Pós-Graduação em Ciência da Computação

"Um Mecanismo de Provisionamento de Tráfego para Serviços Diferenciados"

Por

José Pacífico de Moura Neto

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, Fevereiro/2003

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ PACÍFICO DE MOURA NETO

"Um Mecanismo de Provisionamento de Tráfego para Serviços Diferenciados"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: Judith Kelner

RECIFE, FEVEREIRO / 2003

Agradecimentos

Agradeço a Deus pelo dom da vida e proteção em todos os momentos.

À minha esposa Lalinne, pelo companheirismo e incentivo, que foram decisivos nesta caminhada.

Aos meus pais, pela educação, dedicação e apoio que sempre me deram. E às minhas irmãs, pelo carinho de sempre.

Aos professores e amigos Judith Kelner e Djamel Sadok que com seus conhecimentos me orientaram na elaboração deste trabalho.

A todos do GPRT/UFPE que me auxiliaram, em inúmeros momentos, no desenvolvimento desta dissertação.

Ao Tribunal de Contas do Estado do Piauí, TCE/PI, por acreditar e investir nestes anos de serventia pública, liberando-me temporariamente das atividades funcionais. Este trabalho foi desenvolvido, em parte, com recursos fornecidos pelo Fundo de Modernização do TCE/PI, Lei nº 4.768/95.

Obrigado a todos que direta ou indiretamente colaboraram na elaboração desta dissertação.

Resumo

Podemos verificar que o modelo de melhor esforço não é adequado para atender as necessidades das novas aplicações avançadas na rede. Para melhor disponibilizar estas aplicações é necessário a inclusão de qualidade de serviço como forma de oferecer garantias de desempenho. Algumas arquiteturas vêm sendo desenvolvidas como forma de oferecer garantias de QoS fim a fim, entre elas [1][13][25]. Dentre estas, a Arquitetura *Chameleon* se destaca apresentando um modelo inovador de negociação hierárquica de serviços. Dentre os planos da arquitetura *Chameleon*, faz-se necessário implementar um mecanismo de monitoramento e controle, atuando no plano de operação, com roteadores e tráfego real, avaliando e configurando o provisionamento de tráfego, complementando a arquitetura. Assim, neste trabalho implementamos um mecanismo de provisionamento de tráfego, dentre os modelos que implementam QoS [33]. O modelo de serviços escolhido foi o modelo de serviços diferenciados, definido pela IETF, provendo escalabilidade de diferenciação de serviços na Internet.

Palavras-chave: Qualidade de Serviço, Diferenciação de Serviços, Internet, Controle de Filas, Corretor de Largura de Banda

Abstract

We can verify that the best effort model is not suitable to attend the needs of new advanced network applications. In this manner, to have better availability of these applications, necessary to include Quality of Services as a way to provide warranties of performance. Some kinds of architecture have been developed in order to offer warranties of QoS end to end. Amongst them, the Chameleon Architecture deserves a special attention as it presents an innovator model of hierarchic negotiation of service. Amongst the Chameleon Architecture plans, a need of implementing a monitor and controller mechanism that actuate in the operation plan with router and real traffic, evaluating and configuring the traffic provision to complete the architecture. Thus, we implemented in this work a traffic provision mechanism in the midst of models that implement QoS. The chosen model of services was the model of differentiated services defined by IETF, providing scalability of service differentiation in Internet.

Keywords: *Quality of Service, Differentiated Services, Internet, Queue Control, Bandwidth Broker*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Gráficos

1. Introdução

Neste capítulo, serão apresentadas as considerações inicias do trabalho, uma breve descrição sobre a qualidade de serviço e a estrutura geral da dissertação.

1.1. Considerações Iniciais

Atualmente, todo o tráfego da Internet é provido tão rapidamente quanto possível, isto é, a rede foi projetada para o envio de informações usando o modelo de serviço de melhor esforço sem quaisquer requisitos de qualidade de serviço [19]. O tráfego transmitido é tratado sem nenhuma distinção ao longo do caminho, no interior da rede. Todas as aplicações competem pelos recursos em igualdade, mesmo aqueles que são sensíveis aos requisitos de qualidade. Quando há ocorrência de um congestionamento, os pacotes que chegam vão sendo descartados, independente da origem ou aplicação.

O crescimento considerável da rede mundial, com o aumento do número de usuários e quantidade de tráfego transportado, induz ao aparecimento de novos modelos que ofereçam garantias de qualidade de serviço. Estas possuem mecanismos que permitem a sua adoção de forma mais abrangente. O surgimento de novas aplicações, com diferentes requisitos de qualidade de serviço, sinalizam a necessidade de novos serviços a serem oferecidos na Internet.

Alguns discutem que o advento de novas tecnologias, tornará o oferecimento de largura de banda tão abundante e acessível que qualquer aplicação obterá garantias de qualidade de serviço (QoS) automaticamente. Entretanto, outros, defendem a utilização destes mecanismos argumentando que independentemente da quantidade de largura de banda disponível que uma rede possa oferecer, novas aplicações serão desenvolvidas para consumir toda essa banda.

Em favor disso, toda essa largura de banda disponível, não significa obrigatoriamente garantir baixo atraso, pois se a rede estiver carregada, momentaneamente, o fator determinante e de maior influência será o tempo de espera nas filas congestionadas, caso muitos fluxos a utilizem.

Assim, mesmo com a abundância de largura de banda, há a necessidade de diminuir o atraso dos fluxos para o atendimento dos requisitos de QoS da aplicação. A utilização de mecanismos de provisão de QoS demanda a adoção diferenciada de serviços para os tráfegos oriundos

das aplicações, logo, mecanismos que implementem qualidade de serviço serão ainda necessários. Em [28] é discutida a real necessidade de mecanismos de provisão de QoS na rede.

1.2. Qualidade de Serviço na Internet

Como forma de garantir a oferta de diferentes níveis de serviços na Internet, algumas abordagens têm sido propostas. O *Internet Engineering Task Force* (IETF) desenvolveu uma série de propostas visando enfrentar o problema de oferecer garantias de QoS na rede, destacam-se Serviços Integrados (*Intserv*) [29], Serviços Diferenciados (*Diffserv*) [23], *Multiprotocol Label Switching* (MPLS) [32], Roteamento Baseado em Restrições (QoS *Routing*) [34].

A partir da necessidade de se viabilizar aplicações em tempo real na Internet, foi proposto o modelo de serviços integrados implementado a partir de componentes baseados na extensão e não na modificação do serviço IP. Inclui o serviço de melhor esforço, o serviço garantido, para aplicações intolerantes a variações de atraso e o serviço de carga controlada, para outras aplicações que suportam alguma variação do atraso. O modelo *Intserv* faz uso da premissa de que garantias não podem ser obtidas sem reservas explicitas. Como desvantagens desta proposta citamos a falta de escalabilidade em manter recursos reservados [29].

O modelo de serviços diferenciados é uma proposta utilizando a tentativa de reduzir as desvantagens da garantia de melhor esforço. A idéia principal da proposta D*iffserv* é baseada na agregação de fluxos. A reserva, ao ser feita no início de uma transmissão, deve ser realizada para um conjunto de fluxos relacionados (por exemplo, entre vários fluxos de duas sub-redes). Os pacotes IP são marcados com *tags* originando os pacotes conhecidos como *per-hop behaviors* (PHBs). Essas *tags* são os *Diffserv Code Point* (DSCP) indicados no campo *Type of Service* (ToS) de um pacote IP [23]. O modelo *Diffserv* apresenta dois serviços de encaminhamento nos roteadores: o serviço *Expedited Forwarding* (PHB EF) e o *Assured Forwarding* (PHB AF).

O PHB EF é o mais adequado para encaminhamento de pacotes que fazem partes de fluxos multimídia, obtendo a garantia que os parâmetros de qualidade de serviço negociados durante a comunicação sejam reservados. Não é apropriado para o tráfego em rajadas e proporciona um desperdício de recursos [27].

O PHB AF oferece diferentes níveis de despacho assegurado, provê alta probabilidade da transmissão segura dos pacotes marcados com alta prioridade na rede. A probabilidade dos pacotes ser entregue com segurança e com o respectivo serviço requerido, depende da capacidade da rede em um dado instante. Esta proposta é a mais bem aceita atualmente, devido a sua grande escalabilidade, o bom funcionamento está condicionado ao correto provisionamento dos recursos feito na rede, uma vez que suas garantias são definidas em termos de fluxos agregados, não havendo controle de fluxos individuais [9].

Com o objetivo de organizar como os fluxos atravessam uma rede, evitando uma distribuição desigual de recursos, causados pelos congestionamentos, foi proposto a Engenharia de Tráfego. A engenharia de tráfego é direcionada à otimização de desempenho facilitando a operação eficiente e confiável da rede. Pode ser feita manualmente, ou usando alguma técnica automatizada como MPLS e QoS *Routing*, para descobrir e fixar os caminhos mais adequados a determinados fluxos dentro da rede [32].

Uma das técnicas de descobrimento e fixação de caminhos que utiliza os parâmetros de QoS, é o *Multiprotocol Label Switching* (MPLS). Esta técnica utiliza um rótulo de tamanho fixo a partir do qual o roteador decide por onde enviar os pacotes, é uma padronização de várias implementações existentes nas técnicas de encaminhamento baseado em rótulos (*label switching*) proporcionando um melhor desempenho, criação de caminhos entre roteadores e possibilidade de associar requisitos de QoS em rótulos dos pacotes. [30].

Outro processo de selecionar rotas, é o Roteamento Baseado em Restrições que também se basea nos requisitos de QoS diferentemente da utilização de apenas uma métrica, como no caso do RIP (quantidades de roteadores a ser percorrida), ou OSPF (peso administrativo de cada enlace),

na tentativa de melhorar tanto o serviço recebido como a eficiência global da rede. O QoS *Routing* pode ser utilizado para descobrir as rotas que serão utilizadas por roteadores que implementam *Intserv*, *Diffserv* ou MPLS, por exemplo [31].

A Tabela 01 compara os modelos de serviço de melhor esforço, serviços integrados e serviços diferenciados, quanto à diversidade de fluxos presentes em uma transmissão na rede. No modelo de serviço de melhor esforço, não há distinção de fluxos presentes, nem provê qualquer tipo de isolamento ou garantia de transmissão. No modelo de serviços diferenciados os fluxos individuais são agregados em classes de serviços fornecendo garantias de desempenho a estas classes. No modelo de serviços integrados os fluxos são tratados individualmente, dependendo do estilo de reserva adotado, a partir do isolamento e garantia por sessão utilizado de cada fluxo.

Tabela 01 – Comparação entre modelos de serviço

Modelo	Isolamento	Garantia	Sinalização
Melhor Esforço	não	Não	Não
Serviços Integrados	por sessão	por sessão	por sessão
Serviços Diferenciados	agregado	agregado	Agregado

Em [64] é apresentado uma definição de granulosidade de fluxos onde representa o grau de distinção dos fluxos num domínio. A referência mostra a comparação que o modelo de melhor esforço apresenta granulosidade nula, enquanto que no modelo de serviços integrados a granulosidade é máxima. O modelo de serviços diferenciados apresenta granulosidade de fluxos intermediária entre os dois outros modelos de serviço.

O emprego de requisitos de QoS fim a fim consiste em obter garantias de qualidade de transmissão da origem até o destino de forma a manter os níveis pré-contratados durante a fase de negociação. Devido a essa tendência, o desenvolvimento de arquiteturas que garanta os requisitos de QoS na transmissão fim a fim, apresenta-se como um grande desafio. O

surgimento dessas pesquisas envolvendo mecanismos de alocação de recursos fim a fim na Internet vem, no primeiro momento, atender ao emprego de aplicações distribuídas na rede. Entre estes modelos podemos citar:

Aquila – que tem o objetivo de definir, avaliar e implementar uma arquitetura avançada para QoS na Internet, introduz uma camada de software para controle distribuído e adaptável de recursos. Não oferece soluções para negociação de recursos para serviços fim a fim [1].

Tequila – visa oferecer serviços com garantias de QoS e está envolvido com a definição de SLS, protocolos para a negociação e esquemas de engenharia de tráfego intra e inter-domínios. Tem um escopo mais abrangente [25].

Chameleon – permite a oferta de serviços avançados fim a fim independente da quantidade de domínios envolvidos e do tipo de tecnologia de QoS adotada, apresenta o modelo hierárquico de negociação. Composto pelo Plano de Serviços, para fins de definição e negociação de recursos; Plano de Operação, os domínios implementam os serviços negociados através de uma abordagem para QoS na Internet; e Plano de Monitoramento, ortogonal aos demais e executa medição dos parâmetros de QoS [13].

1.3. Estrutura Geral da Dissertação

Esta dissertação está organizada da seguinte forma:

No Capítulo 2 será apresentada uma visão geral dos modelos que têm sido estudados e propostos na literatura de qualidade de serviço. No Capítulo 3 serão apresentados os mecanismos e algoritmos de escalonamento, bem como os controles de congestionamento e policiamento. No Capítulo 4 será especificado o mecanismo de provisionamento proposto. No Capítulo 5 serão discutidas as análises de desempenho do mecanismo a partir das simulações no *Network Simulator* (NS). No Capítulo 5 as conclusões finais serão apresentadas. O Apêndice A apresenta os códigos fonte necessários à reprodução do experimento usado na simulação. O Apêndice B apresenta um glossário com as principais siglas usadas neste trabalho.

2. O Estado da Arte em Qualidade de Serviço

Neste capítulo, será apresentada uma visão geral dos modelos, arquiteturas e protocolos de qualidade de serviço encontrados na literatura.

2.1. Arquiteturas de Qualidade de Serviço

2.1.1. Serviço de Melhor Esforço

No modelo de serviço de melhor esforço, o tráfego é processado tão rapidamente quanto possível, não oferecendo garantias de qualidade de serviço para as novas aplicações avançadas. Por exemplo, as aplicações tolerantes a atrasos e perda de pacotes competem pelos recursos em forma de igualdade com aplicações sensíveis a estes requisitos. Quando há ocorrência de congestionamento, pacotes são descartados pela ordem de chegada, independente da origem ou aplicação.

Com o surgimento dessas novas aplicações, constatou-se que o modelo não é adequado para atender aos diversos requisitos de qualidade. Este modelo, provê simplesmente a conectividade proporcionando que aplicações tolerantes, como aplicações de correio eletrônico e transferências de arquivos, adaptem-se facilmente às condições da rede. Assim, novos serviços devem ser fornecidos a fim de atender aos diferentes requisitos das aplicações.

Como forma de suportar diferentes níveis de QoS, desenvolveu-se novas classes de serviços provendo garantias, a um determinado preço, satisfazendo as exigências das aplicações. Algumas garantias como atraso, variação do atraso, largura de banda, e confiabilidade nas transmissões está disponível nestas novas classes de serviços possibilitando o tratamento diferenciado dos serviços.

Algumas abordagens têm sido propostas pela IETF visando enfrentar o problema de oferecer garantias de QoS na rede, destacam-se Serviços Integrados (*Intserv*), Serviços Diferenciados (*Diffserv*), *Multiprotocol Label Switching* (MPLS), Roteamento Baseado em Restrições (QoS *Routing*).

2.1.2. Serviços Integrados (*Intserv*)

O modelo de serviços integrados foi proposto como uma extensão do serviço de melhor esforço, baseado na explicita alocação e reservas de recursos. Este modelo especifica diferentes classes de serviços requerendo grandes mudanças na infra-estrutura da rede, principalmente na adição de novos componentes aos roteadores de um domínio *Intserv*. No modelo de serviços integrados, nenhuma garantia pode ser obtida sem reservas.

A Figura 01, [29], mostra o modelo de referência para roteadores que implementam a arquitetura *Intserv*.

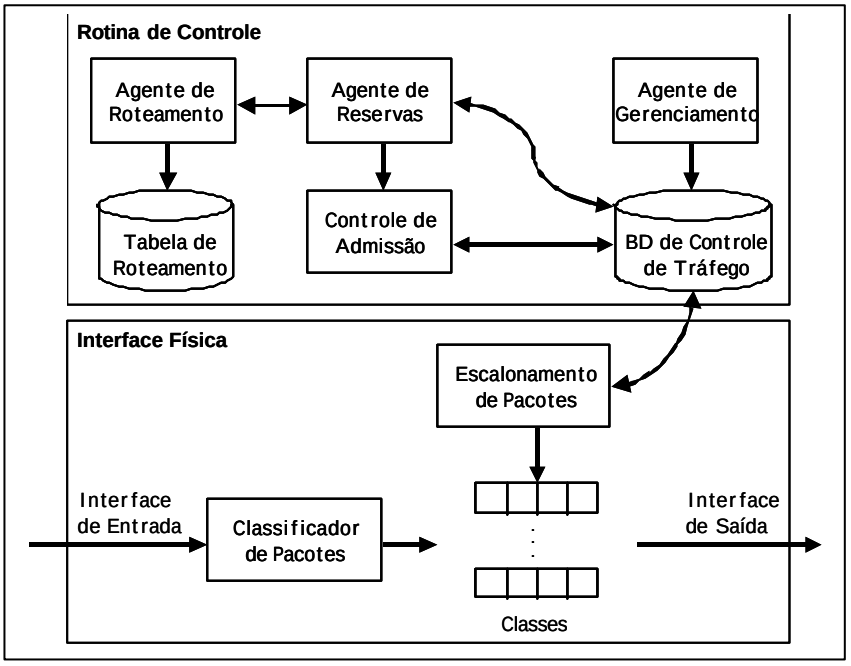

Figura 01 – Modelo de referência para roteadores *Intserv*

O esquema apresenta duas divisões funcionais onde a primeira é representada pela Interface Física, isto é, identifica o caminho das informações, que recebe e envia os dados de acordo com uma política definida em cada roteador. Nesta parte, o roteador classifica os pacotes em filas onde possam ser reordenados e reenviados, mantendo os níveis de QoS dos outros fluxos prioritários.

A segunda divisão funcional é chamada de Rotinas de Controle, executada em segundo plano, responsável por controlar e modificar as bases

de dados do modelo, controlando o caminho a seguir pelas informações. As rotinas de controle são formadas por três agentes: Agente de Roteamento que faz a interface com os protocolos de roteamento para obtenção dos endereços destino; Agente de Reserva que implementa o protocolo de configuração de reservas e o Agente de Gerenciamento que implementa, de forma local, a política de compartilhamento, modificando os parâmetros do controle de admissão.

O modelo de serviços integrados é composto pelos seguintes componentes:

- Controle de Admissão – responsável pelo atendimento das requisições de QoS de um determinado fluxo, controlando a alocação de recurso da rede. É implementado por um algoritmo que verifica o impacto nas reservas já garantidas, levando em consideração a carga de tráfego atual e o perfil de tráfego contratado;
- Classificador – os pacotes pertencentes a um determinado fluxo são mapeados nas classes de serviços existentes no modelo *Intserv*. Os pacotes pertencentes a uma mesma classe recebem o mesmo tratamento definido no perfil contratado de serviços. Um pacote pode ser classificado de maneira diferente por roteadores diferentes ao longo do trajeto [29];
- Escalonador de pacotes – trata o encaminhamento de diversos fluxos utilizando um conjunto de mecanismos de filas e temporizadores. O escalonador coordena as filas de encaminhamento enfileirando os pacotes e comunicando-se com a camada de enlace de dados, a fim de controlar a alocação dos recursos contratados, efetuando medições dos estados dos serviços alocados no escalonador;
- Protocolo de configuração de reserva – é implementado nos elementos de redes, tais como roteadores, adequando-se aos mecanismos de controle do modelo de referência, criando e mantendo os estados de cada fluxo nos *hosts* destinos do domínio *Intserv*. Realiza sinalizações para as aplicações

reservando recursos da rede. O protocolo utilizado é o *Resource Reservation Protocol* (RSVP), desenvolvido pela IETF.

A fim de fornecer qualidade de serviço para determinados fluxos de uma aplicação, o modelo de serviços integrados deve ser capaz de reservar recursos fim a fim, mantendo um perfil contratado pelo usuário. Duas novas classes de serviços são implementadas no modelo de serviços integrados, em adição ao modelo de serviço de melhor esforço para fornecer estas garantias: o Serviço Garantido e o Serviço de Carga Controlada.

2.1.2.1. Serviço Garantido

O Serviço Garantido fornece uma garantia quanto ao limite no atraso fim a fim e uma proteção ao descarte de pacotes que estejam de acordo com os perfis de tráfego contratados.

A garantia deste serviço não está relacionada em minimizar as variações de atraso, mas somente controlar os atrasos máximos de enfileiramentos nos roteadores. Este serviço fornece um nível assegurado de taxa de transmissão. É necessário que todos os roteadores do domínio implementem os serviços garantidos.

Esse serviço atende principalmente as aplicações de tempo real, como certas aplicações multimídias de áudio e vídeo.

2.1.2.2. Serviço de Carga Controlada

O Serviço de Carga Controlada atende às aplicações tolerantes aos atrasos e perdas de pacotes até um determinado limite. Equivale ao serviço de melhor esforço quando a rede não está sobrecarregada.

Para assegurar o controle da carga, o serviço assume que um alto percentual de pacotes será transmitido com sucesso, com um atraso de enfileiramento mínimo, entretanto não são fornecidas garantias quantitativas de desempenho.

Esse serviço atende principalmente as novas aplicações de tempo real adaptativas, que estão sendo desenvolvidas para a Internet, porém não funcionam em redes carregadas com um alto nível de congestionamentos.

2.1.2.1. O Protocolo de Reserva de Recursos (RSVP)

O *Resource Reservation Protocol* (RSVP) é um protocolo de reserva de recursos fim a fim, utilizado no modelo *Intserv*. O protocolo está sendo padronizado pela IETF, visando possibilitar a integração de serviços de forma confiável e estável.

Para manter os níveis de QoS ao longo do caminho, o protocolo efetua sinalizações solicitando e reservando os recursos necessários para cada fluxo de dados a ser transmitido por uma aplicação, sendo utilizado tanto por um *host* como por um roteador de dentro do domínio *Intserv*.

O RSVP solicita recursos somente em uma direção (*simplex*) para uma grande variedade aplicações e receptores (*unicast e multicast*), suportando diferentes estilos de reserva através de filtros. Apesar de ocupar o lugar do protocolo de transporte na pilha de protocolos IP, suportando tanto o IPv4 como o IPv6, o RSVP não transporta dados das aplicações, sendo apenas um protocolo de controle atuando igualmente ao *Internet Control Message Protocol* (ICMP) e ao *Internet Group Management Protocol* (IGMP).

As reservas nos roteadores do domínio *Intserv* são mantidas por estados voláteis (*soft-state*), que precisam ser renovados periodicamente para permanecerem ativos. O receptor atualiza os estados de reserva, automaticamente, permitindo uma melhor adaptação às mudanças de roteamento bem como no re-estabelecimento de reservas em caso de falhas. As mensagens RSVP são encaminhadas transparentemente em roteadores que não implementam RSVP.

Na especificação do RSVP são definidos formatos e tipos de objetos que serão transportados pelas mensagens, além de um conjunto de regras de transmissão e processamento de mensagens. A mensagem RSVP consiste de um cabeçalho comum, seguido de um ou mais objetos de tamanhos variáveis. Temos os seguintes tipos de mensagens:

- Mensagem de caminho (*PATH Message*) – periodicamente, os transmissores enviam uma mensagem de caminho para cada fluxo de dados que irá transmitir no contexto de reserva de recursos, a mensagem é encaminhada em direção ao destino pelo mesmo caminho que será usado pelo fluxo de dados;
- Mensagem de reserva (*RESV Message*) – as mensagens de reserva carregam as solicitações de reserva de recursos originadas nos receptores em direção aos transmissores, utilizando o caminho inverso da mensagem de caminho;
- Mensagem de liberação (*Teardown Message*) – utilizadas para a liberação dos estados de reserva e de caminho entre a origem e o destino e vice-versa. Uma mensagem para liberar o estado de caminho (*PathTear*) e outra para liberar o estado de reserva (*ResvTear*).
- Mensagem de erro (*Error Message*) – carregam as informações a respeito de problemas que eventualmente tenham ocorrido no estabelecimento do caminho (*PathErr*) e no estabelecimento das reservas (*ResvErr*);
- Mensagem de confirmação (*Confirmation Message*) – utilizada para a confirmação de uma reserva, quando na mensagem de reserva é solicitada tal confirmação, enviada diretamente ao *host* usando comunicação *unicast*.

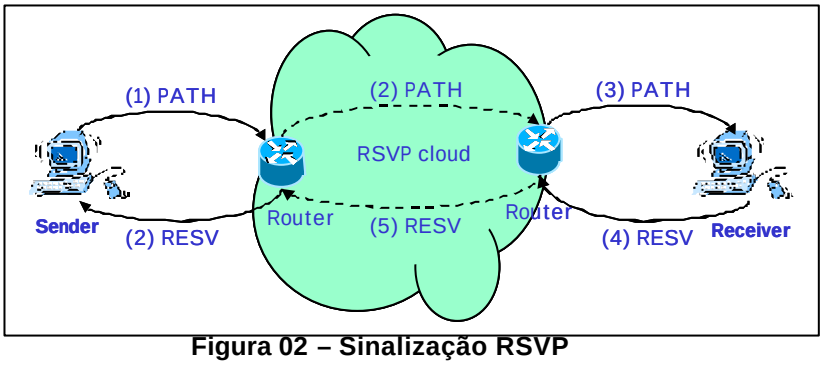

Figura 02 – Sinalização RSVP

A Figura 02 apresenta a troca de mensagens, do protocolo RSVP, entre o transmissor (*sender*) e o receptor (*receiver*). O transmissor envia uma mensagem *PATH* para o receptor, a fim de obter uma reserva de tráfego a

partir dos níveis de QoS da aplicação. Os roteadores intermediários transferem a mensagem *PATH* para os roteadores vizinhos, utilizando um protocolo de roteamento, mantendo as informações de roteamento reverso disponível.

O roteamento reverso é necessário para que a mensagem de reserva possa percorrer o mesmo caminho da mensagem *PATH*. Sem as informações de roteamento reverso, as mensagens de reserva poderiam ser encaminhadas por caminhos diferentes daqueles onde a mensagem *PATH* foi encaminhada.

As mensagens *PATH* são responsáveis em caracterizar o tráfego e o caminho a percorrer das aplicações, enquanto que as mensagens *RESV* efetuam as requisições dos recursos nos roteadores do domínio *Intserv*.

Ao receber uma mensagem *PATH*, o receptor responde com uma mensagem *RESV*, requisitando recursos para os fluxos das aplicações. Os roteadores intermediários, no caminho reverso, aceitam ou rejeitam as mensagens de reserva.

Se uma mensagem de reserva é rejeitada, o roteador envia uma mensagem de erro para o receptor terminando o processo de sinalização. Se uma mensagem de reserva é aceita, o roteador aloca os recursos para o fluxo determinado e as informações de estado relacionadas são armazenadas. Por fim, a aplicação no transmissor recebe uma mensagem de reserva efetivada dando início ao envio das informações para o receptor.

Basicamente, dois problemas impossibilitam a utilização do modelo de serviços integrados em redes com um grande volume de tráfego: a primeira refere-se a grande quantidade de estados necessários para cada fluxo individual; e a segunda refere-se as implementações do modelo de referências nos roteadores *Intserv*, sobrecarregando-os, devido aos altos níveis de processamento e armazenamento simultâneos de vários fluxos das aplicações. Por estas razões, o modelo não é escalável, sendo apenas recomendando para redes confinadas de pequeno tráfego.

2.1.3. Serviços Diferenciados (*Diffserv*)

O modelo de serviços diferenciados foi proposto com o objetivo de fornecer diferentes classes de serviços, sem a necessidade de manter estados individuais de fluxos, provendo escalabilidade a rede. Essas classes de serviços são definidas por alguns aspectos relacionados à transmissão, de acordo com algum parâmetro de desempenho.

Basicamente, o modelo de serviços diferenciados é composto por um conjunto de nós contíguos, onde as complexidades das operações de classificação e marcação de pacotes, se encontram nas fronteiras do domínio *Diffserv*. A implementação dos mecanismos internos é feita da forma mais simples, não necessitando, por exemplo, de operações de sinalização em cada nó da rede. Enquanto que nas fronteiras do domínio existe um estado por fluxo, identificando cada fluxo a um perfil contratado, nos nós interiores há somente um estado por classe de serviço oferecida, aumentando a escalabilidade do modelo.

O modelo de serviços diferenciados é implementado nos seguintes elementos de rede: nos roteadores de entrada (*ingress routers*) e nos roteadores de saída (*egress routers*), que também são chamados de roteadores de borda (*edge routers*) e nos roteadores internos (*core routers*), veja Figura 03.

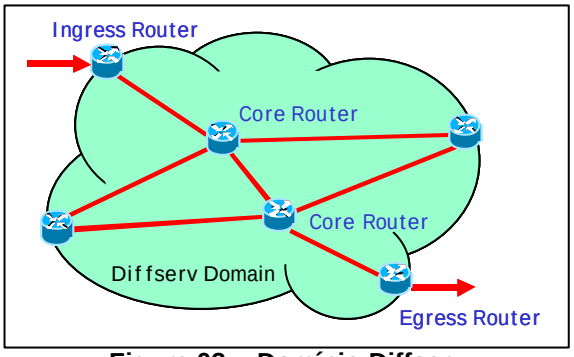

Figura 03 – Domínio Diffserv

Os roteadores de borda são responsáveis pela classificação e condicionamento do tráfego que entra no domínio *Diffserv*. A partir de um perfil contratado, os roteadores de borda marcam os pacotes e definem qual o tratamento de serviço diferenciado os pacotes irão receber. Os roteadores

de ingresso têm papel fundamental neste modelo. Os roteadores internos examinam a marcação dos pacotes, atuando de acordo com as políticas contratadas, encaminhando em seguida os pacotes para o próximo nó da rede.

Com o objetivo de identificar as agregações de fluxos, definindo o comportamento de encaminhamento dos pacotes dentro do domínio, o modelo utiliza uma marcação de um novo campo chamado *Differentiated Services* (DS). O campo DS é obtido a partir do campo *Type of Service* (TOS) do cabeçalho do pacote IPv4 ou do campo *Traffic Class* do cabeçalho do pacote IPv6 [12], a Figura 04 ilustra esses cabeçalhos.

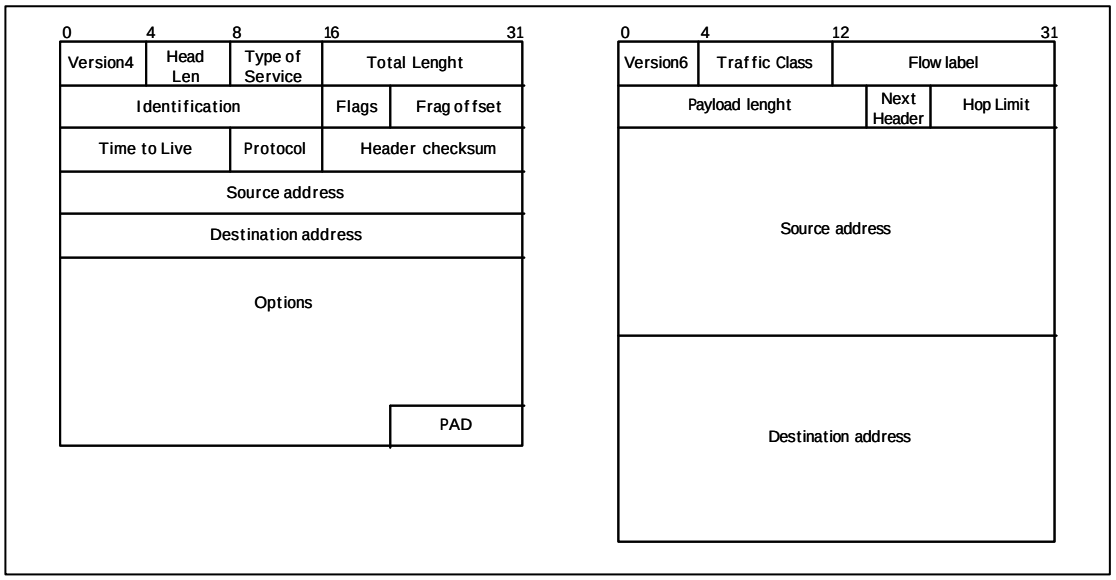

Figura 04 – Diagrama IPv4 e IPv6 Header

O campo DS tem como estrutura, veja Figura 05, um *label* de 8 *bits* onde é composto por dois subcampos, um chamado de *Differrentiated Services Codepoint* (DSCP), que tem o tamanho de 6 *bits* e identifica as agregações de fluxos, e outro de 2 *bits*, ainda não definido, reservado para uma futura utilização.

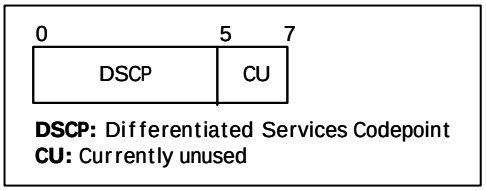

Figura 05 – Definição do campo DS

Os três primeiros *bits* do subcampo DSCP identificam a classe de serviço (001, 010, 011 e 100) enquanto que os três últimos definem o nível de precedência de descarte (010 – precedência baixa, isto é, último a ser descartado; 100 – precedência média; e 110 – precedência alta, isto é, primeiro a ser descartado).

Nos roteadores que implementam o modelo de serviços diferenciados, o valor do subcampo DSCP, do cabeçalho dos pacotes IP, é mapeado para um perfil contratado a cada nó da rede para o seu encaminhamento.

O tratamento recebido pelo pacote, identificando uma agregação de fluxos, dá-se o nome de *Per-Hop Behavior* (PHB). A partir da combinação dos PHBs pode-se fornecer um serviço fim a fim interior ao domínio. Os PHBs são implementados utilizando mecanismos de controle de filas, que serão apresentados no próximo capítulo.

O PHB BE, também chamado PHB *Default*, implementa o tráfego de melhor esforço, não fornecendo garantias de QoS aos perfis de tráfego contratados. O valor recomendado para o subcampo DSCP em decimal é (0).

Outras duas propostas de PHB, são implementadas no modelo de serviços diferenciados:

- Encaminhamento Expresso (*Expedited Forwarding* – PHB EF) – que oferece a garantia de uma taxa mínima de transmissão pré-definida, com baixa perda, baixo atraso e baixa variação de atraso, utilizado para obtenção de serviço fim a fim, equivalendo a um canal dedicado [27];
- Encaminhamento Assegurado (*Assured Forwarding* – PHB AF) – que fornece largura de banda assegurada, não oferecendo garantias quanto ao atraso; apresenta quatro classes de encaminhamento de pacotes, onde para cada classe os pacotes podem ser marcados com até três níveis de precedência de descarte [9].

2.1.3.1. Condicionamento de Tráfego

Dois tipos de classificadores estão presentes neste modelo. O classificador chamado de Comportamento Agregado (*Behaviour Agregate* – BA), que classifica o fluxo baseado apenas no subcampo DSCP, e o classificador Multicampo (*Multifield Classifier* – MF), que verifica múltiplos campos no cabeçalho IP, como por exemplo, os endereços de origem e destino, endereços da porta de origem e destino, além do subcampo DSCP. Os classificadores MF encontram-se nos nós de ingresso, enquanto que nos nós interiores o classificador utilizado é o de comportamento agregado.

O modelo Diffserv contém um conjunto de elementos funcionais implementados nos elementos de rede, que são responsáveis pelo condicionamento do tráfego, estabelecendo um perfil de transmissão a ser atendido no domínio, Figura 06. A política de tráfego, que contém as regras de classificação e as características do tráfego contratado, chama-se de Acordo de Condicionamento de Tráfego (*Traffic Conditioning Agreement* – TCA).

Outros contratos denominados Service Level Agreement (SLA) definem os níveis de encaminhamento de serviços utilizados entre um provedor de serviços e um cliente, que pode ser um usuário final ou outro provedor. As especificações particulares de um determinado serviço, como por exemplo, classificação de pacotes, perfil de tráfego e as garantias de desempenho, que são as especificações técnicas do serviço definido em um SLA, são encontradas nos Service Level Specifications (SLS). Um SLA pode ser classificado como: SLA Estático que são negociados de forma regular e o SLA Dinâmico onde se utilizam protocolos de sinalização para requisitar serviços sob demanda.

Além do classificador, outros elementos que executam o condicionamento de tráfego, são:

- Medidor – verifica a conformidade do tráfego de acordo com um perfil de tráfego pré-definido no TCA;
- Marcador – marca um pacote de uma determinada agregação, estabelecendo o valor do subcampo DSCP;

- Suavizador – retém um ou mais pacotes até que estes estejam em conformidade com o perfil contratado para serem encaminhados para o próximo nó;
- Policiador – descarta os pacotes que não estão em conformidade ao perfil contratado. Em acréscimo, os pacotes que excedem o perfil, podem ser tratados de maneiras diferentes, descartados, suavizados ou novamente remarcados para uma agregação de serviço inferior. Estes parâmetros devem estar definidos no TCA.

A Figura 06 ilustra o diagrama de condicionamento de tráfego, representando o relacionamento entre seus elementos.

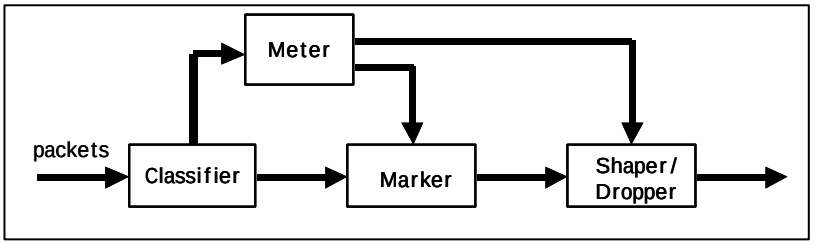

Figura 06 – Diagrama de condicionamento de tráfego

2.1.3.2. Encaminhamento Expresso

O Serviço de Encaminhamento Expresso define garantias rígidas de QoS para aplicações avançadas, provendo um serviço fim a fim, com baixos níveis de perda de pacotes, baixo atraso e variação do atraso, bem como largura de banda garantida.

Com o objetivo de amenizar a permanência dos pacotes, pertencentes a uma determinada agregação de fluxos, em filas no interior de domínio DS, o PHB de Encaminhamento Expresso, fornece uma garantia no recebimento da taxa de transmissão, maior que a taxa de chegada em qualquer momento, apresentando-se como um caminho dedicado, independente de outros tráfegos compartilhados na rede.

O PHB EF pode ser utilizado para tráfegos que necessitem de baixo atraso e baixa variação de atraso com uma garantia de taxa de serviço. Os dispositivos de borda do domínio DS policiam o tráfego EF garantindo que a

sua taxa de chegada esteja em conformidade com a taxa contratada. Os pacotes não conformes com a taxa estabelecida no perfil contratado são descartados.

A Tabela 02 apresenta o valor do sub-campo DSCP, recomendado para a classe de serviço do PHB EF no modelo de serviços diferenciados.

Tabela 02 – Classe de tráfego PHB EF

Precedência de Descarte	Classe EF	
	Binário	Decimal
N/A	101110	46

Alguns mecanismos de controle filas, como o *Priority Round-Robin* (PRR) e *Weighted Roud-Robin* (WRR), implementam o PHB EF.

O mecanismo PRR descreve uma estrutura com uma fila prioritária fornecendo um serviço pré-contratado, para que os pacotes tenham uma permanência mínima na fila.

No mecanismo WRR, na fila que atende a agregação EF, o serviço alocado deve ser correspondente a, no mínimo, a taxa de serviço contratada, dentre um conjunto de filas de diferentes prioridades [64]. Em [27], é realizada uma análise da variação do atraso apresentada por um tráfego CBR quando o PHB EF é implementado por PRR ou por WRR. A conclusão demonstra que a implementação por PRR é o melhor ao invés da implementação por WRR.

2.1.3.3. Encaminhamento Assegurado

O PHB AF permite a entrega de pacotes de uma agregação de fluxos com uma determinada taxa de serviço assegurada, entretanto não fornece nenhuma garantia específica em relação ao atraso. Uma expectativa de serviço é obtida para um determinado tráfego quando, eventualmente, um aumento na carga da rede é apresentado.

Os mecanismos de controle de admissão e condicionamento de tráfego, atuam nos roteadores de borda garantindo a existência dos recursos necessários no domínio, marcando os pacotes em conformidade ao perfil contratado.

Este serviço de encaminhamento oferece diferentes níveis de garantia no encaminhamento dos pacotes das agregações. Os pacotes marcados como conformes, isto é, aqueles que apresentam os parâmetros de QoS de acordo com o requisitos de tráfego pré-contratados, são entregues com alta prioridade enquanto o fluxo agregado não exceder a taxa de serviço contratada definida no perfil de serviço. O tráfego excedente não será entregue com a mesma prioridade alta de entrega.

O Serviço Assegurado define quatro classes independentes com três precedências de descarte de pacotes, onde cada classe contém uma quantidade determinada de recursos (*buffers* e largura de banda). Uma precedência de descarte estabelece a importância relativa de um pacote dentro de uma classe. Os pacotes de um fluxo agregado, com um alto valor de precedência de descarte, são descartados prioritariamente em relação aos pacotes com um baixo valor de precedência de descarte.

A Tabela 03 apresenta os valores do sub-campo DSCP, recomendados para as classes de serviços do PHB AF no modelo de serviços diferenciados.

Tabela 03 – Classes de tráfego PHB AF

Precedência de Descarte	Classe AF1		Classe AF2		Classe AF3		Classe AF4	
	Binário	Decimal	Binário	Decimal	Binário	Decimal	Binário	Decimal
Baixa	001010	10	010010	18	011010	26	100010	34
Média	001100	12	010100	20	011100	28	100100	36
Alta	001110	14	010110	22	011110	30	100110	38

A implementação de um PHB AF procura minimizar os congestionamentos duradouros, permitindo os de curta duração, como os resultantes de tráfegos em rajadas.

A garantia na entrega dos pacotes em um Serviço Assegurado depende da quantidade de recursos alocados para cada classe e em caso de congestionamentos, do valor de precedência de descarte.

2.1.3.4. Corretor de Largura de Banda

O Corretor de Largura de Banda ou *Bandwidth Broker* (BB) é definido no contexto do grupo de pesquisas do projeto *Qbone* da Internet2, chamado *Bandwidth Broker Advisory Council* (BBAC) [18][20].

O BB gerencia os recursos das redes que suportem serviços com QoS, como um tipo de servidor de políticas efetuando algumas tarefas como controle de admissão, alocação de recursos, controle de políticas e prioridades, configuração de dispositivos da rede e outros, de uso especifico de domínios *Diffserv*.

Para gerenciar os recursos no domínio, o BB necessita de informações sobre provisionamento, identificação de fluxos, alocação de recursos, estado da rede, além das questões comerciais. O objetivo principal destas informações é verificar as disponibilidades de recursos, suportando as requisições dos usuários.

As especificações (SLS) estão contidas nos acordos (SLA) dentro de um domínio ou nos domínios adjacentes mantendo informações sobre os serviços disponibilizados.

O BB executa as seguintes funções básicas: alocar largura de banda, atender e notificar as solicitações de requisição de largura de banda, configurar dispositivos de rede, autorizar tráfego no domínio, gerenciar a troca de mensagens no domínio e entre BB adjacentes em diferentes domínios.

Uma mensagem de solicitação de recursos ou *Resource Allocation Request* (RAR) é definida a partir de um usuário no domínio, para um BB, quando se deseja obter recursos na rede. Se a utilização do recurso estiver fora do domínio do BB solicitado, o controle de admissão, baseado nos SLS dos domínios em questão, solicitará aos BB adjacentes os recursos a serem provisionados, efetuando a aceitação ou não da requisição inicial. A resposta à requisição inicial é chamada de *Resource Allocation Answer* (RAA).

Os parâmetros da solicitação RAR, definidos no projeto Qbone, são:

- BB_ID – identificação do BB;
- RAR_ID – identificação da mensagem RAR;
- SRC – origem da solicitação;
- DST – destino da solicitação;
- START_TIME – tempo de início da alocação do recurso;
- END_TIME – tempo final da alocação do recurso;
- DSCP – código do tipo de serviço;
- Outros parâmetros: identificação dos roteadores de *ingress* e *egress*, protocolos, número das portas, parâmetros de QoS negociáveis.

Os parâmetros da resposta RAA, definidos no projeto *Qbone*, são:

- BB_ID – identificação do BB;
- RAR_ID – identificação da mensagem RAR;
- STATUS – confirmação ou rejeição ou indicação em progresso da solicitação.

O projeto *Qbone* define duas fases de admissão executadas através do recebimento e envio de mensagens pelos BB. A requisição de recurso é originada de aplicações ou usuários, bem como dos roteadores da rede para um BB no domínio.

Na primeira fase define-se a admissão local, com um escopo intradomínio, onde os recursos a serem provisionados localizam-se dentro do domínio do BB solicitado. A interação, entre BBs de um domínio, pode ser efetuada em sub-redes utilizando o *Subnet Bandwidth Manager* (SBM). Apenas um dentre os BBs do domínio pode ser responsável pela comunicação entre domínios adjacentes.

A segunda fase define a admissão entre domínios adjacentes, com um escopo interdomínio, onde os recursos a serem provisionados localizam-se em domínios adjacentes ao BB solicitado. O BB gerencia recursos através da troca de mensagens com os BBs adjacentes, através de canais seguros, para que todos os BBs do caminho sejam consultados avaliando os serviços requeridos.

Duas propostas de sinalização entre os BBs podem ser utilizadas na comunicação para alocação de recursos: sinalização fim-a-fim e sinalização com resposta imediata (Figura 07).

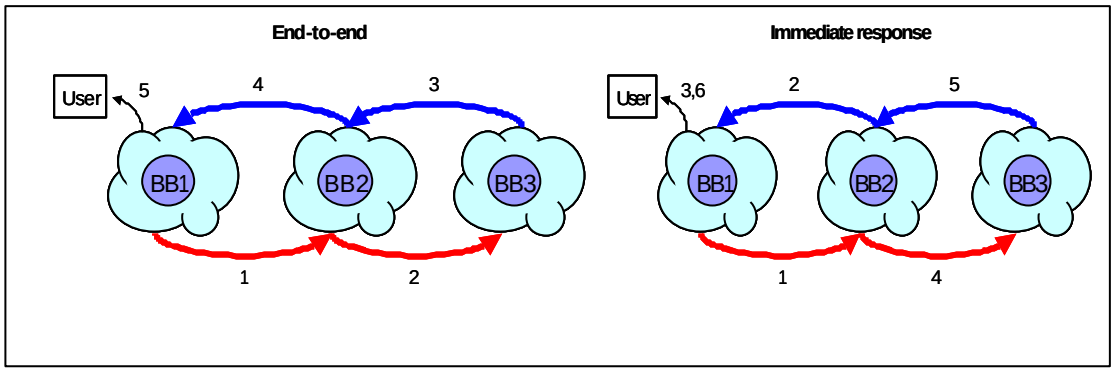

Figura 07 – Sinalização do *Bandwidth* Broker

A sinalização fim-a-fim, onde o usuário que solicitou o recurso só receberá a resposta de aceitação ou não do recurso quando todos os BBs do caminho forem consultados. Quando houver uma falha no caminho do BB a ser consultado, então uma resposta é retornada, não sendo consultados os BBs subseqüentes. Esta proposta de sinalização se mostra mais confiável em comunicações que envolvam a garantia de transmissão para a aplicação.

A sinalização com resposta imediata, o usuário recebe a resposta de aceitação ou não do recurso imediatamente, a cada BB adjacente percorrido. Assim, a aplicação pode ser iniciada imediatamente, enquanto o processo continua nos outros BBs do caminho. A ocorrência de uma falha nas negociações, neste tipo de sinalização, ocasionará na perda de todos os dados da aplicação.

Como forma de priorizar aplicações, em casos de congestionamento, o BB pode utilizar o protocolo RSVP na sinalização de recursos, efetuando as reservas necessárias para o serviço especificado. De maneira geral, o uso das mensagens RSVP está relacionado com o transporte de controles entre dispositivos de rede e os corretores de largura de banda.

Devido ao problema de manter estados individuais de controle, o protocolo RSVP não é recomendado para redes de trânsito com grande volume de tráfego. Por outro lado seu uso em redes finais e pequenas torna-se viável gerenciando adequadamente as mensagens de sinalização [65].

A Figura 08 apresenta a arquitetura do BB, proposta pelo projeto *Qbone*.

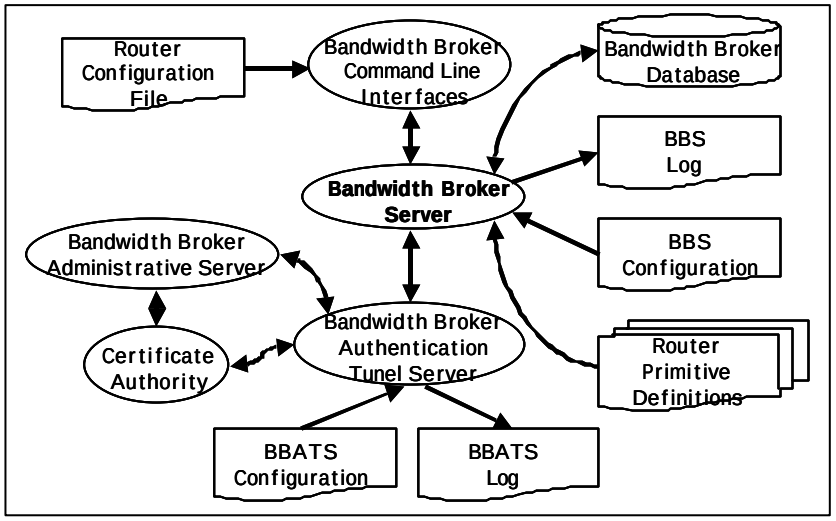

Figura 08 – Arquitetura do *Bandwidth Broker*

A seguir, os componentes da arquitetura do BB com suas principais funções:

- *Bandwidth Broker Database* (BBD) – inclui mecanismos para armazenar todos os dados utilizados pelo BB dentro do domínio;
- *Bandwidth Broker Server* (BBS) – entidade que gerencia a interação entre os diversos componentes; executa a validação dos dados policiando as prioridades das atividades de mudança das informações no BBD; atualiza *logs* de transações;
- *Bandwidth Broker Command Line Interfaces* (BBCLI) – provê comandos para operadores de sistemas interagirem com o BBS e o BBD; permite a manutenção dos SLSs, das manutenções de alocação de banda e geração de arquivos de configuração para os dispositivos de rede;
- *Bandwidth Broker Authentication Tunnel Server* (BBATS) – atua como intermediário entre BBAS e BBS local, garantindo a segurança das transações e o controle administrativo;

- *Bandwidth Broker Administrative Server* (BBAS) – gerencia os negócios e contratos entre os provedores de serviços e os clientes.

2.1.4. Engenharia de Tráfego

A engenharia de tráfego é responsável pelo processo de organização do tráfego na rede para que congestionamentos possam ser evitados. Os congestionamentos podem ser causados pela falta de recursos na rede ou pela irregular distribuição do tráfego.

A falta de recursos pode ser indicada como uma sobrecarga nos roteadores e caminhos percorridos pelo tráfego tendo como uma possível solução o aumento da capacidade da rede provendo aumento nos recursos de infra-estrutura.

A distribuição irregular do tráfego na rede deve-se pela utilização dos atuais protocolos de roteamento (RIP e OSPF) que selecionam o caminho mais curto trazendo como conseqüência a sobrecarga de parte da rede enquanto outras encontram-se levemente carregadas.

Devido a natureza competitiva da Internet, a engenharia de tráfego tornou-se uma função indispensável. Sem engenharia de tráfego poderíamos ter uma situação onde os fluxos de tráfego são encaminhados pelo caminho mais curto, utilizando os protocolos atuais de roteamento. Enquanto que nos caminhos alternativos, não há fluxo, o desempenho da rede como um todo é degradado apesar de haver recursos disponíveis.

De outra forma, com a utilização da engenharia de tráfego, os fluxos de tráfegos são tratados por diferentes caminhos, de acordo com o desempenho desejado, utilizando racionalmente os recursos disponíveis evitando assim sobrecargas na rede [32].

A engenharia de tráfego preocupa-se com a otimização do desempenho das redes englobando a aplicação de princípios tecnológicos e científicos para controlar o tráfego, facilitando a operação eficiente e confiável da rede. Pode ser utilizada para atender os requisitos de QoS de

determinados fluxos, alterando o caminho normalmente utilizado pelos protocolos de encaminhamento.

A engenharia de tráfego pode utilizar algumas técnicas como MPLS ou Roteamento com QoS para descobrir e fixar os caminhos mais adequados a determinados fluxos dentro da rede [33].

2.1.5. Protocolo de Encaminhamento Baseado em Rótulos (MPLS)

O MPLS é um mecanismo aberto e interoperável criado a partir do grupo de pesquisas da arquitetura *Label Based Switching* (LBS) normatizada pela *Internet Engineering Task Force* (IETF). Apresenta como características principais: a agilidade no encaminhamento de pacotes proporcionada pela análise dos rótulos somente na borda do domínio MPLS; implementação de orientação à conexão; suporte as arquiteturas que implementa QoS como o *Intserv* e *Diffserv*; e simplificação na interoperabilidade de rede com tecnologias heterogêneas [50].

O mecanismo MPLS propõe a classificação de um conjunto de pacotes em um conjunto de classes de equivalência chamada *Forward Equivalence Class* (FEC) e mapeia cada classe em um caminho a seguir na rede. Não há distinção de pacotes que sejam classificados com um mesmo FEC.

Um domínio MPLS é composto por todos os nós contíguos que implementem o protocolo de encaminhamento baseado em rótulos. Os rótulos são atribuídos na entrada de um domínio MPLS pelo dispositivo de borda *Label Edge Router* e associado à classe de equivalência de encaminhamento FEC.

Os dispositivos interiores ao domínio MPLS, denominados *Label Switching Routers*, necessitam apenas verificar os rótulos atribuídos a cada pacote, analisam a qual FEC são atendidos e tomam a decisão de encaminhar o pacote [32][60].

O mecanismo MPLS é dividido em dois componentes básicos: componentes de encaminhamento e componentes de controle.

O componente de encaminhamento é composto pelos seguintes elementos:

- *Label* – rótulo de tamanho pequeno e fixo que funciona como índice para a tabela de encaminhamento; é inserido no cabeçalho do pacote pelo LER;
- *Label Switch Router* – dispositivo que executa os algoritmos de encaminhamento e mantém as tabelas de encaminhamento. A fim de manter as tabelas de encaminhamento atualizadas os LSR comunicam-se através de um protocolo chamado *Label Distribuition Protocol*;
- *Label Switch Edge Router* – é um LSR que mantém as funções de encaminhamento e controle, e quando está na entrada do domínio MPLS, atua como responsável pela inclusão do rótulo ao pacote identificando a classe de equivalência correspondente. Quando o LER está na saída do domínio MPLS, atua como responsável pela retirada do rótulo, mantendo a semântica normal do pacote, a fim seguir caminho para redes não MPLS. O processo de ligação de pacotes a uma classe de equivalência, pode ser tão complexo quanto necessário, sem afetar o desempenho geral da arquitetura, devido ao fato de ser efetuado somente na admissão e na saída do pacote do domínio MPLS;
- *Label Switching Forward Tables* - são tabelas responsáveis pelo processo de encaminhamento de pacotes e são mantidas pelos LSR. Consiste basicamente de um campo índice, que é o valor do rótulo de entrada no domínio, uma ou mais entradas, contendo o rótulo de saída do domínio, interface de saída e endereço do próximo salto a seguir.

O rótulo MPLS pode ser obtido a partir de informações contidas na própria camada de ligação de rede no momento da admissão no domínio. Em tecnologias como o ATM é utilizado o par VCI/VPI do cabeçalho das células, enquanto no *Frame Relay*, o rótulo pode ser obtido a partir do campo DLCI. O formato geral para o rótulo MPLS (*Shim Header*) é apresentado na Figura 09,

este campo é adicionado no cabeçalho do pacote entre os cabeçalhos da camada de ligação e camada de rede.

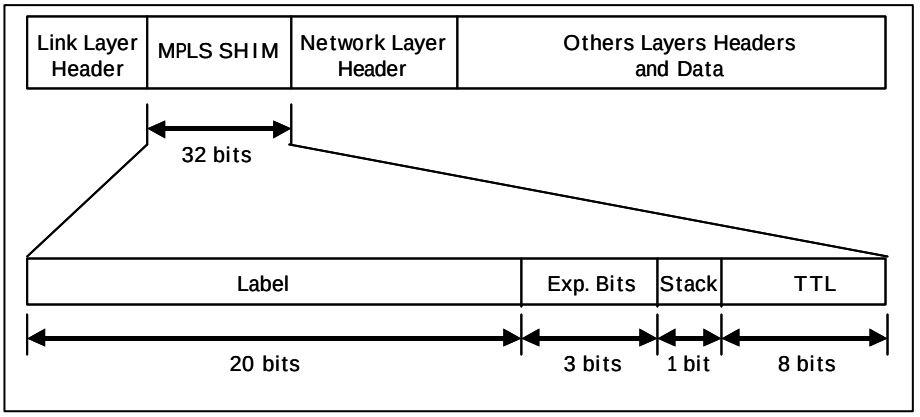

Figura 09 – Formato do Shim Header

O algoritmo utilizado pelo componente de encaminhamento, para troca de rótulos é similar ao processamento de VCI/VPI em redes ATM [52]. Quando um pacote entra num LSR, este analisa o rótulo do pacote e utiliza como um índice para pesquisar na sua tabela de encaminhamento. A partir deste índice, se o rótulo de entrada for localizado então é substituído pelo rótulo de saída, encontrado também na tabela de encaminhamento, e o pacote é enviado pela interface de saída ao próximo nó da rede. Com base nestes rótulos distribuídos são estabelecidos os caminhos de comutação de rótulos *Label Switched Paths*, similar a um circuito virtual do ATM, unidirecional, e podendo ser visto como um túnel.

O processo de encaminhamento de pacotes num domínio MPLS pode ser ilustrado pela Figura 10, que apresenta um LSR interno, quando um pacote é recebido, este é classificado com base no seu rótulo e comutado, sem a necessidade da camada de rede para roteamento, no caso do pacote não possuir um rótulo, ele é roteado de maneira convencional.

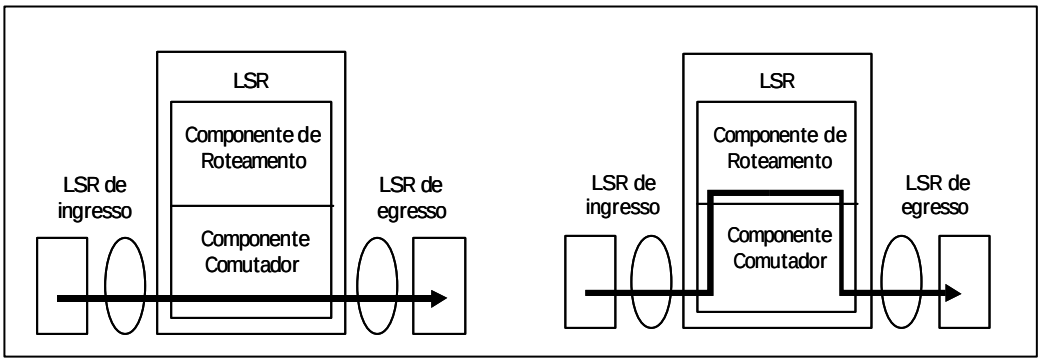

Figura 10 – Encaminhamento de um pacote em MPLS

O componente de controle é responsável por distribuir as informações de roteamento entre os LSR que compõe um domínio MPLS e assim converter estas informações em tabelas de encaminhamento, que são utilizadas pelo componente de encaminhamento.

A partir dos rótulos MPLS, os pacotes podem ser comutados ao longo do caminho da rede, dispensando da análise do cabeçalho em cada dispositivo, este mecanismo contrasta a abordagem convencional onde a cada roteador há uma decisão de qual caminho o pacote irá percorrer. Como parte do componente de controle temos o *Label Distribution Protocol* que executa a distribuição dos rótulos e distribui as informações mantendo as tabelas de encaminhamento nos LSR.

O LDP possui mecanismos de descoberta que auxiliam os LSR a encontrar seus nós adjacentes estabelecendo a comunicação. Define três classes de mensagens baseadas no TCP, provendo entrega confirmada de mensagens, que são o *adjacente* (mensagem de status do LSR, informando atividade, *shutdown* e sessões de inicialização), o *label advertisement* (mensagem de notificação de estabelecimento de ligação de rótulos, renovação e liberação de rótulos), o *notification* (mensagem que provê informações de consulta e sinalização de erro); e uma classe de mensagem baseada no UDP, o *discovery* (mensagem que anuncia um LSR no domínio MPLS).

A descoberta dos LSR vizinhos em um domínio MPLS (LSR *Neighboor Discovery*) é executada periodicamente, via mensagem *discovery*, enviando uma mensagem *multicast* a uma porta conhecida para todos os LSR pertencentes ao grupo.

Em todo o domínio os LSR permanecem esperando esta mensagem. Assim, descobrem os LSR com os quais estão diretamente conectados e armazenam seus endereços. A partir daí, é estabelecida uma conexão, a fim de manter uma sessão LDP, esta, atua de forma bidirecional, requisitando e notificando ligações de rótulos para qualquer LSR vizinho.

Devido a sua escalabilidade, o modelo de serviços diferenciados, vem sendo estudado como um modelo adaptável à arquitetura MPLS, e adequado a grandes redes como a Internet [59]. A classificação e o mapeamento dos pacotes nos PHB num domínio *Diffserv* e a inclusão dos rótulos num domínio

MPLS são feitos de forma semelhante, isto é, executada nos roteadores de entrada ao domínio.

Uma forma do MPLS fornecer suporte ao modelo de serviços diferenciados, é a definição do PHB estar contida no próprio rótulo que é inserido na entrada do domínio MPLS.

No formato geral do rótulo MPLS existe um campo de 3 bits, chamado *Exp. Bits*, que pode ser usado como suporte ao *Diffserv*, podendo assim, estabelecer até oito classes de serviços. Sendo este número de classes inferior ao provido pelo campo DSCP, o suporte ao Diffserv possibilita um nível de diferenciação bastante razoável, comparado ao serviço de melhor esforço atualmente disponível, devido ao fato da arquitetura MPLS ser uma tecnologia de *backbone* das grandes redes.

Essa abordagem é denominada *EXP-Inferred-PSC LSP* onde os caminhos virtuais (LSP) são formados a fim de agregar pacotes de uma mesma classe de serviço, a partir do mapeamento do conteúdo do campo *Exp.* nos PHB dos LSR que compõe o domínio. Cada LSP pode suportar até oito classes de serviços associados a seus respectivos PHB [59].

Outra forma ocorre nas redes com mais classes de serviços como o ATM, onde não há o encapsulamento (*Shim Header*), portanto não existe o campo *Exp.*

No caso do ATM o campo utilizado é o *Cell Loss Priority* (CLP) do cabeçalho da célula como campo de diferenciação de serviços. Esta abordagem é denominada *Label-Only-Inferred-PSC LSP* onde os LSP são formados a partir do mapeamento direto dos rótulos e seu respectivo PHB as FEC. Este mecanismo de distribuição deve permitir a ligação dos rótulos as FEC para um dado PHB, além dos processos efetuados pelos LER. Nesta abordagem é formado um LSP para cada classe de serviço.

Em ambas as abordagens o maior esforço computacional é feito no comutador MPLS da entrada do domínio (LER). Na abordagem E-LSP a utilização do espaço dos rótulos é mais otimizada em vez da abordagem L-LSP, visto que, como é criado um LSP para cada classe de serviço, aumenta-se o número de LSPs, aumentando-se o tamanho da tabela de encaminhamento e consumindo mais espaço dos rótulos disponíveis.

2.1.6. Roteamento baseado em QoS (QoSR)

O roteamento baseado em QoS ou QoS *Routing* (QoSR) surgiu para evitar o desperdício dos recursos da rede, fazendo com que os fluxos de tráfego possam ser distribuídos igualmente [28].

Os protocolos atuais de roteamento utilizam apenas uma métrica, que pode ser a quantidade de saltos ou de roteadores a ser percorrida (RIP), ou o peso administrativo de cada enlace (OSPF).

O QoSR é o processo de selecionar rotas que serão utilizadas pelos pacotes baseados nos requisitos de QoS, como largura de banda e atraso, isto é, utilizando como métricas os parâmetros de qualidade de serviço desejada [34][35]. Pode-se utilizar QoSR para descobrir os caminhos, assim como as disponibilidades dos recursos desejados, nas tabelas de roteamento dos roteadores que implementam como por exemplo: MPLS, *Intserv* ou *Diffserv*.

2.1.7. Roteamento baseado em Restrições

O roteamento baseado em restrições ou *Constraint-Based Routing* (CBR) é utilizado para computar rotas sujeitas a várias restrições [28]. O CBR desenvolveu-se a partir do QoS *Routing*, extendendo-o para outras restrições de rede como políticas administrativas. Tem por objetivos principais: a seleção dinâmica das rotas em busca dos requisitos de QoS do fluxo de tráfego e a otimização dos recursos disponíveis a fim de aumentar eficientemente a utilização da rede.

2.2. Protocolo de Gerenciamento de Políticas (COPS)

O *Common Open Policy Service* é um protocolo de gerenciamento de políticas definido pela troca de informações entre o servidor de políticas e os clientes. É responsável em obter e registrar informações externas dos estados dos fluxos no tráfego da rede.

Como principais características podemos citar:

- emprega o modelo cliente/servidor onde os clientes enviam seus pedidos e atualizações para servidores remotos e locais obtendo em seguida decisões sobre a política;
- usa o protocolo TCP como protocolo de transporte seguro na troca de informações entre clientes e o servidor, entretanto, nenhum mecanismo adicional de segurança é utilizado entre o servidor e os clientes;
- o protocolo é extensível suportando diversos tipos de clientes, não necessitando de modificações no protocolo COPS, criado para geral administração, configuração e aplicação de políticas.

A Figura 11 ilustra o diagrama dos componentes do COPS: *Policy Decision Point*, *Policy Enforcement Point* e *Local Policy Decision Point*.

No COPS são definidos três pontos: um ponto remoto de decisão de políticas, PDP, também nomeado de servidor de políticas; um ponto onde as políticas são aplicadas, PEP; e um ponto de decisão local de políticas, LPDP. As comunicações entre o PEP e o PDP são efetuadas por conexões TCP, iniciando uma sessão de gerenciamento de políticas.

As informações sobre políticas são registradas no PDP. Se houverem requisições ou atualizações de informações pelo PEP, estas serão informadas ao PDP. Ocasionalmente, o PDP, força mudanças no estado dos PEPs a fim de atualizar as informações sobre políticas, mesmo que não tenham sido solicitadas.

O PEP pode tomar decisões locais a partir do LPDP, se por exemplo, o meio de comunicação com o PDP falhar. Freqüentemente o LPDP informa suas decisões ao PDP, assim como o PEP aplica as mudanças das políticas impostas pelo PDP.

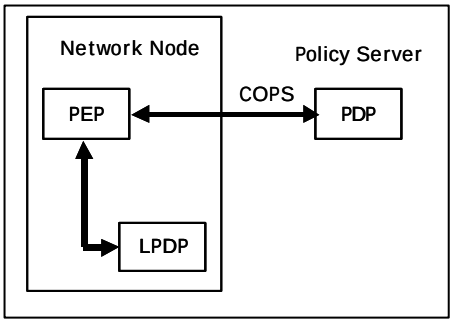

Figura 11 – Diagrama COPS

2.3. Requisitos de Qualidade de Serviço

Oferecer garantias de transmissão como forma de resolver o problema do encaminhamento do serviço de melhor esforço, não é o suficiente, é necessário expressar e definir alguns requisitos de QoS. A garantia de transmissão pode ser expressa como uma combinação dos parâmetros de QoS. As seguintes métricas básicas fazem parte de um conjunto de requisitos para redes que implementam mecanismos de QoS [46].

- Atraso fim-a-fim (*delay*): tempo necessário entre o envio de uma mensagem pela origem e a recepção pelo destino, ocorre no caminho da transmissão ou em um dispositivo da rede, como por exemplo, num roteador, neste caso é também conhecido como atraso de propagação, sendo a diferença entre os tempos de recepção e transmissão. Quanto maior o atraso, menor será a qualidade de serviço prestado pela rede;
- Variação do atraso (*jitter*): é a variação do atraso entre duas mensagens consecutivas durante uma transmissão, representando uma distorção nos tempos de transmissão e chegada. Algumas aplicações, como por exemplo, vídeo sob demanda, é sensível a variação de atraso, causando efeito indesejáveis.
- Largura de banda (*bandwidth*): é a taxa máxima teórica, medida de capacidade de transmissão de dados expressa em bits por segundo entre um transmissor e um receptor. A transferência de um montante de tráfego de dados oriundos de uma fonte para outra na rede é denominada vazão. Além da limitação física do meio de transmissão, esta pode ser limitada pela quantidade de fluxos concorrentes em um determinado trecho do caminho.
- Perda de pacotes (*packet loss*): representa o número de pacotes que foram transmitidos na rede, mas não alcançou o destino em um determinado período.
- Confiabilidade: pode ser visto como a taxa de ocorrência de erros no meio físico, como por exemplo, pacotes corrompidos, descartes, duplicação ou reordenamento de pacotes na rede.

A aplicação de garantias de qualidade de serviço a um fluxo de dados privilegiado, pode ser vista como oferecer um baixo atraso e variação de atraso, grande quantidade de banda e muita confiabilidade, em detrimento de outros fluxos na rede [33].

2.4. Necessidades das aplicações na Internet

Desde alguns anos que a Internet vem se estendendo dos laboratórios acadêmicos de pesquisa para a vida cotidiana. Cada vez mais está presente nos meios educacionais, nos comerciais, no entretenimento e em muitos outros ambientes. Através do modelo de serviço de melhor esforço tem-se de forma natural, acomodado milhões de usuários na rede, utilizando-se dos mais diferenciados serviços que este tem a oferecer.

Assim novas aplicações e necessidades emergiram, novos modelos de serviços foram propostos a fim de oferecer certas garantias de qualidade de serviço a essas demandas. Devido a essas novas necessidades, algumas aplicações (realidade virtual, videoconferência, tele-imersão, etc...) não podem ser acomodadas pelo modelo de serviço de melhor esforço.

Hoje, todo o tráfego na Internet é predominantemente TCP (web, ftp, etc.) com multimídia estática (imagens e animações) e aplicações de comunicação de áudio e vídeo-conferência, de pouca interatividade. Entretanto uma nova geração de aplicações avançadas com intuito de oferecer novas oportunidades e caminhos de comunicação e colaboração surgiu envolvendo interatividade, qualidades sensoriais, manipulações em tempo real, dados distribuídos, acessos a recursos remotos, compartilhamento de realidade virtual, tele-data, tele-imersão e outros.

Observa-se que QoS tem diferentes perspectivas em redes de computadores. Uma é voltada para o desempenho da rede relativo as aplicações, medidas quantitativamente, como largura de banda, atraso, variação do atraso, perda de pacotes, etc. Atua como resultado do ponto de vista da rede com o objetivo de garantir QoS através destas métricas ao mesmo tempo em que tenta maximizar a utilização da rede. Pelo outro lado

temos um conjunto de tecnologias que possibilita às redes oferecer garantias de desempenho, maximizando a utilidade das aplicações. Tendo como efeito a qualidade na habilidade da execução das tarefas requeridas pelas aplicações resultando em interesse pelos usuários.

Em certas ocasiões, QoS pode assumir diferentes interpretações por diferentes grupos de trabalho. Por exemplo, em rede, atraso pode expressar a quantidade de tempo que uma unidade de dados a atravessa por diferentes caminhos, para um desenvolvedor de aplicações de vídeo, este tempo é requerido para os parâmetros de *encoded/decoded*.

O conhecimento das necessidades de QoS das aplicações avançadas é muito importante em ambas às perspectivas. Como resultado temos o entendimento de como os serviços de rede podem tolerar a demanda das aplicações avançadas, e como as aplicações avançadas podem explorar as redes existentes ou novas, beneficiando-se de forma eficiente. Assim, o sucesso das aplicações avançadas na Internet, indica a necessidade direta da cooperação entre aplicação e rede [44].

Valores aproximados para os parâmetros de QoS para diferentes tipos de mídias são encontrados em [36][45], conforme Tabela 04.

Tabela 04 – Parâmetros de QoS das aplicações

Aplicações Típicas	Bandwidth (Kbps)	Delay (ms)	Jitter (ms)	Bit error rate
Vídeo	700...1500...6000	<200	15	10^{-4}
Áudio, Telefone	32	<200	20	10^{-4}
E-mail, Fax	10...64	-	-	10^{-6}
Internet, WWW	500	TCP time-out	TCP time-out	10^{-6}
Voz sobre IP	10...64	<200	15	-

2.5. Arquitetura *Chameleon*

Nos últimos anos, vêm sendo desenvolvidos mecanismos como forma de oferecer garantias de QoS fim a fim, maiores detalhes em [1][13][25]. Dentre estas, a arquitetura *Chameleon* se destaca apresentando um modelo inovador de negociação hierárquica de serviços.

A arquitetura *Chameleon* permite a oferta de serviços fim a fim, independente da tecnologia de QoS adotada entre a fonte e o destino na Internet. Para oferecer estas garantias a arquitetura apresenta as diferentes funções necessárias em três planos distintos, Figura 12.

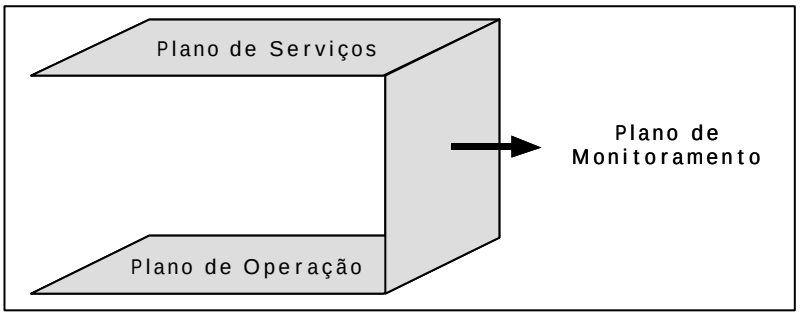

Figura 12 – Arquitetura Chameleon

O Plano de Serviços propicia a definição de um modelo abstrato da rede para que a interface externa de todos os domínios seja similar, para fins de definição e negociação de serviços, é implementada pela combinação de um modelo padronizado de SLS e de um modelo de negociação de serviços.

Tendo um papel fundamental na arquitetura, o plano de serviços é implementado por um Corretor de Serviços ou *Service Broker* (SB). Este elemento negocia os serviços com o domínio, verificando a disponibilidade de recursos a partir de medições obtidas em instantes regularmente espaçados. “O SB pode ser visto como uma extensão de um *Bandwidth Broker* (BB).” [13].

O Corretor de Serviços apresenta duas diferenças fundamentais em relação ao BB. Enquanto que o BB é utilizado especialmente em domínios *Diffserv*, o SB não está vinculado a nenhuma tecnologia de QoS. A outra diferença, é que o SB pode negociar serviços que necessitem de outros requisitos de QoS como atraso e variação do atraso em vez de somente da largura de banda, utilizada nos BB.

Outras tarefas de responsabilidade dos SB são: o controle de admissão e o estabelecimento de configurações para a coleta e transmissão das informações de tráfego.

No Plano de Operação, os domínios mapeiam os serviços negociados para algum mecanismo utilizado para o provisionamento de recursos e configuração dos equipamentos.

O Plano de Monitoração é ortogonal aos demais e executa a medição dos parâmetros de QoS, negociados a cada serviço, a realimentação das informações e possivelmente uma reação.

Como parte integrante do Plano de Serviços, temos o modelo de negociação de serviços consistindo de duas etapas, distintas e independentes. A primeira define a negociação entre os usuários e os provedores de serviços (ISPs), onde o usuário requisita um serviço através de um SLA, não resultando em nenhum tipo de provisionamento de recursos, apenas a intenção de adquirir os serviços.

A segunda, trata das negociações de serviços de transporte que são baseados nas requisições de vários provedores de serviços. Tem por objetivo, efetuar as configurações necessárias e negociar os serviços de transporte.

Como resultado dessa etapa, temos o provisionamento de recursos em diversos domínios, atendendo as requisições de serviços fim a fim, conferindo flexibilidade na negociação dos contratos pelos provedores de serviços.

Enquanto que o modelo bilateral proposto no contexto do BB [17], apresenta uma negociação em cascata, onde cada domínio negocia os serviços com os outros domínios vizinhos, através de algum protocolo de negociação, o modelo hierárquico propõe a agregação de vários domínios em áreas de atuação, coordenados por uma entidade central denominada Central de Serviços (*Service Exchange – SE*).

O novo modelo de negociação da arquiterura *Chameleon*, sugere que todas as negociações de serviços ocorram através dos SEs, realizando reservas antecipadas, além de manter todas as informações necessárias de negociação para os domínios subordinados. A permissão ou negação de

serviços se dá com a troca de informações entre os SB e os SE, e entre SE superiores conforme ilustra a Figura 13.

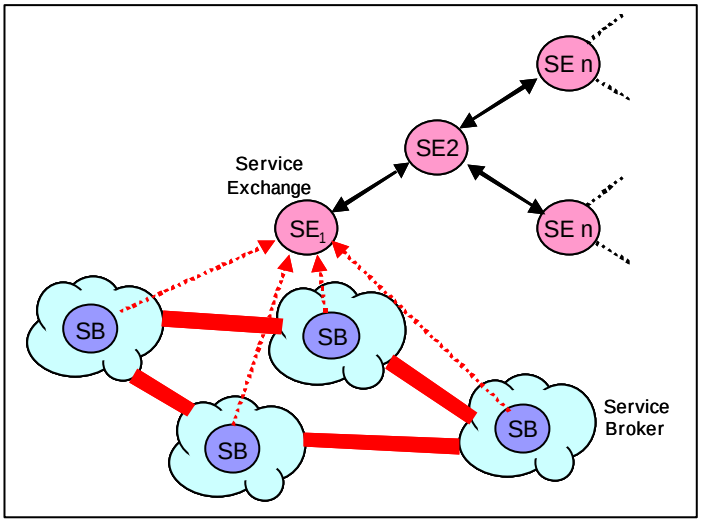

Figura 13 – Negociação Hierárquica

O processo de negociação hierárquica inicia quando os domínios determinam quais os serviços desejam adquirir, informando aos SB do domínio. Os SB enviam informações requisitando os serviços para o SE. O SE superior, calcula quais requisições serão atendidas e quais recursos serão reservados. Por fim, os SE informam aos domínios os serviços concedidos e realizam as reservas antecipadamente até o domínio destino da transmissão.

"Esse modelo permite escalabilidade e eficiência na negociação de serviços." [13]. A escalabilidade refere-se a impossibilidade de gerar grande quantidade de mensagens de sinalização, uma vez que, dentro de sua área de atuação, o SE sempre recebe requisições para tráfego agregado, negociando com outros SE superiores. A eficiência é resultado do conhecimento que o SE tem dos domínios participantes, efetuando negociações a partir de vários critérios de QoS.

Durante o processo de negociação, o SE procura alcançar os seguintes objetivos:

- Atendimento do maior número de solicitações possíveis de serviços;

- Os serviços fim a fim não devem ficar indisponíveis para os usuários finais por falta de serviço;
- Eficiência na implementação;
- Justiça para os clientes e provedores.

Uma vez que o plano de serviços realize a negociação procurando a disponibilidade ou não de recursos fim a fim entre domínios, faz-se necessário implementar um mecanismo de monitoramento e controle, atuando no plano de operação, com roteadores e tráfego real, avaliando e configurando o provisionamento de tráfego, complementando a arquitetura *Chameleon*.

2.6. Trabalhos Relacionados

O trabalho desenvolvido em [6] apresenta métodos para prover alocação de capacidade fim a fim para conexões *Virtual Private Networks* em um simples domínio. Foi implementado um *Bandwidth Broker* (BB) para gerenciar os serviços com seus *Internet Service Providers*. Utilizou-se a configuração de roteadores para demonstrar o estabelecimento de túneis dinâmicos, habilitando QoS, a partir dos BBs.

Através de [10] foram propostos mecanismos escaláveis para realizar o provisionamento dinâmico e controle de admissão interconectando redes Intserv e Diffserv como forma de garantir os serviços usando um agente chamado de *Bandwidth Management Point* (BMP).

Extendendo o tradicional modelo de serviço point-to-point para point-to-set em [24], é proposta uma arquitetura de qualidade de serviço baseado em roteadores de borda em redes privadas. O modelo point-to-set permite a um usuário ter um "*pool*" de serviços, os quais podem ser encaminhados a qualquer destino. Este modelo introduz o *bandwidth tracking* nos roteadores da rede e esquemas de provisionamento.

Outra perspectiva diferente dos serviços diferenciados é apresentada em [48]. Este trabalho apresenta o conceito de diferenciação proporcional, que em vez de oferecer absolutas garantias de largura de banda e atraso para classes de pacotes, oferece serviços relativos partindo da premissa que

o serviço recebido por uma classe de alta prioridade será no mínimo melhor do que uma com baixa prioridade. Neste modelo, a rede não implementa controle de admissão nos roteadores de borda e depende dos mecanismos de filas para criar diferenciação entre diversas classes de pacotes.

Outros trabalhos desenvolvidos para suportar provisionamento e controle de admissão entre diferentes soluções podem ser encontrados em [21][22][26].

2.7. Considerações Finais sobre este Capítulo

Neste capítulo foram apresentados os modelos de qualidade de serviço, as métricas básicas de QoS e as necessidades das aplicações avançadas na Internet. A Arquitetura *Chameleon* foi introduzida com o modelo inovador de negociação hierárquica de serviços. No próximo capítulo serão examinados os mecanismos de controle de filas, responsáveis pelo controle de recursos e identificação dos fluxos de tráfego na rede.

3. Mecanismos de Controle de Filas

Neste capítulo, serão apresentados os mecanismos de escalonamento, controle de tráfego e policiamento.

3.1. Mecanismos de escalonamento, controle de congestionamento, e policiamento

No contexto de qualidade de serviço, existem mecanismos que são responsáveis pelo ajuste de parâmetros que disciplinam determinadas funções, são eles:

- **Mecanismos de escalonamento**: procuram garantir a diferentes fluxos de pacotes a obtenção dos recursos alocados na rede, ordenando o tráfego, utilizando métodos de priorização de encaminhamento (FIFO, PQ, CBQ e WFQ);
- **Mecanismos de controle de congestionamento**: inibem fluxos de pacotes durante um período de forma que as fontes dos fluxos reduzam a sua carga sobre a rede, reduzindo o nível de congestionamento (RED, RIO e WRED);
- **Mecanismos de policiamento**: são responsáveis por identificar fluxos de tráfego controlando e separando em fluxos individuais a fim de modelá-los a uma determinada política (LB, TB, TSW, srTCM e trTCM).

3.1.1. Mecanismos de Escalonamento

3.1.1.1. Escalonamento FIFO

No mecanismo de escalonamento FIFO (First In, First Out), o primeiro que entra é o primeiro que sai. Os pacotes são armazenados quando um congestionamento é encontrado e em seguida são encaminhados pela ordem de chegada quando não houver mais congestionamento. À medida que os pacotes entram na fila de entrada, são colocados na fila de saída na mesma ordem em que foram recebidos.

O escalonador FIFO é considerado como um primeiro mecanismo no controle do tráfego, em muitos casos é o algoritmo padrão, entretanto há a necessidade de algoritmos mais robustos. Este algoritmo não efetua nenhuma decisão sobre a prioridade dos pacotes. A ordem de chegada

determina tanto a largura de banda obtida como a alocação de buffers nos roteadores.

A Figura 14 mostra que, na medida que os pacotes vão entrando na fila pela interface de entrada, o escalonador enfileira os pacotes e, em seguida, coloca-os na fila da interface de saída, na mesma ordem em que foram recebidos.

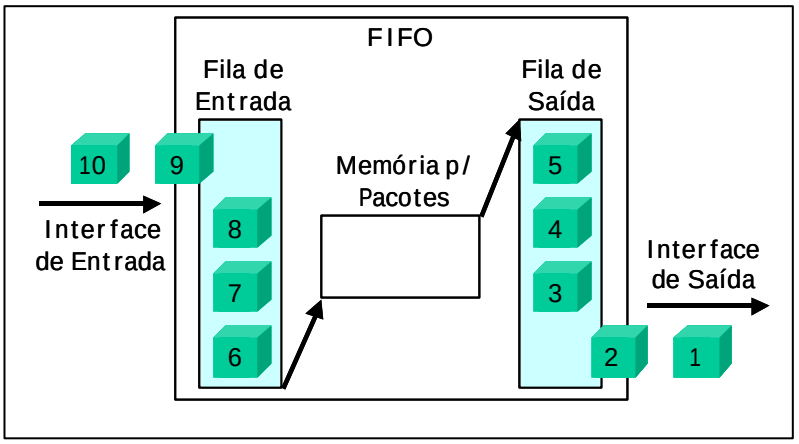

Figura 14 – Escalonador FIFO

Uma principal vantagem deste escalonador refere-se a eficiência de assegurar que tráfegos em rajada de curta duração não causem descarte de pacotes. O retardo de pacotes na fila permanece insignificante, em relação ao tempo da transmissão, para filas de pequeno tamanho. Para filas de tamanho maior, causadas por fontes de tráfego em rajadas de longa duração, os atrasos de enfileiramento são contínuos. Nas aplicações sensíveis ao tempo, um aumento da carga da rede, faz com que as filas de espera tornem-se cheias por longos períodos degenerando o escalonador, ocorrendo o descarte de pacotes [49].

3.1.1.2. Escalonamento por prioridade (PQ)

O mecanismo de escalonamento por prioridade (*Priority Queuing* – PQ) é baseado na priorização de tráfegos, onde os pacotes são recebidos pela fila de entrada e reordenados, a partir dos critérios definidos de prioridade, e só então colocados na fila de saída antes dos pacotes de baixa prioridade.

A Figura 15 mostra que os pacotes são recebidos pela interface de entrada. Em seguida são reordenados a partir de critérios predefinidos, estabelecendo prioridades de encaminhamento, e por fim colocados na interface de saída. Os pacotes de alta prioridade são colocados na fila de saída antes dos de baixa prioridade.

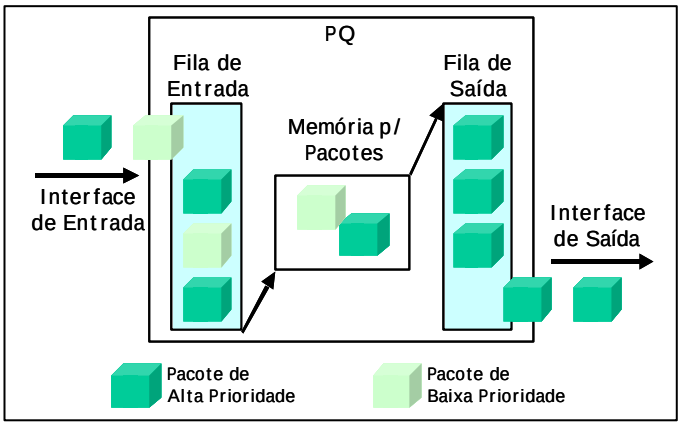

Figura 15 – Escalonador PQ

O desempenho no encaminhamento dos pacotes deste escalonador tem relacionamento direto no processamento interno dos roteadores causado pela análise, manipulação e reordenação dos pacotes. À medida que o volume do tráfego de alta prioridade aumenta, o tráfego de baixa prioridade tende a ser descartado por insuficiência de *buffers* nos roteadores.

Neste escalonador, a latência induzida pelo impacto de espera do tráfego nas filas por extensos períodos, pode causar, em aplicações sensíveis ao tempo, o não funcionamento destas aplicações. Os protocolos de roteamento, que necessitam de confirmação dentro de um período determinado, também sofrem com esta latência para períodos demasiadamente extensos. É o que comumente chamamos de *timeout*. Este escalonador não possui escalabilidade para fornecer, em redes com grande volume de tráfego, bom desempenho.

3.1.1.3. Escalonamento baseado em classes

O Escalonamento Baseado em Classes (CBQ) é uma extensão do escalonador por prioridades. Neste escalonador podem ser definidas a

quantidade de tráfego enfileirado, a quantidade de filas de saída e as preferências em que cada fila será servida. O CBQ permite que várias aplicações, com especificações mínimas de largura de banda ou latência, compartilhem a mesma rede [61].

A utilização desse escalonador provê garantias de largura de banda em um determinado ponto de congestionamento, assegurando ao tráfego especificado uma parte fixa disponível enquanto a banda remanescente é assegurada para a outra parte restante do tráfego.

Como forma de evitar a escassez de *buffers* nos roteadores, sem que haja uma significativa latência induzida, o escalonador categoriza e classifica o tráfego em diversas filas de enfileiramento fazendo com que se diminua o tempo de espera de transmissão em cada fila. O escalonador manipula o tráfego assinalando um espaço na fila para cada classe de pacotes, servindo a fila na modalidade *round-robin* (fila circular que encaminha um pacote de cada fila interna, separadas por classes e prioridades).

A Figura 16 apresenta um escalonador CBQ com diferentes *buffers*. O roteador pode ser configurado para servir os *buffers* com diferentes prioridades (alto, médio e baixo) em cada rotação. Após o processamento do tráfego em cada fila, os pacotes continuam sendo atendidos até que o contador exceda o limite configurado ou a fila esteja vazia. O tráfego então é encaminhado para a interface de saída para a transmissão.

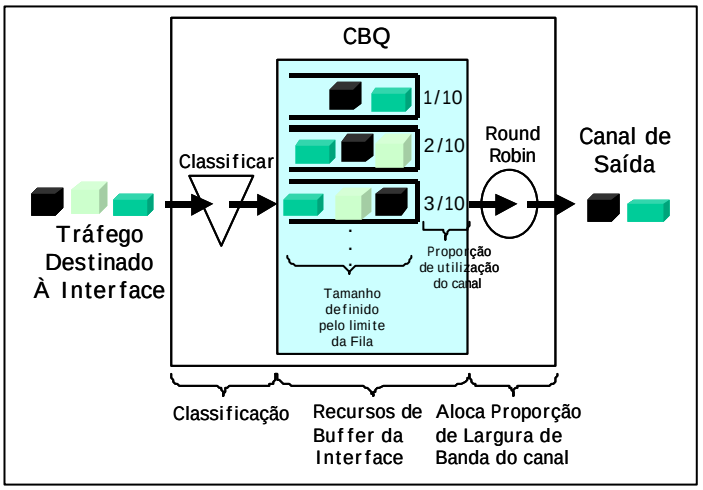

Figura 16 – Escalonador CBQ

Enquanto que no escalonador por prioridades as filas de alta prioridade detêm um modelo absoluto de serviços e falta total de recursos para as outras filas, o escalonador baseado em classes provê um modelo de serviços mais igualitário com um aumento de recursos para as filas de alta prioridade e uma diminuição relativa para as filas de baixa prioridade.

O escalonador CBQ é utilizado na implementação de tecnologias que fornecem compartilhamento de comunicação para classes de serviços tornando-se um método mais eficiente no gerenciamento de recursos de filas.

Assim como no escalonador PQ, o escalonador CBQ não apresenta escalabilidade devido ao processamento interno dos roteadores, responsáveis pela classificação, reordenação e gerenciamento das filas, em transmissão de alta velocidade. Embora forneça mecanismos que provê categorização de classes de serviços, com um bom desempenho, em transmissões de baixa velocidade.

3.1.1.4. Escalonamento WFQ

O escalonador WFQ (*Weighted Fair Queuing*) possui comportamento previsível na entrega de pacotes e assegura a existência de *buffers* para fluxos de tráfego nos roteadores.

O escalonador provê aos fluxos de baixo volume de tráfego tratamentos preferenciais, transmitindo toda a carga em uma porção do tempo, e permite aos fluxos de alto volume o restante da capacidade de enfileiramento, compartilhando proporcionalmente entre eles, como ilustrado na Figura 17.

Uma das principais funções do WFQ é minimizar o esforço de configuração e adaptar-se automaticamente às mudanças nas condições de tráfego [61].

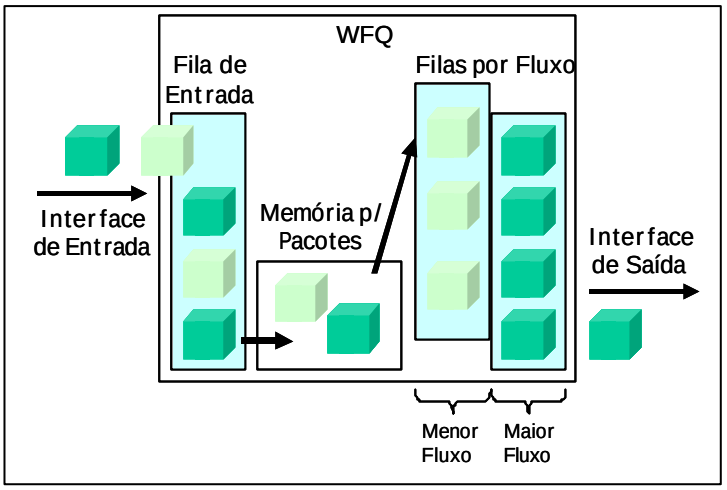

Figura 17 – Escalonador WFQ

O escalonador WFQ apresenta algumas características dos escalonadores PQ e CBQ. É eficiente na medida que pode utilizar a largura de banda disponível para transmitir o tráfego de baixa prioridade se não existir tráfego de alta prioridade. Por outro lado tem problemas de escalabilidade, como em PQ e em CBQ. Com a finalidade de prover equidade para todos os fluxos, o WFQ utiliza-se de pesos como forma de fazer justiça protegendo fluxos de baixo volume de tráfego sobre os outros. Entretanto há falta de controle do mecanismo no ajuste e avaliação dos parâmetros de comportamento dos fluxos sobre outros de prioridades diferentes visto que é definido estaticamente na implementação.

3.1.2. Mecanismos de Controle de Congestionamento

3.1.2.1. RED

O mecanismo RED (*Random Early Detection*) é uma função aleatória para o descarte de pacotes, implementada nos roteadores, com a finalidade de monitorar o nível de congestionamento na rede. Através do RED é efetuada uma monitoração do tamanho da fila no roteador e caso apareça sinais de congestionamento iminente, as fontes de tráfego são notificadas a descartar seus pacotes.

Os roteadores com esse mecanismo descartam pacotes mais cedo que outros roteadores convencionais, fazendo com que haja redução na taxa de transmissão de algumas fontes. Estas por sua vez, reduzem suas janelas de congestionamento, também, mais cedo, evitando um descarte maior de pacotes posteriormente.

No RED, os roteadores em vez de esperar que a fila se torne completamente cheia, o que implicaria no descarte de todos os pacotes, que chegam, utilizando por exemplo FIFO, ele decide descartar cada novo pacote, com uma certa probabilidade, sempre que o tamanho da fila exceda um determinado nível.

Com a finalidade de monitorar o tamanho da fila, algumas variáveis foram propostas em [40]. A partir de uma fila não vazia, o RED, computa em *avglen* o tamanho médio da fila utilizando uma média ponderada por uma variável q_{weight} . Esta determina a tolerância do roteador RED às rajadas de longa duração, quanto maior for q_{weight} maior será o peso do tamanho instantâneo da fila no cálculo do valor do *avglen*. Caso a fila esteja vazia, *avglen* é gradualmente diminuído em função do tempo de ociosidade da fila.

O RED utiliza os patamares min_{th} (*Minimum Threshold*) e max_{th} (*Maximum Threshold*) para determinar a probabilidade de descarte do pacote.

Enquanto o tamanho médio da fila, *avglen*, permanece abaixo de min_{th}, todos os pacotes que chegam no roteador são aceitos na fila. Quando *avglen* está entre os dois patamares, há uma probabilidade *p* do novo pacote ser descartado, onde *p* é uma função de *avglen* e do tempo decorrido desde o descarte do último pacote. Se *avglen* ultrapassa o limite max_{th}, todos os próximos pacotes serão descartados até que a média de ocupação da fila seja reduzida.

Como a probabilidade do roteador RED de descartar pacotes é medida proporcionalmente a parcela da ocupação da fila do roteador, quanto mais pacotes, de um fluxo, chegarem no roteador, maior será a probabilidade que um de seus pacotes sejam descartados. A probabilidade de descarte correspondente a max_{th} é determinada pelo parâmetro P_{max}. A agressividade de descarte de pacotes do mecanismo RED é proporcional ao valor de P_{max} [40][62].

A Figura 18 ilustra o funcionamento do algoritmo RED com base nos parâmetros descritos.

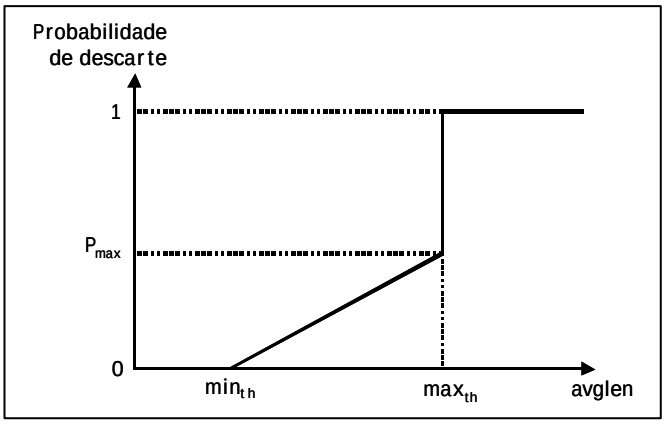

Figura 18 – Probabilidade de descarte do RED

Existem três fases de probabilidade de descarte no mecanismo RED: operação normal, congestionamento iminente e controle de congestionamento. Estas três fases são definidas pela média do tamanho da fila (parâmetro *avglen*), e encontram-se entre os intervalos [0, min_{th}), [min_{th}, max_{th}) e [max_{th}, ∞), respectivamente.

Na fase de operação normal, o roteador está operando sem congestionamentos, a quantidade de pacotes que estão chegando é igual a capacidade do roteador em processá-los. A média do tamanho da fila permanece pequena e não há descarte de pacotes.

Na fase de congestionamento iminente, o roteador observa que a fila gradualmente cresce com o aumento da carga da rede. O mecanismo inicia o descarte de pacotes randomicamente, evitando por exemplo o problema de sincronização global que ocorre no protocolo TCP. Como sinal de congestionamento, para as fontes de tráfego diminuírem suas taxas de transmissão. Nesta fase o algoritmo RED reage mais rapidamente ao detectar a iminência de um congestionamento e descarta alguns pacotes mais cedo.

Na fase de controle de congestionamento, o roteador encontra-se congestionado, com o estouro de uma fila FIFO, todos os pacotes que chegam são descartados. Todas as fontes dos fluxos, neste momento, retraem suas taxa de transmissão ao detectarem a perda de seus pacotes,

causando uma sub-utilização da rede. No protocolo TCP este problema é conhecido como sincronização global, devido ao fato das conexões tentarem recuperar seus pacotes usando mecanismo internos de retransmissão, ocorrendo assim, um sincronismo na retração das fontes de tráfego com períodos de ociosidade seguidos de períodos com congestionamento.

Na escolha dos parâmetros do mecanismo RED, deve-se considerar a caracterização da carga da rede que influência diretamente o tamanho da fila. O tráfego que possui muitas rajadas, *minth* deve ser suficientemente grande para manter uma alta utilização do enlace. A diferença entre os patamares deve ser maior do que o incremento no tamanho médio da fila, caso contrário, não haveria tempo suficiente para as fontes detectarem o descarte com antecedência dos pacotes. Uma boa aproximação é usar *maxth = 2 * minth* dada à diversidade de aplicações atualmente encontrada na Internet [19].

Os pacotes de diferentes fluxos de tráfegos, podem ser marcados como pertencentes a classes distintas. Como já discutido, o serviço AF em ambiente *Diffserv* possui quatro classes de tráfego com até três níveis de precedência (green, yellow, red). Em caso de congestionamentos, os pacotes marcados com baixa precedência (red) são descartados em preferência aos de alta precedência (green, yellow).

Para cada classe pode-se calcular o tamanho médio da fila (*avglen*) de maneira diferente, bem como estabelecer um conjunto de parâmetros (min_{th}, max_{th}, P_{max}) distintos. Em [63], variações do RED são classificadas em quatro categorias como mostra a Figura 19. As quatro categorias são as seguintes:

- Única Média - Único Limiar (SAST) – uma média é calculada para todas as classes, e os parâmetros (min_{th}, max_{th}, P_{max}) são utilizados para todos os pacotes de todas as cores. O algoritmo RED é um exemplo para esta categoria.
- Única Média - Múltiplos Limiares (SAMT) – também uma média é calculada para todas as classes, baseado no número de pacotes da fila, e diferentes parâmetros são utilizados para cada classe de precedência, pode ser exemplificado pelo WRED.

- Múltiplas Médias - Único Limiar (MAST) – nesta categoria é usada um conjunto de parâmetros para todas as classes e várias médias para cada precedência.
- Múltiplas Médias - Múltiplos Limiares (MAMT) – é a mais genérica de todas, o cálculo do tamanho da fila deve ser efetuado para cada nível de precedência nas diferentes classes. Para a classe de maior prioridade o cálculo é feito com o número de pacotes da sua classe, e para classes de menor prioridade são considerados o número de pacotes das classes de maior prioridade e também o número de pacotes desta classe. Classes com menor prioridade consideram todos os pacotes na fila para o cálculo do tamanho médio da fila (*avglen*). Um exemplo de MAMT é o mecanismo RIO que utiliza dois conjuntos de parâmetros e duas médias de ocupação de filas [2]. Esse mecanismo está descrito a seguir.

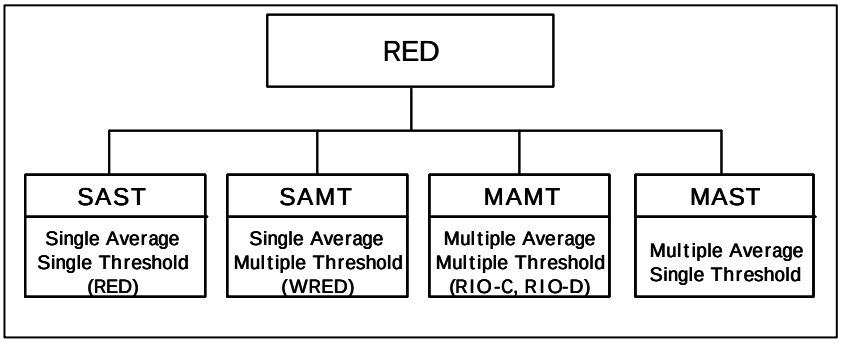

Figura 19 – Variações do RED

3.1.2.2. RIO

O mecanismo de controle de congestionamento RIO (*RED with IN and OUT*) identifica uma diferenciação de serviços através de classes de fluxos marcados pelo campo DS. Através deste mecanismo os pacotes de um fluxo que estejam em conformidade com um perfil de serviço contratado podem ser marcados como *IN* (*in-profile*) e os pacotes que estejam fora do perfil de serviço contratado podem ser marcados como *OUT* (*out-of-profile*). Assim, o gerenciamento das filas nos roteadores internos ao domínio, pode ser feito

com a utilização de dois algoritmos RED, para cada perfil de serviço, tanto para pacotes marcados com *IN* como para os pacotes marcados com *OUT*.

Com o objetivo de reduzir os efeitos do congestionamento evitando o descarte de pacotes *IN*, o mecanismo RED para pacotes *OUT* é configurado de forma mais agressiva visando o descarte prioritário destes pacotes. Isto permite ao mecanismo RIO um tratamento diferenciado entre classe de fluxos marcados com *IN* e *OUT*.

Semelhante ao RED, veja Figura 20, na contabilização do tamanho médio da fila, o mecanismo RIO calcula o tamanho médio da fila para pacotes *IN* (*avgin*) e também para os pacotes *OUT*, sendo que neste último, o cálculo do tamanho médio da fila é feito considerando tanto a quantidade de pacotes *IN* como a quantidade de pacotes *OUT* (*avgtotal*). A probabilidade no descarte de pacotes *IN* e pacotes *OUT* é função de *avgin* e *avgtotal*, respectivamente. Para pacotes marcados com *IN* definem-se os parâmetros min_{IN}, max_{IN} e P_{maxIN} e de forma análoga min_{OUT}, max_{OUT} e P_{maxOUT} correspondendo aos pacotes marcados com OUT.

Existem cinco fases de probabilidade de descarte no mecanismo RIO: operação normal, sensível congestionamento, congestionamento tolerante, alarme de congestionamento e controle de congestionamento [62].

Na fase de operação normal, intervalo [0, min_{OUT}), o roteador RIO está operando sem congestionamentos, a quantidade de pacotes IN e OUT que está chegando é igual a capacidade do roteador em processá-los. A média do tamanho da fila permanece pequena e não há descarte de pacotes.

Na fase de sensível congestionamento, intervalo [min_{OUT}, max_{OUT}), o roteador RIO observa que as filas gradualmente crescem com o aumento da carga da rede. O mecanismo inicia somente o descarte de pacotes OUT, randomicamente. Durante esta fase observa-se que os pacotes IN apresentam-se em pequenas filas instantâneas, estes pacotes nunca são descartados.

Na fase de congestionamento tolerante, intervalo [max_{OUT}, min_{IN}), todos os pacotes OUT são descartados e observa-se que as filas de pacotes IN crescem gradualmente com o aumento da carga da rede

Na fase de alarme de congestionamento, intervalo [min_{IN}, max_{IN}), todos os pacotes OUT continuam sendo descartados e o roteador RIO inicia, também, o descarte de pacotes IN.

Na fase de controle de congestionamento, intervalo [max_{IN}, ∞), o roteador encontra-se congestionado, tanto pacotes OUT como pacotes IN são descartados com probabilidade 1. Se o roteador RIO constantemente operar nesta fase, significa que o sistema foi mal provisionado ou os parâmetros dos condicionadores de tráfego do mecanismo RIO não foram configurados corretamente.

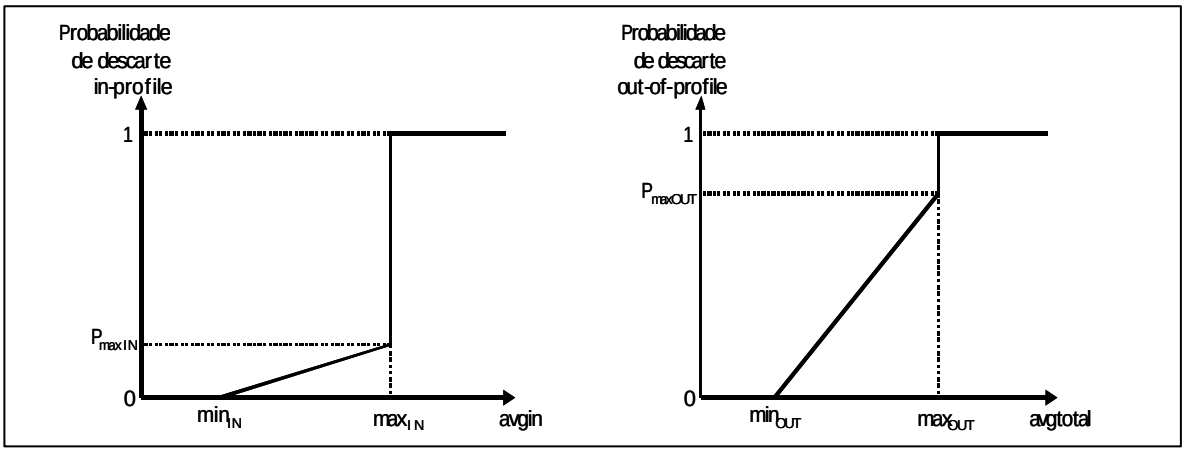

Figura 20 – Probabilidade de descarte do RIO

3.1.2.3. WRED

Um outro mecanismo de controle de congestionamento que combina as funcionalidades do RED e baseia-se na precedência de pacotes IP, é o WRED (*Weighted Random Early Detection*). Este mecanismo é uma implementação da Cisco [67, 68] e funciona descartando pacotes seletivamente, inicialmente descarta os pacotes de menor prioridade, com diferentes pesos para cada classe de serviço. Pacotes com alta precedência de descarte são descartados antecipadamente do que os pacotes de baixa precedência. Entretanto, pode-se desabilitar o descarte baseado na classificação da precedência dos pacotes IP, e habilitar o descarte com base apenas no tamanho da ocupação das filas nos roteadores.

O WRED é utilizado em qualquer interface de saída de pacotes na qual há possibilidade de congestionamentos eminentes. Geralmente, é utilizado nos roteadores internos do domínio (*core routers*). Os roteadores de

borda (*edge routers*) marcam e habilitam a precedência dos pacotes na entrada do domínio.

O WRED utiliza as precedências para determinar como são tratados os diferentes tipos de tráfego. Assim, ele detém de variáveis que identificam o nível de probabilidade de descarte (*thresholds*) e os pesos (*weighted*) para as diferentes precedências, permitindo diferentes níveis de qualidade de serviço. Os pacotes pertencentes ao tráfego de melhor esforço devem ser descartados mais freqüentemente do que os pacotes pertencentes ao tráfego do tipo Premium, durante os períodos de congestionamentos.

Semelhante ao RED o WRED utiliza os patamares min_{th} (*Minimum Threshold*) e max_{th} (*Maximum Threshold*) para determinar a probabilidade de descarte do pacote para cada nível de precedência de descarte. Quando a média de ocupação da fila ultrapassa o min_{th}, o RED inicia o descarte de pacotes, Figura 21. A taxa de descarte de pacotes cresce linearmente com o aumento do tamanho médio da fila até alcançar o max_{th}. Como a probabilidade do RED de descartar pacotes é medida proporcionalmente a parcela da ocupação da fila do roteador, quanto mais pacotes, de um fluxo, chegarem no roteador, maior será a probabilidade que um de seus pacotes sejam descartados. Quando a média de ocupação da fila é maior que max_{th}, então todos os pacotes daquela classe de serviço, são descartados.

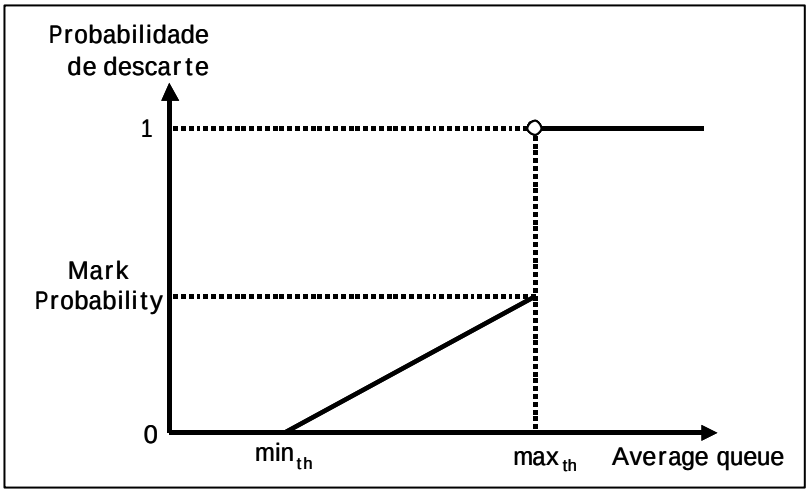

Figura 21 – Probabilidade de descarte do WRED

O valor do min_{th} deve ser maior o bastante para maximizar a utilização do *link*. Se o valor do min_{th} for pequeno, pacotes podem ser descartados

desnecessariamente, sem o uso completo do *link*. Outro dado importante é que, a diferença entre o min_{th} e o max_{th} devem também ser grande o bastante para evitar a sincronização global explicada no capitulo anterior.

3.1.3. Mecanismos de Policiamento

3.1.3.1. Método Leaky Bucket (LB)

Também conhecido como método do balde furado. A semelhança consiste no funcionamento do algoritmo que tem como base um balde com um pequeno furo embaixo. À medida que se coloca água no balde um fluxo de saída é gerado no fundo do balde, a velocidade com que a água entra no balde não tem influência no fluxo de saída. Quando o balde estiver cheio, a água que entrar nele irá escorrer pelos lados e será perdida.

No método LB os pacotes são representados pela água que entra e sai no balde. Cada *host* é conectado a rede por meio de uma interface que contém uma fila interna finita (balde furado). O *host* pode inserir um pacote a cada pulso de relógio da rede, transformando um fluxo irregular de pacotes em um fluxo regular, suavizando rajadas e reduzindo a ocorrência de congestionamento. Se um pacote chegar e a fila estiver cheia, ele será descartado, veja Figura 22.

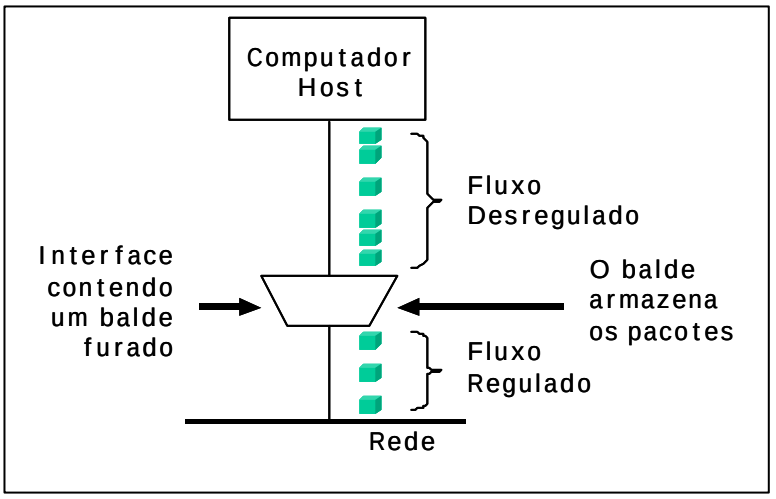

Figura 22 – Método Leaky Bucket

O método *leaky bucket* é eficiente para pacotes de mesmo tamanho, entretanto, se forem utilizados pacotes de tamanho variável, a melhor opção é permitir um número fixo de *bytes* por pulso, em vez de apenas um pacote. O tamanho do balde e a taxa de transmissão geralmente são configuráveis pelo usuário e medido em bytes.

3.1.3.2. Método *Token Bucket* (TB)

Enquanto o método do balde furado fornece um fluxo constante na interface de saída, independentemente da irregularidade do tráfego, o método do balde de *tokens* permite que a interface de saída aumente a sua taxa de transmissão quando chegarem tráfegos de rajadas. Este método gera *tokens* para cada ΔT segundos.

Para os pacotes serem transmitidos é necessário que eles capturem e destruam um *token*. O algoritmo descarta *tokens* quando o balde está cheio, mas nunca descarta pacotes, como ilustrado na Figura 23. O método do baldo de *tokens* permite rajadas em pequenos intervalos de tempo [49]. Uma maneira de regular o tráfego na rede, seria utilizar o método do balde furado após o balde de *tokens*.

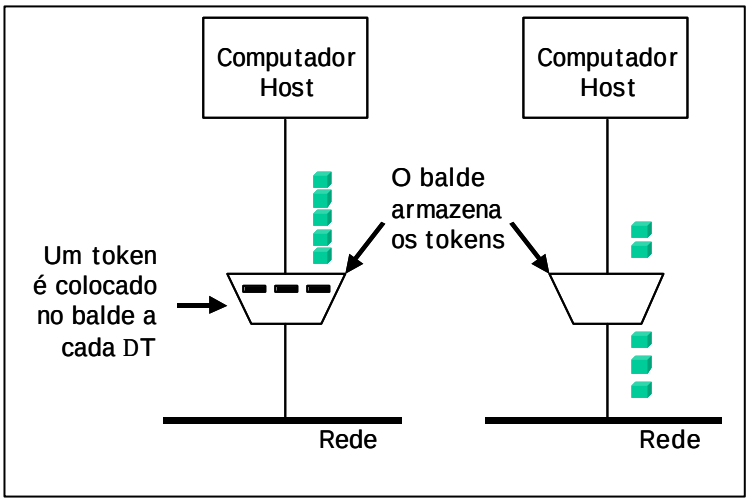

Figura 23 – Método Token Bucket

3.1.3.3. Método TSWTCM

O método TSWTCM (*Time Sliding Windows Three Colour Marker*) é um componente condicionador de tráfego usado nos roteadores de borda de um domínio *Diffserv* que marca pacotes *IN* e *OUT* de acordo com um serviço especificado. Designado para tratar pacotes de fluxos AF correspondendo aos três níveis de precedência (*red*, *yelow* ou *green*).

Possui dois componentes básicos que são: um *Rate Estimator* que calcula as taxas de envio de pacotes em um certo período de tempo e o *Marker* associado a uma cor (*drop precedence*), que marca os pacotes baseados nas taxas do *Rate Estimator*, veja Figura 24.

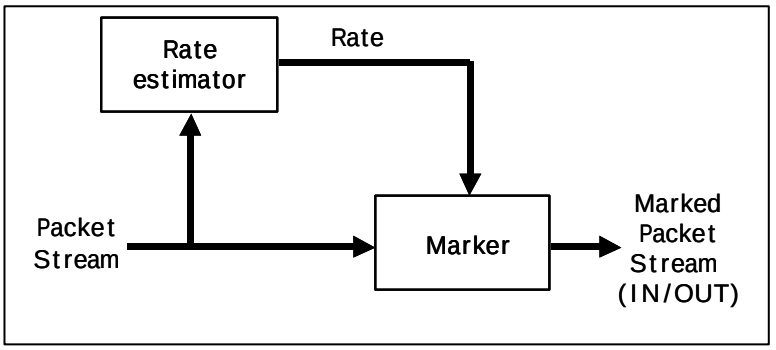

Figura 24 – Diagrama TSWTCM

No mecanismo TSWTCM, a marcação é baseada na medida de desempenho dos fluxos de tráfego comparados a duas variáveis: CTR (*Committed Target Rate*) e PTR (*Peak Target Rate*). Os pacotes com taxas abaixo ou igual a CTR são marcados com *green colour*; pacotes com taxas entre CTR e PTR são marcados com *yellow colour*; e pacotes com excesso de taxa são marcados com *red colour*.

A Tabela 05 mostra o algoritmo TSW *Rate Estimator*. Neste método há somente a necessidade de executar as quatro linhas do código descrito para cada pacote. O algoritmo TSW mantém três variáveis locais: *AVG_INTERVAL* que armazena historicamente as medidas em unidades de tempo e é a única variável do método a ser configurada; *avg-rate* que calcula a taxa estimada de chegada dos pacotes; e *t-front* que é o tempo de chegada do último pacote.

Tabela 05 – Algoritmo TSW Rate Estimator

```
Initially:
        AVG_INTERVAL    =  a Constant;
        avg-rate        =  CTR;
        t-front         =  0;
Update variables:
        Bytes_in_win    =  avg-rate * AVG_INTERVAL;
        New_bytes       =  bytes_in_win + pkt_size;
        avg-rate        =  New_bytes/ (now – t-front + AVG_INTERVAL);
        t-front         =  now;
Where:
        now             =  time of current packet arrival
        pkt_size        =  packet size in bytes
        avg-rate        =  measure arrival rate of traffic stream
        AVG_INTERVAL    =  time window over which history is kept
```

3.1.3.4. Métodos srTCM e trTCM

Os métodos srTCM (*Single Rate Three Color Marker*) e trTCM (*Two Rate Three Color Marker*) são condicionadores de tráfego, também, usados em redes *Diffserv*. Estes métodos marcam os pacotes de acordo com um serviço especificado, efetuando medições nos fluxos de tráfego. Utiliza os três níveis de precedência (*red*, *yelow* ou *green*), similar ao método TSWTCM.

Na Figura 25, apresentamos os componentes de ambos os métodos, srTCM e trTCM

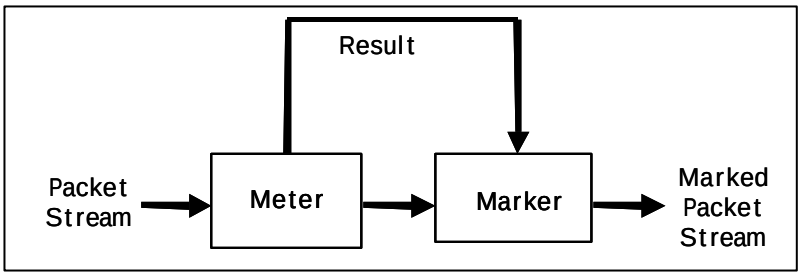

Figura 25 – Diagrama srTCM e trTCM

O componente *Meter* – calcula a taxa de chegada de cada pacote do fluxo de tráfego. Enquanto que no método srTCM, as medidas são calculadas na base de dois *token bucket*, CBS (*Committed Burst Size*) e EBS (*Excess Burst Size*), nas quais ambas compartilham a mesma taxa CIR (*Committed*

Information Rate). O método trTCM utiliza duas taxas, PIR (*Peak Information Rate*) e CIR (*Committed Information Rate*), associadas a dois *token bucket,* PBS (*Peak Burst Size*) e CBS (*Committed Burst Size*), respectivamente.

O componente *Marker* – marca os pacotes de acordo com um serviço especificado utilizando os níveis de precedência. Para o método srTCM os pacotes são marcados com *green colour* se não exceder a taxa CBS, *yellow colour* se excede CBS e não excede EBS, e *red colour* para os outros casos.

No método trTCM os pacotes que excedem a taxa PIR são marcados com *red colour*, *yellow colour* se excedem a taxa CIR e *green colour* se não a excedem.

3.2. Considerações Finais sobre este Capítulo

Neste capítulo foi destacada a importância dos mecanismos de controle de filas, examinando suas características e aplicabilidades. Desta forma, por ser tratar de um assunto complexo no controle do tráfego, a adoção de um mecanismo ou de outro, sempre que possível, se dará pela medição e analise comparativa dos métodos utilizados nestes mecanismos, resultando naquela que melhor se adapte a cada situação. No próximo capítulo será apresentado um mecanismo de provisionamento de tráfego, com o propósito de atender requisições de alocação de recursos na rede a partir de regras definidas em contratos de serviços pré-estabelecidos.

4. Especificação do Mecanismo de Provisionamento

Neste capítulo, será apresentada a especificação do mecanismo de provisionamento para serviços diferenciados.

4.1. Provisionamento e Controle de Admissão

Provisionamento em geral significa fornecer recursos para atender uma determinada aplicação. Atualmente, todo o tráfego da Internet é provido tão rapidamente quanto possível, isto é, o tráfego transmitido é tratado sem nenhuma distinção ao longo do caminho, limitado a uma única classe de serviço, o melhor esforço. Esta classe de serviço transporta todo o tráfego para todas as aplicações. Entretanto, algumas aplicações avançadas (*VoIP*, *Video Streamed*, *etc*) tem restrições para com os parâmetros necessários de QoS na rede. Algumas abordagens sobre diferentes níveis de serviços foram apresentadas no Capitulo 2.

Provisionamento em QoS refere-se em determinar e alocar os recursos necessários a fim de obter, a quantidade necessária de recursos baseados nos requisitos de QoS das aplicações. A determinação e a alocação dos recursos envolve um conjunto de procedimentos compostos tanto pela avaliação da demanda dos serviços de rede como pela analise da tecnologia envolvida. Também neste conjunto de procedimentos, uma tarefa bastante complexa, é a predição de tráfego na rede, na qual consiste em prever o tráfego futuro a partir de amostras obtidas em instantes regularmente espaçados. Esta predição pode ser vista como um grande problema, visto a flexibilidade da Internet de suportar novas aplicações. A avaliação contínua dos requisitos de QoS em intervalos pré-definidos é necessária, como forma de coletar informações para as futuras negociações entre os provedores de serviço e os clientes.

O provisionamento em serviços diferenciados está relacionado com a forma de fornecer os recursos necessários às aplicações, baseados no tipo de tráfego e no mapeamento *codepoint-PHB*, que representa o comportamento agregado de cada serviço no domínio.

Enquanto os provedores de serviços conhecem, a partir de um SLA efetuado com um cliente, uma quantidade determinada de tráfego que entra por um ponto da rede, o volume total para esta classe de serviço não pode ser estimada com exatidão, devido o fato de existir outros SLAs com outros clientes, necessitando de um controle adequado para a melhor aplicação dos

recursos da rede. Baseado nessa necessidade de provisionamento da rede para suportar uma quantidade determinada de serviços, uma estimativa no volume de tráfego no domínio deve ser quantificada. Um ponto que deve ser observado é a necessidade de controle dos recursos da rede a fim de coordenar as políticas de negociação de recursos entre os provedores de serviços e os clientes. Estes controles devem evitar, por exemplo, perdas ou atraso nas transmissões das aplicações avançadas na rede. Outro ponto é que um melhor controle nas admissões de conexões na rede, deve possuir um algoritmo de decisão que determina ou não que um novo fluxo de tráfego atravesse o domínio. Cada fonte de tráfego necessita de uma certa quantidade de recursos de rede para transferir dados de sua origem ao seu destino, esse controle de admissão é usado para controlar a alocação de recursos na rede.

A partir da literatura consultada, e como não foi encontrado o mecanismo ora proposto, propomos o presente trabalho que tem por ponto principal o provisionamento do tráfego em redes *Diffserv*, contendo corretores de recursos para o domínio, baseados no projeto Qbone [18,20], com a finalidade de manter os níveis de QoS fim a fim quando do efetivo estabelecimento dos contratos prestação de serviços (SLAs). A alocação destes recursos pode ser concedida totalmente ou parcialmente e está sujeita ao controle de admissão nos roteadores do domínio.

As solicitações de recursos são regidas pela Equação 01:

$$\mathbf{01.\ C_{alocada} + C_{requerida} \pounds C_{total},\ [66]}$$

Onde C_{total} é a quantidade total de capacidade permitida no domínio num determinado roteador de borda, $C_{alocada}$ é a quantidade de largura de banda já alocada no domínio, e o $C_{requerida}$ é a quantidade requisitada para uma nova conexão. De acordo com a equação acima, as agregações de fluxos são aceitas quando a largura de banda requisitada e a quantidade de largura de banda alocada, não excederem a quantidade total de capacidade permitida em um determinado roteador de borda.

A partir da capacidade de recursos avaliados na rede, os provedores de serviços devem determinar a máxima quantidade de tráfego permitidas,

controlando as admissões de conexões, aceitando ou rejeitando as solicitações. Se a solicitação for aceita, as requisições de recursos devem ser garantidas.

A quantidade de largura de banda avaliada é decrementada da quantidade reservada para uma nova agregação seguindo a Equação 02:

$$\mathbf{02.}\ C_{disponivel} = C_{total} - (C_{alocada} + C_{requerida}),\ \mathbf{[66]}$$

A partir do mecanismo de provisionamento, que utiliza alguns procedimentos de sinalização de mensagens, é possível determinar e efetuar a reserva de recursos na rede. A aplicação que requisitar determinado recurso na rede e este não estiver disponível, será notificada uma indisponibilidade do recurso.

De uma forma geral, um conjunto de procedimentos de medidas de provisionamento em redes *Diffserv* é utilizado para determinar e alocar os recursos necessários, a partir dos tráfegos de chegada até a saída nos nós de borda, conforme o contrato de serviço negociado.

4.2. Monitoramento de Fluxos

Basicamente, o monitoramento de fluxos é um procedimento que observa o tráfego de fluxos que entram no domínio *Diffserv*, procedendo às medições dos parâmetros de QoS em unidades de tempo predeterminadas.

Os fluxos agregados são monitorados e associados aos respectivos serviços. Esse processo analisa as características associadas a cada serviço no domínio, baseado nas informações do mecanismo de provisionamento, disponibilizando uma avaliação de uma amostra do fluxo, a fim de calcular, por exemplo, a vazão, o atraso e a variação do atraso.

Este monitoramento de fluxos representa uma importante ferramenta para o provisionamento na rede uma vez que calcula o volume de serviços disponíveis e alocados no domínio.

O processo de monitoramento é realizado pelo receptor dos dados transmitidos, estes hospedados nos roteadores do domínio. Assim, o receptor

monitora os parâmetros de QoS e envia ao emissor algumas informações como largura de banda oferecida, atrasos de pacotes fim a fim, perdas e outras informações.

Um período inicial de ajuste será considerado para adequar as medições aos seus respectivos serviços com o objetivo de aplicar as políticas determinadas no mecanismo proposto.

4.3. Arquitetura Interna

O mecanismo proposto, baseia-se em três etapas principais:

- A primeira é obtida através do processo de monitoração e tem como objetivo calcular as garantias de transmissão dos fluxos, observando o tráfego dinamicamente, realizando medições em intervalos pré-definidos, calculando, por exemplo, a taxa média de transmissão dos fluxos, caracterização de rajadas, etc.;
- A segunda avalia as características de QoS associados a cada fonte de fluxo identificando cada classe de serviço no domínio; e
- A terceira configura nos roteadores do domínio o controle de admissão dos recursos disponíveis, o gerenciamento dos parâmetros de QoS e o condicionamento de tráfego.

Com a integração dessas etapas, pode-se proceder o provisionamento de recursos no domínio entre quaisquer nós da rede. Deve-se considerar que um período inicial de ajuste será necessário, para adequar estas estimativas de tráfego ao mecanismo de provisionamento.

Na implementação do PHB EF, será necessária a implementação de mecanismos de escalonamento de filas, tais como *Priority Round Robin* (PRR), *Weighted Round Robin* (WRR) ou *Weight Far-Queueing* (WFQ). Em [27] os autores analisam como as implementações do serviço EF comportam-se em termos da variação no retardo (*jitter*). Os resultados demonstram que a implementação por PRR é sempre melhor. Os pacotes do tráfego EF somente são atrasados pela presença de outros pacotes EF em sua fila. No entanto, os roteadores comerciais atuais já implementam o mecanismo de WRR, o que justifica a investigação de alternativas para a melhoria de seu

desempenho [36]. Para o PHB AF, onde a implementação visa minimizar os congestionamentos duradouros, porém os permite em curtos períodos resultantes de rajadas, pode-se obter tal comportamento por um mecanismo de gerenciamento ativo de filas como o *Random Early Detection* (RED) [40]. Este realiza a monitoração do tamanho da fila no roteador e caso haja sinais de um congestionamento iminente, as fontes são notificadas de forma implícita pelo descarte de um de seus pacotes com certa probabilidade.

O gerenciamento de buffers nos roteadores intermediários que implementam o PHB AF é normalmente realizado pelas variações do RED, um para pacotes que estão em conformidade com o perfil de tráfego definido e outro para os pacotes não-conformes. Este mecanismo é conhecido por RIO (RED com IN e OUT). O RED é configurado de forma mais agressiva que o primeiro, visando o descarte prioritário de pacotes OUT [2].

Alguns requisitos são necessários para a configuração do mecanismo de provisionamento, entre estes estão:

- Gerenciamento dos recursos nos roteadores que necessitam de uma pré-configuração para atender as requisições de tráfego, incluem a manutenção dos recursos disponíveis e alocados. Os roteadores de borda suportarão o controle de admissão;
- Gerenciamento de cada interface dos roteadores o qual poderá identificar quais roteadores e interfaces estão sendo utilizados e configurados com os recursos disponíveis;
- Gerenciamento da topologia que manterá os caminhos da rede a serem percorridos pelas requisições de tráfego.

Baseado nestes requisitos tem-se os seguintes componentes, ilustrados na Figura 25:

- BB SLA *Policy Control* (BBSLA), que contém a especificação do tipo de tráfego e seus parâmetros contratados e armazenados na tabela BB SLA *Table*;
- BB *Resources Control* (BBRC), que contém informações dos recursos disponíveis e alocados para diferentes serviços nos roteadores e armazenados na tabela BB *Resources Table*;

- o agente chamado de *Resource Manager Agent* (RMA) que atua diretamente na configuração dos roteadores; e
- BB *Controler Protocol* (BBCP), que efetua as sinalizações de mensagens do mecanismo.

A partir das negociações de QoS efetuados entre os provedores de serviços e os clientes, o mecanismo de provisionamento manterá os endereços da origem e destino da transmissão, bem como os parâmetros contratados fim a fim na tabela denominada, BB SLA *Table* (BBSLA), ilustrada na Figura 26.

O mecanismo de provisionamento proposto incluirá as interfaces dos roteadores de borda (*edge router*), tanto os de entrada (*ingress*) com os de saída do domínio (*egress*), a partir dos endereços de origem e destino. Os parâmetros de QoS atualmente alocados e os valores disponíveis serão verificados, pelo agente RMA na tabela BB *Resources Table* (BBRT). O controle de admissão será executado para gerenciar a chegada do tráfego alocando ou não os recursos na rede.

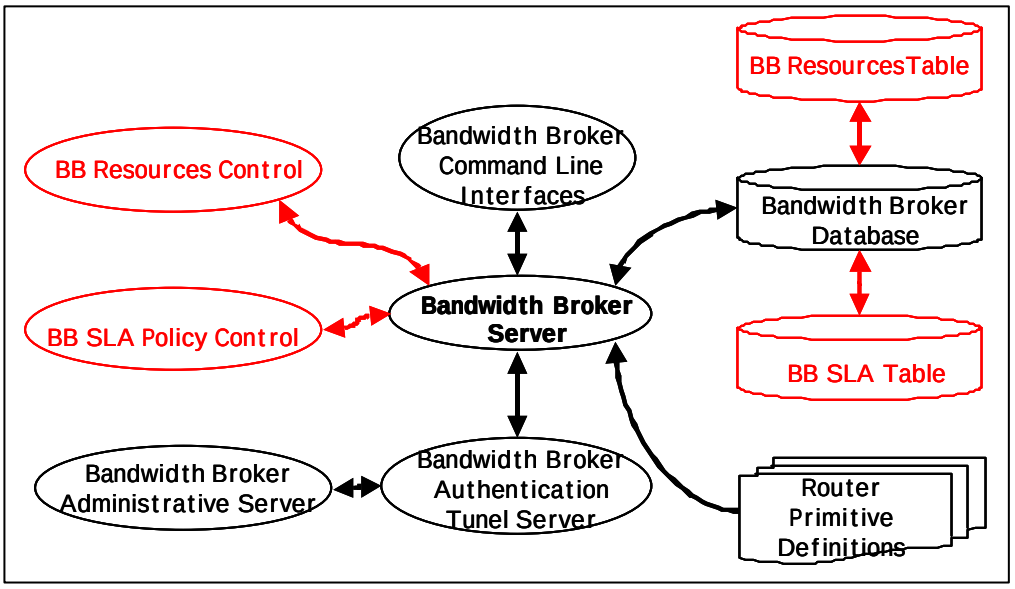

Figura 26 – Arquitetura do Mecanismo de Provisionamento

Uma vez que os parâmetros das tabelas acima serão manipulados como sendo políticas de provisionamento, sua atualização poderá ser feita via um protocolo que trabalhe com políticas como o COPS (*Common Open Policy Service*) [8].

Sendo admitida a conexão, o tráfego será encaminhado para o caminho destino, a partir do descobrimento dos próximos roteadores internos (*core router*) e de suas respectivas interfaces. Os caminhos disponíveis e as interfaces de cada roteador poder ser obtidas utilizando alguma proposta que implemente extensões no algoritmo OSPF, com a adição de métricas baseadas em requisitos de QoS [34] [35].

Depois da descoberta dos próximos roteadores o agente RMA avaliará, a partir dos parâmetros contratados, a disponibilidade de recursos, identificando-o na tabela BBRT. Esta tabela manterá os quantitativos de recursos disponíveis em cada interface dos roteadores internos e encaminhará para o próximo roteador do domínio.

Em intervalos pré-definidos, o mecanismo de provisionamento invocará estimativas de tráfego ao RMA, que devolverá recursos disponíveis fim a fim, que poderão ser negociados pelos provedores de serviço. O mecanismo mantém todo o estado da rede nas tabelas de controle atualizando suas informações periodicamente.

Dessas estimativas de tráfego, o processo de monitoramento poderá encontrar um congestionamento, o que ocasionaria uma violação no serviço contratado, o módulo de provisionamento ajusta novamente os parâmetros de QoS e ocorre então a realimentação das informações nas tabelas de controle.

4.4. Sinalização de Mensagens

O mecanismo de provisionamento proposto mantém em sua base de informação todos os recursos alocados no domínio. Para obter o conhecimento destas informações o mecanismo utiliza um esquema de simples sinalizações fim a fim, do tipo *request-response*, entre os controladores de recursos do mecanismo e os roteadores do domínio. Neste mecanismo este protocolo de sinalização é chamado de BB *Controler Protocol* (BBCP).

Este protocolo de sinalização implementa as seguintes funções:

- Mapeamento das solicitações externas ao domínio para uma classe de serviço *Diffserv*;
- Controle de admissão e/ou reserva de recursos no domínio;
- Notificação de mensagens para os roteadores de ingresso e egresso do domínio;
- Notificação de mensagens de recursos avaliados e alocados para os nós localizados entre os roteadores de ingresso e egresso do domínio; e
- Manutenção das tabelas de controle do mecanismo de provisionamento.

As sinalizações de mensagens do mecanismo podem ser divididas em três tipos genéricos:

- Mensagens *Request* – este tipo de mensagem é usada para solicitar recursos para um determinado tipo de serviço e também coletar informações sobre os recursos disponíveis na rede;
- Mensagens *Response* – neste tipo de mensagem são apresentados a aceitação ou rejeição do acesso ao domínio, bem como a atualização das tabelas de controle do mecanismo;
- Mensagens *Config* – este tipo de mensagem é utilizado para efetuar a configuração nos nós da rede dos recursos alocados no domínio para um determinado serviço.

Uma outra forma de manter as informações do mecanismo seria a utilização do protocolo RSVP [29], como sugerido no projeto Qbone [18,20], para a sinalização das reservas e alocações dos recursos. Por outro lado, um maior quantidade de sinalizações poderia surgir, visto a característica inerente do protocolo RSVP.

4.5. Fluxo de Funcionamento

Suponhamos que um determinado cliente num domínio A quer transmitir dados para serem recebidos num domínio B. Antes dos dados serem transmitidos, entre os domínios, deve ser estabelecido um SLA. O controlador de recursos do domínio A envia uma requisição de serviços para

o domínio B. A solicitação contém informações sobre o tipo de serviços, duração, origem e destino, taxa de transferência e outra informações de controle necessárias. O domínio B examina o SLA proposto e determina qual a quantidade de recursos deve ser reservada, avaliando os serviços já alocados no domínio. Se os recursos são disponíveis, então o controlador de recursos do domínio B envia uma resposta positiva para o domínio A, estabelecendo um SLA entre os dois domínios. Se os recursos não estão disponíveis, então o domínio B tenta negociar os recursos. Se os dois domínios não concordam com um determinado nível de QoS para um SLA, então o acesso para transmissão ao domínio A é negado.

Na Figura 27 apresentamos o fluxo de funcionamento do mecanismo proposto onde:

- (1) – representa a solicitação de uma reserva de recursos ao corretor de serviços do domínio;
- (2) – notificação do acesso negado por indisponibilidade de recursos no domínio;
- (3) – determinação de disponibilidade de recursos e a alocação dos recursos nos nós da rede;
- (4) – atualização das tabelas de controle, informado os recursos alocados;
- (5) – notificação de reserva efetuada no domínio e *start* para inicio de transmissão.

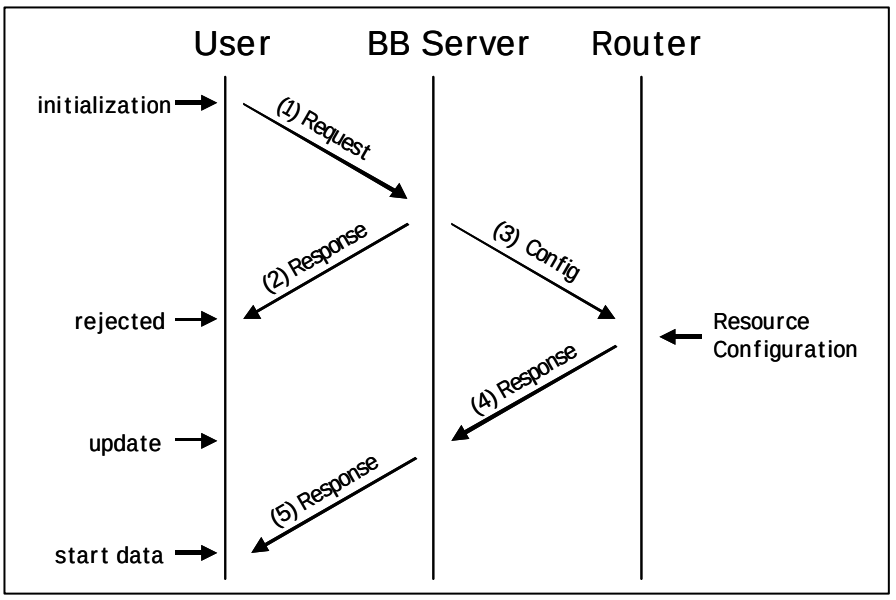

Figura 27 – Fluxo de Funcionamento do Mecanismo

Descrição seqüencial do processo de provisionamento:

- Receber a solicitação de serviços dos clientes; identificar os nós do domínio e os requisitos de QoS em cada nó;
- Processar o pedido de provisionamento no domínio checando a disponibilidade da transmissão de acordo com o contrato de serviço;
- Indicar sucesso ou falha na obtenção dos recursos; provisionar e configurar os caminhos na rede Diffserv;
- Gerar o tráfego entre os nós; medir os parâmetros e o comportamento da rede em intervalos; atualizar nos nós da rede os parâmetros de provisionamento e controle de admissão; e
- Monitorar o tráfego e realimentar as informações.

4.6. Considerações Finais sobre este Capítulo

Neste capítulo foram apresentados os problemas relacionados com a reserva de recursos na rede. Em conseqüência mostrou-se a necessidade de um mecanismo que determinasse e alocasse os recursos na rede, efetuando o controle de admissão, bem como os controles das políticas de provisionamento para as negociações entre os provedores de serviços e os clientes. Foi proposto como solução o mecanismo de provisionamento para serviços diferenciados descrevendo seus componentes e métodos. Este mecanismo de provisionamento de tráfego se propõe a oferecer eficiência e escalabilidade, contribuindo de forma significativa no processo de adoção de garantias de QoS fim a fim na Internet.

No próximo capitulo avaliaremos por meio da simulação a eficiência e desempenho do mecanismo proposto onde serão apresentados os resultados das simulações desenvolvidas neste trabalho.

5. Avaliação de Desempenho

Neste capítulo, será apresentada a avaliação de desempenho do mecanismo proposto utilizando como técnica de avaliação a simulação. Analisaremos os resultados dos experimentos a partir da ferramenta de simulação *Network Simulator* (NS).

5.1. Técnica de Simulação

Dentre as técnicas de avaliação temos medição, simulação e modelagem analítica. Dentre estas técnicas, a mais difundida na área da computação é sem dúvida a simulação devendo-se ao fato de ser mais flexível, permitindo estudar diferentes cenários. Utilizaremos então o *Network Simulator* (NS) [16], ferramenta bastante conhecida no meio acadêmico e científico.

5.2. O simulador de redes *Network Simulator*

O NS foi desenvolvido como parte do projeto *Virtual InterNetwork TestBed* (VINT), surgiu como uma variante do simulador *Realistic and Large* (REAL). A distribuição do NS é gratuita, incluindo o código fonte, e é mantida pela *Information Sciences Institute* (ISI), financiado pela DARPA. Este simulador está baseado em eventos discretos e orientado a objetos. Tem como objetivo principal proporcionar um ambiente de simulação para o desenvolvimento de pesquisas em tornos dos protocolos que constituem a Internet.

O núcleo do simulador foi escrito em C++ (*back-end*) e usa OTcl (*front-end*) como comandos e configurador de interfaces para construção de scripts e modelagem da simulação.

A partir de um *script* pode se definir uma variedade de topologias de rede, utilizando endereços, largura de banda, atraso e módulos de roteamento, bem como, uma vasta biblioteca de protocolos e vários tipos de aplicações. Podem ser simulados, por exemplo, *File Transfer Protocol* (FTP), TELNET e *Hypertext Transfer Protocol* (HTTP) usando TCP como protocolo de transporte e aplicações com tráfego CBR usando UDP, como também, múltiplas filas de policiamento podem ser configuradas. Existe ainda o suporte a algumas tecnologias como GPRS e *Bluetooth*, Wireless, dentre outras.

5.2.1. Suporte *Diffserv* no simulador

O suporte *Diffserv* para o NS foi adicionada pela *Nortel Networks* [16], onde as funcionalidades são capturadas por classes de filas. A classe *dsREDQueue* modela múltiplas filas, cada fila para uma classe de serviço. Cada fila contém mecanismos de congestionamento provido pela classe *redQueue*. As classes *EdgeQueue* e *CoreQueue* são especializações da *dsREDQueue* e modela as interfaces de saída dos roteadores. A classe *PolicyClassifier* é parte integrante da *EdgeQueue* fornecendo as funcionalidades de medição e marcação dos fluxos de tráfego.

O módulo *Diffserv* provê: um classificador *multi-field* (MF) baseado no endereço origem e destino para medir a seleção e o classificador DSCP para selecionar a fila do tráfego de fluxo; medidores tipo TSW2CM, TSW3CM, *Token Bucket*, srTCM e trTCM; filas *Drop Tail*; temporizadores RR, WRR, WIRR e PQ; e algoritmos de descarte utilizando RIO-C, RIO-D, WRED e *Drop-on-Threshold* [16].

5.3. Métricas

Nesta dissertação foram utilizadas três métricas básicas para se avaliar o desempenho do mecanismo proposto:

- Atraso (*delay*) – representado pelo atraso fim a fim entre as fontes e os destinos;
- Variação do atraso (*jitter*) – representado pela diferença entre os tempos de chegada de dois pacotes consecutivos menos os tempos de saída; e
- Vazão (*throughput*) – representado pelo taxa de dados que está sendo efetivamente recebida pelo receptor.

5.4. Topologia e Carga de Trabalho

Visando tornar os resultados do experimento os mais reais possíveis, foram definidos quatro critérios básicos na escolha e configuração da topologia e carga de trabalho:

1. os valores de atraso dos enlaces foram escolhidos de forma que os fluxos consigam atingir uma vazão considerável;
2. o tempo da simulação deve ser razoavelmente escolhido, considerando que o mecanismo necessita de um período inicial de ajuste;
3. a geração de tráfego extra no caminho dos fluxos monitorados, tem a finalidade de causar a inserção de pacotes na rede numa taxa maior do que a rede pode suportar; e
4. além disso, os enlaces de acesso têm uma capacidade maior de largura de banda do que aquela dos enlaces de gargalo, de modo a formar tráfegos em rajadas nestes enlaces internos.

Assim, a topologia utilizada é composta de um domínio *Diffserv* com seis *edge routers* (E1 a E6) e seis *core router* (C1 a C6). Cada fonte de tráfego é modelada com transmissores conforme ilustra a Tabela 06. As fontes N4 a N16 são responsáveis em incluir tráfego na rede com a finalidade de sobrecarregar os fluxos EF, AF e BE das fontes N1-N10, N2-N11 e N3-N12, respectivamente. A Figura 28 reproduz a topologia completa da simulação. O tempo de simulação foi de cem segundos.

Na topologia proposta, foram configuradas fontes que geram diferentes tipos de tráfegos como:

- Tipo CBR (*constant bit rate*) – para as fontes CBR, o tamanho do pacote foi configurado para 1Kb;
- Tipo ON/OFF – as fontes são modelados com processos ON/OFF, distribuídos exponencialmente com durações médias de 1,004s nos períodos “on” e 1,587s nos períodos “off”. Cada fonte gera tráfego CBR de 80Kbps, um codificador PCM de 64Kbps com frames de 20ms, e os cabeçalhos (IP de 20bytes, UDP de 8bytes, etc);

- Fontes Telnet;
- Fontes FTP.

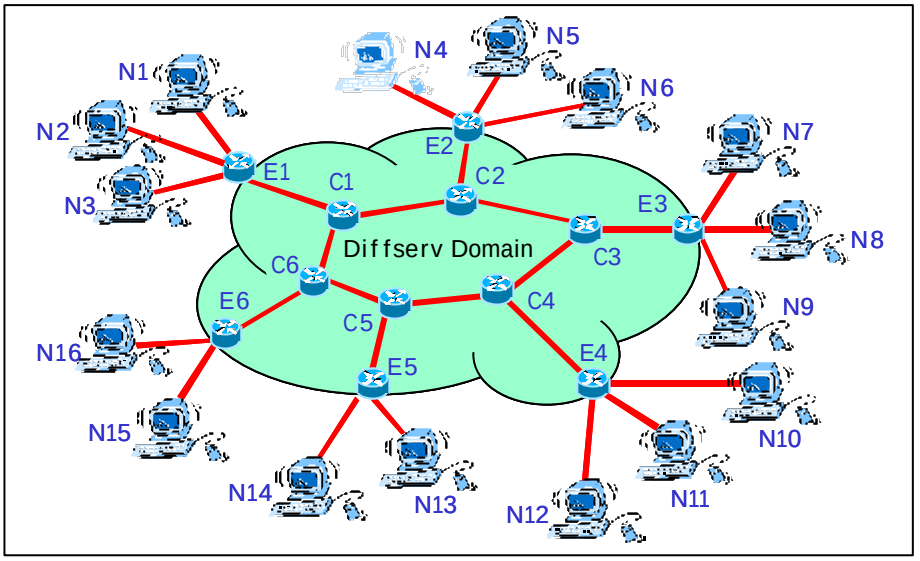

Figura 28 – Topologia da Simulação

As fontes de tráfego utilizadas na simulação estão definidas na Tabela 06.

Tabela 06 – Fontes de Tráfego

No.	Agente	Aplicação	Fonte	Destino	PHB	Taxa	Tempo	
							Início	Fim
01	UDP	CBR	N1	N10	EF	250Kb	0	100
02	UDP	CBR	N4	N7	EF	170Kb	0	100
03	UDP	CBR	N13	N15	EF	120Kb	0	100
04	UDP	CBR	N14	N16	EF	100Kb	0	100
05	UDP	CBR	N14	N16	EF	200Kb	0	100
06	UDP	CBR	N13	N15	EF	150Kb	0	100
07	UDP	CBR	N4	N7	EF	220Kb	0	100
08	UDP	CBR	N4	N7	EF	100Kb	0	100
09	UDP	CBR	N14	N16	EF	110Kb	0	100
10	UDP	CBR	N1	N10	EF	150Kb	0	100
11	UDP	CBR	N1	N10	EF	50Kb	0	100
12	UDP	ON/OFF	N2	N11	AF	6.3Kb	0	100
13	UDP	ON/OFF	N5	N8	AF	6.3Kb	0	100
15	TCP	TELNET	N13	N15	AF	-	0	100
16	TCP	TELNET	N14	N16	AF	-	0	100
17	TCP	FTP	N3	N12	BE	-	0	100
18	UDP	CBR	N3	N12	BE	180Kb	0	100
19	UDP	CBR	N3	N12	BE	320Kb	0	100
20	UDP	CBR	N3	N12	BE	280Kb	0	100
21	UDP	CBR	N3	N12	BE	420Kb	0	100
22	UDP	CBR	N3	N12	BE	480Kb	0	100
23	UDP	CBR	N3	N12	BE	350Kb	0	100
24	UDP	CBR	N3	N12	BE	220Kb	0	100

O mapa da topologia utilizado na simulação está descrito na Tabela 07.

Tabela 07 – Mapa da Topologia

Fonte	Destino	Banda	Atraso	Suporte
N1	E1	100Mb	0.1ms	DropTail
N2	E1	100Mb	0.1ms	DropTail
N3	E1	100Mb	0.1ms	DropTail
N4	E2	100Mb	0.1ms	DropTail
N5	E2	100Mb	0.1ms	DropTail
N6	E2	100Mb	0.1ms	DropTail
N7	E3	100Mb	0.1ms	DropTail
N8	E3	100Mb	0.1ms	DropTail
N9	E3	100Mb	0.1ms	DropTail
N10	E4	100Mb	0.1ms	DropTail
N11	E4	100Mb	0.1ms	DropTail
N12	E4	100Mb	0.1ms	DropTail
N13	E5	100Mb	0.1ms	DropTail
N14	E5	100Mb	0.1ms	DropTail
N15	E6	100Mb	0.1ms	DropTail
N16	E6	100Mb	0.1ms	DropTail
E1	C1	2Mb	5ms	Diffserv
E2	C2	2Mb	5ms	Diffserv
E3	C3	2Mb	5ms	Diffserv
E4	C4	2Mb	5ms	Diffserv
E5	C5	2Mb	5ms	Diffserv
E6	C6	2Mb	5ms	Diffserv
C1	C2	10Mb	20ms	Diffserv
C2	C3	10Mb	20ms	Diffserv
C3	C4	10Mb	20ms	Diffserv
C4	C5	10Mb	20ms	Diffserv
C5	C6	10Mb	20ms	Diffserv
C6	C1	10Mb	20ms	Diffserv

5.5. Experimentos e Resultados

Para avaliar a importância do mecanismo, será realizada uma análise de qualidade do serviço fim a fim em um ambiente de serviços diferenciados. O experimento consistirá em enviar fluxos de fontes de um nó origem para um nó destino passando pelos roteadores internos e de borda no domínio. Assim, iremos examinar o comportamento dos fluxos enviados com e sem intervenção do mecanismo proposto.

Foram simulados alguns cenários, utilizando os mesmos recursos de redes da topologia e carga de trabalho definida para avaliação.

O objetivo desta seção é através dos cenários simulados mostrar a eficiência do mecanismo validando a proposta apresentada.

5.5.1. Primeiro Cenário – Tráfego EF e BE

No primeiro cenário simulado pretende-se analisar o comportamento do mecanismo proposto medindo o desempenho do Serviço Premium e do Serviço de Melhor Esforço. As filas nos roteadores são configuradas para suportar tanto tráfegos EF como BE.

Para o tráfego agregado EF, o fluxo monitorado possui a taxa constante de 250Kbps, onde é gerado por uma fonte CBR. O condicionamento de tráfego é feito pelo medidor/policiador *Token Bucket*.

Para evitar problemas de sincronização, tráfego de *background* é gerado a partir de várias fontes CBR com taxa distribuídas conforme Tabela 06. O tempo de início das fontes de tráfego varia de [0.4s;0.7s]. Os parâmetros utilizados neste cenário, necessários na configuração do domínio *Diffserv*, estão definidos na Tabela 08.

Tabela 08 – Parâmetros da Simulação para o Primeiro Cenário

Classe do Tráfego	Tipo de Tráfego	PHB	DSCP	Bandwidth	Meter	Dropper
Premium	Vídeo	EF	46	15%	Token Bucket CIR 500Kbps CBS 10K bytes	Drop out of profile
Best Effort	Default	BE	0	85%	Token Bucket CIR 700Kbps CBS 10K bytes	Drop out of profile

Os resultados demonstram uma alteração de comportamento em relação à capacidade de vazão dos tráfegos simulados, conforme apresenta os Gráficos 01 e 02. Pode-se perceber um pequeno atraso para o início das transmissões, o que não representa perdas consideráveis, haja vista o tempo configurado no simulador. Alguns pacotes EF foram perdidos pelo fato dos *buffers* nos roteadores não serem suficientes para armazenar todos os

pacotes no domínio. A vazão para os fluxos EF permaneceu praticamente a mesma para os dois gráficos, enquanto que para os fluxos BE, a vazão apresentou uma melhoria no desempenho.

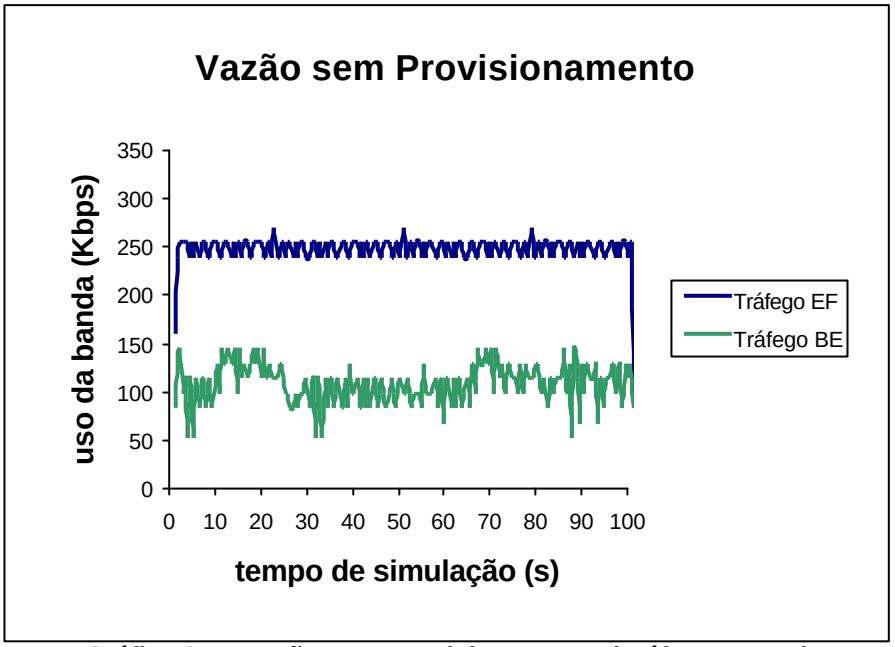

Gráfico 01 – Vazão sem Provisionamento (Tráfego EF-BE)

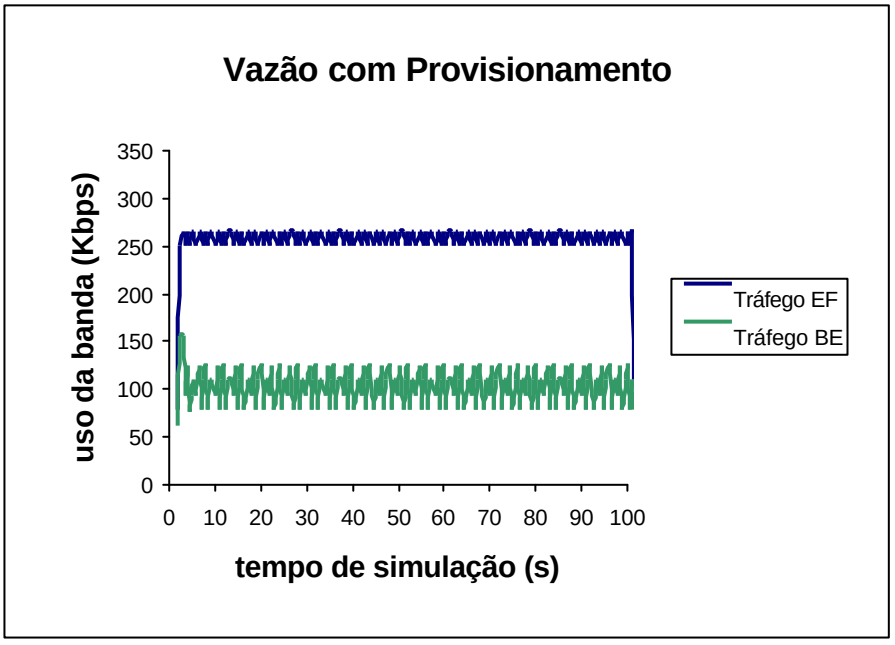

Gráfico 02 – Vazão com Provisionamento (Tráfego EF-BE)

Das 19 requisições de recursos efetuadas pelo mecanismo 32% não foram aceitas e 68% foram aceitas. Isto equivale a 73% das requisições de recurso aceitas para tráfego EF e 63% das requisições de recursos aceitas para tráfego BE.

No Gráfico 03 é apresentado a Vazão Média Fim a Fim dos Tráfegos EF e BE, onde pode-se observar uma leve deformação no gráfico sem provisionamento, no entanto o gráfico com provisionamento manteve-se estável.

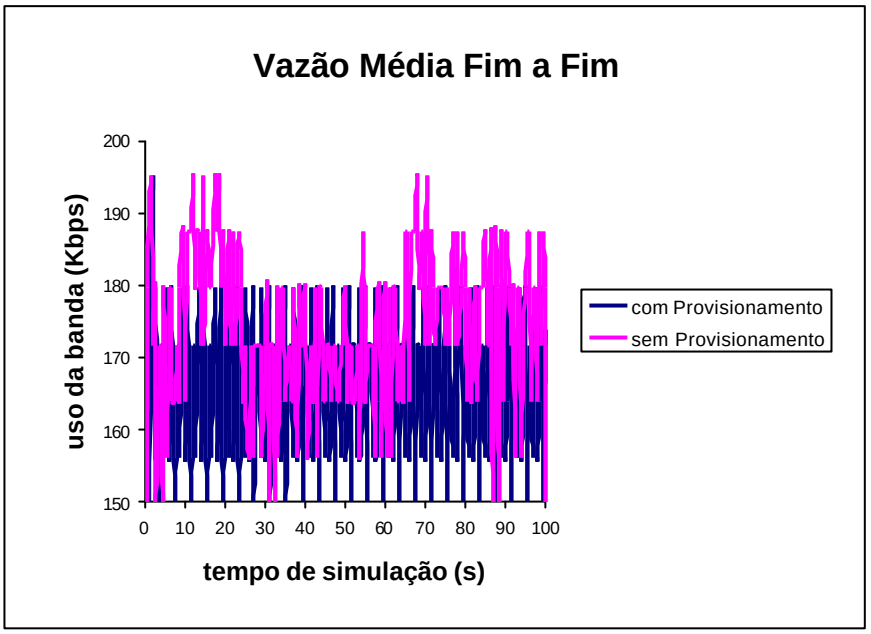

Gráfico 03 – Vazão Média Fim a Fim (Tráfego EF-BE)

O Atraso Médio Fim a Fim com e sem o mecanismo, Gráfico 04, pode ser observado, onde a média dos atrasos em 100% dos pacotes, permaneceu em torno de 320ms, quando o mecanismo não é utilizado e 290ms, quando utilizado.

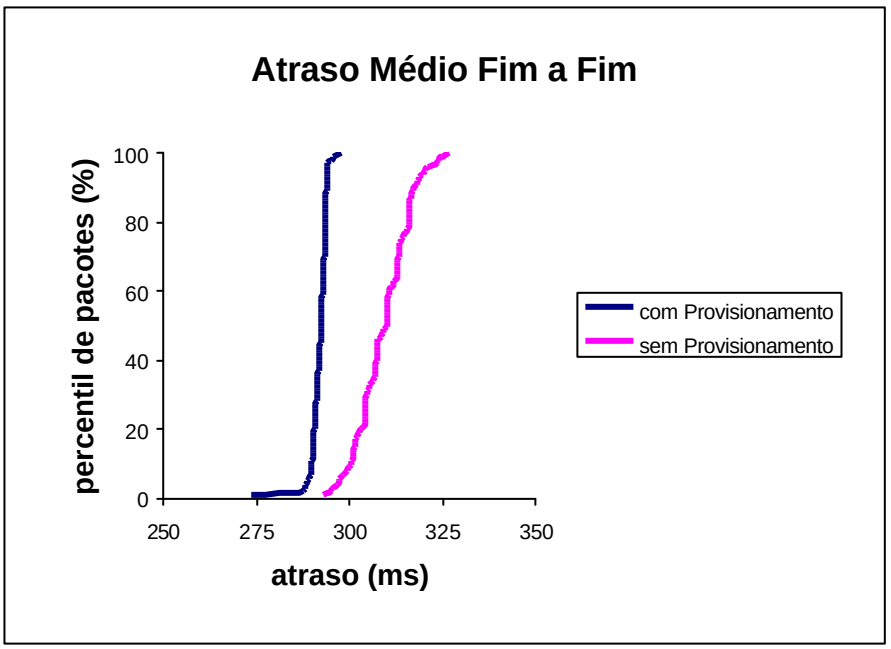

Gráfico 04 – Atraso Médio Fim a Fim (Tráfego EF-BE)

O *Jitter* Médio Fim a Fim, Gráfico 05, com o uso do mecanismo de provisionamento obteve em 100% dos pacotes ganho significativo, não ultrapassou os 12ms, ver Tabela 04, enquanto que sem o uso do mecanismo este valor chegou a aproximadamente 22ms.

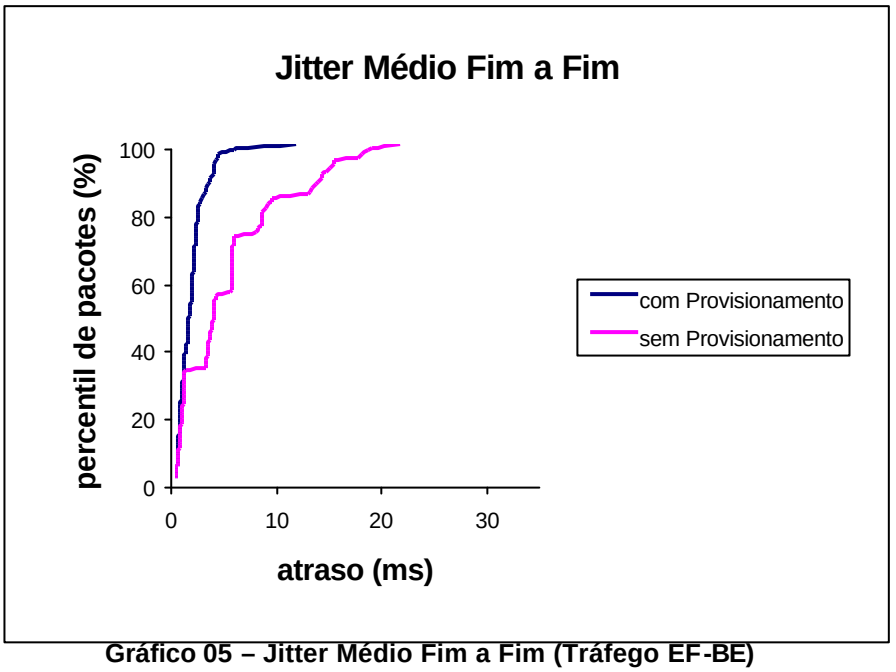

Gráfico 05 – Jitter Médio Fim a Fim (Tráfego EF-BE)

5.5.2. Segundo Cenário – Tráfego AF e BE

No segundo cenário simulado pretende-se analisar o comportamento do mecanismo proposto numa rede congestionada e verificar seu desempenho a partir do descarte de pacotes entre classe de tráfego de mesmo PHB. Neste cenário o Serviço Gold implementa o *Assured Forwarding* (AF PHB) onde suporta diferentes níveis de probabilidade de descarte.

O fluxo monitorado para o tráfego AF é do tipo ON/OFF, distribuídos exponencialmente. Outros fluxos como TELNET também fazem parte deste cenário. O condicionamento de tráfego é feito pelo medidor/policiador *Token Bucket*.

Também neste cenário é gerado tráfego de *background* a partir de várias fontes CBR com taxa distribuídas conforme Tabela 06. O tempo de início das fontes de tráfego varia de [0.4s;0.7s]. Os parâmetros utilizados neste cenário, necessários na configuração do domínio *Diffserv*, estão definidos na Tabela 09.

Tabela 09 – Parâmetros da Simulação para o Segundo Cenário

Classe do Tráfego	Tipo de Tráfego	PHB	DSCP	Bandwidth	Meter	Dropper
Gold	Áudio	AF11 AF12	10 12	85%	Token Bucket CIR 500Kbps CBS 10K bytes	RIO-C Green: 30, 60, 0.06 Yellow: 15, 30, 0.10
Best Effort	default	BE	0	15%	Token Bucket CIR 700Kbps CBS 10K bytes	Drop out of profile

No cenário simulado os períodos de alto congestionamento ocasionam o descarte randômico de pacotes, a partir do método *Random Early Detection* (RED). Os fluxos AF não sofreram grandes conseqüências, este podem ser remarcados como fluxos BE.

Nos Gráficos 06 e 07 pode-se verificar que o tráfego AF permaneceu estável com e sem o uso do mecanismo.

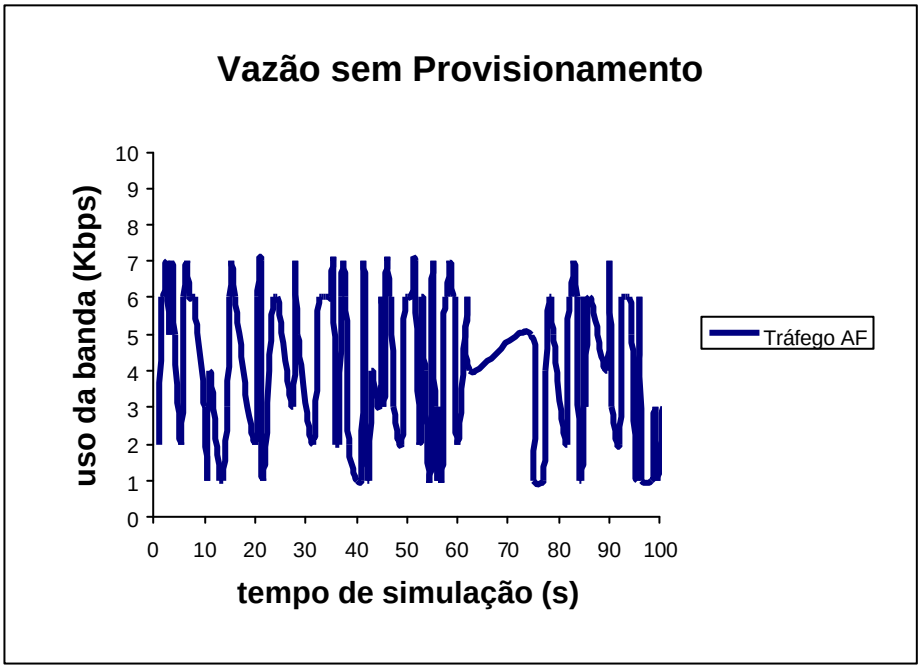

Gráfico 06 – Vazão sem Provisionamento (Tráfego AF)

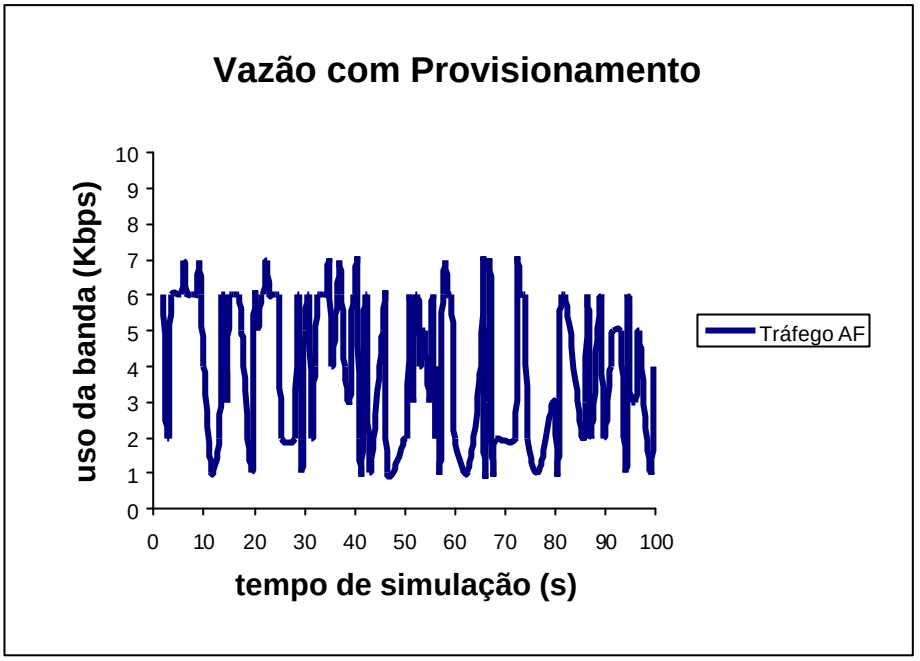

Gráfico 07 – Vazão com Provisionamento (Tráfego AF)

Nos Gráficos 08 e 09 pode-se verificar que o tráfego BE teve um desempenho bem significativo com o uso do mecanismo.

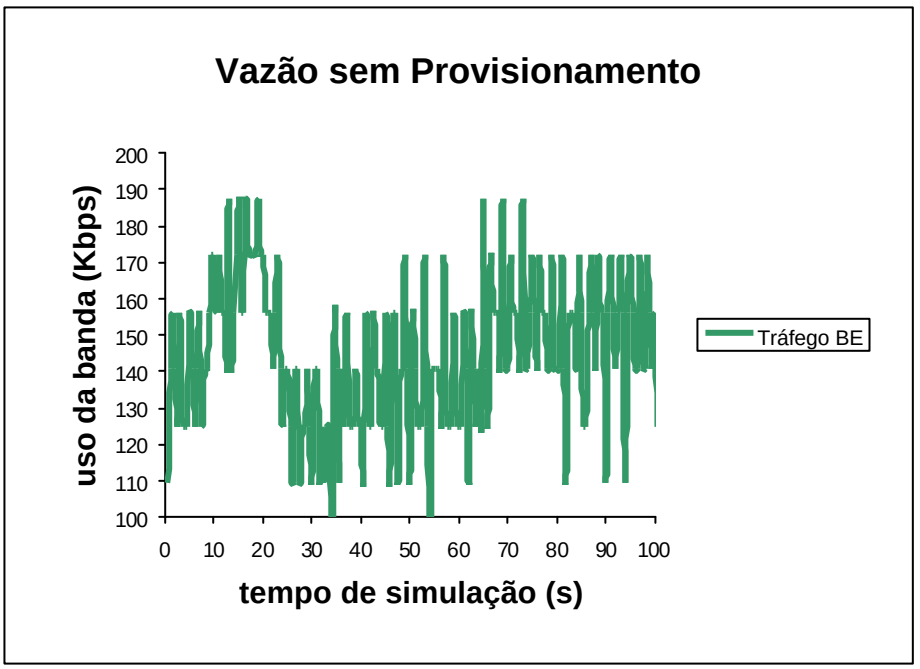

Gráfico 08 – Vazão sem Provisionamento (Tráfego BE)

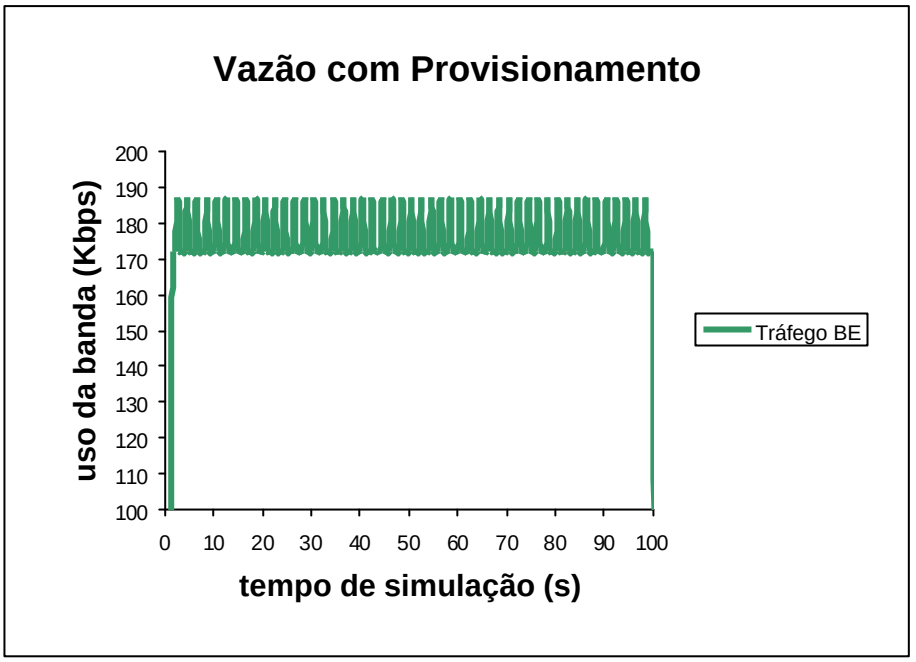

Gráfico 09 – Vazão com Provisionamento (Tráfego BE)

Das 12 requisições de recursos efetuadas pelo mecanismo 58% não foram aceitas e 42% foram aceitas. Isto equivale a 75% das requisições de recursos aceitas para tráfego AF e 25% das requisições de recursos aceitas para tráfego BE.

No Gráfico 08, pode-se observar, a melhoria na eficiência do mecanismo proposto.

A Vazão Média Fim a Fim dos Tráfego AF e BE demonstra que, apesar da remarcação de pacotes AF, o mecanismo atendeu as requisições de recursos tendo um aumento razoável no seu desempenho. Houve uma perda reduzida de pacotes AF motivada pelo congestionamento momentâneo da rede.

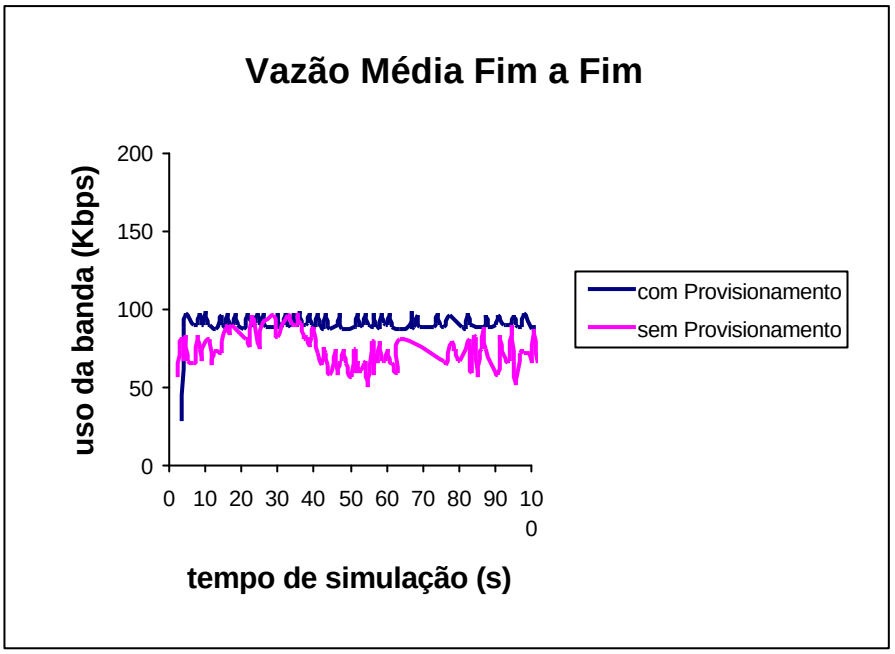

Gráfico 10 – Vazão Média Fim a Fim (Tráfego AF-BE)

A Atraso Médio Fim a Fim dos Tráfego AF e BE foi medido sem o uso do mecanismo, Gráfico 09, alcançando 285ms e 173ms com o uso do mecanismo.

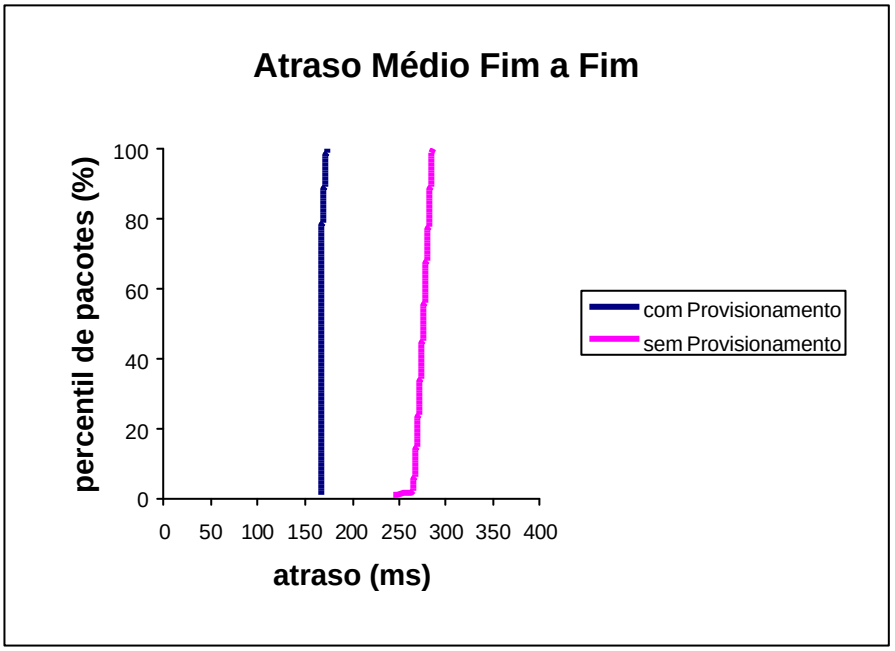

Gráfico 11 – Atraso Médio Fim a Fim (Tráfego AF-BE)

O *Jitter* Médio Fim a Fim dos Tráfego AF e BE foi medido sem o uso do mecanismo, Gráfico 10, alcançando 20ms e 6ms com o uso do mecanismo.

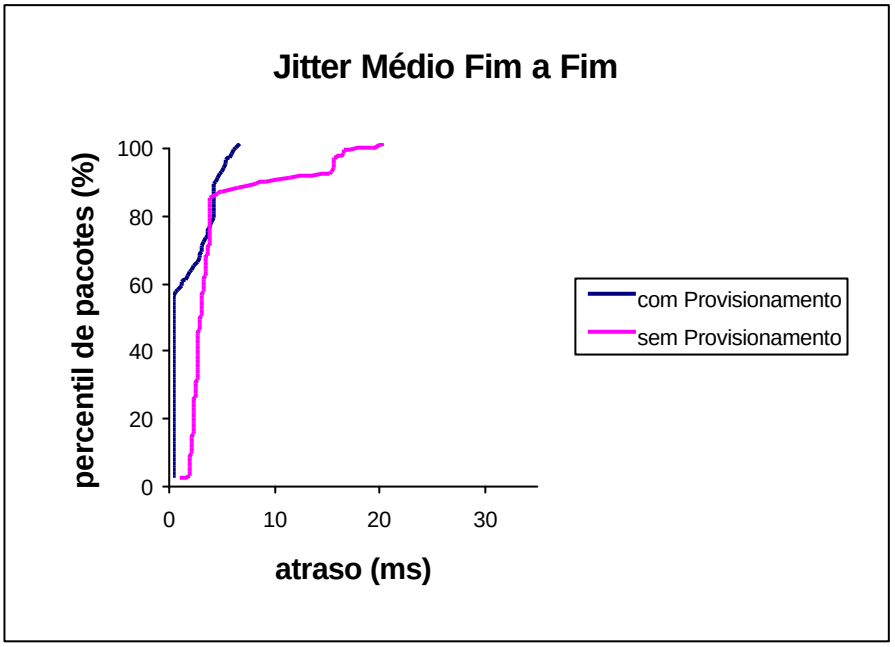

Gráfico 12 – Jitter Médio Fim a Fim (Tráfego AF-BE)

5.5.3. Terceiro Cenário – Tráfego EF, AF e BE

Neste terceiro cenário pretendeu-se explorar o comportamento do mecanismo numa rede com os três tipos de Serviços Premium, Gold e Melhor Esforço, mostrando a versatilidade do mecanismo proposto.

Os parâmetros utilizados neste cenário, necessários na configuração do domínio *Diffserv*, estão definidos na Tabela 10.

Tabela 10 – Parâmetros da Simulação para o Terceiro Cenário

Classe do Tráfego	Tipo de Tráfego	PHB	DSCP	Bandwidth	Meter	Dropper
Premium	Vídeo	EF	46	15%	Token Bucket CIR 500Kbps CBS 10K bytes	Drop out of profile
Gold	Áudio	AF11 AF12	10 12	50%	Token Bucket CIR 500Kbps CBS 10K bytes	RIO-C Green: 30, 60, 0.06 Yellow: 15, 30, 0.10
Best Effort	Default	BE	0	35%	Token Bucket CIR 700Kbps CBS 10K bytes	Drop out of profile

No Gráfico 11, mostramos o comportamento das agregações de fluxos EF, AF e BE, com suas respectivas vazão, sem o mecanismo de provisionamento, e no Gráfico 12 o comportamento dos mesmos fluxos, com o mecanismo de provisionamento.

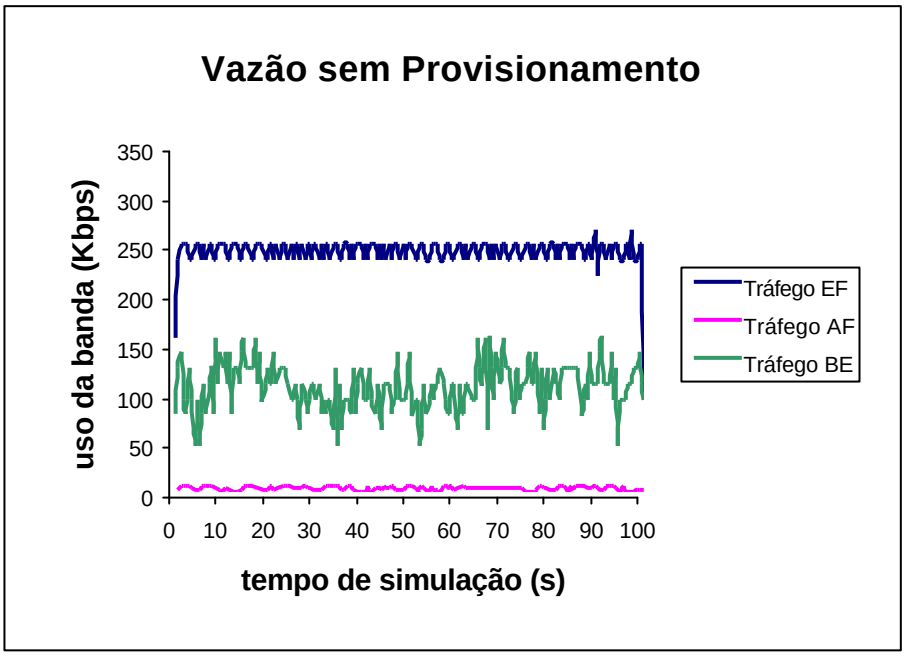

Gráfico 13 – Vazão sem Provisionamento (Tráfego EF-AF-BE)

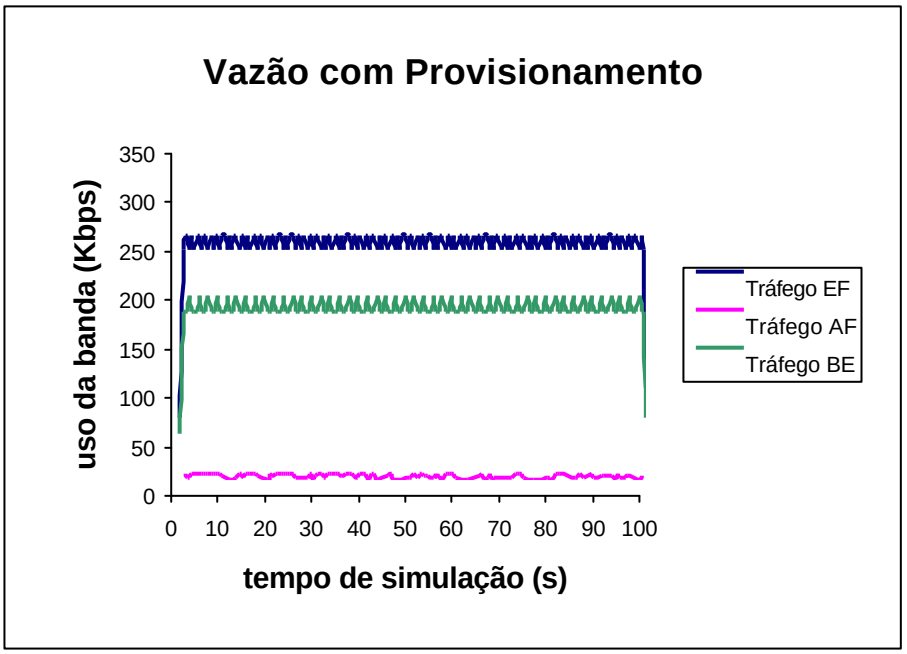

Gráfico 14 – Vazão com Provisionamento (Tráfego EF-AF-BE)

Pelos Gráficos 11 e 12, pode ser observado que o cenário apresenta vazão estável com o uso do mecanismo proposto, mantidas as mesmas

condições e tráfego de fluxos. Vale ressaltar que, numa rede onde encontramos uma sobrecarga generalizada, por falta de recursos disponíveis no domínio, o mecanismo de provisionamento não apresentará melhorias significativas.

Das 23 requisições de recursos efetuadas pelo mecanismo 47% não foram aceitas e 53% foram aceitas. Isto equivale a 82% das requisições de recurso aceitas para tráfego EF, 25% das requisições de recurso aceitas para tráfego AF e 25% das requisições de recursos aceitas para tráfego BE.

O Gráfico 13 apresenta a vazão média fim a fim, mostrado o ganho estável com o uso do mecanismo.

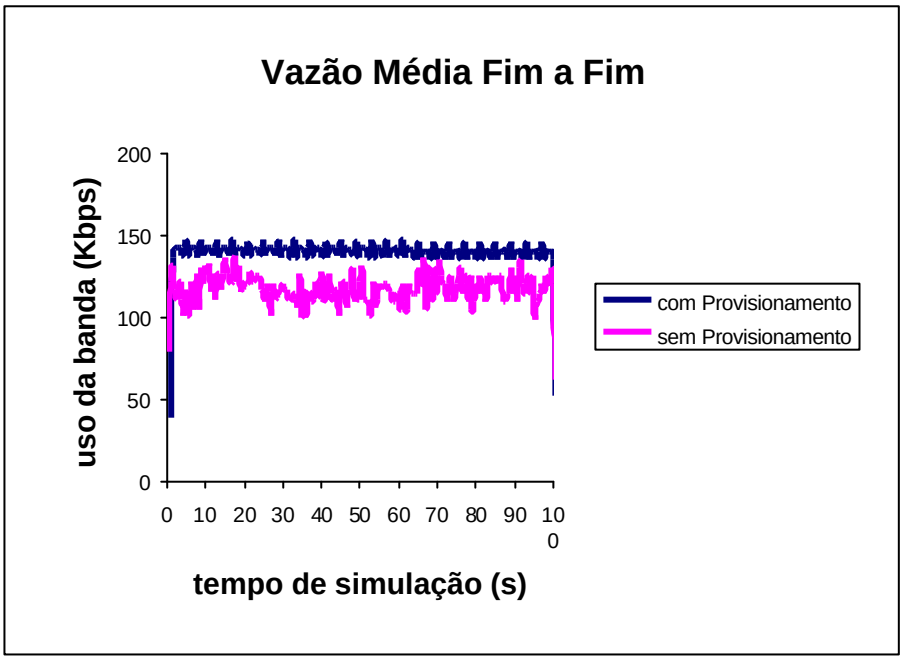

Gráfico 15 – Vazão Média Fim a Fim (Tráfego EF-AF-BE)

Dentro das mesmas condições, avaliamos o atraso médio fim a fim com e sem mapeamento, Gráfico 14. Podemos observar que, a média dos atrasos em 100% dos pacotes, fica em torno de 300ms, quando o mecanismo não é utilizado e 180ms, quando utilizado, valor aceitável para tráfego de aplicações avançadas, ver Tabela 04.

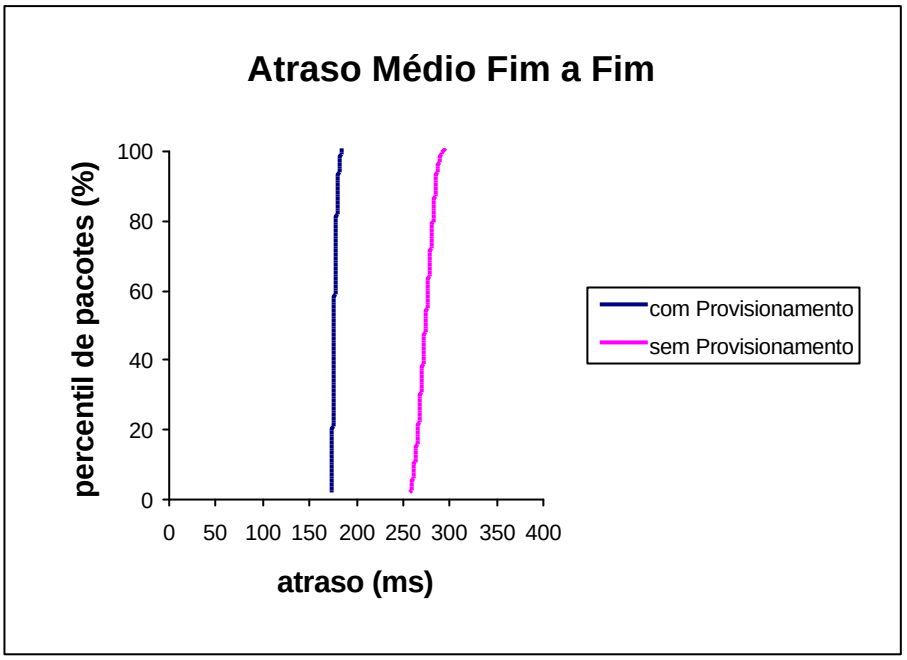

Gráfico 16 – Atraso Médio Fim a Fim (Tráfego EF-AF-BE)

Outra métrica avaliada, o *jitter*, Gráfico 15, que é a variação média dos atrasos fim a fim dos fluxos. Com o uso do mecanismo de provisionamento o *jitter* médio de 100% dos pacotes obteve ganho significativo, não ultrapassou os 11ms, ver Tabela 04, enquanto que sem o uso do mecanismo este valor chegou a aproximadamente 30ms.

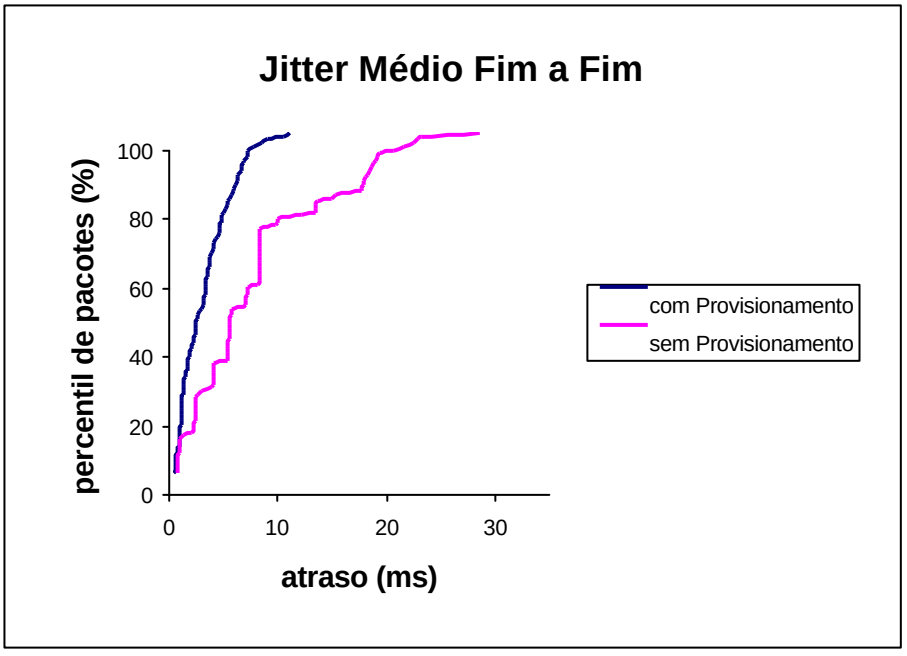

Gráfico 17 – Jitter Médio Fim a Fim (Tráfego EF-AF-BE)

5.6. Validação dos resultados das simulações

Para fins de validação dos resultados das simulações, buscou-se executar várias replicações da proposta. A idéia básica desse processo de validação é a promoção de testes seqüenciais, a fim de encontrar a média e o desvio padrão das variáveis de saída das simulações. Partindo do princípio de que os resultados das simulações são uma distribuição normal, onde a distribuição dos erros é aleatória, o intervalo de confiança pode ser determinado gerando um conjunto de valores dentro dos quais a média se situa.

Para um intervalo de confiança de 95% com um total de 100 observações, para cada cenário simulado, obteve-se os seguintes resultados dispostos nas tabelas abaixo:

Tabela 11 – Intervalo de Confiança – Primeiro Cenário

Intervalo de Confiança					
Largura de Banda		**Atraso**		**Jitter**	
Limite Inferior	Limite Superior	Limite Inferior	Limite Superior	Limite Inferior	Limite Superior
173,68	175,14	307,05	311,33	3,35	7,08
164,27	166,12	291,30	292,35	0,98	2,19

Tabela 12 – Intervalo de Confiança – Segundo Cenário

Intervalo de Confiança					
Largura de Banda		**Atraso**		**Jitter**	
Limite Inferior	Limite Superior	Limite Inferior	Limite Superior	Limite Inferior	Limite Superior
73,34	74,51	274,28	276,96	2,10	6,18
88,31	89,08	167,30	168,67	0,90	2,10

Tabela 13 – Intervalo de Confiança – Terceiro Cenário

Intervalo de Confiança					
Largura de Banda		**Atraso**		**Jitter**	
Limite Inferior	Limite Superior	Limite Inferior	Limite Superior	Limite Inferior	Limite Superior
117,67	118,19	267,33	272,20	5,88	9,99
139,31	140,56	171,32	173,54	2,04	4,01

5.7. Considerações Finais sobre este Capítulo

Neste capítulo foi descrito o ambiente de simulação, as ferramentas e os parâmetros de configuração realizados no *Network Simulator*, a fim de verificar o comportamento da implementação do mecanismo proposto. Através dessas simulações com o uso do mecanismo, obteve-se um melhor controle dos recursos do domínio mantendo os fluxos de tráfego sem grandes alterações. A redefinição dos percentuais alocados a cada classe de tráfego, sem uma mudança dos recursos da rede, pode ocasionar o comprometimento do desempenho alcançado por outros fluxos. A mudança nos percentuais deve ser feita de forma planejada, de forma a evitar interferências nos outros serviços. Como todos os serviços seguem um perfil pré-estabelecido, as mudanças devem ocorrer para melhor utilizar os recursos da rede, não influenciando negativamente os serviços já contratados. Alguns ganhos significativos, como por exemplo na vazão média fim a fim e na medição do atraso e variação do atraso (*jitter*), demonstraram um melhor desempenho com o uso do mecanismo. No próximo capítulo serão apresentadas as conclusões deste trabalho, as suas principais contribuições e algumas propostas para trabalhos futuros.

6. Conclusão e Trabalhos Futuros

Neste capítulo, será apresentada a conclusão baseada nos resultados obtidos da análise de desempenho do mecanismo proposto para serviços diferenciados, como também serão apresentadas as principais contribuições desta dissertação e algumas propostas para trabalhos futuros.

6.1. Considerações Finais

Discutimos neste trabalho, uma variedade de informações que enfoca o problema de alocação e garantias de recursos. Foram apresentados os conceitos básicos, características e requisitos da implementação de qualidade de serviço, necessárias as aplicações avançadas na rede. Alguns modelos utilizados atualmente como Serviços Diferenciados e Engenharia de Tráfego com MPLS foram detalhadamente descritos de forma a propiciar, aos futuros trabalhos, uma rica fonte de informação. Ainda no contexto de qualidade de serviço, caracterizamos os mecanismos presentes nos dispositivos de rede, responsáveis pelo ajuste de parâmetros que disciplinam determinadas funções como controle de fluxo, controle de filas, controle de congestionamentos e controle de policiamento.

Ao propormos o mecanismo de provisionamento de tráfego em redes para serviços diferenciados, que é uma extensão de outras políticas aplicadas dentro do domínio, estamos interessados em exprimir o quão importante é o gerenciamento de QoS fim-a-fim como forma de atender ao emprego das aplicações distribuídas e avançadas na rede

Observamos na simulação que os parâmetros de QoS (como a vazão, atraso e variação do atraso) sofrem a influência significativa de um processo de provisionamento de tráfego eficiente.

Como outra importante vantagem do mecanismo proposto apresentado é que os resultados obtidos, no ambiente *Diffserv*, mostraram melhorias nas métricas de QoS quando utilizados as filas DropTail, escalonador WRR e os condicionadores de balde furado (*Token Bucket*). Se forem utilizados mecanismos mais sofisticados, como por exemplo, fila WRED e outros escalonadores (WFQ, WF2Q), os resultados certamente serão ainda melhores.

6.2. Contribuições

- ✓ Este trabalho apresenta como principal contribuição à proposta de um mecanismo de provisionamento para controle de recursos na rede. Esta proposta inclui a implementação de módulos compatíveis com o projeto *Qbone* [18].
- ✓ A contribuição para o código do *Network Simulator*, por sua vez, é formada pelo modelo do mecanismo proposto nesta dissertação.

6.3. Trabalhos Futuros

Alguns estudos podem ser desenvolvidos a partir deste trabalho.

Como trabalhos futuros sugerimos alguns temas que podem ser desenvolvidos a partir desta dissertação, são eles:

- A adoção de protocolos que trabalhem com políticas, como o COPS, a fim de manter os parâmetros do mecanismo atualizados, uma vez que estes são manipulados como políticas de provisionamento.
- Estudo de técnicas de roteamento, particularmente, o descobrimento de rotas e interfaces dos roteadores do domínio, como por exemplo, a utilização de extensões do algoritmo OSPF, com o objetivo de adicionar métricas baseadas nos parâmetros de QoS adotadas no mecanismo de provisionamento proposto.
- Implementação do nível hierárquico de negociação da Arquitetura *Chameleon*.
- A utilização do mecanismo proposto em ambientes de rede móvel, verificando seus problemas no controle do tráfego, vantagens e desvantagens.
- Implementação que utilize outras variáveis de QoS, com a capacidade de ajustar dinamicamente os parâmetros de acordo com os contratos e políticas especificados.

Referências Bibliográficas

[1] AQUILA Project, http://www-st.inf.tu-dresden.de/aquila, 2000.

[2] D. Clark, W. Fang, Explicit Allocation of Best-Effort Packet Delivery Service, IEEE/ACM Trans. on Networking, 6 (4), pp. 362-373, August 1998.

[3] F. Baumgartner, T. Braun and P. Habergger, Differentiated services: A New Approach for Quality of Service in the Internet, in Proceedings of the 8th IFIP Conference on High Performance Networking – HPN'98, September 1998.

[4] Fankhauser, G., Schweikert, D., and Plattner, B. Service Level Agreement Trading for the Differentiated Services Architecture. Swiss Federal Institute of Technology, Computer Engineering and Networks Lab, Technical Report No. 59. November 1999.

[5] Günter, M. & Braun, T., Evaluation of Bandwidth Broker Signaling, IEEE 7th International Conference on Network Protocols, Outubro 1999.

[6] I. Khalil and T. Braun. Implementation of a Bandwidth Broker for Dynamic End-to-End Resource Reservation in Outsourced Virtual Private Networks. The 25th Annual IEEE Conference on Local Computer Networks (LCN), November 9-10 2000.

[7] I. Stoica and H. Zhang. LIRA: A model for service differentiation in the Internet. In Proceedings of NOSSDAV'98, London, UK, July 1998.

[8] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Raja, A. Sastry, The COPS (Common Open Policy Service) Protocol. RFC 2748, January 2000.

[9] J. Heinanen, F. Baker, W. Weiss and J. Wroclawski, Assured Forwarding PHB Group, Internet RFC 2597, June 1999.

[10] J. Hwang, A Scalable Dynamic Provisioning Mechanism for IntServ-DiffServ Networks for Guaranteed Services. Center for Science and Technology, Syracuse University.

[11] K. Kilkk, Differentiated Services for the Internet, Macmillan Technology Series, 1999.

[12] K. Nichols, S. Blake, F. Baker and D. L. Black, Definition of the Differentiated Services Field (DS field) in the IPv4 IPv6 Headers, Internet RFC 2474, December 1998.

[13] Kamienski, C.A. & Sadok, D., Chameleon: uma Arquitetura para Serviços Avançados Fim a Fim na Internet com QoS, 19º SBRC, Florianópolis/SC, Maio 2001.

[14] Kamienski, C.A. & Sadok, D., Engenharia de Tráfego em uma Rede de Serviços Diferenciados, 18º SBRC, Belo Horizonte/MG, Maio 2000.

[15] N. Semret, R. Liao, A.T. Campbell, and A.A. Lazar. Peering and Provisioning of Differentiated Internet Services. In Proceedings of INFOCOM 2000, Tel-Aviv, Israel, March 2000.

[16] Network Simulator (versão 2.1b8a), http://www.isi.edu/nsnam/ns/, June 2001. Contributed Code: Diffserv Moldel, Nortel Networks, http://www7.nortel.com:8080/ctl/

[17] Nichols, K., Jacobson, V. & Zhang, L., A Two-bit Differentiated Services Architecture for the Internet, RFC 2638, Julho 1999.

[18] QBONE, The Internet2 QBone Bandwidth Broker, 2000. http://www.internet2.edu/qos/qbone/QBBAC.shtml

[19] Rezende, J.F. Avaliação do Serviço Assegurado para a Diferenciação de Serviços na Internet, 17º SBRC, Salvador/BA, Maio 1999.

[20] R. Neilson, J. Wheeler, F. Reichmeyer, S. Hares, A Discussion of Bandwidth Broker Requirements for Internet2 QBone Deployment, version 0.7, http://www.merit.edu/internet/working.groups/i2-qbone-bb/doc/BB_Req7.pdf.

[21] R. R.-F. Liao and A. T. Campbell. Dynamic Core Provisioning for Quantitative Differentiated Service. Technical report, Dept. of Electrical Engineering, Columbia University, April 2001.

[22] R. R.-F. Liao and A. T. Campbell. Dynamic Edge Provisioning for Core Networks. In Proc. IEEE/IFIP Int´l Workshop on Quality of Service, June 2000.

[23] S. Blake, D. L. Black, M. Carlson, E.Davies, Z. Wang and W. Weiss, An Architecture for Differentiated Services, Internet RFC 2475, December 1998.

[24] S. Raghunath, K. Chandrayana, S. Kalyanaraman, Edge-Based Qos Provisioning for Point-to-Set Assured Services. Dept. of ECSE, Rensselaer Polytechnic Institute.

[25] TEQUILA Project, http://www.ist-tequila.org, 2000.

[26] U. Fiedler, P. Huang, B. Plattner, Towards Provisioning Diffserv Intra-Nets. CENL, Swiss Federal Institute of Technology, Zurich. IWQoS 2001. June 2001.

[27] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB, June 1999. RFC 2598.

[28] X. Xiao and L. M. Ni. Internet QoS: A big picture. IEEE Network, 13 (2), March/April 1999.

[29] Braden, R., Clark, D. & Shenker, S., Integrated Services in the Internet Architecture: an Overview, June 1994. RFC 1633.

[30] Rosen, E., Viswanathan, A. & Callon, R., Multiprotocol Label Switching Architecture, January 2001. RFC 3031.

[31] Crawley, E. et al., A Framework for QoS-based Routing in the Internet, August 1998. RFC 2387.

[32] Awduche, D. et al., Requirements for Traffic Engineering over MPLS, September 1999. RFC 2702.

[33] Kamienski, C.A. & Sadok, D., Qualidade de Serviço na Internet, minicurso 18º SBRC, Belo Horizonte/MG, Maio 2000.

[34] R. Guerin, A. Orda, and D. Williams, QoS routing mechanisms and OSPF extensions, in IEEE GLOBECOM'97, Nov. 1997.

[35] G. Apostolopoulos, R. Guerin, and S. Kamat. Implementation and performance measurements of QoS routing extensions to OSPF. In Proc. of INFOCOM'99, March 1999.

[36] Schulzrinne, Henning. Audio and Video Over Packet Networks - Issues, Architecture and Protocols, Interop'94, Paris, France, 1994.

[37] H. Schulzrinne and G. Fokus, RTP Profile for Audio and Video Conferences with Minimal Control, IETF RFC1890, January 1996

[38] Campbell, A., Coulson, G., and D. Hutchison, Flow Management in a Quality of Service Architecture, 5th IFIP Conference on High Performance Networking, Grenoble, France, June 1994.

[39] Ziviani, A., Rezende, J. F., Duarte, Otto, Análise de Desempenho de Mecanismos de Diferenciação de Serviços para Tráfegos de Voz na Internet.

[40] Floyd, S., Jacobson, V., Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking 1, 4. August/1993, 397-413.

[41] Fernandez, M. P., Pedroza, A. C., Rezende, J. F., Qualidade de Serviço em um Domínio Diffserv através de Gerenciamento Baseado em Políticas.

[42] Costa, L. H. M. K. and Duarte, O. C. M. B, Roteamento Baseado em Requisitos de Qualidade de Serviço.

[43] H. G. Perros and K. M. Elsayed, Call admission control schemes: a review. IEEE Commun. Mag, vol. 34, no. 11, pp. 82--91, Nov. 1997.

[44] Sadagic, A., Teitelbaum, B., Leigh, J., Zarki, M. and Liu, H., A Survey on Network Qos Needs of Advanced Internet Applications. Internet2 – QoS Working Group, July 2002.

[45] T-Mobil, TSG-SA Working Group 1, Requirements of IP based Multimedia-Applications over UMTS. Version 7.3.10. May 1999.

[46] Ferguson, P. and Huston, G. ,Quality of Service in the Internet: Fact, Fiction or Compromise?, Appeared in the proceedings of INET'98, July 1998.

[47] Andreozzi, S., Lenzini, L., Rizzo, L., Porras, J., Heikkinen, K., Diffserv Simulations using the network simulator: requirements, issues and solutions, Universitá Degli Stuidi de Pisa, 2000.

[48] Dovrolis, C., Stiliadis, D., and Ramanathan, P. Proportional differentiated services. In Proceedings of SiGCOMM (October 1999), vol. 29.

[49] Ferguson, P. & Huston, G., Quality of Service: Delivering QoS on the Internet and in Corporate Networks. Wiley Computer Publishing, 1999.

[50] Davie, B., Rekhter, Y., MPLS – Technology and Applications. Morgan Kaufmann Publishers. 2000.

[51] Jacobson, V., Nichols, K., Poduri, K., An Expedited Forwarding PHB. June 1999. RFC 2598.

[52] Soares, L.F.G., Lemos, G., Colcher, S., Redes de Computadores – Das Lans, Mans e Wans às Redes ATM. Editora Campus, 1995.

[53] Heinanen, J., Finland, T., Guerin, R., A Single Rate Three Color Marker. Sepetember 1999. RFC 2697.

[54] Heinanen, J., Finland, T., Guerin, R., A Two Rate Three Color Marker. Sepetember 1999. RFC 2698.

[55] Fang, W., Seddigh, N., Nandy, B., A Time Sliding Window Three Color Marker. June 2000. RFC 2859

[56] Chan, K., Seligson, J., Durham, D., Gai, S., McCloghrie, K., Herzog, S., Reichmeyer, F., Yavatkar, R., Smith, A., COPS Usage for Policy Provisioning. March 2001. RFC 3084.

[57] Black, D., Brim, S., Carpenter, B., Le Faucheur, F., Per Hop Behavior Identification Codes. June 2001. RFC 3140.

[58] Mahdavi, J., Paxson, V., IPPM Metrics for Measuring Connectivity. September 1999. RFC 2678.

[59] Le Faucheur, F., Wu, L., Davie, B., Davari, S., Vaananen, P., Krishnan, R., Cheval, P., Heinanen, J., Multi-Protocol Label Switching (MPLS) support of Differentiated Services. May 2002. RFC 3270.

[60] Rosen, E., Viswanathan, A., Callon, R., Multiprotocol Label Switching Architecture. January 2001. RFC 3031

[61] Lopes, E., Westphall, C., Qualidade de Serviço em Redes IP com Diffserv: Avaliação através de Medições. UFSC. Maio 2001.

[62] Fang, W., Differentiated Services: Architecture, Mechanisms and an Evaluation. Princeton University. November 2000.

[63] Goyal, M., Effect of number of drop precedences in Assured Forwarding. Globecom. December 1999. Rio de Janeiro.

[64] Ziviani, A., Duarte, O., Rezende, J., Voz sobre Serviços Diferenciados na Internet. COPPE/UFRJ. Rio de Janeiro. Setembro 1999.

[65] Santos, A., Sadok, D., O Impacto de Controladores de Banda em Domínios Diffserv e Roteamento Dinâmico. Dissertação de Mestrado. UFPE. Recife. Setembro 2002.

[66] Freedman, L., Ni, S., Pinkett, J., Welsh, L., Bandwidth Brokers. Virginia Polytechnic Institute and State University Department of Engineering.

[67] Cisco Quality of Service (QoS) Networking, 2000 http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm

[68] Cisco Systems, Cisco Weighted Random Early Detection, White Paper, 2000.

Apêndice A - Códigos Fonte

Neste apêndice incluímos os códigos fonte dos programas em C++ e Tcl/Otcl desenvolvidos para a implementação do mecanismo de provisionamento no simulador NS. Este código pode ser utilizado de forma livre e gratuita, com a única condição de citar a sua origem e seu autor.

1. Scripts de Simulação: test.tcl e utils.tcl

```
#*****
#***** test.tcl -  Copyright (c) 2002 UFPE jpmn@cin.ufpe.br
#*****
set alg [lindex $argv 0]
set ns [new Simulator]
source utils.tcl
#***** Parametros *****
set testTime 100.0
set qwef 3
set qwaf 10
set qwbe 7
set packetSizeEF 1000
set packetSizeAF 64
set packetSizeBG 1000
set RateEF1 250000
set RateEF2 170000
set RateEF3 120000
set RateEF4 100000
set RateEF5 200000
set RateEF6 150000
set RateEF7 220000
set RateEF8 100000
set RateEF9 110000
set RateEF10 150000
set RateEF11 50000
set RateAF1  64000
set RateBE1 180000
set RateBE2 320000
set RateBE3 280000
set RateBE4 420000
set RateBE5 480000
set RateBE6 350000
set RateBE7 220000
#***** (Kb) 25% ef, 30% af, 45% be *****
set bwtotal_ef 500000
set bwtotal_af 600000
set bwtotal_be 900000
set cir0  500000
set cbs0   10000
set cir1  500000
set cbs1   10000
set cir2  700000
set cbs2   10000
#set rndstartTime [new RNG]
#$rndstartTime seed 0
#set startTime1 [$rndstartTime uniform 0 1]
```

```
#set startTime2 [$rndstartTime uniform 0 1]
#set startTime3 [$rndstartTime uniform 0 1]
#set startTime4 [$rndstartTime uniform 0 1]
#set startTime5 [$rndstartTime uniform 0 1]
#puts "$startTime1"
#puts "$startTime2"
#puts "$startTime3"
#puts "$startTime4"
#puts "$startTime5"
set startTime1 0.4
set startTime2 0.6
set startTime3 0.5
set startTime4 0.4
set startTime5 0.7
#*****************************************************
$ns color 0 Red
$ns color 1 Blue
$ns color 2 Black
$ns color 3 Purple
$ns color 4 Orange
$ns color 5 White
$ns color 6 Coral
$ns color 7 Brown
$ns color 8 Yellow
$ns color 9 Green
$ns color 10 Pink
#***** Arquivos de Saida *****
set f0 [open vazao_EF.tr w]
set f1 [open vazao_AF.tr w]
set f2 [open vazao_BE.tr w]
set utillink [open util_link_c1_e3.tr w]
set pktrec [open pkt_rec_c1_e3.tr w]
set pktperd [open pkt_perd_c1_e3.tr w]
set pktrec0 [open pkt_rec_EF.tr w]
set pktperd0 [open pkt_perd_EF.tr w]
set pktrec1 [open pkt_rec_AF.tr w]
set pktperd1 [open pkt_perd_AF.tr w]
set tp0 0
set tp1 0
set totalpkt0 [open totalpkt_EF.tr w]
set totalpkt1 [open totalpkt_AF.tr w]
#***** Nam *****
#set nf [open out.nam w]
#$ns namtrace-all $nf
#***** Trace all *****
#set trace [open trace.tr w]
#$ns trace-all $trace
#***** Nos e Enlaces da Rede *****
set node_(n1) [$ns node]
set node_(n2) [$ns node]
set node_(n3) [$ns node]
set node_(n4) [$ns node]
set node_(n5) [$ns node]
set node_(n6) [$ns node]
set node_(n7) [$ns node]
set node_(n8) [$ns node]
set node_(n9) [$ns node]
set node_(n10) [$ns node]
set node_(n11) [$ns node]
set node_(n12) [$ns node]
set node_(n13) [$ns node]
```

```
set node_(n14) [$ns node]
set node_(n15) [$ns node]
set node_(n16) [$ns node]
set node_(e1) [$ns node]
set node_(e2) [$ns node]
set node_(e3) [$ns node]
set node_(e4) [$ns node]
set node_(e5) [$ns node]
set node_(e6) [$ns node]
set node_(c1) [$ns node]
set node_(c2) [$ns node]
set node_(c3) [$ns node]
set node_(c4) [$ns node]
set node_(c5) [$ns node]
set node_(c6) [$ns node]
$ns duplex-link $node_(n1) $node_(e1) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n2) $node_(e1) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n3) $node_(e1) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n4) $node_(e2) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n5) $node_(e2) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n6) $node_(e2) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n7) $node_(e3) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n8) $node_(e3) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n9) $node_(e3) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n10) $node_(e4) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n11) $node_(e4) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n12) $node_(e4) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n13) $node_(e5) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n14) $node_(e5) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n15) $node_(e6) 100Mb 0.1ms DropTail
$ns duplex-link $node_(n16) $node_(e6) 100Mb 0.1ms DropTail
$ns simplex-link $node_(e1) $node_(c1) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(c1) $node_(e1) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(e2) $node_(c2) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(c2) $node_(e2) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(e3) $node_(c3) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(c3) $node_(e3) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(e4) $node_(c4) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(c4) $node_(e4) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(e5) $node_(c5) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(c5) $node_(e5) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(e6) $node_(c6) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(c6) $node_(e6) 2Mb 5ms dsRED/edge
$ns simplex-link $node_(c1) $node_(c2) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c2) $node_(c1) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c2) $node_(c3) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c3) $node_(c2) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c3) $node_(c4) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c4) $node_(c3) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c4) $node_(c5) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c5) $node_(c4) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c5) $node_(c6) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c6) $node_(c5) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c6) $node_(c1) 10Mb 20ms dsRED/core
$ns simplex-link $node_(c1) $node_(c6) 10Mb 20ms dsRED/core
#$ns duplex-link-op $node_(n1) $node_(e1) orient down-right
#***** Numeros dos links do dominio diffserv *****
#set link_e1_c1 [[$ns set link_([$node_(e1) id]:[$node_(c1) id])] set
id_]
#set link_e2_c1 [[$ns set link_([$node_(e2) id]:[$node_(c1) id])] set
id_]
```

```
#set link_c1_e3 [[$ns set link_([$node_(c1) id]:[$node_(e3) id])] set
id_]
#***** Parametros diffserv *****
set qe1c1 [[$ns link $node_(e1) $node_(c1)] queue]
set qc1e1 [[$ns link $node_(c1) $node_(e1)] queue]
set qe2c2 [[$ns link $node_(e2) $node_(c2)] queue]
set qc2e2 [[$ns link $node_(c2) $node_(e2)] queue]
set qe3c3 [[$ns link $node_(e3) $node_(c3)] queue]
set qc3e3 [[$ns link $node_(c3) $node_(e3)] queue]
set qe4c4 [[$ns link $node_(e4) $node_(c4)] queue]
set qc4e4 [[$ns link $node_(c4) $node_(e4)] queue]
set qe5c5 [[$ns link $node_(e5) $node_(c5)] queue]
set qc5e5 [[$ns link $node_(c5) $node_(e5)] queue]
set qe6c6 [[$ns link $node_(e6) $node_(c6)] queue]
set qc6e6 [[$ns link $node_(c6) $node_(e6)] queue]
set qc1c2 [[$ns link $node_(c1) $node_(c2)] queue]
set qc2c1 [[$ns link $node_(c2) $node_(c1)] queue]
set qc2c3 [[$ns link $node_(c2) $node_(c3)] queue]
set qc3c2 [[$ns link $node_(c3) $node_(c2)] queue]
set qc3c4 [[$ns link $node_(c3) $node_(c4)] queue]
set qc4c3 [[$ns link $node_(c4) $node_(c3)] queue]
set qc4c5 [[$ns link $node_(c4) $node_(c5)] queue]
set qc5c4 [[$ns link $node_(c5) $node_(c4)] queue]
set qc5c6 [[$ns link $node_(c5) $node_(c6)] queue]
set qc6c5 [[$ns link $node_(c6) $node_(c5)] queue]
set qc6c1 [[$ns link $node_(c6) $node_(c1)] queue]
set qc1c6 [[$ns link $node_(c1) $node_(c6)] queue]
#** de e1 para c1 **
$qe1c1 set numQueues_ 3
$qe1c1 setNumPrec 2
$qe1c1 addPolicyEntry [$node_(n1) id] [$node_(n10) id] TokenBucket 46
$cir0 $cbs0
$qe1c1 addPolicyEntry [$node_(n2) id] [$node_(n11) id] TSW2CM 10
$cir1
$qe1c1 addPolicyEntry [$node_(n3) id] [$node_(n12) id] TokenBucket 0
$cir2 $cbs2
$qe1c1 addPolicyEntry [$node_(e1) id] [$node_(e4) id] TokenBucket 0
$cir2 $cbs2
$qe1c1 addPolicerEntry TokenBucket 46 51
$qe1c1 addPolicerEntry TSW2CM 10 12
$qe1c1 addPolicerEntry TokenBucket 0
$qe1c1 addPHBEntry 46 0 0
$qe1c1 addPHBEntry 51 0 1
$qe1c1 addPHBEntry 10 1 0
$qe1c1 addPHBEntry 12 1 1
$qe1c1 addPHBEntry 0 2 0
$qe1c1 setMREDMode WRED 0
$qe1c1 setMREDMode RIO-C 1
$qe1c1 setMREDMode DROP 2
$qe1c1 setSchedularMode WRR
$qe1c1 addQueueWeights 0 $qwef
$qe1c1 addQueueWeights 1 $qwaf
$qe1c1 addQueueWeights 2 $qwbe
$qe1c1 configQ 0 0 15 35 0.06
$qe1c1 configQ 0 1 5 15 0.30
$qe1c1 configQ 1 0 30 60 0.6
$qe1c1 configQ 1 1 15 30 0.10
$qe1c1 configQ 2 0 60 110 0.02

#** de c1 para e1 **
$qc1e1 set numQueues_ 3
```

```
$qc1e1 setNumPrec 2
$qc1e1 addPolicyEntry [$node_(n10) id] [$node_(n1) id] TokenBucket 46
$cir0 $cbs0
$qc1e1 addPolicyEntry [$node_(n11) id] [$node_(n2) id] TSW2CM 10
$cir1
$qc1e1 addPolicyEntry [$node_(n12) id] [$node_(n3) id] TokenBucket 0
$cir2 $cbs2
$qc1e1 addPolicyEntry [$node_(e4) id] [$node_(e1) id] TokenBucket 0
$cir2 $cbs2
$qc1e1 addPolicerEntry TokenBucket 46 51
$qc1e1 addPolicerEntry TSW2CM 10 12
$qc1e1 addPolicerEntry TokenBucket 0
$qc1e1 addPHBEntry 46 0 0
$qc1e1 addPHBEntry 51 0 1
$qc1e1 addPHBEntry 10 1 0
$qc1e1 addPHBEntry 12 1 1
$qc1e1 addPHBEntry 0 2 0
$qc1e1 setMREDMode WRED 0
$qc1e1 setMREDMode RIO-C 1
$qc1e1 setMREDMode DROP 2
$qc1e1 setSchedularMode WRR
$qc1e1 addQueueWeights 0 $qwef
$qc1e1 addQueueWeights 1 $qwaf
$qc1e1 addQueueWeights 2 $qwbe
$qc1e1 configQ 0 0 15 35 0.06
$qc1e1 configQ 0 1 5 15 0.30
$qc1e1 configQ 1 0 30 60 0.6
$qc1e1 configQ 1 1 15 30 0.10
$qc1e1 configQ 2 0 60 110 0.02
#** de e2 para c2 **
$qe2c2 set numQueues_ 3
$qe2c2 setNumPrec 2
$qe2c2 addPolicyEntry [$node_(n4) id] [$node_(n7) id] TokenBucket 46
$cir0 $cbs0
$qe2c2 addPolicyEntry [$node_(n6) id] [$node_(n14) id] TokenBucket 46
$cir0 $cbs0
$qe2c2 addPolicyEntry [$node_(n5) id] [$node_(n8) id] TSW2CM 10 $cir1
$qe2c2 addPolicyEntry [$node_(e2) id] [$node_(e3) id] TokenBucket 0
$cir2 $cbs2
$qe2c2 addPolicerEntry TokenBucket 46 51
$qe2c2 addPolicerEntry TSW2CM 10 12
$qe2c2 addPolicerEntry TokenBucket 0
$qe2c2 addPHBEntry 46 0 0
$qe2c2 addPHBEntry 51 0 1
$qe2c2 addPHBEntry 10 1 0
$qe2c2 addPHBEntry 12 1 1
$qe2c2 addPHBEntry 0 2 0
$qe2c2 setMREDMode WRED 0
$qe2c2 setMREDMode RIO-C 1
$qe2c2 setMREDMode DROP 2
$qe2c2 setSchedularMode WRR
$qe2c2 addQueueWeights 0 $qwef
$qe2c2 addQueueWeights 1 $qwaf
$qe2c2 addQueueWeights 2 $qwbe
$qe2c2 configQ 0 0 15 35 0.06
$qe2c2 configQ 0 1 5 15 0.30
$qe2c2 configQ 1 0 30 60 0.6
$qe2c2 configQ 1 1 15 30 0.10
$qe2c2 configQ 2 0 60 110 0.02
#** de c2 para e2 **
$qc2e2 set numQueues_ 3
```

```
$qc2e2 setNumPrec 2
$qc2e2 addPolicyEntry [$node_(n7) id] [$node_(n4) id] TokenBucket 46
$cir0 $cbs0
$qc2e2 addPolicyEntry [$node_(n14) id] [$node_(n6) id] TokenBucket 46
$cir0 $cbs0
$qc2e2 addPolicyEntry [$node_(n8) id] [$node_(n5) id] TSW2CM 10 $cir1
$qc2e2 addPolicyEntry [$node_(e3) id] [$node_(e2) id] TokenBucket 0
$cir2 $cbs2
$qc2e2 addPolicerEntry TokenBucket 46 51
$qc2e2 addPolicerEntry TSW2CM 10 12
$qc2e2 addPolicerEntry TokenBucket 0
$qc2e2 addPHBEntry 46 0 0
$qc2e2 addPHBEntry 51 0 1
$qc2e2 addPHBEntry 10 1 0
$qc2e2 addPHBEntry 12 1 1
$qc2e2 addPHBEntry 0 2 0
$qc2e2 setMREDMode WRED 0
$qc2e2 setMREDMode RIO-C 1
$qc2e2 setMREDMode DROP 2
$qc2e2 setSchedularMode WRR
$qc2e2 addQueueWeights 0 $qwef
$qc2e2 addQueueWeights 1 $qwaf
$qc2e2 addQueueWeights 2 $qwbe
$qc2e2 configQ 0 0 15 35 0.06
$qc2e2 configQ 0 1 5 15 0.30
$qc2e2 configQ 1 0 30 60 0.6
$qc2e2 configQ 1 1 15 30 0.10
$qc2e2 configQ 2 0 60 110 0.02
#** de e3 para c3 **
$qe3c3 set numQueues_ 3
$qe3c3 setNumPrec 2
$qe3c3 addPolicyEntry [$node_(n7) id] [$node_(n4) id] TokenBucket 46
$cir0 $cbs0
$qe3c3 addPolicyEntry [$node_(n9) id] [$node_(n13) id] TokenBucket 46
$cir0 $cbs0
$qe3c3 addPolicyEntry [$node_(n8) id] [$node_(n5) id] TSW2CM 10 $cir1
$qe3c3 addPolicyEntry [$node_(n9) id] [$node_(n13) id] TSW2CM 10
$cir1
$qe3c3 addPolicyEntry [$node_(n9) id] [$node_(n13) id] TokenBucket 0
$cir2 $cbs2
$qe3c3 addPolicyEntry [$node_(e3) id] [$node_(e2) id] TokenBucket 0
$cir2 $cbs2
$qe3c3 addPolicerEntry TokenBucket 46 51
$qe3c3 addPolicerEntry TSW2CM 10 12
$qe3c3 addPolicerEntry TokenBucket 0
$qe3c3 addPHBEntry 46 0 0
$qe3c3 addPHBEntry 51 0 1
$qe3c3 addPHBEntry 10 1 0
$qe3c3 addPHBEntry 12 1 1
$qe3c3 addPHBEntry 0 2 0
$qe3c3 setMREDMode WRED 0
$qe3c3 setMREDMode RIO-C 1
$qe3c3 setMREDMode DROP 2
$qe3c3 setSchedularMode WRR
$qe3c3 addQueueWeights 0 $qwef
$qe3c3 addQueueWeights 1 $qwaf
$qe3c3 addQueueWeights 2 $qwbe
$qe3c3 configQ 0 0 15 35 0.06
$qe3c3 configQ 0 1 5 15 0.30
$qe3c3 configQ 1 0 30 60 0.6
$qe3c3 configQ 1 1 15 30 0.10
```

```
$qe3c3 configQ 2 0 60 110 0.02
#** de c3 para e3 **
$qc3e3 set numQueues_ 3
$qc3e3 setNumPrec 2
$qc3e3 addPolicyEntry [$node_(n4) id] [$node_(n7) id] TokenBucket 46
$cir0 $cbs0
$qc3e3 addPolicyEntry [$node_(n13) id] [$node_(n9) id] TokenBucket 46
$cir0 $cbs0
$qc3e3 addPolicyEntry [$node_(n5) id] [$node_(n8) id] TSW2CM 10 $cir1
$qc3e3 addPolicyEntry [$node_(n13) id] [$node_(n9) id] TSW2CM 10
$cir1
$qc3e3 addPolicyEntry [$node_(n13) id] [$node_(n9) id] TokenBucket 0
$cir2 $cbs2
$qc3e3 addPolicyEntry [$node_(e2) id] [$node_(e3) id] TokenBucket 0
$cir2 $cbs2
$qc3e3 addPolicerEntry TokenBucket 46 51
$qc3e3 addPolicerEntry TSW2CM 10 12
$qc3e3 addPolicerEntry TokenBucket 0
$qc3e3 addPHBEntry 46 0 0
$qc3e3 addPHBEntry 51 0 1
$qc3e3 addPHBEntry 10 1 0
$qc3e3 addPHBEntry 12 1 1
$qc3e3 addPHBEntry 0 2 0
$qc3e3 setMREDMode WRED 0
$qc3e3 setMREDMode RIO-C 1
$qc3e3 setMREDMode DROP 2
$qc3e3 setSchedularMode WRR
$qc3e3 addQueueWeights 0 $qwef
$qc3e3 addQueueWeights 1 $qwaf
$qc3e3 addQueueWeights 2 $qwbe
$qc3e3 configQ 0 0 15 35 0.06
$qc3e3 configQ 0 1 5 15 0.30
$qc3e3 configQ 1 0 30 60 0.6
$qc3e3 configQ 1 1 15 30 0.10
$qc3e3 configQ 2 0 60 110 0.02
#** de e4 para c4 **
$qe4c4 set numQueues_ 3
$qe4c4 setNumPrec 2
$qe4c4 addPolicyEntry [$node_(n10) id] [$node_(n1) id] TokenBucket 46
$cir0 $cbs0
$qe4c4 addPolicyEntry [$node_(n11) id] [$node_(n2) id] TSW2CM 10
$cir1
$qe4c4 addPolicyEntry [$node_(n12) id] [$node_(n3) id] TokenBucket 0
$cir2 $cbs2
$qe4c4 addPolicyEntry [$node_(e4) id] [$node_(e1) id] TokenBucket 0
$cir2 $cbs2
$qe4c4 addPolicerEntry TokenBucket 46 51
$qe4c4 addPolicerEntry TSW2CM 10 12
$qe4c4 addPolicerEntry TokenBucket 0
$qe4c4 addPHBEntry 46 0 0
$qe4c4 addPHBEntry 51 0 1
$qe4c4 addPHBEntry 10 1 0
$qe4c4 addPHBEntry 12 1 1
$qe4c4 addPHBEntry 0 2 0
$qe4c4 setMREDMode WRED 0
$qe4c4 setMREDMode RIO-C 1
$qe4c4 setMREDMode DROP 2
$qe4c4 setSchedularMode WRR
$qe4c4 addQueueWeights 0 $qwef
$qe4c4 addQueueWeights 1 $qwaf
$qe4c4 addQueueWeights 2 $qwbe
```

```
$qe4c4 configQ 0 0 15 35 0.06
$qe4c4 configQ 0 1 5 15 0.30
$qe4c4 configQ 1 0 30 60 0.6
$qe4c4 configQ 1 1 15 30 0.10
$qe4c4 configQ 2 0 60 110 0.02
#** de c4 para e4 **
$qc4e4 set numQueues_ 3
$qc4e4 setNumPrec 2
$qc4e4 addPolicyEntry [$node_(n1) id] [$node_(n10) id] TokenBucket 46
$cir0 $cbs0
$qc4e4 addPolicyEntry [$node_(n2) id] [$node_(n11) id] TSW2CM 10
$cir1
$qc4e4 addPolicyEntry [$node_(n3) id] [$node_(n12) id] TokenBucket 0
$cir2 $cbs2
$qc4e4 addPolicyEntry [$node_(e1) id] [$node_(e4) id] TokenBucket 0
$cir2 $cbs2
$qc4e4 addPolicerEntry TokenBucket 46 51
$qc4e4 addPolicerEntry TSW2CM 10 12
$qc4e4 addPolicerEntry TokenBucket 0
$qc4e4 addPHBEntry 46 0 0
$qc4e4 addPHBEntry 51 0 1
$qc4e4 addPHBEntry 10 1 0
$qc4e4 addPHBEntry 12 1 1
$qc4e4 addPHBEntry 0 2 0
$qc4e4 setMREDMode WRED 0
$qc4e4 setMREDMode RIO-C 1
$qc4e4 setMREDMode DROP 2
$qc4e4 setSchedularMode WRR
$qc4e4 addQueueWeights 0 $qwef
$qc4e4 addQueueWeights 1 $qwaf
$qc4e4 addQueueWeights 2 $qwbe
$qc4e4 configQ 0 0 15 35 0.06
$qc4e4 configQ 0 1 5 15 0.30
$qc4e4 configQ 1 0 30 60 0.6
$qc4e4 configQ 1 1 15 30 0.10
$qc4e4 configQ 2 0 60 110 0.02
#** de e5 para c5 **
$qe5c5 set numQueues_ 3
$qe5c5 setNumPrec 2
$qe5c5 addPolicyEntry [$node_(n13) id] [$node_(n15) id] TokenBucket
46 $cir0 $cbs0
$qe5c5 addPolicyEntry [$node_(n14) id] [$node_(n16) id] TokenBucket
46 $cir0 $cbs0
$qe5c5 addPolicyEntry [$node_(n13) id] [$node_(n15) id] TSW2CM 10
$cir1
$qe5c5 addPolicyEntry [$node_(n14) id] [$node_(n16) id] TSW2CM 10
$cir1
$qe5c5 addPolicyEntry [$node_(n13) id] [$node_(n15) id] TokenBucket 0
$cir2 $cbs2
$qe5c5 addPolicyEntry [$node_(n14) id] [$node_(n16) id] TokenBucket 0
$cir2 $cbs2
$qe5c5 addPolicyEntry [$node_(e5) id] [$node_(e6) id] TokenBucket 0
$cir2 $cbs2
$qe5c5 addPolicerEntry TokenBucket 46 51
$qe5c5 addPolicerEntry TSW2CM 10 12
$qe5c5 addPolicerEntry TokenBucket 0
$qe5c5 addPHBEntry 46 0 0
$qe5c5 addPHBEntry 51 0 1
$qe5c5 addPHBEntry 10 1 0
$qe5c5 addPHBEntry 12 1 1
$qe5c5 addPHBEntry 0 2 0
```

```
$qe5c5 setMREDMode WRED 0
$qe5c5 setMREDMode RIO-C 1
$qe5c5 setMREDMode DROP 2
$qe5c5 setSchedularMode WRR
$qe5c5 addQueueWeights 0 $qwef
$qe5c5 addQueueWeights 1 $qwaf
$qe5c5 addQueueWeights 2 $qwbe
$qe5c5 configQ 0 0 15 35 0.06
$qe5c5 configQ 0 1 5 15 0.30
$qe5c5 configQ 1 0 30 60 0.6
$qe5c5 configQ 1 1 15 30 0.10
$qe5c5 configQ 2 0 60 110 0.02
#** de c5 para e5 **
$qc5e5 set numQueues_ 3
$qc5e5 setNumPrec 2
$qc5e5 addPolicyEntry [$node_(n15) id] [$node_(n13) id] TokenBucket
46 $cir0 $cbs0
$qc5e5 addPolicyEntry [$node_(n16) id] [$node_(n14) id] TokenBucket
46 $cir0 $cbs0
$qc5e5 addPolicyEntry [$node_(n15) id] [$node_(n13) id] TSW2CM 10
$cir1
$qc5e5 addPolicyEntry [$node_(n16) id] [$node_(n14) id] TSW2CM 10
$cir1
$qc5e5 addPolicyEntry [$node_(n15) id] [$node_(n13) id] TokenBucket 0
$cir2 $cbs2
$qc5e5 addPolicyEntry [$node_(n16) id] [$node_(n14) id] TokenBucket 0
$cir2 $cbs2
$qc5e5 addPolicyEntry [$node_(e6) id] [$node_(e5) id] TokenBucket 0
$cir2 $cbs2
$qc5e5 addPolicerEntry TokenBucket 46 51
$qc5e5 addPolicerEntry TSW2CM 10 12
$qc5e5 addPolicerEntry TokenBucket 0
$qc5e5 addPHBEntry 46 0 0
$qc5e5 addPHBEntry 51 0 1
$qc5e5 addPHBEntry 10 1 0
$qc5e5 addPHBEntry 12 1 1
$qc5e5 addPHBEntry 0 2 0
$qc5e5 setMREDMode WRED 0
$qc5e5 setMREDMode RIO-C 1
$qc5e5 setMREDMode DROP 2
$qc5e5 setSchedularMode WRR
$qc5e5 addQueueWeights 0 $qwef
$qc5e5 addQueueWeights 1 $qwaf
$qc5e5 addQueueWeights 2 $qwbe
$qc5e5 configQ 0 0 15 35 0.06
$qc5e5 configQ 0 1 5 15 0.30
$qc5e5 configQ 1 0 30 60 0.6
$qc5e5 configQ 1 1 15 30 0.10
$qc5e5 configQ 2 0 60 110 0.02
#** de e6 para c6 **
$qe6c6 set numQueues_ 3
$qe6c6 setNumPrec 2
$qe6c6 addPolicyEntry [$node_(n15) id] [$node_(n13) id] TokenBucket
46 $cir0 $cbs0
$qe6c6 addPolicyEntry [$node_(n16) id] [$node_(n14) id] TokenBucket
46 $cir0 $cbs0
$qe6c6 addPolicyEntry [$node_(n15) id] [$node_(n13) id] TSW2CM 10
$cir1
$qe6c6 addPolicyEntry [$node_(n16) id] [$node_(n14) id] TSW2CM 10
$cir1
```

```
$qe6c6 addPolicyEntry [$node_(n15) id] [$node_(n13) id] TokenBucket 0
$cir2 $cbs2
$qe6c6 addPolicyEntry [$node_(n16) id] [$node_(n14) id] TokenBucket 0
$cir2 $cbs2
$qe6c6 addPolicyEntry [$node_(e6) id] [$node_(e5) id] TokenBucket 0
$cir2 $cbs2
$qe6c6 addPolicerEntry TokenBucket 46 51
$qe6c6 addPolicerEntry TSW2CM 10 12
$qe6c6 addPolicerEntry TokenBucket 0
$qe6c6 addPHBEntry 46 0 0
$qe6c6 addPHBEntry 51 0 1
$qe6c6 addPHBEntry 10 1 0
$qe6c6 addPHBEntry 12 1 1
$qe6c6 addPHBEntry 0 2 0
$qe6c6 setMREDMode WRED 0
$qe6c6 setMREDMode RIO-C 1
$qe6c6 setMREDMode DROP 2
$qe6c6 setSchedularMode WRR
$qe6c6 addQueueWeights 0 $qwef
$qe6c6 addQueueWeights 1 $qwaf
$qe6c6 addQueueWeights 2 $qwbe
$qe6c6 configQ 0 0 15 35 0.06
$qe6c6 configQ 0 1 5 15 0.30
$qe6c6 configQ 1 0 30 60 0.6
$qe6c6 configQ 1 1 15 30 0.10
$qe6c6 configQ 2 0 60 110 0.02
#** de c6 para e6 **
$qc6e6 set numQueues_ 3
$qc6e6 setNumPrec 2
$qc6e6 addPolicyEntry [$node_(n13) id] [$node_(n15) id] TokenBucket
46 $cir0 $cbs0
$qc6e6 addPolicyEntry [$node_(n14) id] [$node_(n16) id] TokenBucket
46 $cir0 $cbs0
$qc6e6 addPolicyEntry [$node_(n13) id] [$node_(n15) id] TSW2CM 10
$cir1
$qc6e6 addPolicyEntry [$node_(n14) id] [$node_(n16) id] TSW2CM 10
$cir1
$qc6e6 addPolicyEntry [$node_(n13) id] [$node_(n15) id] TokenBucket 0
$cir2 $cbs2
$qc6e6 addPolicyEntry [$node_(n14) id] [$node_(n16) id] TokenBucket 0
$cir2 $cbs2
$qc6e6 addPolicyEntry [$node_(e5) id] [$node_(e6) id] TokenBucket 0
$cir2 $cbs2
$qc6e6 addPolicerEntry TokenBucket 46 51
$qc6e6 addPolicerEntry TSW2CM 10 12
$qc6e6 addPolicerEntry TokenBucket 0
$qc6e6 addPHBEntry 46 0 0
$qc6e6 addPHBEntry 51 0 1
$qc6e6 addPHBEntry 10 1 0
$qc6e6 addPHBEntry 12 1 1
$qc6e6 addPHBEntry 0 2 0
$qc6e6 setMREDMode WRED 0
$qc6e6 setMREDMode RIO-C 1
$qc6e6 setMREDMode DROP 2
$qc6e6 setSchedularMode WRR
$qc6e6 addQueueWeights 0 $qwef
$qc6e6 addQueueWeights 1 $qwaf
$qc6e6 addQueueWeights 2 $qwbe
$qc6e6 configQ 0 0 15 35 0.06
$qc6e6 configQ 0 1 5 15 0.30
$qc6e6 configQ 1 0 30 60 0.6
```

```
$qc6e6 configQ 1 1 15 30 0.10
$qc6e6 configQ 2 0 60 110 0.02
#** de c1 para c2 **
$qc1c2 set numQueues_ 3
$qc1c2 setNumPrec 2
$qc1c2 addPHBEntry 46 0 0
$qc1c2 addPHBEntry 51 0 1
$qc1c2 addPHBEntry 10 1 0
$qc1c2 addPHBEntry 12 1 1
$qc1c2 addPHBEntry 0 2 0
$qc1c2 configQ 0 0 15 35 0.06
$qc1c2 configQ 0 1 5 15 0.30
$qc1c2 configQ 1 0 30 60 0.6
$qc1c2 configQ 1 1 15 30 0.10
$qc1c2 configQ 2 0 60 110 0.02
#** de c2 para c1 **
$qc2c1 set numQueues_ 3
$qc2c1 setNumPrec 2
$qc2c1 addPHBEntry 46 0 0
$qc2c1 addPHBEntry 51 0 1
$qc2c1 addPHBEntry 10 1 0
$qc2c1 addPHBEntry 12 1 1
$qc2c1 addPHBEntry 0 2 0
$qc2c1 configQ 0 0 15 35 0.06
$qc2c1 configQ 0 1 5 15 0.30
$qc2c1 configQ 1 0 30 60 0.6
$qc2c1 configQ 1 1 15 30 0.10
$qc2c1 configQ 2 0 60 110 0.02
#** de c2 para c3 **
$qc2c3 set numQueues_ 3
$qc2c3 setNumPrec 2
$qc2c3 addPHBEntry 46 0 0
$qc2c3 addPHBEntry 51 0 1
$qc2c3 addPHBEntry 10 1 0
$qc2c3 addPHBEntry 12 1 1
$qc2c3 addPHBEntry 0 2 0
$qc2c3 configQ 0 0 15 35 0.06
$qc2c3 configQ 0 1 5 15 0.30
$qc2c3 configQ 1 0 30 60 0.6
$qc2c3 configQ 1 1 15 30 0.10
$qc2c3 configQ 2 0 60 110 0.02
#** de c3 para c2 **
$qc3c2 set numQueues_ 3
$qc3c2 setNumPrec 2
$qc3c2 addPHBEntry 46 0 0
$qc3c2 addPHBEntry 51 0 1
$qc3c2 addPHBEntry 10 1 0
$qc3c2 addPHBEntry 12 1 1
$qc3c2 addPHBEntry 0 2 0
$qc3c2 configQ 0 0 15 35 0.06
$qc3c2 configQ 0 1 5 15 0.30
$qc3c2 configQ 1 0 30 60 0.6
$qc3c2 configQ 1 1 15 30 0.10
$qc3c2 configQ 2 0 60 110 0.02
#** de c3 para c4 **
$qc3c4 set numQueues_ 3
$qc3c4 setNumPrec 2
$qc3c4 addPHBEntry 46 0 0
$qc3c4 addPHBEntry 51 0 1
$qc3c4 addPHBEntry 10 1 0
$qc3c4 addPHBEntry 12 1 1
```

```
$qc3c4 addPHBEntry 0 2 0
$qc3c4 configQ 0 0 15 35 0.06
$qc3c4 configQ 0 1 5 15 0.30
$qc3c4 configQ 1 0 30 60 0.6
$qc3c4 configQ 1 1 15 30 0.10
$qc3c4 configQ 2 0 60 110 0.02
#** de c4 para c3 **
$qc4c3 set numQueues_ 3
$qc4c3 setNumPrec 2
$qc4c3 addPHBEntry 46 0 0
$qc4c3 addPHBEntry 51 0 1
$qc4c3 addPHBEntry 10 1 0
$qc4c3 addPHBEntry 12 1 1
$qc4c3 addPHBEntry 0 2 0
$qc4c3 configQ 0 0 15 35 0.06
$qc4c3 configQ 0 1 5 15 0.30
$qc4c3 configQ 1 0 30 60 0.6
$qc4c3 configQ 1 1 15 30 0.10
$qc4c3 configQ 2 0 60 110 0.02
#** de c4 para c5 **
$qc4c5 set numQueues_ 3
$qc4c5 setNumPrec 2
$qc4c5 addPHBEntry 46 0 0
$qc4c5 addPHBEntry 51 0 1
$qc4c5 addPHBEntry 10 1 0
$qc4c5 addPHBEntry 12 1 1
$qc4c5 addPHBEntry 0 2 0
$qc4c5 configQ 0 0 15 35 0.06
$qc4c5 configQ 0 1 5 15 0.30
$qc4c5 configQ 1 0 30 60 0.6
$qc4c5 configQ 1 1 15 30 0.10
$qc4c5 configQ 2 0 60 110 0.02
#** de c5 para c4 **
$qc5c4 set numQueues_ 3
$qc5c4 setNumPrec 2
$qc5c4 addPHBEntry 46 0 0
$qc5c4 addPHBEntry 51 0 1
$qc5c4 addPHBEntry 10 1 0
$qc5c4 addPHBEntry 12 1 1
$qc5c4 addPHBEntry 0 2 0
$qc5c4 configQ 0 0 15 35 0.06
$qc5c4 configQ 0 1 5 15 0.30
$qc5c4 configQ 1 0 30 60 0.6
$qc5c4 configQ 1 1 15 30 0.10
$qc5c4 configQ 2 0 60 110 0.02
#** de c6 para c5 **
$qc6c5 set numQueues_ 3
$qc6c5 setNumPrec 2
$qc6c5 addPHBEntry 46 0 0
$qc6c5 addPHBEntry 51 0 1
$qc6c5 addPHBEntry 10 1 0
$qc6c5 addPHBEntry 12 1 1
$qc6c5 addPHBEntry 0 2 0
$qc6c5 configQ 0 0 15 35 0.06
$qc6c5 configQ 0 1 5 15 0.30
$qc6c5 configQ 1 0 30 60 0.6
$qc6c5 configQ 1 1 15 30 0.10
$qc6c5 configQ 2 0 60 110 0.02
#** de c5 para c6 **
$qc5c6 set numQueues_ 3
$qc5c6 setNumPrec 2
```

```
$qc5c6 addPHBEntry 46 0 0
$qc5c6 addPHBEntry 51 0 1
$qc5c6 addPHBEntry 10 1 0
$qc5c6 addPHBEntry 12 1 1
$qc5c6 addPHBEntry 0 2 0
$qc5c6 configQ 0 0 15 35 0.06
$qc5c6 configQ 0 1 5 15 0.30
$qc5c6 configQ 1 0 30 60 0.6
$qc5c6 configQ 1 1 15 30 0.10
$qc5c6 configQ 2 0 60 110 0.02
#** de c6 para c1 **
$qc6c1 set numQueues_ 3
$qc6c1 setNumPrec 2
$qc6c1 addPHBEntry 46 0 0
$qc6c1 addPHBEntry 51 0 1
$qc6c1 addPHBEntry 10 1 0
$qc6c1 addPHBEntry 12 1 1
$qc6c1 addPHBEntry 0 2 0
$qc6c1 configQ 0 0 15 35 0.06
$qc6c1 configQ 0 1 5 15 0.30
$qc6c1 configQ 1 0 30 60 0.6
$qc6c1 configQ 1 1 15 30 0.10
$qc6c1 configQ 2 0 60 110 0.02
#** de c1 para c6 **
$qc1c6 set numQueues_ 3
$qc1c6 setNumPrec 2
$qc1c6 addPHBEntry 46 0 0
$qc1c6 addPHBEntry 51 0 1
$qc1c6 addPHBEntry 10 1 0
$qc1c6 addPHBEntry 12 1 1
$qc1c6 addPHBEntry 0 2 0
$qc1c6 configQ 0 0 15 35 0.06
$qc1c6 configQ 0 1 5 15 0.30
$qc1c6 configQ 1 0 30 60 0.6
$qc1c6 configQ 1 1 15 30 0.10
$qc1c6 configQ 2 0 60 110 0.02
#************ Fontes de Trafego1 ************
#----------- Trafego EF -----------
set udp0 [new Agent/UDP]
$ns attach-agent $node_(n1) $udp0
$udp0 set packetSize_ $packetSizeEF
$udp0 set class_ 1
set EF0 [new Application/Traffic/CBR]
$EF0 attach-agent $udp0
$EF0 set packet_size_ $packetSizeEF
$EF0 set rate_ $RateEF1
$EF0 set random_ 0
set tanque0 [new Agent/LossMonitor]
$ns attach-agent $node_(n10) $tanque0
$ns connect $udp0 $tanque0

set udp2 [new Agent/UDP]
$ns attach-agent $node_(n4) $udp2
$udp2 set packetSize_ $packetSizeEF
$udp2 set class_ 1
set EF1 [new Application/Traffic/CBR]
$EF1 attach-agent $udp2
$EF1 set packet_size_ $packetSizeEF
$EF1 set rate_ $RateEF2
$EF1 set random_ 0
set null0 [new Agent/LossMonitor]
```

```
$ns attach-agent $node_(n7) $null0
$ns connect $udp2 $null0

set udp3 [new Agent/UDP]
$ns attach-agent $node_(n13) $udp3
$udp3 set packetSize_ $packetSizeEF
$udp3 set class_ 1
set EF2 [new Application/Traffic/CBR]
$EF2 attach-agent $udp3
$EF2 set packet_size_ $packetSizeEF
$EF2 set rate_ $RateEF3
$EF2 set random_ 0
set null1 [new Agent/LossMonitor]
$ns attach-agent $node_(n15) $null1
$ns connect $udp3 $null1

set udp4 [new Agent/UDP]
$ns attach-agent $node_(n14) $udp4
$udp4 set packetSize_ $packetSizeEF
$udp4 set class_ 1
set EF3 [new Application/Traffic/CBR]
$EF3 attach-agent $udp4
$EF3 set packet_size_ $packetSizeEF
$EF3 set rate_ $RateEF4
$EF3 set random_ 0
set null2 [new Agent/LossMonitor]
$ns attach-agent $node_(n16) $null2
$ns connect $udp4 $null2

set udp13 [new Agent/UDP]
$ns attach-agent $node_(n14) $udp13
$udp13 set packetSize_ $packetSizeEF
$udp13 set class_ 1
set EF4 [new Application/Traffic/CBR]
$EF4 attach-agent $udp13
$EF4 set packet_size_ $packetSizeEF
$EF4 set rate_ $RateEF5
$EF4 set random_ 0
set null10 [new Agent/LossMonitor]
$ns attach-agent $node_(n16) $null10
$ns connect $udp13 $null10

set udp14 [new Agent/UDP]
$ns attach-agent $node_(n13) $udp14
$udp14 set packetSize_ $packetSizeEF
$udp14 set class_ 1
set EF5 [new Application/Traffic/CBR]
$EF5 attach-agent $udp14
$EF5 set packet_size_ $packetSizeEF
$EF5 set rate_ $RateEF6
$EF5 set random_ 0
set null11 [new Agent/LossMonitor]
$ns attach-agent $node_(n15) $null11
$ns connect $udp14 $null11

set udp15 [new Agent/UDP]
$ns attach-agent $node_(n4) $udp15
$udp15 set packetSize_ $packetSizeEF
$udp15 set class_ 1
set EF6 [new Application/Traffic/CBR]
$EF6 attach-agent $udp15
```

```
$EF6 set packet_size_ $packetSizeEF
$EF6 set rate_ $RateEF7
$EF6 set random_ 0
set null12 [new Agent/LossMonitor]
$ns attach-agent $node_(n7) $null12
$ns connect $udp15 $null12

set udp16 [new Agent/UDP]
$ns attach-agent $node_(n4) $udp16
$udp16 set packetSize_ $packetSizeEF
$udp16 set class_ 1
set EF7 [new Application/Traffic/CBR]
$EF7 attach-agent $udp16
$EF7 set packet_size_ $packetSizeEF
$EF7 set rate_ $RateEF8
$EF7 set random_ 0
set null13 [new Agent/LossMonitor]
$ns attach-agent $node_(n7) $null13
$ns connect $udp16 $null13

set udp17 [new Agent/UDP]
$ns attach-agent $node_(n14) $udp17
$udp17 set packetSize_ $packetSizeEF
$udp17 set class_ 1
set EF8 [new Application/Traffic/CBR]
$EF8 attach-agent $udp17
$EF8 set packet_size_ $packetSizeEF
$EF8 set rate_ $RateEF9
$EF8 set random_ 0
set null14 [new Agent/LossMonitor]
$ns attach-agent $node_(n16) $null14
$ns connect $udp17 $null14

set udp18 [new Agent/UDP]
$ns attach-agent $node_(n1) $udp18
$udp18 set packetSize_ $packetSizeEF
$udp18 set class_ 1
set EF9 [new Application/Traffic/CBR]
$EF9 attach-agent $udp18
$EF9 set packet_size_ $packetSizeEF
$EF9 set rate_ $RateEF10
$EF9 set random_ 0
set null15 [new Agent/LossMonitor]
$ns attach-agent $node_(n10) $null15
$ns connect $udp18 $null15

set udp19 [new Agent/UDP]
$ns attach-agent $node_(n1) $udp19
$udp19 set packetSize_ $packetSizeEF
$udp19 set class_ 1
set EF10 [new Application/Traffic/CBR]
$EF10 attach-agent $udp19
$EF10 set packet_size_ $packetSizeEF
$EF10 set rate_ $RateEF11
$EF10 set random_ 0
set null16 [new Agent/LossMonitor]
$ns attach-agent $node_(n10) $null16
$ns connect $udp19 $null16

#----------- Trafego AF -----------
set udp1 [new Agent/UDP]
```

```
$ns attach-agent $node_(n2) $udp1
$udp1 set packetSize_ $packetSizeAF
$udp1 set class_ 2
set AF0 [new Application/Traffic/Exponential]
$AF0 attach-agent $udp1
$AF0 set packet_size_ $packetSizeAF
$AF0 set burst_time_ 1004ms
$AF0 set idle_time_ 1587ms
$AF0 set rate_ 6.3k
set tanque1 [new Agent/LossMonitor]
$ns attach-agent $node_(n11) $tanque1
$ns connect $udp1 $tanque1

set udp5 [new Agent/UDP]
$ns attach-agent $node_(n5) $udp5
$udp5 set packetSize_ $packetSizeAF
$udp5 set class_ 2
set AF1 [new Application/Traffic/Exponential]
$AF1 attach-agent $udp5
$AF1 set packet_size_ $packetSizeAF
$AF1 set burst_time_ 1004ms
$AF1 set idle_time_ 1587ms
$AF1 set rate_ 6.3k
set null3 [new Agent/LossMonitor]
$ns attach-agent $node_(n8) $null3
$ns connect $udp5 $null3

set tcp0 [new Agent/TCP]
$tcp0 set fid_ 3
$ns attach-agent $node_(n13) $tcp0
set telnet0 [new Application/Telnet]
$telnet0 attach-agent $tcp0
$telnet0 set interval 0
set sink0 [new Agent/TCPSink]
$ns attach-agent $node_(n15) $sink0
$ns connect $tcp0 $sink0

set tcp1 [new Agent/TCP]
$tcp1 set fid_ 3
$ns attach-agent $node_(n14) $tcp1
set telnet1 [new Application/Telnet]
$telnet1 attach-agent $tcp1
$telnet1 set interval 0
set sink1 [new Agent/TCPSink]
$ns attach-agent $node_(n16) $sink1
$ns connect $tcp1 $sink1

#----------- Trafego BE -----------

set tcp2 [new Agent/TCP]
$tcp2 set fid_ 4
$ns attach-agent $node_(n3) $tcp2
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $node_(n12) $sink2
$ns connect $tcp2 $sink2

set udp6 [new Agent/UDP]
$ns attach-agent $node_(n3) $udp6
$udp6 set packetSize_ $packetSizeBG
```

```
$udp6 set class_ 4
set BE0 [new Application/Traffic/CBR]
$BE0 attach-agent $udp6
$BE0 set packet_size_ $packetSizeBG
$BE0 set rate_ $RateBE1
$BE0 set random_ 0
set tanque2 [new Agent/LossMonitor]
$ns attach-agent $node_(n12) $tanque2
$ns connect $udp6 $tanque2

set udp7 [new Agent/UDP]
$ns attach-agent $node_(n3) $udp7
$udp7 set packetSize_ $packetSizeBG
$udp7 set class_ 4
set BE1 [new Application/Traffic/CBR]
$BE1 attach-agent $udp7
$BE1 set packet_size_ $packetSizeBG
$BE1 set rate_ $RateBE2
$BE1 set random_ 0
set null4 [new Agent/LossMonitor]
$ns attach-agent $node_(n12) $null4
$ns connect $udp7 $null4

set udp8 [new Agent/UDP]
$ns attach-agent $node_(n3) $udp8
$udp8 set packetSize_ $packetSizeBG
$udp8 set class_ 4
set BE2 [new Application/Traffic/CBR]
$BE2 attach-agent $udp8
$BE2 set packet_size_ $packetSizeBG
$BE2 set rate_ $RateBE3
$BE2 set random_ 0
set null5 [new Agent/LossMonitor]
$ns attach-agent $node_(n12) $null5
$ns connect $udp8 $null5

set udp9 [new Agent/UDP]
$ns attach-agent $node_(n3) $udp9
$udp9 set packetSize_ $packetSizeBG
$udp9 set class_ 4
set BE3 [new Application/Traffic/CBR]
$BE3 attach-agent $udp9
$BE3 set packet_size_ $packetSizeBG
$BE3 set rate_ $RateBE4
$BE3 set random_ 0
set null6 [new Agent/LossMonitor]
$ns attach-agent $node_(n12) $null6
$ns connect $udp9 $null6

set udp10 [new Agent/UDP]
$ns attach-agent $node_(n3) $udp10
$udp10 set packetSize_ $packetSizeBG
$udp10 set class_ 4
set BE4 [new Application/Traffic/CBR]
$BE4 attach-agent $udp10
$BE4 set packet_size_ $packetSizeBG
$BE4 set rate_ $RateBE5
$BE4 set random_ 0
set null7 [new Agent/LossMonitor]
$ns attach-agent $node_(n12) $null7
$ns connect $udp10 $null7
```

```
set udp11 [new Agent/UDP]
$ns attach-agent $node_(n3) $udp11
$udp11 set packetSize_ $packetSizeBG
$udp11 set class_ 4
set BE5 [new Application/Traffic/CBR]
$BE5 attach-agent $udp11
$BE5 set packet_size_ $packetSizeBG
$BE5 set rate_ $RateBE6
$BE5 set random_ 0
set null8 [new Agent/LossMonitor]
$ns attach-agent $node_(n12) $null8
$ns connect $udp11 $null8

set udp12 [new Agent/UDP]
$ns attach-agent $node_(n3) $udp12
$udp12 set packetSize_ $packetSizeBG
$udp12 set class_ 4
set BE6 [new Application/Traffic/CBR]
$BE6 attach-agent $udp12
$BE6 set packet_size_ $packetSizeBG
$BE6 set rate_ $RateBE7
$BE6 set random_ 0
set null9 [new Agent/LossMonitor]
$ns attach-agent $node_(n12) $null9
$ns connect $udp12 $null9
#***** Mecanismo de provisionamento *****
if {$alg == "NO"} {
    set now [$ns now]
    $ns at $now "$EF0 start"
    $ns at $testTime "$EF0 stop"
    $ns at $now "$EF1 start"
    $ns at $testTime "$EF1 stop"
    $ns at $now "$EF2 start"
    $ns at $testTime "$EF2 stop"
    $ns at $now "$EF3 start"
    $ns at $testTime "$EF3 stop"
    $ns at $now "$EF4 start"
    $ns at $testTime "$EF4 stop"
    $ns at $now "$EF5 start"
    $ns at $testTime "$EF5 stop"
    $ns at $now "$EF6 start"
    $ns at $testTime "$EF6 stop"
    $ns at $now "$EF7 start"
    $ns at $testTime "$EF7 stop"
    $ns at $now "$EF8 start"
    $ns at $testTime "$EF8 stop"
    $ns at $now "$EF9 start"
    $ns at $testTime "$EF9 stop"
    $ns at $now "$EF10 start"
    $ns at $testTime "$EF10 stop"
    $ns at $now "$AF0 start"
    $ns at $testTime "$AF0 stop"
    $ns at $now "$AF1 start"
    $ns at $testTime "$AF1 stop"
    $ns at $now "$telnet0 start"
    $ns at $testTime "$telnet0 stop"
    $ns at $now "$telnet1 start"
    $ns at $testTime "$telnet1 stop"
    $ns at $now "$ftp0 start"
    $ns at $testTime "$ftp0 stop"
```

```
    $ns at $now "$BE0 start"
    $ns at $testTime "$BE0 stop"
    $ns at $now "$BE1 start"
    $ns at $testTime "$BE1 stop"
    $ns at $now "$BE2 start"
    $ns at $testTime "$BE2 stop"
    $ns at $now "$BE3 start"
    $ns at $testTime "$BE3 stop"
    $ns at $now "$BE4 start"
    $ns at $testTime "$BE4 stop"
    $ns at $now "$BE5 start"
    $ns at $testTime "$BE5 stop"
    $ns at $now "$BE6 start"
    $ns at $testTime "$BE6 stop"
} else {
Agent/Corretor instproc recv {from raa id} {
      global ns testTime EF0 EF1 EF2 EF3 EF4 EF5 EF6 EF7 EF8 EF9 EF10
AF0 AF1 telnet0 telnet1 ftp0 ftp1 ftp2 ftp3 BE0 BE1 BE2 BE3 BE4 BE5
BE6
      set now [$ns now]
      $self instvar node_
      if {$raa == 1} {
         puts "node $from aceitou o RAR de [$node_ id] com RAA $raa."
         #*********** Fontes de Trafego ************
         #----------- Trafego EF -----------
         if {$id == 1} {
            $ns at $now "$EF0 start"
            $ns at $testTime "$EF0 stop"
         }
         if {$id == 2} {
            $ns at $now "$EF1 start"
            $ns at $testTime "$EF1 stop"
         }
         if {$id == 3} {
            $ns at $now "$EF2 start"
            $ns at $testTime "$EF2 stop"
         }
         if {$id == 4} {
            $ns at $now "$EF3 start"
            $ns at $testTime "$EF3 stop"
         }
         if {$id == 5} {
            $ns at $now "$EF4 start"
            $ns at $testTime "$EF4 stop"
         }
         if {$id == 6} {
            $ns at $now "$EF5 start"
            $ns at $testTime "$EF5 stop"
         }
         if {$id == 7} {
            $ns at $now "$EF6 start"
            $ns at $testTime "$EF6 stop"
         }
         if {$id == 8} {
            $ns at $now "$EF7 start"
            $ns at $testTime "$EF7 stop"
         }
         if {$id == 9} {
            $ns at $now "$EF8 start"
            $ns at $testTime "$EF8 stop"
         }
```

```
         if {$id == 10} {
            $ns at $now "$EF9 start"
            $ns at $testTime "$EF9 stop"
         }
         if {$id == 11} {
            $ns at $now "$EF10 start"
            $ns at $testTime "$EF10 stop"
         }
         #----------- Trafego AF -----------
         if {$id == 12} {
            $ns at $now "$AF0 start"
            $ns at $testTime "$AF0 stop"
         }
         if {$id == 13} {
            $ns at $now "$AF1 start"
            $ns at $testTime "$AF1 stop"
         }
         if {$id == 14} {
            $ns at $now "$telnet0 start"
            $ns at $testTime "$telnet0 stop"
         }
         if {$id == 15} {
            $ns at $now "$telnet1 start"
            $ns at $testTime "$telnet1 stop"
         }
         #----------- Trafego BE -----------
         if {$id == 16} {
            $ns at $now "$ftp0 start"
            $ns at $testTime "$ftp0 stop"
         }
         if {$id == 17} {
            $ns at $now "$BE0 start"
            $ns at $testTime "$BE0 stop"
         }
         if {$id == 18} {
            $ns at $now "$BE1 start"
            $ns at $testTime "$BE1 stop"
         }
         if {$id == 19} {
            $ns at $now "$BE2 start"
            $ns at $testTime "$BE2 stop"
         }
         if {$id == 20} {
            $ns at $now "$BE3 start"
            $ns at $testTime "$BE3 stop"
         }
         if {$id == 21} {
            $ns at $now "$BE4 start"
            $ns at $testTime "$BE4 stop"
         }
         if {$id == 22} {
            $ns at $now "$BE5 start"
            $ns at $testTime "$BE5 stop"
         }
         if {$id == 23} {
            $ns at $now "$BE6 start"
            $ns at $testTime "$BE6 stop"
         }
      } else {
            puts "node $from nao aceitou o RAR de [$node_ id] com RAA
$raa."
```

```
      }
}
set arm1 [new Agent/Corretor]
$ns attach-agent $node_(e1) $arm1
$arm1 addcorretor 2 $bwtotal_ef
$arm1 addcorretor 1 $bwtotal_af
$arm1 addcorretor 0 $bwtotal_be

set arm2 [new Agent/Corretor]
$ns attach-agent $node_(e2) $arm2
$arm2 addcorretor 2 $bwtotal_ef
$arm2 addcorretor 1 $bwtotal_af
$arm2 addcorretor 0 $bwtotal_be

set arm3 [new Agent/Corretor]
$ns attach-agent $node_(e3) $arm3
$arm3 addcorretor 2 $bwtotal_ef
$arm3 addcorretor 1 $bwtotal_af
$arm3 addcorretor 0 $bwtotal_be

set arm4 [new Agent/Corretor]
$ns attach-agent $node_(e4) $arm4
$arm4 addcorretor 2 $bwtotal_ef
$arm4 addcorretor 1 $bwtotal_af
$arm4 addcorretor 0 $bwtotal_be

set arm5 [new Agent/Corretor]
$ns attach-agent $node_(e5) $arm5
$arm5 addcorretor 2 $bwtotal_ef
$arm5 addcorretor 1 $bwtotal_af
$arm5 addcorretor 0 $bwtotal_be

set arm6 [new Agent/Corretor]
$ns attach-agent $node_(e6) $arm6
$arm6 addcorretor 2 $bwtotal_ef
$arm6 addcorretor 1 $bwtotal_af
$arm6 addcorretor 0 $bwtotal_be

$ns connect $arm1 $arm4
$ns connect $arm2 $arm3
$ns connect $arm5 $arm6

$ns at $startTime1 "$arm1 send 1 [$node_(n1) id] [$node_(n10) id] 0.0
$testTime $RateEF1 2"
$ns at $startTime5 "$arm2 send 2 [$node_(n4) id] [$node_(n7) id] 0.0
$testTime $RateEF2 2"
$ns at $startTime2 "$arm5 send 3 [$node_(n13) id] [$node_(n15) id]
0.0 $testTime $RateEF3 2"
$ns at $startTime3 "$arm5 send 4 [$node_(n14) id] [$node_(n16) id]
0.0 $testTime $RateEF4 2"
$ns at $startTime1 "$arm1 send 5 [$node_(n14) id] [$node_(n16) id]
0.0 $testTime $RateEF5 2"
$ns at $startTime5 "$arm2 send 6 [$node_(n13) id] [$node_(n15) id]
0.0 $testTime $RateEF6 2"
$ns at $startTime2 "$arm5 send 7 [$node_(n4) id] [$node_(n7) id] 0.0
$testTime $RateEF7 2"
$ns at $startTime3 "$arm5 send 8 [$node_(n4) id] [$node_(n7) id] 0.0
$testTime $RateEF8 2"
$ns at $startTime1 "$arm1 send 9 [$node_(n14) id] [$node_(n16) id]
0.0 $testTime $RateEF9 2"
```

```
$ns at $startTime5 "$arm2 send 10 [$node_(n1) id] [$node_(n10) id]
0.0 $testTime $RateEF10 2"
$ns at $startTime5 "$arm2 send 11 [$node_(n1) id] [$node_(n10) id]
0.0 $testTime $RateEF11 2"

$ns at $startTime5 "$arm1 send 12 [$node_(n2) id] [$node_(n11) id]
0.0 $testTime $RateAF1 1"
$ns at $startTime1 "$arm2 send 13 [$node_(n5) id] [$node_(n8) id] 0.0
$testTime $RateAF1 1"
$ns at $startTime3 "$arm5 send 14 [$node_(n13) id] [$node_(n14) id]
0.0 $testTime $RateAF1 1"
$ns at $startTime4 "$arm5 send 15 [$node_(n14) id] [$node_(n16) id]
0.0 $testTime $RateAF1 1"

$ns at $startTime2 "$arm1 send 16 [$node_(n3) id] [$node_(n12) id]
0.0 $testTime $RateBE1 0"
$ns at $startTime4 "$arm1 send 17 [$node_(n3) id] [$node_(n12) id]
0.0 $testTime $RateBE1 0"
$ns at $startTime2 "$arm1 send 18 [$node_(n3) id] [$node_(n12) id]
0.0 $testTime $RateBE2 0"
$ns at $startTime3 "$arm1 send 19 [$node_(n3) id] [$node_(n12) id]
0.0 $testTime $RateBE3 0"
$ns at $startTime5 "$arm1 send 20 [$node_(n3) id] [$node_(n12) id]
0.0 $testTime $RateBE4 0"
$ns at $startTime2 "$arm1 send 21 [$node_(n3) id] [$node_(n12) id]
0.0 $testTime $RateBE5 0"
$ns at $startTime1 "$arm1 send 22 [$node_(n3) id] [$node_(n12) id]
0.0 $testTime $RateBE6 0"
$ns at $startTime1 "$arm1 send 23 [$node_(n3) id] [$node_(n12) id]
0.0 $testTime $RateBE7 0"
}
#***** Contabilizacao e Monitores *****
PktStats set src_ 0
PktStats set dst_ 0
newPktStats $node_(e4) $node_(n10) $udp0 $tanque0 EF.log
newPktStats $node_(e4) $node_(n11) $udp1 $tanque1 AF.log
newPktStats $node_(e4) $node_(n12) $udp6 $tanque2 BE.log
#***** Sequencia da Simulacao  *****
$ns at 0.0 "record"
set interval 0.5
set qmon_ [$ns monitor-queue $node_(c4) $node_(e4) [$ns get-ns-
traceall] $interval]
$ns at 0.0 "util_link $qmon_ $utillink"
$ns at 0.0 "qtd_pkt_rec $qmon_ $pktrec"
$ns at 0.0 "qtd_pkt_perd $qmon_ $pktperd"
$ns at 0.0 "calc_pkt_rec"
$ns at 0.0 "calc_pkt_perd"
#$ns at $testTime "$qe5c5 printPHBTable"
#$ns at $testTime "$qe5c5 printPolicyTable"
#$ns at $testTime "$qe5c5 printPolicerTable"
$ns at [expr $testTime + 1.0] "finish"
puts "Executando script ..."
$ns run

#
#***** utils.tcl -  Copyright (c) 2002 UFPE jpmn@cin.ufpe.br
#
proc cbr_traffic {srcid dstid src dst class null PacketSize Rate
start stop} {
        global ns Sink_
        set udp_($srcid) [new Agent/UDP]
```

```
        $udp_($srcid)    set class_ $class
        $ns attach-agent $src $udp_($srcid)
        set cbr_($srcid) [new Application/Traffic/CBR]
        $cbr_($srcid)    attach-agent $udp_($srcid)
        $cbr_($srcid)    set packet_size_ $PacketSize
        $udp_($srcid)    set packetSize_  $PacketSize
        $cbr_($srcid)    set rate_ $Rate
        if {($null == 0)} {
                set Sink_($dstid) [new Agent/LossMonitor]
                $Sink_($dstid) set flowid_ $class
        } else {
                set Sink_($dstid) [new Agent/Null]
        }
        $ns attach-agent $dst  $Sink_($dstid)
        $ns connect $udp_($srcid) $Sink_($dstid)
        $ns at $start "$cbr_($srcid) start"
        $ns at $stop  "$cbr_($srcid) stop"
}
proc telnet_traffic {id src dst fid inter start stop} {
    global ns
    set tcp_($id) [new Agent/TCP]
#    $tcp_($id) set packetSize_ $packetsize
    $tcp_($id) set fid_ $fid
    set sink_($id) [new Agent/TCPSink]
    $ns attach-agent $src $tcp_($id)
    $ns attach-agent $dst $sink_($id)
    set telnet_($id) [new Application/Telnet]
    $telnet_($id) attach-agent $tcp_($id)
    $telnet_($id) set interval $inter
    $ns connect $tcp_($id) $sink_($id)
    $ns at $start "$telnet_($id) start"
    $ns at $stop "$telnet_($id) stop"
}
proc ftp_traffic {id src dst fid start stop} {
      global ns
      set tcp_($id) [new Agent/TCP]
#     $tcp_($id) set packetSize_ $packetsize
      $tcp_($id) set fid_ $fid
      $ns attach-agent $src $tcp_($id)
      set ftp_($id) [new Application/FTP]
      $ftp_($id) attach-agent $tcp_($id)
      set sink_($id) [new Agent/TCPSink]
      $ns attach-agent $dst $sink_($id)
      $ns connect $tcp_($id) $sink_($id)
      $ns at $start "$ftp_($id) start"
      $ns at $stop "$ftp_($id) stop"
}
SimpleLink instproc set-target {tg} {
    $self instvar link_
    $link_ target $tg
}
SimpleLink instproc get-target {} {
    $self instvar link_
    set tg [$link_ target]
    return $tg
}
Simulator instproc insert-pktstats {pktstats n1 n2} {
    $self instvar link_
    set sid [$n1 id]
    set did [$n2 id]
    set templink $link_($sid:$did)
```

```
    set linktarget [$templink get-target]
    $templink set-target $pktstats
    $pktstats target $linktarget
}
proc newPktStats {n1 n2 srcAgent sinkAgent filename} {
    global ns end $srcAgent $sinkAgent
    set pktstats [new PktStats]
    set fich [open $filename w]
    $pktstats set-channel $fich
    $pktstats set src_ [$srcAgent set agent_addr_]
    $pktstats set dst_ [$sinkAgent set agent_addr_]
    $ns insert-pktstats $pktstats $n1 $n2
    $ns at 0.1 "$pktstats record-on"
    $ns at 500.0 "$pktstats record-off"
    $ns at 500.0 "close $fich"
    return $pktstats
}
proc record {} {
      global tanque0 tanque1 tanque2 ns f0 f1 f2
      set time 0.5
      set bw0 [$tanque0 set bytes_]
      set bw1 [$tanque1 set bytes_]
      set bw2 [$tanque2 set bytes_]
      set now [$ns now]
        if { $now != 0 && [expr $bw0/$time*8] != 0.0} {
      puts $f0 "$now [expr $bw0/$time*8]"}
        if { $now != 0 && [expr $bw1/$time*8] != 0.0} {
      puts $f1 "$now [expr $bw1/$time*8]"}
        if { $now != 0 && [expr $bw2/$time*8] != 0.0} {
      puts $f2 "$now [expr $bw2/$time*8]"}
      $tanque0 set bytes_ 0
      $tanque1 set bytes_ 0
      $tanque2 set bytes_ 0
      $ns at [expr $now+$time] "record"
}
proc util_link { queueMon_ file } {
        $queueMon_ instvar bdepartures_
      set ns [Simulator instance]
        set time 0.5
        set utlzn_bdepartures [expr $bdepartures_ / $time * 8.0 ]
        set now [$ns now]
        puts $file "$now\t$utlzn_bdepartures"
        $queueMon_ set bdepartures_ 0
        $ns at [expr $now+$time] "util_link $queueMon_ $file"
}
proc qtd_pkt_rec { queueMon_ file } {
        $queueMon_ instvar  pdepartures_
        set ns [Simulator instance]
      set time 0.5
        set utlzn_pdepartures [expr $pdepartures_ ]
        set now [$ns now]
        puts $file "$now\t$utlzn_pdepartures"
        $queueMon_ set pdepartures_ 0
        $ns at [expr $now+$time] "qtd_pkt_rec $queueMon_ $file"
}
proc qtd_pkt_perd { queueMon_ file } {
        $queueMon_ instvar  pdrops_
        set ns [Simulator instance]
      set time 0.5
        set utlzn_pdrops [expr $pdrops_ ]
        set now [$ns now]
```

```
        puts $file "$now\t$utlzn_pdrops"
        $queueMon_ set pdrops_ 0
        $ns at [expr $now+$time] "qtd_pkt_perd $queueMon_ $file"
}
proc calc_pkt_rec { } {
      global ns pktrec0 pktrec1 totalpkt0 totalpkt1 tp0 tp1 tanque0
tanque1
      set time 0.5
        set now [$ns now]
      set partial0 [$tanque0 set npkts_]
        puts $pktrec0 "$now\t$partial0"
      set partial1 [$tanque1 set npkts_]
        puts $pktrec1 "$now\t$partial1"
      set auxtotal [expr $partial0 + $tp0]
      set tp0 $auxtotal
      puts $totalpkt0 "$now\t$tp0"
      set auxtotal [expr $partial1 + $tp1]
      set tp1 $auxtotal
      puts $totalpkt1 "$now\t$tp1"
      $ns at [expr $now+$time] "calc_pkt_rec"
      $tanque0 set npkts_ 0
      $tanque1 set npkts_ 0
}
proc calc_pkt_perd { } {
      global ns pktperd0 pktperd1 tanque0 tanque1
      set time 0.5
        set now [$ns now]
      set partial0 [$tanque0 set nlost_]
      puts $pktperd0 "$now\t$partial0"
      set partial1 [$tanque1 set nlost_]
        puts $pktperd1 "$now\t$partial1"
      $ns at [expr $now+$time] "calc_pkt_perd"
      $tanque0 set nlost_ 0
      $tanque1 set nlost_ 0
}
proc finish {} {
#      global nf trace
       global ns f0 f1 f2
#      $ns flush-trace
       close $f0
       close $f1
       close $f2
#      close $trace
#      close $nf
#      exec xgraph video.tr voz.tr -geometry 800x400 &
#      exec nam out.nam &
       exit 0
       puts "Done."
}
```

2. hdr_rar.h e hdr_rar.cc

```
// hdr_rar.h  -  Copyright (c) 2002 UFPE jpmn@cin.ufpe.br
#ifndef ns_rar_h
#define ns_rar_h
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
```

```
#include "address.h"
#include "ip.h"
struct hdr_rar {
      char ret;
      char raa;
      int id;
      nsaddr_t src, dst;
      double start_time, end_time;
      double peak_rate;
      int phb;
      static int offset_; // required by PacketHeaderManager
      inline static int& offset() { return offset_; }
      inline static hdr_rar* access(const Packet* p) {
            return (hdr_rar*) p->access(offset_);
      }
};
#endif // ns_rar_h


// hdr_rar.cc  -  Copyright (c) 2002 UFPE jpmn@cin.ufpe.br
#include "hdr_rar.h"
int hdr_rar::offset_;
static class rarHeaderClass : public PacketHeaderClass {
public:
      rarHeaderClass() : PacketHeaderClass("PacketHeader/RAR",
                                    sizeof(hdr_rar)) {
            bind_offset(&hdr_rar::offset_);
      }
} class_rarhdr;
```

3. corretor.h e corretor.cc

```
// corretor.h  -  Copyright (c) 2002 UFPE jpmn@cin.ufpe.br
#ifndef ns_corretor_h
#define ns_corretor_h
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
#include "hdr_rar.h"
#include "queue.h"
#define MAX_TABLE 20
struct corretorTable {
    int phb_;     //0=BE 1=AF 2=EF
    double bwtotal_;
    double bwalloc_;
};
struct slatTable {
    int id_;
    nsaddr_t src_, dst_;
    double stime_, etime_, prate_;
    int phb_;     //0=BE 1=AF 2=EF
};
class corretorAgent : public Agent {
public:
```

```
      corretorAgent();
      virtual int command(int argc, const char*const* argv);
      virtual void recv(Packet*, Handler*);
      corretorTable corretor_[MAX_TABLE];
      int corretorEntries;
      void addcorretorTable(int phb_, double bwtotal_, double
bwalloc_);
      int getcorretorTable(int phb_);
      void printcorretorTable();
      slatTable slat_[MAX_TABLE];
      int slatEntries;
      void addslatTable(int id_, nsaddr_t src_, nsaddr_t dst_, double
stime_, double etime_, double prate_, int phb_);
      int getslatTable(nsaddr_t src_, nsaddr_t dst_, int phb_);
      void printslatTable();
protected:
//    Queue *service;
};
#endif // ns_corretor_h


// corretor.cc  -  Copyright (c) 2002 UFPE jpmn@cin.ufpe.br
#include "corretor.h"
#include "hdr_rar.h"
#include "queue.h"
int hdr_rar::offset_;
int hdr_ip::offset_;
static class corretorClass : public TclClass {
public:
      corretorClass() : TclClass("Agent/Corretor") {}
      TclObject* create(int, const char*const*) {
            return (new corretorAgent());
      }
} class_corretor;
corretorAgent::corretorAgent() : Agent(PT_RAR)
{
      bind("packetSize_", &size_);
      corretorEntries = 0;
      slatEntries = 0;
}
int corretorAgent::command(int argc, const char*const* argv)
{
  if (argc == 9) {
    if (strcmp(argv[1], "send") == 0) {
      Packet* pkt = allocpkt();
      hdr_ip* hdrip = hdr_ip::access(pkt);
      hdr_rar* hdr = hdr_rar::access(pkt);
      hdr->ret = 0;
      hdr->raa = 0;
      hdr->id = atoi(argv[2]);
      hdr->src = atoi(argv[3]);
      hdr->dst = atoi(argv[4]);
      hdr->start_time = atoi(argv[5]);
      hdr->end_time = atoi(argv[6]);
      hdr->peak_rate = atoi(argv[7]);
      hdr->phb = atoi(argv[8]);    //0=BE 1=AF 2=EF
      send(pkt, 0);
      return (TCL_OK);
    }
  }
  if (argc == 4) {
```

```
    if (strcmp(argv[1], "addcorretor") == 0) {
      addcorretorTable(atoi(argv[2]), atoi(argv[3]), 0.0);
      return (TCL_OK);
    }
  }
  return (Agent::command(argc, argv));
}
void corretorAgent::addcorretorTable(int phb, double bwtotal, double
bwalloc) {
    if (corretorEntries == MAX_TABLE) {
      printf("ERROR: Corretor Table size limit Exceeded. \n");
    } else {
        corretor_[corretorEntries].phb_ = phb;
      corretor_[corretorEntries].bwtotal_ = bwtotal;
      corretor_[corretorEntries].bwalloc_ = bwalloc;
      corretorEntries++;
    }
}
void corretorAgent::addslatTable(int id, nsaddr_t src, nsaddr_t dst,
double stime, double etime, double prate, int phb) {
    if (slatEntries == MAX_TABLE) {
      printf("ERROR: Slat Table size limit Exceeded. \n");
    } else {
      slat_[slatEntries].id_ = id;
      slat_[slatEntries].src_ = src;
      slat_[slatEntries].dst_ = dst;
        slat_[slatEntries].stime_ = stime;
        slat_[slatEntries].etime_ = etime;
        slat_[slatEntries].prate_ = prate;
        slat_[slatEntries].phb_ = phb;
      slatEntries++;
    }
}
int corretorAgent::getcorretorTable(int phb) {
    for (int i=0; i < corretorEntries; i++) {
      if (corretor_[i].phb_ == phb)   return(i);
    }
    return (-1);
}
int corretorAgent::getslatTable(nsaddr_t src, nsaddr_t dst, int phb)
{
    for (int i=0; i < slatEntries; i++) {
      if ((slat_[i].src_ == src) && (slat_[i].dst_ == dst) &&
(slat_[i].phb_ == phb)) {
          return(i);
      }
    }
    return (-1);
}
void corretorAgent::printcorretorTable() {
    printf("\nCorretor Table:");
    for (int p= 0; p < corretorEntries; p++) {
        printf("\nPhb %d, Bwtotal %d, Bwalloc %d", corretor_[p].phb_,
corretor_[p].bwtotal_, corretor_[p].bwalloc_);
    }
}
void corretorAgent::printslatTable() {
    printf("\nSlat Table:");
    for (int p= 0; p < slatEntries; p++) {
```

```
        printf("\nId %d, Src %d, Dst %d, Stime %d, Etime %d, Prate
%d, Phb %d", slat_[p].id_, slat_[p].src_, slat_[p].dst_,
slat_[p].stime_, slat_[p].etime_, slat_[p].prate_, slat_[p].phb_);
    }
}
void corretorAgent::recv(Packet* pkt, Handler*)
{
  hdr_ip* hdrip = hdr_ip::access(pkt);
  hdr_rar* hdr = hdr_rar::access(pkt);
  if ((hdr->ret == 0) && (hdr->raa == 0)) {
    int getslat, getcorretor;
    char old_raa = hdr->raa;
    int old_id = hdr->id;
    nsaddr_t old_src = hdr->src;
    nsaddr_t old_dst = hdr->dst;
    double old_start_time = hdr->start_time;
    double old_end_time = hdr->end_time;
    double old_peak_rate = hdr->peak_rate;
    int old_phb = hdr->phb;
    getslat = getslatTable(hdr->src, hdr->dst, hdr->phb);
    getcorretor = getcorretorTable(hdr->phb);
    if (getslat == -1) {
        addslatTable(hdr->id, hdr->src, hdr->dst, hdr->start_time,
hdr->end_time, hdr->peak_rate, hdr->phb);
        getslat = getslatTable(hdr->src, hdr->dst, hdr->phb);
    }
    if (corretor_[getcorretor].bwtotal_ >=
corretor_[getcorretor].bwalloc_ + slat_[getslat].prate_) {
      corretor_[getcorretor].bwalloc_ += slat_[getslat].prate_;
        old_raa = 1;
    } else {
        old_raa = 0;
    }
    Packet::free(pkt);
    Packet* pktret = allocpkt();
    hdr_rar* hdrret = hdr_rar::access(pktret);
    hdrret->ret = 1;
    hdrret->raa = old_raa;
    hdrret->id = old_id;
    hdrret->src = old_src;
    hdrret->dst = old_dst;
    hdrret->start_time = old_start_time;
    hdrret->end_time = old_end_time;
    hdrret->peak_rate = old_peak_rate;
    hdrret->phb = old_phb;
    send(pktret, 0);
  } else {
    char out[100];
    sprintf(out, "%s recv %d %d %d", name(),
            hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
          hdr->raa, hdr->id);
    Tcl& tcl = Tcl::instance();
    tcl.eval(out);
    Packet::free(pkt);
  }
}
```

Apêndice B - Glossário

1. Lista de Abreviações

AF	Assured Forwarding
ATM	Asynchronous Transfer Mode
BB	Bandwidth Broker
BE	Best Effort Model
C++	Linguagem orientada a objetos C++
CBQ	Class-Based Queuing
COPS	Common Open Policy Server
Diffserv	Differentiated Services Work Group
DS	Diffserv Mechanism
DSCP	DS Codepoint
EF	Expedited Forwarding
FEC	Forward Equivalence Class
FIFO	First In First Out
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
IEEE	Intitute of Eletrical and Eletronics Engineers
IETF	Internet Engineering Task Force
Intserv	Integrated Services Work Group
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISDN	Integrated Service Digital Network
ISI	Information Sciences Institute
LDP	Label Distribution Protocol
LER	Label Edge Router
LPDP	Local Policy Decision Point
LSP	Label Switched Path
LSR	Label Switching Router
MAMT	Multiple Average Multiple Threshlod
MAST	Multiple Average Single Threshlod

MF	Multi-field classifier
MPLS	Multiprotocol Label Switching
NS	Network Simulator
OSPF	Open Shortest Path First
OTcl	Object Tool Control Language
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PQ	Priority Queueing
QoS	Quality of Service
RED	Random Early Detection
RIO-C	RIO Coupled
RIO-D	RIO De-coupled
RIP	Routing InformationProtocol
RR	Round Robin Scheduler
RSVP	Resource ReServation Protocol
SAMT	Single Average Multiple Threshlod
SAST	Single Average Single Threshlod
SLS	Service Level Specification
srTCM	Single Rate Three Color Marker
TCP	Transport Control Protocol
ToS	Type of Service
trTCM	Two Rate Three Color Marker
TSW	Time Sliding Window
UDP	User Datagram Protocol
VINT	Virtual Internetwork Testbed
WFQ	Weighted Fair Queuing algorithm
WIRR	Weighted Interleaved Round Robin Scheduler
WRED	Weighted RED
WRR	Weighted Round Robin Scheduler