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

Um Sistema para Monitoração de Redes IP baseado em Políticas

por

MARCELO BORGES RIBEIRO

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

Prof[a]. Dr[a]. Maria Janilce Bosquiroli Almeida
Orientadora

Porto Alegre, novembro de 2003

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Ribeiro, Marcelo Borges

Um Sistema para Monitoração de Redes IP baseado em Políticas/ por Marcelo Borges Ribeiro. Porto Alegre: PPGC da UFRGS, 2003.

99p. : il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2003. Orientadora: Almeida, Maria Janilce Bosquiroli.

1. Gerência de redes. 2. Gerência de performance. 3. Qualidade de serviço. 4. Monitoração de QoS. 5. Monitoração de QoS distribuído. I. Almeida, Maria Janilce Bosquiroli. II Título.

Agradecimentos

Agradeço a todos que de alguma forma acabaram contribuindo para a minha formação, em especial a professora Janilce Bosquiroli de Almeida, minha orientadora nesse trabalho, aos colegas da graduação, da pós-graduação, meus amigos, minha família e a todos da Datacom-Telemática pela sensibilidade e flexibilidade ao permitir que eu fizesse o mestrado e trabalhasse. Agradeço também a Deus (pois nunca se sabe, quero terminar bem com todo mundo) e como de costume, tenho a minha parcela de contribuição nisso; portanto, agradeço a mim também.

Sumário

Lista de Abreviaturas

API	Application Program Interface
ARP	Address Resolution Protocol
CCB	Customer Care and Billing
COPS	Common Open Policy Service
COPS-PR	Common Open Policy Service Provisioning
CPU	Central Processing Unit
DiffServ	Differentiated Services
DNS	Domain Name Server
DoS	Denial of Service
DSCP	DiffServ Code Point
HTML	Hypertext Markup Language
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IntServ	Integrated Service
IOS	Internetwork Operating System
IP	Internet Protocol
IPX	Internetwork Packet Exchange
ISP	Internet Service Provider
LDAP	Lightweight Directory Access Protocol
LSAP	Link Service Access Point
MAC	Media Access Control
NFS	Network File System
NSAP	Network Service Access Point
ODBC	Open Database Connectivity
PBNM	Policy Based Network Management
PC	Personal Computer
PCI	Peripheral Component Interconnect
PDP	Policy Decision Point
PDU	Protocol Data Unit
PEP	Policy Enforcement Point
POP	Point Of Presence
QoS	Quality of Service
QPIM	QoS Policy Information Model
RMON	Remote Network Monitoring
RSVP	Resource Reservation Protocol
RTANS	Real Time Application Name Server
RTFM	Real Time Flow Management
SLA	Service-Level Agreement
SLM	Service-Level Management
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
TCP	Transport Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol

VoIP	Voice Over IP
WAP	Wireless Aplication Protocol
WML	Wireless Markup Language

Lista de Figuras

Lista de Tabelas

Resumo

O fornecimento de facilidades de QoS em redes de computadores tem por objetivo introduzir níveis das garantias que são ausentes, por exemplo, no paradigma de melhor-esforço das redes IP. Diferentes arquiteturas de QoS possuem padrões diferentes para os serviços fornecidos, que conduzem a um cenário em que a QoS prevista não possa ser sempre fornecida corretamente pela arquitetura utilizada. Neste contexto, uma monitoração da QoS é necessária para verificar a QoS observada e para compará-la com a QoS esperada/ contratada.

Em uma rede que utilize gerenciamento baseado em políticas, a QoS esperada é definido pelas políticas que regem o comportamento da rede. O *Internet Engineering Task Force* (IETF) tem padronizado vários elementos para um sistema de gerenciamento de redes baseado em políticas (PBNM), no qual políticas são definidas utilizando-se linguagem de alto nível, armazenadas em repositórios para políticas, aplicadas com o auxilio de *Policy Decision Points* (PDPs) e mantidas por *Enforcement Points* (PEPs). Pela definição do IETF, uma vez que uma política é aplicada com sucesso, o sistema de PBNM não mais checará o comportamento desta, e a QoS definida é assumida com a fornecida pela rede.

De fato, isso nem sempre é verdade. A arquitetura da qual provém a QoS pode apresentar-se instável ou mediante problemas, logo a QoS esperada não seria atingida. Para verificar a degradação na QoS em ambientes de gerenciamento reais, atualmente, o administrador da rede monitora a rede e determina a QoS fornecida. Tal QoS é, por sua vez, manualmente comparada com a QoS definida pelas políticas de rede. Finalmente, se diferenças são encontradas, o administrador procede com medidas que levem a arquitetura a um estado consistente.

Nos dias de hoje, as definições e aplicações de políticas e monitoração de QoS são tarefas executadas separadamente pelas soluções disponíveis. Além disso, como demonstrado anteriormente, a verificação da QoS fornecida contra a QoS definida pelas políticas da rede é deixada a cargo do administrador da rede. Nesse contexto, a automação da monitoração das políticas de QoS e a integração entre as tarefas citadas são necessárias do ponto de vista do administrador da rede.

Nesta dissertação, é proposta uma definição (e um sistema) para a monitoração de QoS em que as definições de políticas são dados de entrada utilizados para checar a QoS observada. No momento em que uma degradação na QoS é detectada, o sistema de monitoração notifica um gerente com suporte a SNMP utilizando mensagens do tipo *InformRequest*. A arquitetura do sistema é dividida internamente em monitores de QoS e controladores de monitores de QoS. Cada controlador de monitor de QoS controla vários monitores de QoS, os quais coletam dados da rede. Tais dados são comparados com as políticas traduzidas pelo controlador, e, caso sejam detectadas degradações, o controlador de monitor de QoS notifica o gerente. A comunicação entre controlador de monitores de QoS e monitores de QoS também é baseada em SNMP. O principal objetivo do trabalho é fornecer uma solução que integre monitoração de QoS e PBNM em um único ambiente de gerenciamento.

Palavras-chave: gerência de redes, gerência de performance, qualidade de serviço, monitoração de QoS, monitoração de QoS distribuído.

TITLE: “A SYSTEM TO MONITOR POLICY-BASED IP NETWORKS”

Abstract

The provisioning of QoS facilities in computer networks is intended to introduce levels of guaranties that are absent, for example, in the IP best-effort approach. Different QoS architectures have different accuracy for the provided services, which leads to a scenario where the expected QoS may not be always properly provided by the underlying architecture. In this context, QoS monitoring is required to verify the observed QoS and compare it with the expected/contracted QoS.

In a policy-based network, the expected QoS is defined by policies that orchestrate the network behavior. The Internet Engineering Task Force (IETF) has standardized several elements of a Policy-Based Network Management (PBNM) system, in which policies are defined using high-level languages, stored in policy repositories, deployed with the help of Policy Decision Points (PDPs) and enforced by Policy Enforcement Points (PEPs). In the IETF approach, once a policy is successfully deployed, the PBNM system no longer checks the policy behavior, and the QoS defined in the policy is assumed to be provided by the network.

In fact, that is not always true. The underlying QoS provisioning architecture can be unstable or face problems, and the expected QoS may not be achieved. In order to verify QoS degradation in real management environments, today, the network administrator monitors the network and determines the provided QoS. Such QoS is then manually compared with the QoS defined by the network policies. Finally, if differences are found, the administrator proceeds with actions to lead the underlying architecture to a consistent state.

To date, policy definition and deployment, and QoS monitoring are tasks executed separately by the available solutions. Also, as shown before, the verification of the provided QoS against the QoS defined by the network policies is left to the network administrator. In this context, automation of QoS-policy monitoring and integration among the cited tasks are required from the network management point-ofview.

In this work we propose an approach (and a system) for QoS monitoring, where policy definitions are input data used to check the observed QoS. When a QoS degradation is detected, the monitoring system notifies an SNMP-enabled manager using InformRequest messages. The system’s architecture is internally divided in QoS monitors and QoS monitoring controllers. Each QoS monitoring controller controls several QoS monitors that collect data from the network. Such data is compared with the policies translated by the controller, and if degradations are detected the monitoring controller notifies the SNMP-enabled manager. The communication between controllers and monitors is also SNMP-based. The main goal here is to provide a solution that integrates QoS monitoring and PBNM in a unique management environment.

Keywords: network management, performance management, quality of service, QoS monitoring, QoS distribution monitoring.

1 Introdução

Hoje as redes de computadores não são só utilizadas para o compartilhamento e transferência de arquivos, mas também para a prestação de serviços que antes eram suportados por outros meios de comunicação. Esses serviços (voz, vídeo, etc) exigem novos mecanismos para assegurar o seu funcionamento pela rede de uma forma aceitável. Em alguns casos, a chegada dos dados ao seu destino, mesmo que com o seu conteúdo integro, não é mais uma condição satisfatória e suficiente para a validação dos mesmos. No modelo tradicional de redes de computadores, do ponto de vista de uma aplicação, é esperado que a rede forneça um canal livre de erros entre dois computadores e que, na pior das hipóteses, pacotes que cheguem fora de ordem sejam reordenados automaticamente. Agora, com serviços de vídeo e voz integrados às redes de computadores, o fator *tempo* soma-se aos pré-requisitos para se validar um pacote (integridade, ordenação dos quadros, etc) no que se refere à correção dos dados que trafegam na rede. Em outras palavras: os pacotes (dados) que chegarem fora de um limite pré-estabelecido de tempo, serão invalidados pela rede, e essa deve, por esse motivo, possuir algum mecanismo que garanta ou facilite o tráfego dos dados dentro dos limites de validade dos mesmos. Dentro dessa nova realidade, é necessário fazer uso do que se chama qualidade de serviço ou QoS (*Quality of Service*) [TEI 98] no fornecimento de serviços que necessitem alguma garantia quanto ao desempenho ou à qualidade no funcionamento de certas aplicações como serviços de vídeo, entre outras.

A maioria das redes de computadores no mundo utiliza o protocolo IP (*Internet Protocol*) como protocolo de rede. É natural que se assuma que o protocolo IP será utilizado também para suporte ao tráfego de serviços multimídia. O principal problema, no que diz respeito a aplicações multimídia, é que o protocolo IP não fornece nenhum esquema que garanta que pacotes chegarão dento de um intervalo de tempo específico; na verdade, ele não garante nem a entrega dos pacotes. A única garantia que se tem é a da rede utilizar toda a largura de banda disponível no momento para qualquer pacote. É o que se chama de paradigma do melhor esforço (*best effort*) das redes IP. Por causa dessa característica, é necessário que haja alguma modificação no sentido de priorizar tráfego em roteadores, descarte seletivo de pacotes, etc, para diminuir a aleatoriedade gerada pelo melhor esforço de uma rede IP. Entretanto, não basta o usuário ou aplicação solicitarem os serviços com QoS, tampouco basta o provedor do serviço e os demais computadores envolvidos aprovarem as exigências do usuário para que realmente o serviço solicitado funcione conforme suas necessidades; é necessário que se tome alguma medida que vise verificar a QoS fornecida. É por causa desse suporte, mesmo que deficiente, de QoS, que uma monitoração constante da QoS fornecida pela rede se faz necessária [JIA 2000a], uma vez que a QoS contratada pode não ser a fornecida por motivos de falhas nos equipamentos, por má configuração, ou até mesmo, por um problema inerente de redes IP.

Em uma rede com gerenciamento baseado em políticas [COE 2002], a QoS esperada é definida através de políticas, as quais coordenam o comportamento da rede. O IETF (*Internet Engineering Task Force*) padronizou vários elementos para gerenciamento de redes baseado em políticas (PBNM - *Policy-Based Network Management*) [MOO 2002]. Nessas redes, as políticas são definidas utilizando-se linguagens de alto nível e são armazenadas em repositórios. Após, elas são utilizadas com o auxílio de um PDP (*Policy Decision Points*) e aplicadas nos PEPs (*Policy Enforcement Point*) [WES 2001]. No modelo do IETF, uma vez que uma política é aplicada com sucesso, o PBNM não mais verifica o comportamento da política e a QoS

definida é assumida como sendo a que é fornecida pela rede. Na prática, o que acontece é que sistemas na rede podem falhar e a QoS fornecida não condizer com a especificada. A fim de verificar se a QoS fornecida está dentro das especificações, a rede deve ser constantemente monitorada e os resultados dessa monitoração serão normalmente comparados manualmente com os valores solicitados pelas políticas. Caso a rede monitorada apresente um desempenho inferior ao requerido, o administrador da rede deveria ser informado para agir de acordo a levar a rede a um estado estável novamente. Entretanto, os sistemas atuais de monitoração de redes não comparam o comportamento verificado com o comportamento expresso nas políticas, obrigando o administrador da rede a realizar tal comparação de forma manual.

A definição de uma política de monitoração, a sua aplicação por parte do administrador da rede e a monitoração são tarefas executadas separadamente por uma pessoa [GRA 2001]. Conforme explicado anteriormente, a verificação da QoS fornecida com a esperada é uma tarefa executada manualmente pelo administrador da rede. Nesse contexto, há a necessidade natural de uma integração, do ponto de vista do administrador da rede, entre essas duas operações de modo que a simples definição de uma política e a sua tradução para regras de monitoração acarretassem a sua pronta monitoração sem necessidade de intervenção humana [RIB 2002]. Nessa dissertação, é proposto um sistema para monitorar QoS, o qual utiliza políticas de gerenciamento como dados de entrada para comparar com a QoS observada. Uma vez que uma degradação de QoS é detectada, o sistema notifica o administrador da rede através de mensagens SNMP. A arquitetura do sistema está dividida internamente entre Monitores de QoS e Controladores de Monitores. Cada Controlador de Monitor controla um ou mais Monitores de QoS, os quais monitoram a rede em busca de degradações de QoS. A comunicação entre os Controladores de Monitor e os Monitores de QoS também é feita via SNMP. O principal objetivo do sistema é integrar o gerenciamento baseado em políticas com a monitoração de QoS em um único ambiente.

O restante desta dissertação está dividido como segue. O capítulo 2 faz uma análise de soluções existentes para monitoração de QoS. O capítulo 3 descreve a arquitetura da solução proposta. O capítulo 4 aborda questões que dizem respeito à operação do sistema, modelagem e uma implementação possível para a solução. Finalmente, o capítulo 5 encerra esta dissertação apresentando as conclusões e trabalhos futuros.

2 Soluções para Monitoração de QoS

Nas seções que se seguem serão apresentadas ferramentas que são utilizadas na monitoração de QoS. É importante perceber que as técnicas de monitoração analisadas nesse capítulo (fim-a-fim e por pontos de degradação) são utilizadas em todas as ferramentas disponíveis, não somente para a monitoração, mas também na manutenção da QoS.

2.1 Monitoração de QoS fim-a-fim

Para o funcionamento adequado de aplicações críticas com requisitos estritos de QoS, é necessário que os pontos inicial e final da transmissão sejam monitorados. Nesse caso, os elementos intermediários do caminho entre origem e destino dos fluxos não são investigados. Os valores para os parâmetros de QoS observados são ditos fim-a-fim e a degradação de QoS fim-a-fim é a soma da degradação de cada segmento da rede entre os dois pontos finais de monitoração.

Se por acaso apenas um segmento da rede estiver introduzindo alguma degradação, essa situação será percebida pela monitoração fim-a-fim, mas não será possível especificar precisamente em qual ponto da rede a degradação ocorre.

A monitoração nos sistemas finais permite às aplicações perceberem as características da comunicação encarando a rede de computadores como uma caixa-preta (fig. 2.1), já que não existe monitoração interna à rede. Como cada aplicação pode implementar seu próprio processo de monitoração, não existe a necessidade de se ter nenhum recurso extra (por exemplo, protocolo especial na pilha de protocolos dos sistemas finais) para se proceder com essa análise. Cada aplicação implementa suas observações internamente e nenhuma modificação na rede é necessária.

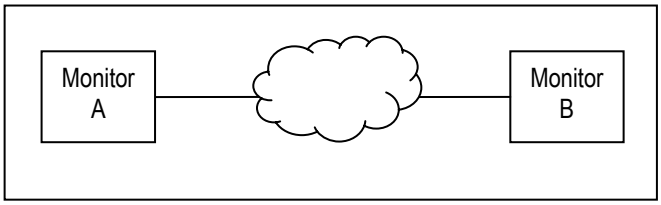

FIGURA 2.1 - Monitoração de QoS fim-a-fim.

Por outro lado, a monitoração nos sistemas finais é frágil porque não permite, do ponto de vista do gerenciamento, identificar quais são os pontos de rede críticos ou problemáticos que estão introduzindo as degradações percebidas nos sistemas finais. Sem essa informação, o gerente da rede é forçado a verificar cada possível ponto falho separadamente, o que é uma tarefa que consome muito tempo, tempo este provavelmente inexistente.

2.2 Monitoração de QoS por pontos de degradação

A monitoração por pontos de degradação tem por objetivo não apenas detectar a existência de degradação de QoS, mas também determinar quais os pontos da rede que geram a degradação observada, como mostra a figura 2.2. Para tal, uma coleta de informações sobre a QoS fornecida deve ser realizada. Os dados coletados são analisados por processos específicos que verificam se a QoS esperada está sendo atendida.

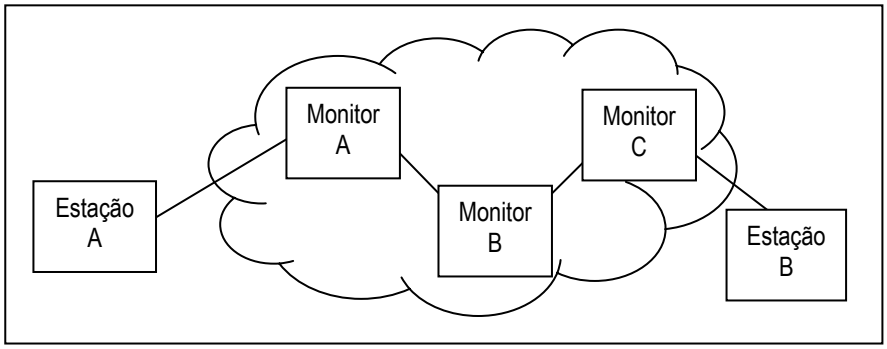

FIGURA 2.2 - Monitoração de QoS por pontos de degradação.

Coletar as informações de QoS da rede envolve a introdução de estruturas de monitoração anteriormente inexistentes. Agentes especiais são responsáveis por analisar o comportamento das transmissões e repassar estas informações para análise. Assim, agentes monitores encontram-se espalhados na rede de computadores, normalmente nos segmentos mais críticos que exigem monitoração mais freqüente.

2.3 NeTraMet

Essa seção apresenta a ferramenta de monitoração conhecida como NeTraMet [BRO99].

O NeTraMet foi a primeira implementação da arquitetura de monitoração do grupo de trabalho RTFM (*Realtime Traffic Flow Measurement*) do IETF [MIL91]. A arquitetura do NeTraMet possui três componentes principais:

- Medidores (*Meters*): são *hosts* agregados a uma subrede e que são responsáveis pela monitoração naquele segmento.

- Leitores dos Medidores (*Meters´s Readers*): são as "entidades" que têm a incumbência de recuperar as informações coletadas pelos medidores.

- Gerentes (*Managers*): têm a tarefa de instruir os Medidores sobre quais fluxos realizar a monitoração, também quais Leitores de Medidores devem coletar informações e também quais informações devem ser coletadas num intervalo também a ser especificado.

Um Leitor de Medidor pode coletar dados de vários Medidores, e cada Medidor pode ter suas informações coletadas por diversos Leitores de Medidores. O protocolo

escolhido no RTFM para a comunicação entre os módulos foi o SNMP. O tráfego (fluxo a ser monitorado) é definido pelo usuário a partir de uma série de regras estabelecidas. A figura 2.3 ilustra essas definições.

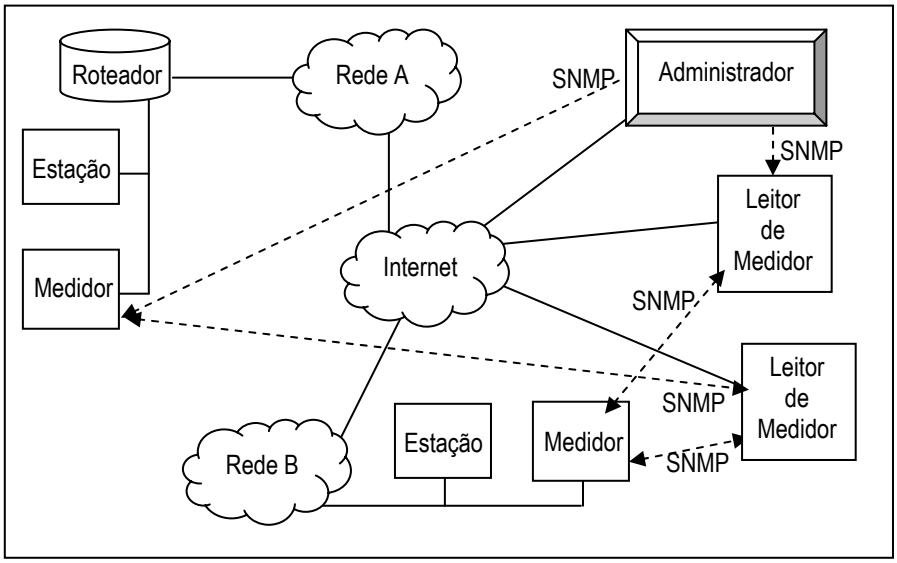

FIGURA 2.3 - Funcionamento do NeTraMet

Na figura anterior, as linhas tracejadas representam ligações lógicas entre as entidades, e as linhas contínuas representam ligações físicas.

A versão 4.3 do NeTraMet está disponível para as seguintes plataformas: Solaris, Sun OS, Irix e Linux. A versão para Linux depende do uso em conjunto com a biblioteca de captura de pacotes Libpcap[LIB2002]. Em um PC 386 SX 25 MHz, com o NeTraMet, é possível analisar 1250 pacotes por segundo e capaz de suportar uma rajada com pico de 2250 pacotes por segundo por vários segundos. Em um 486 SX 40 MHz, o NeTraMet consegue monitorar 3000 pacotes por segundo de pico, num Pentium 60 MHz, com barramento PCI, o NeTraMet pode suportar picos de até 6000 pacotes por segundo.

Juntamente com o NeTraMet, encontra-se disponível o NeMAC (*NeTraMet Manager/Collector*).

- Fluxo

O NeTraMet trabalha com o conceito de fluxo de tráfego, que vem a ser um conjunto de *bytes* transmitidos pela rede de um *host* a outro. Esses *hosts* são identificados como emissor e receptor dos pacotes do fluxo. Para reconhecer um fluxo, o NeTraMet lê, de um pacote, os atributos de endereços do emissor e do receptor. Esses atributos estão definidos, de acordo com o modelo OSI, em três tipos:

TABELA 2.1 - Tipos de atributos suportados

Atributo	Camada de Atuação
Adjacente	nível de enlace
Peer	nível de rede
Transporte	nível de transporte

Deve-se configurar os medidores fornecendo informações sobre cada um dos atributos das camadas descritas na tabela 2.1. Esses atributos são os endereços de cada *host* segundo padrões de endereçamentos diferentes. Por exemplo, um computador para uma rede pode possuir um endereço IP igual a 192.168.0.10 e um endereço MAC sendo: 00:00:21:CE:5D:C0: essas seriam duas formas diferentes, porém necessárias, de identificarmos um computador em uma rede.

Conforme mostra a tabela 2.1, caso se esteja em um ambiente 802.3 (ethernet) e se queira monitorar apenas endereços dessa rede local, o NeTraMet usará, como campo de endereço, o endereço adjacente. Novamente, segundo a tabela 2.1, o endereço adjacente faz parte do nível de enlace. Um endereço *peer* pode ser um endereço IP, um endereço IPX (número de rede Novell), um endereço EtherTalk, CLNS NSAP ou um endereço DECnet Phase IV. Esses são os cinco endereços que podem ser utilizados como endereços de *peer*. Caso se esteja interessado em monitorar o nível de transporte, pacotes TCP ou UDP, também será possível através do atributo de transporte.

Em se tratando de IP, os endereços de origem, destino, porta de origem e porta de destino são utilizados. Tipos similares de detalhamento são utilizados nos demais protocolos.

O Medidor é responsável por colocar *Time Stamp* em todos os pacotes para que se possa saber o momento em que um pacote chega e sai em cada nó da rede. Essa informação, juntamente com as demais (endereços de rede e portas), é armazenada na memória como estruturas de dados.

Há a possibilidade de se armazenar todos os pacotes de todos os fluxos que passem indiscriminadamente, mas isso acabaria por exaurir a memória disponível no sistema. Para resolver esse problema, um conjunto de regras foi criado para que o NeTraMet possa inferir quais pacotes devem ser armazenados na memória. Apenas os pacotes que interessam são considerados.

Cada regra testa um atributo do fluxo, usando uma máscara para especificar quais bits são de interesse. Dessa forma, pode-se construir uma hierarquia de regras que classificam os pacotes que saem e chegam em fluxos. Cada pacote pode ser, então, contabilizado no seu respectivo fluxo, conforme será visto mais adiante.

Os atributos de um fluxo são convenientemente arranjados em cinco grupos: endereços Adjacentes, endereço do "parceiro" (*Peer*), endereço de Transporte, endereço do Assinante (*subscriber*) e endereço Geral (*general*). Como o NeTraMet não pode determinar qualquer tipo de informação sobre o assinante apenas analisando os pacotes que trafegam, essa funcionalidade não é implementada, podendo ficar a cargo de um Medidor que tenha acesso a um servidor com informações de usuário, uma vez que precisariam ser combinadas informações como o endereço do emissor do pacote (isso é possível) com o usuário relacionado a esse endereço (não é possível somente com o NeTraMet).

Atributos de endereços Adjacentes mantêm-se os mesmos para quaisquer tipos de endereços utilizados no fluxo, uma vez que só fará sentido analisar atributos adjacentes (nível de enlace) dentro de uma mesma rede local, em que é necessário que se utilize o mesmo protocolo tanto no receptor quanto no emissor.

Os endereços *Peer* e de Transporte, diferentemente do atributo Adjacente, variam toda vez que os protocolos utilizados diferem. Isso faz sentido uma vez que um computador utilizando o protocolo IPX pode ser uma extremidade de um fluxo, e um computador que utilize o protocolo IP pode ser a outra. Como o atributo adjacente diz respeito ao endereço MAC (nível de enlace como mostra a tabela 2.1), não há motivos

para se efetuar uma monitoração, em nível de enlace, em redes que utilizem protocolos distintos.

- Atributo Adjacente

A implementação do NeTraMet apenas suporta interfaces de rede passivas, isto é, o fluxo de destino (*DestAdjacentType*) e o de origem (*SourceAdjacentType*) têm, obrigatoriamente, o mesmo tipo de endereço adjacente. Vale ressaltar que esses atributos contêm o próprio endereço a que dizem respeito.

Os endereços *SourceAdjacentType* e *DestAdjacentType* nada mais são do que os endereços MAC do adaptador de rede Ethernet. Como se sabe, um endereço MAC é formado por seis bytes hexadecimais separados por hifens. Como, por exemplo: 00-00-C0-00-13-A5, podem ser utilizados pontos ao invés de traços.

Os campos, *SourceAdjacentType* e *DestAdjacentType* serão utilizados sempre que se queiram regras para testar endereços adjacentes.

- Atributo *Peer* (do "parceiro")

O atributo *Peer* depende do protocolo de rede de cada um dos *hosts* de um fluxo. Cada protocolo possui uma forma única de identificar um computador na rede. É baseado nessa informação única, que o NeTraMet caracteriza esse atributo. Todos os protocolos do nível de rede que são suportados serão analisados nessa seção. Algumas variáveis de atributos foram omitidas para simplificar a análise da ferramenta. Para uma lista mais detalhada deve-se consultar o manual de referência do NeTraMet [BRO 99].

- Atributos IP

Os campos *SourcePeerType* e *DestPeerType* são iguais a 1. E a versão do protocolo IP é a 4. A versão 6 foi introduzida na versão 4.3 do NeTraMet e o seu número *PeerType* é 2.

SourcePeerAddress e *DestPeerAddress* são os endereços IP das duas extremidades do fluxo. São formados por quatro bytes decimais separados por pontos. Pode-se, ainda, especificar a máscara utilizada nos campos *SourcePeerMask* e *DestPeerMask*.

- Atributos DECnet

Os campos *SourcePeerType* e *DestPeerType* são iguais a 13. *SourcePeerAddress* e *DestPeerAddress* são os endereços DECnet *phase IV* que são formados por quatro bytes decimais separados por pontos.

- Atributos IPX da Novell

Os campos *SourcePeerType* e *DestPeerType* são iguais a 11. *SourcePeerAddress* e *DestPeerAddress* são determinados da seguinte forma: como o endereço IPX da Novell é formado pela combinação do endereço MAC Ethernet com o endereço de rede propriamente dito, essa combinação não é aceita pela "compilação *default*" do NeTraMet. Deve-se, portanto, utilizar o endereço de rede apenas como identificador do computador. No caso de se precisar utilizar todo o endereço Novell, isso é possível apenas através da recompilação do NeTraMet com a opção *FULL_IPX*.

- Atributos EtherTalk

Os campos *SourcePeerType* e *DestPeerType* são iguais a 12. *SourcePeerAddress* e *DestPeerAddress* é o endereço *AppleTalk* de cada *host.* É formado, a exemplo dos demais, por quatro bytes decimais separados por pontos.

- Atributos CLNS

Os campos *SourcePeerType* e *DestPeerType* são iguais a 3. Esse tipo de endereço ocupa um espaço considerável na memória e pode ser desabilitado em tempo de compilação, sendo habilitado por *default*.
SourcePeerAddress e *DestPeerAddress* são os endereços NSAP dos dois *hosts* , formados por até vinte bytes em hexadecimal separados por hifens.

- Atributos "Outro"

Os campos *SourcePeerType* e *DestPeerType* são iguais a 6. Quaisquer outros pacotes que não se enquadrem dentre os descritos anteriormente são tratados como "outro" (*other*). É possível especificar regras para o reconhecimento desses pacotes de forma que não sejam confundidos com pacotes *dummy*, que são gerados pelo NeTraMet quando a rede encontra-se com baixa utilização para medir a ocupação do processador. *SourcePeerAddress* contém o valor do campo Ethernet II do pacote, codificado como dois bytes hexadecimais separados por hífen. *DestPeerAddress* contém o campo LSAP do cabeçalho do pacote (para pacotes Ethernet): para demais pacotes essa variável recebe o valor zero.

- Atributos Gerais

Os atributos gerais são aqueles atributos que têm relação apenas com os fluxos em si. Esses atributos estão disponíveis para quaisquer tipos de fluxos, independentemente das características dos protocolos envolvidos. Por exemplo, a interface de rede do emissor e do destinatário, o tipo de fluxo que trafega na rede (classe do serviço, por exemplo), etc.

- Utilização

De forma resumida, utiliza-se o NeTraMet para fazer a monitoração dos fluxos. Para a apresentação dos dados faz-se uso do NeMAC, que vem a ser uma ferramenta que faz parte do NeTraMet, e que tem a única função de gerar relatórios a partir dos dados do NeTraMet. As informações ficam disponíveis para o administrador da rede em arquivos texto. A figura 2.4 é um exemplo de arquivo texto gerado pelo NeMAC após receber dados de uma monitoração efetuada pelo NeTraMet.

A primeira informação armazenada no arquivo inicia com um duplo sinal de "#" e marca o início (*time stamp*) da amostragem pelo NeTraMet, assim como os parâmetros utilizados pelo NeMAC. A seguir, tem-se uma marca de "#" com a palavra "*format*", que apresenta o formato com que as informações estão dispostas no restante do arquivo. A tabela 2.2 contém uma lista de abreviaturas utilizadas no arquivo gerados pelo NeMAC.

```
##NeTraMet v3.2:  -c300 -r rules.lan  -e rules.default
   test_meter -i eth0  4000 flows  starting at 12:31:27 Wed  1 Feb 1995
#Format: flowruleset flowindex firsttime  sourcepeertype sourcepeeraddress
destpeeraddress  topdus frompdus  tooctets fromoctets
#Time: 12:31:27 Wed  1 Feb 1995 130.216.14.251 Flows from 1 to 3642
#Stats: aps=478 apb=11 mps=636 mpb=48 lsp=0 avi=97.3 mni = 93.4 fiu=44 frc=0
gci=5 rpp=1.9 tpp=0.0 cpt=1.0 tts=1024 tsu=38
1 2 13  5 31.32.0.0 33.34.0.0  1138 0  121824 0
1 3 13  2 11.12.0.0 13.14.0.0  4215 0  689711 0
1 4 13  7 41.42.0.0 43.34.0.0  1432 0  411712 0
1 5 13  6 21.22.0.0 23.24.0.0  8243 0  4338744 0
3 6 3560  2 130.216.14.0 130.216.3.0  0 10  0 1053
3 7 3560  2 130.216.14.0 130.216.76.0  59 65  4286 3796
3 8 3560  7 0.0.255.0 1.144.200.0  0 4  0 222
3 9 3560  2 130.216.14.0 130.216.14.0  118 1  32060 60
3 10 3560  6 130.216.0.28 130.216.0.192  782 1  344620 66
3 11 3560  7 0.0.255.0 0.128.113.0  0 1  0 73
3 12 3560  5 59.3.13.0 4.1.152.0  1 1  60 60
3 13 3560  7 0.128.94.0 0.129.27.0  2 2  120 158
3 14 3560  5 59.3.40.0 4.1.153.0  2 2  120 120
3 15 3560  5 0.0.0.0 4.1.153.0  0 1  0 60
.  .  .  .  .  .  .  .  .
3 43 3560  7 0.128.42.0 0.128.43.0  0 1  0 60
3 44 3560  7 0.128.42.0 0.128.41.0  0 1  0 60
3 45 3560  7 0.128.42.0 0.129.2.0  0 1  0 60
3 46 3560  5 4.1.152.0 59.2.208.0  2 2  120 120
3 47 3560  5 59.3.46.0 4.1.150.0  2 2  120 120
3 48 3560  5 4.1.152.0 59.2.198.0  2 2  120 120
3 49 3560  5 0.0.0.0 59.2.120.0  0 1  0 60
3 50 3664  5 4.1.152.0 59.2.214.0  0 1  0 60
3 51 3664  5 0.0.0.0 4.2.142.0  0 1  0 60
3 52 3664  5 4.1.153.0 59.3.45.0  4 4  240 240
#EndData
```

FIGURA 2.4 - Análise de fluxo pelo NeTraMet

TABELA 2.2 - Siglas do NeMAC

Sigla	Significado
aps	average packets/second
apb	average packet backlog
cpt	compares per count
lsp	number of packets lost
mps	maximum packets/second
mpb	maximum packet backlog
avi	average processor idle %
mni	minimum processor %
fiu	flows in use
frc	flows recovered
Gci	garbage collection interval (seconds)
Rpp	rules matched per packet
Tpp	counts per packet
Tsu	count tables in use
Tts	total count tables allocated

De posse de todas essas informações, o administrador da rede deve ser capaz de inferir as informações que desejar desse arquivo texto.

2.4 Ntop

A exemplo do NeTraMet, o Ntop[DER2001] é um *software* livre e de domínio público. Ele foi inspirado no comando TOP do Unix (fig. 2.5) que demonstra a

utilização de cada processo na CPU. Maiores referências ao Ntop podem ser encontradas no seu manual de utilização [DER2001].

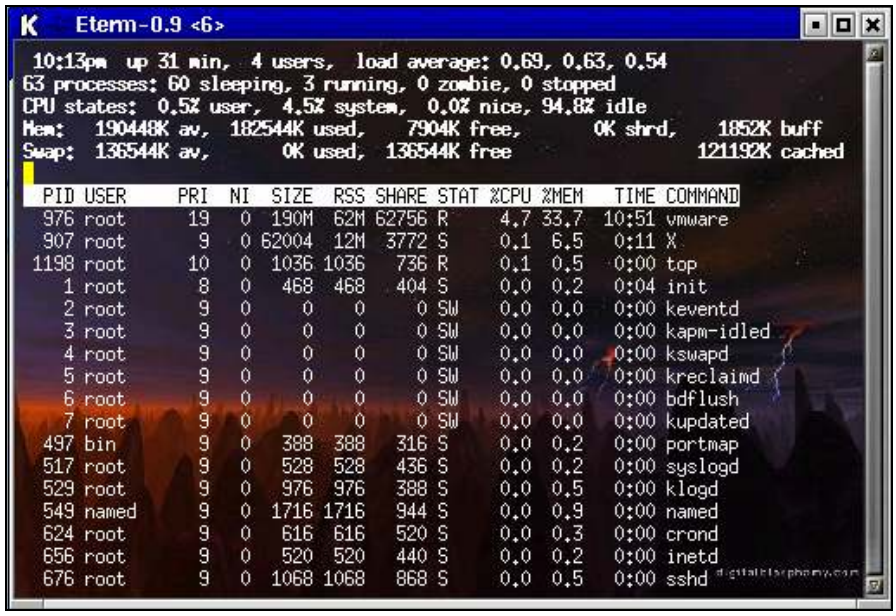

FIGURA 2.5 - Comando TOP mostra a utilização da CPU

O Ntop é baseado na biblioteca de captura de pacotes *libpcap* [LIB2001] e está disponível para todas as arquiteturas baseadas em UNIX e há versões também para Windows (graças a um porte da *libpcap* para a API Win32 conhecido como *winpcap* [WIN2002]).

A principal diferença entre o Ntop e as demais arquiteturas de monitoração de tráfego é a preocupação com a apresentação dos resultados obtidos. Certamente, do ponto de vista funcional, o Ntop não apresenta nada que o destaque dos demais. Conforme Deri et al. [DER2001], se não fosse a dificuldade de integração com as demais ferramentas de monitoração de tráfego, o Ntop seria apenas um *front-end* para alguma outra aplicação que apenas tivesse uma função de *sniffer*.

O Ntop é centrado nas seguintes características:

- medição de tráfego;
- monitoração e caracterização de tráfego;
- planejamento e otimização de redes;
- segurança de redes.

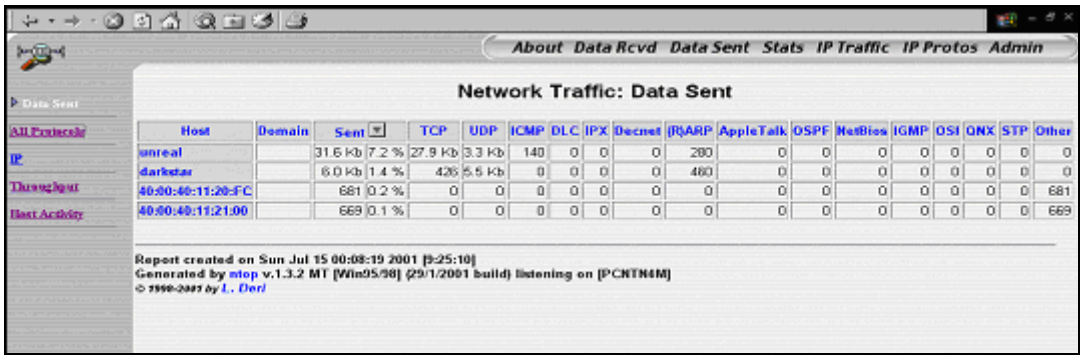

FIGURA 2.6 - Preocupação com a apresentação das informações

Como pode ser visto na figura 2.6 (protocolos utilizados), há uma preocupação considerável em formatar as informações que serão passadas ao administrador da rede. Como também pode ser notado, os dados são acessados via *browser* que se comunica com um agente responsável por "capturar" a informação desejada, conforme mostra a figura 2.7.

Há a presença de um agente efetuando monitoração no enlace (fig. 2.7). Porém por ser uma tarefa onerosa a de ficar interpretando arquivos de *log*, o agente possui um servidor WEB embutido que visa facilitar o trabalho de obtenção de dados do usuário de uma forma mais amigável e também segura, uma vez que a comunicação entre o agente e o *browser* é feita via SSL. Caso o usuário deseje, há a possibilidade de acessar as informações via linha de comando.

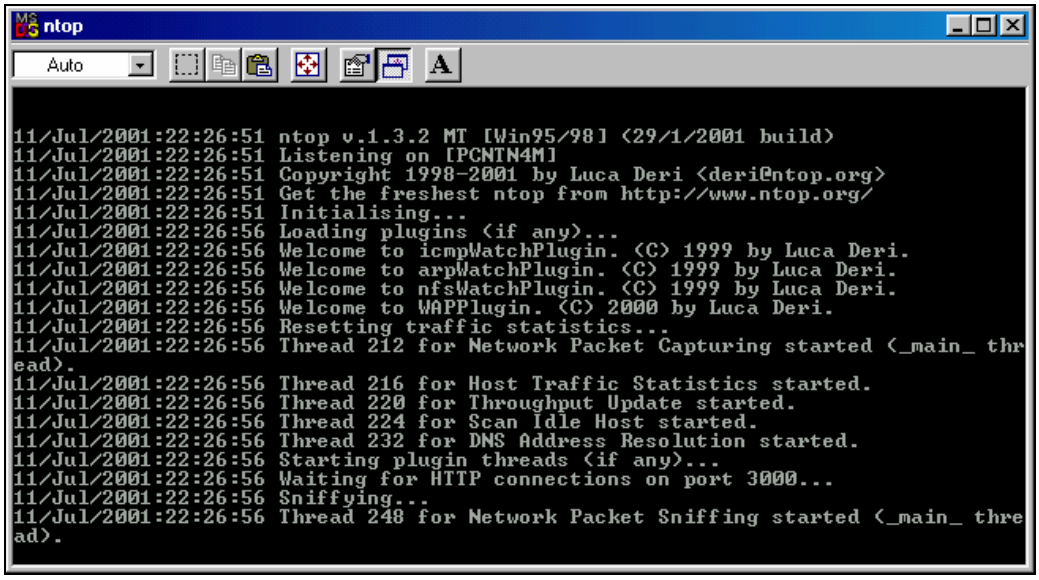

FIGURA 2.7 - Processo responsável por obter as informações

- Arquitetura

Conforme escrito anteriormente, a comunicação entre usuário (*WEB browser*) e aplicação (agente que efetua a monitoração) se dá via SSL, o que garante uma interface segura.

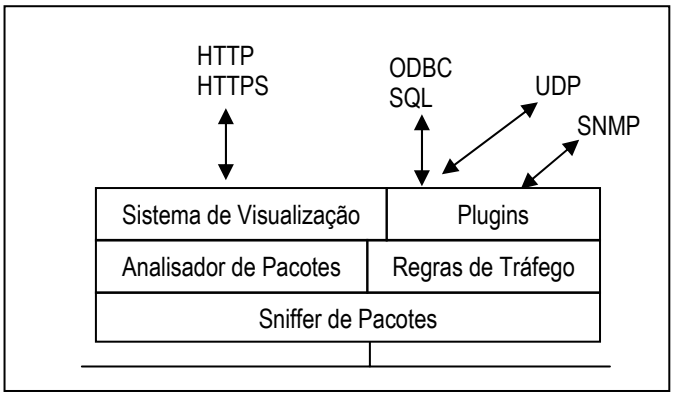

FIGURA 2.8 - Arquitetura do Ntop

Um esquema da arquitetura do Ntop pode ser visto na figura 2.8 que demostra os protocolos utilizados por cada parte do sistema, como, por exemplo, o analisador de pacotes recebe as regras de tráfego pelo administrador da rede via SNMP, que em conjunto com o *Sniffer* de pacotes fornecem as informações que serão passadas para o usuário, através do sistema de visualização por HTTP ou HTTPS ou enviadas a *plugins* que podem efetuar algum processamento adicional sobre os dados, como, por exemplo, armazená-los em uma base de dados via ODBC SQL.

Além de ser utilizado para monitoração de tráfego, o Ntop, com o passar do tempo, teve para si acrescidas as funcionalidades de medição de tráfego e detecção de problemas na rede, tais como má configuração de *gateway*, protocolos desnecessários, etc. O restante dessa seção abordará somente a monitoração de tráfego do Ntop. Para maiores informações, pode ser consultado o artigo em que essa seção foi baseada [DER2001].

Para que seja possível integração com ferramentas comerciais, o protocolo SNMP foi adotado para comunicação direta do agente com alguma plataforma de gerenciamento. O *sniffer* de pacotes, conforme mencionado anteriormente, foi implementado através da *libpcap* [LIB2001]. O analisador de pacotes é responsável por "aprender" sobre a rede, isto é, é capaz de identificar fluxos TCP, descobrir roteadores e servidores DNS. O Sistema de visualização faz a integração da aplicação com o *browser* por meio de um servidor web embutido na aplicação para que o usuário não necessite instalar e configurar um. Além de HTML, é possível acessar as informações através de WML pela tecnologia WAP. Caso haja necessidade, o usuário pode utilizar *pluggins* (executam em *threads* separadas) para obter informações sobre outros protocolos, como ICMP, ARP e NFS. Ou para acessar o Ntop via SNMP ou ODBC.

- Utilização

Após iniciar o processo de monitoração de pacotes – que se dá pela execução do agente – o usuário tem à disposição uma grande quantidade de informações a respeito do fluxo da rede local que vão desde a largura de banda utilizada até o momento por cada *host,* como mostra a figura 2.9, em que dois computadores têm monitorados os usos de banda da rede local, até endereços de adaptadores de rede.

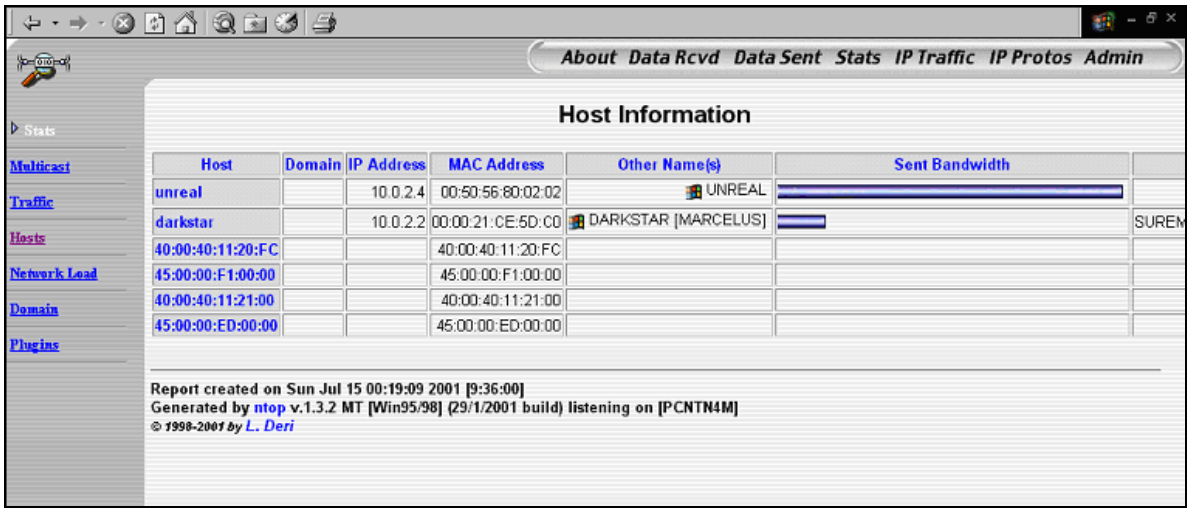

FIGURA 2.9 - Informações sobre o host

Caso não se deseje efetuar uma monitoração em um ambiente de rede local, o Ntop só é capaz de identificar as conexões TCP abertas no computador local, como se pode ver figura 2.10, em que o Ntop não foi capaz de identificar o computador de destino, mas sim o seu endereço IP, que está em representação hexadecimal. Caberia ao usuário determinar as informações restantes através de outro Ntop (seria interessante que se utilizasse *plugins* para a troca de informações entre instancias remotas do Ntop).

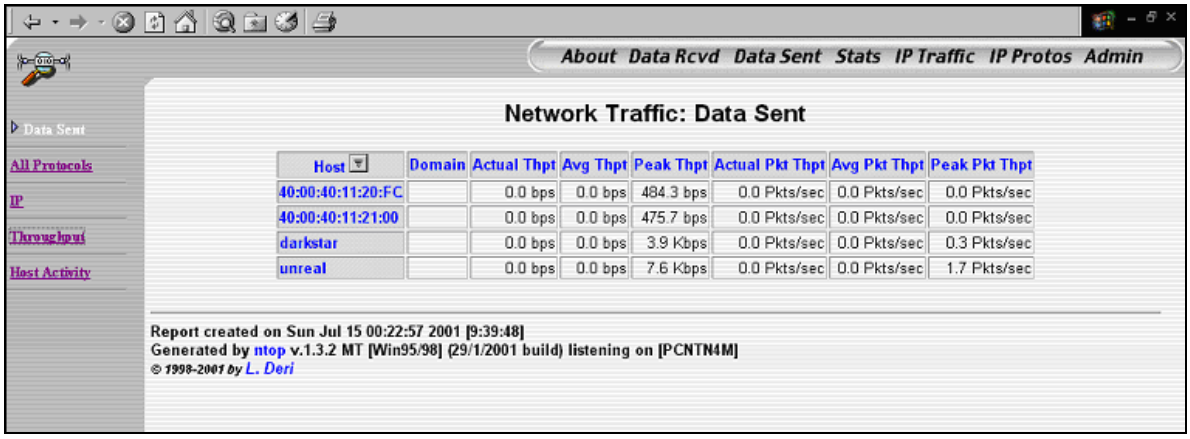

FIGURA 2.10 - Não há informações sobre fluxo fora da subrede

Informações como vazão da rede, mesmo que seja apenas no segmento de escopo do Ntop sempre é útil.

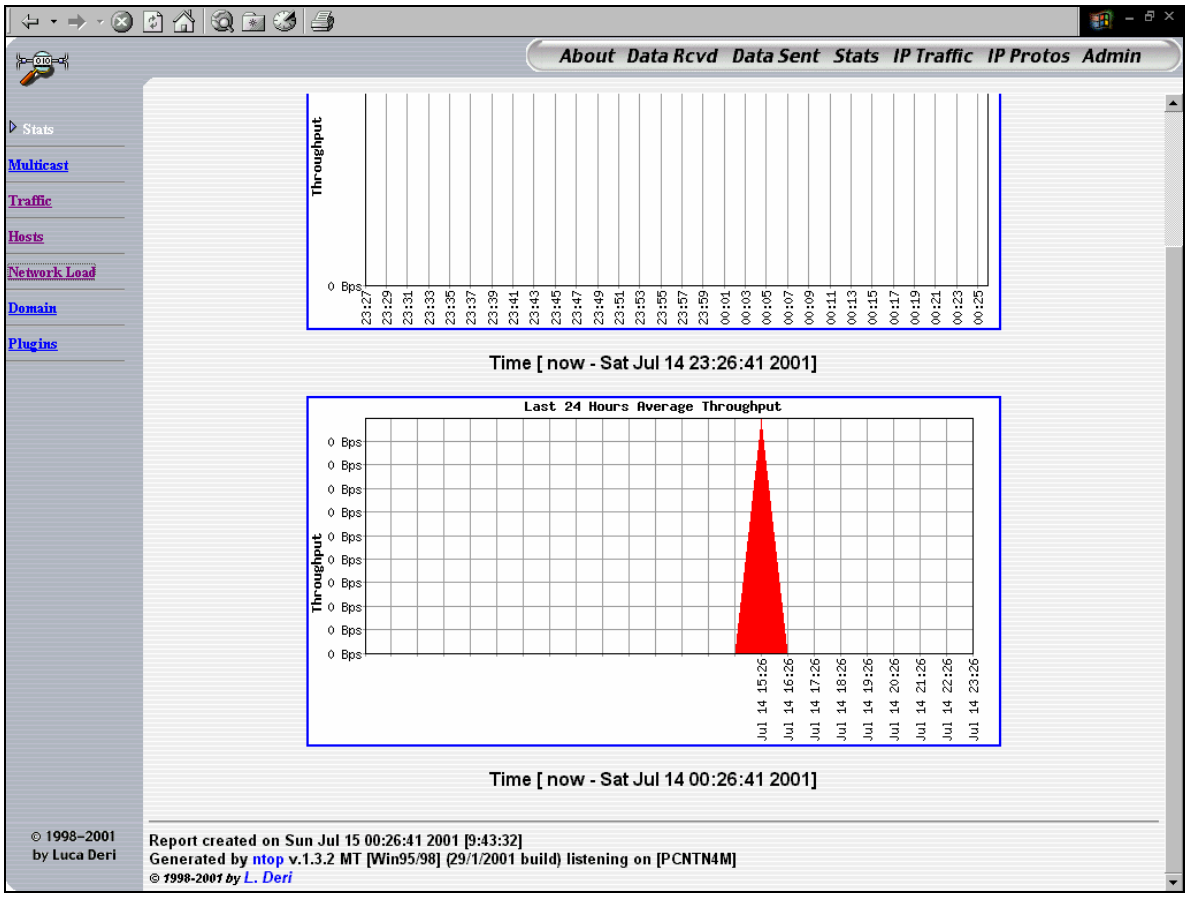

FIGURA 2.11 - Vazão da rede

Infelizmente, a figura 2.11 não pode fornecer muitos detalhes, pois a versão utilizada era restrita a captura de 1000 pacotes, mas é capaz de se observar os "picos" de utilização da rede, mesmo que a versão de demonstração (para windows) seja limitada e não forneça uma escala de *throughput* conforme mostra a figura 2.11.

2.5 NetEnforcer

O NetEnforcer, ao contrário dos *softwares* analisados até agora, é uma aplicação proprietária comercializada pela Allot Communications [ALL2002A]. É uma ferramenta projetada para ser utilizada pelos ISP (*Internet Service Provider*). Provedores com uma rede baseada em POP (*Point Of Presence*) estão agora preocupados em oferecer uma maior disponibilidade no serviço prestado (SLA, *Service Level Agreement*) aos consumidores. Estes, por sua vez, necessitam que haja compromisso com a precisão na tarifação e prestação do serviço em termos de largura de banda disponível, tempo de resposta, etc.

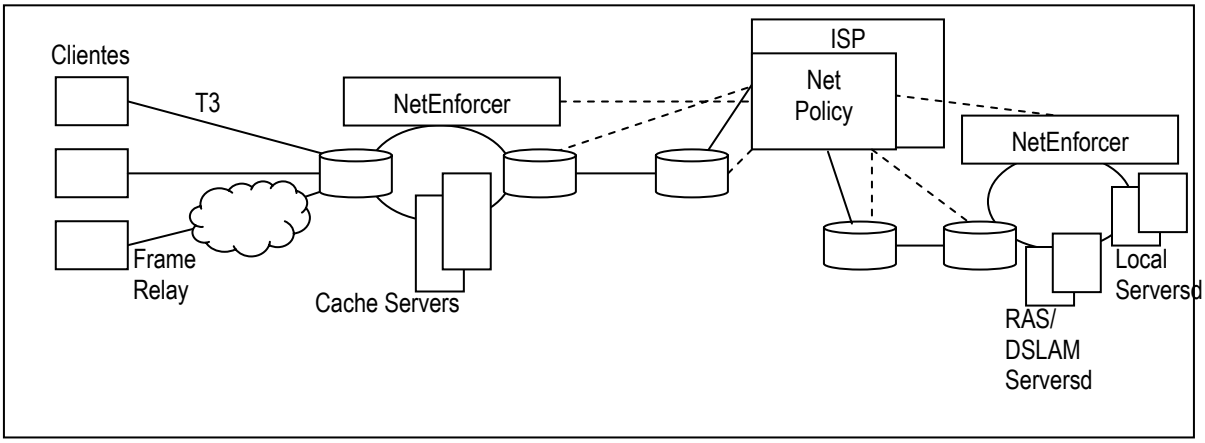

FIGURA 2.12 - Topologia de um ISP

A figura 2.12 dá uma visão da topologia envolvida em um ISP. As linhas tracejadas representam ligações lógicas entre elementos enquanto as linhas contínuas representam ligações físicas. O NetEnforcer é apenas uma das ferramentas utilizadas para monitorar o uso da rede pelas aplicações. Há ainda o NetPolicy que fornece uma visualização de QoS e faz interface com CCB: o CacheEnforcer que fornece a habilidade de redirecionar cache. O NetAccountant fornece estatísticas de tráfego para tarifação e o NetBalancer fornece um balanço da carga do servidor. Com exceção do NetEnforcer, as demais aplicações não serão estudadas nessa seção.

- Características

Segundo a empresa Allot Communications [ALL2002A], o NetEnforcer é capaz de criar classes de serviços (ouro, prata e bronze) para cada grupo de clientes, para cada aplicação ou para cada usuário. Pode priorizar o tráfego para um determinado grupo de usuários, priorizar tipos de serviços que necessitem garantia de banda como VoIP e videoconferência. Há a possibilidade de uma comunicação com programas externos para a tarifação através de uma interface para exportar informações referentes a tarifas. O gerenciamento de usuários é feito através de interação com um servidor LDAP. Todo o tráfego capaz de ser armazenado em *caches* é redirecionado para um *cache* local. Para

os serviços diferenciados, ele utiliza um *gateway* com TOS interno. Monitora a aplicação, os usuários e a utilização de banda e auxilia na resolução de problemas com congestionamento com uma monitoração em tempo real e estatísticas dos pacotes IP.

Com relação a segurança, ele consegue evitar ataques de DoS através de *ping storm* em servidores. O NetEnforce tem implementado técnicas de tolerância a falhas que incluem capacidade de *bypass* e redundância total.

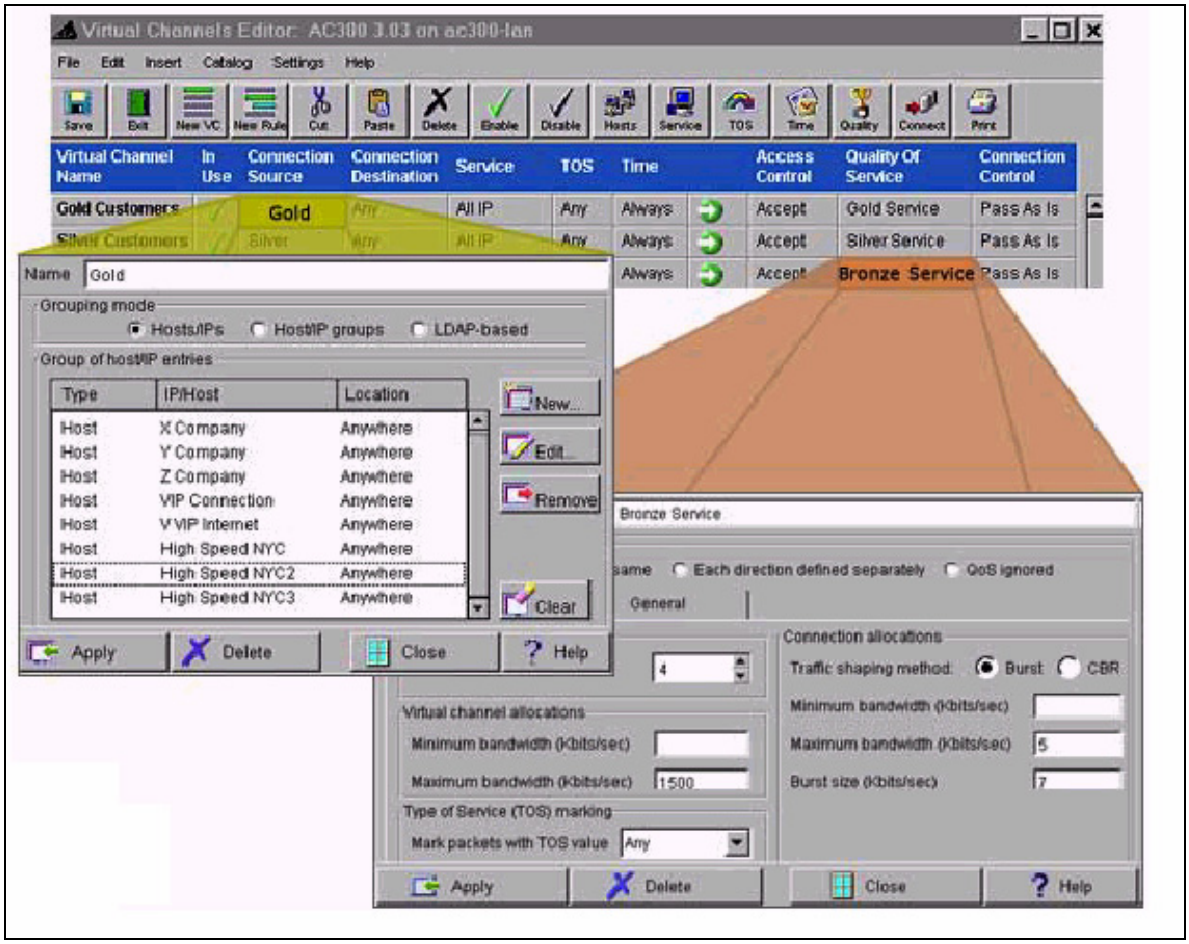

FIGURA 2.13 - Criando canais virtuais como NetEnforcer [ALL 2000a]

A figura 2.13 mostra como é possível definir classes de serviços. Elas são úteis quando se possui uma infra-estrutura de rede que é composta por vários roteadores e se deseja implantar qualidade de serviço.

Por se tratar de uma ferramenta comercial, a sua análise será mais restrita do que as demais implementações acadêmicas, uma vez que as informações sobre implementação e arquitetura são limitadas. As seções seguintes fazem um estudo de modelos de enfileiramento (*queueing*) adotados nos roteadores e traz uma solução implementada no NetEnforcer e divulgadas pela *Allot Communications* [ALL2000b].

- Enfileiramento baseado em classes

Quando um fluxo de dados de uma conexão chega em um roteador, seus pacotes componentes são armazenados em uma fila, de acordo com a prioridade designada pela sua classe de serviço associada e esperam a vez de serem despachados. À primeira vista esse algoritmo parece bastante justo, uma vez que o fluxo será priorizado de acordo com a sua classe de serviço, mas basta outro fluxo da mesma classe chegar no roteador que

esse cenário, anteriormente justo, mostra-se bastante ineficiente. Conforme pode ser visto na figura 2.14, um pacote do fluxo dois só irá deixar a fila no momento que o último pacote do fluxo um que entrou na fila deixar o roteador.

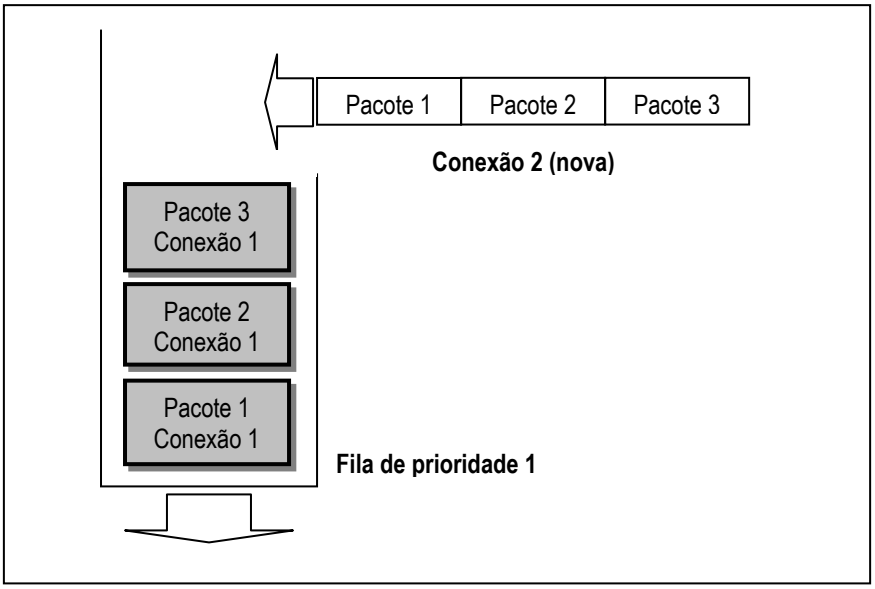

FIGURA 2.14 - Enfileiramento baseado em classes

Como se pode perceber, esse tipo de enfileiramento pode levar a resultados imprevisíveis e injustos (*fairless*) do ponto de vista do escalonamento de filas.

- Enfileiramento por fluxo (*Per-Flow Queuing*-PFQ)

O enfileiramento por fluxo é o algorítmo de escalonamento utilizado pelo NetEnforcer. Ele consiste em criar uma nova fila (dinamicamente) à medida que chegam conexões distintas para, dessa forma, garantir que todos os fluxos de mesma classe (prioridade) sejam processados simultaneamente (fig 2.15).

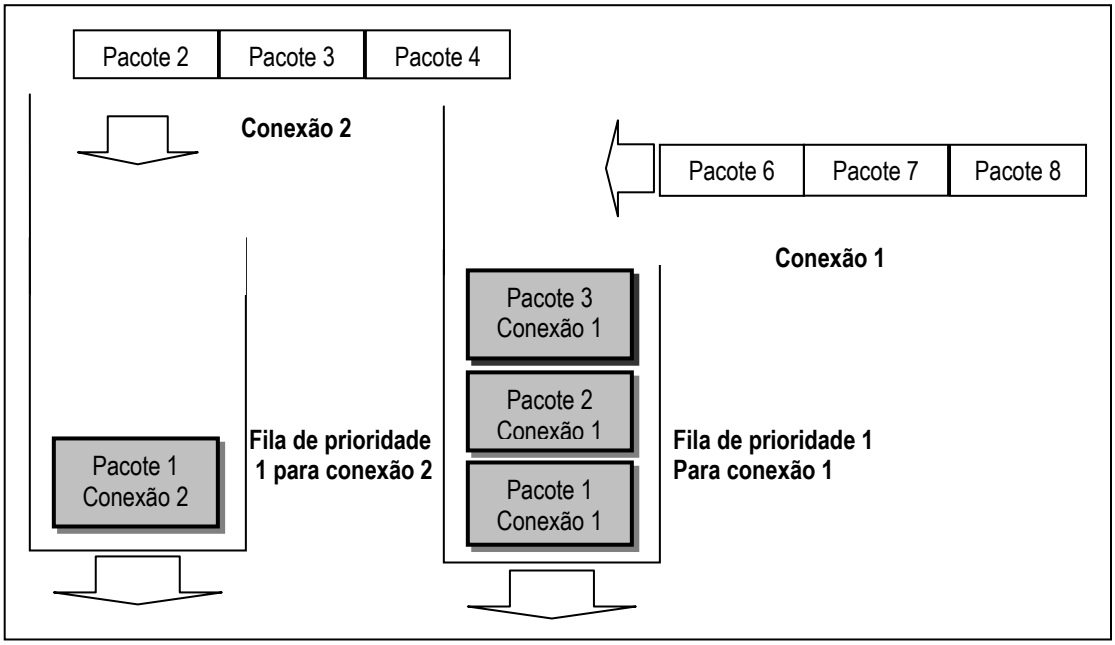

FIGURA 2.15 - Modelo de fila adotado no NetEnforcer

Uma vez que todos os pacotes de classes de uma mesma prioridade estão enfileirados, é utilizada uma combinação de algoritmos de escalonamento baseados em bandas e mecanismos de limitações de fluxo. É importante notar que essa estratégia para a manutenção do QoS também é utilizada na monitoração, em que o sistema utiliza as filas de prioridades para efetuar o cálculo do QoS atual e "realimentar" o sistema de filas.

Além de controlar a banda, todos os pacotes IPs são marcados de acordo com as definições de DiffServ [BLA 98] para facilitar o controle (monitoração) fim-a-fim do QoS.

2.6 VQmon

O VQmon é um software comercial desenvolvido pela Telchemy [TEL2002] para monitorar o QoS de transmissão de voz sobre pacotes IP.

O VQmon realiza uma monitoração passiva – não interferindo na carga da rede. São analisados fluxos RTP e parâmetros como perdas de pacotes em rajada, *jitter,* latência, o *codec* utilizado e ruído na transmissão. De posse desses dados, é obtido o que se denomina fator "R", utilizado para estimar a qualidade da transmissão de voz percebida pelo interlocutor.

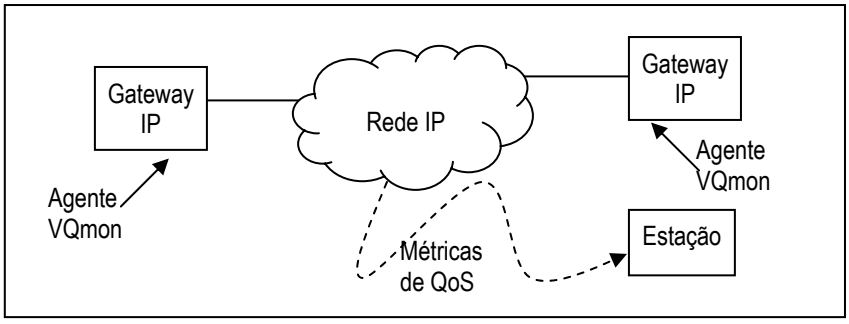

FIGURA 2.16 - Arquitetura do VQmon

A idéia principal do VQmon é que a pessoa que esteja utilizando o serviço de VoIP perceba, cada vez menos, a diminuição de qualidade na transmissão quando da perda de pacotes em rajada. Isso implica ações a serem tomadas de imediato no momento em que se detecte perdas em rajada: seja através de enlaces defeituosos, seja de atraso demasiado de vários pacotes, etc.

Para que seja possível ser adotada a estratégia anterior [BAJ2002], é necessário que se adotem políticas de estabelecimento de conexões e algoritmos de gerenciamento de banda baseados no fator "R".

- Operação

O VQmon é composto basicamente de um agente que é integrado em VoIP *gateways*, *gateways* residenciais, *IP Phones* e qualquer outro sistema terminal de VoIP que possua uma interface SNMP ou *specialized Service Management System* (fig. 2.16). As informações para monitoração são obtidas diretamente do CODEC de áudio (análise feita por chamada) e, em tempo real, através de RTP estatísticas são analisadas para satisfazer os requisitos de QoS.

O principal problema, em se tratando de voz sobre IP, é estimar a qualidade da voz num determinado momento. Essa estimativa subjetiva é obtida em tempo real baseada em dois fatores: *Mean Opinion Score* (MOS) e o fator "R". Essas informações podem ser recuperadas durante ou após uma chamada, armazenadas juntamente com as estatísticas das chamadas e comparadas com informações pré-gravadas fornecendo, dessa forma, relatórios automáticos sobre o estado de uma chamada.

Ligações telefônicas via comutação de circuitos, diferentemente de ligações via celular, possuem um grau de tolerância à degradação, por parte do usuário, muito baixo. Uma vez que o fator MOS de uma conversação é apenas uma estimativa baseada em testes subjetivos, a qualidade de uma chamada pode diminuir sensivelmente. É necessário, também, que se possam especificar métricas de monitoração objetivas e mensuráveis tais como atraso, *jitter*, vazão. Foi para isso que se introduziu o fator "R".

Serviços de QoS em uma rede são de vital importância; porém, bastante complexos de serem projetados e implantados. O paradigma atual de melhor esforço das redes IP não é suficiente para garantir qualidade em serviços dependentes do desempenho da rede como, por exemplo, serviços de multimídia.

A percepção da degradação numa dada transmissão pela monitoração do enlace já se justifica pelo fato de existirem mecanismos que trabalhem em cima da diminuição dessa qualidade, bastando apenas que ela seja detectada pelo sistema.

2.7 NetFlow

O NetFlow [CIS 2000] é um conjunto de ferramentas (ver fig. 2.17) que fazem uso do suporte existente em algumas versões de IOS dos produtos da *Cisco Systems.* O objetivo é coletar informações diversas sobre pacotes à medida que estes entram por uma interface de um roteador. Com base nesses dados, é possível extrair informações como: causa de congestionamento, classe de serviço (CoS) para cada usuário e aplicação, QoS associado e a rede de origem e destino para o tráfego observado.

No que diz respeito à QoS, este é determinado analisando o valor do campo ToS de um pacote IP. O principal objetivo do NetFlow é coletar dados e estatísticas para uma posterior análise, realizada por uma ferramenta chamada *Flow Collector*, que ajuda a compor a arquitetura do NetFlow (fig. 2.18). Vale notar que essa análise é realizada em uma estação separada (HP-UX ou Solaris) que processa dados de diversos roteadores para derivar estatísticas de uso.

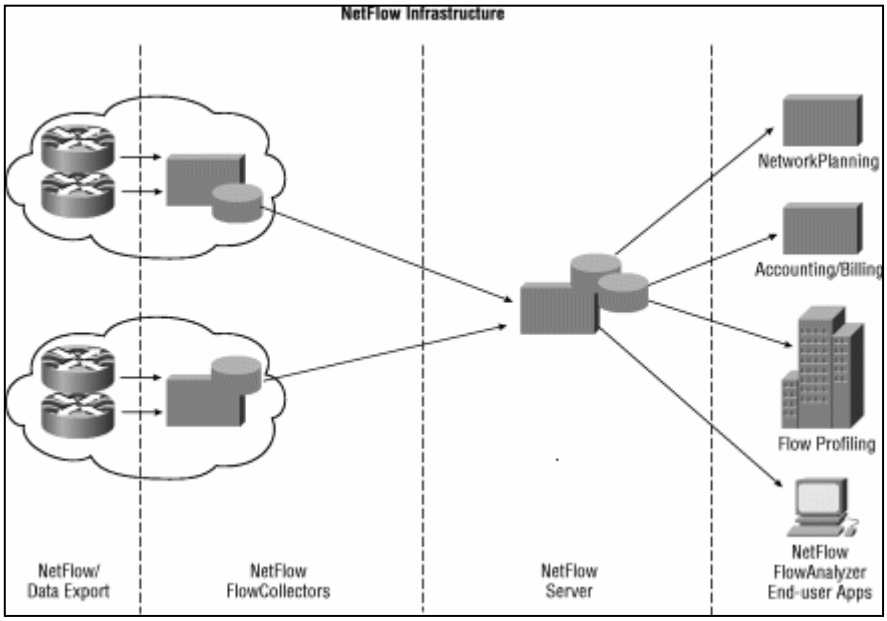

FIGURA 2.17 - Ferramentas que compõe o NetFlow [CIS 99]

O uso agregado do *FlowCollector* pode ser útil na obtenção de dados para:

- Conciliação de tráfego para o tipo de tráfego baseado em pares/pontos de transito.
- Análise do tráfego em um POP para planejamento de infra-estrutura.
- Análise de dados de *Web-hosting*.
- Dados para cobrança e SLM.
- Venda de serviços para outros provedores de redes.

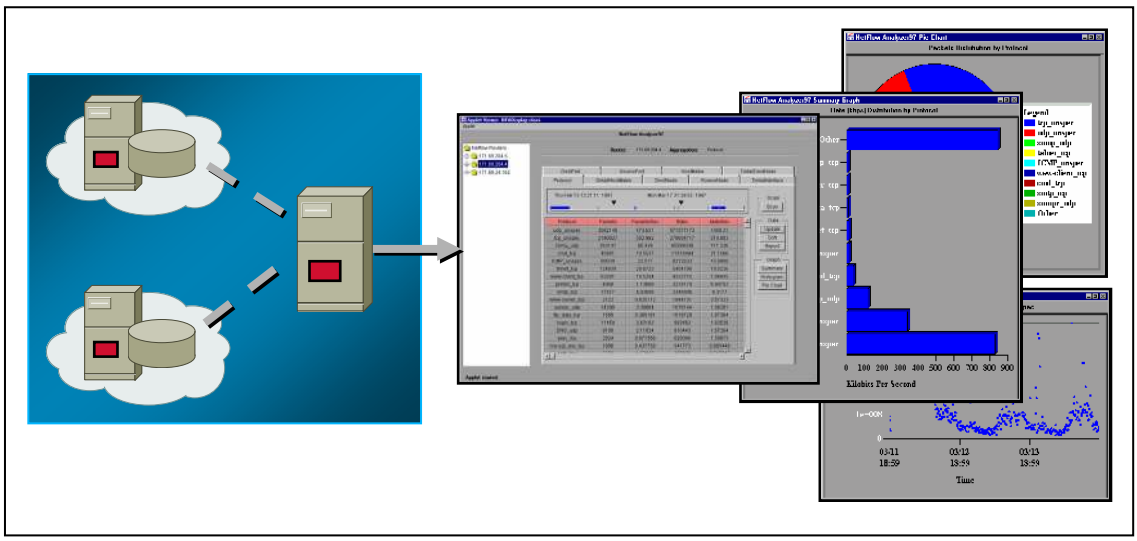

FIGURA 2.18 - Análise de dados da rede [CIS 2000]

O foco do NetFlow está no seu uso para aprimorar o método com o qual as empresas provedoras de acesso à internet (ISP ou provedores de *backbone*) possam tarifar de forma diferenciada os seus clientes. Com o estudo e caracterização do tráfego de cada cliente, poder-se-ia cobrar de maneira diferenciada o acesso à internet, baseado no uso e não no tempo de conexão, ou por taxas constantes independentes do uso (*flat rate*). Os clientes poderiam pagar tarifas diferenciadas, por exemplo, ao acessar a

internet pela rede da empresa ou de casa. Haveria a possibilidade de se tarifar por aplicação, estação ou classe de serviço.

As informações típicas de uma análise de fluxo encontrada nos registros de dados do NetFlow incluem:

- Endereços IP de origem e destino.
- Portas TCP/UDP de origem e destino.
- Type of service (ToS).
- Contagem de bytes e pacotes.
- *Timestamps* de início e fim.
- Números de interface de entrada e saída.
- *Flags* do TCP e protocolos encapsulados (TCP/UDP).
- Informação de roteamento (endereço do *next-hop*, número do sistema autônomo de origem (AS), número do AS de destino, prefixo da máscara de origem, prefixo da máscara de destino).

A monitoração da rede pelo NetFlow é feita através de análise de fluxos que permite a geração de um padrão de tráfego para um determinado fluxo. Esse padrão pode ser utilizado juntamente com um serviço de *troubleshooting* para o gerenciamento pró-ativo da rede, quando associado com outros roteadores e *switches*. Há ainda a opção de se utilizar um *Data Warehouse* para auxiliar no gerenciamento de uma rede que utilize o NetFlow.

2.8 Discussão

Nesse capítulo, foram analisadas cinco ferramentas que monitoram a qualidade do enlace de rede. Nem todas foram idealizadas para fornecer parâmetros de QoS para o administrador da rede, mas alguns são possíveis de serem amostrados, mesmo que tenham que ser obtidos via alguma combinação de informações que já se provenham. Algumas características em comum foram encontradas em todas as ferramentas, como o uso do protocolo SNMP para integração com plataformas de gerenciamento. Nenhuma das aplicações foi projetada exclusivamente para trabalhar com QoS: a maioria atua como se fosse um *sniffer* de rede, fornecendo ao usuário tantas informações quantas ele possa desejar, mas são informações não específicas para QoS devendo haver um certo processamento por parte do administrador. As aplicações comerciais implementam soluções proprietárias e fornecem uma monitoração que se adapta apenas a produtos do mesmo fabricante. A seguir é feito um resumo das carências de cada ferramenta, tomando como base de comparação a arquitetura proposta nessa dissertação.

O NeTraMet é a ferramenta, das analisadas, que mais se aproxima da solução proposta nessa dissertação, por possuir uma arquitetura distribuída e por poder ser utilizado quase como um *sniffer* de rede. O ponto fraco é a linguagem de baixo nível que deve ser utilizada na sua programação, pois ele não possui integrado nenhum mecanismo de políticas que controle a sua monitoração, além de deixar para o administrador da rede a tarefa de extrair os dados para derivar, por exemplo, taxa de perdas.

Já o Ntop não possui arquitetura distribuída e a sua grande vantagem está na forma como os dados são apresentados ao usuário (administrador da rede), dados esses que, para fins de cálculo de QoS atual, são deficitários, uma vez que com uma arquitetura centralizada seria impossível calcular a taxa de perdas.

Ao contrário das soluções anteriores, o NetEnforcer agrega o uso de políticas (alto nível de interação com o administrador da rede) com uma arquitetura distribuída;

conseguindo, assim, derivar a QoS proposta. A desvantagem está no fato da falta de integração com outras plataformas de gerenciamento de redes. A forma de monitoração é baseada em DiffServ e classes de serviços (CoS), o que acaba por restringir o uso dessa ferramenta a redes que suportem o uso de DiffServ.

O VQMon é uma ferramenta projetada para monitorar transmissão de voz sobre IP (VoIP) e, assim como acontece com o NetEnforcer, acaba por restringir o seu uso apenas a ambientes que suportem voz sobre IP.

Finalizando, o NetFlow se caracteriza pela coleta de dados (apenas em roteadores Cisco, executando o IOS adequado) para um posterior processamento, o que torna proibitivo o seu uso para monitorar QoS e avisar o administrador da rede, o mais rápido possível, em caso de degradação. Isso pode ser evidenciado pela própria empresa, que prega que o seu produto, em se tratando de QoS, apenas captura o nível da QoS de cada fluxo [CIS 99].

Nenhuma das ferramentas analisadas possui uma solução que integre gerenciamento de redes em alto nível (baseado em políticas) à monitoração de parâmetros de QoS (vazão, latência, taxa de perdas e *jitter*), e que, além disso, faça uso de um protocolo padronizado (SNMP) para uma fácil integração com outras plataformas de gerenciamento e, principalmente, que possa ser utilizado com o mínimo de adaptação das redes existentes sem necessidade de hardware proprietário. Por esses motivos, justifica-se a arquitetura proposta nessa dissertação.

3 Arquitetura Proposta

Como visto anteriormente, a monitoração de QoS é uma necessidade, e a monitoração por pontos de degradação é de especial importância para se determinar os pontos de degradação [RIB 2001]. Nesse contexto, este capítulo apresenta um sistema de monitoração baseado em políticas.

3.1 Políticas

Políticas são regras: nesse caso, regras de monitoração, que o administrador da rede define em uma linguagem no mais alto nível possível. Em uma rede de computadores com gerenciamento baseado em políticas, o administrador da rede tem a função de definir uma política baseada em parâmetros de QoS. Durante a definição, a política tem que ser validada de alguma forma, o que foge ao escopo dessa dissertação; após isso, ela deve ser aplicada na rede.

De acordo com o trabalho do IETF, um sistema típico de PBNM (*Policy-Based Network Management*) é composto por 4 elementos principais (fig. 3.1) [MOO 2002]:

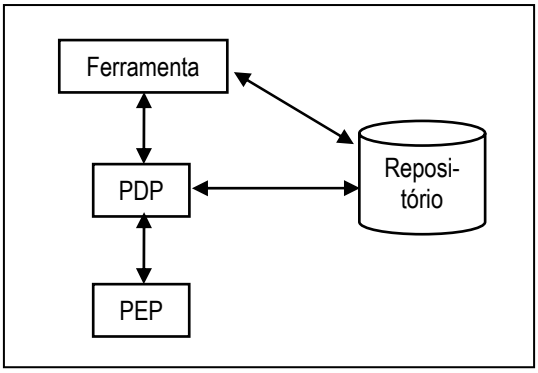

FIGURA 3.1 - Sistema PBNM.

A **aplicação de gerência** (ferramenta) fornece a interface com o usuário que permite que o administrador da rede edite, armazene e aplique políticas na rede. Para aplicar uma política, primeiramente, a política é editada e armazenada no **repositório de políticas**. O administrador da rede determina então em que dispositivos, e **PEPs** (*Policy Enforcement Point*), as políticas armazenadas deverão ser aplicadas. O sistema de PBNM, por sua vez, seleciona os **PDPs** (*Policy Decision Point*) para serem utilizados na tarefa da aplicação da política. Cada PDP selecionado é contatado pela aplicação de gerência, e a política é transferida. Finalmente, o PDP traduz a política em ações específicas para cada dispositivo, configurando os PEPs internos dos dispositivos selecionados previamente pelo administrador da rede (fig. 3.1).

Em uma rede com o gerenciamento baseado em políticas, cada dispositivo contém PEPs, os quais são elementos internos que executam as políticas definidas pelo administrador da rede. Exemplos de PEPs são as disciplinas de filas das interfaces internas de um roteador, os processos internos de priorização em dispositivos e o elemento de marcação em um roteador com DiffServ[BLA 98]. Um PEP é configurado ao se acessar o dispositivo que contém o PEP, usando algum protocolo de comunicação. O IETF sugere o protocolo de COPS/COPS-PR [DUR 2000] [CHA 2001], mas o SNMP, Telnet, HTTP, entre outros, podem ser usados também. Uma vez que um PEP é

configurado, o sistema de PBNM supõe que a política aplicada será respeitada corretamente, e nenhuma verificação adicional da política é executada.

Um PDP recebe políticas do repositório de políticas e tenta aplicá-las. Os métodos de transferência do tipo *push ou pull* podem ser usados para transferir uma política a um PDP. No método *pull*, o PDP é notificado sobre uma nova política armazenada no repositório de políticas. O PDP efetua o *downloads* da política usando algum protocolo (por exemplo, HTTP, FTP ou LDAP). No método *push*, entretanto, a política é transferida da plataforma de gerência que a recupera do repositório de política e a transfere por *upload* no PDP. Embora o IETF sugira o uso de LDAP [WAH 97] como o serviço para armazenar e transferir políticas, não há nenhum consenso em uma única solução. Por exemplo, mesmo o IETF, no contexto do grupo de trabalho do *snmpconf*, está trabalhando em um outro mecanismo de transferência de políticas baseado em SNMP [WAL 2002], enquanto Martinez et o al. [MAR 2002] propõem o uso de *script* MIB [LEV 2001] como a forma de transferência das políticas.

Após ter recebido uma política, o PDP prossegue com uma verificação dos possíveis conflitos na política [LUP 99]. Se nenhum conflito for detectado, o PDP contata os PEPs para configurá-los de uma maneira a suportar a política que está sendo aplicada. Pode-se dizer que os PDPs efetuam a tradução da política, partindo da sua definição até a configuração específica para um PEP.

O repositório de políticas é onde se armazenam as políticas da rede. Ao acessar o repositório de políticas, o administrador da rede pode definir e armazenar políticas novas, recuperar e modificar políticas previamente definidas, e pode remover as políticas velhas não mais usadas. Uma vez que uma política é definida e colocada no repositório, ela está pronta para ser utilizada.

3.2 Monitoração

O Sistema para Monitorar QoS é subdividido em um Controlador de Monitor e Monitores de QoS (fig. 3.2). Essa divisão é necessária, pois a monitoração é de natureza distribuída (por pontos de degradação conforme visto no capítulo anterior). Cabe ao Controlador de Monitor receber os parâmetros de QoS na forma de políticas, descobrir quais os Monitores de QoS que cobrem a "área" a ser monitorada e, mesmo com a ajuda do administrador, dividir o fluxo de monitoração em tarefas para cada Monitor de QoS relevante, de acordo com a localização de cada um.

- Controlador de Monitor

Do ponto de vista de um usuário, o Controlador de Monitor é um serviço que é executado em uma estação com a qual o administrador da rede interage. Cabe ao Controlador de Monitor a tarefa de receber as políticas de monitoração e estabelecer a melhor estratégia para, junto aos Monitores de QoS, coletar os dados relevantes àquela política e, após a devida compilação das informações, repassá-las ao administrador da rede. Para que esse mecanismo de constante interação com os Monitores de QoS funcione, é necessário que o Controlador de Monitor, além de ficar sempre aguardando as requisições do administrador da rede, seja capaz de programar o(s) Monitor(es) de QoS em tempo de execução.

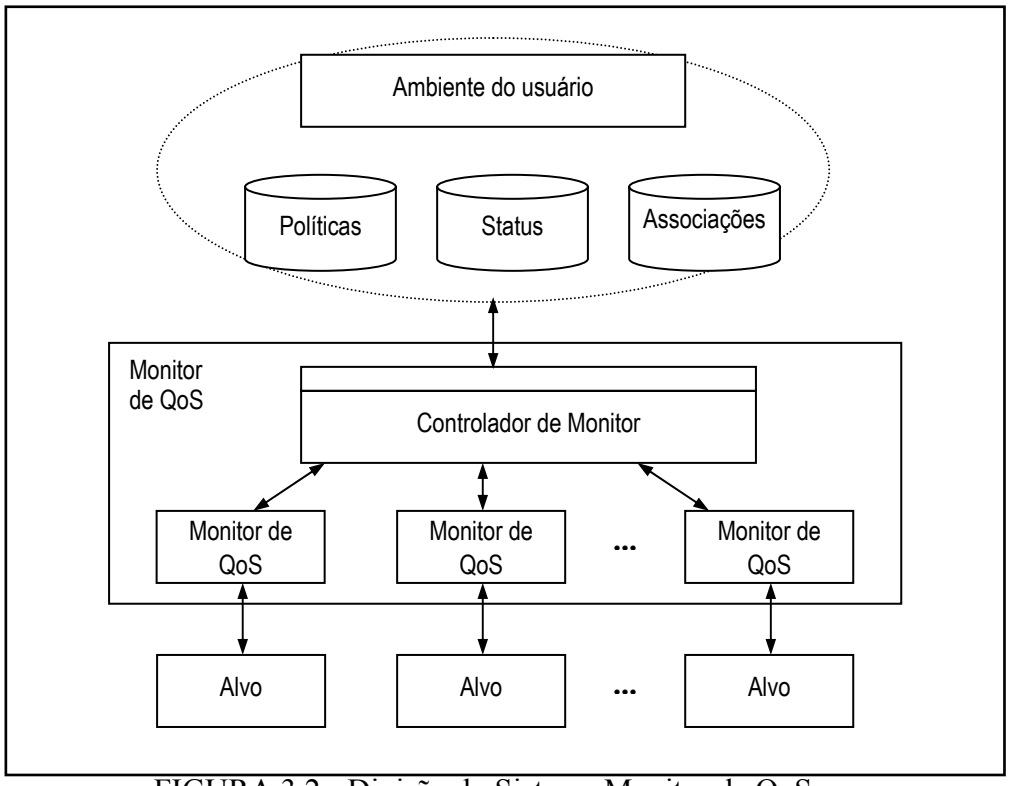

FIGURA 3.2 - Divisão do Sistema Monitor de QoS

Após receber os resultados das monitorações dos Monitores de QoS, o Controlador de Monitor é capaz de calcular parâmetros de QoS como o *jitter* e o atraso.

- Monitores de QoS

Um Monitor de QoS tem como principal função ficar monitorando fluxos ou agregados, coletando dados e enviando informações relativas aos pedidos de monitoração que recebeu do Controlador de Monitor. É importante notar que os Monitores de QoS não trocam mensagens entre si, apenas com o Controlador de Monitor, assim como só são capazes de informar sobre o estado do enlace e a vazão atual. Parâmetros de QoS como o atraso e o *jitter* do enlace não podem ser derivados diretamente por apenas um Monitor de QoS, uma vez que são necessários dados de vários Monitores de QoS para o seu cálculo.

Caso um fluxo crítico estiver recebendo menos banda que o necessário, o Monitor de QoS associado ao nodo problemático detecta essa condição e informa ao administrador da rede. Quando pacotes de um mesmo fluxo ou agregado chegam nas interfaces dos dispositivos monitorados em intervalos irregulares, a variação é detectada e o Monitor de QoS novamente informa o administrador da rede desse evento. Alguns parâmetros de QoS podem ser determinados no próprio Monitor de QoS, como a vazão e no caso de estar localizado em um roteador a taxa de perdas. Outros parâmetros de QoS, entretanto, só podem ser determinados pelo Controlador de Monitor, como é o caso do atraso, do *jitter* e, em alguns casos, da taxa de perdas. Por exemplo, a determinação do atraso enfrentado por um pacote é conseguida analisando-se o tempo de saída desse pacote na interface de um dispositivo com o tempo de chegada do mesmo pacote na interface de entrada de outro dispositivo. Essas informações são coletadas acessando-se os Monitores de QoS localizados em cada um dos dispositivos envolvidos.

3.3 Ciclo da monitoração baseada em políticas

Após a correta aplicação de uma política em uma rede, o administrador da rede deve verificar se a política definida está sendo implantada na pratica. É necessário algum esquema de monitoração na rede para que o administrador possa comparar, manualmente, a QoS observada na rede com aquela que foi definida e validada pelo consumidor de políticas. Tendo como base esta situação, na qual o administrador da rede é responsável por definir uma política, encontrar uma forma de monitorar a rede e, manualmente, comparar os valores definidos com os monitorados, foi proposto que as políticas sejam utilizadas como dados de entrada na monitoração de QoS. Bastaria, portanto que o administrador da rede definisse uma política, e essa, automaticamente, iniciasse uma monitoração na rede e informasse o administrador do estado do QoS, como mostra a figura 3.3.

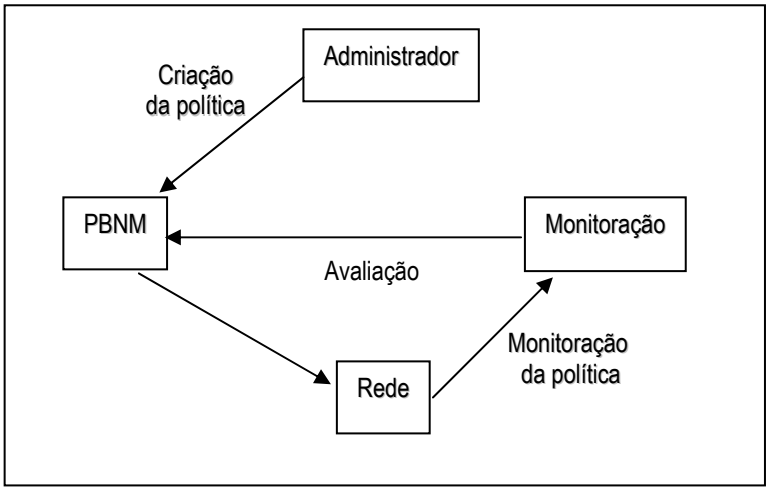

FIGURA 3.3 - Ciclo de uso de uma política

O administrador da rede entra com as políticas na estação de gerência, e elas são armazenadas no repositório de políticas. A seqüência de passos pode ser resumida à seguinte:

- uma vez definidas as políticas, resta informar na estação de gerenciamento quais pontos da rede deverão participar da coleta de informações do fluxo ou agregado que foi definido através das políticas;

- a estação de gerência sinaliza para a aplicação que for monitorar a QoS que, por fim, realiza a monitoração na rede comparando o resultado da monitoração com o das políticas retiradas do repositório;

- o estado de QoS é então armazenado em um lugar apropriado e fica disponível para o administrador da rede na estação de gerenciamento;

- se por acaso um alarme for gerado pela monitoração de QoS, uma notificação é enviada para o administrador da rede para que ele verifique o estado dos alarmes. Caso a notificação enviada pela monitoração de QoS não atinja o seu destino por qualquer motivo, o administrador da rede vai, mesmo assim,

perceber o estado de QoS fazendo um *polling* periódico onde o estado da política é armazenado.

3.4 Integração

A arquitetura padrão de PBNM é estendida para incluir os elementos capazes de fornecer alguma forma de se monitoração QoS de forma integrada [RIB 2003]. A figura 3.4 desse capítulo apresenta a arquitetura estendida proposta.

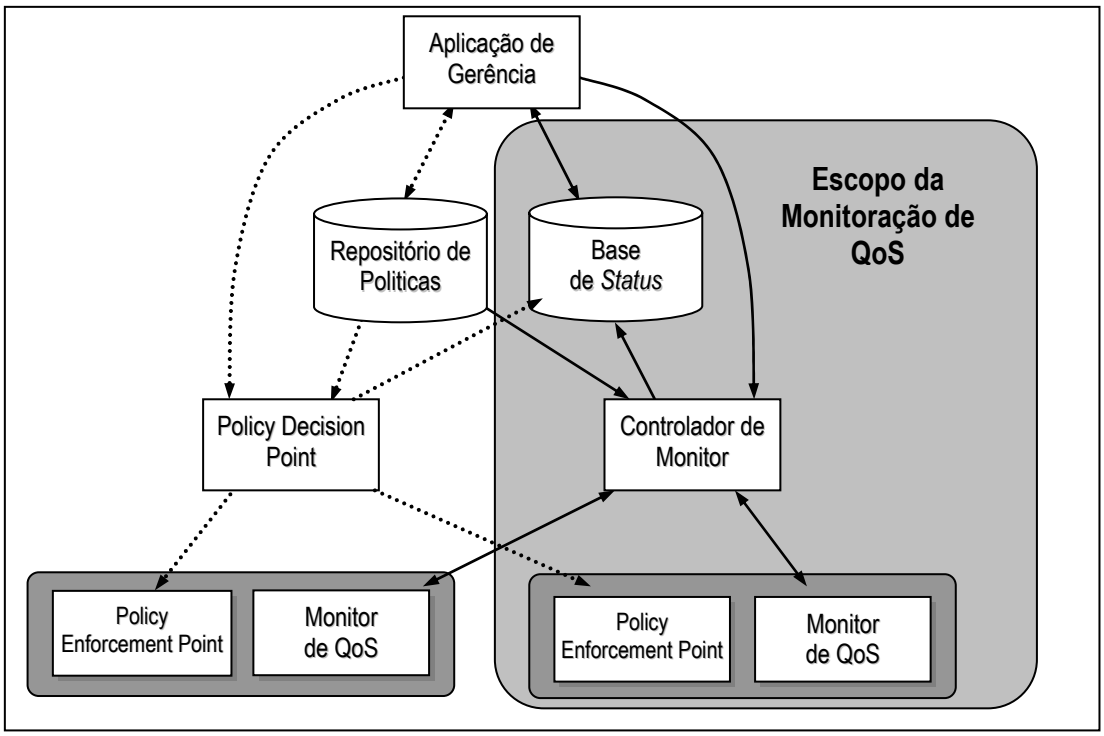

FIGURA 3.4 - Arquitetura PBNM estendida para suportar Monitoração de QoS.

No lado esquerdo, pode-se ver os elementos padrões de PBNM (repositório de política, PDP e PEP) e as interações comuns através das linhas pontilhadas. No lado direito, os novos elementos introduzidos são apresentados, e as linhas contínuas mostram as interações entre esses elementos. Partindo da figura 3.3, as subseções seguintes explicam as operações gerais executadas por elementos de cada arquitetura, de uma perspectiva *bottom-up*.

Conforme visto anteriormente (fig 3.4), os consumidores de políticas (PDP) implantam as políticas nos dispositivos da rede traduzindo descrições de políticas em comandos específicos para cada equipamento (alvo). Os Monitores de QoS analisam o tráfego de segmentos críticos da rede e comparam o comportamento real com o comportamento esperado, para garantir que as políticas especificadas estão sendo aplicadas na prática. Se a diferença de comportamento for maior que um limiar crítico, o Monitor de QoS envia uma notificação ao administrador da rede (aplicação de gerência) indicando uma degradação de QoS (fig 3.4).

- Dispositivos, PEPs e Monitores de QoS

Para permitir verificações adicionais, usa-se um Monitor de QoS que executa uma verificação de vazão e *jitter* para um fluxo ou agregado definido dentro de um

dispositivo de rede. O conceito de um sistema para monitorar QoS foi introduzido previamente por Jiang, Tham e Ko [THA 2000] [JIA 2000b].

O Monitor de QoS é configurado com as definições dos fluxos a serem monitorados, que são descritas em termos de:

- endereços de rede de origem e destino,
- portas de origem e destino,
- protocolo de transporte,
- DSCP/ToS [NIC 98],
- e outros, como será descrito no decorrer dessa dissertação.

Cada fluxo é monitorado em uma interface de rede específica, observando-se o sentido em que os dados trafegam: entrando, saindo ou ambos (em relação ao endereço de origem que se esta sendo monitorado). O Monitor de QoS mantém-se a par do *jitter* e da vazão associados a cada fluxo ou agregado definidos e notifica a elementos externos se o *jitter* ou a vazão observados excederem algum dos limites fornecidos pelo administrador da rede. Um Monitor de QoS é também responsável por fornecer as informações dos fluxos a elementos externos, o que é uma exigência para se obter a taxa de perdas e o atraso em um fluxo.

É importante observar que nem o PEP nem um Monitor de QoS estão cientes das políticas da rede. O PEP e o Monitor de QoS são configurados apenas por elementos externos que, por sua vez, compreendem as políticas da rede.

- PDP e Controlador de Monitor

Da perspectiva de um dispositivo, o PDP e o Controlador de Monitor são os elementos externos que estão cientes sobre as políticas da rede.

Os Controladores de Monitor podem ser vistos como os PDPs do processo de monitoração de QoS. Exatamente como os PDPs, os Controladores de Monitor acessam as políticas do repositório de políticas (através de *push* ou *pull*, utilizando algum protocolo de transferência): eles verificam problemas com uma política através da monitoração de um dispositivo e notificam elementos externos quando degradações na política são observadas. Uma diferença importante entre PDPs e os Controladores de Monitor é que o Controlador de Monitor verifica políticas após a distribuição da política, enquanto um PDP deve verificar a política durante sua distribuição. A verificação da política executada pelo Controlador de Monitor requer alguma interação Controlador de Monitor/Monitor de QoS, como segue:

1. A política transferida é analisada e as suas definições de fluxo e parâmetros de QoS são determinadas. Nessa dissertação se está tratando com quatro parâmetros principais de QoS: vazão, atraso, taxa de perdas e *jiiter*;

2. O Controlador de Monitor contata um conjunto de Monitores de QoS e configura-os informando: (a) o fluxo a ser monitorado, (b) a vazão prevista e *jitter* associados ao fluxo e (c) os limiares de monitoração para a vazão e o *jitter*.

Uma vez que problemas na vazão ou *jitter* são detectados pelo Monitor de QoS, uma notificação é emitida ao Controlador de Monitor. Esta notificação provoca uma atualização na base de dados de estado (base de *status*) efetuada pelo Controlador de Monitor. Opcionalmente, a notificação pode ser enviada à aplicação de gerência, a qual pode tratar o problema detectado. A fim de verificar atraso e os problemas de perda, o

Controlador de Monitor da rede analisa os dados coletados nos Monitores de QoS. O atraso e a perda medidos são comparados com a máxima perda e atraso definidos pelo administrador da rede. Se uma violação for detectada, novamente, o controlador atualiza a base de dados de estado e, opcionalmente, notifica a aplicação de gerência. Pode-se comparar Controladores de Monitores e Monitores de QoS considerando a verificação dos parâmetros de QoS: o Controlador de Monitor verifica atraso e problemas com perdas, enquanto os Monitores de QoS cuidam de *jitter* e vazão.

- Repositório de Políticas, Base de *Status* e Aplicação de Gerenciamento

Como visto antes, o processo de distribuição da política pode falhar. Uma política pode falhar, também, mesmo após sua distribuição, devido à degradação de QoS. Assim, as políticas que estão sendo usadas possuem um estado o qual deve ser acessado pelo administrador da rede para verificar o sucesso da política. Tal estado de uma política é fornecido pela base de dados de *status*. Essa base de dados é acessada tanto pelo PDPs quanto pelo Controlador de Monitor. Os PDPs atualizam o estado de uma política no momento em que os problemas são detectados na distribuição da política. Por outro lado, o Controlador de Monitor atualiza o estado de uma política quando a degradação de QoS é detectada pelo Sistema de Monitoração após a distribuição da política.

Embora o IETF não defina uma separação explícita entre a definição de uma política e o armazenamento do *status* de uma política, acredita-se que essa distinção é necessária porque nem toda a política armazenada é também uma política que está sendo utilizada. Além disso, as políticas tendem a ser definidas uma vez e aproveitadas diversas vezes. Isso implica o uso de um serviço de armazenamento do tipo *write-once/read-several*, como é o caso do LDAP [WAH 97]. O *status* de uma política, entretanto, tende a ser escrito e lido diversas vezes: escrito quando os problemas na rede são detectados pelos PDPs ou pelo Controlador de Monitor, e lido no momento em que o administrador da rede prossegue com uma verificação do estado de uma política. Nesse caso, um serviço de armazenamento do tipo *write-once/read-several* não é apropriado.

O elemento superior da arquitetura é a aplicação de gerência. É o ponto central de onde a distribuição da política e os processos que monitoram QoS são controlados. A fim de prosseguir com tal controle, a aplicação de gerenciamento utiliza ambos, o repositório de políticas e a base de dados de *status*, e tem acesso direto aos PDPs e aos Controladores de Monitores.

3.5 Questões Operacionais

Nessa sessão serão discutidas questões relacionadas ao funcionamento do sistema proposto tais como a seleção e o posicionamento dos Monitores de QoS e questões relacionadas ao uso de políticas, uma “pseudo base de dados” ao invés de uma relacional, sem abordar aspectos de implementação, os quais serão discutidos no próximo capitulo.

Ao se fazer uso de uma aplicação de gerência, as seguintes etapas são executadas tipicamente na distribuição das políticas e nos processos de monitoração.

1. O administrador da rede define as políticas a serem usadas na rede e as armazena no repositório de políticas.

2. O administrador seleciona os PEPs [WES 2001] nos quais as políticas serão aplicadas.

3. A aplicação de Gerência, baseada nos PEPs selecionados, determina quais PDPs [WES 2001] devem ser usados na aplicação da política e transfere aos PDPs a definição da política e uma lista dos PEPs previamente selecionados pelo administrador. Significa que no processo de aplicação de políticas o administrador da rede não toma conhecimento sobre os PDPs, mas sim dos PEPs.

4. Para verificar a evolução do processo de aplicação de uma política, o administrador acessa a base de *Status* da política.

5. As políticas críticas têm que ser monitoradas. O administrador da rede informa à aplicação de gerência algumas políticas críticas e limiares de monitoração e, baseado na informação da aplicação prévia da política, a aplicação da gerência seleciona os Controladores de Monitor e os Monitores de QoS que verificam as políticas críticas selecionadas.

6. Opcionalmente, o Administrador pode selecionar Controladores de Monitores e Monitores de QoS adicionais, manualmente, ou desselecionar os que selecionou previamente.

7. Então, a aplicação da gerência transfere as políticas críticas e os limiares críticos ao Sistema de Monitoração, enviando também uma lista de Monitores de QoS ao Controlador de monitor.

8. Finalmente, o administrador verifica se uma política crítica está sendo respeitada na rede acessando a base de dados de S*tatus* da política, ou opcionalmente o Sistema de Monitoração notifica ao administrador diretamente.

- Seleção de Monitores de QoS Relevantes

Um ponto crucial do sistema de monitoração de QoS é a seleção dos Monitores de QoS relevantes. Tham, Jiang e Ko [THA 2000] propõem o uso de Monitores de QoS que se registram em um "Servidor de Nomes para Aplicações de Tempo-Real", *Real-Ttime Application Name Server* (RTANS) para cada fluxo percebido que passa através dos Monitores de QoS. A aplicação da gerência pode determinar quais os Monitores de QoS relevantes de um fluxo que é percebido através do RTANS. Estes procedimentos facilitam a seleção dos Monitores de QoS relevantes na aplicação de gerência, mas forçam cada Monitor de QoS a manterem-se a par de cada fluxo percebido. Na proposta dessa dissertação, por outro lado, a determinação de Monitores de QoS relevantes não é assim fácil, mas permite que os Monitores de QoS selecionados armazenem e analisem poucos fluxos fornecidos pela aplicação de gerência [RIB 2001a]. Será utilizado o seguinte cenário para ilustrar essa questão de seleção dos Monitores de QoS:

A figura 3.5 apresenta uma rede com Gerenciamento Baseado em Políticas com alguns Monitores de QoS. O computador A gera um fluxo que será emitido ao computador B. Este fluxo é roteado através dos dispositivos 2, 5, 4, 6 e 7.

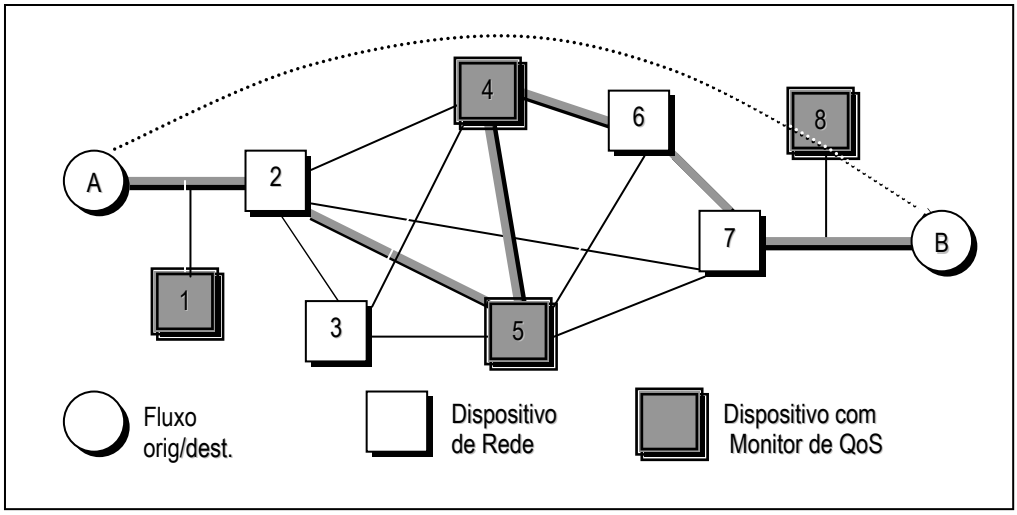

FIGURA 3.5 - Seleção de Monitores de QoS Relevantes

Alguns dispositivos têm internamente um Monitor de QoS (dispositivos 4 e 5); outros não (dispositivos 2, 6 e 7). Há também os dispositivos que não roteiam o fluxo gerado, mas são utilizados no processo de monitoração (dispositivos 1 e 8). Há duas formas para se selecionar esses Monitores de QoS. A primeira consiste em fornecer uma interface de usuário que permita ao administrador da rede declarar explicitamente os Monitores de QoS que devem ser usados. A segunda é mais sofisticada e consiste em fornecer mecanismos automatizados que investiguem a rede gerenciada e forneçam a identificação dos Monitores de QoS atravessados por um fluxo. Para o exemplo, o administrador da rede poderia informar o fluxo a ser monitorado, e a aplicação da gerência prosseguiria com a detecção automática do trajeto do fluxo e de seus Monitores de QoS associados. Em ambos os casos, uma lista dos Monitores de QoS a ser usados é gerada, manualmente (no primeiro caso) ou automaticamente (no segundo caso).

A lista dos Monitores de QoS pode ser editada antes que os Monitores de QoS sejam programados. Esta é uma exigência porque o administrador da rede pode estar interessado em escopos diferentes da monitoração de um fluxo. Para o exemplo, se a QoS fim-a-fim for mais importante do que a QoS do núcleo da rede, somente os Monitores de QoS 1 e 8 poderiam ser utilizados. Se a QoS de um *hop* específico for importante, poder-se-ia usar os Monitores de QoS 4 e 5. É necessário observar que em alguns casos não é possível obter-se a QoS de um segmento. Por exemplo, não é possível recuperar a QoS entre os dispositivos 6 e 7. Significa que a monitoração de QoS do núcleo da rede depende da disposição e disponibilidade dos Monitores de QoS.

A fim de iniciar o processo de monitoração, a etapa seguinte é transferir a política a ser monitorada ao sistema de monitoração. Uma vez que os Monitores de QoS não compreendem políticas de rede, um elemento que centralize os Monitores de QoS tem que ser usado. Embora a seleção dos Monitores de QoS inclua uma interação com o administrador da rede, a seleção de Controladores de Monitor não é necessária.

- Seleção e Operação do Controlador de Monitor

A seleção do(s) Controlador(es) de Monitor é coordenada totalmente pela aplicação de Gerência. Virtualmente, nenhuma interação com o administrador da rede é

requerida. O algoritmo para a seleção dos controladores é completamente simples: encontre um conjunto de Controladores de Monitor que podem alcançar tantos quanto possíveis Monitores de QoS selecionados e possam também ser notificado por tais Monitores de QoS.

Com esta suposição, pode-se concluir que um único fluxo poderia ser monitorado usando um único Controlador de Monitor, mas um único Controlador de Monitor pode controlar a monitoração de diversos fluxos usando Monitores de QoS diferentes. Além disso, um único Monitor de QoS pode ser controlado por Controladores de Monitor diferentes no processo de monitoração de diferentes fluxos que estão no escopo do Monitor de QoS.

Depois de selecionados os Controladores de Monitor para coordenar a monitoração de um fluxo, a aplicação de gerência está pronta para iniciar o processo de monitoração. A fim de dar início, a aplicação de gerência contata os Controladores de Monitor selecionados e emite (usando os modelos *push* ou *pull,* discutidos anteriormente):

- A política a ser monitorada;
- A lista dos Monitores de QoS Selecionados;
- Os limites para os parâmetros de QoS;

Uma vez que estas informações estão disponíveis em um Controlador de Monitor selecionado, a primeira ação executada é a derivação dos parâmetros de QoS a serem monitorados, baseados na política transferida. Então, o Controlador de Monitor contata os Monitores de QoS da lista e configura-os para monitorar a vazão e o *jitter* baseados nos valores de QoS e nos limites estabelecidos. A fim verificar os problemas de atraso e perdas de pacotes, o Controlador de Monitor programa também os Monitores de QoS para emitirem, em caso extremo, uma informação indicando o estado do QoS para que uma ação apropriada possa ser executada pelo Administrador da Rede. Em situação normal de uso, o Controlador de Monitor faz um *poll* periódico nos Monitores de QoS para obter o valor do QoS e deixa-lo disponível para consulta.

4 Implementação da Solução

Este capítulo trata de uma implementação possível para o Sistema Monitor de QoS. Inicialmente será dada ênfase à utilização do LDAP (*Lightweight Directory Access Protocol*) como meio de armazenagem de políticas. Há ainda, a necessidade de se adotar alguma definição de políticas para QoS, uma vez que será estudado apenas esse subconjunto de políticas. Após isso, é necessário que se utilize algum mapeamento de políticas de QoS para um esquema de dados que possa ser implantado em um servidor LDAP.

Após a implementação de um servidor LDAP com políticas de QoS, é discutida a implementação de um Controlador de Monitor, assim como uma MIB definida para fazer a interface com o administrador da rede. Finalmente, assim como o feito com o Controlador de Monitor, é discutida a implementação de um Monitor de QoS e a sua validação. Durante a fase de desenvolvimento do protótipo, foram feitas análises sobre as possíveis formas de se implementar o sistema.

A seguir serão abordados alguns aspectos da implementação, as soluções encontradas, assim como algumas considerações sobre possíveis vantagens ou desvantagens sobre as demais. A seção 4.8 descreve o processo de implementação de um Agente Monitor de QoS, com exemplos de funções que implementam algumas capacidades consideradas principais nos agentes. A seção 4.9 descreve como foi implementado o agente RMON, e a seção 4.10 trata de alguns detalhes da implementação de um Controlador de Monitor. Todos os exemplos que são apresentados estão escritos em linguagem C e fazem parte de uma implementação real. A validação do protótipo discutido nesse capítulo será apresentada em uma seção subseqüente.

4.1 Uso do LDAP

Pode-se dizer que o LDAP é uma base de dados simplificada com métodos de acesso específicos. Para a obtenção de QoS e a sua constante monitoração, são definidas políticas de QoS, que são os parâmetros passados pelo administrador da rede para o sistema que informam as características que se desejam nos serviços. O LDAP é usado para o administrador enviar as políticas de monitoração para os Monitores de QoS, que fazem a monitoração. Acaba por fazer a "interface" entre o computador que armazena as políticas para o gerenciamento da rede e toda a parte que faz a monitoração, compreendendo os Monitores de QoS e o Controlador de Monitor, ao qual os Monitores de QoS se reportam e com o qual o administrador interage.

Para que se tenha mais facilidade de uso, uma maior funcionalidade e os recursos na rede sejam usados da melhor maneira possível, no que diz respeito a uma arquitetura distribuída na rede, uma forma segura e robusta de armazenagem de dados comuns às aplicações se faz necessária. Foi dessa necessidade de vários aplicativos distribuídos na rede acessarem dados comuns que surgiu o LDAP.

Informações, por exemplo, sobre contas de usuários, aplicativos, arquivos, impressoras e quaisquer outros recursos da rede são, muitas vezes, armazenadas em bases de dados especiais denominadas **diretórios**. O LDAP é um padrão desenvolvido para esses fins. São definidos métodos no LDAP para o acesso e a atualização de dados num diretório. Está, cada vez mais, ganhando a preferência como método de acesso a diretório pelos desenvolvedores de aplicações para a Internet e pelas empresas, já sendo incorporado a um número crescente de aplicativos.

No caso específico de uma política, a qual, via de regra, é escrita apenas uma vez e lida várias vezes, o LDAP se constitui em uma solução bastante recomendável para o armazenamento das políticas a serem utilizadas.

Há duas formas de uma política de gerenciamento ser transferida do Administrador da rede para o Controlador de Monitor:

- A política de gerenciamento pode ser **enviada** diretamente para o Controlador de Monitor. Esse tipo de transferência denomina-se ***push model*;**

- A **localização** da política de gerenciamento é informada pelo administrador da rede ao Controlador de Monitor e este é responsável pela transferência do *script* (***pull model***).

4.2 Implementação e uso de políticas de QoS em um servidor LDAP

O administrador da rede pode ser instruído a implementar uma política que tenha por objetivo tornar o comportamento do tráfego de VoIP similar ao tráfego de voz em uma rede telefônica. Esse comportamento é caracterizado em termos de atraso, vazão e *jitter*, por exemplo. E esse conjunto de informações é denominado *Business Rule* [SNI 2003]. Por outro lado, os requerimentos para uma transação via Web ou o tráfego de correio eletrônico podem ser expressos em termos de outras variáveis que não as citadas anteriormente. É essa variedade no mapeamento de requerimentos para condições específicas que recebe o nome de QPIM (*QoS Policy Information Model*) [SNI 2003]. O QPIM é necessário pois mapeia os requisitos de QoS, que são comuns e conhecidos pelo administrador, às capacidades específicas de cada dispositivo.

Há ainda a tarefa de se determinar o papel (*role*) de cada dispositivo no desempenho geral da rede [MOO 2001], [MOO 2002], por exemplo, interfaces diferentes de um mesmo roteador podem desempenhar papéis diferentes no funcionamento da rede. Uma mesma interface pode ter portas diferentes com importâncias diferentes também. Essa é a necessidade de se definir o papel de cada elemento da rede, sempre levando em conta fatores subjetivos como a topologia da rede, os protocolos envolvidos, etc.

Os construtores de políticas definem a funcionalidade necessária para propiciar as condições ideais de tráfego à rede para um tipo particular de tráfego. As funções envolvidas irão depender do tipo de tecnologia utilizada, por exemplo, RSVP ou DiffServ. A modelagem dessas classes, por exemplo, estão definidas no QPIM.

É importante notar que a definição de uma política é independente de sua implementação. A figura 4.1 faz uma representação do fluxo de informações que inicia com o administrador da rede e termina na configuração dos dispositivos.

- Orientação voltada à definição de políticas

O objetivo principal do QPIM é modelar uma política de forma que essa possa ser descrita o mais perto possível da forma como é feita na comunicação humana. Portanto, o QPIM não tem por objetivo auxiliar na configuração de equipamentos ou da rede.

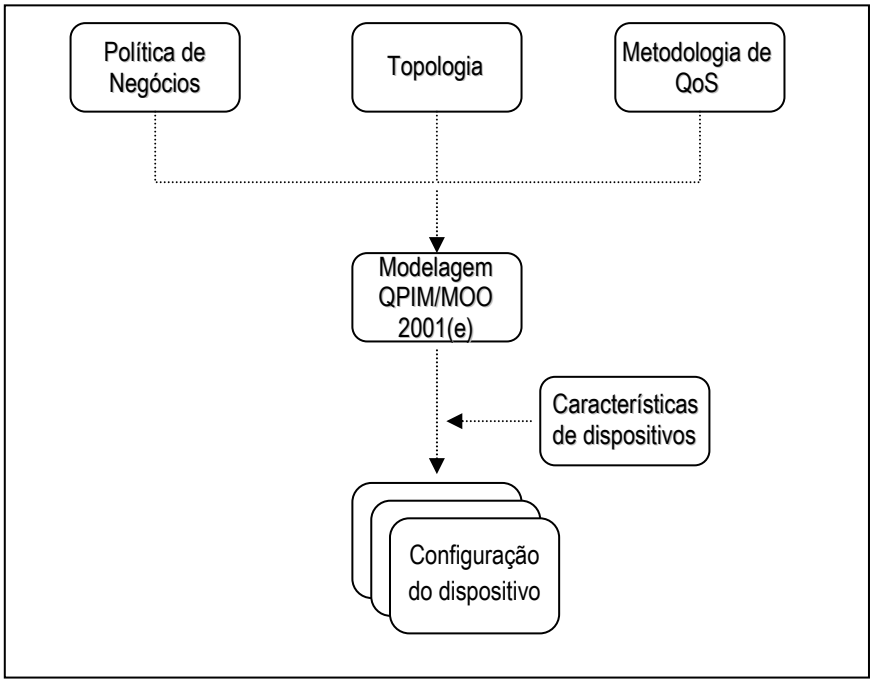

FIGURA 4.1 - Fluxo de Informações das Definições de QoS

Há várias ramificações na forma como se especifica uma política. Em primeiro lugar, o QPIM utiliza regras para definir uma política [MOO 2001], [MOO 2002]. Após isso, o QPIM utiliza extensivamente organizações hierárquicas e informações de políticas. Além disso, o QPIM não força a que a especificação da política inclua detalhes de configuração. Essa tarefa fica designada para os PDPs.

- Modelagem baseada em Regras

Segundo o *Policy Framework Working Group* [MOO 2001] e [MOO 2002], uma política é melhor descrita através de uma modelagem baseada em regras. Uma regra de política de QoS é estruturada a partir de cláusula de condição e uma cláusula de ação. Caso a condição seja avaliada como verdadeira, um conjunto de ações (especificados na clausula de ação) é executado. Por exemplo, a regra:

"O tráfego WEB deve receber no mínimo 50% da largura de banda disponível ou mais, caso mais esteja disponível"

pode ser formalizado como:

"<If protocol == HTTP> then <minimum BW = 50%>"

onde a primeira cláusula entre os operadores "<" e ">" é a cláusula de condição e a segunda é a cláusula de ação a ser executada.

- Organizar informação hierarquicamente

A organização de informação do QPIM foi projetada para ser hierárquica. Para alcançar esse objetivo, QPIM utiliza o agregado *PolicySetComponent* [MOO 2002] para

fornecer um estrutura hierárquica arbitrária de informações sobre políticas. Um grupo de políticas funciona como se fosse um container de regras ou grupos de políticas. Uma regra de política também pode conter regras ou grupos, o que torna possível uma relação de uma regra com uma sub-regra.

A decisão hierárquica do projeto é baseada na percepção de que é natural para seres humanos organizar regras de políticas em grupos. Quebrar uma política complexa em um conjunto de regras simples é um processo que segue a maneira com que as pessoas tendem a pensar e a analisar sistemas. A abstração de complexidade torna possível que uma política complexa seja desdobrada em uma hierarquia de regras simples e um agrupamento de regras simples.

- Definição de políticas orientadas a objetivo

QPIM facilita a definição orientada a objetivo de uma política de QoS. Isto significa que o processo de definir a política de QoS está focalizado no efeito desejado das políticas, ao contrário dos meios de executar a política em dispositivos da rede.

QPIM foi projetado para definir uma especificação mínima do comportamento desejado da rede. É função de agentes específicos de configuração de cada dispositivo interpretar a política expressada em uma maneira padrão e preencher os detalhes necessários de configuração que são requeridos para cada aplicação particular. O benefício de usar QPIM é que fornece uma linguagem comum que cada um dos agentes específicos de dispositivo e/ou de configuração possam utilizar. Isso ajuda a assegurar uma interpretação comum da política em geral também como auxilia o administrador a especificar uma política comum a ser executada em dispositivos diferentes. Isto é análogo ao paradigma fundamental de orientação a objetos de separar a especificação da implementação. Ao utilizar QPIM, o condicionamento de tráfego pode ser especificado de uma forma geral que possa auxiliar a diferentes implementações a satisfazerem um objetivo comum.

- Modelo de domínio de uma política

Um objetivo importante do QPIM é fornecer meios para definir políticas de modo a que elas se estendam a vários dispositivos. Este objetivo diferencia QPIM dos modelos da informação a nível de dispositivo, que são projetados modelando a política que controla um único dispositivo, seus mecanismos e potencialidades.

Primeiramente, papéis [MOO 2001] são utilizados para definir políticas através de múltiplos dispositivos. Em segundo lugar, o uso de políticas abstratas livra o processo da definição de política de ter que tratar das peculiaridades de cada dispositivo e deixa a interpretação e a configuração a serem modeladas por PDPs ou por outros agentes de configuração. Em terceiro lugar, QPIM permite o reuso extensivo de todos os blocos de construção de políticas em múltiplas regras utilizadas dentro de diferentes dispositivos.

- Modelagem de políticas de QoS independente de modelo ou fabricante

QPIM modela política de QoS de uma forma projetada a ser independente de todo o dispositivo ou fabricante em particular. Isso permite que redes compostas por diferentes dispositivos com características diferentes possam ser gerenciadas e controladas usando um único conjunto de políticas padrão. Usar um conjunto único de

políticas é importante porque, de outra forma, a política refletirá as diferenças entre as implementações de cada dispositivo.

- Uso de papéis para mapear políticas em dispositivos de rede

A política que está sendo mapeada funciona como se fosse um elemento da rede. Essa é uma forma simples e compacta de se representar e, até mesmo, de armazenar a definição de uma política. Uma única política abstrata pode ser mapeada para um conjunto de elementos da rede sem que se precise entrar em detalhes específicos de cada dispositivo ou fabricante.

A definição de uma política foi feita para que se possam agregar dispositivos em um mesmo papel. Por exemplo, se duas interfaces do núcleo de uma rede operarem em taxas diferentes, não precisa se definirem duas regras separadas de política para expressar a mesma política abstrata (alocar 30% da largura de banda disponível na interface para um determinado fluxo).

- Reusabilidade

Objetos reusáveis, como definidos pelo *Policy Framework Working Group* [MOO 2001] e [MOO 2002], são os meios para compartilhar blocos de construção de políticas, permitindo assim a gerência central de conceitos globais. QPIM fornece a habilidade de reuso a todos os blocos de construção de políticas: variáveis e valores, condições e ações, perfis do tráfego, e grupos de política e regras de política. Isso fornece a flexibilidade requerida para gerenciar grandes conjuntos de regras de políticas em grandes domínios de políticas.

Por exemplo, a seguinte regra emprega os objetos anteriormente definidos sendo reutilizados (referenciados):

"If <DestinationAddress == FinanceSubNet> then <DSCP = MissionCritical>"

Nesta regra, a condição faz referência a um objeto nomeado *FinanceSubNet*, que é um valor ou, possivelmente, um conjunto de valores definidos e mantidos em um container de objetos reusáveis. A ação de QoS usa um valor denominado *MissionCritical*, que é também um objeto reutilizável. A vantagem de especificar uma política dessa forma é a flexibilidade de herança.

- Política aplicável

Uma política definida pelo QPIM deve ser aplicável na prática, significando que a sua aplicabilidade em dispositivos já deve ter sido testada por um PDP. Apesar de ser independente de dispositivo e fabricante, há três condições que devem ser satisfeitas pelo QPIM para que uma política possa ser aplicada:

1. A política especificada pelo QPIM deve ser capaz de ser mapeada para os valores atuais da rede.
2. A política mapeada pelo QPIM deve ser capaz de controlar funções de QoS sem referenciar nenhum fabricante ou dispositivo.
3. A política especificada pelo QPIM deve ser passível de tradução em configuração para elementos da rede.

- QPIM cobre ambos os tipos de QoS: sinalizado e fornecido

Tanto DiffServ [BLA 98] como IntServ [BRA 94] são suportados pelo QPIM. O DiffServ fornece uma maneira de reforçar as políticas que se aplicam a um grande número de dispositivos de uma forma escalar. O QPIM fornece as ações e as condições que controlam a classificação, policiamento e a conformação feita dentro dos limites estabelecidos pelo serviço diferenciado, assim como as ações que controlam o comportamento *per-hop* dentro do núcleo de uma rede com DiffServ.

Os serviços integrados (IntServ), juntamente com o seu protocolo de sinalização, RSVP [BRA 97] que fornece meios para que os nodos fim-a-fim (ou de borda) requisitem QoS da rede, são suportados pelo QPIM, uma vez que o QPIM possui definidas ações para o requisito de QoS através da rede.

4.3 Modelagem de Políticas Abstratas

O objetivo do QPIM é fornecer um modelo de informação de políticas que possam ser transformadas facilmente do modelo humano de comunicação para uma definição conceitual de política que pode ser utilizada na configuração de um dispositivo que vise prover QoS para a rede. Todas as definições anteriores possuem um ponto em comum: partir de um modelo abstrato para um modelo concreto, de uma política para a configuração de dispositivos, passando por vários níveis de tradução muitas vezes.

Quando seres humanos tratam com políticas, estas são relatadas utilizando termos do cotidiano das pessoas, como uma política de negócios, por exemplo:

"Aplicações de recursos humanos devem ter uma QoS melhor do que simples aplicações Web".

Essa sentença pode ser mais bem traduzida como segue:

"Todo o tráfego gerado por/ou aplicações de recursos humanos deve possuir uma probabilidade maior de se comunicar com o seu destino do que o tráfego gerado por pessoas utilizando a rede com aplicações não-críticas".

Embora essa ultima tradução possua um nível de abstração menor do que a primeira, tal tradução ainda não é suficiente para conseguir configurar um dispositivo de rede. Faz-se necessário uma nova tradução para uma linguagem que utilize os "termos da rede". É importante notar que uma próxima etapa de tradução que mapeie essa ultima sentença (dita sentença de negócios) nas configurações especificas para cada dispositivo é função do PEP, uma vez que cada dispositivo pode possuir um comando diferente para realizar a mesma tarefa.

4.4 Hierarquia de Regras

Uma política pode ser descrita como sendo um conjunto de hierarquia de regras que podem ser agrupadas em subconjuntos. Ao se agregar o componente *PolicySetComponent* [MOO 2002] , está-se representando a junção de uma regra de política ou grupo em uma nova regra de política.

A definição de regras de política de forma hierárquica torna a regra mais legível e favorece a reusabilidade. No QPIM, a alocação de banda, ações de políticas e ações para limiares de descartes utilizam esse contexto hierárquico.

4.5 Esquema de Políticas de QoS em um Serviço de Diretórios

Nessa sessão será proposto um esquema de mapeamento de informações de políticas de uma forma que essas informações possam ser armazenadas em um serviço de diretórios que utilize LDAP como protocolo de acesso.

- Mapeamento

Os modelos de informação são, por definição, independentes do tipo de repositório que se utilize. Há vários *drafts* tratando do mapeamento de cada um dos tipos de políticas que são desenvolvidas pelo *Policy Framework Working Group*. Nessa dissertação, estará coberto apenas o mapeamento necessário para a definição das políticas utilizadas pelo Sistema Monitor de QoS.

Para as classes do modelo de informação, o mapeamento é basicamente de um para um: classes do modelo de informação são mapeadas para classes do LDAP e as propriedades do modelo de informação são mapeadas para atributos do LDAP.

O esquema de políticas de QoS, por si só, pode ser insuficiente para representar todos os tipos de QoS e serviços [SNI 2002] uma vez que o propósito da documentação existente (*drafts*) é ser o mais genérica possível. Em geral, pode-se tornar insuficiente em três aspectos: o primeiro é que políticas específicas para aplicações não são especificadas. Nesse caso, uma extensão deve ser definida de modo a permitir a representação das políticas desejadas. As extensões podem ser de duas formas:

- as novas funções podem ser representadas como subclasses de classes previamente definidas;
- podem ser definidos novos atributos para as classes que já existam.

Ambos as formas descritas anteriormente são suficientes para que se formalize classes específicas de forma a modelar, com detalhamento suficiente, uma plataforma de QoS em particular. O segundo aspecto no qual o esquema existente hoje [SNI 2002] pode ser insuficiente para uma modelagem em particular é o fato de o mapeamento definido vir a ser insuficiente para uma determinada implementação de um servidor LDAP, uma vez que pode haver variações dependentes de fabricante. Como as especificações são idealizadas para a maioria dos servidores LDAP, pode haver a necessidade de se fazerem algumas adaptações em algumas funções. O terceiro aspecto é uma variação do segundo, pois nem todo o esquema especificado acaba sendo mapeável em uma implementação específica de LDAP, uma vez que a implementação pode não seguir totalmente o padrão LDAP.

A figura que segue (fig. 4.2) é uma representação da implementação de um esquema de dados, utilizado para armazenar as políticas definidas pelo administrador da rede e já validadas por um consumidor de políticas.

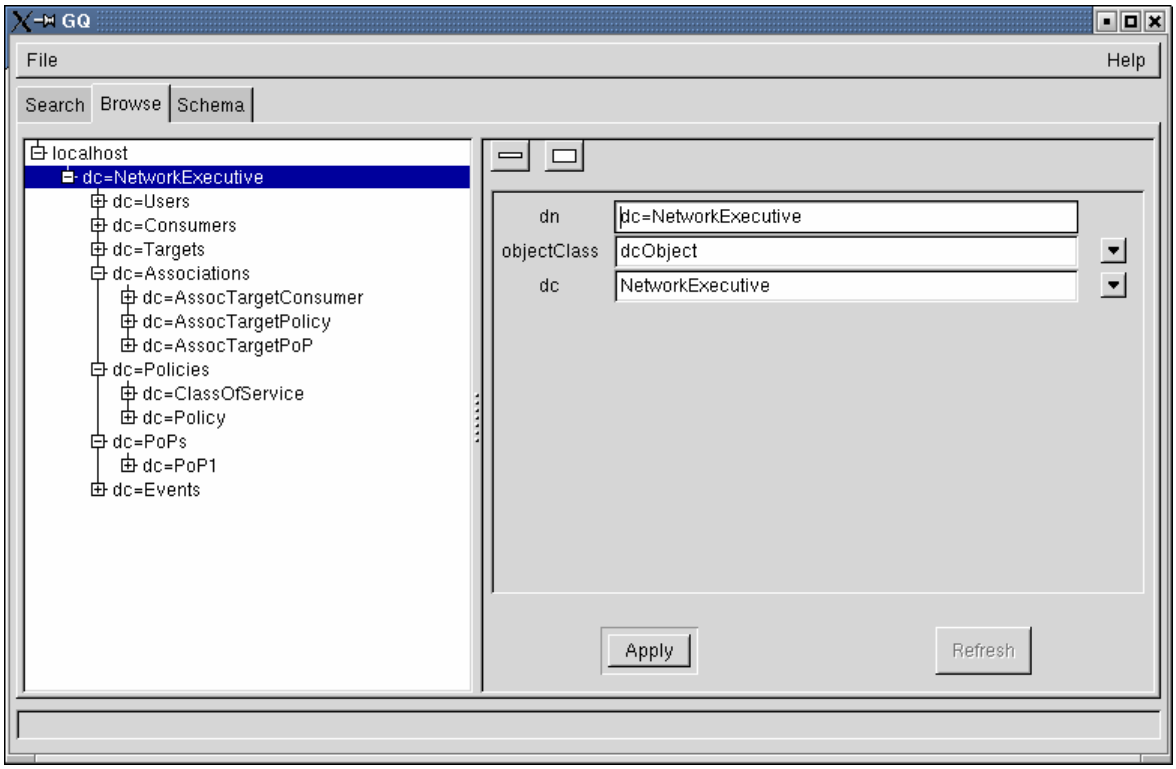

FIGURA 4.2 - Implementação de um esquema de políticas de QoS

A classe de interesse no caso do Controlador de Monitor é a classe *Policy*. Os objetos dentro dessa classe estão representados como descrito por Coelho et al [COE 2002]:

attributeType (1.3.6.1.4.1.12619.1.2.2.14 NAME 'throughput'
DESC 'Network Executive: Throughput'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

attributeType (1.3.6.1.4.1.12619.1.2.2.15 NAME 'delay'
DESC 'Network Executive: Delay'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

attributeType (1.3.6.1.4.1.12619.1.2.2.16 NAME 'jitter'
DESC 'Network Executive: Jitter'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

attributeType (1.3.6.1.4.1.12619.1.2.2.17 NAME 'loss-factor'
DESC 'Network Executive: Loss Factor'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

attributeType (1.3.6.1.4.1.12619.1.2.2.18 NAME 'source-ip'
DESC 'Network Executive: Source IP'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE)

attributeType (1.3.6.1.4.1.12619.1.2.2.19 NAME 'source-port'
DESC 'Network Executive: Source Port'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

attributeType (1.3.6.1.4.1.12619.1.2.2.20 NAME 'target-ip'
DESC 'Network Executive: Target IP'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE)

attributeType (1.3.6.1.4.1.12619.1.2.2.21 NAME 'target-port'
DESC 'Network Executive: Target Port'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

attributeType (1.3.6.1.4.1.12619.1.2.2.22 NAME 'protocol'
DESC 'Network Executive: IP Protocol'
EQUALITY caseIgnoreMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE)

Como descrito anteriormente, o administrador da rede informa a entrada no LDAP para o controlador de monitor, que pode recuperar dessa forma uma política que foi armazenada. Dessa maneira, as políticas ficam isoladas da monitoração de forma a permitir que uma mesma política seja referenciada várias vezes, não ficando confinada a um fluxo ou agregado apenas. A derivação das políticas armazenadas para as variáveis utilizadas no cálculo da QoS pode ser feita como mostra a tabela 4.1:

TABELA 4.1 - Tradução de Políticas Armazenadas no LDAP

Valor LDAP	Significado
Throughput	vazão
Jitter	jitter
Delay	atraso
loss-factor	taxa de perdas
source-ip	IP de origem
target-ip	IP de destino
source-port	porta de origem
target-port	porta de destino
protocol	protocolo

4.6 Definição do Ambiente de Desenvolvimento

É proposto o desenvolvimento de um Sistema para Monitorar QoS cujos agentes sejam executados em um ambiente UNIX. Para tanto, foi escolhido o sistema operacional Linux pela disponibilidade de máquinas e roteadores com esse sistema operacional. A implementação pode ser tanto em Java quanto em C. Por questões de performance e integração com o agente SNMP do Linux (como será descrito adiante), a linguagem C foi escolhida.

Caso um serviço de transmissão de dados necessite de um fluxo contínuo, esse pode e provavelmente passe por mais de uma rede de computadores: não basta que se coletem os dados do lado do cliente e os comparem com os dados enviados.

Como discutido anteriormente, a perda de performance deve ser verificada em toda a "extensão" da transmissão, ou seja, os Monitores de QoS devem estar distribuídos ao longo da transmissão. Essa análise acarreta alguns problemas: é preciso saber quais Monitores de QoS fazem parte de cada conexão monitorada, pois não faz sentido receber dados de Monitores que não fazem parte da conexão avaliada. Tem-se, ainda, que sincronizar o recebimento de mensagens dos Monitores e optar pela freqüência com que serão feitas as amostragens dos valores recebidos. Uma maior freqüência acarreta uma análise mais precisa, porém um número excessivo de mensagens na rede acaba interferindo na performance da rede envolvida.

A fim de simplificar a arquitetura a ser implementada, optou-se por deixar a seleção dos Monitores de QoS relevantes a cargo do administrador da rede.

4.7 MIBS desenvolvidas

Para a comunicação entre os módulos do Sistema Monitor de QoS foi adotado o protocolo SNMP, por ser um protocolo padronizado de gerenciamento, permitindo, assim que tanto o Controlador de Monitor quanto os Monitores de QoS possam ser acessados pelo administrador da rede caso seja necessário. A figura que segue (fig. 4.3) é uma demonstração da MIB de um Monitor de QoS [RIB 2003a]. Vale ressaltar as duas tabelas que formam a MIB:*qmFlowTable*, que é a tabela utilizada para se programar o Monitor de QoS. Através dela, é descrito o fluxo ou o agregado em termos de portas (tanto as de entrada quanto as de saída), os endereços de rede envolvidos, o protocolo de transporte, entre outros. Vale ressaltar que alguns desses termos são apenas restrições, uma vez que se nada for descrito e uma entrada nessa tabela for criada, todos os pacotes que passarem pela interface de rede serão monitorados.

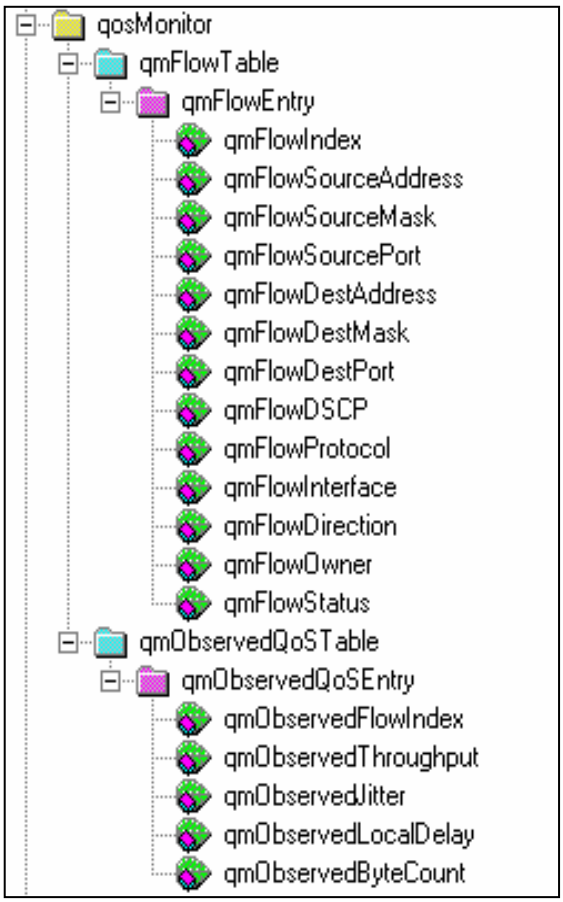

FIGURA 4.3 - MIB do Monitor de QoS.

A segunda tabela *qmObservedOsSTable* serve para informar o resultado da operação do Monitor de QoS.

Alem dessa MIB, ainda é utilizada no Monitor de QoS, a tabela de alarmes da MIB RMON [WAL 2000] (fig. 4.4), que serve para se programar os limiares de QoS que devem ser comparados com os valores de QoS observados pelo Monitor de QoS.

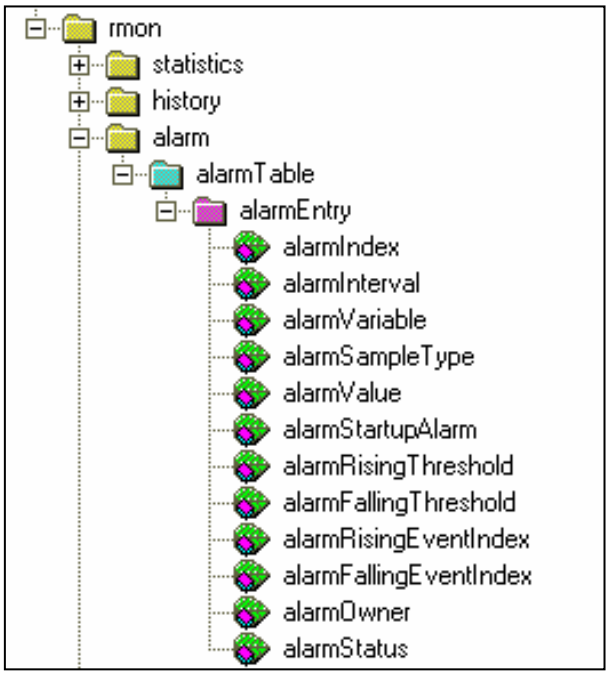

FIGURA 4.4 - Tabela de Alarmes da MIB RMON

No caso de algum valor de QoS exceder ou ficar abaixo de um limiar especificado na tabela de alarmes da MIB RMON, os eventos de notificação do *Probe* Rmon são utilizados e uma *Trap* ou *Inform Request* ou ambos são enviados. A figura 6.3 ilustra o recebimento de um *inform request* definido na MIB RMON. Podemos destacar, nas PDUs do *Inform Request* os valores de alarmes definidos no Monitor de QoS, assim como o índice para a tabela de fluxos, representada na figura anterior (fig. 4.4) pelo objeto *alarmVariable* que contém o objeto monitorado *qmFlowObservedThtoughput.5*, em que 5 é o objeto que representa o fluxo ou agregado monitorado.

No caso específico do exemplo, são mostrados dois *Inform Requests*, cada um representando os dois tipos de alarmes que um *Probe* Rmon é capaz de gerar. Devido ao mecanismo de histerese, um agente *RMON* não pode gerar, num mesmo fluxo ou agregado, o mesmo evento duas vezes seguidas. Se uma *Trap* for enviada porque a vazão em um determinado fluxo ficou abaixo do limite mínimo estabelecido, mesmo que esse fato se repita, uma nova *Trap* não será enviada, a menos que nesse mesmo fluxo seja detectada uma vazão maior que o limiar máximo (e com isso seja enviada uma *Trap* também). Só então, na presença de uma nova medição menor que a vazão mínima estabelecida, uma nova *Trap* pode ser gerada. Em outras palavras, não é possível a ocorrência de duas *Traps* do mesmo tipo para o mesmo fluxo, somente a intercalação de *Traps* (ou *Inform Requests*) como mostra o exemplo na figura 4.5.

Há, ainda uma MIB relacionada ao Controlador de Monitor (fig. 4.6), que serve como ponto de entrada para o sistema monitorar o QoS da rede. Por ter que trabalhar com vários Monitores de QoS, há uma tabela que serve para que sejam cadastrados os Monitores de QoS que estão disponíveis para serem utilizados pelo Controlador de Monitor (*qmMonitorControllerTable*).

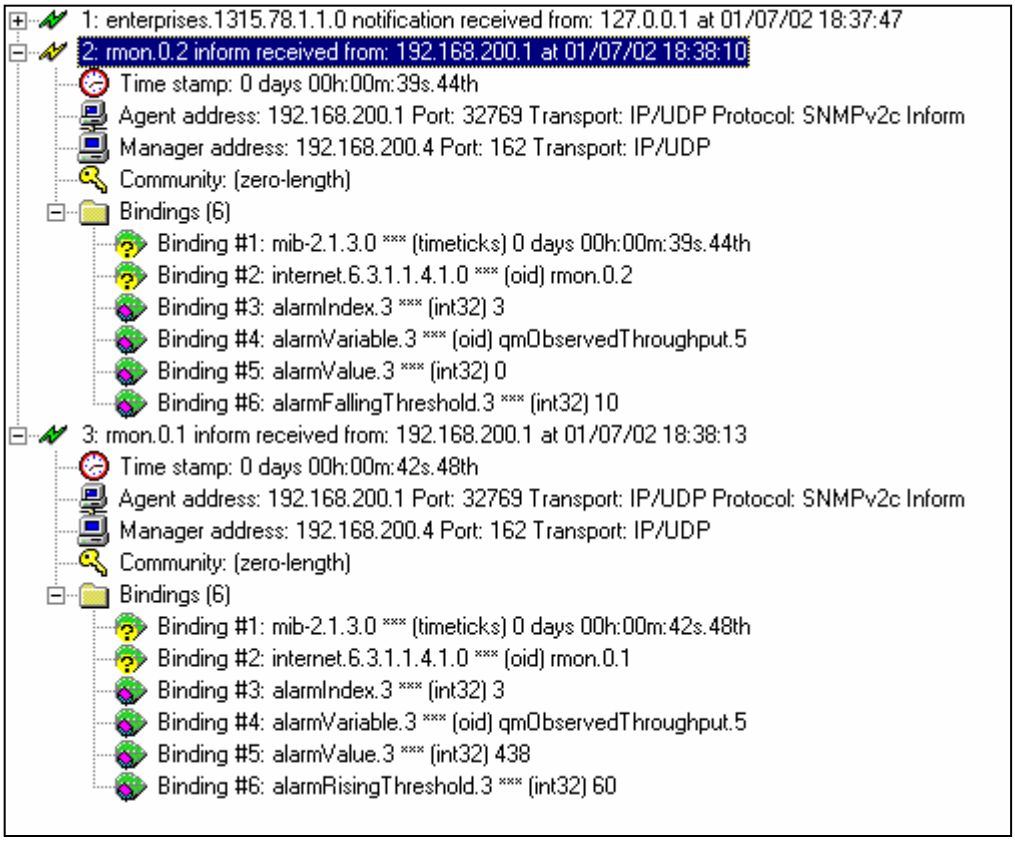

FIGURA 4.5 - Notificação enviada por um Monitor de QoS

O próximo passo é cadastrar uma entrada para o repositório de políticas onde está a política que deverá ser monitorada (*qmMonitorPolicyTable*). Feito isso, resta apenas preencher a próxima tabela que é responsável pela junção de um Monitor de QoS com uma política (*qmQoSAggregateMonitorTable*). Essa tabela, ainda possui uma variável (*qmQOSAggregatePolicyVariation*) que serve para que seja especificada a tolerância nos limiares que serão atribuídos ao agente *RMON* para um agregado em particular. Por exemplo, uma política define uma vazão de 10Kbit/s e o administrador da rede pode definir, para um monitor em particular, que é aceita uma variabilidade de 10% nesse valor, ou seja, um limite inferior de 9Kbit/s e um limite superior de 11Kbit/s. Após essa parte, tem-se início a monitoração no Monitor de QoS selecionado. A última tabela (*qmQoSPairTable*) serve para que sejam especificados, dois a dois os Monitores de QoS com as respectivas monitorações, através das entradas *qmQoSPairSource AggregateNumber*, para indicar o Monitor de QoS de origem e *qmQoSPairDs tAggregateNumber*, para especificar o Monitor de QoS de destino responsáveis por monitorar um determinado segmento do fluxo ou agregado. Para verificar o resultado da monitoração, há as demais variáveis dessa tabela (que são somente leitura).

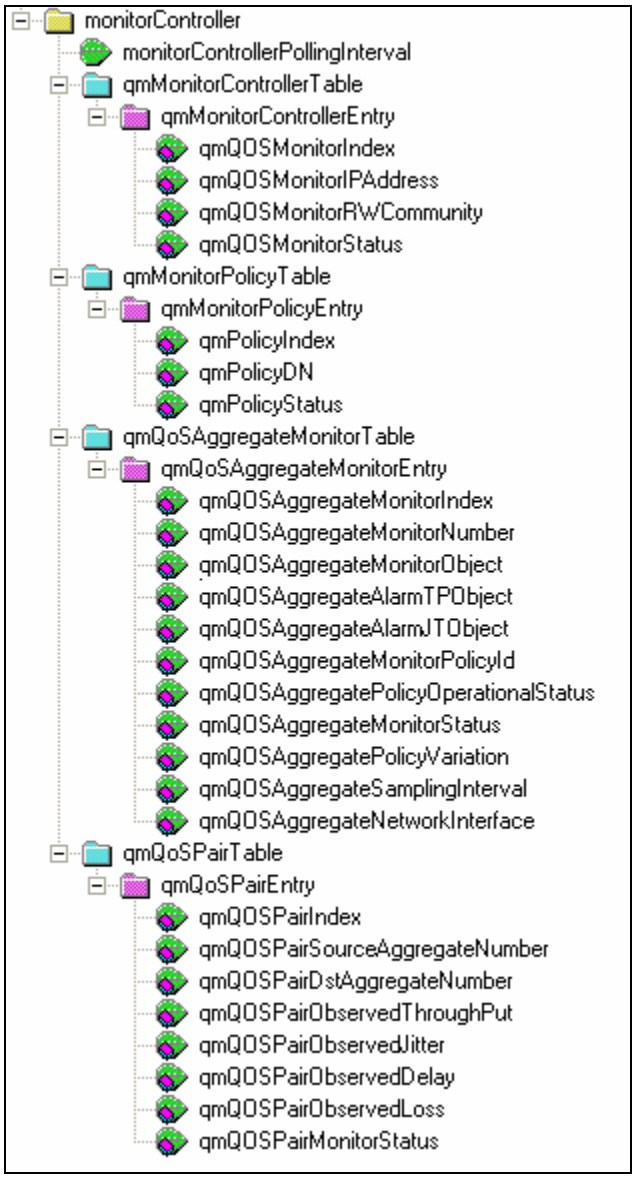

FIGURA 4.6 - MIB do Controlador de Monitor

4.8 Implementação de um Monitor de QoS

Como foi descrito no início desse capítulo, o Monitor de QoS é basicamente um *sniffer* de rede; portanto, deve ser capaz de operar como um. Há algumas formas possíveis de se obter pacotes em uma rede (é considerado que se esteja em uma rede 802.3, *ethernet*). A forma que foi escolhida para se fazer a captura de informações (pacotes) na rede foi a de utilizar uma biblioteca para captura de pacotes conhecida como libpcap [LIB2002]. Uma das facilidades dessa biblioteca é o fato de a estrutura retornada pela biblioteca de captura ser exatamente a mesma que se utiliza para "montar" um pacote, sendo de fácil acesso, se comparado a um *parser* na saída de um *pipe*. A figura 4.7 mostra a função que é chamada a cada vez que um pacote é capturado.

```
....

// Função de callback que é chamada toda a vez que
//um pacote é capturado. Os parâmetros são sempre estes.

void packet_dispatch(u_char *user, struct pcap_pkthdr *header, u_char
*packet)
{
  int i, sport, dport;
  struct iphdr *ip;
  struct tcphdr *tcp;
  struct udphdr *udp;

  // Pula o cabeçalho do frame ethernet e posiciona
  // no cabeçalho IP

  Ip = (struct iphdr*)(packet + 14);

  // Se não for IPv4 cai fora
  if (ip->version != 4)
    return;

  // Se for TCP
  if (ip->protocol == 6)
  {
    tcp = (struct tcphdr*)(ip + 1);
    sport = ntohs(tcp->source);
    dport = ntohs(tcp->dest);
  }
  // Se for UDP
  else if (ip->protocol == 17)
  {
    udp = (struct udphdr*)(ip + 1);
    sport = ntohs(udp->source);
    dport = ntohs(udp->dest);
  }
  // Se nao for TCP nem UDP, cai fora
  else
    return;

....
```

FIGURA 4.7 - Uso da Libpcap

Poder-se-ia fazer uso de algum programa já existente que capturasse os pacotes e apenas tivéssemos o trabalho de analisar a saída desse programa, que seria chamado através de um *pipe*. Essa tentativa realmente foi feita, mas há alguns pontos contra a utilização de um *sniffer* através de *pipes*.

Primeiramente, o uso de *pipes,* por si só, é oneroso para o sistema, uma vez que consiste na chamada de outro processo, aumentando o *overhead* geral o que pode resultar na perda de pacotes. Além disso, há limitações no uso de *pipes* (chamada de função *popen*), na qual a saída do *sniffer* é direcionada para um *stream* de caracteres que, infelizmente, fica indisponível para leitura até que o *pipe* seja fechado (programa *sniffer* termine a execução). Se não bastassem esses problemas a serem solucionados, há ainda o fato de se ter que implementar um *parser* (analisador sintático) na saída do *pipe* que seja tão eficiente a ponto de analisar a saída, descartando todas as informações não úteis e passá-las adiante continuamente. A probabilidade de que o *stream* seja alterado

antes do fim da análise deve ser levada em conta, sem contar a dificuldade de interpretar toda a saída do *pipe*, que pode conter mensagens inesperadas, o que provocaria uma exceção no Monitor de QoS e talvez o fim de sua execução.

O exemplo a seguir ilustra a implementação da funcionalidade de *sniffer* para um agente através de *pipes* (fig. 4.8).

```
...
  read_fp = popen("/usr/projetos/ipgrab-0.8.2/build/ipgrab -b", "r");
  if(read_fp != NULL) {

     chars_read = fread(buffer, sizeof(char), BUFSIZ, read_fp);

         if(chars_read > 0 )

         coleta_dados(buffer);
         printf("Dormindo por 3 segundos....\n");
         sleep(3);

     pclose(read_fp);
   }

  }
  exit(EXIT_FAILURE);
  return(EXIT_SUCCESS);
}
```

FIGURA 4.8 - Uso de *sniffer* através de *pipes*

Além da utilização de *pipes*, que será explicada no próximo parágrafo, o código na figura 4.8 também faz uso de um outro processo que fica monitorando as comunicações e de uma estrutura de memória compartilhada para a comunicação entre esses dois processos.

A parte em destaque é responsável por iniciar um processo, o programa *ipgrab* com a opção *–b*, que é responsável pela análise do fluxo na rede e disponibiliza sua saída num *buffer* chamado *read_fp*. Conforme discutido anteriormente, o maior desafio no uso de *pipes* é fazer como que o sistema consiga monitorar o estado da rede em tempo real, isto é: nesse exemplo em particular, toda a parte do código que inicia no *while(true)* precisaria ser executada com um atraso máximo respeitado; porém, na prática, o processo fica bloqueado na linha *chars_read = fread(buffer, sizeof(char), BUFSIZ, read_fp)*, esperando com que o programa *ipgrab* encerre ou que o *pipe* exceda um limite máximo. É importante notar que para cada leitura de *buffer* é necessária uma chamada à função (não implementada) *coleta_dados(buffer)*, responsável pela análise e interpretação das informações contidas no *buffer*. Essa função não foi implementada pelo fato de ser impossível de se determinar os possíveis conteúdos desse *buffer*, conforme discutido anteriormente.

Devido a essas dificuldades relacionadas, o uso de *pipe*, pode ser usado para fins meramente de testes do restante do código ou para mensurar o tempo de resposta do restante do sistema. A função de *sniffer* pode, também, ser disponibilizada da seguinte forma: utiliza-se um programa para a captura dos pacotes, e a sua saída é redirecionada para um arquivo texto por um período pré-determinado de tempo. De posse desse arquivo, podemos, após uma análise feita manualmente para remover as inconsistências do texto como mensagens de erro ou *debug*, etc, utilizá-lo como se fosse o *sniffer* associado ao agente, eliminando-se, assim, o problema de o *pipe* não nos fornecer um fluxo contínuo para a análise. Infelizmente, esse método também não serve para uma

implementação real, mas serve para que se possa continuar o processo de implementação de um agente sem que se possua um função "real" de *sniffer*.

No momento em que um Monitor de QoS captura um pacote, ele deve verificar, por exemplo, no cabeçalho IP, os endereços de origem e destino de cada pacote, além das respectivas portas de origem e destino que são obtidas nos cabeçalhos TCP e UDP, dependendo do tipo de pacote. Pode-se, ainda, analisar o campo ToS do cabeçalho IP em busca do valor associado à prioridade do pacote. Essas informações são úteis para que o Monitor de QoS possa reconhecer os fluxos de dados que devem ser monitorados. A informação de quais portas e endereços monitorar é passada pelo Controlador de Monitor que, através do administrador da rede, sabe quais os Monitores de QoS devem ser encarregados de monitorar um determinado fluxo.

- Uso adequado de *threads*

O principal problema com o uso de *threads* é a forma de como se irá terminar a *thread*. Por definição, toda a *thread* deve terminar em um *join*, mesmo quando esse não está explícito pelo programador, quando por exemplo, uma *thread* termina a sua execução normalmente. Internamente, o *Kernel* do sistema operacional é responsável por desalocar todas as estruturas de uma *thread* no momento do seu término, e isso é feito quando há um *join*. Para que uma *thread* possa terminar normalmente, é necessário algum mecanismo de sinalização. Graças à natureza de uma *thread*, isso é feito através de variável compartilhada com uma outra *thread.* Dessa forma, através de um *flag*, pode-se sinalizar para uma *thread* terminar a sua execução como mostra a figura 4.9.

```
/*Programa principal*/
vetor_thread[i]->thread_corrente=threadCreate()
....

//Sinaliza para a thread em questão terminar

vetor_thread[i]->terminate=TRUE;


/*Codigo da thread*/
while (!minhaThread->terminate){

  //Faz algo de útil


}
```

FIGURA 4.9 - Terminação adequada de uma *thread*

Esse é o procedimento correto com relação às *threads*, mas há casos em que não se pode sinalizar para que ela termine e o fim da sua execução é necessário, para tanto, ela deve ser destruída por uma outra *thread* através de uma chamada de sistema (fig. 4.10 e 4.11).

Há ainda o problema de desalocar as estruturas utilizadas pela *thread*, o que não irá ocorrer, uma vez que não há *join* implícito nem explícito. Uma chamada de procedimento bloqueante é o exemplo mais freqüente em que é necessário que uma *thread* deve ser terminada por outra de forma sumária (sem sinalização externa para que ela mesma se termine).

```
/*Programa principal*/
vetor_thread[i]->thread_corrente=threadCreate()
....

//Sinaliza para a thread em questão terminar
//porém, ela pode nunca receber esse sinal.

vetor_thread[i]->terminate=TRUE;


/*Codigo da thread*/
while (!minhaThread->terminate){

  mensagem=recebeMensagemDeSocket();


}
```

FIGURA 4.10 - *Thread* bloqueada indefinidamente

Para que isso seja possível, a *thread* deve ser inicializada como *detached*.

```
#include <pthread.h>

void* funcao_da_thread(void* argumento){

mensagem=recebeMensagemDeSocket();
…
}

int main () {

pthread_attr attr;
pthread_t thread;

pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
pthread_create(&thread,&attr,&funcao_da_thread,NULL);
pthread_attr_destroy(&attr);

/*Faz o que tiver que fazer pois não há necessidade de join*/
....
/*cancela a thread através de uma chamada externa*/
pthread_cancel(thread);

return 0;
}
```

FIGURA 4.11 - Cancelamento externo de uma *thread*

A vantagem de se mudar esse atributo de inicialização da *thread* é que agora ela pode ser cancelada externamente sem a necessidade de um *join* externo. A figura 4.11 exemplifica a criação e a destruição de uma *thread detached*.

- Cálculo de vazão e *jitter*

O cálculo de duas variáveis de QoS (vazão e *jitter*) acontece dentro do Monitor de QoS, e a sua monitoração acontece no agente RMON, como poderá ser visto adiante. O cálculo da vazão de um determinado agregado foi implementado através da soma

acumulada do número de bytes que trafegam pelo agregado, divido pelo tempo em que foi feita a amostragem (figura 4.12).

```
//Acumula o número de bytes e remove o cabeçalho ethernet
newFlow->TemporaryTp += (pkthdr->len);
/*Pula o cabecalho ethernet*/
newFlow->TemporaryTp -= sizeof(struct ether_header);
```

FIGURA 4.12 - Cálculo de vazão no protótipo implementado

Para que pudesse ser calculado o *jitter* foi necessário implementar o cálculo do intervalo de tempo entre cada pacote uma vez que o *jitter* é a variação desse intervalo. Para conseguir expressar essa variação temporal foi utilizada uma estrutura do tipo *timeval* para computar esses intervalos de tempo.

Finalmente, para dar suporte SNMP à solução adotada, foi utilizado o agente *UCD-SNMP* [UCD 2003], sendo todas as implementações que necessitem suporte à SNMP uma *thread* desse agente. Foi utilizado o utilitário *mib2c* que é fornecido junto com o agente *UCD-SNMP*, para efetuar a integração.

4.9 Implementação de um agente RMON

A implementação do *probe* RMON é bastante simples, uma vez que apenas um subconjunto do agente é necessário. No protótipo construído foi implementada apenas a tabela de alarmes da MIB RMON. Por residirem ambos no mesmo espaço de execução (o Monitor de QoS e o *probe* RMON) há a possibilidade de internamente um agente poder acessar as estruturas internas de outro através de variáveis globais. Isso é importante para poder se fazer uma melhor validação das entradas na MIB, aceitando, por exemplo, referências de alarme a um fluxo ou agregado que realmente existam no Monitor de QoS.

O agente RMON é basicamente um *loop* que acessa os valores calculados pelo Monitor de QoS (conforme mostrado anteriormente) e compara com os seus próprios valores programados pelo administrador. Após isso, é verificado a necessidade ou não de se emitir uma notificação SNMP. Essa verificação é necessária devido ao controle de histerese inerente a agentes RMON. Depois desse ciclo, essa *thread* dorme pelo tempo estipulado como intervalo entre as verificações. Uma vez que não se esteja mais interessado na monitoração de um determinado agregado, é destruída a instância da tabela de alarmes do RMON (uma instância para vazão e outra para *jitter*) e é destruída a instância para monitoração no Monitor de QoS.

4.10 Implementação de um Controlador de Monitor

Diferentemente de um Monitor de QoS, um Controlador de Monitor não possui um *sniffer* de rede integrado, sendo um problema a menos, uma vez que não se precisa fazer uma ligação com a biblioteca de captura de pacotes. Por outro lado, é o Controlador de Monitor que deve acessar a base de *status*, o LDAP, e ainda se comunicar via SNMP com os Monitores de QoS. Além dessas questões, há ainda a necessidade de se interagir com o administrador da rede de alguma forma, ou via uma aplicação de gerência, ou diretamente pela MIB do Controlador de Monitor. Foi

utilizado o *OpenLDAP* [OPE 2003] como servidor LDAP juntamente com as suas bibliotecas para integração

Do ponto de vista da programação, não há maiores problemas em se implementar esse tipo de funcionalidade, uma vez que é pressuposto o sincronismo entre os relógios dos Monitores de QoS. Há um algoritmo básico que, após receber um sinal em um objeto específico da MIB, deve buscar, junto ao LDAP, a política referenciada na entrada correspondente da MIB e, a partir dela, obter os parâmetros necessários para programar os Monitores de QoS que forem especificados na MIB do Controlador de Monitor. É de responsabilidade do programador verificar que todos os dados necessários, desde a aquisição da política junto ao LDAP até o contato com os Monitores de QoS estejam disponíveis na MIB antes de aceitar um *set* que inicie o processo de monitoração. Pode ser aproveitada a mesma sub-implementação de agente RMON (só é necessária a implementação da tabela de alarmes da MIB RMON).

- Cálculo do atraso e da taxa de perdas

Para se calcular o atraso de um pacote são pressupostos dois requisitos: os Monitores de QoS devem estar sincronizados no tempo, e todas as estações devem conhecer, com antecedência, qual o pacote que vai ser usado para medir o atraso para que ele possa ser reconhecido pelos Monitores de QoS.

Nesta dissertação, não foi encontrada uma forma para se medir o atraso de um pacote, uma vez que introduzir um pacote arbitrário e enviá-lo ao destino apenas mediria o atraso do enlace e não do agregado. A taxa de perdas é calculada através da subtração da vazão entre os pontos que se deseje obtê-la num determinado intervalo de tempo e teria que ser compensada pelo atraso entre os dois pontos (para poderem ser amostrados os mesmos pacotes). Dessa forma, por depender do atraso, a taxa de perdas também não poderá ser calculada, ficando o seu cálculo como objeto de pesquisa para próximos trabalhos.

4.11 Validação da solução

Após a implementação do protótipo, foi realizado um teste com o Monitor de QoS para verificar a eficácia do *sniffer* que foi agregado ao agente SNMP, assim como o mecanismo de cálculo de vazão de um determinado fluxo. Essa seção descreve o procedimento utilizado pra validar o Monitor de QoS cujos detalhes principais de implementação foram discutidos nas seções anteriores.

- Detalhes da rede monitorada

A rede onde se realizou a monitoração foi a rede do Laboratório de Comunicação de Dados do Instituto de Informática da Universidade Federal do Rio Grande do Sul. A figura 4.13 representa a topologia da parte da rede utilizada para testar o Monitor de QoS.

Conforme pode ser visto na figura que segue (fig. 4.12), há a presença de um *sniffer* de rede e de um Monitor de QoS no mesmo segmento de rede. Para gerar o tráfego necessário à medição da vazão do fluxo, foi utilizada uma imagem de teste de uma videoconferência durante dois dias consecutivos (29 e 30 de julho de 2002). O computador inscrito na conferência (via multicast) foi o mesmo computador a executar

um *sniffer* real, que, como já foi explicado, serviu para validar os dados do Monitor de QoS. O programa utilizado foi o *Lan Explorer* [LAN 2002].

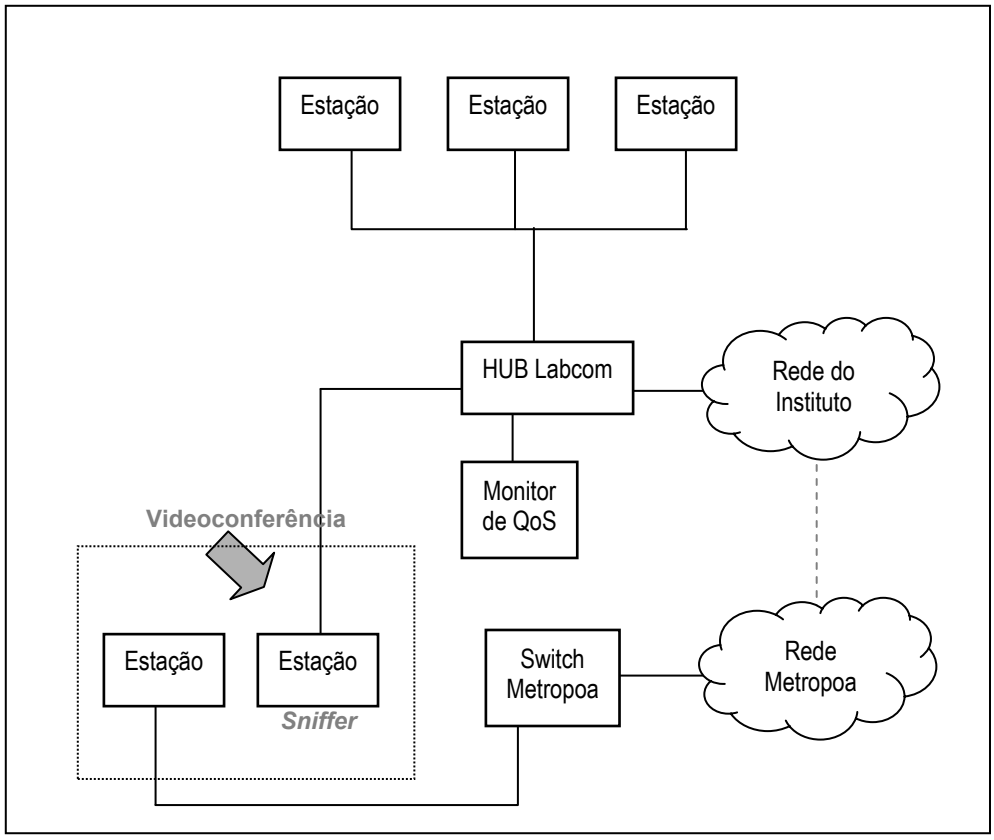

FIGURA 4.13 - Topologia da Rede Utilizada

- Realização do experimento

Como descrito anteriormente, foi utilizada uma imagem de testes para uma videoconferência a fim de gerar o tráfego necessário à monitoração. Durante os períodos de monitoração do tráfego, foi interrompida propositalmente a transmissão para que se pudesse verificar se o "mini" agente RMON implementado estava agindo conforme o estabelecido. Os parâmetros utilizados para a programação do Monitor de QoS e do agente RMON estão nas tabelas 4.2 e 4.3:

TABELA 4.2 - Valores utilizados na programação do "mini" agente RMON

Variável	Valor
alarmVariable	qmObservedThroughput.1
alarmRisingThreshold	32000 (32KB/s)
alarmFallingThreshold	15000 (15KB/s)

Conforme mostra a tabela 4.2 e anteriormente mencionado, apenas alguns elementos da tabela de alarmes da MIB RMON precisam ser preenchidos. No teste realizado foi monitorada a variável *qmObservedThroughput.1* da MIB do Monitor de QoS. Os valores limites foram estabelecidos com base numa monitoração prévia do comportamento da transmissão do fluxo que se desejava monitorar. Uma vez calculado o pico de transmissão, foi colocado um valor que certamente seria superado para testar o

limitante superior do alarme do agente RMON. Para se testar o limitante inferior, basta interromper momentaneamente a transmissão. Esses testes são importantes para que seja validado o mecanismo de histerese do agente RMON.

A tabela seguinte (tab. 4.3) contém os valores que foram atribuídos ao Monitor de QoS para a utilização no processo de validação.

TABELA 4.3 - Valores utilizados na programação do Monitor de QoS

Variável	Valor
qmFlowSourceAddress	200.17.63.183
qmFlowSourceMask	Sem restrição
qmFlowSourcePort	Sem restrição
qmFlowDestAddress	Sem restrição
qmFlowDestMask	Sem restrição
qmFlowDestPort	Sem restrição
qmFlowDSCP	Sem restrição
qmFlowProtocol	UDP
qmFlowInterface	eth0
qmFlowDirection	3 (Sem restrição)

As variáveis dessa tabela (tab. 4.3) caracterizam o fluxo como sendo constituído por pacotes UDP, originados do endereço 200.17.63.183, sendo monitorado na interface "eth0". Esse valor de endereço de origem corresponde ao endereço do transmissor da videoconferência, e os pacotes UDP foram selecionados para que não sejam pegos pacotes outros que não digam respeito ao *stream* de áudio e vídeo. Esses valores foram obtidos através de uma experimentação prévia da videoconferência e da captura de parte dos seus pacotes pelo *software* Lan Explorer [LAN 2002].

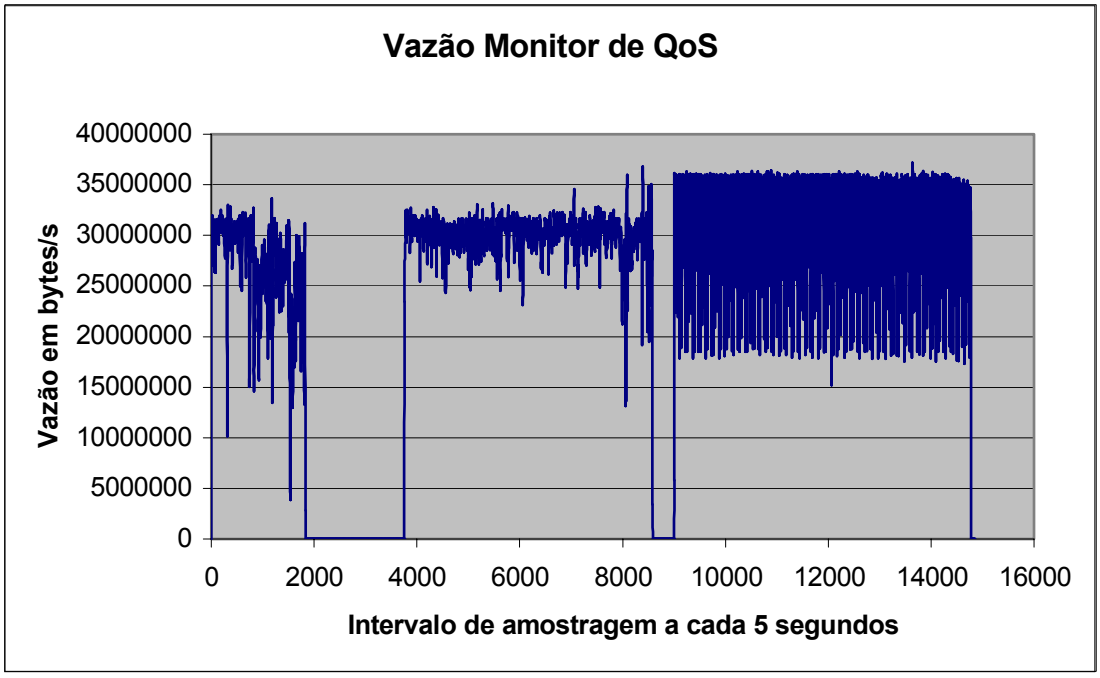

FIGURA 4.14 - Monitoração realizada em 29 de julho de 2002

Os gráficos representam a vazão em bytes/s do fluxo que compõe a videoconferência que estava sendo monitorada. Os primeiros dois gráficos (fig. 4.14 e 4.15) representam a vazão da transmissão detectada pelo Monitor de QoS. O terceiro gráfico (fig. 4.16) mostra a vazão detectada pelo *software* Lan Explorer.

O gráfico da figura 4.14 representa a vazão do fluxo detectada pelo Monitor de QoS com uma taxa de amostragem de cinco segundos, ou seja, a cada cinco segundos o acumulador de bytes do Monitor de QoS (contador) tem o seu conteúdo dividido por cinco e o seu resultado gravado num arquivo que foi utilizado para gerar esse gráfico. Os dados foram amostrados durante quatro horas e doze minutos.

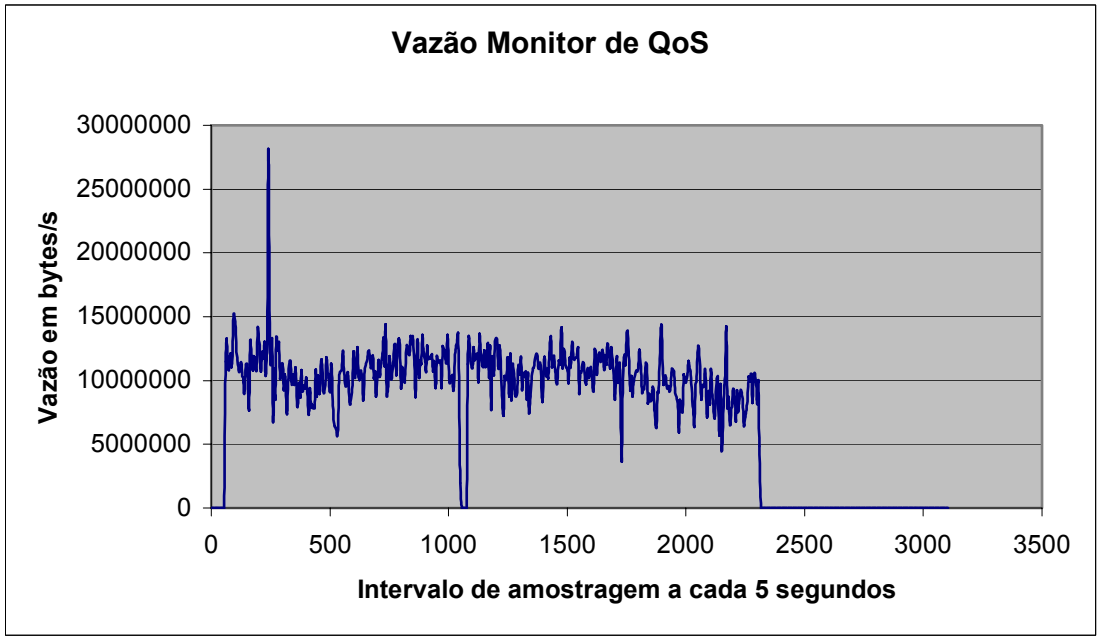

FIGURA 4.15 - Monitoração realizada em 30 de julho de 2002

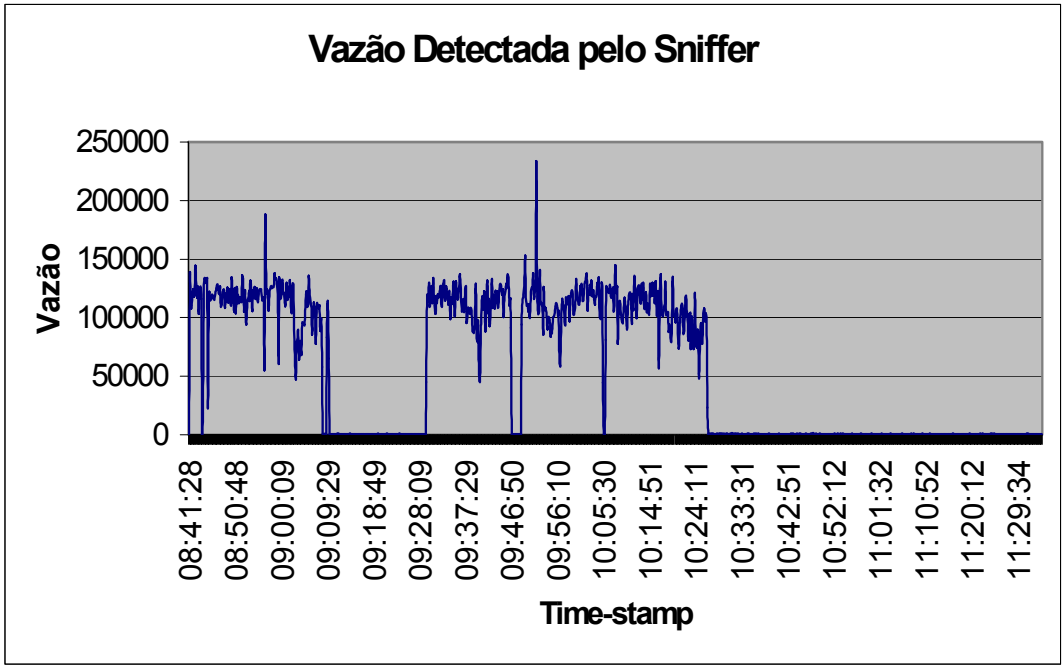

FIGURA 4.16 - Monitoração do *sniffer* em 30 de julho de 2002

Vale enfatizar que a videoconferência monitorada era apenas uma imagem de teste utilizada para que os roteadores e servidores da videoconferência fossem configurados corretamente. Isso pode explicar a diferença na vazão do fluxo durante os dois dias em que foram colhidas amostras do comportamento do tráfego.

O gráfico da figura 4.15 foi obtido através de uma amostra de 52 minutos. O período de amostragem do gráfico da figura 4.15 coincide com o da figura seguinte (fig. 4.16). A diferença reside no fato de que o fluxo foi amostrado durante quase quatro horas pelo Lan Explorer e por apenas 52 minutos pelo Monitor de QoS.

Vale salientar que o período de amostragem do *Lan Exlporer* tem a duração de 10 segundos, o que é responsável pela "suavização" dos picos e dos vales do gráfico da figura 4.16 se comparado com o da figura 4.15; porém, como é possível observar, o comportamento de ambos os gráficos é bastante semelhante, o que já havia sido constatado em experiência anterior e é comprovado agora.

- Interfaces com o usuário

Por utilizar o protocolo SNMP, que é um protocolo padronizado para gerenciamento, há várias formas de se apresentar a informação obtida para o usuário, assim como de se obter dele entradas necessárias à configuração do sistema. Os protótipos do Monitor de QoS e do Controlador de Monitor foram validados via *script*, conforme pode ser observado na figura 4.17.

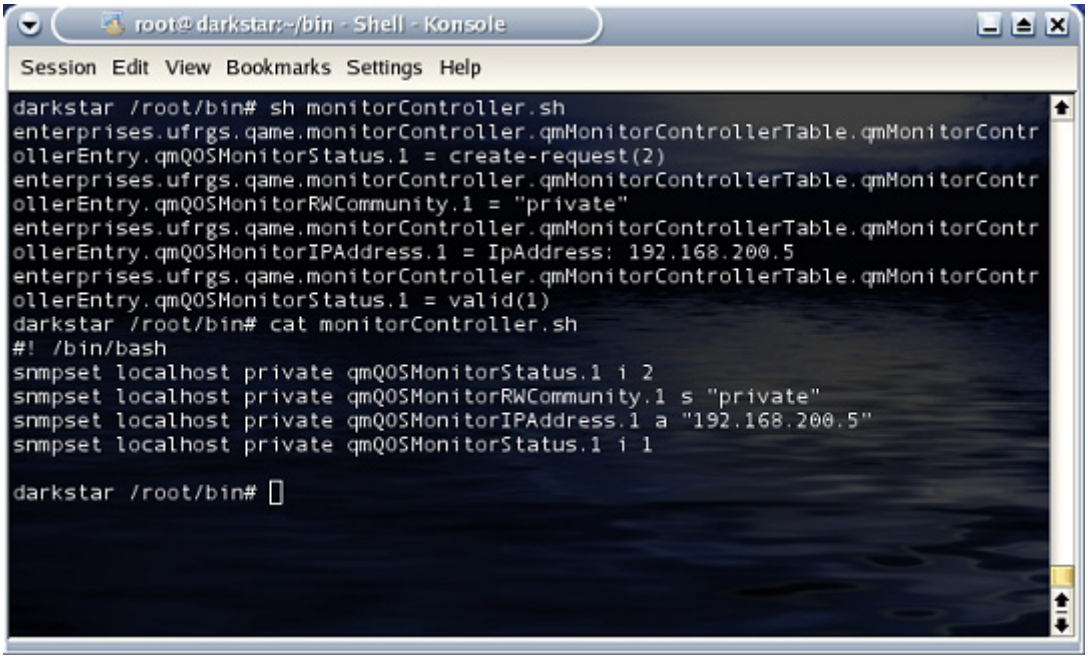

FIGURA 4.17 - Utilização do Controlador de Monitor via script

A configuração e a apresentação dos dados de saída deram-se pela própria tela do terminal, ou via arquivo de *log*. Mas outras formas poderiam ser utilizadas como, por exemplo, uma interface *Web* como mostram as figuras 4.18, 4.19 e 4.20.

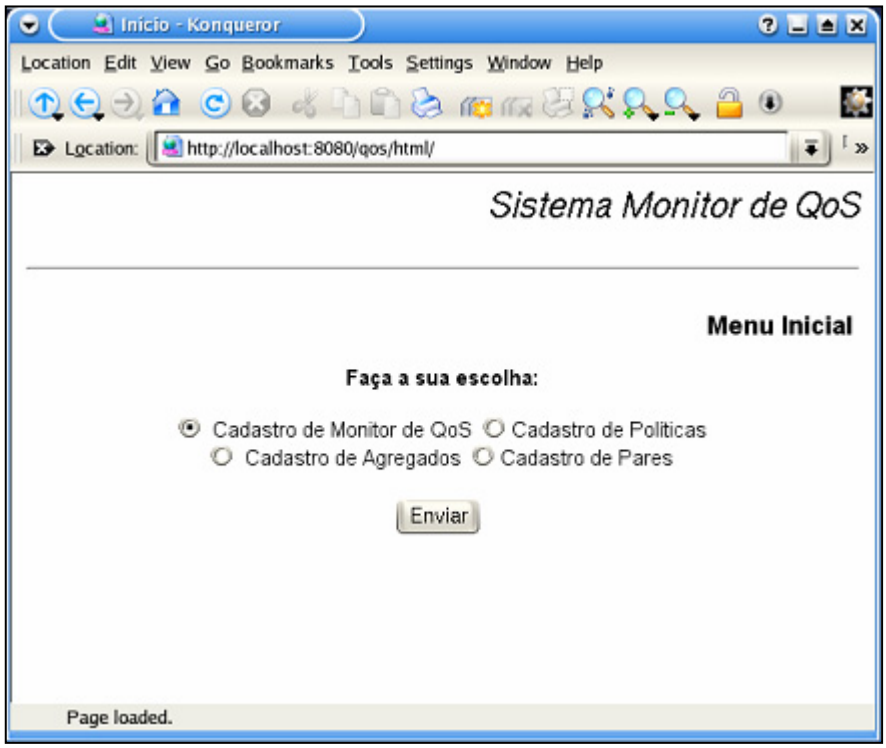

FIGURA 4.18 - Menu inicial do Controlador de Monitor via WEB.

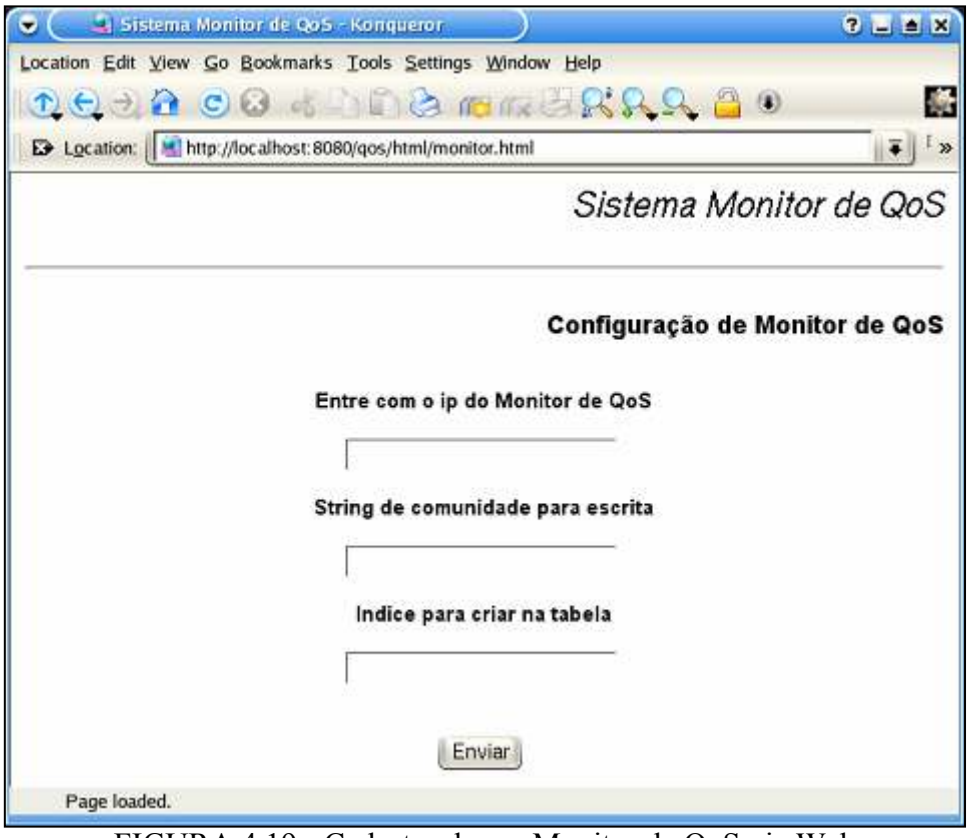

FIGURA 4.19 - Cadastro de um Monitor de QoS via Web

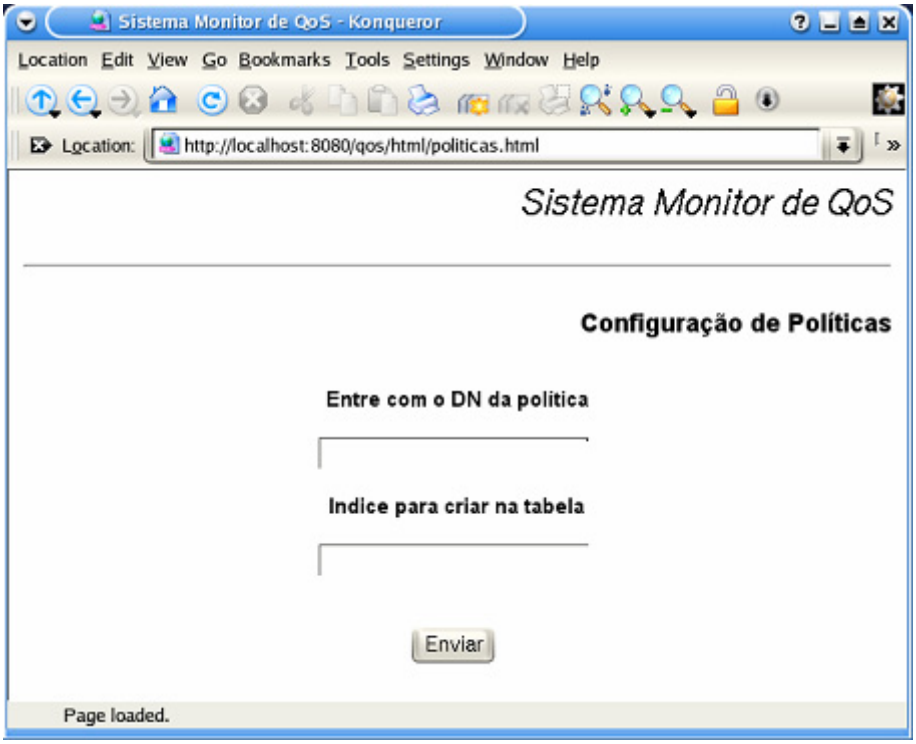

FIGURA 4.20 - Cadastro de uma política de QoS via Web

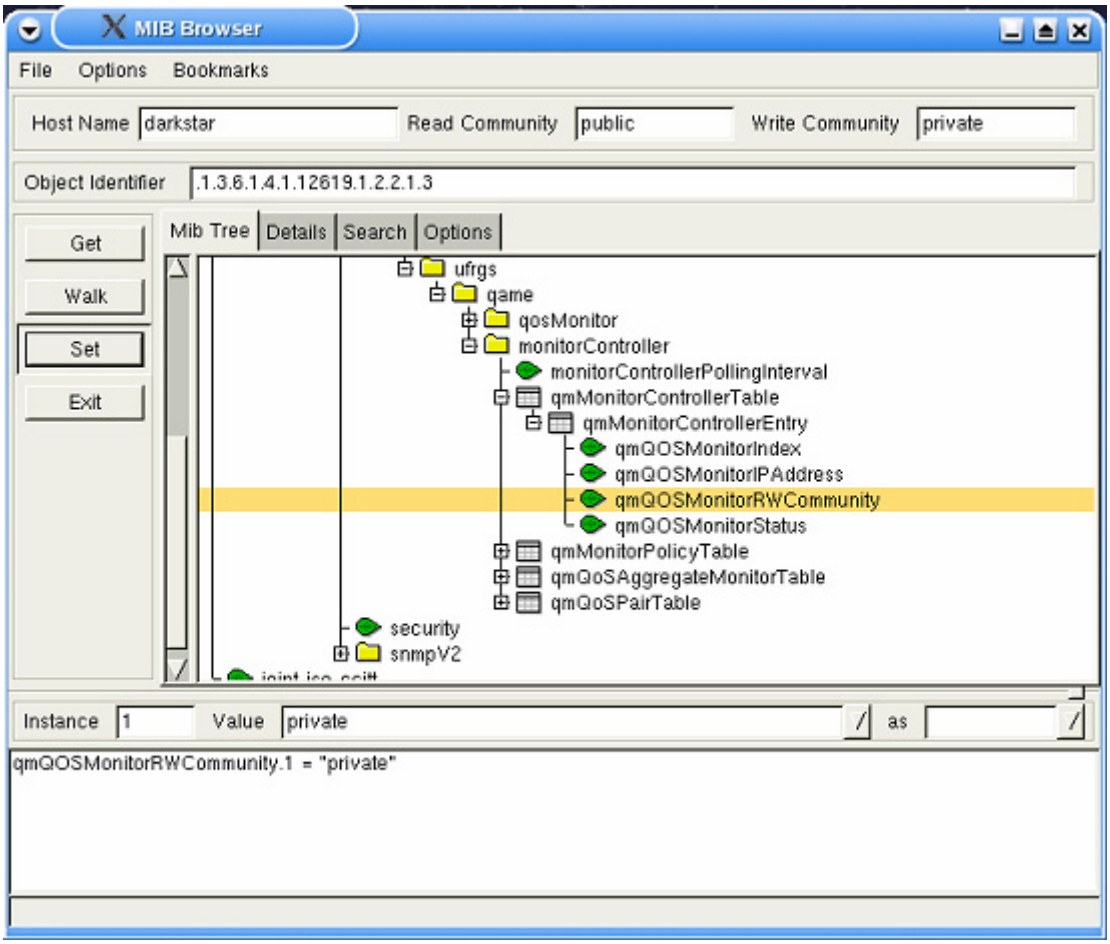

FIGURA 4.21 - Uso de um *MIB Browser* para programar o Controlador de Monitor

As figuras anteriores (4.18 à 4.20) fazem referência à configuração do Controlador de Monitor implementado via Web, assim como a figura 4.17 mostra a mesma configuração via script. Há ainda a possibilidade de se utilizar um *MIB Browser* para efetuar a configuração e visualizar os dados, como mostra a figura 4.21.

Conforme foi salientada anteriormente e pode ser constatada, a principal vantagem do uso de um protocolo padronizado, como o SNMP, na implementação de agentes para o gerenciamento de redes está no grande número de opções, não excludentes, que se dispõe na implementação e no uso do sistema.

5 Conclusões e Trabalhos Futuros

Nesta dissertação foi apresentada uma arquitetura desenvolvida para integrar gerenciamento baseado em políticas com monitoração de redes de computadores com suporte a QoS. A principal contribuição deste trabalho provém do fato de que políticas originalmente definidas para configurar dispositivos de redes podem ser agora utilizadas no processo de monitoração de QoS. A vantagem dessa técnica pode ser verificada de duas perspectivas distintas: da perspectiva da monitoração de QoS, e da perspectiva do gerenciamento baseado em políticas.

Tomando-se como base as soluções atuais para monitoração de QoS, pode-se observar que ao administrador da rede geralmente cabe: (a) definir fluxos de monitoração, (b) definir a QoS esperada, (c) monitorar a rede e descobrir a QoS fornecida, (d) comparar a QoS fornecida com a QoS esperada na tentativa de identificar a degradação da QoS, e, finalmente, (e) proceder com alguma ação no caso de degradações serem detectadas. Todas essas ações são executadas manualmente. Na arquitetura proposta, por outro lado, (a) o administrador da rede usa as definições de fluxos e de QoS esperadas de políticas previamente definidas, (b) o sistema por si só verifica a QoS fornecida e compara com a QoS especificada nas políticas, e (c), em caso de degradação, o sistema notifica o administrador da rede. Nesse novo cenário, duas vantagens principais se salientam: (1) o administrador da rede é liberado para proceder com outras atividades enquanto a QoS fornecida é automaticamente verificada, e (2) a definição de fluxo e QoS esperada é realizada somente uma vez, reduzindo múltiplas definições da mesma informação entre diferentes soluções (nesse caso, entre um sistema baseado em políticas e um sistema monitor de QoS).

Do ponto de vista de gerenciamento baseado em políticas, o sistema introduz a verificação de políticas após a sua aplicação. Entretanto, algumas abordagens de gerenciamento baseado em políticas consideram monitoração da política: a abordagem do IETF não considera. Nesse caso, se as políticas do IETF estiverem sendo usadas, o administrador da rede é forçado a usar sistemas complementares para manualmente monitorar as políticas. Entretanto, com o sistema proposto, a monitoração de políticas é implementada através do processo de monitoração de QoS. Também significa que outra contribuição deste trabalho se deve ao fato de ele complementar a abordagem do IETF introduzindo a monitoração de políticas, a qual é uma característica ausente na definição original do IETF. Como resultado prático desta dissertação, pode-se constatar a eficácia do protocolo SNMP tanto para a comunicação entre os módulos (Monitores de QoS e Controladores de Monitor de QoS) como para interface com o usuário ou, ainda, como interface com algum *front-end* que possa ser desenvolvido (por exemplo, a interface *Web* que foi utilizada para testar a implementação do protótipo). A medição da QoS, através dos 4 parâmetros considerados (vazão, *jitter,* atraso e taxa de perdas), apresentou algumas restrições: a primeira diz respeito à sincronização dos Monitores de QoS (nesta dissertação considera-se que há sincronismo entre os relógios das estações que executam os monitores de QoS). Há ainda o problema de não ser possível de se obter o valor do atraso e, conseqüentemente, da taxa de perdas de um pacote pelos motivos anteriormente citados na seção 4.10.

A validação do protótipo implementado foi dividida em 2 partes: a validação do *sniffer* propriamente dita, que foi comparado contra um analisador de pacotes comercial, e a validação da solução, que engloba desde a entrada de uma política no LDAP até a sua monitoração. Um ponto importante observado foi o fato da arquitetura escolhida para a implementação do protótipo (assim como a própria implementação) mostrar-se

estável em um teste de longa duração (vale notar que a estabilidade da ferramenta não era o escopo dessa dissertação) e serviu para classificar como válida a escolha das bibliotecas e ferramentas utilizadas na implementação da arquitetura.

Como trabalho futuro a ser desenvolvido, inclui-se a melhora da interface gráfica com o usuário. Atualmente, é utilizada uma interface com o usuário baseada em linha de comando ou uma interface *Web* primitiva para controlar os elementos do sistema.. Outra melhoria que deve ser programada é uma implementação de algoritmos para o gerenciamento pró-ativo dentro dos controladores de monitores de QoS e dos monitores de QoS. Tais algoritmos podem detectar futuras degradações nas políticas de QoS e notificar o administrador da rede antes de ocorrer a degradação, permitindo ao administrador proativamente regular a rede gerenciada e as suas políticas.

Anexo 1 MIBs Desenvolvidas

A seguir será apresentada a listagem das MIBS desenvolvidas em conjunto com a elaboração dessa dissertação.

```
 UFRGS-MIB DEFINITIONS ::= BEGIN


 IMPORTS
  enterprises
    FROM RFC1155-SMI;

 ufrgs OBJECT IDENTIFIER ::= { enterprises 12619 }

END

QOS-MONITOR-MIB DEFINITIONS ::= BEGIN

IMPORTS
  OBJECT-TYPE  FROM RFC-1212
  ufrgs        FROM UFRGS-MIB
  OwnerString,
  EntryStatus  FROM RMON-MIB;

qame OBJECT IDENTIFIER ::= { ufrgs 1 }

qosMonitor OBJECT IDENTIFIER ::= { qame 1 }

qmFlowTable OBJECT-TYPE
      SYNTAX SEQUENCE OF QmFlowEntry
      ACCESS not-accessible
      STATUS mandatory
    DESCRIPTION
           "A list of definition of flows to be monitored."
      ::= { qosMonitor 1 }

qmFlowEntry OBJECT-TYPE
      SYNTAX QmFlowEntry
      ACCESS not-accessible
      STATUS mandatory
    DESCRIPTION
           "A collection of fields that defines a flow."
      INDEX { qmFlowIndex }
      ::= { qmFlowTable 1 }

QmFlowEntry ::= SEQUENCE {
        qmFlowIndex               INTEGER,
        qmFlowSourceAddress   IpAddress,
        qmFlowSourceMask      IpAddress,
        qmFlowSourcePort      INTEGER,
        qmFlowDestAddress     IpAddress,
        qmFlowDestMask        IpAddress,
        qmFlowDestPort        INTEGER,
      qmFlowDSCP            INTEGER,
        qmFlowProtocol        INTEGER,
      qmFlowInterface       OCTET STRING,
      qmFlowDirection           INTEGER { input(1), output(2), both(3)
default },
      qmFlowOwner           OwnerString,
        qmFlowStatus          EntryStatus
```

```
    }

qmFlowIndex OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
    DESCRIPTION
            "The index of this flow definition."
      ::= { qmFlowEntry 1 }

qmFlowSourceAddress OBJECT-TYPE
      SYNTAX IpAddress
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
            "The IP source address of this flow definition."
      ::= { qmFlowEntry 2 }

qmFlowSourceMask OBJECT-TYPE
      SYNTAX IpAddress
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
            "The IP source maks of this flow definition."
      ::= { qmFlowEntry 3 }

qmFlowSourcePort  OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
            "The transport source port of this flow definition."
      ::= { qmFlowEntry 4 }

qmFlowDestAddress OBJECT-TYPE
      SYNTAX IpAddress
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
            "The IP destination address of this flow definition."
      ::= { qmFlowEntry 5 }

qmFlowDestMask OBJECT-TYPE
      SYNTAX IpAddress
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
            "The IP destination mask of this flow definition."
      ::= { qmFlowEntry 6 }

qmFlowDestPort    OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
            "The transport destination port address of this flow
definition."
      ::= { qmFlowEntry 7 }

qmFlowDSCP  OBJECT-TYPE
      SYNTAX INTEGER
```

```
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
           "The DSCP of this flow definition.

            The contents of this field is compared
            to an user defined mask (in decimal for this MIB)

          Bits 0-2:  Precedence.
      Bit    3:  0 = Normal Delay,      1 = Low Delay.
      Bits   4:  0 = Normal Throughput, 1 = High Throughput.
      Bits   5:  0 = Normal Relibility, 1 = High Relibility.
      Bit  6-7:  Reserved for Future Use.

         0     1     2     3     4     5     6     7
      +-----+-----+-----+-----+-----+-----+-----+-----+
      |                 |     |     |     |     |     |
      |   PRECEDENCE    |  D  |  T  |  R  |  0  |  0  |

      |                 |     |     |     |     |     |
      +-----+-----+-----+-----+-----+-----+-----+-----+

        Precedence

          111 - Network Control
          110 - Internetwork Control
          101 - CRITIC/ECP
          100 - Flash Override
          011 - Flash
          010 - Immediate
          001 - Priority
          000 - Routine"
      ::= { qmFlowEntry 8 }

qmFlowProtocol    OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
           "The transport protocol of this flow definition."
      ::= { qmFlowEntry 9 }

qmFlowInterface OBJECT-TYPE
      SYNTAX OCTET STRING
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
           "The ifIndex where this flow is observed."
      ::= { qmFlowEntry 10 }

qmFlowDirection OBJECT-TYPE
      SYNTAX INTEGER { input(1), output(2), both(3) }
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
           "The direction observed for this flow. If input(1), the
           flow is observed when a packet is received in the interface
           defined in qmFlowInterface. If output(2), the flow is
           observed when a packet is sent through the interface
           defined in qmFlowInterface."
      ::= { qmFlowEntry 11 }
```

```
qmFlowOwner OBJECT-TYPE
      SYNTAX OwnerString
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
           "The network administrator that created this enter. If
           this probe is the owner, qmFlowOwner will indicate
monitor."
      ::= { qmFlowEntry 12 }

qmFlowStatus      OBJECT-TYPE
      SYNTAX EntryStatus
      ACCESS read-write
      STATUS mandatory
    DESCRIPTION
           "The control entry for this flow definition."
      ::= { qmFlowEntry 13 }

qmObservedQoSTable OBJECT-TYPE
      SYNTAX SEQUENCE OF QmObservedQoSEntry
      ACCESS not-accessible
      STATUS mandatory
      ::= { qosMonitor 2 }

qmObservedQoSEntry OBJECT-TYPE
      SYNTAX QmObservedQoSEntry
      ACCESS not-accessible
      STATUS mandatory
      INDEX { qmObservedFlowIndex }
      ::= { qmObservedQoSTable 1 }

QmObservedQoSEntry::= SEQUENCE {
        qmObservedFlowIndex     INTEGER,
        qmObservedThroughput    INTEGER,
        qmObservedJitter        INTEGER,
            qmObservedLocalDelay    INTEGER,
            qmObservedByteCount    INTEGER
    }

qmObservedFlowIndex     OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmObservedQoSEntry 1 }

qmObservedThroughput  OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmObservedQoSEntry 2 }

qmObservedJitter  OBJECT-TYPE
      SYNTAX INTEGER

      ACCESS read-only
      STATUS mandatory
      ::= { qmObservedQoSEntry 3 }

qmObservedLocalDelay    OBJECT-TYPE
      SYNTAX INTEGER
```

```
      ACCESS read-only
      STATUS mandatory
      ::= { qmObservedQoSEntry 4 }

qmObservedByteCount     OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmObservedQoSEntry 5 }

END

QOS-MONITOR-CONTROLLER-MIB DEFINITIONS ::= BEGIN

IMPORTS
  OBJECT-TYPE  FROM RFC-1212
  ufrgs        FROM UFRGS-MIB;

qame OBJECT IDENTIFIER ::= { ufrgs 1 }

monitorController OBJECT IDENTIFIER ::= { qame 2 }

monitorControllerPollingInterval   OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-write
      STATUS mandatory
      ::= { monitorController 1 }


--Nesta tabela se registra cada monitor que o gerente da rede tem
conhecimento


qmMonitorControllerTable OBJECT-TYPE
      SYNTAX SEQUENCE OF QmMonitorControllerEntry
      ACCESS not-accessible
      STATUS mandatory
      ::= { monitorController 2 }

qmMonitorControllerEntry OBJECT-TYPE
      SYNTAX QmMonitorControllerEntry
      ACCESS not-accessible
      STATUS mandatory
      INDEX { qmQOSMonitorIndex }
      ::= { qmMonitorControllerTable 1 }

QmMonitorControllerEntry::= SEQUENCE {
      qmQOSMonitorIndex INTEGER,
      qmQOSMonitorIPAddress IpAddress,
    qmQOSMonitorRWCommunity OCTET STRING,
      qmQOSMonitorStatus      INTEGER { ready(1), create-request(2),
under-creation(3), invalid(4)}

    }

qmQOSMonitorIndex OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmMonitorControllerEntry 1 }
```

```
qmQOSMonitorIPAddress  OBJECT-TYPE
      SYNTAX IpAddress
      ACCESS read-write
      STATUS mandatory
      ::= { qmMonitorControllerEntry 2 }


qmQOSMonitorRWCommunity  OBJECT-TYPE
      SYNTAX OCTET STRING
      ACCESS read-write
      STATUS mandatory
      ::= { qmMonitorControllerEntry 3 }

qmQOSMonitorStatus  OBJECT-TYPE
      SYNTAX INTEGER { valid(1), create-request(2), under-creation(3),
invalid(4)}
      ACCESS read-write
      STATUS mandatory
      ::= { qmMonitorControllerEntry 4 }


--Nesta tabela se registra a politica que se esta querendo monitorar


qmMonitorPolicyTable OBJECT-TYPE
      SYNTAX SEQUENCE OF QmMonitorPolicyEntry
      ACCESS not-accessible
      STATUS mandatory
      ::= { monitorController 3 }

qmMonitorPolicyEntry OBJECT-TYPE
      SYNTAX QmMonitorPolicyEntry
      ACCESS not-accessible
      STATUS mandatory
      INDEX { qmPolicyIndex }
      ::= { qmMonitorPolicyTable 1 }

QmMonitorPolicyEntry::= SEQUENCE {
      qmPolicyIndex     INTEGER,
      qmPolicyDN  OCTET STRING,
      qmPolicyStatus INTEGER { valid(1), create-request(2), under-
creation(3), invalid(4)}

    }

qmPolicyIndex     OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmMonitorPolicyEntry 1 }

qmPolicyDN  OBJECT-TYPE
      SYNTAX OCTET STRING
      ACCESS read-write
      STATUS mandatory
      ::= { qmMonitorPolicyEntry 2 }


qmPolicyStatus   OBJECT-TYPE
      SYNTAX INTEGER { valid(1), create-request(2), under-creation(3),
invalid(4)}
```

```
      ACCESS read-write
      STATUS mandatory
      ::= { qmMonitorPolicyEntry 3 }


--Nesta tabela se faz a ligacao entre os monitores cadastrados na 1a
tabela e as politicas definidas na 2a tabela


qmQoSAggregateMonitorTable OBJECT-TYPE
      SYNTAX SEQUENCE OF QmQoSAggregateMonitorEntry
      ACCESS not-accessible
      STATUS mandatory
      ::= { monitorController 4 }

qmQoSAggregateMonitorEntry OBJECT-TYPE
      SYNTAX QmQoSAggregateMonitorEntry
      ACCESS not-accessible
      STATUS mandatory
      INDEX { qmQOSAggregateMonitorIndex }
      ::= { qmQoSAggregateMonitorTable 1 }

QmQoSAggregateMonitorEntry::= SEQUENCE {
    qmQOSAggregateMonitorIndex     INTEGER,
    qmQOSAggregateMonitorNumber    OBJECT IDENTIFIER,
    qmQOSAggregateMonitorObject INTEGER,
      qmQOSAggregateAlarmTPObject INTEGER,
      qmQOSAggregateAlarmJTObject INTEGER,
      qmQOSAggregateMonitorPolicyId OBJECT IDENTIFIER,
      qmQOSAggregatePolicyOperationalStatus INTEGER { active(1),
inactive(2)},
      qmQOSAggregateMonitorStatus  INTEGER { valid(1), create-
request(2), under-creation(3), invalid(4)},
      qmQOSAggregatePolicyVariation INTEGER,
      qmQOSAggregateSamplingInterval INTEGER,
    qmQOSAggregateNetworkInterface OCTET STRING
    }

qmQOSAggregateMonitorIndex   OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmQoSAggregateMonitorEntry 1 }

qmQOSAggregateMonitorNumber  OBJECT-TYPE
      SYNTAX OBJECT IDENTIFIER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
           "Point to qmMonitorControllerTable entry."
      ::= { qmQoSAggregateMonitorEntry 2 }

qmQOSAggregateMonitorObject OBJECT-TYPE
SYNTAX  INTEGER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
           "Object index in qmMonitorControllerTable."
      ::= { qmQoSAggregateMonitorEntry 3 }

qmQOSAggregateAlarmTPObject OBJECT-TYPE
```

```
SYNTAX  INTEGER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
           "Throughput Object index in qmMonitorControllerTable."
      ::= { qmQoSAggregateMonitorEntry 4 }

qmQOSAggregateAlarmJTObject OBJECT-TYPE
SYNTAX  INTEGER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
           "Jitter Object index in qmMonitorControllerTable."
      ::= { qmQoSAggregateMonitorEntry 5 }

qmQOSAggregateMonitorPolicyId  OBJECT-TYPE
      SYNTAX OBJECT IDENTIFIER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
           "Point to qmMonitorPolicyTable entry."
      ::= { qmQoSAggregateMonitorEntry 6 }


qmQOSAggregatePolicyOperationalStatus   OBJECT-TYPE
      SYNTAX INTEGER { active(1), inactive(2)}
      ACCESS read-only
      STATUS mandatory
      ::= { qmQoSAggregateMonitorEntry 7 }



qmQOSAggregateMonitorStatus   OBJECT-TYPE
      SYNTAX INTEGER { valid(1), create-request(2), under-creation(3),
invalid(4)}
      ACCESS read-write
      STATUS mandatory
      ::= { qmQoSAggregateMonitorEntry 8 }


qmQOSAggregatePolicyVariation   OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
           "Variation acceptance (%) in QoS variables."
      ::= { qmQoSAggregateMonitorEntry 9 }

qmQOSAggregateSamplingInterval   OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
           "Interval in seconds to collect samples of traffic."
      ::= { qmQoSAggregateMonitorEntry 10 }

qmQOSAggregateNetworkInterface  OBJECT-TYPE
      SYNTAX OCTET STRING
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
```

```
            "Network interface to set in promisc. mode"
      ::= { qmQoSAggregateMonitorEntry 11 }

--Tabela com os pares de monitores de QoS

qmQoSPairTable OBJECT-TYPE
      SYNTAX SEQUENCE OF QmQoSPairEntry
      ACCESS not-accessible
      STATUS mandatory
      ::= { monitorController 5 }

qmQoSPairEntry OBJECT-TYPE
      SYNTAX QmQoSPairEntry
      ACCESS not-accessible
      STATUS mandatory
      INDEX { qmQOSPairIndex }
      ::= { qmQoSPairTable 1 }

QmQoSPairEntry::= SEQUENCE {
      qmQOSPairIndex      INTEGER,
      qmQOSPairSourceAggregateNumber OBJECT IDENTIFIER,
      qmQOSPairDstAggregateNumber   OBJECT IDENTIFIER,
      qmQOSPairObservedThroughPut   INTEGER,
      qmQOSPairObservedJitter INTEGER,
      qmQOSPairObservedDelay  INTEGER,
      qmQOSPairObservedLoss   INTEGER,
      qmQOSPairMonitorStatus  INTEGER { valid(1), create-request(2),
under-creation(3), invalid(4)}

    }

qmQOSPairIndex OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmQoSPairEntry 1 }

qmQOSPairSourceAggregateNumber  OBJECT-TYPE
      SYNTAX OBJECT IDENTIFIER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
            "Point to qmMonitorControllerTable entry."
      ::= { qmQoSPairEntry 2 }

qmQOSPairDstAggregateNumber  OBJECT-TYPE
      SYNTAX OBJECT IDENTIFIER
      ACCESS read-write
      STATUS mandatory
      DESCRIPTION
            "Point to qmMonitorControllerTable entry."
      ::= { qmQoSPairEntry 3 }

qmQOSPairObservedThroughPut    OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmQoSPairEntry 4 }


qmQOSPairObservedJitter  OBJECT-TYPE
```

```
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmQoSPairEntry 5 }

qmQOSPairObservedDelay   OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmQoSPairEntry 6 }

qmQOSPairObservedLoss    OBJECT-TYPE
      SYNTAX INTEGER
      ACCESS read-only
      STATUS mandatory
      ::= { qmQoSPairEntry 7 }


qmQOSPairMonitorStatus OBJECT-TYPE
      SYNTAX INTEGER { valid(1), create-request(2), under-creation(3),
invalid(4)}
      ACCESS read-write
      STATUS mandatory
      ::= { qmQoSPairEntry 8 }



END
```

Anexo 2 Artigo Publicado

Neste anexo é apresentado o artigo "An Integrated System for QoS Monitoring of Policy-Based Networks" desenvolvido durante o mestrado. Este artigo foi aprovado para publicação no SBRC 2003 (*Simpósio Brasileiro de Redes de Computadores*). O artigo é a mais recente publicação até a defesa desta dissertação.
Título: An Integrated System for QoS Monitoring of Policy-Based Networks
Evento: SBRC 2003 - Simpósio Brasileiro de Redes de Computadores
URL: http://www.sbrc2003.ufrn.br
Datas: De 19 a 23 de maio de 2003
Local: Natal, Brasil

An Integrated System for QoS Monitoring of Policy-Based Networks

Marcelo Borges Ribeiro, Lisandro Zambenedetti Granville
Maria Janilce Bosquiroli Almeida, Liane Margarida Rockenbach Tarouco

Federal University of Rio Grande do Sul - UFRGS
Institute of Informatics
Av. Bento Gonçalves, 9500 - Bloco IV
Porto Alegre, RS - Brazil

{mribeiro, granville, janilce, liane}@inf.ufrgs.br

Abstract. *Policy-based management and QoS monitoring are both tasks related to the management of modern QoS-enabled network. Although related to each other, these tasks are currently executed in a non integrated fashion. This paper presents an architecture that integrates policy-based management and QoS monitoring through the extension of the original IETF policy-based management architecture. The main advantage of using our proposed approach is that network administrators are freed to execute other tasks, while the QoS-enabled network is still monitored. Another advantage is that we are monitoring policies and verifying if they are respected in the network even after its deployment, which is a new feature absent in the IETF solution.*

1. Introduction

The provisioning of QoS facilities in computer networks is intended to introduce levels of guaranties that are absent, for example, in the IP best-effort approach. Different QoS architectures have different accuracy for the provided services, which leads to a scenario where the expected QoS may not be always properly provided by the underlying architecture. In this context, QoS monitoring [Eder and Nag, 2001] is required to verify the observed QoS and compare it with the expected/contracted QoS.

In a policy-based network, the expected QoS is defined by policies that orchestrate the network behavior. The Internet Engineering Task Force (IETF) has standardized several elements of a Policy-Based Network Management (PBNM) system [Moore et al., 2001], in which policies are defined using high-level languages, stored in policy repositories, deployed with the help of Policy Decision Points (PDPs) and enforced by Policy Enforcement Points (PEPs) [Westerinen et al., 2001]. In the IETF approach, once a policy is successfully deployed, the PBNM system no longer checks the policy behavior, and the QoS defined in the policy is assumed to be provided by the network.

In fact, that is not always true. The underlying QoS provisioning architecture can be unstable or face problems, and the expected QoS may not be achieved. In order to verify QoS degradation in real management environments, today, the network administrator monitors the network and determines the provided QoS. Such QoS is then

manually compared with the QoS defined by the network policies. Finally, if differences are found, the administrator proceeds with actions to lead the underlying architecture to a consistent state.

To date, policy definition and deployment, and QoS monitoring are tasks executed separately by the available solutions [Granville et al., 2001]. Also, as shown before, the verification of the provided QoS against the QoS defined by the network policies is left to the network administrator. In this context, automation of QoS-policy monitoring and integration among the cited tasks are required from the network management point-of-view.

In this paper we propose an approach (and a system) for QoS monitoring, where policy definitions are input data used to check the observed QoS. When a QoS degradation is detected, our monitoring system notifies an SNMP-enabled manager using InformRequest messages. The system's architecture is internally divided in QoS monitors and QoS monitoring controllers. Each QoS monitoring controller controls several QoS monitors that collect data from the network. Such data is compared with the policies translated by the controller, and if degradations are detected the monitoring controller notifies the SNMP-enabled manager. The communication between controllers and monitors is also SNMP-based. The main goal here is to provide a solution that integrates QoS monitoring and PBNM in a unique management environment.

The remaining of this paper is organized as follows. Section 2 reviews QoS monitoring and policy-based management, while Section 3 introduces the proposed QoS monitoring system. The solution's internal architecture and communication are described in Section 4. Finally, Section 5 concludes de paper and outlines future work.

2. Related Work

Several monitoring systems have been proposed and are widely available. Some systems, like NetSaint [Ballard, 2001] and MRTG [Oetiker, 1998], verify services availability using SNMP-based polling, but such systems introduce management messages that consume network resources. Other systems, like ntop [Deri et al., 2001] and NeTraMet [Brownlee, 1997], operate as network sniffers watching the network traffic without introducing management messages. Even in this case, some management messages will exist to provide communication between the monitoring system and the management station. RMON and RMON2 [Waldbusser, 1997] are also examples of such systems.

Although the amount of available monitoring systems is expressive, specific QoS monitoring systems are not. IETF has been working in QoS monitoring-related issues, and its rmonmib working group has proposed the Application Performance Measurement (APM) MIB [Waldbusser, 2001] and the Application Response Time (ART) MIB [Warth and McQuaid, 1999]. The general approach is to extend the RMON MIB to provide information about end-user perceived application performance (APM MIB) and network provided services performance (ART MIB). An SNMP-enabled manager retrieves performance information and checks if the observed performance is consistent with the contracted QoS. Although such MIBs are still IETF drafts (ART MIB is currently an expired draft), some implementations can be already found [Systems, 2001]. Tham, Jiang and Ko [Tham et al., 2000] have developed important work defining QoS

monitoring schemes that are able to retrieve not only flow end-to-end QoS, but also the per hop QoS experienced by a flow within a network. This is possible because several QoS monitors are deployed in diverse network segments. The monitors are coordinated by a higher level entity that collects monitored flows information and calculates the provided QoS in each network segment [Jiang et al., 2000]. From a policy-based perspective, one important drawback about these QoS monitoring systems is the fact that no integration with policy-based systems is provided.

Policy-Based Network Management (PBNM) has been seriously investigated. Universities, standardization bodies and network device providers are running research projects about several PBNM issues, from formal aspects (like definition of policy description languages [Lobo et al., 1999] and mechanisms to detect/solve policy conflicts [Lupu and Sloman, 1999]) to more concrete aspects (like prototype implementation and complete PBNM systems [Coelho et al., 2001]). Several works have been already published in important international management events like NOMS and IM. In 1999, the first whole policy-oriented workshop was conceived (POLICY 99), followed by POLICY 2001 and POLICY 2002.

PBNM systems have several functionalities and among them one can find: policy definition, policy translation and policy deployment verification. Once a policy is defined (ideally using a high level abstract language) a translation element (a Policy Decision Point - PDP) is supposed to translate such policy to device-specific actions, i.e. translations are required to deploy the policy into the network devices. Policy verification is needed because the policy deployment may fail due to lack of resources or device unavailability, among other reasons [Lupu and Sloman, 1999]. If the policy deployment fails, the PDP notifies the management station, but once the deployment succeeds no further policy verification is performed by the PBNM system.

A critical problem with PBNM and QoS monitoring is the lack of integration. To check if a policy fails after its deployment, the network administrator is forced to activate a separated QoS monitoring system to retrieve the real QoS provided by the underlying QoS architecture. Then, a comparing procedure is manually executed by the administrator to check QoS degradations, using as input data the policy definition and the collected QoS information. In this context, we believe that QoS monitoring and PBNM integration is important for the verification of QoS degradation in an automated fashion. The QoS monitoring system presented in this paper is designed to provide an automated QoS verification based on network policies.

3. Policy-Based QoS Monitoring System

According to the IETF work, a typical PBNM system is composed by 4 main elements [Moore et al., 2001]: management application, Policy Repository, Policy Decision Point (PDP) and Policy Enforcement Point (PEP). The management application provides the user interface that allows a network administrator to edit, store and deploy network policies. To deploy a policy, first the policy is edited and stored in the policy repository. The network administrator then determines in which devices (and PEPs) the stored policy is going to be deployed. The PBNM system, on its turn, selects the PDPs to be used in the policy deployment task. Each selected PDP is contacted by the management

application and the policy is transferred. Finally, the PDP translates the policy to device-specific actions, configuring the devices' internal PEPs previously selected by the network administrator. For further details on PBNM elements one can check the IETF work [Moore et al., 2001].

In this work, we extend this standard PBNM architecture to include elements able to provide QoS monitoring facilities in an integrated environment. Figure 1 presents the proposed extended architecture. On the left hand, one can see the standard PBNM elements (policy repository, PDP and PEP) and the common interactions through the dotted lines. On the right hand, the new introduced elements are presented, and the solid lines show the interactions among these elements. Taking figure 1, the next subsections explain the general operations performed by each architecture's elements from a bottom-up perspective.

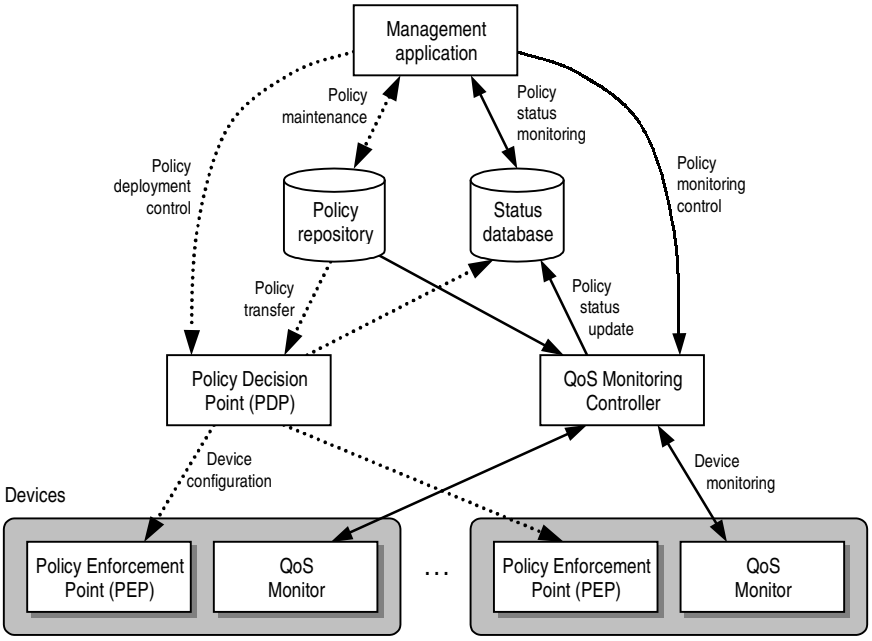

Figure 1: Policy-based QoS monitoring system

3.1. Devices, PEPs and QoS Monitors

In a policy-based network, each device holds PEPs that are internal elements that enforce the policies defined by the network administrator. Examples of PEPs are the queuing disciplines of the interfaces of a router, internal devices' prioritization processes and the marking element in a DiffServ-enabled router. A PEP is configured accessing the device that holds the PEP using some communication protocol. IETF suggests COPS/COPS-PR [Chan et al., 2001] protocol, but SNMP, Telnet, HTTP, among others, can be used as well.

Once a PEP is configured, the PBNM system assumes that the deployed policy is going to be correctly respected, and no further policy check is performed. To allow these further verifications, we use a QoS monitor that performs throughput and jitter checking for defined network flows inside a device. The concept of a QoS monitor was previously introduced by the Jiang, Tham and Ko [Tham et al., 2000] [Jiang et al., 2000]. Here, we slightly change the monitor internal operations (further verified), but the general concept

is the same. The monitor is configured through the definition of flows to be monitored, which are described by: source and destination network addresses, source and destination transport ports, transport protocol and DSCP/ToS [Nichols et al., 1998], if desired. Each flow is monitored in a specific device's interface and observed in an interface's direction: in, out or both. The monitor keeps track of the jitter and throughput associated to each defined flow and notifies external elements if the observed jitter or throughput exceeds some limits defined by the network administrator. The QoS monitor is also responsible for providing flow information to external elements, which is a requirement for flow loss and delay checking.

It is important to notice that neither the PEP nor the QoS monitor are aware of the network policies. PEP and QoS monitor are just configured by external elements that, on their turn, understand the network policies.

3.2. PDPs and QoS Monitoring Controllers

From a device's perspective, the PDPs and QoS monitoring controllers are the external elements that are aware about the network policies. PDPs receive policies from the policy repository and try to deploy them. Pull or push transfer approaches can be used to send a policy to a PDP. In the pull approach the PDP is notified about a new policy stored in the policy repository. The PDP then downloads such policy using some protocol (for example, HTTP, FTP or LDAP). In the push approach, however, the policy is sent by the management application that retrieves the policy in the policy repository and uploads it in the PDP. Although IETF suggests LDAP [Wahl et al., 1997] as the service to store and transfer policies, there is no consensus in a single solution. For example, even the IETF, in the context of the snmpconf working group, is working in another policy transfer mechanism based on SNMP [Waldbusser et al., 2002], while Martinez et al. [Martinez et al., 2002] propose the use of Script MIB to proceed with this transfer.

After receiving a policy the PDP proceeds with verification of possible policy conflicts [Lupu and Sloman, 1999]. If no conflict is detected, the PDP contacts PEPs to configure them in a way to support the policy being deployed. If a problem occurs in the deployment process, the PDP updates the policy status in the status database indicating the deployment problem and that the policy previously transferred can't be applied. In a successful process, however, one can say that PDPs proceed with policy translation, from policy definition to PEP-specific configuration.

The QoS controllers can be seen as the PDPs of the QoS monitoring process. Exactly as PDPs, QoS controllers access policies from the policy repository (through the pull or push approach using some transfer protocol), they check policy problems through device monitoring and notify external elements when policy degradations are observed. One important difference between PDPs and QoS monitoring controllers is that QoS monitoring controllers verify policies after the policy deployment, while PDPs verify policies during their deployment. The policy verification performed by the QoS monitoring controller requires some QoS monitoring controller/monitor interaction, as follows:

1. The transferred policy is analyzed and its flow definition and QoS parameters are determined. In our research we are dealing with the four main QoS parameters: throughput, delay, loss and jitter;

2. The controller contacts a set of QoS monitors and configure them informing: (a) the flow to be monitored, (b) the expected throughput and jitter associated to the flow and (c) the monitoring thresholds for the expected throughput and jitter.

Once throughput or jitter problems are detected in the monitor, a notification is sent to the monitoring controller. This notification triggers a status database update performed by the controller. Optionally, the notification can be forwarded to the management application, that can handle the detected problem. In order to verify delay and loss problems, the network controller analyzes collected data in the monitors. The experienced delay and loss are compered with the max delay and loss defined by the network administrator. If violation is detected, again, the controller updates the status database and optionally notifies the management application. Comparing QoS monitoring controllers and QoS monitors regard QoS parameters verification, the former checks for delay and loss problems, while QoS monitors take care about jitter and throughput. Section 4 provides details about these verifications inside QoS monitoring controllers and monitors.

3.3. Policy Repository, Status Database and Management Application

The policy repository is the storage for network policies. Accessing the policy repository the network administrator is able to define and store new policies, to retrieve and modify previously defined policies, and to remove old policies not used anymore. Once a policy is defined and placed in the repository, it is ready to be used.

As seen before, the policy deployment process may fail. Also, a policy may fail even after its deployment, due to QoS degradation. Thus, policies being used have a policy status that should be accessed by the network administrator to check the policy successfulness. Such policy status is supported by the status database. This database is accessed by either PDPs and QoS monitoring controllers. PDPs updates the policy status when problems are detected in the policy deployment. On the other hand, QoS monitoring controllers update the policy status when QoS degradation is detected by the monitoring system after the policy deployment.

Finally, the top element of our architecture is the management application. It is the central point from where the policy deployment and QoS monitoring processes are controlled. In order to proceed with such control, the management application uses both policy repository and status database, and have direct access to PDPs and QoS monitoring controllers. Next section presents the system's operations, that are executed under the control of the management application.

4. System's Operational Issues

Using the management application, the following steps are typically taken in the policy deployment and monitoring processes.

1. The network administrator defines policies to be used in the network and stores these policies in the policy repository;
2. The administrator selects PEPs where some policy is going to be deployed;

3. The management application, based on the selected PEPs, determines which PDPs have to be used in the policy deployment and transfers to those PDPs the policy definition and a list of PEPs previously selected by the administrator. It means that in the policy deployment process the network administrator is unaware about the PDPs, but the PEPs;
4. To check the evolution of the policy deployment process the administrator accesses the policy status database;
5. Critical policies have to be monitored. The network administrator informs to the management application some critical policies and monitoring thresholds and, based on its previous deployment information, the management application selects QoS monitoring controllers and QoS monitors that verify the critical selected policies;
6. Optionally, the administrator can select further controllers and monitors by hand, or deselect previously selected ones;
7. Then, the management application transfers the critical policies and thresholds to the monitoring system, sending also a list of QoS monitors to the selected QoS monitoring controllers;
8. Finally, the administrator checks if the critical policy is being respected in the network accessing the policy status database, or optionally the monitoring system notifies the administrator directly.

The next subsections present further considerations about the system that allows the execution of the steps above.

4.1. Relevant QoS Monitors Selection

A key aspect of the QoS monitoring system is the selection of relevant QoS monitors. Tham, Jiang and Ko [Tham et al., 2000] proposes the use of monitors that register themselves in a real-time application name server (RTANS) for each perceived flow that passes through the monitors. The management application can determine the relevant monitors of a flow accessing the RTANS. These procedures ease the relevant monitors selection in the management application, but force each monitor to keep track of every single flow perceived. In our approach, on the other hand, the determination of relevant monitors is not so easy, but it allows the selected monitors to store and analyze few specific flows provided by the management application. We use the following scenario to illustrate the monitor selection question.

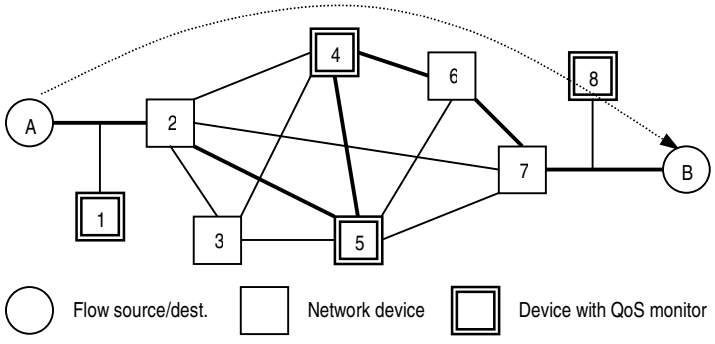

Figure 2: Relevant QoS monitors selection example

Figure 2 presents a policy-based network with some QoS monitors. Host A generates a flow that is sent to host B. This flow is routed through the network devices ♯2, ♯5, ♯4, ♯6 and ♯7. Some devices have an internal QoS monitor (devices ♯4 and ♯5); others don't (devices ♯2, ♯6 and ♯7). There are also devices that do not route the generated flow, but are used in the monitoring process (devices ♯1 and ♯8). Two approaches can be used to select these monitors. The first one consists in providing a user interface that allow the network administrator to explicitly declare the monitors to be used. The second one is more sophisticated and consists in providing automated facilities that investigate the managed network and provides the identification of the QoS monitors bypassed by a flow. For example, the network administrator could inform the flow to be monitored and the management application would proceed with automatic detection of the flow path and its associated monitors. In both cases, a list of QoS monitors to be used is generated, either manually (in the first case) or automatically (in the second case).

If end-to-end QoS is more important than core network QoS, than only monitors ♯1 and ♯8 could be used. If a specific hop QoS is important, one could use monitors ♯4 and ♯5. It is important to notice that in some cases it is not possible to retrieve the QoS of a segment. For instance, it is not possible to retrieve the QoS between devices ♯6 and ♯7. It means that core network QoS monitoring depends on the QoS monitors' location and availability.

In order to start the monitoring process, the next step is to transfer the policy to be monitored to the monitoring system. Since the QoS monitors do not understand network policies, QoS monitoring controllers have to be used. Although QoS monitor selection includes an interaction with the network administrator, QoS monitoring controller selection don't.

4.2. QoS Monitoring Controllers Selection and Operations

QoS monitoring controllers selection is totally coordinated by the management application. Virtually, no network administrator interaction is required. The algorithm for controllers selection is quite simple: find a set of controllers that are able to access as much as possible selected QoS monitors and are also able to be notified by such monitors.

With this assumption, we can conclude that a single flow could be monitored using a single controller, but a single controller can control the monitoring of several flows using several different monitors. Moreover, a single QoS monitor can be controlled by different controllers in the monitoring process of different flows that bypass the QoS monitor.

After the selection of the QoS monitoring controllers to coordinate the monitoring of a flow, the management application is ready to start the monitoring process. In order to proceed with that, the management application contacts the selected controllers and sends (using push or pull approaches discussed in the previous sections):

- The policy to be monitored;
- The list of selected QoS monitors;
- The thresholds for the QoS parameters.

Once these informations are available in a selected QoS controller, the first action executed is the derivation of QoS parameters to be monitored, based on the policy transferred. Then, the controller contacts the QoS monitors from the list and configure

them to monitor throughput and jitter based on the QoS values and thresholds. In order to verify delay and loss problems, the controller also programs the monitors to periodically send back to the controller quantitative information about the observed flow.

4.3. Prototype Implementation

We have implemented a system that uses the architecture proposed in the Section 3. We have used an SNMP-based approach to build the functional elements, which includes the definition of management objects to configure and control the QoS monitors and QoS monitoring controllers, and traps (actually, SNMPv2 InformRequest messages) that implements notification mechanisms used by the elements.

The QoS monitor was coded to work inside a Linux-based router, and uses `libpcap` [McCanne et al., 2002] in order to implement a network sniffer and access network traffic. The monitor is controlled through the QoSMonitor MIB (fig. 3) that allows the definition of the flows to be monitored in the *qmFlowTable*. Table programming follows the RMON approach for control tables programming, which includes, for example, objects for table control (*qmFlowStatus*) and for owner identification (*qmFlowOwner*). Flows throughput and jitter observed by the monitor are stored in the *qmObservedQoSTable*. Each valid row in the *qmFlowTable* corresponds to another row in the *qmObservedQoSTable*.

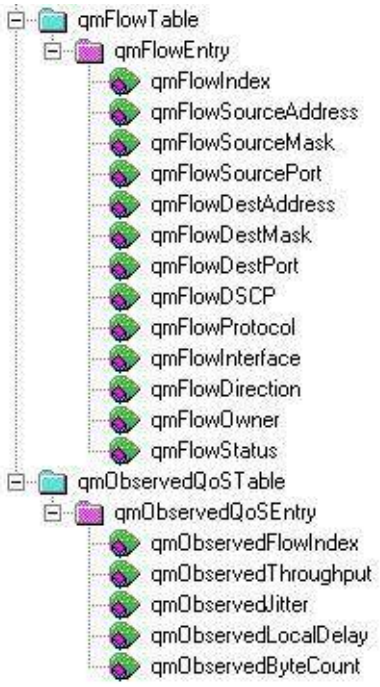

Figure 3: QoSMonitor MIB

We have also implemented the alarm and event groups from the RMON MIB. Such groups allow the automation of the qmObservedQoSTable objects verification and permits the definition of thresholds for jitter and throughput. Once such thresholds are crossed by a flow with QoS degradation, RMON events are sent to external managers (in this case, such managers are the QoS monitoring controllers). Although RMON events are originally implemented through SNMP traps, in our QoS monitor we have used SNMPv2

InformRequest messages due to robustness, since InformRequest can be acknowledge by the receiver, and events retransmission mechanisms can be provided to handle events lost in a congested network.

A MIB related to Monitor Controller has been defined (fig. 4), it serves as entry point to system monitor the QoS of the net. In order to work with several QoS Monitors, there is one table that serves to register those QoS Monitors making them available to be used by Monitor Controller *(qmMonitorControllerTable)*.

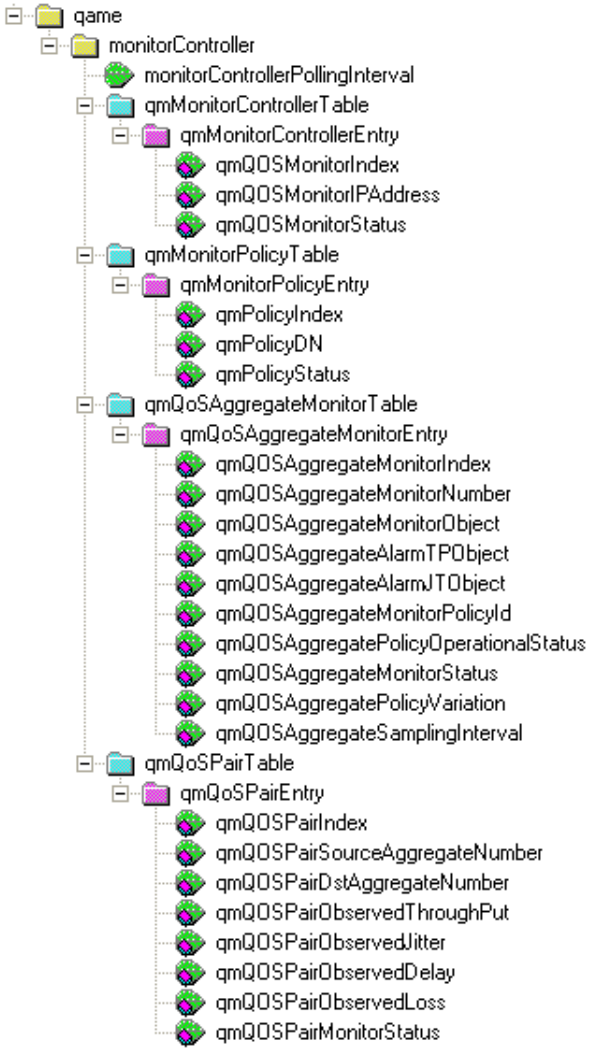

Figure 4: Monitor-Controler MIB

The next step is to register an entrance for the repository of policies where there is the policy that will be monitored *(qmMonitorPolicyTable)*. After that, it only remains to fill the next table that is responsible for the junction of a QoS Monitor with one policy *(qmQoSAggregateMonitorTable)*. This table, still has an variable *(qmQOSAggregatePolicyVariation)* that serves to specify the tolerance in the thresholds that will be attributed to RMON agent for an aggregate in particular. For example, one policy defines an outflow of 10Kbit/s and the Network Manager can define, for a particular QoS Monitor, a variability of 10% in this value is accepted, in other words, an inferior

limit of 9Kbit/s and a superior limit of 11Kbit/s. After this part, the monitoring process can initiate in the selected QoS Monitor. The last table *(qmQoSPairTable)* serves to be specified, two by two, QoS Monitors with these respective aggregates, through the entrances *qmQoSPairSourceAggregateNumber*, to indicate QoS Monitor of origin and *qmQoSPairDstAggregateNumber*, to specify a destination QoS Monitor, both responsible for monitoring one segment of the added flow. In order to verify the monitoring result, there are defined these remaining variables (with read-only purposes).

In order to transfer a policy to a QoS monitoring controller, we have implemented the pull transfer model. A QoSMonitoringController MIB allows the identification of the URL from where the controller can download a policy using LDAP. Since QoS parameters of a flow are defined within a policy, the QoSMonitoringController MIB, differently from the QoSMonitor MIB, does not need to implement objects for the definition of QoS delay or loss rates: this information is already coded in the downloaded policies.

5. Conclusions and Future Work

In this paper we have presented an architecture designed to integrate policy-based management with the monitoring of QoS-enabled networks. The main contribution of this work comes from the fact that policies originally defined to configure network devices can now also be used in the QoS monitoring process. The advantages of this approach can be verified from two different perspectives: from a QoS monitoring perspective, and from a policy-based management perspective.

Checking the current solutions for QoS monitoring, one can verify that the network administrator is often supposed to: (a) define monitoring flows, (b) define expected QoS, (c) monitor the network and discover the provided QoS, (d) compare the provided QoS with the expected QoS trying to identify QoS degradations, and finally, (e) proceed with action if degradations are detected. All these actions are manually executed. With the proposed architecture, on the other hand, (a) the network administrator uses the flows and expected QoS definition from previously defined policies, (b) the system itself checks the provided QoS and compares with the QoS specified in the policies, and (c) in case of degradation, the system notifies the network administrator. In this new scenario two main advantages arise: (1) the network administrator is freed to proceed with other actions while the provided QoS is automatically verified, and (2) the definition of flows and expected QoS is done just once, reducing multiple definition of the same information across different solutions (in this case, across a policy-based system and a QoS monitoring system).

From a policy-based management point-of-view, the system introduces the verification of policies after their deployment. Although some policy-base management approaches consider policy monitoring, the IETF approach does not. In this case, if IETF-based policies are used, the network administrator is forced to use complementary systems to manually monitor policies. However, with the proposed system the monitoring of policies is implemented through the QoS monitoring process. It also means that another contribution of our work is the fact that it compliments the IETF approach introducing the monitoring of policies, which is a feature absent in the original IETF definitions. Future work to be developed includes the improvement of the system's graphical user

interface. Currently, we are using command line oriented interface to control the elements of the system. We started to develop a Web-based interface to enhance the user interaction. Another improvement that should be coded is the implementation of pro-active management algorithms inside QoS monitoring controllers and QoS monitors. Such algorithms can detect future QoS policy degradations and notify the network administrator before the actual degradations occur, allowing the administrator to pro-actively tune the managed network and its policies.

References

Ballard, J. (2001). Netsaint Watches Over Your Network. Network Computing, 11 Jun. 2001.

Brownlee, N. (1997). Traffic Flow Measurement: Experiences with Netramet. RFC 2123, 1997.

Chan, K., Seligson, J., Durham, D., Gai, S., McCloghrie, K., Herzog, S., Reichmeyer, F., Yavatkar, R., and Smith, A. (2001). Cops usage for policy provisioning (COPS-PR). RFC 3084, Mar. 2001.

Coelho, G. A., Granville, L. Z., Almeida, M. J., and Tarouco, L. M. (2001). Network executive: A policy-based network management tool. IEEE Latin America Network Operations and Management Symposium (LANOMS), 2001.

Deri, L., Carbone, R., and Suin, S. (2001). Monitoring networks using nTop. IFIP/IEEE International Symposium on Integrated Network Management (IM), 2001.

Eder, M. and Nag, S. (2001). Service management architectures issues and review. RFC 3052, Jan. 2001.

Granville, L. Z., Ceccon, M. B., Tarouco, L. M., Almeida, M. J., and Carissimi, A. S. (2001). An approach for integrated management of network with quality of service support using qame. IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM), Nancy-France, 2001.

Jiang, Y., Tham, C., and Ko, C. (2000). Challenges and approaches in providing QoS monitoring. Wiley/ACM International Journal of Network Management, 10(6):323-334, Nov./Dec. 2000.

Lobo, J., Bhatia, R., and Naqvi, S. (1999). A policy description language. AAAI, 1999.

Lupu, E. and Sloman, M. (1999). Conflicts in policy-based distributed systems management. IEEE Transactions on Software Engineering, 25(6): 852-869. Nov.1999.

Martinez, P., Brunner, M., Quittek, J., Strauss, F., Schoenwaelder, J., Mertens, S., and Klie, T. (2002). Using the Script MIB for policy-based configuration management. Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), 2002.

McCanne, S., Leres, C., and Jacobson, V. (2002). libpcap. http://www.tcpdump.org, 2002.

Moore, B., Elleson, E., Strassner, J., and Westerinen, A. (2001). Policy core information model - version 1 specification. RFC 3060 Feb. 2001.

Nichols, K., Blake, S., Baker, F., and Black, D. (1998). Definition of the Differentiated Services Field (DS field) in the ipv4 and ipv6 headers. RFC 2474, Dec. 1998.

Oetiker, T. (1998). MRTG, Multi Router Traffic Grapher. Systems Administration Conference (LISA), 1998.

Systems, C. (2001). Cisco catalyst 6000 series network analysis software version 1.2 (new features - application response time mib). Product Bulletin No. 1398, Oct. 2001.

Tham, C., Jiang, Y., and Ko, C. (2000). Monitoring QoS distribution in multimedia networks. Wiley/ACM International Journal of Network Management, 10(2):75-90, March/April 2000.

Wahl, M., Howes, T., and Kill, S. (1997). Lightweight directory access protocol (v3). RFC 2251, Dec. 1997.

Waldbusser, S. (1997). Remote network monitoring management information base version 2 using SMIv2. RFC 2021, Jan. 1997.

Waldbusser, S. (2001). Application performance measurement mib. draft-ietf-rmonmib-apm-mib-06.txt, Feb. 2001.

Waldbusser, S., Saperia, J., and Hongal, T. (2002). Policy based management mib. draft-ietf-snmpconf-pm-10.txt, Feb. 2002.

Warth, A. and McQuaid, J. (1999). Application response time mib. draft-warth-rmon2-artmib-01.txt (expired draf), Oct. 1999.

Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., and Waldbusser, S. (2001). Terminology for policy-based management. RFC 3198 Nov. 2001.

Referências

[ALL 2000a] ALLOT COMMUNICATIONS. **Managing ISP Point of Presence Traffic.** Sept. 2000. Disponível em: <http://www.allot.com>. Acesso em: 2002.

[ALL 2000b] ALLOT COMMUNICATIONS. **Multi-level Traffic Management - Technical Overview.** Sept. 2000. Disponível em: <http://www.allot.com>. Acesso em: 2002.

[BAJ 2001] BAJOREK, C. **Real-time QoS Monitoring For VoIP**. Disponível em: <http://www.computertelephony.com/article/\ CTM20010202S0004>. Acesso em: 2001.

[BLA 98] BLAKE, S.; BLACK, D. **An Architecture for Differentiated Services**: RFC 2475. [S.l.]: Internet Engineering Task Force (IETF), Dec. 1998.

[BRA 94] BRADEN, R.; CLARK, D.; SHENKER, S. **Integrated Services in the Internet Architecture:** an Overview: RFC 1633. [S.l.]: Internet Engineering Task Force (IETF), June 1994.

[BRA 97] BRADEN, R. et al. **Resource ReSerVation Protocol (RSVP) - Functional Specification**: RFC 2205. [S.l.]: Internet Engineering Task Force (IETF), Sept. 1997.

[BRO 99] BROWNLEE, N. **NeTraMet & NeMAC Reference Manual.** Auckland, New Zealand: Information Technology Systems & Services, The University of Auckland, June 1999. Disponível em:\ <http://www2.auckland.ac.nz/net/Accounting/ntmref.pdf>.Acesso em: 2003.

[CHA 2001] CHAN, K. et al. **COPS Usage for Policy Provisioning (COPS-PR)**: RFC 3084. [S.l.]: Internet Engineering Task Force (IETF), Mar. 2001.

[CIS 99] CISCO SYSTEMS, Inc. **NetFlow Services and Applications White Paper.** 1999. Disponível em: <http://www.cisco.com/warp/public/cc/pd/iosw/ioft/\ neflct/tech/napps_wp.pdf>. Acesso em out. 2003.

[CIS 2000] CISCO SYSTEMS, Inc. **IOS NetFlow Technology Data *Sheet.*** 2000. Disponível em:\ <http://www.cisco.com/warp/public/cc/pd/iosw/prodlit/iosnf_ds.pdf>. Acesso em fev. 2003.

[COE 2002] COELHO, G.; GRANVILLE, L.; ALMEIDA, M.; TAROUCO, L. Network Executive. In: CQR, 2002, Okinawa. **Workshop proceedings.** [S.l. : s.n.], 2002.

[CON 99] CONOVER, J. Policy-Based Network Management. **Network Computing.** [S.l.], Nov. 1999. Disponível em: <http://www.networkcomputing.com /1024/1024f1.html>. Acesso em: set. 2002.

[DER 2001] DERI, L.; CARBONE, R; SUIN, S. **Monitoring Networks Using Ntop.** [S.l.]: Netikos S.p.A, Jan. 2001.

[DUR 2000] DURHAM, D. et al. **The COPS (Common Open Policy Service) Protocol**: RFC 2748. [S.l.]: Internet Engineering Task Force (IETF), Jan. 2000.

[GRA 2001] GRANVILLE, L. Z.; TAROUCO, L. QAME - An Environment to Support QoS Management Related Tasks on IP Networks. In: IEEE INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS, ICT, 2001, Bucharest. **Proceedings...** [S.l. : s.n.] **,** 2001.

[JIA 2000a] JIANG, Y.; THAM, C.; KO,C. Providing Quality of Service Monitoring: Challenges and Approaches. In: NOMS,2000. **Proceedings...** [S.l. : s.n.] **,** 2000.

[JIA 2000b] JIANG,Y.; THAM, C.; KO, C. Challenges and approaches in providing QoS monitoring. **International Journal of Network Management**, New York, v.10, n.6, p.323-334, Nov./Dec. 2000.

[LAN 2002] LAN - EXPLORER. Disponível em: <http://www.lan-explorer.co.uk/>. Acesso em: 2002.

[LEV 2001] LEVI, D.; SCHOENWAELDER, J. **Definitions of Managed Objects for the Delegation of Management Scripts**: Internet draft. [S.l.]: June 2001. Disponível em: <draft-ietf-disman-script-mib-v2-04.txt>. Acesso em: 2003.

[LIB 2002] LIBPCAP. Disponível em: <http://www.tcpdump.org>. Acesso em: 2002.

[LUP 99] LUPU, E.; SLOMAN, M. Conflicts in Policy-Based Distributed Systems Management. **IEEE Transactions on Software Engineering**, New York, v.25, n. 6, p. 852-869, Nov. 1999.

[MAR 2002] MARTINEZ, P. et al. Using the Script MIB for Policy-based Configuration Management. In: IEEE/IFIP NOMS, 2002, Italy. **Proceedings...** [S.l. : s.n.] , 2002.

[MIL 91] MILLS, C; HIRSH, D; RUTH, G. **Internet Accounting:** Background: RFC 1272. [S.l.]: Internet Engineering Task Force (IETF), Nov. 1991.

[MOO 2001] MOORE, B. et al. **Policy Core Information Model – Version 1 Specification**: RFC 3060. [S.l.]: Internet Engineering Task Force (IETF), Feb. 2001.

[MOO 2002] MOORE, B. **Policy Core Information Model Extensions**: Internet Draft. May 2002. Disponível em: <draft-ietf-policy-pcim-ext-08.txt>. Acesso em: 2002.

[NIC 98] NICHOLS, K. et al. **Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers**: RFC 2474. [S.l.]: Internet Engineering Task Force (IETF), Dec. 1998.

[OPE 2003] OPENLDAP. Disponível em: <http://www.openldap.org/>. Acesso em: 2003.

[RIB 2001] RIBEIRO, M.; G.; GRANVILLE, L.; ALMEIDA, M.; TAROUCO, L. Distributed Monitoring Over IP Networks. In: CSCI, 2001, Orlando **Proceedings...** [S.l.:s.n.], 2001.

[RIB 2001a] RIBEIRO, M.; G.; GRANVILLE, L.; ALMEIDA, M.; TAROUCO, L. QoS Monitoring System on IP Networks. In: MMNS, 2001, Illinois. **Proceedings...** [S.l.:s.n.], 2001.

[RIB 2002] RIBEIRO, M.; G.; GRANVILLE, L.; ALMEIDA, M.; TAROUCO, L. Distributed QoS Monitoring on IP Networks. In: CQR, 2002, Okinawa. **Workshop proceedings.** [S.l.:s.n.], 2002.

[RIB 2003] RIBEIRO, M.; G.; GRANVILLE, L.; ALMEIDA, M.; TAROUCO, L. An Architecture to Monitor QoS in a Policy-Based Network. In: ICT, 2003, Papeete. **Proceedings...** [S.l.:s.n.], 2003.

[RIB 2003a] RIBEIRO, M.; G.; GRANVILLE, L.; ALMEIDA, M.; TAROUCO, L. An Integrated System for QoS Monitoring of Policy-Based Networks. In: SBRC, 21., 2003, Natal. **Proceedings...** [S.l.:s.n.], 2003.

[SNI 2002] SNIR, Y. et al. **QoS Policy Schema:** Internet Draft. [S.l.]: Feb. 2002. Disponível em: <draft-ietf-policy-qos-schema-16.txt>. Acesso em: 2002.

[SNI 2003] SNIR, Y. et al. **Policy QoS Information Model:** Internet Draft. [S.l.]: May 2003. Disponível em: <draft-ietf-policy-qos-info-model-05.txt>. Acesso em: 2003.

[TEI 98] TEITELBAUM, B.; HANSS, T. **QoS Requirements for Internet2:** Internet Draft. [S.l.]: 1998. Disponível em:\ <http://www.internet2.edu/qos/may98Workshop/\ html/requirements.html>. Acesso em: 2003.

[TEL 2002] TELCHEMY. **Monitoring Voice Quality in Voice Over IP Networks**. Disponível em: <http:// www.telchemy.com/>. Acesso em: 2002.

[THA 2000] THAM, C.; JIANG, Y.; KO, C. Monitoring QoS Distribution in Multimedia Networks. **International Journal of Network Management**, New York, v.10, n.2, p.75-90, Mar./Apr. 2000.

[UCD 2003] UCD-SNMP. Disponível em: <http://net-snmp.sourceforge.net/>. Acesso em: 2003.

[WAL 2000] WALDBUSSER, S. **Remote Network Monitoring Management Information Base:** RFC 2819 [S.l.]: Internet Engineering Task Force (IETF), May 2000.

[WAL 2002] WALDBUSSER, S.; SAPERIA, J.; HONGAL, T. **Policy Based Management MIB:** Internet Draft. [S.l.]: Feb. 2002. Disponível em: **<**draft-ietfsnmpconf-pm-10.txt**>**. Acesso em: Fev. 2002.

[WAH 97] WAHL, M.; HOWES, T.; KILLE, S. **Lightweight Directory Access Protocol (v3):** RFC 2251 [S.l.]: Internet Engineering Task Force (IETF), Dec. 1997.

[WES 2001] WESTERINEN, A. et al. **Terminology for Policy-Based Management:** RFC 3198. [S.l.]: Internet Engineering Task Force (IETF), Nov. 2001.

[WIN 2002] WINPCAP. Disponível em: <http://netgroup-serv.polito.it/winpcap/>. Acesso em: 2002.