2000]. As redes ativas permitem que implementações de um serviço, ou mesmo aplicações finais, insiram sua porção de software de gerenciamento na rede. O software injetado é então executado por nodos de rede *ativos*. Como um dos pontos fortes do gerenciamento baseado em redes ativas é o isolamento da funcionalidade de gerenciamento, a tecnologia não aparece como uma boa solução para implementar aplicações cooperativas, que tipicamente possuem alta conectividade.

Questões em aberto para as soluções baseadas em agentes móveis e redes ativas incluem segurança e manutenibilidade. É preciso desenvolver métodos para entender o comportamento de (a) um sistema cooperativo com um elevado número de com-

ponentes móveis autônomos ou (b) sistemas com um grande número de aplicações individuais de gerenciamento agindo nos nodos.

É importante comentar ainda a utilização de CORBA (*Common Object Request Broker Architecture*) [OMG 99] como uma arquitetura de gerenciamento distribuído. Embora, não seja destinada especificamente ao gerenciamento, ela tem sido utilizada nessa área [KRI 97, ITU 96b] Apud [SCH 2000]. Hegering [HEG 99] aponta algumas razões técnicas que são determinantes na proliferação dessa tecnologia. A principal delas é que os fabricantes consideram interessante usar a mesma arquitetura para a comunicação normal e de gerenciamento, principalmente por razões de custo. Isso significa que o mesmo modelo de objetos e as mesmas ferramentas podem ser usados em ambos os casos.

A principal desvantagem de CORBA é que nenhuma forma de estruturação das informações de gerenciamento comparável às MIBs foi até agora definida. O desenvolvimento de procedimentos padronizados de tradução que permitam que as MIBs sejam usadas em ambientes CORBA constituem uma solução de curto/médio prazo para esse problema. Outra desvantagem é que, comparada à arquitetura SNMP, a arquitetura CORBA é complexa e cara, o que pode representar um obstáculo na fabricação de dispositivos de baixo custo.

Para atender a um dos requisitos impostos à arquitetura, o de que ela suporte a execução descentralizada de tarefas (conforme apresentado na seção 1.3, página 18), o paradigma de gerenciamento adotado encontra-se no limiar entre fracamente distribuído e fortemente distribuído. Nos cenários onde a arquitetura será usada certamente existirá mais de um gerente, mas dificilmente esse número será maior do que o de agentes. Além disso, a comunicação entre gerentes intermediários é inexistente. A opção por uma arquitetura distribuída em detrimento de uma cooperativa deve-se à natureza da abordagem utilizada (apresentada na seção 2.7) e da pouca maturidade das tecnologias disponíveis para desenvolver soluções cooperativas.

4.1.2 A MIB Script: um voto de confiança à arquitetura SNMP

A arquitetura de gerenciamento SNMP, mesmo com as suas limitações, tem sido amplamente utilizada. Uma das razões para isso é a sua simplicidade. Com o advento de novas tecnologias como agentes móveis, redes ativas e plataformas de programação distribuída boa parte da comunidade científica vem dando tratamento de tecnologia legada à arquitetura.

Muitos pesquisadores, no entanto, continuam investindo na arquitetura SNMP e argumentam que direcionar as pesquisas em gerenciamento para outras tecnologias justamente no momento em que se chegou a um consenso entre os fabricantes é inadequado. Além disso, outro ponto ponderado é que as novas tecnologias, em sua grande maioria, apresentam uma série de problemas (destacados na seção anterior) que precisam ser solucionados. No caso da arquitetura SNMP, até mesmo o fator segurança já foi resolvido com a definição do SNMPv3.

Pelas razões mencionadas, optou-se pela utilização da MIB Script na proposta da arquitetura Trace. A adoção dessa MIB, por se tratar de um padrão do IETF, permite que a arquitetura seja implantada sem que seja necessária uma ruptura com as soluções baseadas em SNMP já amplamente disseminadas nas organizações. Além disso, a organização/automação de tarefas de gerenciamento através de *scripts* é um procedimento já amplamente adotado por administradores e gerente de rede. A arquitetura Trace contribui, de certa forma, com a sistematização da execução

dos mesmos, mantendo a cultura de administração baseada em *scripts* já totalmente assimilada pelos profissionais que atuam nessa área.

A MIB Script [LEV 2001] foi desenvolvida pelo grupo de trabalho DISMAN (*Distributed Network Management*), do IETF. Conforme já mencionado, ela permite que processos remotos (ou *scripts*) sejam iniciados, controlados e terminados através do *framework* de gerenciamento SNMP. A especificação da MIB é muito sucinta ao descrever o que são *scripts*; até onde lhe diz respeito, um *script* é um pedaço de código capaz de ser executado pelo agente remoto que implementa a MIB. Isso significa que quem implementa a MIB pode escolher as linguagens de programação em que os *scripts* são implementados.

Estrutura da MIB

A operação da MIB Script resume-se a operações realizadas nas seis tabelas que a constituem (vide figura 4.3). Um gerente que deseja delegar um *script* deve primeiro verificar a tabela *smLangTable* (1), que lista todas as linguagens suportadas pela implementação da MIB Script no agente. Além de verificar essa tabela, ele também pode verificar a tabela *smExtsnTable* (omitida da figura), que contém todas as extensões suportas pelas linguagens instaladas (por exemplo, um pacote Java para acesso SNMP). Essas duas tabelas só podem ser acessadas para leitura, já que é o agente quem possui controle sobre quais são as linguagens e extensões suportadas.

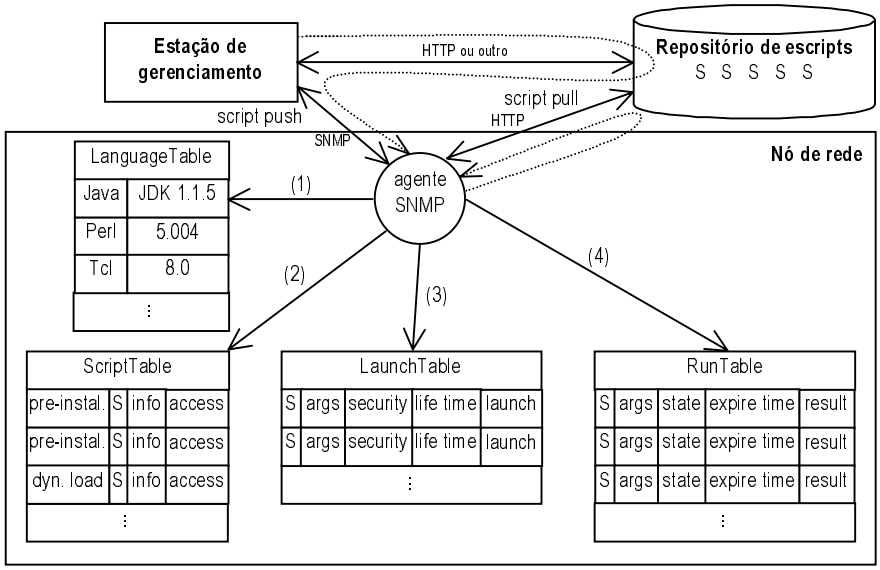


FIGURA 4.3 – Estrutura da MIB Script

Baseado nas informações obtidas nas tabelas de linguagens e extensões, o gerente deve selecionar um *script* apropriado de um repositório. Esse *script* deve ser registrado em uma nova linha criada na tabela *smScriptTable* (2), que contém uma lista de todos os *scripts* conhecidos pelo agente. Esses *scripts* podem ser instalados permanentemente ou podem ser obtidos através dos modelos *client pull* ou *server push*. O modelo *client pull* requer que apenas um URL seja escrito nessa linha da tabela para que o agente possa fazer o *download* do *script*. Já o modelo *server push* requer que o *script* seja escrito linha por linha na tabela *smCodeTable*, através de diversas PDUs (*Protocol Data Units*) contendo *SetRequests* SNMP. A

implementação desse último modelo não é obrigatória pela definição da MIB Script. Outras operações, como a leitura e a remoção de *scripts*, também são suportadas pela tabela *smScriptTable*.

Para que os *scripts* sejam disparados é preciso que uma entrada seja criada na tabela *smLaunchTable* (3). Nessa tabela é descrito o ambiente de execução para cada *script*, incluindo os argumentos que serão passados, o tempo máximo de vida e a identificação do dono do *script*. Um objeto gerenciável nessa tabela, chamado *smLaunchStart*, inicia a execução de um *script* com apenas um *SetRequest* SNMP. Várias instâncias de um mesmo *script* podem ser criadas através de uma única entrada nessa tabela e várias entradas dessa tabela podem apontar para um mesmo *script* na tabela *smScriptTable*. Dessa forma pode-se ter vários *scripts* iguais em execução, mas com argumentos ou permissões diferentes.

Cada *script* disparado possui uma linha criada na tabela *smRunTable* (4) automaticamente. Essa tabela possui a lista de todos os *scripts* que estão executando ou que terminaram a sua execução recentemente. Através dessa tabela o gerente pode controlar os *scripts* em execução, assim como pode obter os resultados intermediários ou finais gerados por eles. O resultado final é mantido no agente até que o seu tempo de vida expire.

Notificações

Existem duas notificações (*traps*) atualmente definidas na MIB Script. A primeira delas é a *smScriptAbort*, que é disparada sempre que um *script* em execução termina com um código de saída (*smRunExitCode*) diferente de *noError*. A segunda delas é a *smScriptResult*, que pode ser utilizada por *scripts* para notificar outras aplicações de gerência (possivelmente gerentes) sobre os seus resultados. Essa notificação não é automaticamente gerada pela implementação da MIB Script. É responsabilidade do *script* em execução enviar essa notificação quando lhe for apropriado. Para tal é necessário que o sistema de execução do *script* forneça mecanismos para o envio de notificações (uma biblioteca ou módulo externo). O funcionamento das notificações, assim como de outras mensagens geradas entre gerente e *script*, pode ser observado no diagrama apresentado na figura 4.4.

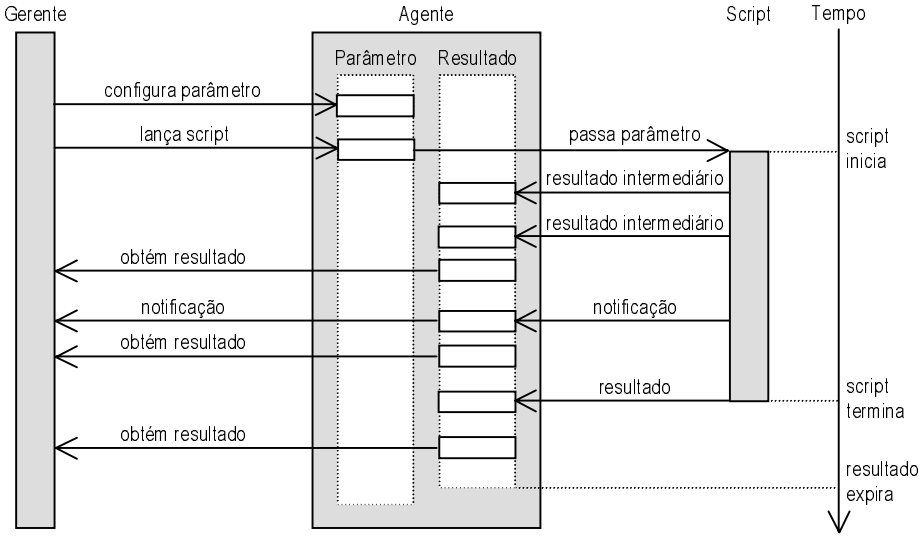


FIGURA 4.4 – Modelo de comunicação gerente-*script*

Aspectos de segurança

Os requisitos de segurança para a MIB Script são diferentes dos de muitas outras MIBs. Enquanto MIBs convencionais operam apenas com permissões de leitura, escrita e notificação, a MIB Script, adicionalmente, exige permissão de execução. Além disso, a MIB provê mecanismos para a criação de perfis de execução de *scripts* com diferentes níveis de restrições.

A MIB Script implementa os requisitos de segurança de acesso fornecendo a cada usuário da MIB uma área reservada dentro de uma área maior, onde todos os *scripts* residem. Uma separação lógica é feita para restringir o acesso. Essa abordagem necessita de uma estrutura de indexação de tabelas para operar em conjunto com os mecanismos de controle de acesso (*View-based Access Control Model*) e segurança de mensagens (*User-based Security Model*) do protocolo SNMPv3 [BLU 99, WIJ 99].

As permissões para acessar recursos na estação que irá executar os *scripts* são obtidas a partir do mapeamento do objeto *smLaunchOwner* para perfis do sistema operacional e de segurança do sistema de execução de *scripts*. Os perfis de segurança do sistema operacional definem os serviços do sistema que podem ser usados pelo *script* a ser executado. O perfil de segurança do sistema de execução, por sua vez, define o conjunto de operações autorizado por um sistema de execução seguro (ex: uma máquina virtual Java ou um interpretador Tcl seguro). Mais detalhes sobre os aspectos de segurança abordados pela MIB Script podem ser encontrados em [SCH 99].

4.2 Componentes da arquitetura

A figura 4.5 ilustra um esquema da arquitetura Trace, destacando seus componentes e as relações estabelecidas entre eles. Essa figura é referenciada nas sub-seções que seguem, onde são apresentadas as atribuições da estação de gerenciamento, dos gerentes intermediários e dos agentes de monitoração e ação.

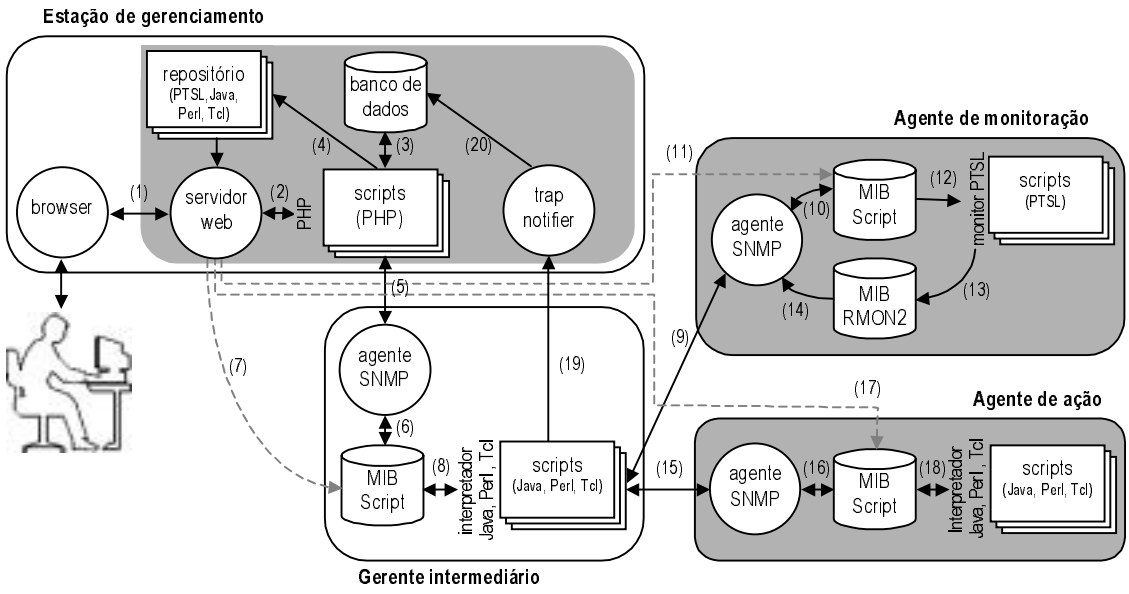


FIGURA 4.5 – Componentes da arquitetura Trace

4.2.1 Estação de gerenciamento

A arquitetura é composta por uma ou mais estações de gerenciamento (gerentes). Desde que respeitadas as interfaces com os demais componentes da arquitetura, nada impede que as aplicações de gerenciamento sejam desenvolvidas com tecnologias variadas. A figura 4.5, entretanto, sugere que a interface do gerente de rede com a arquitetura de gerenciamento seja baseada na Web. As vantagens da utilização da Web para o gerenciamento de redes já foram discutidas por vários pesquisadores [ROS 96, STE 97] Apud [MOU 98] e [BRU 96, MAZ 1996, HON 97, THO 98, BAR 97, STM 98]. Através de um navegador Internet, o gerente acessa o ambiente de gerenciamento em um servidor Web. Nosso grupo de pesquisa elegeu a linguagem PHP [PHP 2001] e o banco de dados mySQL [MYS 2001] para desenvolver o ambiente (detalhado no capítulo 5).

Os módulos destacados na estação de gerenciamento (figura 4.5) podem estar hospedados na própria estação onde se encontra o gerente ou em uma outra estação. Se houver mais de uma estação de gerenciamento, elas poderão compartilhar o mesmo núcleo do ambiente. A seguir são apresentadas as atividades mais importantes realizadas pelo gerente de rede a partir de uma estação de gerenciamento:

- *Cadastramento de gerentes intermediários e agentes*: para facilitar a coordenação entre gerente, gerentes intermediários e agentes, o gerente da rede deve determinar quem são os gerentes intermediários de sua rede, bem como os agentes subordinados a cada um deles. Esse amarramento é importante para organizar que gerentes são responsáveis por que agentes. Ao programar uma tarefa de gerenciamento o gerente intermediário manipulará os agentes a ele subordinados. As interações necessárias para o cadastramento são representadas na figura 4.5 por (1, 2, 3). Essa numeração será usada ao longo da seção para ilustrar o fluxo de dados na arquitetura.

- *Especificação de um traço de protocolo (script em PTSL)*: usando a linguagem apresentada no capítulo 3 é possível especificar um traço de protocolo. Através de um *wizard* (ilustrado no próximo capítulo), o gerente de rede pode usar o ambiente para especificar um traço a partir do zero ou para recuperar traços existentes no repositório e derivar uma nova especificação a partir de um traço já definido (1, 2, 3). Para que possa ser utilizada posteriormente por um agente de monitoração, a especificação do traço, armazenada no banco de dados, precisa ser mapeada para um arquivo texto e disponibilizada no repositório (4).

- *Especificação de uma ação (script em Tcl, Java ou Perl)*: os *scripts* de ação não precisam obrigatoriamente ser especificados usando as funcionalidades do ambiente. É possível realizar o *upload* de um *script* para o repositório (1, 2, 4). Antes disso, é recomendável que o mesmo seja exaustivamente testado.

- *Especificação de uma tarefa de gerenciamento*: a tarefa de gerenciamento é uma rotina (*script*) que descreve a seqüência de passos a ser executada por um agente intermediário para realizar um procedimento ligado a gerenciamento. Um exemplo foi apresentado na seção 2.7 (tarefa para monitorar a disponibilidade do serviço de resolução de nomes). Outros exemplos aparecem na seção 4.3. Ao especificar uma tarefa de gerenciamento (1, 2, 3), mais uma

vez através de um *wizard*, o gerente de rede informa (a) o traço a ser observado, (b) o objeto da MIB RMON2 estendida (apresentada na seção 4.2.3) a ser consultado (no agente de monitoração), (c) o intervalo entre consultas, (d) os limiares aceitáveis para o objeto consultado, (e) a ação a ser executada (caso os limiares sejam ultrapassados) e (f) a notificação a ser enviada à estação de gerenciamento. No caso das duas últimas informações (e e f), a identificação de apenas uma delas é suficiente. Essas especificações, a exemplo do que ocorre com as especificações PTSL, são mantidas no banco de dados.

- *Delegação de uma tarefa de gerenciamento*: para delegar uma tarefa o gerente precisa, após recuperá-la do banco de dados (1, 2, 3), identificar o gerente intermediário, o agente de monitoração e o agente de ação (opcional) que serão envolvidos na sua execução. A partir das informações armazenadas, um *script* em Tcl correspondente é gerado automaticamente e disponibilizado no repositório (4). Em seguida a execução do *script* é delegada ao gerente intermediário (5, 6) via SNMP (MIB Script).

- *Acompanhamento de uma tarefa de gerenciamento*: durante a execução de uma tarefa de gerenciamento, o gerente pode consultar o gerente intermediário para obter resultados parciais da execução da tarefa (1, 2, 5, 6).

- *Interrupção de uma tarefa de gerenciamento*: a interrupção de uma tarefa de gerenciamento requer a desprogramação dos agentes de monitoração e ação envolvidos. Só após é possível finalizar a execução do *script* que corresponde à tarefa de gerenciamento no gerente intermediário (1, 2, 5, 6).

- *Recebimento e visualização de traps*: o gerente pode receber *traps* oriundas dos gerentes intermediários (19). Quando recebidas elas são armazenadas no banco de dados (20). As *traps* são permanentemente recuperadas por um *script* (3) que atualiza o navegador do gerente (2, 1) (tecnologia *push*) para que ele receba imediatamente a notificação.

4.2.2 Gerente intermediário

O gerente intermediário tem por função executar e acompanhar tarefas de gerenciamento, delegadas pela estação central de gerenciamento, e reportar eventos significativos a ela. Uma rede pode possuir um ou mais gerentes intermediários. O número de gerentes intermediários é determinado pelo gerente da rede e depende de fatores como tamanho e complexidade da infra-estrutura a ser gerenciada.

A delegação de uma tarefa a um agente intermediário, como já mencionado, é realizada pela estação de gerenciamento através de primitivas SNMP, suportadas pela linguagem PHP (5, 6). Ao receber, entre outros parâmetros, o endereço URL do *script* a ser executado, a MIB Script recupera o *script* do repositório via HTTPS (*Secure HTTP*) (7) e inicia a execução do mesmo (8).

Os *scripts* executáveis pelo gerente intermediário podem ser codificados em qualquer linguagem de programação, desde que o ambiente de gerenciamento implemente o mapeamento da especificação da tarefa de gerenciamento, mantida no banco de dados, para a linguagem. A implementação da MIB Script usada na prototipação da arquitetura é o Jasmin [TUB 99] (em conjunto com o agente NET-SNMP [NES 2002]), que atualmente suporta Tcl, Java e Perl. Optou-se por gerar os

scripts em Tcl pelo fato de a linguagem possuir diversas bibliotecas que implementam operações ligadas a gerenciamento de rede, além de ser flexível e portável. A complexidade dos *scripts* executados pelo gerente intermediário não chega a ser um fator crítico, uma vez que a especificação e a delegação das tarefas de gerenciamento são realizadas através de *wizards*.

A figura 4.6 apresenta um *script* exemplo simplificado. Trata-se da tarefa que realiza a monitoração do serviço de resolução de nomes (descrita na seção 2.7, página 47). Um agente de monitoração é programado para observar a ocorrência do traço *Indisponibilidade do serviço de resolução de nomes*, descrito na seção 3.5.1. O gerente intermediário consulta, a cada 30 segundos, o agente de monitoração para saber se o traço foi observado na rede. Caso ele tenha ocorrido mais do que três vezes em um intervalo o agente de ação, hospedado na estação onde se encontra o serviço, é programado para executar um *script* que reinicia o serviço de resolução de nomes.

```
1  package require Tnm 3.0
2  package require Trace 1.0
3
4  set oid "protocolDist.protocolDistStatsEntry.protocolDistStatsPkts.1.10"
5  set prev 0
6
7  if { [catch {::Tnm::snmp generator -address $agent} s] } {
8     ::Tnm::smx exit -code runtimeError "Error creating SNMP session: $s"
9     }
10 if {[catch {Trace::InstallScript $ma $m_owner $m_name $m_lang $m_src \
11    $m_descr $m_args $m_ltime $m_etime $m_mrun $m_mcomp} e]} {
12    ::Tnm::smx exit -code runtimeError "Error installing script: $e"
13    }
14 if {[catch {Trace::InstallScript $aa $a_owner $a_name $a_lang $a_src \
15    $a_descr $a_args $a_ltime $a_etime $a_mrun $a_mcomp} e]} {
16    ::Trace::UninstallScript $ma $m_owner $m_name
17    ::Tnm::smx exit -code runtimeError "Error installing script: $e"
18    }
19 ::Trace::RunScript $ma $m_owner $m_name 0
20 proc monitor {} {
21    global s oid prev
22    set val [$s get $oid]
23    set val [lindex [lindex $val 0] 2];
24    if {[expr $val - $prev] > 3} {
25       ::Trace::RunScript $aa $a_owner $a_name 1
26    }
27    set prev $val
28 }
29 ::Tnm::job create \
30    -interval 30000 -error {::Tnm::smx exit -code runtimeError $errorInfo} \
31    -exit {::Tnm::smx exit} -command {monitor}
32 vwait forever
```

FIGURA 4.6 – Exemplo de *script* executado pelos gerentes intermediários

Os *scripts* gerados possuem um formato padrão para todas as tarefas de gerenciamento. As declarações e os procedimentos mais significativos são os seguintes:

- declaração dos módulos usados pelo *script* (linhas 1–2),

- declaração de variáveis (linhas 4–5),
- instalação do *script* de monitoração (linhas 10–13),
- instalação do *script* de ação (linhas 14–18),
- execução do *script* de monitoração (linha 19),
- identificação do objeto a ser monitorado (linha 4),
- laço de monitoração (linhas 29–32),
- dependendo do valor observado (linha 24), ou solicita a execução do *script* de ação (linha 25) e/ou envia uma notificação ao gerente (não usada nesse exemplo).

Às variáveis que aparecem ao longo do *script* (*tokens* que iniciam com o caracter $) são atribuídos valores reais no momento de sua instanciação, ou seja, quando a estação de gerenciamento delega a execução da tarefa ao gerente intermediário (5, 6).

A comunicação do gerente intermediário com agentes de monitoração e agentes SNMP convencionais, realizada pelos *scripts* Tcl, se dá usando primitivas SNMP disponíveis pela linguagem (linhas 7–9). O mesmo ocorre na comunicação do gerente intermediário com a estação de gerenciamento na geração de *traps* (interação 19 na figura 4.5). Já a programação da MIB Script nos agentes de monitoração e ação (interações 9, 10 e 15, 16) é feita com o uso da biblioteca Trace (linha 2) que permite a manipulação dessa MIB através de primitivas de mais alto nível (linhas 10–11, 14–15, 16, 19 e 25).

4.2.3 Agente de monitoração

Os agentes de monitoração contabilizam a ocorrência de traços no segmento onde se encontram. São denominados extensíveis porque os traços a serem monitorados podem ser configurados dinamicamente, sem a necessidade de recompilar esses agentes. Essa flexibilidade é obtida através da linguagem PTSL, apresentada no capítulo 3. Na prática, os agentes realizam a leitura de arquivos PTSL, organizam algumas estruturas em memória e iniciam o processo de monitoração.

A determinação de que traços devam ser monitorados em um dado momento é realizada pelo gerente intermediário. A interface de comunicação entre o gerente intermediário e o agente de monitoração é a MIB Script (interações 9,10 na figura 4.5). No *script* executado pelo gerente intermediário, apresentado na figura 4.6, é possível observar como é feita a programação de um agente de monitoração (linhas 10–13 e 19). Um dos parâmetros informados nesse processo é o endereço URL do *script* (especificação PTSL) a ser executado (variável $m_src). Ao receber do gerente intermediário a solicitação de execução de um determinado *script*, esse é recuperado do repositório via HTTPS (11) e executado (12).

Na realidade, uma especificação PTSL não é executável. A semântica associada à *RunScript* (linha 19 da figura 4.6) é fazer com que o agente de monitoração passe a monitorar o novo traço. De forma análoga, a interrupção de um *script* na MIB Script significa programar o agente de monitoração para que ele cesse a monitoração do traço definido por esse *script*.

O agente de monitoração foi implementado na plataforma Linux usando a linguagem C e a biblioteca de *threads* POSIX [IEE 95]. Uma visão mais detalhada da arquitetura do agente é apresentada na figura 4.7. A *thread* gerente PTSL é responsável pela integração entre a MIB Script e a máquina de execução PTSL. Ela atualiza as estruturas usadas pela máquina PTSL sempre que um novo traço é programado para ser monitorado ou um traço existente é removido (por não ser mais necessário). Outras três *threads* – fila, máquina PTSL e RMON2 – operam segundo o paradigma produtor/consumidor. A primeira é responsável por capturar todos os pacotes usando a biblioteca *libpcap* e adicioná-los a uma fila circular. Embora essa biblioteca suporte a especificação de filtros utilizando a notação BPF (*BSD Packet Filter*), conforme foi apresentado na seção 2.2, esse recurso não está sendo utilizado pelo agente. A segunda *thread* processa cada pacote presente na fila, sem retirá-lo dela, com o objetivo de identificar se o mesmo possui as características esperadas para permitir que um ou mais traços evoluam na máquina de estados. Caso afirmativo, o pacote recebe marcações especiais. A *thread* RMON2, por fim, retira cada pacote da fila e, de acordo com as marcações, faz as devidas contabilizações no banco de dados (mySQL). Essa base de dados é consultada por uma extensão ao agente NET-SNMP [NES 2002] que implementa a MIB RMON2 estendida (detalhada a seguir). Mais informações sobre a implementação do agente de monitoração são encontradas em [MEN 2001, MEN 2002].

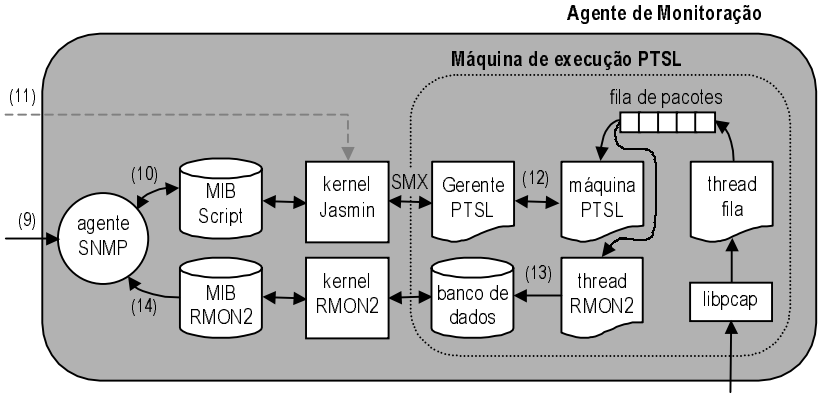


FIGURA 4.7 – Estrutura interna do agente de monitoração

A base de informações de gerenciamento

Toda vez que a ocorrência do traço é observada entre qualquer par de estações, informações são armazenadas em uma MIB similar à RMON2 [WAL 97, GAS 98, GAS 99a, GAS 99b, GAS 99c]. Uma das diferenças da MIB usada com relação à RMON2 é que o grupo *protocolDir*, que indica os protocolos que o agente é capaz de monitorar, passa a permitir a inclusão de traços monitorados pelo agente de monitoração. Conseqüentemente, a granularidade da monitoração torna-se maior. Além de armazenar estatísticas globais sobre o tráfego gerado por um determinado protocolo, estatísticas também são geradas de acordo com a ocorrência de traços especificados. A tabela 4.1 ilustra um conjunto de entradas que poderiam constar no diretório de protocolos de um agente de monitoração.

O processo de inclusão de traços na tabela *protocolDir* é realizado de forma automática pelo agente de monitoração, tão logo ele receba solicitação para monitorar

um novo traço. O ID (*protocolDirID*) é composto de $n \times 4$ bytes, onde n representa o número de protocolos que fazem parte do encapsulamento [BIE 2000]. A numeração usada para identificar encapsulamentos ethernet (00-00-00-01), IP (00-00-08-00) ou protocolos de camadas superiores nunca ultrapassa 16 bits (2 bytes). Assim, para evitar conflitos com a numeração de protocolos convencionais, os traços recebem identificadores superiores a 65535. Na tabela acima os traços *Duração de conexão TCP*, *Sondagem de portas*, *Ataque por inundação*, *Indisponibilidade do serviço de resolução de nomes*, *Inconformidade do protocolo POP3* e *Requisições HTTP retornadas com erro* são identificados, respectivamente por 00-01-00-01, 00-01-00-02, 00-01-00-03, 00-01-00-04, 00-01-00-05 e 00-01-00-06.

TABELA 4.1 – Tabela *protocolDir* da MIB RMON2

ID	*LocalIndex*	*Descr*
00-00-00-01-00-00-08-00	1	ether2.ip
00-00-00-01-00-00-08-00-00-01-00-01	2	ether2.ip.duração de cone...
00-00-00-01-00-00-08-00-00-01-00-02	3	ether2.ip.sondagem de por...
00-00-00-01-00-00-08-00-00-01-00-03	4	ether2.ip.ataque por inun...
00-00-00-01-00-00-08-00-00-00-00-17	5	ether2.ip.tcp
00-00-00-01-00-00-08-00-00-00-00-17-00-00-00-50	6	ether2.ip.tcp.http
00-00-00-01-00-00-08-00-00-00-00-17-00-01-00-04	7	ether2.ip.tcp.indisponibi...
00-00-00-01-00-00-08-00-00-00-00-17-00-01-00-05	8	ether2.ip.tcp.inconformid...
00-00-00-01-00-00-08-00-00-00-00-17-00-01-00-06	9	ether2.ip.tcp.requisições...

Os grupos da MIB RMON2, apresentados na seção 2.1, contabilizam a ocorrência dos traços programados. A tabela 4.2 ilustra o conteúdo da tabela *alMatrixSD*, pertencente ao grupo *alMatrix*. Como é possível observar, ela continua armazenando o número de pacotes e octetos observados entre cada par de estações (cliente/servidor), ou seja, sua semântica foi mantida. Para determinar o número de ocorrências de um traço entre uma estação e outra é preciso dividir o número de pacotes observados pelo número de interações que compõem o traço. No caso da segunda linha da tabela (comunicação entre 125.120.10.200 e 172.16.108.25), em que foram observados 250 pacotes para o traço *Sondagem de Portas*, é possível inferir que o número de ocorrências do mesmo foi de 125, já que ele é composto por duas interações.

TABELA 4.2 – Informações obtidas com consulta à tabela *alMatrixSD*

SourceAddress	*DestAddress*	*Protocol*	*Pkts*	*Octets*
125.120.10.200	172.16.108.25	1 (ether2.ip)	250	3.256
125.120.10.200	172.16.108.25	3 (sondagem de portas)	250	3.256
172.16.108.26	172.16.108.25	1 (ether2.ip)	32	33.041
172.16.108.26	172.16.108.25	6 (http)	32	33.041
172.16.108.26	172.16.108.25	9 (requisições http retornadas com erro)	32	33.041
172.16.108.43	172.16.108.2	1 (ether2.ip)	2	2.204
172.16.108.43	172.16.108.2	7 (indisponibilidade do serviço de...)	2	2.204
125.120.10.27	172.16.108.25	1 (ether2.ip)	1.022	433.204
125.120.10.27	172.16.108.25	4 (ataque por inundação)	1.022	433.204
172.16.108.1	172.16.108.2	1 (ether2.ip)	4	4.350
172.16.108.1	172.16.108.2	7 (indisponibilidade do serviço de...)	4	4.350

A desvantagem em usar a MIB RMON2 é que ela não possui objetos capazes de armazenar informações relacionadas a desempenho. Por essa razão, avaliou-se inicialmente a possibilidade de utilizar uma extensão da RMON2, a MIB APM (*Application Performance MIB*) [WAL 2002]. Por ser uma MIB extensa, de difícil implementação, e ainda se encontrar em fase de padronização (com vários pontos divergentes), seu uso de forma integral foi descartado. Alternativamente, optou-se por utilizar um sub-conjunto da mesma. O detalhamento formal da MIB proposta pode ser encontrado no anexo B.

A MIB APM resumida é composta por dois grupos: *apmAppDirectory* e *apmReport*. O primeiro é usado para realizar configurações a respeito das aplicações monitoradas, incluindo a determinação de sete faixas de desempenho. O segundo grupo, por sua vez, controla a criação e a recuperação de relatórios sobre o desempenho dessas aplicações. Nos relatórios as informações coletadas podem ser agrupadas por fluxo, clientes, servidores ou aplicações (conforme foi apresentado na seção 2.4). As estatísticas oferecidas pela MIB após a agregação de um conjunto de transações são:

- *TransactionCount*: número de transações observado durante o período;
- *SuccessfulTransactions*: número de transações que completaram com sucesso;
- *UnsuccessfulTransactions*: número de transações que não completaram com sucesso;
- *ResponsivenessMean*: média aritmética do tempo de resposta (em segundos) de todas as transações que completaram com sucesso;
- *ResponsivenessMin*: tempo de resposta máximo observado (em segundos), entre todas as transações que completaram com sucesso;
- *ResponsivenessMax*: tempo de resposta mínimo observado (em segundos), entre todas as transações que completaram com sucesso;
- $ResponsivenessB_x$: número de transações que completaram com sucesso, cujo desempenho se enquadra em uma das sete faixas especificadas. Como o desempenho de cada aplicação varia muito, o valor dessas faixas pode ser especificado separadamente para cada aplicação monitorada (grupo *apmAppDirectory*).

4.2.4 Agente de ação

Através dos agentes de monitoração, o gerente intermediário tem condições de avaliar a ocorrência ou não de um traço. Conforme foi apresentado, os traços podem indicar a falha em um serviço de rede, a tentativa de intrusão, a queda de desempenho em algum serviço, entre outros. Nesse contexto, os agentes de ação são responsáveis pela execução de um procedimento de gerenciamento criado para combater, sem a intervenção humana e de forma automática, o problema detectado.

Tomemos como exemplo a tarefa de gerenciamento ligada à monitoração da disponibilidade do serviço de resolução de nomes. O gerente intermediário, ao detectar que o serviço não está disponível (através do laço de monitoração), solicita ao agente de ação, localizado na estação onde o serviço reside, que execute um procedimento para reiniciar o *daemon named*. A figura 4.8 ilustra um *script* em Perl que pode ser usado para tal.

```
#!/usr/bin/perl

my $pid;

# verifica se o processo named esta executando
if (-e "/var/run/named.pid") {
   $pid = `/bin/cat /var/run/named.pid`;
}

# se named esta executando, reinicia atraves de um sinal HUP, senao inicia o
# processo novamente
if (defined $pid) {
   print "Restarting named (sending HUP signal)...\n";
   `/bin/kill -HUP $pid`;
} else {
   print "Starting named (was not running)...\n";
   `/usr/sbin/named &`;
}

# verifica se o processo esta em execucao
if (-e "/var/run/named.pid") {
   $pid = `/bin/cat /var/run/named.pid`;
   print "The named daemon is up and running as PID $pid\n";
} else {
   print "The named daemon could not be started!\n";
}
```

FIGURA 4.8 – *Script* Perl para reiniciar o *daemon named*

A comunicação entre o gerente intermediário e o agente de ação também se dá através da MIB Script (15,16). A criação e instanciação do *script* no agente de ação é similar ao realizado para programar o agente de monitoração (linhas 14–18 e 25 do *script* ilustrado na figura 4.6). O endereço URL de um *script* localizado no repositório é passado para a MIB Script executar (15, 16, 17, 18).

4.3 Exemplos de uso da arquitetura

A figura 4.9 ilustra um cenário real de gerenciamento, composto de três domínios. A organização desses domínios é uma tarefa que o gerente da rede precisa realizar para fazer uso eficiente da arquitetura. Essa tarefa é concretizada via ambiente de gerenciamento, no momento do cadastramento de gerentes intermediários e agentes.

O domínio 1 é composto por equipamentos e serviços relacionados ao acesso da organização à Internet (em cinza escuro na figura). O roteador, por intermédio de uma interface serial, é o elo de ligação com a mesma. Além da interface serial, o roteador possui duas interfaces Ethernet. À primeira interface está conectado um *hub*, que tem ligado a ele duas estações: uma hospeda o servidor de DNS e outra, um servidor HTTP. À segunda interface do roteador está ligada uma estação com duas interfaces de rede. Esta executa um *firewall* e, portanto, representa a divisa entre a rede externa e a interna. Ligado à outra interface do *firewall* (interface interna) está um *hub*. Nele estão conectados o servidor Web responsável pela Intranet da

organização e um *switch* que segmenta a rede interna em várias sub-redes. Os demais equipamentos (em branco na figura) são a estação de gerenciamento, o gerente intermediário do domínio 1 (ambos conectados ao *switch*) e duas estações dedicadas à tarefa de monitoração.

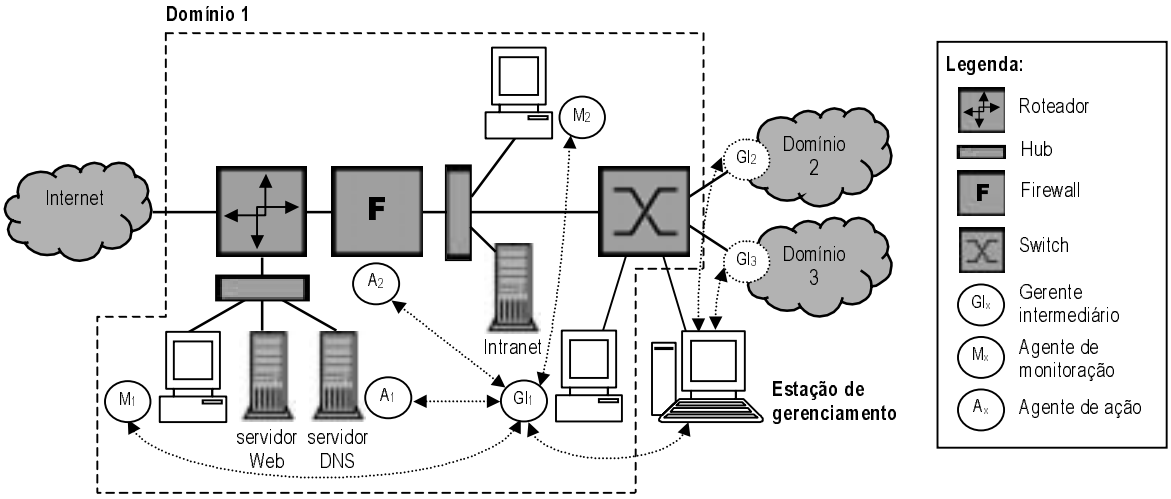


FIGURA 4.9 – Uma rede e alguns de seus serviços

Para ilustrar a utilização da arquitetura apresenta-se, a seguir, alguns exemplos de tarefas de gerenciamento que poderiam ser implantadas no cenário descrito acima.

4.3.1 Monitoração da disponibilidade do serviço de resolução de nomes

Essa tarefa (mencionada nas seções 2.7 e 4.2.2) consiste em identificar se o *daemon named* não se encontra em execução. O gerente intermediário programa um agente de monitoração para observar a ocorrência do traço *Indisponibilidade do serviço de resolução de nomes*, descrito na seção 3.5.1. Em seguida, o gerente intermediário passa a consultar, a cada 30 segundos, o agente de monitoração para saber se o traço foi observado na rede (objeto *protocolDistStatsPkts*). Caso ele tenha ocorrido mais do que três vezes em um intervalo (seis pacotes), o agente de ação, hospedado na estação onde se encontra o serviço, é programado para executar um *script* que reinicia o serviço de resolução de nomes (apresentado na figura 4.8). O *script* executado pelo gerente intermediário para realizar essa tarefa de gerenciamento é listado na figura 4.6.

No cenário apresentado acima a execução dessa tarefa poderia ser delegada ao gerente intermediário GI_1, que programaria o agente de monitoração M_1 e o agente de ação A_1. A opção pelo agente de monitoração M_1 deve-se ao fato de ele estar posicionado no mesmo segmento de rede que o servidor DNS, observando todas as requisições que são feitas a ele (e os respectivos retornos).

4.3.2 Contabilização dos acessos ao servidor HTTP

Um conjunto grande de tarefas de gerenciamento ligadas à contabilização de acessos a um servidor HTTP pode ser implantado usando a arquitetura. Algumas sugestões foram citadas nas seções 1.1 e 3.5.3. Uma delas, a contabilização do volume de requisições HTTP retornadas com erro, pode ser especificada da seguinte forma:

o gerente intermediário programa um agente de monitoração para que passe a monitorar a ocorrência do traço *Requisições HTTP retornadas com erro* (apresentado na seção 3.5.3). A partir desse momento, o gerente intermediário passa a consultar o agente a cada 60 segundos para obter o número de ocorrências do traço no intervalo (objeto *protocolDistStatsPkts*). Sempre que o valor obtido ultrapassar um limiar estabelecido pela tarefa (ex: 30 ocorrências, ou seja, 60 pacotes), a notificação *"O número de requisições HTTP retornadas com erro 4XX superou 30 ocorrências."* é enviada ao gerente de rede. É importante destacar que os limiares usados nesse e nos demais exemplos são hipotéticos; uma boa escolha desses limiares depende do conhecimento que o gerente possui sobre o que deva ser considerado tráfego normal e anômalo (*baseline*), que varia de um ambiente para outro.

O objeto *protocolDistStatsPkts* contabiliza o número de ocorrências do traço sem fazer distinção de origem e destino. Se o gerente estiver interessado em monitorar as ocorrências do traço resultantes do acesso a um servidor HTTP específico, então o objeto a ser consultado é o *alHostInPkts*. Se desejar uma granularidade ainda maior e estiver interessado em acompanhar a ocorrência do traço entre duas estações específicas, o objeto *alMatrixSDPkts* é o mais indicado (vide tabela 4.2).

Para monitorar os dois servidores HTTP presentes no cenário apresentado na figura 4.9 duas instâncias da tarefa de gerenciamento acima seriam delegadas ao gerente intermediário GI_1. A primeira instância, responsável pelo servidor localizado na rede externa, programaria o agente de monitoração M_1, enquanto a segunda programaria o agente M_2.

4.3.3 Medição do tempo de resposta de uma aplicação em rede

Essa tarefa consiste em monitorar o tempo médio de resposta das interações de um protocolo que constituem transações de uma aplicação em rede. Como o protocolo HTTP tem sido amplamente utilizado na implementação dessas aplicações (ex: comércio eletrônico), uma tarefa de gerenciamento bastante apropriada monitora o tempo médio de resposta de requisições HTTP que são retornadas com sucesso (código 2XX). Para tal, o gerente intermediário precisa inicialmente solicitar a um agente de monitoração que observe a ocorrência do traço GET – HTTP/1.1 2XX (muito similar ao ilustrado na figura 3.5.3). Iniciam-se então as consultas periódicas (ex: a cada 60 segundos) ao agente de monitoração para obter o tempo médio de resposta das transações HTTP que completaram com sucesso no último intervalo (objeto *ResponsivenessMean*). Se o valor obtido superar 12 (segundos) a notificação *"O tempo médio de resposta da aplicação X superou 12 segundos."* é enviada ao gerente.

No ambiente ilustrado pela figura 4.9, o servidor HTTP localizado na rede externa mereceria esse tipo de monitoração. Para tal, a tarefa supracitada deveria ser transferida para o gerente intermediário GI_1, que programaria o agente de monitoração M_1.

4.3.4 Monitoração da segurança de serviços e aplicações em rede

Os servidores DNS e HTTP localizados na rede externa estão vulneráveis a ataques maliciosos, oriundos da Internet. A monitoração da segurança nesses servidores consiste em verificar se estão sendo vítimas de varredura de portas, de ataques de negação de serviço, entre outros detectáveis através da monitoração passiva.

Uma tarefa de gerenciamento para monitorar se o servidor de DNS está sendo vítima de varredura de portas inicia pela programação do agente de monitoração para que passe a observar a ocorrência do traço *Sondagem de portas*. A tarefa prossegue com consultas periódicas ao agente de monitoração a cada 60 segundos para determinar o número de ocorrências do traço (objeto *alHostInPkts*). Se em um intervalo de consulta o número de ocorrências superar 30 (60 pacotes), a notificação *"O servidor de DNS sofreu uma sondagem de portas."* é enviada à estação de gerenciamento.

Outra tarefa interessante ligada a gerenciamento de segurança é a que permite identificar se o servidor de DNS está sendo vítima de um ataque por inundação. O gerente intermediário solicita ao agente de monitoração que passe a monitorar a ocorrência do traço *Ataque por inundação* e passa a consultá-lo a cada 60 segundos para obter o número de ocorrências em que o traço não completou com sucesso (objeto *UnsuccessfulTransactions*). Se o valor obtido for superior a 100, uma notificação é enviada ao gerente.

A implantação de ambas as tarefas no cenário apresentado na figura 4.9 seria realizada pela programação do gerente intermediário GI_1 e do agente de monitoração M_1. As mesmas tarefas poderiam ser implementadas para monitorar o servidor HTTP responsável pela Intranet. Nesse caso, o gerente intermediário precisaria programar o agente de monitoração M_2; além disso, as tarefas poderiam ser incrementadas com a solicitação de execução de uma ação, junto ao *firewall* (agente de ação A_2), que o reconfigurasse para não mais admitir a entrada, à rede interna, de pacotes oriundos das estações de onde partiu o tráfego hostil.

4.4 Análise da arquitetura

Ao longo do capítulo foram apresentadas as características da plataforma, os componentes que dela fazem parte e as interações estabelecidas entre esses componentes para permitir a implantação de tarefas de gerenciamento ligadas a protocolos de alto nível, serviços e aplicações em rede. Esta seção encerra o capítulo com uma reflexão crítica a respeito da arquitetura, traçando um paralelo entre os requisitos que ela deveria atender – ser *integrada*, *distribuída* e *rápida e facilmente adaptável* (abordados na seção 1.3) – e o que, de fato, ela oferece. Outros aspectos como impacto do uso da arquitetura na infra-estrutura, aderência ao *framework* SNMP e segurança também foram considerados.

4.4.1 Quanto à integração

As tarefas de gerenciamento apresentadas na seção 4.3 e os respectivos traços (seção 3.5) ilustram a adequação da arquitetura para modelar procedimentos ligados a diferentes áreas funcionais de gerenciamento como falhas, contabilização, desempenho e segurança. Essas quatro áreas são, de certa forma, beneficiadas em função de a arquitetura se utilizar da técnica de monitoração passiva do tráfego de rede. O gerenciamento de configuração, de natureza ativa, pouco aproveita os benefícios proporcionados pela linguagem PTSL e pelos agentes de monitoração, mas pode ser realizado com a utilização da MIB Script (através de *scripts* de configuração), de forma descentralizada.

A flexibilidade recém mencionada permite que a arquitetura seja usada em

substituição a um conjunto de ferramentas específicas de gerenciamento (ex: para gerenciar servidor HTTP, disponibilidade de serviços e intrusões). Alguns dos benefícios obtidos com a utilização de uma arquitetura integrada incluem (a) redução no tempo de configuração do ambiente de gerenciamento, (b) menor esforço na manutenção desse ambiente, (c) maior facilidade de uso e (d) possibilidade de ter uma visão única do ambiente gerenciado.

As ferramentas que a arquitetura é capaz de substituir são aquelas que oferecem informações que podem ser obtidas a partir da análise do tráfego de rede ou através de consultas a bases de informações de gerenciamento (MIBs). É o caso das aplicações que monitoram protocolos de alto nível como Tivoli Web Services Manager e Tivoli Web Services Analyser, das aplicações que monitoram a disponibilidade de serviços como NetSaint [NET 2002] e, até mesmo, dos sistemas de detecção de intrusão baseados em rede como Snort [SNO 2002] e Bro [PAX 98].

A sobreposição entre as funcionalidades oferecidas pelas ferramentas supracitadas e pela arquitetura não é completa. Um estudo que aborda as vantagens e limitações da arquitetura Trace quando utilizada como um sistema de detecção de intrusão menciona esse fato [MEN 2001, MEN 2002]. Certamente as ferramentas somadas oferecem mais funcionalidades que a arquitetura. Há que se considerar, por outro lado, que o contrário também acontece, ou seja, há funcionalidades que são oferecidas pela arquitetura Trace que não estão disponíveis nas ferramentas (ex: capacidade de correlacionar pacotes). Assim, a eliminação dessas ferramentas depende da importância das funcionalidades em uso que não são suportadas pela arquitetura. Um ponto a favor da arquitetura, nesse caso, é a sua capacidade de ser facilmente estendida para suportar novas funcionalidades (aspecto analisado adiante).

4.4.2 Quanto à distribuição

Ao abordar os ganhos e prejuízos obtidos com a proposta de uma arquitetura para gerenciamento descentralizado, o critério macro a ser considerado é a sua escalabilidade. Para um detalhamento maior na análise da arquitetura Trace esse critério foi sub-dividido nos seguintes tópicos: tráfego de gerenciamento, robustez e manutenibilidade da arquitetura, carga da estação central de gerenciamento e desempenho do agente de monitoração (componente chave para validar a arquitetura).

Tráfego de gerenciamento

Pelo fato de as tarefas de gerenciamento serem baseadas em *polling*, não há redução sensível no volume de tráfego de gerenciamento (se a arquitetura for comparada a uma solução centralizada). Entretanto, como o maior número de interações SNMP ocorre entre o gerente intermediário e os agentes de monitoração a ele subordinados, esse tráfego tende a ficar confinado em cada domínio de gerenciamento. Se os domínios forem organizados considerando a localização geográfica da infraestrutura, o uso da arquitetura proporcionará uma redução considerável do tráfego de gerenciamento nos canais que interligam esses domínios, cujo uso é comumente cobrado por pacote transmitido.

Uma visão mais precisa sobre esse volume de tráfego pode ser obtida a partir de um exemplo. A figura 4.10 ilustra uma rede corporativa organizada em quatro domínios, interligados por um *backbone* de baixa velocidade (64 Kbit/s). Considere que em cada domínio deseja-se implantar quinze tarefas de gerenciamento, seme-

lhantes às ilustradas na seção 4.3. Considere, ainda, que o intervalo entre consultas aos agentes de monitoração, para cada tarefa, é de 60 segundos.

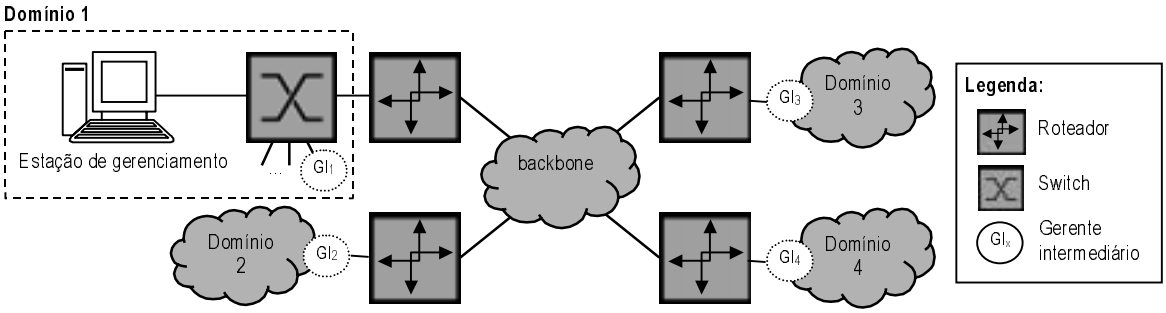


FIGURA 4.10 – Uma rede corporativa com *backbone* de baixa velocidade

Segundo medições realizadas por Hia, em [HIA 2001], o tamanho de um pacote IP carregando uma requisição SNMPv3 do tipo *GetRequest*, com autenticação e privacidade, ao objeto *sysContact* é de 167 bytes. O pacote IP enviado em resposta a essa requisição possui 191 bytes. Se uma requisição como essa é disparada 60 vezes a cada 60 segundos, tem-se aproximadamente $[(167+191)\times 8]\times 60$ bits sendo gerados a cada 60 segundos. Desconsiderou-se nesse cálculo o *overhead*, em cada pacote, do protocolo do nível de enlace e os pacotes de descoberta gerados pelo SNMPv3 [STA 99].

A implantação do cenário supracitado, considerando uma arquitetura centralizada de gerenciamento, provocaria $[(167+191)\times 8]\times 45$, ou seja, 128.880 bits de tráfego, a cada 60 segundos, no canal que interliga o domínio 1 (onde se encontra a estação de gerenciamento) ao restante da rede[1]. Considerando que o canal tem capacidade para transmitir 64.000×60 (3.840.000) bits em 60 segundos, a taxa de utilização do canal com tráfego de gerenciamento é dada por $(128.880 \div 3.840.000)\times 100$, o que resulta em 3,35 por cento. Essa taxa subiria para 33,5 por cento se, ao invés de 45, houvesse 450 tarefas (cenário possível em uma rede de maior porte), tornando o gerenciamento inviável.

Com a utilização da arquitetura Trace esse tráfego não percorre os canais que interligam os domínios, permanecendo dentro das sub-redes. No exemplo acima, o montante de $[(167+191)\times 8]\times 60$, isto é, 171.840 bits a cada 60 segundos seria dividido igualmente entre as quatro sub-redes. Ainda que esse volume de tráfego fosse 100 vezes maior, vale destacar que as tecnologias de redes locais operam em taxas bastante superiores aos canais WAN e, portanto, esse volume não comprometeria o seu desempenho.

O tráfego entre a estação de gerenciamento e os gerentes intermediários, provocado pela delegação e acompanhamento das tarefas de gerenciamento e pelo recebimento de *traps*, é pouco freqüente e de pequena proporção, não consumindo demasiadamente a banda dos canais. Segundo medição realizada por Hia em [HIA 2001], o volume de tráfego gerado para delegar a execução de um *script* de 18.110 bytes a um gerente intermediário é de 26.512 bytes (incluindo sua instalação, passagem de parâmetros e solicitação de sua instanciação).

[1] As quinze tarefas restantes encontram-se no mesmo domínio onde se encontra a estação de gerenciamento e, portanto, sua execução não provoca tráfego nos canais WAN.

Robustez e manutenibilidade

A robustez proporcionada pela arquitetura ao processo de execução de tarefas de gerenciamento merece ser comentada. Ao delegar a sua execução a gerentes intermediários que estejam bem perto dos agentes observados, essas tarefas podem ser executadas mesmo que ocorram problemas de comunicação entre a estação central de gerenciamento e o gerente intermediário (situação comum, por exemplo, quando existem interconexões baseadas em canais WAN pouco confiáveis).

Por ser distribuída e baseada na execução remota de *scripts*, a arquitetura exige mais trabalho para ser controlada. Tarefas como distribuição e atualização de *scripts*, desnecessárias em outras soluções, precisam ser realizadas na arquitetura Trace. Para impedir inconsistências nesse processo, (a) cada usuário do ambiente de gerenciamento (apresentado no próximo capítulo) tem acesso apenas às tarefas, traços e *scripts* de ação definidos por ele próprio. Além disso, (b) todos os *scripts* são sempre gerados pelo ambiente de gerenciamento, no momento em que a execução de uma tarefa é solicitada, e transferidos para os respectivos componentes da arquitetura (gerente intermediário, agente de monitoração e agente de ação); (c) sempre que uma tarefa for interrompida ou finalizar espontaneamente, os *scripts* envolvidos na sua execução são eliminados da MIB Script para evitar que *scripts* semelhantes ou idênticos comecem a se acumular nos componentes da arquitetura. Ao operacionalizar as três ações citadas (a, b e c), garante-se que a MIB Script dos gerentes intermediários, agentes de monitoração e agentes de ação conterá apenas *scripts* de tarefas em andamento.

Carga da estação central de gerenciamento

A estação de gerenciamento, que em arquiteturas centralizadas é responsável por realizar todas as consultas, receber e processar os resultados, é a maior beneficiada na arquitetura Trace. O tráfego originado por ela limita-se ao provocado pela delegação de tarefas aos gerentes intermediários e pelo acompanhamento, em geral esporádico, de tarefas em andamento. Quanto ao tráfego recebido, este se restringe às notificações que são enviadas pelos gerentes intermediários. A redução de tráfego oriundo e recebido pela estação de gerenciamento vem seguida de um drástico relaxamento da carga da CPU e demais recursos da estação.

Conforme comentado anteriormente, a sobrecarga originalmente concentrada na estação de gerenciamento é, agora, distribuída entre os gerentes intermediários. A carga de cada gerente intermediário está diretamente relacionada com o número de tarefas que lhe forem delegadas. Assim, é importante destacar a relevância da organização dos domínios de gerenciamento, a fim de balancear as tarefas a serem executadas entre os gerentes intermediários e evitar que estes sejam sobrecarregados.

Desempenho do agente de monitoração

Por se tratar de um componente chave para a arquitetura, realizou-se algumas medições para identificar o desempenho do protótipo do agente de monitoração. O ambiente de teste foi montado em um segmento isolado de rede local IEEE 802.3 (100 Mbit/s) composto por três estações: uma gerando tráfego, uma recebendo tráfego e a terceira com o agente de monitoração. O agente foi executado em uma estação K6II 450 MHz com 64 Mbytes de memória RAM. Os resultados obtidos foram os

seguintes:

- A capacidade sustentada do agente em termos de datagramas/segundo está em torno de 30 datagramas/segundo, com um traço sendo monitorado. Este número foi obtido pela geração de consecutivas seqüências de datagramas UDP que correspondessem à especificação do traço utilizado;
- Aumentando o número de traços monitorados o desempenho do agente sofre degradação. Com 5 traços e tráfego gerado especialmente para esses traços, a capacidade do agente é reduzida para 22 datagramas/segundo;
- Gerando tráfego na ordem de 10 Mbits/s (em torno de 5000 datagramas/segundo), com todos os datagramas fazendo parte do traço, o agente descarta pacotes;
- Em uma situação de tráfego normal, em uma rede operando a 10 Mbits/s, com o agente configurado para monitorar a ocorrência de 10 traços, não se observou descarte de pacotes.

A abordagem baseada na observação de traços a partir da monitoração passiva do tráfego de rede mostrou-se onerosa computacionalmente. A partir dos resultados obtidos, investigou-se como obter melhor desempenho dos agentes de monitoração a fim de que suportem tráfego mais intenso. As soluções encontradas foram:

- *Uso de estações com mais de um processador*: como a implementação utiliza ostensivamente *threads*, a existência de mais de um processador seria benéfica;
- *Distribuição dos traços em mais de uma estação*: como os traços não possuem ligação entre si, a programação de diferentes traços em diferentes agentes não causa nenhum tipo de prejuízo; assim, os agentes, atuando de modo passivo na captura e observação do tráfego, assumiriam funções de inspeção complementares uns aos outros;
- *Implementação de filtros BPF na biblioteca de captura*: filtrando previamente os pacotes que contêm os encapsulamentos esperados pelos traços, reduziria a quantidade de pacotes que precisariam passar pela rotina de análise dos pacotes; essa otimização, no entanto, alteraria a semântica de funcionamento do agente RMON2;
- *Eliminação do banco de dados mySQL*: o armazenamento das contabilizações realizadas pelo agente é realizado em uma base mySQL; a utilização da memória para esse fim reduziria o tempo de atualização da base e das consultas SNMP direcionadas ao agente.

4.4.3 Quanto à adaptabilidade

A adaptabilidade da arquitetura é analisada, a seguir, considerando dois enfoques: quanto à dificuldade para definir uma nova tarefa de gerenciamento e quanto à dificuldade para definir outros tipos de tarefas de gerenciamento (diferentes das ilustradas ao longo deste capítulo).

Especificação de uma nova tarefa de gerenciamento

No final do capítulo 2 discutiu-se sobre a dificuldade de personalizar as soluções existentes para gerenciar ou simplesmente monitorar um novo cenário. Para tal elas dependem de atualização do *firmware* do equipamento de monitoração, da programação em linguagens de baixo nível, da utilização de SDKs (*Software Development Kits*), entre outros artifícios. Essa dificuldade inexiste na arquitetura Trace. A definição de traços de protocolos, tarefas de gerenciamento e *scripts* de ação proporciona grande flexibilidade e facilidade ao processo de especificação de novos cenários.

Definição de outros tipos de tarefa de gerenciamento

As tarefas de gerenciamento ilustradas ao longo do trabalho são similares quanto ao seu comportamento: (a) elas programam um agente de monitoração para observar um traço, (b) passam a consultar o agente periodicamente e (c) solicitam a execução de um *script* a um agente de ação quando o valor obtido nessa consulta ultrapassa um determinado limiar. Outros tipos de tarefa, entretanto, podem ser imaginadas. Um exemplo seria uma que realizasse (a), (b), armazenasse localmente o valor obtido em cada consulta e gerasse um gráfico (como o ilustrado na figura 4.11), que pudesse ser posteriormente recuperado pelo gerente.

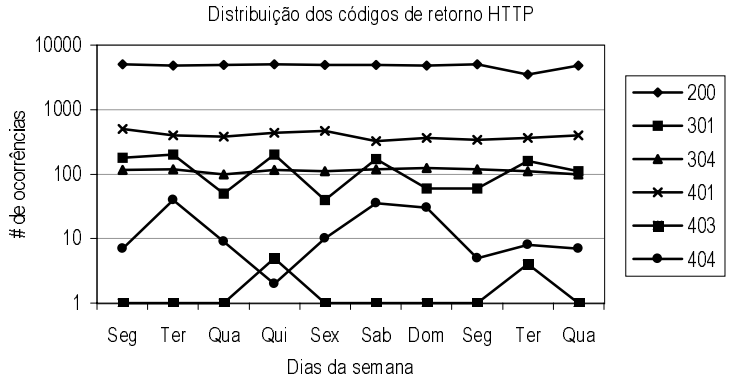


FIGURA 4.11 – Histórico dos códigos de retorno enviados pelo servidor HTTP

A arquitetura Trace permite a execução de tarefas que apresentem comportamento diferente, como o exemplo supracitado. Entretanto, o suporte à especificação das mesmas não é oferecido pelo ambiente de gerenciamento (detalhado no próximo capítulo). Duas alternativas são então possíveis: (a) especificar a tarefa de gerenciamento sem fazer uso do ambiente, ou seja, descrever os *scripts* diretamente em Tcl, ou (b) estender o ambiente de gerenciamento para que passe a permitir a especificação desse tipo de tarefa.

Se a linguagem Tcl for familiar ao gerente, a especificação da tarefa não será difícil. Vale ressaltar, no entanto, que com maior freqüência serão especificados *scripts* com erros léxicos e sintáticos, que não ocorreriam se o ambiente pudesse ser utilizado. Por outro lado, a modificação do ambiente de gerenciamento pode ser implementada com uma certa facilidade. Por ser desenvolvido usando a linguagem PHP, a extensão do ambiente para contemplar um novo tipo de tarefa requer (a) a programação dos formulários e *scripts* que os manipularão, (b) a extensão da base

de dados para armazenar o novo tipo de tarefa e (c) a criação de um *script* que converta a especificação em alto nível da tarefa para um *script* Tcl.

4.4.4 Quanto ao impacto do uso da arquitetura na infra-estrutura

Um dos méritos da arquitetura é a utilização de uma abordagem totalmente não intrusiva. As estações alvo de monitoração não precisam executar software de gerenciamento, as aplicações a serem gerenciadas não precisam ter o seu código fonte modificado, e não é necessário gerar transações sintéticas para poder medir o seu desempenho (ao contrário das iniciativas apresentadas na seção 2.5). O único impacto provocado pela arquitetura à infra-estrutura é o tráfego SNMP que, se bem distribuído, não prejudicará o funcionamento da rede.

A implantação da arquitetura pode ser realizada em uma estrutura paralela, como ilustrado na figura 4.9. Com exceção dos agentes de ação, os gerentes intermediários e os agentes de monitoração podem residir em estações dedicadas a essa tarefa. Assim, a implantação e a manutenção da arquitetura pode ocorrer sem que seja necessário interromper o funcionamento normal da rede, serviços e aplicações.

4.4.5 Quanto à aderência ao *framework* de gerenciamento SNMP

Baseada em um padrão para gerenciamento distribuído proposto pelo IETF (MIB Script), a arquitetura Trace adere completamente a sistemas SNMP em uso nas organizações. O uso de MIBs como fonte de informação para a implantação de tarefas de gerenciamento torna a arquitetura Trace aberta, ao contrário de outras abordagens que se utilizam de mecanismos proprietários.

4.4.6 Quanto à segurança

Em relação à segurança, a MIB Script suporta todas as facilidades oferecidas pelo protocolo SNMPv3, incluindo o USM (*User-based Security Model*) e o VACM (*View-based Access Control*). Schönwälder e Quittek descrevem em [SCH 99] os aspectos de segurança relacionados à MIB Script. Ao utilizar essas facilidades garante-se que os agentes de monitoração e ação não serão reprogramados por uma pessoa não autorizada a fazê-lo. Além disso, a utilização do protocolo HTTPS para armazenar e recuperar *scripts* do repositório confere o sigilo e a inviolabilidade dos mesmos quando estiverem em trânsito.

5 A Plataforma de Gerenciamento Trace

Este capítulo apresenta o protótipo de um ambiente, a plataforma de gerenciamento Trace [GAS 2002a, GAS 2002b], desenvolvido para permitir a manipulação facilitada da arquitetura descrita no capítulo anterior. A plataforma não é de propósito geral como, por exemplo, HP OpenView e Sun NetManager, mas especialmente voltada a oferecer suporte (a) à especificação de traços, *scripts* de ação e tarefas de gerenciamento, (b) à delegação coordenada desses elementos aos respectivos componentes da arquitetura e (c) ao acompanhamento de sua execução. O capítulo está organizado em duas partes: a seção 5.1 descreve as funcionalidades disponíveis e a seção 5.2 aborda aspectos ligados à implementação da plataforma.

5.1 Funcionalidades oferecidas

A figura 5.1 ilustra a janela principal da plataforma, apresentada ao gerente logo após a autenticação bem sucedida. Cabe ressaltar que o acesso à mesma se dá através do protocolo HTTPS. As funções disponíveis aparecem no menu à esquerda. Como é possível observar, elas estão dispostas em três grupos: configuração, especificação e operação. Ao primeiro grupo estão associadas as funcionalidades ligadas ao cadastramento de gerentes intermediários, agentes de monitoração e ação, linguagens e *traps*. O grupo especificação oferece mecanismos para a descrição de traços de protocolos, *scripts* de ação e tarefas de gerenciamento. Por fim, no grupo operação estão disponíveis as funcionalidades para a delegação e o acompanhamento de tarefas de gerenciamento e para a visualização de *traps* recebidas. Algumas dessas funções são detalhadas nas sub-seções que seguem.

FIGURA 5.1 – Funções disponíveis na plataforma

5.1.1 Cadastramento de gerentes intermediários e agentes

O cadastramento de um gerente intermediário é apresentado na figura 5.2. Ao realizar esse cadastramento é preciso informar um nome fantasia para o gerente intermediário e seu endereço IP, uma breve descrição sobre ele, a porta em que está aguardando requisições SNMP e as comunidades de leitura e escrita (b).

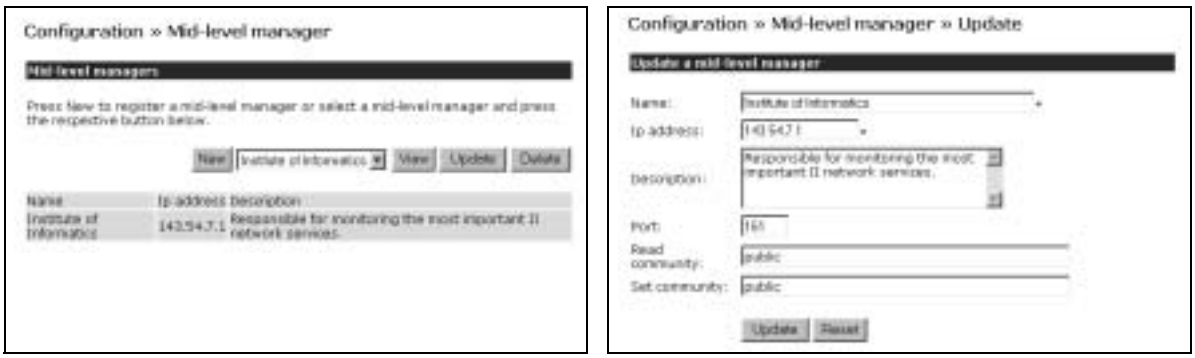


FIGURA 5.2 – Cadastramento de um gerente intermediário

O cadastramento de agentes de monitoração e ação é realizado de forma similar. Além das mesmas informações requeridas para o cadastramento de um gerente intermediário, é preciso determinar a que gerente intermediário o agente sendo cadastrado irá ficar subordinado (vide figura 5.3).

FIGURA 5.3 – Cadastramento de um agente de monitoração

5.1.2 Especificação de um traço

Traços de protocolos são especificados na plataforma com o auxílio de um *wizard*, conforme pode ser observado na figura 5.4. A especificação ocorre em cinco passos: *Identification*, *Fields*, *Messages*, *Groupers* e *State machine*.

Em *Identification* (b) são informados o nome fantasia do traço, sua versão, uma breve descrição do mesmo, a porta TCP ou UDP do protocolo sendo especificado (quando o traço for restrito a um único protocolo de aplicação) e algumas palavras-chave.

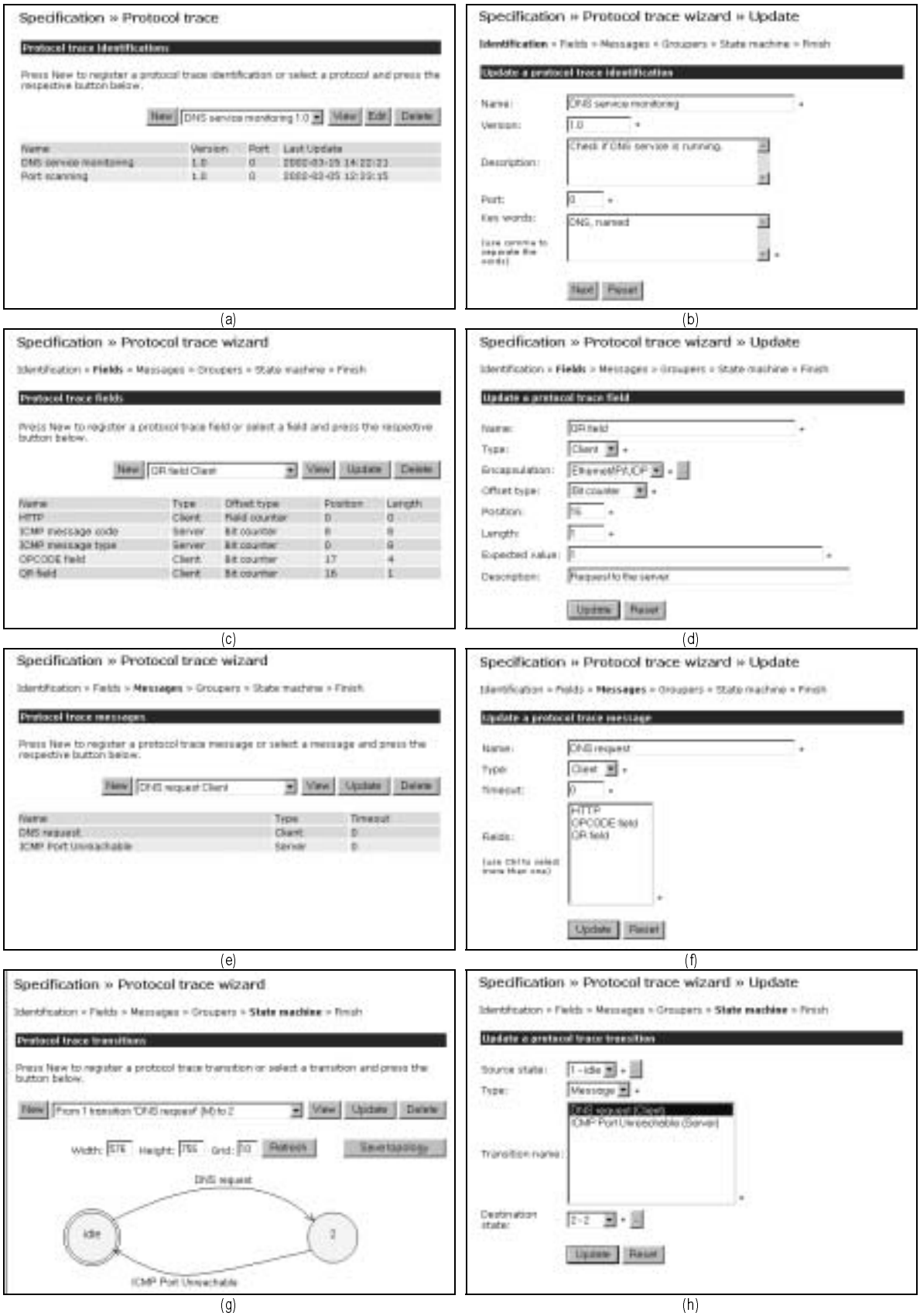


FIGURA 5.4 – Especificação de um traço

A identificação de campos de protocolos é realizada, em seguida, no passo *Fields* (c e d). As informações necessárias a essa identificação são: o nome fantasia do campo, seu tipo (*Client* ou *Server*), seu encapsulamento, a estratégia a ser utilizada para sua localização (*FieldCounter*, *BitCounter* ou *NoOffset*), sua posição, seu comprimento, o valor esperado e sua descrição.

Os campos descritos no passo *Fields* podem, então, ser usados para definir as mensagens que farão parte do traço (e e f) (passo *Messages*). Para cada mensagem é preciso informar um nome fantasia e um tipo (*Client* ou *Server*). Em seguida, é preciso selecionar os campos (do mesmo tipo, previamente definidos) que dela fazem parte.

O *wizard* prossegue com a definição dos agrupamentos (omitida na figura) e, por fim, da máquina de estados no passo *State machine* (g e h). Nesse ponto são criados estados e transições. Ao criar uma transição é preciso identificar dois estados (origem e destino) e uma mensagem ou agrupamento (previamente definido). À medida que o traço vai sendo modelado através da manipulação dos formulários, a representação gráfica do mesmo, em conformidade com G-PTSL, vai sendo apresentada.

5.1.3 Especificação de um *script* de ação

A especificação de um *script* de ação é realizada com o apoio do formulário apresentado na figura 5.5. Para incluir um novo *script* no repositório é preciso informar um nome fantasia para o mesmo, uma breve descrição a seu respeito, a linguagem em que foi desenvolvido e o seu código fonte, que pode ser digitado ou carregado de um arquivo (b).

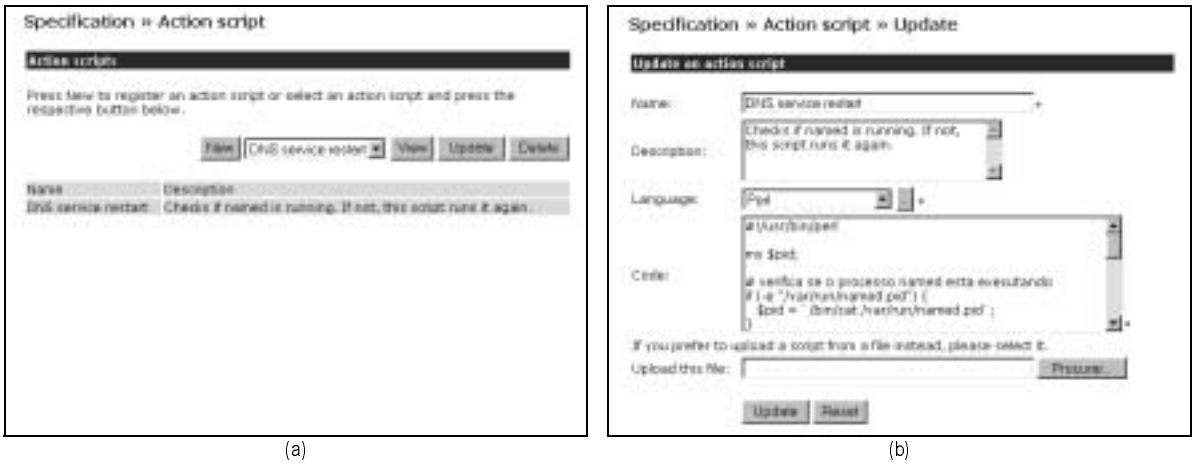


FIGURA 5.5 – Especificação de um *script* de ação

5.1.4 Especificação de uma tarefa de gerenciamento

A especificação de uma tarefa de gerenciamento também é realizada com o apoio de um *wizard*, em três passos: *Identification*, *Monitoring* e *Action* (vide figura 5.6. No passo *Identification* são informados o nome fantasia e a descrição da tarefa (b). No passo *Monitoring* são informados o traço a ser observado, o OID (*Object Identifier*) do objeto da MIB RMON2 estendida a ser consultado, o intervalo entre consultas (em segundos) e o tipo de amostragem (*Absolute* ou *Delta*) (c). O *wizard* encerra com o passo *Action* (d e e). Nesse passo é possível incluir uma ou mais

regras que serão aplicadas a cada consulta realizada ao objeto identificado no passo anterior. A definição de uma regra compreende a especificação de um operador de comparação (ex: *is greater than*), de um limiar, da identificação de uma ação a ser executada (previamente definida) e de uma notificação a ser enviada à estação de gerenciamento (e).

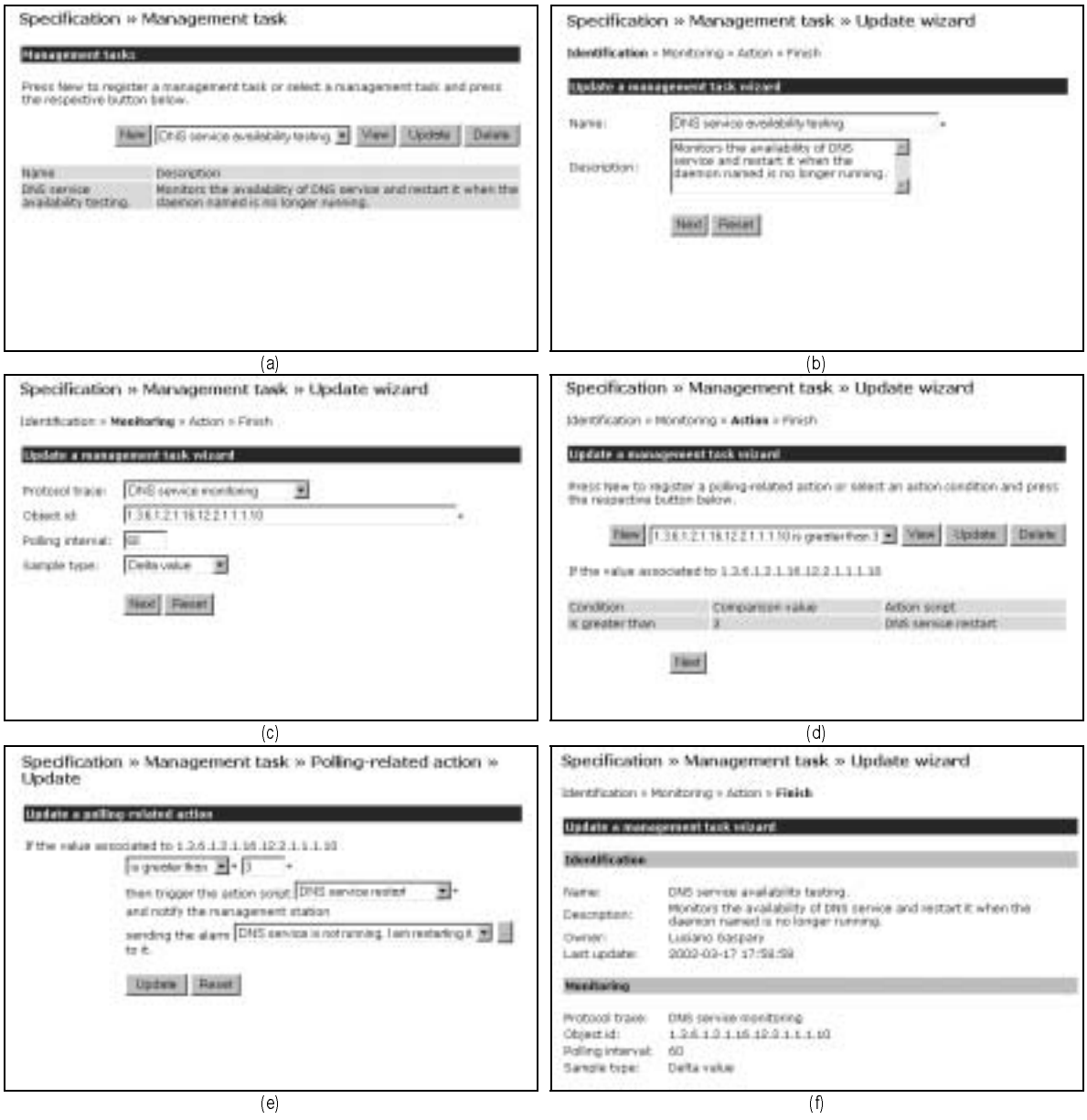


FIGURA 5.6 – Especificação de uma tarefa de gerenciamento

5.1.5 Delegação de uma tarefa de gerenciamento

A figura 5.7 ilustra a seqüência de passos a serem realizados para delegar uma tarefa de gerenciamento. Novamente um *wizard*, organizado em *Delegate to*, *Mid-level manager*, *Monitoring agent* e *Action agent*, é utilizado. No primeiro passo, *Delegate to*, o gerente informa o gerente intermediário, o agente de monitoração e o agente de ação (quando for o caso) que serão responsáveis pela execução da tarefa (b). Nos próximos passos (c, d e e), o gerente informa parâmetros para a execução

dos *scripts*. São eles: argumentos a serem passados ao *script* (*Arguments*), número máximo de *scripts* idênticos que podem executar concorrentemente (*MaxRunning*), tempo máximo que o *script* pode executar (*LifeTime*) e tempo que o *script* permanece cadastrado na MIB (*ExpireTime*).

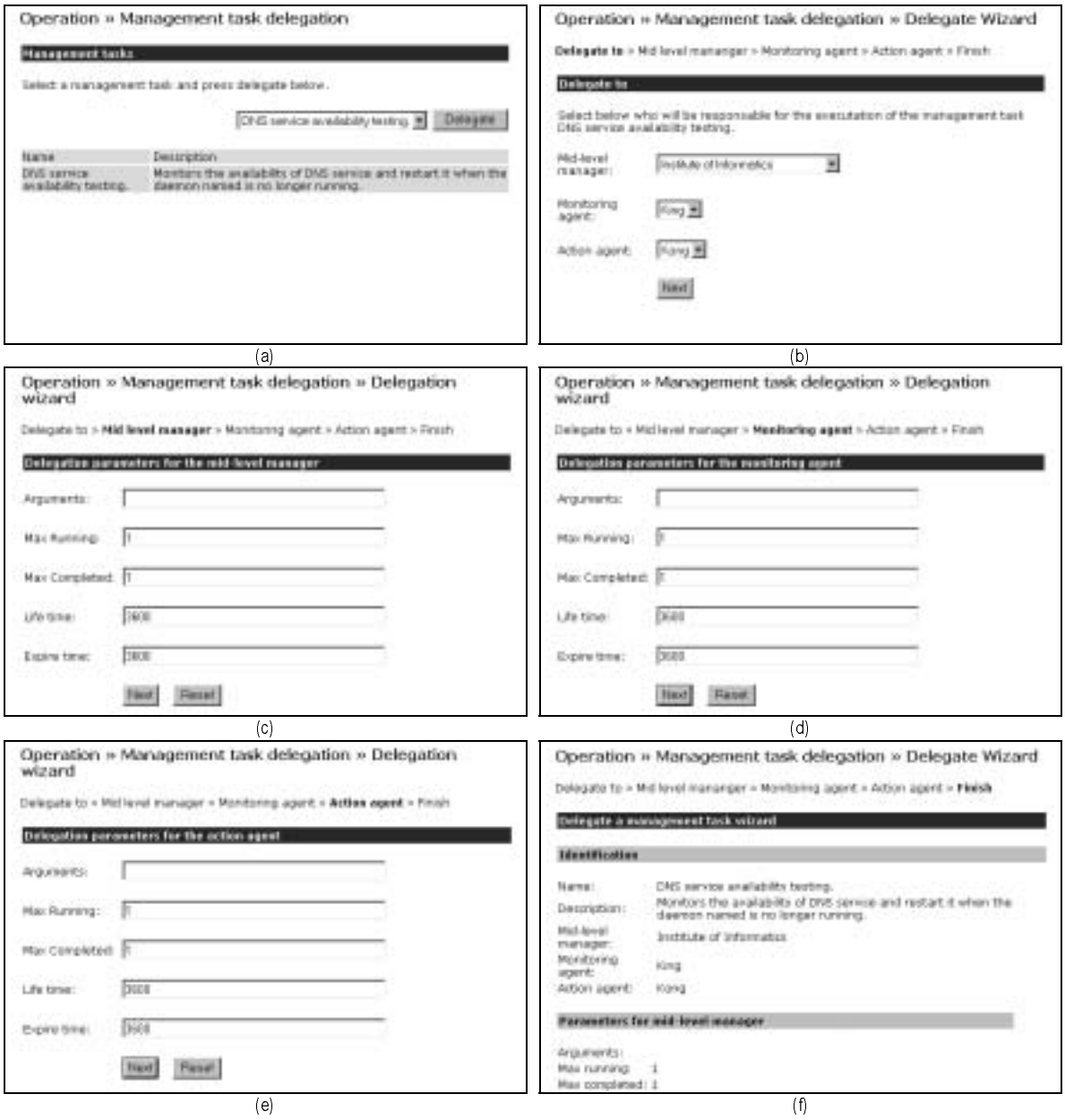


FIGURA 5.7 – Delegação de uma tarefa de gerenciamento

Como o objetivo desta seção foi apresentar uma visão geral da plataforma, apenas as funcionalidades mais significativas foram descritas. As relacionadas ao cadastramento de linguagens e *traps*, ao acompanhamento de tarefas de gerenciamento e ao recebimento de *traps* foram, por essa razão, omitidas.

5.2 Informações sobre a implementação

A plataforma de gerenciamento Trace foi desenvolvida sobre o sistema operacional Linux, utilizando o servidor HTTP Apache [APA 2001]. Os pacotes complementares instalados no servidor do sistema são o PHP4 [PHP 2001], linguagem de *script* utilizada na implementação dos processamentos no servidor, e o banco de dados mySQL [MYS 2001], usado para armazenar as informações manipuladas pelo usuário ao interagir com a plataforma. A figura 5.8 detalha o modelo de dados em uso.

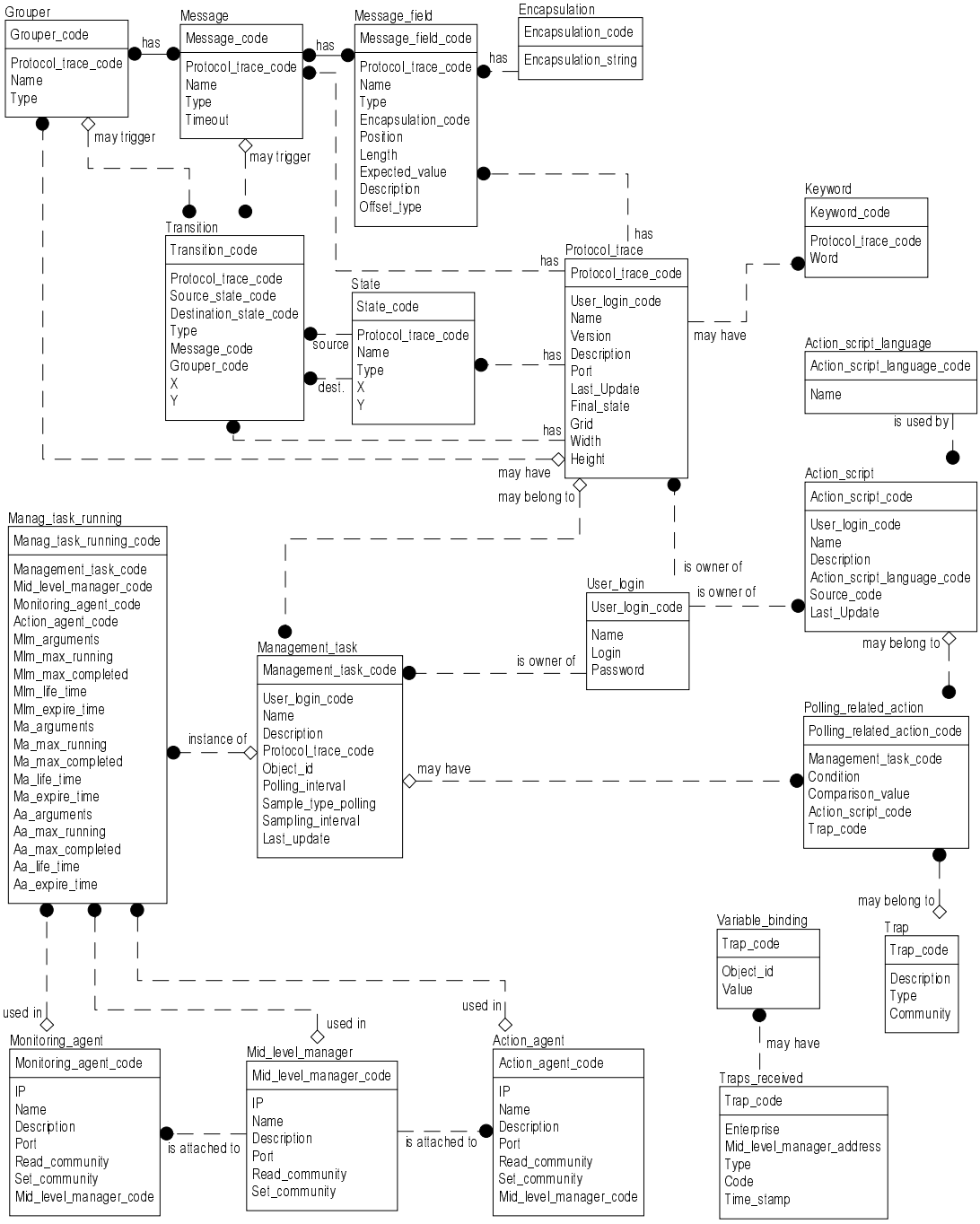


FIGURA 5.8 – Modelo de dados da plataforma Trace

6 Conclusões

Muitas pesquisas têm sido realizadas na área de gerenciamento de protocolos de alto nível, serviços e aplicações em rede. Conforme abordado no capítulo 1, e detalhado no 2, essas pesquisas vêm acompanhadas de um bom número de aplicações e plataformas que, embora suporte um crescente conjunto de funcionalidades, preocupa-se pouco (a) em contemplar diferentes áreas de gerenciamento, (b) em oferecer suporte à distribuição de tarefas e, principalmente, (c) em prover mecanismos flexíveis para a definição de novos cenários. Além dessas limitações, as soluções abusam de técnicas intrusivas para obter informações úteis ao gerenciamento (ex: execução de software de gerenciamento nas estações alvo de monitoração, instrumentação do código fonte de aplicações e utilização de tráfego sintético).

Esta tese propôs uma abordagem radicalmente oposta, ao explorar a técnica de observação passiva do tráfego de rede para a implantação de tarefas de gerenciamento. A identificação dos comportamentos dos protocolos, serviços e aplicações a serem monitorados é realizada através da linguagem PTSL (*Protocol Trace Specification Language*), abordada no capítulo 3, que permite a descrição de seqüências de pacotes (PDUs) a serem observadas. As especificações resultantes podem ser vistas como *assinaturas*, que são usadas para modelar tarefas de gerenciamento ligadas a falhas, contabilização, desempenho e segurança. Para viabilizar a abordagem proposta criou-se a arquitetura de gerenciamento Trace, apresentada no capítulo 4, que estende a infra-estrutura centralizada de gerenciamento SNMP para, através de um modelo em três níveis, suportar o gerenciamento distribuído de protocolos de alto nível, serviços e aplicações em rede. Para facilitar o manuseio da arquitetura, desenvolveu-se um ambiente de gerenciamento, a plataforma Trace, introduzido no capítulo 5.

6.1 Contribuições da tese

Os resultados obtidos são favoráveis à abordagem proposta, conforme foi antecipado nas seções 3.6 e 4.4. Cabe, nesse momento, listar as principais contribuições da tese e destacar algumas considerações e conclusões mais gerais sobre essas contribuições.

A linguagem PTSL

A linguagem gráfica/textual proposta (a) é de fácil utilização, (b) permite a especificação de traços variados (contemplando diversas áreas funcionais de gerenciamento), (c) permite a definição de seqüências de pacotes a serem observados (característica não explorada em nenhuma outra aplicação ou plataforma estudada), (d) suporta a representação de traços de protocolos de quaisquer níveis (incluindo os de transporte e aplicação) e (e) oferece condições de representar cenários com mais de uma seqüência possível de interações.

As características supracitadas conferem à linguagem flexibilidade e poder de expressão, sem perda de conforto no processo de especificação de traços. Credita-se esse conforto à utilização do conceito de máquinas de estado, amplamente utilizado na área de redes de computadores, e à definição de uma linguagem descritiva, ao

invés de uma procedural.

Com relação aos objetivos da tese, a linguagem contribuiu para a criação de um ambiente *integrado*, ou seja, não limitado a uma área de gerenciamento específica e, principalmente, para que este fosse *rápida e facilmente adaptável*.

A arquitetura Trace

Com relação à arquitetura, sua contribuição mais evidente é (a) suportar o gerenciamento de protocolos de alto nível, serviços e aplicações em rede, ao permitir a delegação de tarefas de gerenciamento a gerentes intermediários, que interagem com agentes de monitoração e agentes de ação para executá-las. A arquitetura não se limita à monitoração. Ao contrário, (b) provê uma solução mais completa e abrangente que inclui a execução de ações, viabilizando a automação de procedimentos.

Conforme detalhado na seção 4.4, a utilização da arquitetura contribui (c) para o gerenciamento *integrado*, tornando desnecessário o uso de um conjunto de aplicações e plataformas para gerenciar protocolos, serviços e aplicações específicos. Por ser *descentralizada*, a arquitetura permite (d) o confinamento de tráfego de gerenciamento nos domínios (evitando que passe pelo *backbone*) e (e) o desafogamento da estação central de gerenciamento (divisão de carga entre os gerentes intermediários), (f) além de oferecer maior robustez à execução de tarefas. Por ser distribuída, a arquitetura exige mais trabalho para ser controlada. A operacionalização do procedimento listado na seção 4.4, no entanto, elimina esse problema.

O agente de monitoração apresentou comportamento apenas razoável, mostrando-se inadequado para monitorar segmentos de rede de alta velocidade. Credita-se suas limitações à quantidade de processamento que precisa realizar para cada pacote recebido e, sobretudo, a alguns aspectos da implementação, apresentados na seção 4.4, que merecem ser revistos.

Outras contribuições relevantes da arquitetura são (g) sua capacidade de adaptação para monitorar novos cenários, (h) o pequeno impacto provocado na infra-estrutura existente durante a sua implantação, (i) a não exigência de ruptura com o que já existe implantado em boa parte das organizações (que demorou anos até ser consolidado) e (j) a segurança proporcionada pela mesma na criação, delegação e execução das tarefas de gerenciamento.

Identificação de cenários de aplicação da arquitetura

A investigação de um conjunto significativo de cenários de gerenciamento permitiu comprovar que a linguagem PTSL e a arquitetura Trace se prestam ao gerenciamento de diversas áreas funcionais. Os cenários identificados representam uma contribuição importante da tese, uma vez que enquanto anomalias e comportamentos observados na infra-estrutura física da rede vêm sendo mapeados desde o surgimento da arquitetura SNMP, poucos esforços têm sido realizados para realizar o trabalho nas camadas superiores.

Implementação de um protótipo

Essencial para a validação da arquitetura, a implementação do protótipo comprovou sua exeqüibilidade e permitiu a experimentação da arquitetura (com ênfase ao agente de monitoração). Por ter sido desenvolvida com software aberto e gratuito,

a implementação fica disponível (a) para utilização (com custo zero) e, principalmente, (b) para ser usada como base para outros projetos na área.

6.2 Trabalhos futuros

O encerramento desta tese é acompanhado do levantamento de um conjunto de atividades que, se realizadas, enriquecerão o trabalho já desenvolvido:

- *Adaptação/extensão da linguagem para representação de diálogos*: a linguagem PTSL foi criada para permitir que gerentes de rede possam especificar diálogos/traços de protocolos os quais tenham interesse em monitorar. Da forma como foi proposta originalmente, a linguagem oferece construtores que permitem a especificação de cenários básicos ligados a falhas, contabilização, desempenho e segurança. Estudos realizados recentemente apontam, no entanto, que a linguagem apresenta limitações quando se procura representar traços ligados à segurança [MEN 2002]. Esse problema foi contornado com a extensão da mesma, conforme mencionado na seção 3.6. Assim, é oportuno realizar o mesmo tipo análise considerando as demais áreas funcionais de gerenciamento e, se for o caso, estender a linguagem para que ela contemple a definição de um conjunto maior de cenários;

- *Redesenho de aspectos ligados à facilidade de uso da plataforma*: o protótipo da plataforma foi desenvolvido com base em tecnologias Web, através da linguagem PHP [PHP 2001] e do banco de dados mySQL [MYS 2001]. Assim, toda a interação do gerente de rede com a plataforma (ex: na especificação de novas tarefas de gerenciamento e no acompanhamento das mesmas) se dá através de formulários HTML (*Hypertext Markup Language*). Por ser um tanto quanto burocrática, a plataforma acaba desviando a atenção do gerente para problemas que não necessariamente dizem respeito à gerência de protocolos, serviços e aplicações. Nesse contexto, sugere-se a implementação de mecanismos que ofereçam um maior conforto na utilização da mesma. Entre esses mecanismos estão incluídas a descoberta/visualização automática da rede, a recuperação de traços do repositório através de palavras-chave e a possibilidade de reusar cenários previamente especificados na descrição de novos traços;

- *Otimização do agente de monitoração*: o aprimoramento do agente de monitoração constitui um dos trabalhos futuros mais relevantes. Algumas sugestões para torná-lo mais eficiente foram apresentadas na seção 4.4 e merecem ser consideradas. Soma-se a elas a necessidade de investigar estruturas de dados que tornem mais ágil o processo de análise dos pacotes capturados;

- *Geração de informações sobre desempenho*: embora tenha sido proposta uma MIB para armazenar informações sobre o desempenho de transações, ela não foi implementada no protótipo da plataforma. Assim, dada a definição de uma transação a ser monitorada, apenas o número de ocorrências dessa transação em um determinado intervalo de tempo é contabilizado. A implementação dessa MIB é aconselhável porque permitirá a implantação das tarefas de gerenciamento ligadas a desempenho como as ilustradas no capítulo 4;

- *Criação de mecanismos para facilitar instalação e configuração da plataforma*: conforme já comentado, a plataforma foi desenvolvida com base em tecnologias Web. Além da linguagem PHP [PHP 2001] e do banco de dados mySQL [MYS 2001], outras tecnologias de código aberto e livre foram utilizadas, a saber: os pacotes NET-SNMP [NES 2002] e Jasmin [TUB 99]. Como conseqüência, o processo de instalação e configuração da plataforma é complicado. Assim, implementar mecanismos que tornem esse processo menos trabalhoso (o mais automatizado possível) é recomendável.

Anexo 1 Descrição formal da linguagem PTSL

Este anexo descreve a gramática textual correspondente à sintaxe da linguagem *Textual* PTSL. Para tal foi utilizada uma BNF simplificada, similar à utilizada por Almeida em [ALM 94]. As extensões à linguagem, apresentadas na seção 3.6, foram omitidas da especificação.

```
<Char> ::= qualquer caracter ASCII (octetos 0-127)
<Char-sp> ::= qualquer caracter ASCII, exceto espaço
<UpAlpha> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
              R | S | T | U | V | W | X | Y | Z
<LoAlpha> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q |
              r | s | t | u | v | w | x | y | z
<Alpha> ::= <UpAlpha> | <LoAlpha>
<Digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<CR> ::= caracter ASCII, carriage return (13)
<LF> ::= caracter ASCII, linefeed (10)
<SP> ::= caracter ASCII, espaço (32)
<UL> ::= caracter ASCII, underline (95)
<DQ> ::= caracter ASCII, aspas duplas (34)
<DT> ::= caracter ASCII, ponto final (46)
<SL> ::= caracter ASCII, barra (47)
<WildCard> ::= caracter ASCII, asterisco (42)
<CRLF> ::= <CR> <LF>
<Informal text> ::= <Char>+
<Double quotted informal text> ::= <DQ> <Informal text> <DQ>
<Protocol name> ::= <Alpha> [<Alpha>]*
<Protocol encapsulation> ::= <Protocol name> [<SL><Protocol name>]*
<Keyword> ::= <Alpha> [<Alpha> | <SP>]*
<Identifier> ::= <Alpha> [<Alpha> | <Digit> | <UL> | <SP>]*
<Double quotted identifier> ::= <DQ> <Identifier> <DQ>
<Verb identifier> ::= <Char-sp>+
<Unsigned int> ::= qualquer número inteiro entre 0 e 65.535
<Version number> ::= <Digit>+ <DT> <Digit>+
<Date and time> ::= Wkday "," SP Date SP Time SP "GMT"
<Wkday> ::= "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"
<Date> ::= <Digit><Digit> SP Month SP <Digit><Digit><Digit><Digit>
<Month> ::= "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul" | "Aug" | "Sep" |
            "Oct" | "Nov" | "Dec"
<Time> ::= <Digit><Digit> ":" <Digit><Digit> ":" <Digit><Digit>

TraceSpecification ::= Trace <SP> <Double quotted identifier> <CRLF>
                       [<Header>]
                       <MessagesSection>
                       <GroupsSection>
                       <StatesSection>
                       EndTrace
<Header> ::= [Version: <Version number> <CRLF>]
             [Description: <Informal text> <CRLF>]
             [Key: <Keyword> {, <Keyword>}* <CRLF>]
             [Port: <Unsigned int> <CRLF>]
             [Owner: <Informal text> <CRLF>]
             [Last Update: <Date and time> <CRLF>]
<MessagesSection> ::= MessagesSection
                      <MessagesSectionContent>
                      EndMessagesSection
```

```
<GroupsSection> ::= GroupsSection
                    <GroupsSectionContent>
                    EndGroupsSection
<StatesSection> ::= StatesSection
                    <StatesSectionContent>
                    EndStatesSection

<MessagesSectionContent> ::= <MessageContent>+
<MessageContent> ::= Message <SP> <Double quotted identifier> <CRLF>
                     MessageType: {client | server} <CRLF>
                     [MessageTimeout: <Unsigned int>} <CRLF>]
                     {<FieldCounter> | <BitCounter> | <NoOffset>}+
                     EndMessage <CRLF>
<FieldCounter> ::= FieldCounter <SP>
                   <Protocol encapsulation> <SP>
                   <Unsigned int> <SP>
                   {<WildCard>|<Verb identifier>} <SP>
                   <Double quotted informal text> <CRLF>
<BitCounter> ::= BitCounter <SP>
                 <Protocol encapsulation> <SP>
                 <Unsigned int> <SP>
                 <Unsigned int> <SP>
                 {<WildCard>|<Verb identifier>} <SP>
                 <Double quotted informal text> <CRLF>
<NoOffset> ::= NoOffset <SP>
               <Protocol encapsulation> <SP>
               <Verb identifier> <SP>
               <Double quotted informal text> <CRLF>

<GroupsSectionContent> ::= <GroupContent>+
<GroupContent> ::= Group <SP> <Double quotted identifier> <CRLF>
                   Messages: <Double quotted identifier>
                   {, <Double quotted identifier>}* <CRLF>
                   EndGroup <CRLF>

<StatesSectionContent> ::= FinalState <SP> <Identifier> <CRLF>
                           <StateContent>+
<StateContent> ::= State SP <Identifier> <CRLF>
                   {<Double quotted identifier> <SP> GotoState <SP>
                   <Identifier>}+ <CRLF>
                   EndState <CRLF>
```

Anexo 2 Descrição da MIB APM resumida

Este anexo descreve a MIB APM resumida, proposta na seção 4.2.3. Essa MIB corresponde a um sub-conjunto da MIB proposta por Waldbusser em [WAL 2002], sendo composta somente pelos grupos *apmAppDirectory* e *apmReport*. Apenas um objeto, *UnsuccessfulTransactions*, foi acrescentado à definição original.

```
APM-MIB DEFINITIONS ::= BEGIN
IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    NOTIFICATION-TYPE,
    Counter32, Integer32, Unsigned32            FROM SNMPv2-SMI
    TEXTUAL-CONVENTION, RowStatus, TimeStamp,
    TimeInterval, TruthValue, DateAndTime       FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP,
    NOTIFICATION-GROUP                          FROM SNMPv2-CONF
    SnmpAdminString                             FROM SNMP-FRAMEWORK-MIB
    rmon, OwnerString                           FROM RMON-MIB
    DataSource, protocolDirLocalIndex           FROM RMON2-MIB;

apm MODULE-IDENTITY
    LAST-UPDATED "200202271500Z"  -- February 27, 2002
    ORGANIZATION "IETF RMON MIB Working Group"
    CONTACT-INFO
        "Steve Waldbusser

        Phone: +1-650-948-6500
        Fax:   +1-650-745-0671
        Email: waldbusser@nextbeacon.com"
    DESCRIPTION
        "The MIB module for measuring application performance
        as experienced by end-users."

    REVISION "200202271500Z"    -- February 27, 2002
    DESCRIPTION
        "The original version of this MIB, published as RFCXXXX."
    ::= { rmon 23 }

AppLocalIndex ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
        "A locally arbitrary unique identifier associated with an
        application or application verb.

        All objects of type AppLocalIndex are assigned by the agent
        out of a common number space. In other words, AppLocalIndex
        values assigned to entries in one table must not overlap with
        AppLocalIndex values assigned to entries in another
        table. Further, every protocolDirLocalIndex value registered
        by the agent automatically assigns the same value out of the
        AppLocalIndex number space.

        For example, if the protocolDirLocalIndex values { 1, 3, 5, 7 }
        have been assigned, and the apmHttpFilterAppLocalIndex values
        { 6, 8, 9 } have been assigned:
```

```
            - Assignment of new AppLocalIndex values must not use the
              values { 1, 3, 5, 6, 7, 8, 9 }.
            - AppLocalIndex values { 1, 3, 5, 7 } are automatically
              assigned and are associated with the identical value of
              protocolDirLocalIndex. In particular, an entry in the
              apmAppDirectoryTable indexed by a value provides further
              information about a protocol indexed by the same value
              in the protocolDirectoryTable of RMON2.

        The value for each supported application must remain constant
        at least from one re-initialization of the entity's network
        management system to the next re-initialization, except that
        if an application is deleted and re-created, it must be
        re-created with a new value that has not been used since the
        last re-initialization.

        The specific value is meaningful only within a given SNMP
        entity. An AppLocalIndex value must not be re-used until the
        next agent restart."
    SYNTAX       Unsigned32 (1..2147483647)

-- The APM Application Directory Group

apmAppDirectoryTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ApmAppDirectoryEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The APM MIB directory of applications and application
        verbs. The agent will populate this table with all
        applications/verbs of any responsivenessType it has the
        capability to monitor. Since the agent populates this table
        with every entry it has the capability to monitor, the
        entries in this table are read-write, allowing the management
        station to modify parameters in this table but not to add new
        entries or delete entries (however, entries may be
        disabled). If new entries are added to the apmHttpFilterTable
        or the apmUserDefinedAppTable, the agent will add the
        corresponding entries to this table.

        It is an implementation-dependent matter as to how the agent
        sets these default parameters. For example, it may leave
        certain entries in this table 'off(0)' if the agent developer
        believes that combination will be infrequently used, allowing
        a manager that needs that capability to set it to 'on(1)'.

        Some applications are registered in the RMON2 protocol
        directory and some are registered in other tables in this
        MIB. Regardless of where an application is originally
        registered, it is assigned an AppLocalIndex value that is the
        primary index for this table.

        The contents of this table affect all reports and exceptions
        generated by this agent. Accordingly, modification of this
        table should be performed by a manager acting in the role of
        administrator. In particular, management software should not
        require or enforce particular configuration of this table - it
        should reflect the preferences of the site administrator, not
```

```
        the software author. As a practical matter, this requires
        management software to allow the administrator to configure
        the values it will use so that it can be adapted to the site
        policy."
    ::= { apm 1 }

apmAppDirectoryEntry OBJECT-TYPE
    SYNTAX      ApmAppDirectoryEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The APM MIB directory of applications and application
        verbs. An entry will exist in this table for all applications
        for which application performance measurement is supported."
    INDEX { apmAppDirectoryAppLocalIndex }
    ::= { apmAppDirectoryTable 1 }

ApmAppDirectoryEntry ::= SEQUENCE {
    apmAppDirectoryAppLocalIndex            AppLocalIndex,
    apmAppDirectoryConfig                   INTEGER,
    apmAppDirectoryResponsivenessBoundary1  Integer32,
    apmAppDirectoryResponsivenessBoundary2  Integer32,
    apmAppDirectoryResponsivenessBoundary3  Integer32,
    apmAppDirectoryResponsivenessBoundary4  Integer32,
    apmAppDirectoryResponsivenessBoundary5  Integer32,
    apmAppDirectoryResponsivenessBoundary6  Integer32
}

apmAppDirectoryAppLocalIndex OBJECT-TYPE
    SYNTAX      AppLocalIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The AppLocalIndex assigned for this application Directory
        entry."
    ::= { apmAppDirectoryEntry 1 }

apmAppDirectoryConfig OBJECT-TYPE
    SYNTAX      INTEGER {
                  off(0),
                  on(1)
                }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "This object describes and configures support for application
        performance measurement for this application.

        If the value of this object is on(1), the agent supports
        measurement of application performance metrics for this
        application and is configured to measure such metrics for all
        APM MIB functions and all interfaces.  If the value of this
        object is off(0), the agent supports measurement of
        application performance for this application but is configured
        to not measure these metrics for any APM MIB functions or
        interfaces.  Whenever this value changes from on(1) to off(0),
        the agent shall delete all related entries in all tables in
        this MIB."
```

```
    ::= { apmAppDirectoryEntry 3 }

apmAppDirectoryResponsivenessBoundary1 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The boundary value between bucket1 and bucket 2. If this
        value is modified, all entries in the apmReportTable must be
        deleted."
    ::= { apmAppDirectoryEntry 4 }

apmAppDirectoryResponsivenessBoundary2 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The boundary value between bucket2 and bucket 3. If this
        value is modified, all entries in the apmReportTable must be
        deleted."
    ::= { apmAppDirectoryEntry 5 }

apmAppDirectoryResponsivenessBoundary3 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The boundary value between bucket3 and bucket 4. If this
        value is modified, all entries in the apmReportTable must be
        deleted."
    ::= { apmAppDirectoryEntry 6 }

apmAppDirectoryResponsivenessBoundary4 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The boundary value between bucket4 and bucket 5. If this
        value is modified, all entries in the apmReportTable must be
        deleted."
    ::= { apmAppDirectoryEntry 7 }

apmAppDirectoryResponsivenessBoundary5 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The boundary value between bucket5 and bucket 6. If this
        value is modified, all entries in the apmReportTable must be
        deleted."
    ::= { apmAppDirectoryEntry 8 }

apmAppDirectoryResponsivenessBoundary6 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The boundary value between bucket6 and bucket 7. If this
```

```
        value is modified, all entries in the apmReportTable must be
        deleted."
    ::= { apmAppDirectoryEntry 9 }

-- The APM Name Table

apmNameTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ApmNameEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A client machine may have multiple addresses during a period
        of monitoring. The apmNameTable assigns a long-lived
        identifier to a client and records what addresses were
        assigned to that client for periods of time. Various
        implementation techniques exist for tracking this mapping but
        if an agent is unable to track client address mappings, it may
        map client identifiers to client addresses rather than to
        distinct client machines.

        A particular apmNameClientID should be a constant attribute of
        a particular client. When available, the agent may also record
        the machine name and/or user name which may be valuable for
        displaying to humans. The apmNameMachineName and
        apmNameUserName are relatively constant, changing only if
        these attributes actually change on the client.

        The agent will store a historical log of these entries, aging
        out old entries as the log becomes too large. Since this table
        contains information vital to the interpretation of other
        tables (e.g. the apmReportTable), the agent should ensure that
        the log doesn't age out entries that would be referenced by
        data in those tables.

        Note that an entry for a clientID is active from it's
        StartTime until the StartTime of another entry (for the same
        clientID) that supercedes it, or 'now' if none supercede
        it. Therefore, if a clientID only has a single entry, it is by
        definition very new and should never be aged out. No entry for
        a clientID should be aged out unless it has been updated by a
        new entry for the client (i.e. with an updated address) and
        only if the new entry is 'old' enough.

        To determine how old is old enough, compute the maximum value
        of Interval * (NumReports + 1) of all entries in the
        apmReportControlTable (the '+ 1' is to allow a reasonable
        period of time for the report to be downloaded). Then take the
        larger of this value and the age in seconds of the oldest
        entry in the current transaction table. If an entry for a
        clientID is superceded by another entry whose StartTime is
        more than this many seconds ago, then the older entry may be
        deleted."
    ::= { apm 8 }

apmNameEntry OBJECT-TYPE
    SYNTAX      ApmNameEntry
    MAX-ACCESS  not-accessible
    STATUS      current
```

```
    DESCRIPTION
        "An entry in the APM name table. An entry exists for each
        period of time that a client has been associated with a
        particular address.

        The protocolDirLocalIndex value in the index identifies
        the network layer protocol for the ClientAddress for this
        entry."
    INDEX { apmNameClientID,
            protocolDirLocalIndex, apmNameClientAddress,
            apmNameMappingStartTime }
    ::= { apmNameTable 1 }

ApmNameEntry ::= SEQUENCE {
    apmNameClientID                 Unsigned32,
    apmNameClientAddress            OCTET STRING,
    apmNameMappingStartTime         DateAndTime,
    apmNameMachineName              SnmpAdminString,
    apmNameUserName                 SnmpAdminString
}

apmNameClientID OBJECT-TYPE
    SYNTAX      Unsigned32 (0..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "A unique ID assigned to the machine represented by this
         mapping. This ID is assigned by the agent using an
         implementation-specific algorithm."
    ::= { apmNameEntry 1 }

apmNameClientAddress OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The network client address for this client when this mapping
        was active.

        This is represented as an octet string with
        specific semantics and length as identified
        by the protocolDirLocalIndex component of the index.

        Since this object is an index variable, it is encoded in the
        index according to the index encoding rules.  For example, if
        the protocolDirLocalIndex component of the index indicates an
        encapsulation of ip, this object is encoded as a length octet
        of 4, followed by the 4 octets of the ip address, in network
        byte order."
    ::= { apmNameEntry 2 }

apmNameMappingStartTime OBJECT-TYPE
    SYNTAX      DateAndTime
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The time that the agent first discovered this mapping
        as active."
```

```
    ::= { apmNameEntry 3 }

apmNameMachineName OBJECT-TYPE
    SYNTAX      SnmpAdminString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The human readable name of the client machine.

        If the client has no machine name or the agent is
        unable to learn the machine name, this object will be
        a zero-length string."
    ::= { apmNameEntry 4 }

apmNameUserName OBJECT-TYPE
    SYNTAX      SnmpAdminString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The human readable name of a human user using the client
        machine. If more than one user name are available
        simultaneously, it is an implementation-dependent matter as to
        which is used here. However, if the user name changes, this
        object should change to reflect that change.

        Non-human user names like 'root' or 'administrator' aren't
        intended as values for this object. If the client has no
        recorded user name or the agent is unable to learn a user
        name, this object will be a zero-length string."
    ::= { apmNameEntry 5 }

-- The APM Report Group

apmReportControlTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ApmReportControlEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Parameters that control the creation of a set of reports that
        aggregate application performance."
    ::= { apm 9 }

apmReportControlEntry OBJECT-TYPE
    SYNTAX      ApmReportControlEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A conceptual row in the apmReportControlTable.

        An example of the indexing of this table is
        apmReportControlInterval.3"
    INDEX { apmReportControlIndex }
    ::= { apmReportControlTable 1 }

ApmReportControlEntry ::= SEQUENCE {
    apmReportControlIndex            Integer32,
    apmReportControlDataSource       DataSource,
    apmReportControlAggregationType  INTEGER,
```

```
    apmReportControlInterval         Unsigned32,
    apmReportControlRequestedSize    Unsigned32,
    apmReportControlGrantedSize      Unsigned32,
    apmReportControlRequestedReports Unsigned32,
    apmReportControlGrantedReports   Unsigned32,
    apmReportControlStartTime        TimeStamp,
    apmReportControlReportNumber     Unsigned32,
    apmReportControlInsertsDenied    Unsigned32,
    apmReportControlDroppedFrames    Counter32,
    apmReportControlOwner            OwnerString,
    apmReportControlStatus           RowStatus
}

apmReportControlIndex OBJECT-TYPE
    SYNTAX      Integer32 (1..65535)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An index that uniquely identifies an entry in the
        apmReportControlTable.  Each such entry defines a unique
        report whose results are placed in the apmReportTable on
        behalf of this apmReportControlEntry."
    ::= { apmReportControlEntry 1 }

apmReportControlDataSource OBJECT-TYPE
    SYNTAX      DataSource
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The source of the data for APM Reports generated on
        behalf of this apmReportControlEntry.

        If the measurement is being performed by a probe, this should
        be set to interface or port where data was received for
        analysis. If the measurement isn't being performed by a probe,
        this should be set to the primary interface over which the
        measurement is being performed. If the measurement isn't being
        performed by a probe and there is no primary interface or this
        information isn't known, this object should be set to 0.0.

        This object may not be modified if the associated
        apmReportControlStatus object is equal to active(1)."
    ::= { apmReportControlEntry 2 }

apmReportControlAggregationType OBJECT-TYPE
    SYNTAX      INTEGER {
                  flows(1),    -- Least Aggregation
                  clients(2),
                  servers(3),
                  applications(4) -- Most Aggregation
                }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The type of aggregation being performed for this set of
        reports.

        The metrics for a single transaction are the responsiveness of
```

```
        the transaction and whether the transaction succeeded (a
        boolean). When such metrics are aggregated in this MIB, these
        metrics are replaced by averages and distributions of
        responsiveness and availability. The metrics describing
        aggregates are constant no matter which type of aggregation is
        being performed. These metrics may be found in the
        apmReportTable.

        The flows(1) aggregation is the simplest. All transactions
        that share common application/server/client 3-tuples are
        aggregated together, resulting in a set of metrics for all
        such unique 3-tuples.

        The clients(2) aggregation results in somewhat more
        aggregation (i.e. fewer resulting records). All transactions
        that share common application/client tuples are aggregated
        together, resulting in a set of metrics for all such unique
        tuples.

        The servers(3) aggregation usually results in still more
        aggregation (i.e. fewer resulting records). All transactions
        that share common application/server tuples are aggregated
        together, resulting in a set of metrics for all such unique
        tuples.

        The applications(4) aggregation results in the most
        aggregation (i.e. the fewest resulting records). All
        transactions that share a common application are aggregated
        together, resulting in a set of metrics for all such unique
        applications.

        Note that it is not meaningful to aggregate applications, as
        different applications have widely varying characteristics. As a
        result, this set of aggregations is complete.

        This object may not be modified if the associated
        apmReportControlStatus object is equal to active(1)."
    ::= { apmReportControlEntry 3 }

apmReportControlInterval OBJECT-TYPE
    SYNTAX      Unsigned32
    UNITS       "Seconds"
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The interval in seconds over which data is accumulated before
        being aggregated into a report in the apmReportTable.  All
        reports with the same apmReportControlIndex will be based on
        the same interval. This object must be greater than zero.
        Many users desire that these reports be synchronized to within
        seconds of the beginning of the hour because the results may
        be correlated more meaningfully to business behavior and so
        that data from multiple agents is aggregated over the same
        time periods. Thus management software may take extra effort
        to synchronize reports to the beginning of the hour and to one
        another. However, the agent must not allow reports to 'drift'
        over time as they will quickly become unsynchronized. In
        particular, if there is any fixed processing delay between
```

```
        reports, the reports should deduct this time from the interval
        so that reports don't drift.

        This object may not be modified if the associated
        apmReportControlStatus object is equal to active(1)."
    DEFVAL { 3600 }
    ::= { apmReportControlEntry 4 }

apmReportControlRequestedSize OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The number of entries requested to be allocated for each
        report generated on behalf of this entry."
    ::= { apmReportControlEntry 5 }

apmReportControlGrantedSize OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of entries per report the agent has allocated
        based on the requested amount in apmReportControlRequestedSize.
        Since multiple reports are saved, the total number of entries
        allocated will be this number multiplied by the value of
        apmReportControlGrantedReports, or 1 if that object doesn't
        exist.

        When the associated apmReportControlRequestedSize object is
        created or modified, the agent should set this object as
        closely to the requested value as is possible for the
        particular implementation and available resources. When
        considering resources available, the agent must consider its
        ability to allocate this many entries for all reports.

        Note that while the actual number of entries stored in the
        reports may fluctuate due to changing conditions, the agent
        must continue to have storage available to satisfy the full
        report size for all reports when necessary. Further, the agent
        must not lower this value except as a result of a set to the
        associated apmReportControlRequestedSize object."
    ::= { apmReportControlEntry 6 }

apmReportControlRequestedReports OBJECT-TYPE
    SYNTAX      Unsigned32 (0..65535)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The number of saved reports requested to be allocated on
        behalf of this entry."
    ::= { apmReportControlEntry 7 }

apmReportControlGrantedReports OBJECT-TYPE
    SYNTAX      Unsigned32 (0..65535)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
```

```
        "The number of saved reports the agent has allocated
        based on the requested amount in
        apmReportControlRequestedReports. Since each report can have
        many entries, the total number of entries allocated will be
        this number multiplied by the value of
        apmReportControlGrantedSize, or 1 if that object doesn't
        exist.

        When the associated apmReportControlRequestedReports object is
        created or modified, the agent should set this object as
        closely to the requested value as is possible for the
        particular implementation and available resources. When
        considering resources available, the agent must consider its
        ability to allocate this many reports each with the number of
        entries represented by apmReportControlGrantedSize, or 1 if
        that object doesn't exist.

        Note that while the storage required for each report may
        fluctuate due to changing conditions, the agent must continue
        to have storage available to satisfy the full report size for
        all reports when necessary. Further, the agent must not lower
        this value except as a result of a set to the associated
        apmReportControlRequestedSize object."
    ::= { apmReportControlEntry 8 }

apmReportControlStartTime OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime when the system began processing the
        report in progress. Note that the report in progress is not
        available.

        This object may be used by the management station to figure
        out the start time for all previous reports saved for this
        apmReportControlEntry, as reports are started at fixed
        intervals."
    ::= { apmReportControlEntry 9 }

apmReportControlReportNumber OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of the report in progress. When an
        apmReportControlEntry is activated, the first report will be
        numbered zero."
    ::= { apmReportControlEntry 10 }

apmReportControlInsertsDenied OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of failed attempts to add an entry to reports for
        this apmReportControlEntry because the number of entries
        would have exceeded apmReportControlGrantedSize.
```

```
        This number is valuable in determining if enough entries have
        been allocated for reports in light of fluctuating network
        usage. Note that since an entry that is denied will often be
        attempted again, this number will not predict the exact number
        of additional entries needed, but can be used to understand
        the relative magnitude of the problem.

        Also note that there is no ordering specified for the entries
        in the report, thus there are no rules for which entries will
        be omitted when not enough entries are available. As a
        consequence, the agent is not required to delete 'least
        valuable' entries first."
    ::= { apmReportControlEntry 11 }

apmReportControlDroppedFrames OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "The total number of frames which were received by the agent
        and therefore not accounted for in the *StatsDropEvents, but
        for which the agent chose not to count for this entry for
        whatever reason.  Most often, this event occurs when the agent
        is out of some resources and decides to shed load from this
        collection.

        This count does not include packets that were not counted
        because they had MAC-layer errors.

        Note that if the apmReportTables are inactive because no
        applications are enabled in the application directory, this
        value should be 0.

        Note that, unlike the dropEvents counter, this number is the
        exact number of frames dropped."
    ::= { apmReportControlEntry 12 }

apmReportControlOwner OBJECT-TYPE
    SYNTAX      OwnerString
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The entity that configured this entry and is
        therefore using the resources assigned to it."
    ::= { apmReportControlEntry 13 }

apmReportControlStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The status of this apmReportControlEntry.

        An entry may not exist in the active state unless all
        objects in the entry have an appropriate value.

        If this object is not equal to active(1), all
```

```
        associated entries in the apmReportTable shall be deleted
        by the agent."
    ::= { apmReportControlEntry 14 }

-- The APM Report Table

apmReportTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ApmReportEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The data resulting from aggregated APM reports. Consult the
        definition of apmReportControlAggregationType for the
        definition of the various types of aggregations."
    ::= { apm 10 }

apmReportEntry OBJECT-TYPE
    SYNTAX      ApmReportEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A conceptual row in the apmReportTable.

        The apmReportControlIndex value in the index identifies the
        apmReportControlEntry on whose behalf this entry was created.
        The apmReportIndex value in the index identifies which report
        (in the series of reports) this entry is a part of.
        The apmAppDirectoryAppLocalIndex value in the index identifies
        the common application of the transactions aggregated in this
        entry.
        The apmAppDirectoryResponsivenessType value in the index
        identifies the type of responsiveness metric reported by
        this entry and uniquely identifies this entry when more
        than one responsiveness metric is measured for a flow.
        Entries will only exist in this table for those
        combinations of AppLocalIndex and ResponsivenessType
        that are configured 'on(1)'.
        The protocolDirLocalIndex value in the index identifies
        the network layer protocol of the apmReportServerAddress.
        When the associated apmReportControlAggregationType value is
        equal to application(4), this value will equal 0.
        The apmReportServerAddress value in the index identifies the
        network layer address of the server in transactions aggregated
        in this entry.
        The apmNameClientID value in the index identifies the
        client in transactions aggregated in this entry. If the
        associated apmReportControlAggregationType is equal to
        application(4) or server(3), then this object will be set to
        0.

        An example of the indexing of this entry is
        apmReportTransactionCount.3.15.3.1.8.4.192.168.1.2.3232235788"
    INDEX { apmReportControlIndex, apmReportIndex,
            apmAppDirectoryAppLocalIndex,
            protocolDirLocalIndex, apmReportServerAddress,
            apmNameClientID }
    ::= { apmReportTable 1 }
```

```
ApmReportEntry ::= SEQUENCE {
    apmReportIndex                  Integer32,
    apmReportServerAddress          OCTET STRING,
    apmReportTransactionCount       Integer32,
    apmReportSuccessfulTransactions Integer32,
    apmReportUnsuccessfulTransactions Integer32,
    apmReportResponsivenessMean     Integer32,
    apmReportResponsivenessMin      Integer32,
    apmReportResponsivenessMax      Integer32,
    apmReportResponsivenessB1       Integer32,
    apmReportResponsivenessB2       Integer32,
    apmReportResponsivenessB3       Integer32,
    apmReportResponsivenessB4       Integer32,
    apmReportResponsivenessB5       Integer32,
    apmReportResponsivenessB6       Integer32,
    apmReportResponsivenessB7       Integer32
}

apmReportIndex OBJECT-TYPE
    SYNTAX      Integer32 (0..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The value of apmReportControlReportNumber for the report to
        which this entry belongs."
    ::= { apmReportEntry 1 }

apmReportServerAddress OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The network server address for this apmReportEntry.

        This is represented as an octet string with
        specific semantics and length as identified
        by the protocolDirLocalIndex component of the index.

        Since this object is an index variable, it is encoded in the
        index according to the index encoding rules.  For example, if
        the protocolDirLocalIndex indicates an encapsulation of ip,
        this object is encoded as a length octet of 4, followed by the
        4 octets of the ip address, in network byte order.

        If the associated apmReportControlAggregationType is equal to
        application(4) or client(2), then this object will be a null
        string and will be encoded simply as a length octet of 0."
    ::= { apmReportEntry 2 }

apmReportTransactionCount OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The total number of transactions aggregated into this record."
    ::= { apmReportEntry 3 }

apmReportSuccessfulTransactions OBJECT-TYPE
```

```
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The total number of successful transactions aggregated into
        this record."
    ::= { apmReportEntry 4 }

apmReportUnsuccessfulTransactions OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The total number of unsuccessful transactions aggregated into
        this record."
    ::= { apmReportEntry 5 }

apmReportResponsivenessMean OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The arithmetic mean of the responsiveness metrics for all
        successful transactions aggregated into this record."
    ::= { apmReportEntry 6 }

apmReportResponsivenessMin OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The minimum of the responsiveness metrics for all
        successful transactions aggregated into this record."
    ::= { apmReportEntry 7 }

apmReportResponsivenessMax OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The maximum of the responsiveness metrics for all
        successful transactions aggregated into this record."
    ::= { apmReportEntry 8 }

apmReportResponsivenessB1 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of successful transactions aggregated into this
        record whose responsiveness was less than boundary1 value for
        this protocol."
    ::= { apmReportEntry 9 }

apmReportResponsivenessB2 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
```

```
    DESCRIPTION
        "The number of successful transactions aggregated into this
        record whose responsiveness did not fall into Bucket 1 and was
        greater than or equal to the boundary1 value for this
        application and less than the boundary2 value for this
        application."
    ::= { apmReportEntry 10 }

apmReportResponsivenessB3 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of successful transactions aggregated into this
        record whose responsiveness did not fall into Bucket 1 or 2
        and as greater than or equal to the boundary2 value for this
        application and less than the boundary3 value for this
        application."
    ::= { apmReportEntry 11 }

apmReportResponsivenessB4 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of successful transactions aggregated into this
        record whose responsiveness did not fall into Buckets 1
        through 3 and was greater than or equal to the boundary3 value
        for this application and less than the boundary4 value for
        this application."
    ::= { apmReportEntry 12 }

apmReportResponsivenessB5 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of successful transactions aggregated into this
        record whose responsiveness did not fall into Buckets 1
        through 4 and was greater than or equal to the boundary4 value
        for this application and less than the boundary5 value for
        this application."
    ::= { apmReportEntry 13 }

apmReportResponsivenessB6 OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of successful transactions aggregated into this
        record whose responsiveness did not fall into Buckets 1
        through 5 and was greater than or equal to the
        boundary5 value for this application and less than the
        boundary6 value for this application."
    ::= { apmReportEntry 14 }

apmReportResponsivenessB7 OBJECT-TYPE
    SYNTAX      Integer32
```

```
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of successful transactions aggregated into this
        record whose responsiveness did not fall into Buckets 1
        through 6 and was greater than or equal to the boundary6 value
        for this application."
    ::= { apmReportEntry 15 }

    rmonConformance   OBJECT IDENTIFIER ::= { rmon 20 }
    apmCompliances    OBJECT IDENTIFIER ::= { rmonConformance 11 }
    apmGroups         OBJECT IDENTIFIER ::= { rmonConformance 12 }

apmCompliance MODULE-COMPLIANCE
    STATUS  current
    DESCRIPTION
        "Describes the requirements for conformance to
        the APM MIB"
    MODULE  -- this module
        MANDATORY-GROUPS { apmAppDirectoryGroup, apmReportGroup }
    ::= { apmCompliances 1 }

apmAppDirectoryGroup OBJECT-GROUP
    OBJECTS { apmAppDirectoryConfig,
              apmAppDirectoryResponsivenessBoundary1,
              apmAppDirectoryResponsivenessBoundary2,
              apmAppDirectoryResponsivenessBoundary3,
              apmAppDirectoryResponsivenessBoundary4,
              apmAppDirectoryResponsivenessBoundary5,
              apmAppDirectoryResponsivenessBoundary6,
              apmNameMachineName, apmNameUserName }
    STATUS  current
    DESCRIPTION
        "The APM MIB directory of applications and application verbs."
    ::=  { apmGroups 1 }

apmReportGroup OBJECT-GROUP
    OBJECTS { apmReportControlDataSource,
              apmReportControlAggregationType,
              apmReportControlInterval,
              apmReportControlRequestedSize,
              apmReportControlGrantedSize,
              apmReportControlRequestedReports,
              apmReportControlGrantedReports,
              apmReportControlStartTime,
              apmReportControlReportNumber,
              apmReportControlInsertsDenied,
              apmReportControlDroppedFrames,
              apmReportControlOwner,
              apmReportControlStatus,
              apmReportTransactionCount,
              apmReportSuccessfulTransactions,
              apmReportUnsuccessfulTransactions,
              apmReportResponsivenessMean,
              apmReportResponsivenessMin,
              apmReportResponsivenessMax,
              apmReportResponsivenessB1,
              apmReportResponsivenessB2,
```

```
            apmReportResponsivenessB3,
            apmReportResponsivenessB4,
            apmReportResponsivenessB5,
            apmReportResponsivenessB6,
            apmReportResponsivenessB7 }
    STATUS  current
    DESCRIPTION
        "The apm report group controls the creation and retrieval of
        reports that aggregate application performance."
    ::=  { apmGroups 3 }

END
```

Bibliografia

[ALM 94] ALMEIDA, Maria Janilce B. **Especificação de Sistemas na Área de Redes de Computadores**: uma Abordagem Orientada a Objetos. 1994. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[APA 2001] APACHE. **Apache Homepage**. Disponível em: <http://www.apache.org/>. Acesso em: 2001.

[APR 2002] APRISMA. **Aprisma Management Technologies Homepage**. Disponível em: <http://www.aprisma.com/>. Acesso em: jan. 2002.

[ALB 98] ALBAYRAK, S.; GARIJ, F. J. (Ed.). **Intelligent Agents for Telecommunications Applications**. Berlin: Springer, 1998. (Lecture Notes in Artificial Intelligence, v.1437).

[BAR 97] BARILLAUD, Franck; DERI, Luca; FERIDUN, Metin. Network Management using Internet Technologies. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 5., 1997, San Diego, USA. **Proceedings...** London: Chapman & Hall, 1997.

[BIE 2000] BIERMAN, Andy; BUCCI, C.; IDDON, Robin. **Remote Network Monitoring MIB Protocol Identifier Reference**, RFC 2895. Aug. 2000. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[BOA 99] BOARDMAN, Bruce. Finding the Best Approach to App Monitoring. **Network Computing**, [S.l.], July 1999. Disponível em: <http://www.networkcomputing.com/1014/1014ws1.html>. Acesso em: jan. 2002.

[BLU 99] BLUMENTHAL, U.; WIJNEN, B. **User-based Security Model for (USM) for version 3 of the Simple Network Management Protocol**, RFC 2574. Apr. 1999. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[BRO 99a] BROWNLEE, Nevil. **Traffic Flow Measurement**: Meter MIB, RFC 2720. Oct. 1999. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[BRO 99b] BROWNLEE, Nevil. **RTFM**: Applicability Statement, RFC 2721. Oct. 1999. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[BRO 99c] BROWNLEE, Nevil; MILLS, Cyndi; RUTH, Greg. **Traffic Flow Measurement**: Architecture, RFC 2722. Oct. 1999. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[BRO 99d] BROWNLEE, Nevil. **SRL**: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups, RFC 2723. Oct. 1999. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[BRO 2001] BROWNLEE, Nevil. Using NeTraMet for Production Traffic Measurement. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 7., 2001, Seattle, USA. **Proceedings...** [S.l.: s.n.], 2001.

[BRU 96] BRUINS, Barry. Some Experiences with Emerging Management Technologies. **The Simple Times**, [S.l.], v. 4, n. 3, July 1996. Disponível em: <http://www.simple-times.org/pub/simple-times/issues/4-3.html>. Acesso em: maio 1999.

[CAL 98] CALVERT, K. L.; BHATTACHARJEE, S.; ZEGURA, E.; STERBENZ, J. Directions in Active Networks. **IEEE Communications Magazine**, New York, v. 36, n. 10, p. 72–78, Oct. 1998.

[CAN 2002] CANDLE. **Candle Corporation Homepage**. Disponível em: <http://www.candle.com>. Acesso em: jan. 2002.

[COM 2002] COMPUTER ASSOCIATES. **Computer Associates International, Inc. Homepage**. Disponível em: <http://www.ca.com/>. Acesso em: jan. 2002.

[CON 2002] CONCORD. **Concord Communications Inc. Homepage**. Disponível em: <http://www.concord.com>. Acesso em: jan. 2002.

[DER 99] DERI, Luca; SUIN, Stefano. Ntop: Beyond Ping and Traceroute. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 10., 1999, Zurich, Switzerland. **Proceedings...** Berlin: Springer, 1999. p. 271–283. (Lecture Notes in Computer Science, v.1700).

[DER 2000a] DERI, Luca; SUIN, Stefano. Effective Traffic Measurement using ntop. **IEEE Communications Magazine**, New York, v. 38, n. 5, p. 138–145, May 2000.

[DER 2000b] DERI, Luca; SUIN, Stefano. Improving Network Security Using ntop. In: WORKSHOP ON THE RECENT ADVANCES IN INTRUSION DETECTION, RAID, 2000, Toulouse, France. **Proceedings...** [S.l.: s.n.], 2000.

[ENG 98] ENGEL, Fred. Application Behavior and Statistics via RMON2. In: IEEE INTERNATIONAL WORKSHOP ON SYSTEMS MANAGEMENT, 3., 1998, New Port, Rhode Island, USA. **Proceedings...** [S.l.: s.n.], 1998.

[FIE 99] FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. **Hypertext Transfer Protocol - HTTP/1.1**, RFC 2616. June 1999. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[GAS 98] GASPARY, Luciano P. **Estudo do padrão RMON2**. 1998. Trabalho Individual (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[GAS 99a] GASPARY, Luciano P.; TAROUCO, Liane R. Distributed Applications and High-level Protocols Traffic Monitoring Based on RMON2 Accounting Mechanisms. In: JORNADAS ARGENTINAS DE INFORMÁTICA E INVESTIGACÍON OPERATIVA, REDES, 28., 1999, Buenos Aires, Argentine. **Proceedings...** Buenos Aires: [s.n.], 1999. p. 11–24.

[GAS 99b] GASPARY, Luciano P.; TAROUCO, Liane R. Characterization and Measurements of Enterprise Network Traffic with RMON2. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 10., 1999, Zurich, Switzerland. **Proceedings...** Berlin: Springer, 1999. p. 229–242. (Lecture Notes in Computer Science, v.1700).

[GAS 99c] GASPARY, Luciano P.; TAROUCO, Liane R. Managing Users, Applications and Resources with RMON2. In: IEEE GLOBAL TELECOMMUNICATIONS CONFERENCE, SYMPOSIUM ON ENTERPRISE APPLICATIONS AND SERVICES, GLOBECOM, 1999, Rio de Janeiro, Brazil. **Proceedings...** Piscataway, USA: IEEE Press, 1999. p. 1997–2001.

[GAS 2000a] GASPARY, Luciano P. **Gerenciamento de Protocolos de Alto Nível e Aplicações Distribuídas**. 2000. Exame de Qualificação (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[GAS 2000b] GASPARY, Luciano P.; TAROUCO, Liane R. An Extensible Approach for the Management of High-Layer Protocols and Services. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 11., 2000, Austin, USA. **Proceedings...** [S.l.: s.n.], 2000.

[GAS 2001a] GASPARY, Luciano P.; BALBINOT, Luis F.; STORCH, Roberto; WENDT, Fabrício; TAROUCO, Liane R. Towards a Programmable Agent-based Distributed Architecture for Enterprise Application and Service Management. In: IEEE/IEC ENTERPRISE NETWORKING APPLICATIONS AND SERVICES CONFERENCE, ENTNET, 2001, Atlanta, USA. **Proceedings...** Piscataway, USA: IEEE Operations Center, 2001. p. 39–46.

[GAS 2001b] GASPARY, Luciano P.; BALBINOT, Luis F.; STORCH, Roberto; WENDT, Fabrício; TAROUCO, Liane R. Uma Arquitetura para Gerenciamento Distribuído e Flexível de Protocolos de Alto Nível e Serviços de Rede. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 19., 2001, Florianópolis. **Anais...** Florianópolis: [s.n.], 2001.

[GAS 2001c] GASPARY, Luciano P.; BALBINOT, Luis F.; STORCH, Roberto; WENDT, Fabrício; TAROUCO, Liane R. Distributed Management of High-Layer Protocols and Network Services through a Programmable Agent-Based Architecture. In: INTERNATIONAL CONFERENCE ON NETWORKING, ICN, 1., 2001, Colmar, France. **Proceedings...** Berlin: Springer, 2001. p. 204–217. (Lecture Notes in Computer Science, v.2093, n. 1).

[GAS 2001d] GASPARY, Luciano P.; BALBINOT, Luis F.; TAROUCO, Liane R. Monitoring High-Layer Protocol Behavior Using the Trace Architecture. In: LATIN AMERICAN NETWORK OPERATION AND MANAGEMENT SYMPOSIUM, LANOMS, 2., 2001, Belo Horizonte. **Proceedings...** Belo Horizonte: [s.n.], 2001. p. 99–110.

[GAS 2002a] GASPARY, Luciano P.; MENEGHETTI, Edgar; WENDT, Fabrício; BRAGA, Lucio; STORCH, Roberto; TAROUCO, Liane R. Trace: An Open Platform for High-layer Protocolos, Services and Networked Applications Management. In: IFIP/IEEE NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 8., 2002, Florence, Italy. **Proceedings...** Piscataway, USA: IEEE Operations Center, 2002. p. 871–873.

[GAS 2002b] GASPARY, Luciano P.; MENEGHETTI, Edgar; WENDT, Fabrício; BRAGA, Lucio; STORCH, Roberto; TAROUCO, Liane R. High-layer Protocol and Service Management Based on Passive Network Traffic Monitoring: the Trace Management Platform. In: IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS, ISCC, 7., 2002, Taormina, Italy. **Proceedings...** [S.l.: s.n.], 2002.

[GOL 96] GOLDSZMIDT, German S. **Distributed Management by Delegation**. 1996. Ph.D Thesis – Graduate School of Arts and Sciences, Columbia University, New York.

[GRA 2000] GRANVILLE, Lisandro Z.; GASPARY, Luciano P.; TAROUCO, Liane R. Uma Proposta para Fornecimento de QoS para Aplicações Legadas em Redes TCP/IP. In: WORKSHOP DE TMN, 2000, Belo Horizonte. **Anais...** Belo Horizonte: [s.n.], 2000. p. 79–90.

[HAN 99] HANDELMAN, Sig; STIBLER, Stephen; BROWNLEE, Nevil; RUTH, Greg. **RTFM**: New Attributes for Traffic Flow Measurement, RFC 2724. Oct. 1999. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[HEG 99] HEGERING, Heinz-Gerd; ABECK, Sebastian; BERNHARD, Neumair. **Integrated Management of Networked Systems**. USA: Morgan Kaufmann, 1999.

[HEW 2002] HEWLETT-PACKARD. **Hewlett-Packard Company Homepage**. Disponível em: <http://www.hp.com/>. Acesso em: jan. 2002.

[HIA 2001] HIA, H. Erik **Secure SNMP-Based Network Management in Low Bandwidth Networks**. 2001. Master thesis – Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

[HON 97] HONG, J. W.; KONG, J. Y.; YUN, T. H. KIM, J. S.; PARK, J. T.; BAEK, J. W. Web-based Intranet Services and Network Management. **IEEE Communications Magazine**, New York, v. 35, n. 10, p. 100–110, Oct. 1997.

[IEE 95] IEEE. **Pthreads**: POSIX threads standard, IEEE Std 1003.1c-1995. [S.l.], 1995.

[ISO 91] ISO/EIC. Information Technology, Open Systems Interconnection. **Conformance Testing Methodology and Framework (CTMF)**, IS 9646. Geneva, 1991.

[ISO 92] ISO/EIC. Information Technology, Open Systems Interconnection. **OSI CTMF Part 3**: The Tree and Tabular Combined Notation, IS 96463. Geneva, 1992.

[ITU 94] ITU. **Recommendation X.680683**: Abstract Syntax Notation One (ASN.1). Geneva, 1994.

[ITU 96a] ITU. **Recommendation Z. 120**: Message Sequence Chart (MSC). Geneva, 1996.

[ITU 96b] ITU. **Recommendation M. 3010**: Principles for a Telecommunications Management Network. Geneva, 1996.

[KAR 98] KARMOUCH, A. (Guest Editor). Mobile Software Agents for Telecommunications. **IEEE Communications Magazine**, New York, USA, v. 36, n. 7, July 1998.

[KRI 97] TINA-C. **Service Architecture. Technical Report Version 5**. [S.l.], June 1997.

[LAR 99] LARMOUTH, J. **ASN.1 Complete**. [S.l.]: Open Systems Solutions, 1999.

[LEV 2001] LEVI, D.; SCHÖNWÄLDER, J. **Definitions of Managed Objects for the Delegation of Management Scripts**, RFC 3165. Aug. 2001. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[LUC 2002] LUCENT. **Lucent Technologies Homepage**. Disponível em: <http://www.lucent.com/>. Acesso em: jan. 2002.

[MAG 99] MAGEDANZ, T.; GLITHO, R. (Ed.). Mobile Agent-Based Network and Service Management. **Journal of Network and Systems Management**, USA, v. 7, Sept. 1999.

[MAR 97a] MARTIN-FLATIN, J. P.; ZNATY, S.; HUBAUX, J. P. A Survey of Distributed Network and Systems Management Paradigms. **Journal of Network and Systems Management**, USA, v.7, n. 1, p. 9–26, 1997.

[MAR 97b] MARTIN-FLATIN, J. P.; ZNATY, S. A Simple Typology of Distributed Network Management Paradigms. In: INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 8., 1998, Sydney, Australia. **Proceedings...** [S.l.: s.n.], 1998.

[MAR 2000] MARTIN-FLATIN, Jean-Philippe. **Web-Based Management of IP Networks and Systems**. 2000. Ph.D. Thesis – Swiss Fed. Inst. of Technology (EPFL), Lausanne.

[MAU 97] MAUW, S.; RENIERS, M. A. High-Level Message Sequence Charts. In: SDL FORUM, 8., 1997, Evry, France. **Proceedings...** [S.l.]: Elsevier Science Publishers, 1997.

[MAZ 1996] MAZUMDAR, S.; SWANSON, K. Web Based Management - CORBA/SNMP Gateway Approach. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, DSOM, 7., 1996, L'Aquila, Italy. **Proceedings...** [S.l.: s.n.], 1996.

[MCC 93] MCCANNE, S.; JACOBSON, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In: USENIX CONFERENCE, 1993, San Diego, USA. **Proceedings...** [S.l.: s.n.], 1993. p. 259–269.

[MEN 2001] MENEGHETTI, Edgar; GASPARY, Luciano P.; TAROUCO, Liane R. Uma Ferramenta de Monitoração Programável Voltada à Detecção de Intrusão. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, 2001, Merida, Venezuela. **Proceedings...** Merida: [s.n.], 2001. p. 131–135.

[MEN 2002] MENEGHETTI, Edgar; GASPARY, Luciano P.; TAROUCO, Liane R. Um Agente SNMP para Detecção de Intrusão baseada na Monitoração de Interações de Protocolos. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 20., 2002, Buzios. **Anais...** Rio de Janeiro: Núcleo de Computação Eletrônica (UFRJ), 2001. p. 831–845.

[MIC 2002] MICROMUSE. **Micromuse Inc. Homepage**. Disponível em: <http://www.micromuse.com/>. Acesso em: jan. 2002.

[MOC 87a] MOCKAPETRIS, P. **Domain Names – Concepts and Facilities**, RFC 1034. Nov. 1987. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[MOC 87b] MOCKAPETRIS, P. **Domain Names – Implementation and Specification**, RFC 1035. Nov. 1987. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[MOU 98] MOURA, Ana Lúcia; ISHIKAWA, Edison; LIMA, Michele; RODRIGUEZ, Noemi. Aplicações de Gerência Extensíveis. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 16., 1998, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 1998. p. 125–143.

[MYS 2001] MYSQL. **MySql AB Homepage**. Disponível em: <http://www.mysql.org/>. Acesso em: 2001.

[NET 2002] NETIQ. **NetIQ Corporation Homepage**. Disponível em: <http://www.netiq.com/>. Acesso em: jan. 2002.

[NER 2002] NETRAMET. **NeTraMet Homepage**. Disponível em: <http://www2.auckland.ac.nz/net/NeTraMet/>. Acesso em: mar. 2002.

[NES 2002] NET-SNMP. **NET-SNMP Project Homepage**. Disponível em: <http://net-snmp.sourceforge.net/>. Acesso em: jan. 2002.

[NEC 2002] NETSCOUT. **NetScout Systems, Inc. Homepage**. Disponível em: <http://www.netscout.com/>. Acesso em: jan. 2002.

[NET 2002] NETSAINT. **NetSaint Network Monitor**. Disponível em: <http://www.netsaint.org/>. Acesso em: mar. 2002.

[NTO 2002] NTOP. **ntop - network top Homepage**. Disponível em: <http://www.ntop.org>. Acesso em: mar. 2002.

[NOR 99] NORTHCUTT, Stephen. **Network Intrusion Detection**: an Analyst's Handbook. USA: New Riders Publishing, 1999.

[NOR 2001] NORTHCUTT, S.; NOVAK, J; MCLACHLAN, D. **Segurança e Prevenção em Redes**. São Paulo: Ed. Berkeley, 2001.

[NUN 97] NUNES, Cristina M. **Um Discriminador Inteligente de Eventos de Rede para o Ambiente Cinema**. 1997. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[OMG 99] OMG. **The Common Object Request Broker**: Architecture and Specification. Technical Report Revision 2.3.1. [S.l.], 1999.

[PAX 98] PAXSON, V. Bro: A System for Detecting Network Intruders in Real-Time. In: USENIX SECURITY SYMPOSIUM, 7., 1998, San Antonio, USA. **Proceedings...** [S.l.: s.n.], 1998.

[PER 98] PERKINS, David T. **RMON - Remote Monitoring of SNMP-Managed LANs**. USA: Prentice Hall, 1998.

[PHP 2001] PHP. **PHP Hypertext Preprocessor Homepage**. Disponível em: <http://www.php.net/>. Acesso em: 2001.

[POS 81] POSTEL, Jon. **Internet Control Message Protocol**, RFC 0792. Sept. 1981. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[PRO 2002] PROGRESS SOFTWARE. **Progress Software Corporation Homepage**. Disponível em: <http://www.progress.com/>. Acesso em: jan. 2002.

[ROS 91] ROSE, M. **Post Office Protocol - Version 3**, RFC 1225. 1991. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[ROS 96] ROSE, M. Industry Comment. **The Simple Times**, [S.l.], v. 4, n. 3, July 1996. Disponível em: <http://www.simple-times.org/pub/simple-times/issues/4-3.html>. Acesso em: maio 1999.

[SCH 99] SCHÖNWÄLDER, J.; QUITTEK, J. Secure Management by Delegation within the Internet Management Framework. In: INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 6., 1999, Boston, USA. **Proceedings...** [S.l.: s.n.], 1999.

[SCH 2000] SCHÖNWÄLDER, Jürgen. Building Distributed Management Applications with the IETF Script MIB. **IEEE Journal on Selected Areas in Communications**, New York, v.18, n. 5, p. 702–714, Mar. 2000.

[SNO 2002] SNORT. **Snort The Open Source Network Intrusion Detection System**. Disponível em: <http://www.snort.org/>. Acesso em: 2002.

[STA 96] STALLINGS, William. **SNMP, SNMPv2 and RMON**: Practical Network Management. USA: Addison Wesley, 1996.

[STA 99] STALLINGS, William. **SNMP, SNMPv2, SNMPv3, and RMON 1 and 2**. USA: Addison Wesley, 1999.

[STM 98] STAMATELOPOULOS, F.; ASTITHAS, P.; MAGLARIS, B. Web-based, Integrated, Enterprise Management. In: INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS, ICT, 1998, Chalkidiki, Greece. **Proceedings...** [S.l.: s.n.], 1998.

[STE 97] STEINKE, Steve. Network Management meets the Web. **Network**, USA, v.12, n. 12, p. 44–50.

[TEN 97] TENNENHOUSE, D. L.; SMITH, J. M.; SINCOSKIE, W. D.; WETHERALL, D. J.; MINDEN, G. J. A Survey of Active Network Research. **IEEE Communications Magazine**, New York, v. 35, n. 1, p. 80–86, Jan. 1997.

[TIV 2002] TIVOLI. **Tivoli Systems, Inc. Homepage**. Disponível em: <http://www.tivoli.com>. Acesso em: jan. 2002.

[THO 98] THOMPSON, Patrick. Web-based Enterprise Management Architecture. **IEEE Communications Magazine**, New York, v. 36, n. 3, p. 80–86, Mar. 1998.

[TUB 99] TU BRAUNSCHWEIG, NEC C&C RESEARCH LABORATORIES. **Jasmin - A Script MIB Implementation**. Disponível em: <http://www.ibr.cu.tu-bs.de/projects/jasmin>. Acesso em: 1999.

[WAL 95] WALDBUSSER, Steven. **Remote Network Monitoring Management Information Base**, RFC 1757. 1995. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[WAL 97] WALDBUSSER, Steven. **Remote Network Monitoring Management Information Base Version 2 using SMIv2**, RFC 2021. Jan. 1997. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[WAL 2002] WALDBUSSER, Steven. **Application Performance Measurement MIB**, Internet Draft. Feb. 2002. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.

[WIJ 99] WIJNEN, B.; PRESUHN, R.; MCCLOGHRIE, K. **View-based Access Control Model (VACM) for the Simple Network Management Protocol**, RFC 2575. Apr. 1999. Disponível em: <http://www.ietf.org/>. Acesso em: jan. 2002.