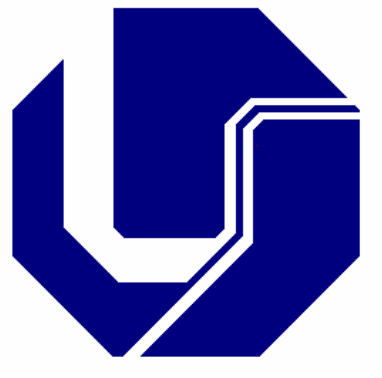
UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



PROPOSTA DE UM MECANISMO PARA ALOCAÇÃO OTIMIZADA DE RECURSOS EM DOMÍNIOS *DIFFSERV*

Lúcio Guimarães dos Reis

Abril

2006

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

PROPOSTA DE UM MECANISMO PARA ALOCAÇÃO OTIMIZADA DE RECURSOS EM DOMÍNIOS DIFFSERV

Lúcio Guimarães dos Reis

Dissertação apresentada à Universidade Federal de Uberlândia para obtenção do título de Mestre em Engenharia Elétrica, aprovada em 20 de Abril de 2006 pela banca examinadora:

Paulo Roberto Guardieiro, Dr. - Orientador (UFU)

Ruy de Oliveira, Ph.D. (CEFET-MT)

Edna Lúcia Flôres, Dra. (UFU)

PROPOSTA DE UM MECANISMO PARA ALOCAÇÃO OTIMIZADA DE RECURSOS EM DOMÍNIOS DIFFSERV

Lúcio Guimarães dos Reis

Dissertação apresentada à Universidade Federal de Uberlândia como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.

Prof. Paulo Roberto Guardieiro, Dr.	Prof. Darizon Alves de Andrade, Dr.
Orientador	Coordenador do curso de Pós-Graduação

Dedicatória

A DEUS,

Aos meus pais Gilberto e Fátima,

À minha esposa Karina, pela compreensão, carinho e apoio,

no cumprimento de mais esta jornada de nossas vidas.

"O primeiro passo para chegar a qualquer lugar é
decidir que não vais permanecer onde estás"
(J. Morgan)

Agradecimentos

Agradeço a Deus pela minha saúde, capacidade e força para a realização de mais este objetivo em minha vida.

Aos meus pais Gilberto e Maria de Fátima, e aos meus irmãos Vinícius e Cecília, pelo amor em todos estes anos, pela minha educação e incentivo aos estudos.

À minha esposa Karina pela compreensão, pelo amor e pela confiança inabalada de sempre em mim.

Ao meu orientador Prof. Dr. Paulo Roberto Guardieiro pela competente orientação, buscando sempre o melhor de mim e o melhor para a faculdade.

À Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia pela minha formação acadêmica, por um ensino de excelência e qualidade, mesmo em um país que destina tão poucos recursos à educação.

A CTBC pelo apoio e a todos aqueles que contribuíram de alguma forma, direta ou indireta, para a realização deste trabalho.

Resumo

Embora o *Bandwidth Broker* tenha sido proposto com o objetivo de agregar maior inteligência às redes *DiffServ*, possibilitando a alocação de recursos de forma dinâmica às aplicações de rede, uma vez concedidos os recursos, estes permanecem alocados estaticamente ao longo do período de reserva. Neste trabalho propõe-se um mecanismo avançado de alocação de recursos baseado no fato de que a carga de tráfego instantânea das aplicações é geralmente dinâmica por natureza e, consequentemente, a largura de banda requerida é por diversas vezes menor do que a largura de banda alocada. Resultados baseados em modelagem e simulação demonstraram que o mecanismo proposto conduz a uma melhor utilização dos recursos e respeita os limites de perda de pacotes e atraso fim-a-fim.

Palavras-chave: Qualidade de Serviço (QoS), *DiffServ*, *Bandwidth Broker* (BB), gerenciamento de recursos, aplicações em tempo real.

Abstract

Although the Bandwidth Broker has been proposed with the objective to give more intelligence to DiffServ networks, allowing the allocation of resources in a dynamic way for applications, once the resources are granted they remain statically allocated along the requested period. In this article, we propose an enhanced resource allocation mechanism, based on the fact that the instantaneous traffic load of the applications are generally dynamic in nature, and consequently, the required bandwidth is less than the allocated bandwidth during some periods. Simulation results demonstrated that the proposed mechanism yields better resource utilization while respecting limits of drop of packets and end-to-end delay.

Keywords: Quality of Service (QoS), SLA, DiffServ, Bandwidth Broker (BB), resource management, real-time applications.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AF: *Assured Forwarding*

BA: *Behavior Aggregate*

BB: *Bandwidth Broker*

CBR: *Constant Bit Rate*

CBS: *Committed Burst Size*

CIR: *Committed Information Rate*

COPS: *Common Open Policy Service*

CoS: *Class of Service*

CS: *Class Selector*

CSCF: *Call Session Control Function*

DiffServ: *Differentiated Services*

DS: *Differentiated Services*

DSCP: *Differentiated Service Code Point*

EF: *Expedited Forwarding*

ETSI: *European Telecommunications Standard Institute*

FIFO: *First In First Out*

FTP: *File Transfer Protocol*

IEEE: *Institute of Electrical and Electronics Engineers*

IETF: *Internet Engineering Task Force*

IntServ: *Integrated Services*

IP: *Internet Protocol*

LDP: *Label Distribution Protocol*

LSP: *Label Switched Path*

LSR: *Label Switched Router*

MBAC: *Measurement Based Admission Control*

MGCP: *Media Gateway Control Protocol*

MPLS: *MultiProtocol Label Switching*

MRFC: *Multimedia Resource Function Controller*

NGN: *Next Generation Networks*

NS: *Network Simulator*

PDB: *Per Domain Behavior*

PHB: *Per Hop Behavior*

PIB: *Policy Information Base*

PIR: *Peak Information Rate*

PQ: *Priority Queueing*

PSTN: *Public Switched Telephone Network*

QoS: *Quality of Service*

RAA: *Request Allocation Answer*

RAR: *Resource Allocation Request*

RR: *Roudin Robin*

RED: *Random Early Detection*

RSVP: *Resource Reservation Protocol*

RTT: *Round Trip Time*

SIP: *Session Initiation Protocol*

SLA: *Service Level Agreement*

SNMP: *Simple Network Management Protocol*

TCP: *Transmission Control Protocol*

TE: *Traffic Engineering*

TOS: *Type of Service*

TSW2CM: *Time Sliding Window Two Color Marker*

TSW3CM: *Time Sliding Window Three Color Marker*

UDP: *User Datagram Protocol*

VBR: *Variable Bit Rate*

WRR: *Weighted Roudin Robin*

Capítulo 1

Introdução

A Internet originalmente foi concebida com o tipo de serviço “Melhor Esforço” (BE - *Best Effort*), o que significa que a Internet fará o possível para transportar cada pacote do remetente ao destinatário, porém, não oferece garantias quanto à perda de pacotes, limites para atrasos e vazão mínima. Além do mais, pacotes de aplicações distintas e/ou de usuários distintos recebem tratamento idêntico pela rede.

Contudo com o advento de um número grande de novas aplicações, no início da década de 80, surgiu a necessidade de que a Internet fosse capaz de prover diferentes níveis de serviço, além do *Best Effort*. A premissa básica era que devido às características técnicas distintas das aplicações, a rede deveria também prover tratamentos distintos.

Juntamente com os requisitos técnicos das aplicações, os requisitos comerciais, como a própria comercialização da Internet pelos provedores de serviço (ISP – *Internet Service Provider*), conduziram à primeira onda de discussão e de desenvolvimento de técnicas que pudessem oferecer “Qualidade de Serviço” (QoS – *Quality of Service*) [1].

O IETF (*Internet Engineering Task Force*) [2] é a entidade mundial responsável pela padronização de tecnologias ligadas à Internet, propôs alguns modelos para a provisão de QoS. Dentre os principais, destacam-se as seguintes arquiteturas: Serviços Integrados (*IntServ - Integrated Services*) [3] e Serviços Diferenciados (*DiffServ - Differentiated Services*) [4].

Atualmente a arquitetura *DiffServ* tem se mostrado como uma solução viável para prover qualidade de serviço nos *backbones* IP dos provedores de serviço. Contudo, uma nova onda de discussão e pesquisa tem se apresentado em virtude da exigência de uma qualidade de serviço mais rígida, de forma que os requisitos de serviço não aceitam variações de performance no tratamento dado pela rede sobre quaisquer condições.

Os principais fatores que têm trazido essas novas exigências são os serviços de multimídia em tempo real e a migração do tráfego telefônico das tradicionais redes comutadas por circuito (PSTN – *Public Switched Telephone Network*) para as redes comutadas por pacotes, particularmente utilizando-se o protocolo IP (*Internet Protocol*) [5]. Alguns autores definem este novo cenário como: *Next Generation Internet* [6], [7] e [8].

Dentro desse contexto, o IETF tem proposto o acréscimo de novos blocos funcionais na arquitetura *DiffServ*, mais especificamente no plano de controle. As primeiras especificações propunham um gerenciamento estático dos recursos da rede, no qual embora fosse relativamente fácil de implementar não era capaz de assegurar uma rígida QoS ou ainda conduzia a uma baixa utilização dos recursos da rede. Já as últimas especificações [9], [10], [11] e [12] propõem um gerenciamento dinâmico dos recursos da rede, através de um processo de controle de admissão. Neste sentido a rede suporta um controle de congestionamento do tipo preventivo, uma vez que o acesso à rede é realizado mediante uma solicitação de recursos pelo cliente.

A decisão sobre aceitar ou rejeitar uma solicitação leva em consideração a carga de tráfego na rede, e este processo conduz a um gerenciamento dinâmico dos recursos da rede no sentido de que a alocação dos recursos ocorre sob demanda. Entretanto, uma vez concedidos os recursos, estes permanecem totalmente alocados durante o período solicitado, independente da utilização total ou não dos recursos.

Portanto, esse tipo de gerenciamento de recursos não leva em conta a carga de tráfego instantânea, que frequentemente é variável por natureza. Isto significa que recursos, como a largura de banda, podem ser desperdiçados nos momentos em que a largura de banda imposta pela carga de tráfego da fonte encontra-se abaixo do valor alocado.

Nesta dissertação propõe-se um mecanismo para alocação de recursos em domínios *DiffServ* através do monitoramento periódico da carga de tráfego instantânea das fontes e no status de recursos do domínio. Assim sendo, esses recursos podem ser periodicamente realocados, de forma que quando a largura de banda utilizada por uma fonte encontra-se abaixo da largura de banda alocada, a porção não utilizada retorna para um *pool* de largura de banda disponível, administrado pelo *Bandwidth Broker* (BB).

Por meio de experimentos baseados em modelagem e simulação, demonstra-se uma melhor utilização dos recursos da rede, possibilitando situações onde a reserva de recursos em um domínio possa ser feita acima de sua capacidade. Além disso, procura-se mostrar que o objetivo básico de um processo de controle de admissão (controle preventivo de congestionamento) será respeitado por meio da análise de parâmetros de QoS, tais como: perda de pacotes e atraso fim-a-fim.

Observa-se que as funções de gerenciamento de recursos e controle de admissão são diretamente interdependentes. Desta forma, propostas de esquemas de gerenciamento de recursos normalmente envolvem adequações no processo de controle de admissão.

Ressalta-se que enquanto a maioria das propostas focaliza em um gerenciamento de recursos anterior ao processo de controle de admissão (pré-controle de admissão), o mecanismo aqui proposto focaliza em um gerenciamento de recursos posterior ao processo de controle de admissão (pós-controle de admissão).

Este trabalho está organizado da seguinte forma:

- No capítulo 2 apresenta-se a evolução dos modelos para provimento de QoS na Internet, destacando-se as principais arquiteturas, características fundamentais, contribuições, limitações e desafios;
- No capítulo 3 faz-se uma descrição das funções dos planos de dados e de controle. Apresentam-se ainda os blocos funcionais de cada um dos planos e a forma como os mesmos se interagem;
- No capítulo 4 discute-se em detalhes o elemento *Bandwidth Broker*, sua principal arquitetura de referência, trabalhos relacionados com esta arquitetura, e por fim, arquiteturas alternativas;
- No capítulo 5 é apresentado o mecanismo para alocação de recursos em domínios *DiffServ*, sua forma de funcionamento, características e principais contribuições. Discutem-se ainda trabalhos relacionados e as diferenças com o aqui proposto;
- No capítulo 6 são detalhados os procedimentos utilizados para a avaliação da proposta apresentada. Descrevem-se os ambientes de rede e de simulação utilizados, bem como os parâmetros e cenários envolvidos nos experimentos. Em seguida, são apresentados os resultados obtidos nas simulações realizadas;
- Finalmente, no capítulo 7 apresentam-se as conclusões gerais, assim como sugestões para trabalhos futuros relacionados com o tema estudado.

Capítulo 2

A Evolução das Arquiteturas/Modelos para Provimento de Qualidade de Serviço na Internet

2.1 Introdução

Neste capítulo descreve-se a evolução das arquiteturas e modelos para a provisão de qualidade de serviço na Internet, apresentando sua evolução cronológica até os dias atuais, e destacando quais foram os fatores motivadores para a formulação de cada um desses.

Na seção 2.2 é definido o conceito e objetivos de qualidade de serviço em redes IP. Na seção 2.3 são descritos as arquiteturas e modelos propostos para prover qualidade de serviço, destacando-se seus conceitos, pontos fortes e pontos fracos. Por fim, na seção 2.4 discuti-se a utilização do protocolo MPLS [13] como uma ferramenta para realizar engenharia de tráfego [14] em redes IP e auxiliar no controle de recursos da rede.

2.2 Definindo Qualidade de Serviço

Define-se como qualidade de serviço o conjunto de requisitos de serviço que devem ser atendidos pela rede, enquanto transporta um determinado fluxo de dados. Os requisitos de serviço são associados a parâmetros mensuráveis capazes de representar o tratamento desejado.

Por sua vez, o tratamento dado pela rede pode ser baseado em questões técnicas, como por exemplo os requisitos mínimos de QoS para determinadas aplicações, ou mesmo baseado em questões comerciais, como por exemplo priorizar o tráfego da empresa “A” em relação ao tráfego da empresa “B”.

Os principais parâmetros de QoS são: largura de banda, atraso fim-a-fim, *jitter* e perda de pacotes. Apresenta-se nas subseções seguintes uma breve definição quanto a cada um desses parâmetros.

Ressalta-se porém que a utilização de QoS em redes IP não significa um acréscimo de recursos na rede. Desta forma, caso alguns pacotes recebam um melhor tratamento, outros irão receber um pior tratamento. Neste sentido, alguns cuidados devem ser considerados ao se projetar uma rede, como por exemplo: evitar que aplicações elásticas (http, ftp, smtp) “morram por inanição”, pois um pacote de menor prioridade nunca é transmitido enquanto houver pacotes de maior prioridade.

2.2.1 Vazão

O parâmetro vazão é definido como sendo o número de bytes (ou bits) por segundo recebidos pelo destinatário da conexão. É uma grandeza finita, normalmente limitada pelas

características físicas do meio de comunicação, capacidade dos enlaces, protocolos, etc.

2.2.2 Atraso fim-a-fim

Atraso fim-a-fim é o tempo decorrido desde a geração de uma mensagem em um sistema terminal até o seu recebimento no sistema terminal destinatário.

O atraso fim-a-fim resulta do somatório dos atrasos nodais ao longo de sua rota. O atraso nodal é composto dos seguintes componentes:

- Atraso de processamento: é o tempo requerido para examinar o cabeçalho do pacote e determinar para onde direcioná-lo. Esse tipo de atraso depende principalmente da capacidade de processamento do nó, e o atraso sofrido em cada pacote é praticamente constante;
- Atraso de fila: é o tempo que um pacote espera em uma fila até que esteja disponível para ser transmitido no enlace. Esse tipo de atraso depende da quantidade de pacotes que chegaram antes e que já estão na fila esperando pela transmissão no enlace. Desta forma, o atraso sofrido em cada pacote pode variar significativamente;
- Atraso de transmissão (também chamado de serialização): é o tempo exigido para a transmissão de todos os bits do pacote para o enlace. Esse tipo de atraso depende basicamente do tamanho do pacote (em bits) e da velocidade de transmissão do enlace (em bits por segundo);
- Atraso de propagação: é o tempo necessário para que o pacote se propague por toda a extensão do enlace. Esse tipo de atraso depende basicamente do meio físico do enlace (fibra óptica, fio de cobre, cabo coaxial, etc), e o atraso sofrido em cada pacote é praticamente constante.

2.2.3 *Jitter*

Devido ao fato de que componentes do atraso fim-a-fim são variáveis, o próprio atraso fim-a-fim tem comportamento variável. O *jitter* é justamente a diferença entre os atrasos fim-a-fim de pacotes subsequentes.

2.2.4 Perda de Pacotes

Perda de pacotes é a razão entre a quantidade de pacotes perdidos e a quantidade de pacotes enviados. Os pacotes podem ser perdidos na rede por descarte nas filas, devido ao congestionamento nos roteadores, ou ainda devido a erros detectados pelo roteador nos pacotes (relação com a taxa de erros de bits).

2.3 Arquiteturas e modelos para a provisão de QoS

2.3.1 Modelo Orientado ao Pacote

A RFC 791 [15], publicada em 1981, descreve um modelo de tratamento de pacotes através da designação de indicadores em cada um dos pacotes. Cada pacote tem um octeto chamado de TOS (*Type of Service*), onde a importância do pacote (campo *Precedence*) e as necessidades do serviço (campo TOS) são identificadas.

De acordo com a RFC 791:

- O campo TOS provê uma indicação dos parâmetros abstratos da qualidade de serviço desejada. Esses parâmetros são usados para conduzir a seleção dos atuais

parâmetros do serviço, quando transmitindo um datagrama IP pela rede. Diversas redes oferecem serviço de precedência, que de certa forma trata o tráfego de maior precedência como sendo o mais importante do que outros tráfegos. A principal questão é quanto à escolha dos seguintes tratamentos: baixo atraso, alta disponibilidade e alta vazão.

Infelizmente, nenhuma arquitetura foi desenvolvida utilizando-se desses campos e, na maioria dos casos, nenhum mecanismo sequer existia na rede para prover o tratamento diferenciado aos pacotes. Uma tentativa de redefinir o campo TOS foi feita na RFC 1349 [16], através da definição de novos parâmetros de tratamento. Contudo, o modelo ainda não era capaz de associar valores mensuráveis aos parâmetros do campo TOS.

Observa-se que ambas as RFCs operam sob a suposição implícita de que o campo TOS pode ser caracterizado em uma escala linear, variando de melhor ou pior, contudo sem uma métrica quantificável.

2.3.2 Modelo Orientado ao Serviço

No início da década de 90, com os primeiros experimentos com áudio e vídeo na Internet, surgia um novo interesse em qualidade de serviço em redes IP. Em vista disso, foi proposto o modelo chamado de Serviços Integrados [3] (mais tarde chamado de *IntServ-Integrated Services*) através da RFC 1633, publicada em 1994.

A RFC 1663 apontava ainda outras questões relacionadas à qualidade de serviço:

- Qualidade de serviço para aplicações em tempo real não é a única questão para uma próxima fase no gerenciamento de tráfego na Internet. Operadores de rede estão solicitando a capacidade de controlar o compartilhamento de largura de banda em

um determinado enlace em diferentes classes de tráfego. Os operadores querem ser capazes de dividir o tráfego em poucas classes e designar para cada uma dessas uma percentagem mínima da largura de banda do enlace, mesmo em condições de sobrecarga. Essas classes podem representar diferentes grupos de usuários, ou diferentes famílias de protocolos. Tal facilidade de gerenciamento é comumente chamada de enlace compartilhado controlado.

A arquitetura *IntServ* definiu o conceito de qualidade de serviço ao nível de fluxo[1] e mecanismos de controle de admissão e reserva de recursos. Propunha-se que os recursos (como por exemplo: largura de banda) fossem explicitamente gerenciados de forma a corresponder com as necessidades das aplicações. Isto implica que a reserva de recursos e o controle de admissão são blocos fundamentais presentes em cada roteador *IntServ*, conforme ilustra a Figura 2.1.

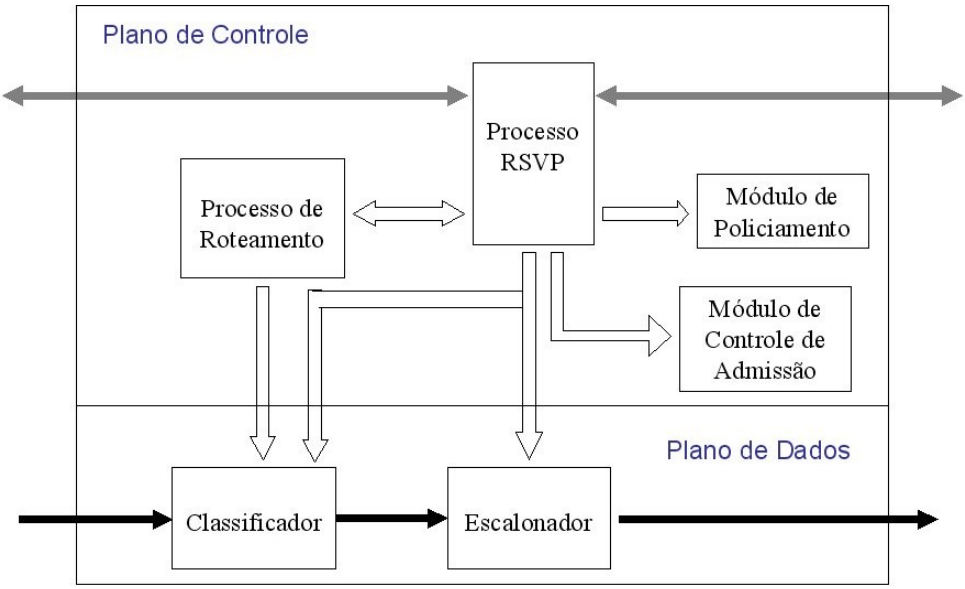


Figura 2.1: Roteador *IntServ*

[1] Um fluxo pode ser definido de várias formas. Uma forma comum refere-se a uma combinação de endereços IP de origem e destino, números de porta de origem e destino e um identificador de sessão. Uma definição mais ampla é que um fluxo é um conjunto de pacotes gerados de uma determinada aplicação, interface ou *host*.

O processo de reserva de recursos e controle de admissão é realizado pelo protocolo de sinalização RSVP (*Resource Reservation Protocol*) [17], através do envio de mensagens de sinalização em cada roteador, ao longo do caminho do fluxo. O protocolo tem como finalidade verificar a disponibilidade de recursos em cada roteador e, caso haja disponibilidade, proceder com a reserva de recursos conforme a necessidade do fluxo. Caso não haja disponibilidade de recursos em qualquer um dos roteadores, então é negada a admissão do fluxo na rede. A Figura 2.2 ilustra as mensagens do protocolo RSVP.

No contexto da arquitetura *IntServ* foram propostos inicialmente 2 classes de serviços:

- Serviço de Carga Controlada (CLS - *Controlled Load Service*) [18]: seu objetivo é atender às aplicações tolerantes ao atraso e pequenas perda de pacotes, em casos de congestionamento na rede;
- Serviço Garantido (GS - *Guaranteed Service*) [19]: garante um limite de atraso fim-a-fim e de perdas de pacotes nas filas dos roteadores. Esse serviço pretende suportar aplicações com requisitos rígidos de QoS, como as aplicações em tempo real.

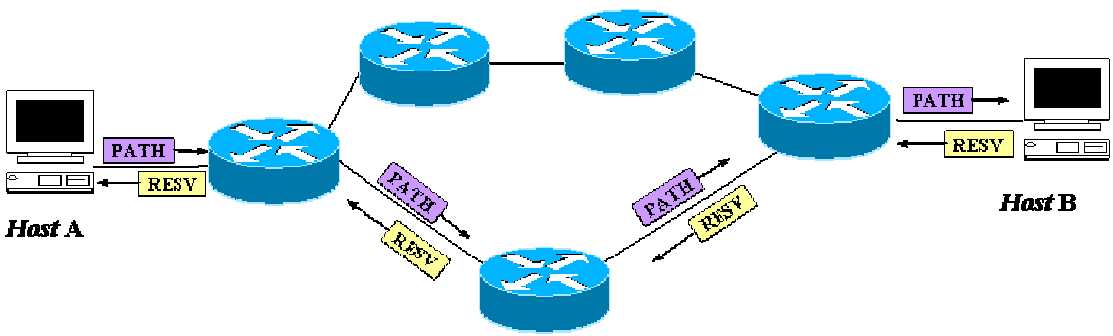


Figura 2.2: Mensagens do Protocolo de Sinalização RSVP

Observa-se aqui uma analogia bastante próxima quanto ao procedimento de estabelecimento (*call-setup*) de uma chamada telefônica nas tradicionais centrais de

comutação por circuitos, onde um protocolo de sinalização verifica a disponibilidade de recursos e faz a reserva necessária para que depois então se inicie a transmissão dos dados.

A principal vantagem da arquitetura *IntServ* é a obtenção de uma QoS com granularidade, também visto como uma QoS quantitativa. Esta performance é obtida graças à utilização do fluxo como unidade fundamental e os seus processos de reserva de recursos e controle de admissão.

Contudo, essas mesmas características trazem consigo um sério problema de escalabilidade, uma vez que os roteadores devem manter o estado de cada fluxo na rede. Um outro problema é que a arquitetura *IntServ* desconsidera a Internet como um conjunto de sistemas autônomos (AS – *Autonomous System*) independentes administrativamente e conectados de forma hierárquica.

Em suma, apesar da arquitetura *IntServ* ter apresentado fortes mecanismos para a provisão de QoS, fatores relacionados com a escalabilidade e o conflito com a própria arquitetura da Internet foram responsáveis pela não adoção dessa arquitetura.

2.3.3 Modelo Orientado ao Domínio da Rede e ao Pacote

Em 1998 a RFC 2474 apresentava a arquitetura *DiffServ* [4] como um modelo escalável, capaz de ser implementado em fases, acompanhando o crescimento da rede e dentro de domínios de rede administrativos independentes.

Na arquitetura *DiffServ* a unidade fundamental é o tráfego agregado, que consiste de fluxos individuais com os mesmos requisitos de QoS, sendo que cada tráfego agregado é designado a uma classe de serviço (CoS – *Class of Service*). Desta forma, fluxos individuais que pertençam ao mesmo tráfego agregado receberão tratamento semelhante

pela rede, sendo que este tratamento é chamado de comportamento agregado (BA – *Behavior Aggregated*).

A identificação de cada CoS é feita de forma explícita pela marcação de bits no cabeçalho do pacote IP no campo DSCP (*Differentiated Service Code Point*). Este campo é na verdade uma redefinição do campo TOS (*Type of Service*), sendo que agora somente 6 bits do total de 8 são utilizados.

Desta forma a marcação do campo DSCP determina o tratamento que será dado no encaminhamento dos pacotes em cada roteador e tal característica é chamada de "comportamento por salto" (PHB - *Per Hop Behavior*).

O IETF definiu 3 PHBs, em adição ao serviço padrão *Best-Effort*:

- *Class Selector* (CS) – RFC 2474 [20]: procura assegurar compatibilidade com os bits *IP Precedence* usado no campo TOS;
- *Expedited Forwarding (EF)* – RFC 2598 [21] e RFC 3246 [22]: foi especificado para serviços que requerem um baixo atraso, pequenas perdas de pacotes, controle do *jitter* e largura de banda assegurada. A idéia é fazer com que pacotes com o DSCP marcado como EF encontrem filas de encaminhamento muito pequenas nos roteadores. Diante dessas características, o EF PHB destina-se principalmente a aplicações multimídia em tempo real e missão crítica. Este tipo de serviço é normalmente referido como "linha privativa virtual" (VLL – *Virtual Leased Line*), devido ao fato de que os recursos da rede são designados praticamente de forma exclusiva.
- *Assured Forwarding (AF)* – RFC 2597 [23]: foi especificado para serviços que requerem perda de pacotes controlada e largura de banda assegurada, porém não apresenta garantias quanto a atraso e *jitter*. AF PHB pode ser subdividido em 4

classes, sendo que para cada classe é alocada uma quantidade de recursos como espaço em *buffer* e largura de banda, e dentro de cada uma destas classes existem 3 níveis de prioridade de descarte.

Enquanto que na arquitetura *IntServ* os processos de reserva de recursos e controle de admissão são realizados diretamente pelos roteadores, na arquitetura *DiffServ* estes processos são movidos para o nível do domínio da rede. Desta forma, são as políticas do domínio que determinam quais pacotes serão admitidos junto à rede.

Essas políticas residem normalmente em uma entidade central, e são aplicadas nas bordas da rede. Mantêm informações restritas ao domínio, como por exemplo: valores de DSCPs associados aos tráfegos agregados, como também informações sobre os contratos de prestação de serviço (SLA – *Service Level Agreement*), negociados entre o domínio e os seus clientes.

O SLA discrimina o comportamento que o tráfego do cliente deve receber dentro do domínio. Contemplam parâmetros, tais como: taxa de pico de informação (PIR – *Peak Information Rate*), taxa de informação comprometida (CIR – *Committed Information Rate)*, tamanho da rajada comprometida (CBS – *Committed Burst Size*), atraso fim-a-fim, *jitter*, entre outros. A tradução do SLA em uma informação técnica apropriada é chamada de especificação técnica do serviço (SLS – *Service Level Specification*).

Na arquitetura *DiffServ* a complexidade da rede é direcionada para as bordas, permitindo o tratamento e o encaminhamento de pacotes no núcleo da rede de forma simples e rápida.

Os roteadores de borda (também chamados de *Edge Router*) devem executar as seguintes funções:

- Classificação: associa o tráfego ao seu respectivo SLA;

- Medição: mede as taxas de geração do tráfego (PIR, CIR, CBS);
- Marcação: designa o valor de DSCP correspondente ao PHB esperado pelos pacotes junto à rede;
- Policiamento: determina ações para aqueles pacotes que estejam fora do perfil do SLA. Por exemplo: descarta os pacotes, designa um valor de DSCP com uma menor qualidade de serviço, retarda o encaminhamento dos pacotes.

Já os roteadores de núcleo ou interior da rede aplicam o PHB de acordo com a marcação no campo DSCP, utilizando-se de técnicas de gerenciamento de filas, escalonamento e descarte de pacotes. A Figura 2.3 ilustra as funções dos roteadores de borda e de núcleo.

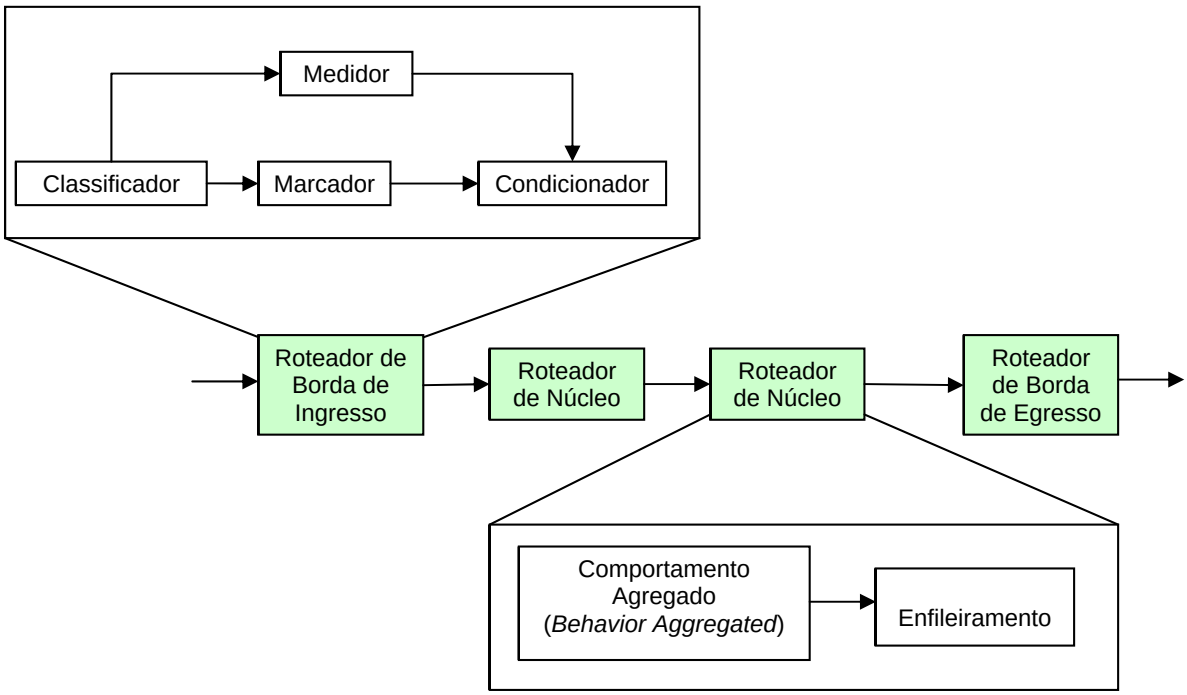


Figura 2.3: Arquitetura *DiffServ*

Define-se como “comportamento por domínio” (PDB - *Per Domain Behavior*) o tratamento uniforme recebido por uma determinada classe de serviço, ao longo de todo o domínio.

A remoção dos processos de reserva de recursos e controle de admissão do nível dos roteadores para o nível do domínio da rede deve ser visto como uma separação entre o plano de dados e o plano de controle. A Figura 2.4 ilustra o processo de controle de admissão na arquitetura *DiffServ*.

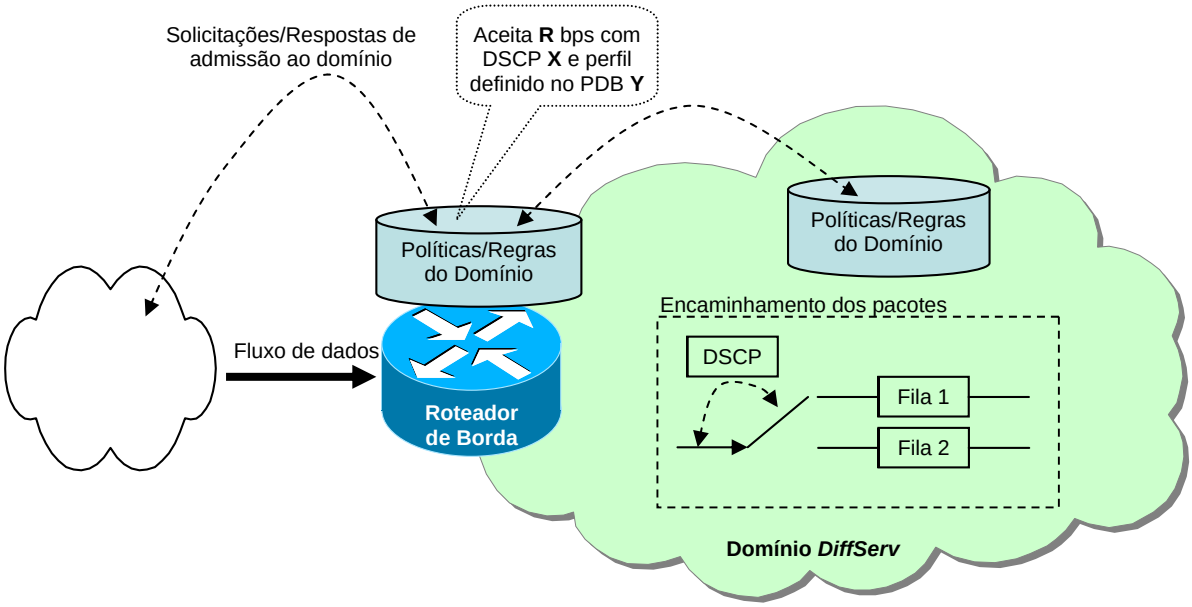


Figura 2.4: Controle de Admissão na Arquitetura *DiffServ*

No processo de controle de admissão um fluxo é admitido junto a um tráfego agregado, associado a um determinado PDB. Esse processo pode ser realizado de forma estática ou dinâmica, sendo que nesse último caso é necessário um protocolo de sinalização por fluxo.

Ainda com relação ao processo de controle de admissão, observa-se que as características mencionadas abaixo são importantes para assegurar a escalabilidade da arquitetura *DiffServ*:

- Mudanças no controle de admissão, como por exemplo, nas políticas do domínio, podem resultar em reconfigurações na borda da rede, contudo, sem alterar as configurações do interior da rede;
- A decisão quanto à aceitação ou não do fluxo é centralizada em uma base de dados, refletindo as políticas do domínio, e a mesma é distribuída aos roteadores de borda;

Comparando-se ainda com a arquitetura *IntServ*, a arquitetura *DiffServ* reconhece a Internet como uma interconexão de redes independentes administrativamente e conectados de forma hierárquica, de forma que um domínio *DiffServ* pode ser visto como um sistema autônomo (AS – *Autonomous System*).

Apesar da arquitetura *DiffServ* ter se apresentado como uma solução escalável desde a sua concepção, existem ainda alguns desafios a serem atingidos:

- Provisão de uma qualidade de serviço rígida com limites bem definidos. O fato de que o tratamento dado pela rede é baseado no tráfego agregado, torna-se de certa forma um obstáculo para a garantia de QoS no nível de fluxos individuais, uma vez que todos os fluxos pertencentes àquele tráfego agregado receberão o mesmo comportamento (BA – *Behavior Aggregated*). Isto faz com que as pequenas diferenças de requisitos de QoS entre os fluxos sejam tratadas de forma uniforme. Outro ponto também relacionado com a própria arquitetura é que as classes de serviços recebem tratamentos diferenciados, no sentido de que uma classe de serviço receberá melhor tratamento em detrimento de outra, define-se então apenas aspectos qualitativos;
- Assegurar que os PHBs sejam invariantes em todo o domínio, independente da carga de tráfego agregado em cada um dos roteadores, de forma que o PHB realizado em cada roteador seja uniforme o bastante para que o tratamento recebido

por um determinado tráfego agregado em todo o domínio seja caracterizado como um PDB;

- Desenvolver um mecanismo de reserva de recursos e controle de admissão interdomínios, seja de forma estática ou dinâmica (através de um protocolo de sinalização).

As respostas para as questões apresentadas acima são temas abertos à pesquisa e desenvolvimento e têm sido tratadas através da especificação e padronização das funções do plano de controle.

Algumas das propostas consideram, por exemplo:

- A utilização de engenharia de tráfego pelo domínio como um mecanismo para auxiliar no controle da utilização dos recursos de rede;
- A utilização do protocolo RSVP, ou mesmo uma variante deste, como o protocolo de sinalização interdomínio.

2.4 MPLS-TE como Ferramenta Auxiliar no Provimento de QoS

2.4.1 Sobre o MPLS

O protocolo MPLS (*MultiProtocol Label Swiching*) surgiu inicialmente como uma técnica para encaminhamento rápido de pacotes que opera entre as camadas 2 (enlace) e 3 (rede).

Seu princípio é baseado no processo de “etiquetar” ou “rotular” cada pacote IP com um cabeçalho específico na borda da rede MPLS, de forma que a partir deste momento o

pacote seja encaminhado até o outro extremo da rede através de um circuito virtual.

O roteador responsável por inserir/retirar o rótulo MPLS nos pacotes IP é chamado de LER (*Label Edge Router*) e está localizado nas bordas da rede. A Figura 2.5 apresenta o cabeçalho MPLS, composto por 4 bytes, distribuídos em 4 campos.

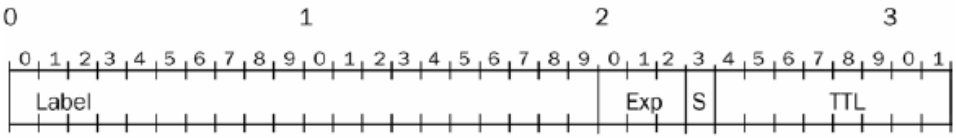


Figura 2.5: Cabeçalho MPLS

Os rótulos podem ser distribuídos na rede de forma estática ou dinâmica, por um protocolo específico. Um protocolo bastante conhecido é o LDP (*Label Distribution Protocol*) [24]. Já os roteadores chamados de LSR (*Label Switching Router*), utilizam-se destes rótulos para encaminhar os pacotes dentro de circuitos virtuais, chamados de LSP (*Label Switching Path*). Os LSPs são estabelecidos entre dois roteadores LER. A Figura 2.6 ilustra o encaminhamento de um pacote IP dentro de uma rede MPLS.

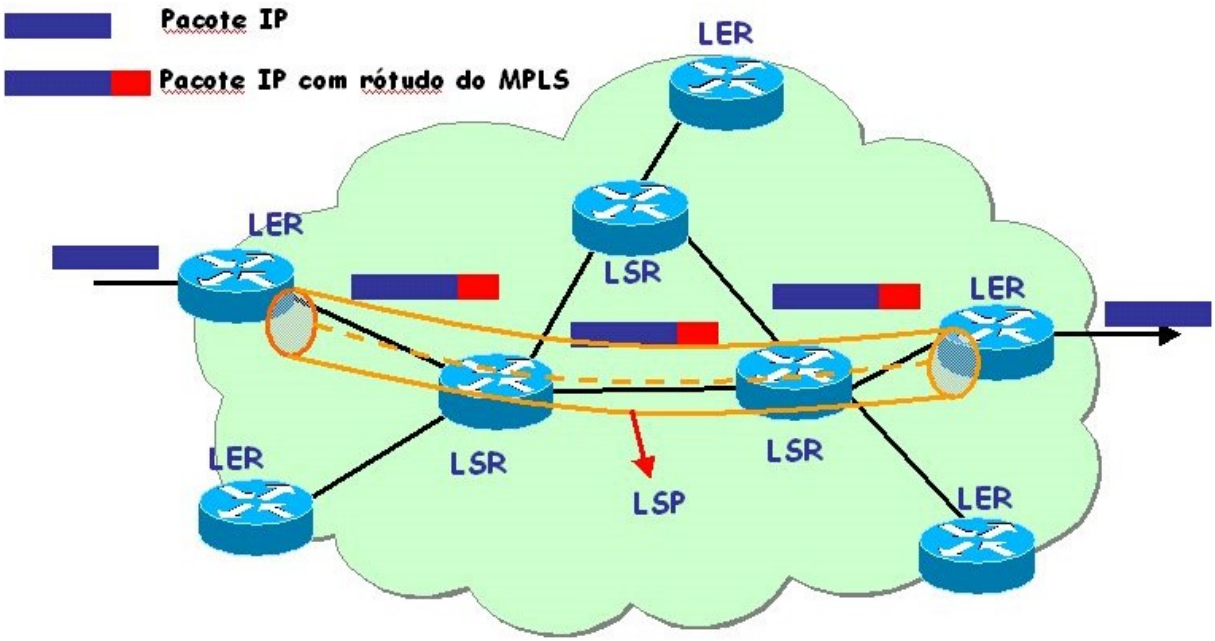


Figura 2.6: Arquitetura MPLS

Observa-se que em uma mesma interface física do roteador é possível ter vários LSPs, cada um definindo um circuito virtual. Têm-se ainda a divisão dos LSPs em troncos, sendo que cada tronco representa um conjunto de fluxos de dados com os mesmos requisitos de QoS. A representação dos fluxos de dados, troncos, LSPs, e interface física são ilustradas na Figura 2.7.

Os fluxos de dados agrupados no mesmo tronco recebem o mesmo tratamento pela rede, conhecido como "classe equivalente de encaminhamento" (FEC- *Forwarding Equivalent Class*). Observa-se aqui uma convergência muito próxima com a arquitetura *DiffServ* no que diz respeito ao conceito de tronco e tráfego agregado, FEC e BA.

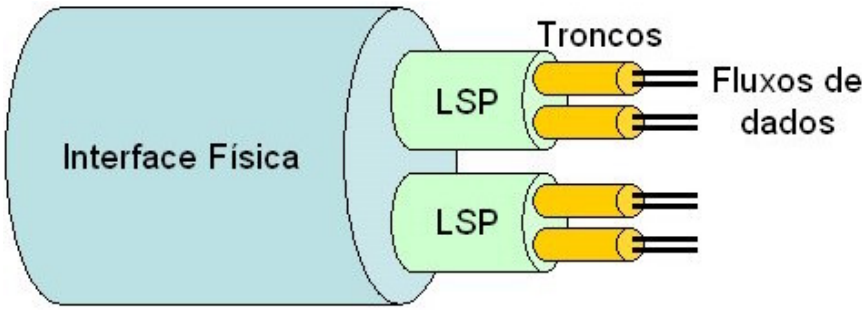


Figura 2.7: Fluxos de dados, troncos e LSPs

O IETF apresentou propostas para a utilização em conjunto de *DiffServ/MPLS*, com destaque para a RFC3270 [25]. O princípio básico é que os valores de DSCPs sejam mapeados de forma adequada no cabeçalho do MPLS. Para isto existem duas propostas:

- E-LSP (*EXP Inferred Perhop Scheduling Class LSP*): O campo DSCP (6 bits) do *DiffServ* é mapeado no campo EXP (3 bits) do cabeçalho do MPLS;
- L-LSP (*Label Inferred PSC LSP*): O campo DSCP (6 bits) do *DiffServ* é mapeado no campo LABEL (20 bits) do cabeçalho do MPLS.

2.4.2 Engenharia de Tráfego

Os protocolos de roteamento utilizados nas redes IP são na sua maioria baseados em algoritmos que determinam o menor caminho entre dois *hosts* distintos. Já as redes baseadas em MPLS são capazes de oferecer roteamento explícito, ou seja, um administrador da rede pode determinar o caminho pelo qual os pacotes serão encaminhados na rede. Como por exemplo: definir um caminho baseado na disponibilidade de largura de banda, ou ainda, definir caminhos redundantes para utilização no caso de falhas.

É esta característica de suportar roteamento explícito que torna o MPLS uma ótima ferramenta para prover engenharia de tráfego (TE – *Traffic Engineering*). Por sua vez a capacidade de um administrador de rede em realizar engenharia de tráfego proporciona um maior controle sobre os recursos da rede, e consequentemente potencializa a sua utilização. Como exemplo, pode-se realizar engenharia de tráfego com o objetivo de distribuir o tráfego na rede de forma homogênea, fazendo uso uniforme dos recursos da rede e evitando-se situações de congestionamento, conforme ilustra as Figuras 2.8 e 2.9.

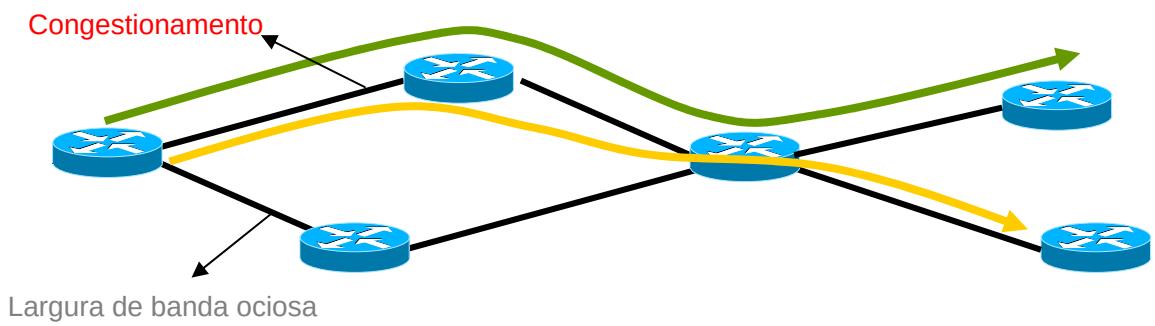


Figura 2.8: Rede MPLS sem TE

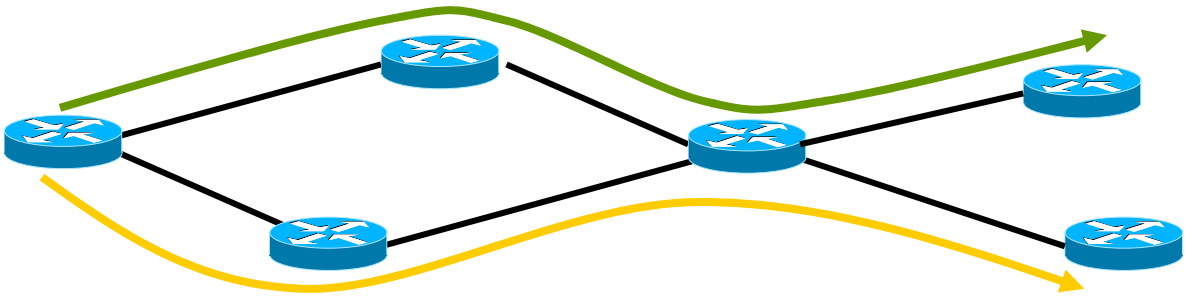


Figura 2.9: Rede MPLS com TE

Inicialmente a única forma de executar engenharia de tráfego era estaticamente, através de um administrador de rede. Contudo, com o objetivo de automatizar esse processo foram propostos alguns protocolos, tais como: CR-LDP [26] e OSPF-TE [27]. De forma geral esses protocolos estabelecem circuitos virtuais ou rotas com base em restrições ou premissas atribuídas pelo administrador de rede. Por exemplo, uma restrição poderia ser: estabelecer o circuito com o menor atraso fim-a-fim, ou ainda com a maior largura de banda. Observa-se que nesse processo as restrições devem ser armazenadas em todos os roteadores da rede, conforme ilustra a Figura 2.10.

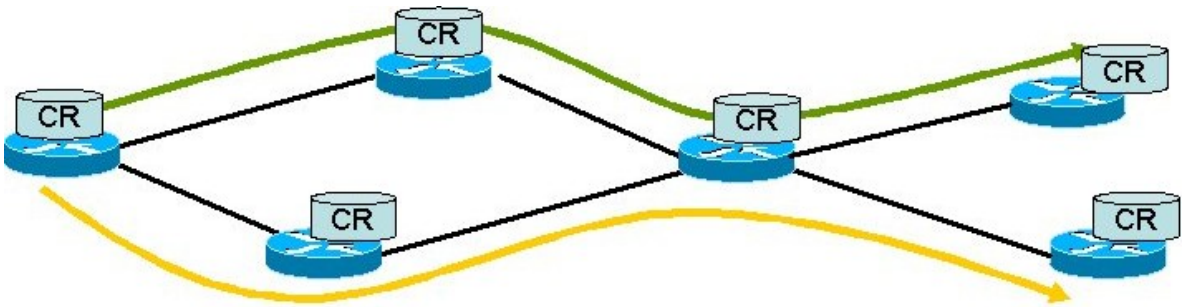


Figura 2.10: Processo distribuído para automatização de TE

Contudo, ressalta-se ainda uma nova proposta para automatizar o processo de engenharia de tráfego alinhado com a arquitetura *DiffServ*. A idéia é levar o processo de engenharia de tráfego para junto do plano de controle do *DiffServ*, de forma que teria-se

uma base de dados centralizada com as informações de restrições do domínio, interagindo com outras bases de dados centrais, tais como: políticas do domínio, topologia da rede, status de utilização dos enlaces e dos circuitos virtuais. A Figura 2.11 ilustra este cenário.

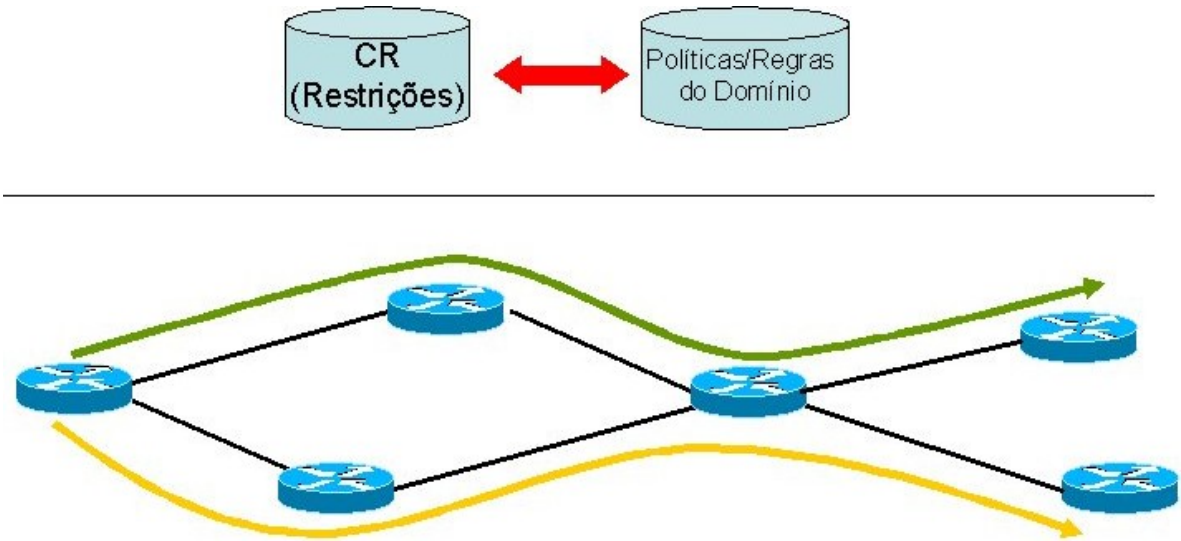


Figura 2.11: Processo centralizado para automatização de TE

2.5 Conclusões

Neste capítulo foi apresentado inicialmente o objetivo de aplicar-se qualidade de serviço, sendo que em seguida foi apresentada a evolução dos modelos propostos para prover qualidade de serviço, ressaltando suas características e modos de funcionamento.

Mostrou-se que o desenvolvimento das arquiteturas foi baseado inicialmente no modelo orientado ao próprio pacote IP (*IP Precedence/TOS*). A idéia era de que através das informações contidas no cabeçalho IP dos pacotes, a rede pudesse realizar o tratamento adequado aos pacotes. Contudo, questões como a carência de uma definição clara sobre

qual o tratamento que deveria ser dado pela rede, e também na definição de métricas quantificáveis, impediram a implantação desse modelo.

Já a arquitetura seguinte (*IntServ*) é baseada no modelo orientado ao serviço oferecido pela rede. A idéia era de que as próprias aplicações pudessem sinalizar à rede suas necessidades de QoS, informando parâmetros tais como: atraso fim-a-fim, largura de banda, *jitter*, entre outros. Foi nessa arquitetura que funções do plano de controle foram inicialmente apresentadas, tais como: controle de admissão e reserva de recursos. Apesar da arquitetura ser capaz de oferecer uma qualidade de serviço rígida, nunca chegou a ser implantada em grandes *backbones* IP, devido a questões de escalabilidade e conflitos com a própria arquitetura da Internet.

Já a arquitetura (*DiffServ*) é baseada no modelo orientado ao pacote IP e ao domínio administrativo do provedor de serviços, e é a arquitetura atualmente utilizada nos *backbones* IP dos provedores de serviços. Seu sucesso é atribuído à capacidade de escalabilidade na rede, e dentre alguns fatores relacionados, ressalta-se a separação dos planos de dados e de controle em diferentes níveis. Entretanto, ressaltam-se também alguns dos desafios a serem atingidos por essa arquitetura, sendo um tema aberto a pesquisas, principalmente nas funções do plano de controle.

Por fim, apresentou-se como o protocolo MPLS pode ser utilizado em conjunto com a arquitetura *DiffServ* para prover um maior controle dos recursos da rede, e consequentemente, garantir uma rígida qualidade de serviço.

Capítulo 3

Planos de Dados e de Controle na Arquitetura *DiffServ*

3.1 Introdução

Plano de dados é o plano onde efetivamente ocorre a transmissão dos pacotes IP entre um *host* origem e um *host* destino, passando por elementos de rede intermediários, de acordo com a estrutura TCP/IP.

As principais funções do plano de dados estão diretamente relacionadas com o tratamento aplicado pela rede aos pacotes IP, através de mecanismos de escalonamento, gerenciamento de filas, marcadores e condicionadores de tráfego.

Já o plano de controle é responsável por coordenar, monitorar e executar ações junto ao plano de dados de forma a prover as seguintes funções:

- ✓ **Controle de admissão:** verifica se a rede dispõe de recursos para aceitar o pedido

de um novo fluxo sem afetar a qualidade dos fluxos já existentes na rede;

- ✓ **Gerenciamento de recursos da rede:** procura promover uma máxima utilização dos recursos da rede, fazendo com que os recursos estejam totalmente alocados em qualquer instante de tempo;
- ✓ **Provisão de recursos na rede**: responsável pelas informações de configuração nos roteadores de borda da rede, de acordo com as políticas do domínio.

Na arquitetura *IntServ* os planos de dados e controle estão justapostos no mesmo nível. Isto porque cada roteador envolvido no encaminhamento dos pacotes de dados, também realiza controle de admissão e reserva de recursos através do protocolo de sinalização RSVP.

Já na arquitetura *DiffServ* as funções de controle estão separadas do plano de dados, de forma que os roteadores de núcleo não mantêm qualquer informação de reserva de recursos, seja por fluxo ou tráfego agregado, conforme é descrito em [34] e [35]. A Figura 3.1 ilustra esta separação.

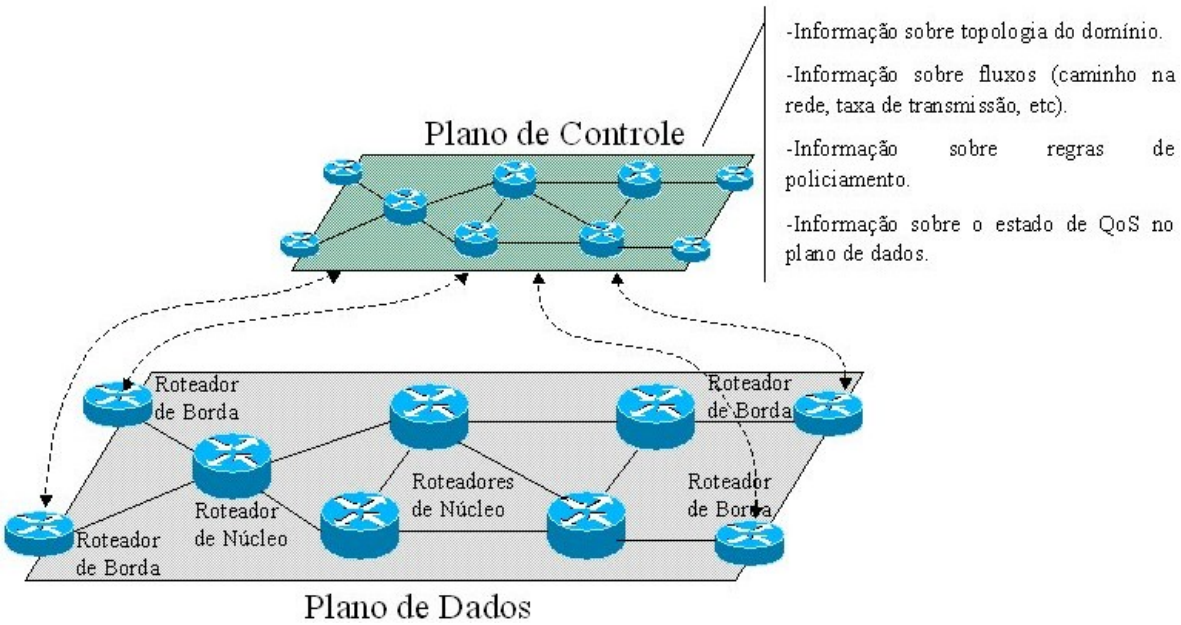


Figura 3.1: Plano de controle e de dados

Na seção 3.2 deste capítulo são apresentados os blocos funcionais do planto de dados, enquanto que na seção 3.3 apresentam-se os blocos funcionais do plano de controle.

3.2 Blocos Funcionais do Plano de Dados

3.2.1 Classificadores de Tráfego

A classificação consiste em associar um fluxo de dados a um determinado tráfego agregado, de acordo com as políticas do domínio, refletindo assim o tratamento dado pela rede. Definem-se dois tipos de classificadores:

- Classificador BA: executa a classificação dos pacotes baseando-se somente no conteúdo do campo DSCP;
- Classificador MF (*Multiple Field*): executa a classificação dos pacotes levando em conta diversos campos do cabeçalho do pacote, tais como: endereço de origem, endereço de destino, porta de origem, porta de destino, ID (identificador) do protocolo, entre outros.

3.2.2 Marcadores de tráfego

A marcação consiste basicamente em definir o valor do campo DSCP, baseado no processo de classificação. Por exemplo: uma aplicação de VoIP é classificada como um tráfego de alta prioridade, a partir daí os pacotes são marcados com um determinado DSCP que reflete a qualidade de serviço desejada. Comenta-se na próxima seção em quais situações os valores de DSCP podem ser remarcados.

3.2.3 Medidores/Condicionadores de Tráfego

Os medidores e condicionadores de tráfego têm como objetivo medir alguns parâmetros do tráfego gerado pelo cliente e assegurar que estes valores não ultrapassem os valores previamente acordados no SLA (esta atividade também é chamada de policiamento). Os principais parâmetros analisados são:

- CIR (*Committed Information Rate*): É a quantidade de pacotes que podem ser transmitidos dentro de um intervalo de tempo;
- PIR (*Peak Information Rate*): É a quantidade de pacotes que podem ser transmitidos em um curto intervalo de tempo;
- CBS (*Committed Burst Size*): É o número máximo de pacotes transmitidos em um intervalo de tempo extremamente curto (comumente chamado de rajada).

O tráfego do cliente é dito estar em conformidade (*in-profile*) se todos os seus parâmetros estiverem com valores abaixo ou igual daqueles acordados no SLA. Por outro lado, caso qualquer parâmetro esteja com o valor acima daquele acordado no SLA, esse tráfego será considerado não em conformidade (*out-of-profile*).

As seguintes ações podem ser executadas quando o tráfego está em não conformidade:

- Os pacotes IP serão descartados;
- Os pacotes IP podem ser remarcados com novos valores de DSCPs que representam uma maior probabilidade de descarte do que a designada anteriormente. A quantidade de novos valores de DSCPs e a relação de prioridade de descarte entre esses depende do tipo de condicionador de tráfego utilizado;
- Os pacotes são retidos no *buffer* como forma de aumentar o intervalo de tempo entre o envio de pacotes subsequentes. Essa ação é comumente denominada de *shaping*.

3.2.3.1 TSW3CM

O condicionador de tráfego TSW3CM (*Time Sliding Window Three Color Marker*) [36] foi projetado para classificar os pacotes IP com as seguintes cores: verde, amarelo e vermelho, representando cada uma destas cores a prioridade de descarte dos pacotes. Pacotes considerados *in-profile* serão classificados como verde e possuirão a menor prioridade de descarte de pacotes. Já os pacotes classificados como amarelo e vermelho serão considerados *out-of-profile* e possuirão uma prioridade de descarte de pacotes média e alta, respectivamente.

A classificação é baseada na medição da taxa de transmissão do tráfego, comparado com os parâmetros de CIR e PIR estabelecidos no SLA. Pacotes com taxas de transmissão abaixo ou igual ao CIR são classificados como verde. Pacotes com taxas de transmissão acima do CIR e abaixo do PIR são classificados como amarelo, e por fim pacotes com taxas de transmissão acima do PIR são classificados como vermelho.

O condicionador TSW3CM consiste de dois componentes independentes: um estimador da taxa de transmissão e um marcador, conforme ilustra a Figura 3.2.

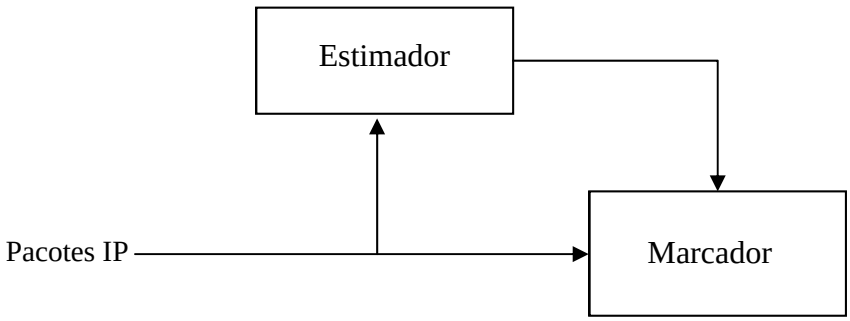


Figura 3.2: Diagrama de blocos TSW3CM

3.2.3.2 TSW2CM

O condicionador de tráfego TSW2CM (*Time Sliding Window Two Color Marker*) representa uma variante simplificada do TSW3CM. Este condicionar utiliza-se somente do parâmetro de CIR e de duas cores, verde e vermelho.

Pacotes considerados *in-profile* serão classificados como verde e possuirão uma baixa prioridade de descarte de pacotes. Já os pacotes classificados como vermelho serão considerados *out-of-profile* e possuirão uma alta prioridade de descarte de pacotes.

3.2.3.3 Token Bucket

O condicionador de tráfego *Token Bucket* [37] utiliza-se dos parâmetros de CIR e CBS, e de dois níveis de classificação, representando diferentes probabilidades de descarte de pacotes. Para entender quando um pacote é considerado *in-profile* ou *out-of-profile* faz-se necessário compreender o seu algoritmo.

O algoritmo define “r” como a taxa de transmissão do tráfego e “b” como o tamanho máximo de uma rajada. Fazendo-se uma analogia com um balde, tem-se: “b” como a capacidade máxima do balde e “r” como a taxa na qual fichas são inseridas regularmente.

Uma ficha corresponde à permissão para transmitir uma quantidade de bits. Quando chega um pacote o seu tamanho é comparado com a quantidade de fichas no balde. Se existir uma quantidade suficiente de fichas, o pacote é considerado *in-profile*, caso contrário, o pacote é considerado *out-of-profile*. O funcionamento do algoritmo *Token Bucket* é ilustrado na Figura 3.3.

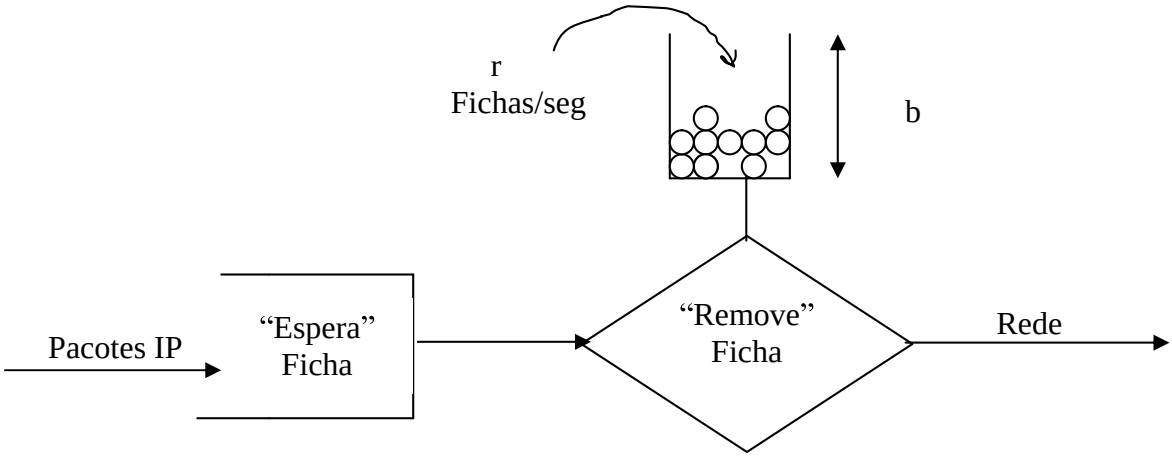


Figura 3.3: Funcionamento do *Token Bucket*

3.2.4 Gerenciadores de filas

De forma geral os gerenciadores de filas têm como objetivo:

- Distribuir o tráfego de dados em filas específicas, conforme a classe de serviço designada para cada pacote;
- Monitorar o volume de pacotes armazenados no *buffer* do roteador para cada fila;
- Descartar pacotes em situações de congestionamento, conforme a política de QoS do gerenciador de fila utilizado.

O algoritmo de gerenciamento de fila básico nos roteadores é conhecido como *Drop Tail*. Este algoritmo simplesmente armazena qualquer pacote no *buffer* desde que haja espaço suficiente, de forma que o descarte de pacotes ocorrerá quando o *buffer* estiver totalmente cheio.

3.2.4.1 RED

O gerenciador de fila RED (*Random Early Detection*) [38] procura detectar situações de

congestionamento de forma preventiva, através do monitoramento to tamanho médio da fila e definindo limiares mínimo e máximo para controle. A Figura 3.4 ilustra o funcionamento do gerenciador de fila RED.

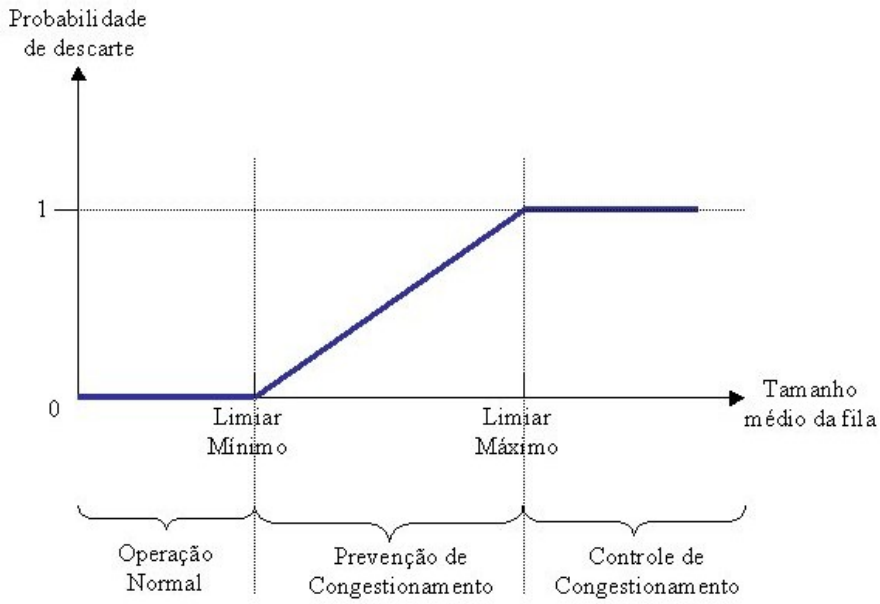


Figura 3.4: Funcionamento do gerenciador de fila RED

Dependendo do tamanho médio da fila e dos limiares definidos, os pacotes podem passar pelas seguintes fases:

- Operação Normal: caso o tamanho médio da fila seja menor do que o limiar mínimo, nenhum pacote é descartado;
- Prevenção de Congestionamento: caso o tamanho médio da fila esteja entre os limiares mínimo e máximo, pacotes são descartados de acordo com uma determinada função probabilidade;
- Controle de Congestionamento: caso o tamanho médio da fila seja maior do que o limiar máximo, todos os pacotes recebidos são descartados.

Na arquitetura *DiffServ* existe a possibilidade de que cada fila física seja subdividida em filas virtuais, representando cada uma delas diferentes níveis de prioridade de descarte de pacotes. Estes níveis de prioridade de descarte são definidos ajustando-se os limiares mínimos e máximos em cada fila virtual.

3.2.5 Mecanismos de Escalonamento

Os mecanismos de escalonamento (comumente chamados de *schedulers*) têm como objetivo definir a política de como os pacotes enfileirados são escolhidos para transmissão no enlace de saída do roteador. Este procedimento denomina-se disciplina de escalonamento.

Os primeiros modelos de roteadores possuíam somente uma fila e utilizavam-se da disciplina de escalonamento FIFO (*First-In-First-Out*). Dessa forma, todo e qualquer tipo de pacote era enfileirado numa única fila, e transmitido no enlace de saída respeitando-se a ordem de chegada, ou seja, pacotes que chegam primeiro são transmitidos primeiro. Observa-se então que este mecanismo não é capaz de oferecer diferenciação de tratamento.

Desta forma, novos tipos de escalonadores foram definidos com a finalidade de que os pacotes atingissem o enlace de saída de acordo com as suas respectivas prioridades. Estes escalonadores baseiam-se no processo de classificação e marcação para enfileirar pacotes, de forma que pacotes com a mesma marcação são encaminhados a uma mesma fila, determinando assim a sua classe de serviço. Examinam-se a seguir outras disciplinas de escolonamento importantes.

3.2.5.1 *Priority Queueing* (PQ)

Utiliza-se filas distintas para cada classe de prioridade e as filas são servidas de acordo com as suas respectivas prioridades. Assim, ao escolher um pacote a ser transmitido, o escalonador PQ decidirá sempre por um pacote da classe de maior prioridade, desde que haja pacotes na fila.

A Figura 3.5 ilustra o processo de enfileiramento prioritário de pacotes. O uso do escalonador PQ é bastante útil quando se quer garantir um atraso fim-a-fim máximo a um determinado tipo de tráfego.

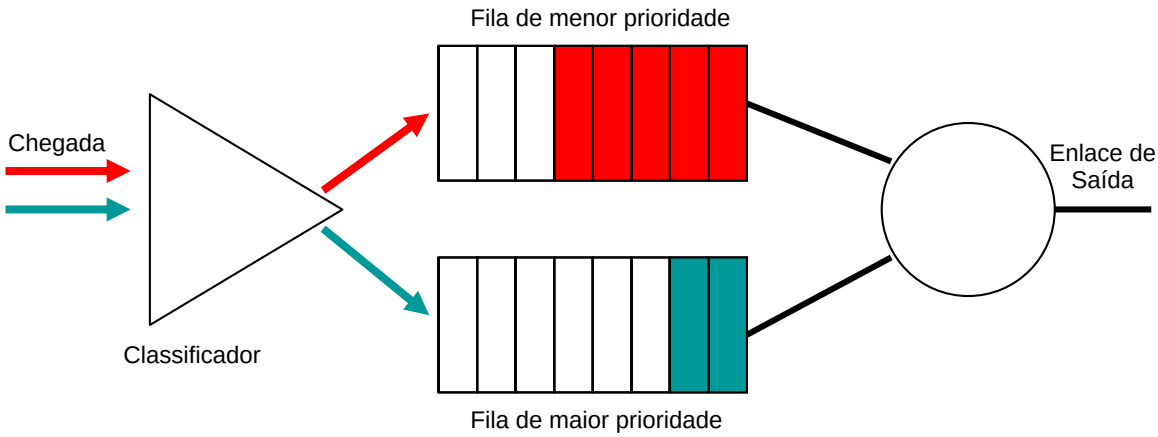


Figura 3.5: Enfileiramento prioritário de pacotes

Na utilização do escalonador PQ existe a possibilidade de ocorrer "inanição" (*starvation*) nas demais filas, isto porque pacotes nas filas de menor prioridade nunca poderão ser encaminhados enquanto houver pacotes nas filas de maior prioridade. Para se evitar essa situação, logo após a classificação, deve-se policiar o tráfego, de forma a evitar que o tráfego em não conformidade ocupe toda a largura de banda.

3.2.5.2 WRR

O escalonador WRR (*Weighted Round Robin*) é uma variação do escalonador RR (*Round Robin*). Este último escalonador serve suas filas através de um mecanismo de varredura cíclica. Por exemplo: considerando-se que existam 3 filas (cada uma representando uma classe de serviço), o escalonador RR servirá a 1ª fila, enviando um pacote para o enlace de saída, depois servirá a 2ª fila, enviando um pacote para o enlace de saída, depois servirá a 3ª fila, enviando um pacote para o enlace de saída, e então se inicia novamente o ciclo. Ou seja, o escalonador RR realiza uma varredura em todas as filas de forma cíclica, servindo um pacote por vez em cada uma dessas filas.

O escalonador WRR utiliza-se do mesmo processo de varredura cíclica, a diferença está relacionada ao fato de que são atribuídos pesos às filas, de forma que estes pesos determinam quantos pacotes devem ser servidos pelo escalonador em cada fila durante um ciclo. A Figura 3.6 ilustra o funcionamento do escalonador WRR.

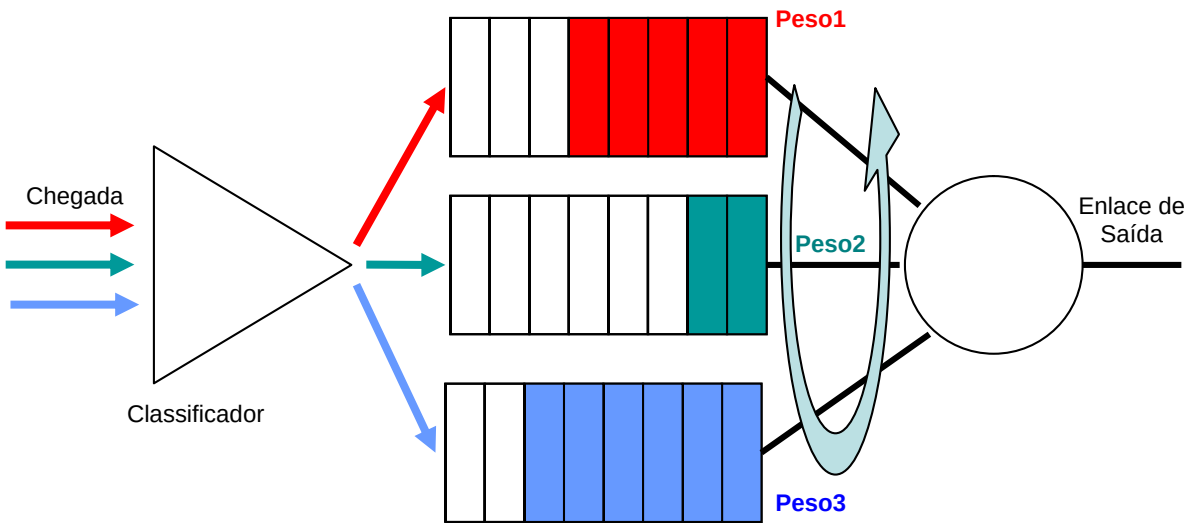


Figura 3.6: Escalonador WRR

3.3 Blocos Funcionais do Plano de Controle

3.3.1 Gerenciamento de Recursos

O gerenciamento de recursos da rede opera em conjunto com o processo de controle de admissão e procura promover uma máxima utilização dos recursos da rede, sem comprometer a qualidade de serviço solicitada pelos clientes.

O gerenciamento de recursos pode ser realizado de duas formas: estático ou dinâmico. No gerenciamento estático os recursos da rede são provisionados de forma estática pelo administrador da rede e eventualmente alterada pelo mesmo dentro de longos períodos de tempo. Esta forma de gerenciamento é relativamente simples, contudo, não consegue maximizar a utilização dos recursos da rede.

Já no gerenciamento dinâmico, os recursos da rede são provisionados de forma dinâmica através da utilização de protocolos de sinalização e de aprovisionamento. Neste caso os recursos da rede são alocados na medida em que são demandados pelos clientes. Apesar desta forma de gerenciamento ser mais complexa que a anterior, proporciona uma melhor utilização dos recursos da rede.

As subseções seguintes apresentam as características do gerenciamento estático e dinâmico.

3.3.1.1 Gerenciamento Estático

Nas primeiras especificações da arquitetura *DiffServ* os SLAs entre os clientes e os seus provedores de serviços ocorriam de forma pré-estabelecida e definitiva, o que conduzia a

uma alocação de recursos totalmente estática.

Embora este modelo seja escalável e fácil de implementar, não é capaz de prover uma qualidade de serviço confiável, e pode ainda desperdiçar os recursos da rede. Isto porque no gerenciamento estático o policiamento executado pelo roteador de borda de ingresso junto ao tráfego do cliente não leva em consideração a disponibilidade de recursos do domínio. O policiamento realizado no roteador de borda tem como objetivo verificar se o tráfego enviado pelo cliente encontra-se dentro do SLA acordado.

O exemplo a seguir ressalta essa característica. Considera-se aqui um cenário simples, conforme ilustrado na Figura 3.7. O provedor de serviço dispõe de uma rede com capacidade para atender uma carga de tráfego de até 30 Mbps, e possui um enlace "gargalo" para interconexão à Internet com capacidade de 15 Mbps.

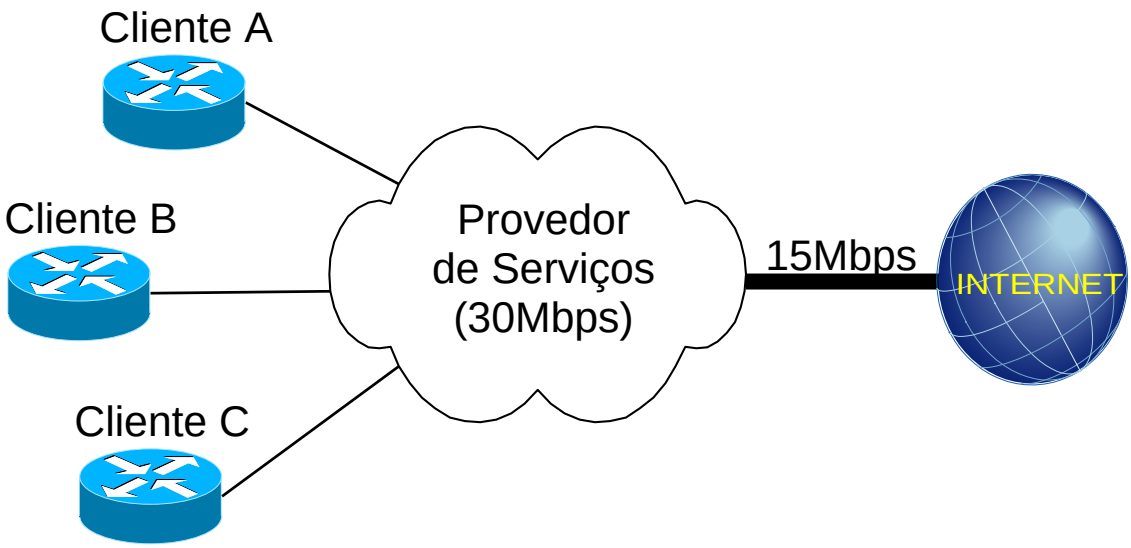


Figura 3.7: Exemplo de gerenciamento estático

Supõe-se que inicialmente o provedor de serviço estabeleça um SLA com os clientes A, B e C com CIR igual a 10 Mbps, e que em um determinado momento todos esses clientes utilizem suas conexões à Internet via enlace "gargalo".

Observa-se que a partir deste momento a carga de tráfego será excedida no enlace, e consequentemente os pacotes IP serão descartados ou atrasados em demasia, após utilizar substanciais recursos da rede. Neste caso o provedor de serviços não pôde honrar o SLA acordado com os seus clientes.

Alternativamente, o provedor de serviços poderia estabelecer um SLA com os clientes A, B e C com CIR igual a 5 Mbps. Neste caso, o provedor de serviços honraria o SLA acordado, entretanto conduziria a uma utilização altamente ineficaz dos recursos da rede, uma vez que somente 15 Mbps estariam sendo utilizados, da capacidade total de 30 Mbps.

3.3.1.2 Gerenciamento Dinâmico

Com o objetivo de alocar os recursos da rede de uma forma mais inteligente foi introduzido na arquitetura *DiffServ* o elemento chamado de *Bandwidth Broker* (BB) [9], conforme ilustra a Figura 3.8. O *Bandwidth Broker* é responsável por executar as funções do plano de controle. Dentre estas, destacam-se as seguintes:

- Alocação de recursos de forma dinâmica, do ponto de vista que os recursos são alocados no momento da solicitação (sob demanda). Neste caso, apesar do SLA ter sido pré-negociado entre o cliente e o seu provedor de serviços, a alocação de recursos somente será realizada após uma solicitação explícita por parte do cliente, condicionada à disponibilidade de recursos naquele período;
- Controle de congestionamento do tipo preventivo: Por meio do processo de controle de admissão o BB verifica se os recursos solicitados pelo cliente podem ser suportados pela rede. Este processo é mais complexo do que um simples policiamento na borda da rede, pois leva em consideração o estado de QoS da rede,

verificando se existem recursos suficientes no domínio, não somente na borda da rede, mas também no caminho a ser percorrido dentro da rede;

- Clara separação entre o plano de dados e de controle, assim como dos elementos responsáveis por executar as funções associadas a cada plano. Ao contrário do modelo de controle de admissão da arquitetura *IntServ*, onde todos os roteadores participam na sinalização nó a nó na reserva de recursos, no modelo "*DiffServ* + BB" as funções de controle estão separadas do plano de dados, de forma que os roteadores de núcleo não mantêm qualquer informação de reserva de recursos, seja por fluxo ou tráfego agregado.

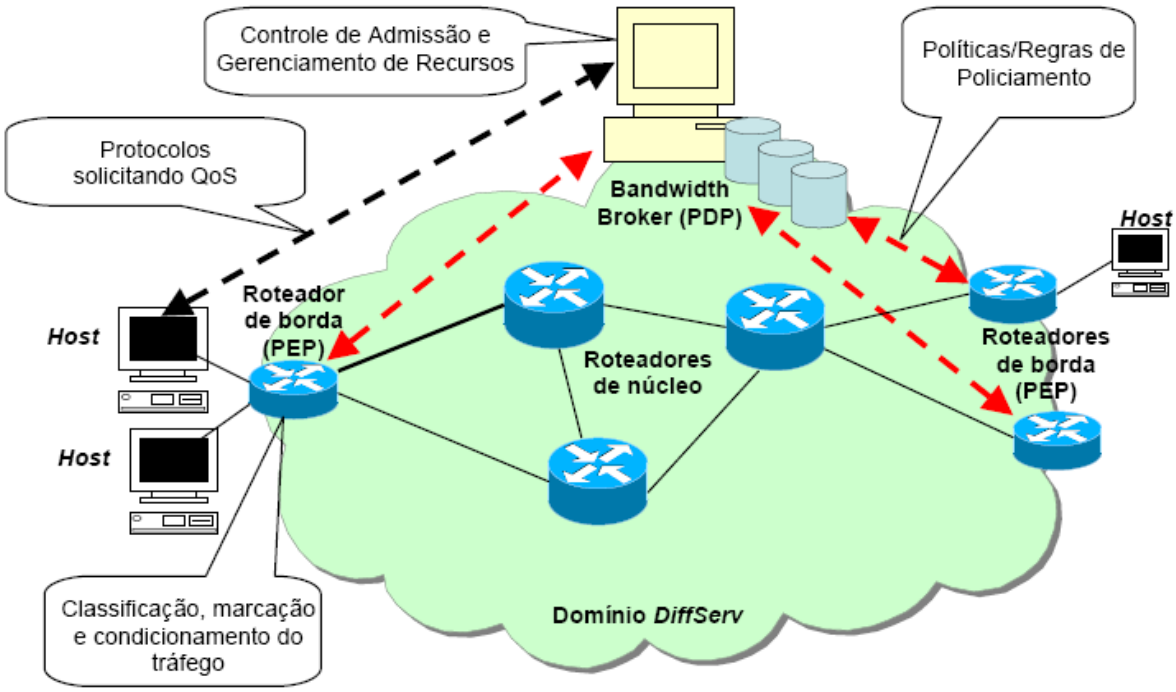


Figura 3.8: Arquitetura *DiffServ* com o BB

Os artigos [39] a [42] abordam diferentes técnicas de gerenciamento dinâmico de recursos.

3.3.2 Controle de Admissão

O controle de admissão tem como tarefa verificar se a rede dispõe de recursos suficientes para prover os requisitos de QoS solicitados por um novo fluxo de dados, sem afetar a qualidade de serviço dos fluxos já existentes. Verifica-se ainda se a solicitação deste novo fluxo encontra-se de acordo com os parâmetros de QoS previamente estabelecido no SLA.

Caso as condições mencionadas sejam satisfeitas, os recursos da rede são então provisionados, configurando os roteadores de borda da rede com as políticas do domínio. O exemplo abaixo ilustra as etapas do processo de controle de admissão:

- Uma aplicação de VoIP solicita ao domínio uma largura de banda mínima de 12 Kbps e um atraso fim-a-fim máximo de 100 ms, entre dois *hosts* distintos;
- O domínio por sua vez verifica se a rede dispõe de recursos e se os recursos solicitados encontram-se de acordo com o SLA;
- Caso as condições acima sejam cumpridas, o domínio então classifica este fluxo como sendo de alta prioridade e a sua marcação com o DSCP da classe EF;
- Em seguida, estas informações são transferidas para os roteadores de borda.

A solicitação de recursos pode ocorrer de forma explícita através de um protocolo de sinalização. No caso específico de aplicações de telefonia IP, as principais entidades de padronização do setor de telecomunicações, entre estas o 3GPP [43], *Packet Cable* [44] e *MultiService Switching Forum* [45], definiram uma arquitetura de rede chamada NGN (*Next Generation Networks*) para o provimento do serviço de telefonia sobre uma rede IP, utilizando-se dos seguintes protocolos de sinalização: H.323 [46], SIP [47], MGCP [48] e MEGACO/H.248 [49]. A Figura 3.9 apresenta a arquitetura NGN.

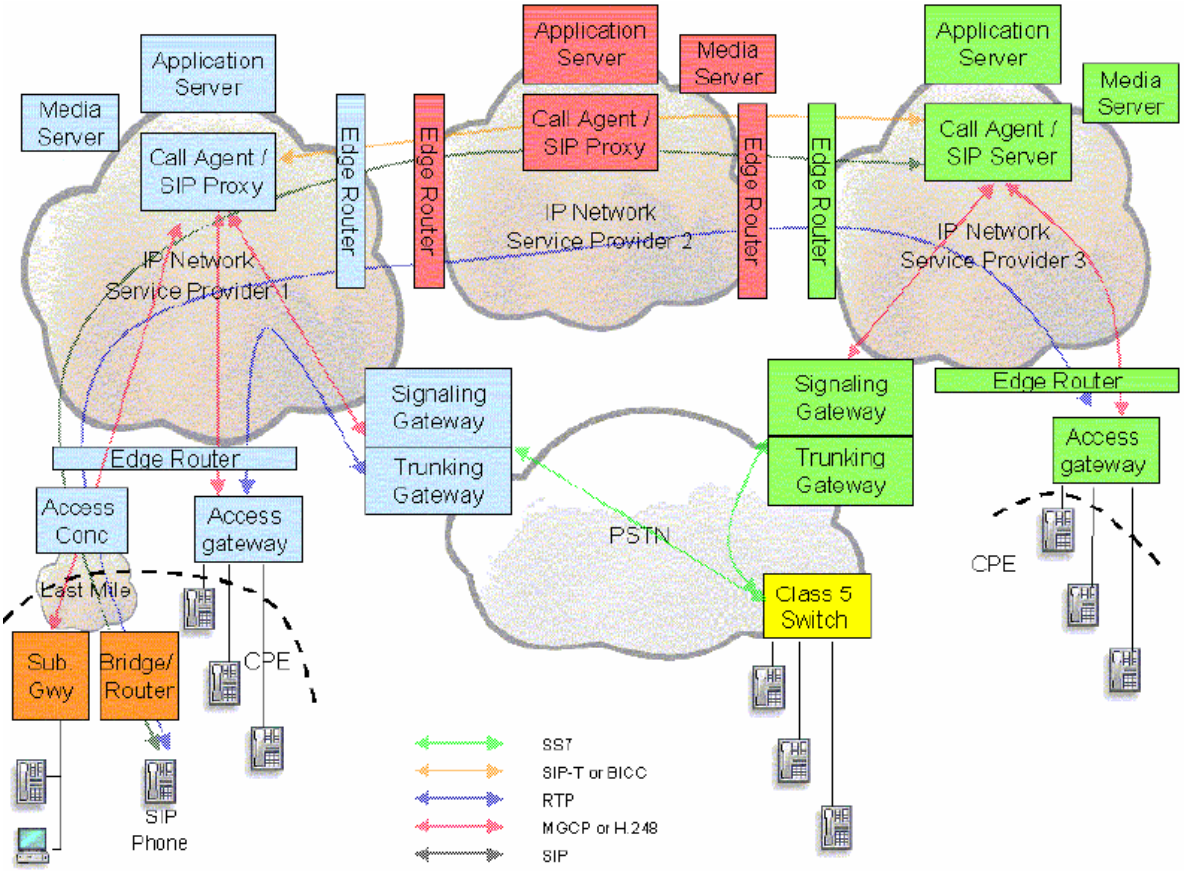


Figura 3.9: Arquitetura NGN [45]

Já o processo de verificação de recursos na rede consiste em estimar a carga de tráfego atual na rede, acrescentar a carga de tráfego solicitada pelo fluxo, e então comparar com a disponibilidade de recursos na rede.

Para estimar a carga de tráfego existente na rede existem basicamente dois métodos: *Measurament Based* e *Parameter Based*, nos quais existem diversas técnicas atualmente propostas para ambas as abordagens, tais como em [50] e [51].

Em linhas gerais o método *Parameter Based* considera que a rede conhece previamente o perfil ou característica de cada fluxo, através de parâmetros como: CIR, PIR e CBS, e a partir daí são realizados cálculos estatísticos para estimar a carga de tráfego. Contudo, esse método apresenta algumas dificuldades devido aos seguintes fatores:

- Tráfegos que apresentam comportamentos diferentes dentro de escalas de tempo, e logo não seguem um único padrão;
- Possibilidade de que o tráfego seja sobre ou sub-estimado;
- Demanda uma certa complexidade computacional devido aos cálculos relacionados com os parâmetros de cada tráfego;
- É necessário que o roteador de borda mantenha estado de cada fluxo.

Já o método *Measurament Based* estima a carga de tráfego através de um processo de medições diretamente sobre o tráfego em questão, dentro de um intervalo de tempo que é chamado de janela de medição. A idéia é que dentro dessa janela ocorram várias medições subsequentes, de forma a caracterizar a carga de tráfego.

A principal questão que surge com relação a esse método é na definição da periodicidade da janela de medição. Pequenos intervalos entre as janelas de medição proporcionam uma melhor representação da carga de tráfego atual na rede, e consequentemente uma maior precisão no processo de controle de admissão. Contudo, as próprias medições produzem um acréscimo na carga de tráfego na rede, assim quanto menor for o intervalo das janelas, maior será a carga de tráfego adicional.

Por outro lado, um grande intervalo entre as janelas de medição proporciona uma rede mais estável, contudo pode ocorrer falhas no processo de controle de admissão, uma vez que a caracterização da carga de tráfego não será tão fiel e poderá levar a uma tomada de decisão baseada em uma informação desatualizada.

O artigo [52] apresenta uma comparação entre os dois métodos de estimativa da carga de tráfego. A Figura 3.10 ilustra a largura de banda real utilizada por um determinado tráfego ao longo do tempo e as suas estimativas através dos métodos *Parameter Based* e *Measurament Based* (com periodicidade de 2 e 10 minutos entre as janelas).

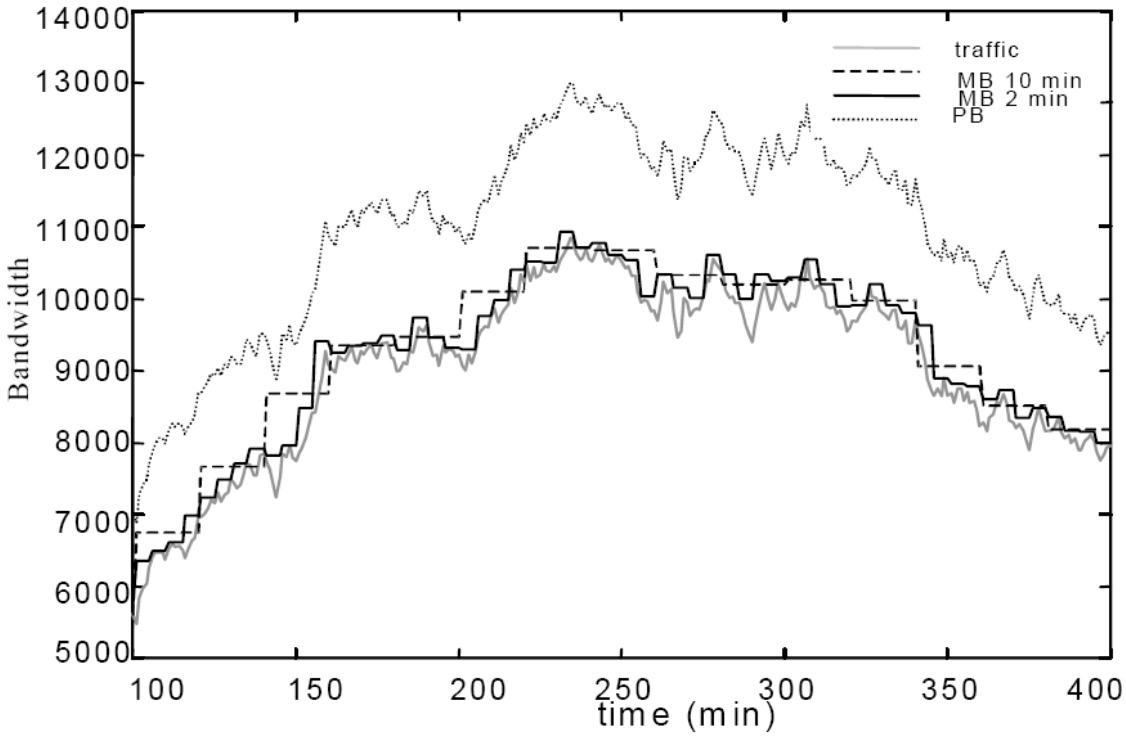


Figura 3.10: Estimativa de tráfego via *Parameter Based* e *Measurament Based* [52]

De forma geral os métodos de estimativa da carga de tráfego devem prover informações quanto ao status dos enlaces, dos circuitos virtuais e do ganho estatístico devido a multiplexação das redes de pacotes.

3.3.3 Provisão de Recursos

O processo de alocação de recursos na rede ocorre através de um mecanismo de aprovisionamento, responsável pelo envio das políticas e regras do domínio aos roteadores de borda, e utiliza-se de um protocolo de aprovisionamento específico para a comunicação entre os planos de dados e de controle.

Dentre os principais protocolos de aprovisionamento destaca-se o COPS (*Commom Open Policy Service*) [53]. Sendo que este apresenta duas variantes COPS/RSVP [54] e

COPS-PR [55], de acordo com o mecanismo utilizado no processo de solicitação de recursos (controle de admissão). A Figura 3.11 ilustra os 2 cenários.

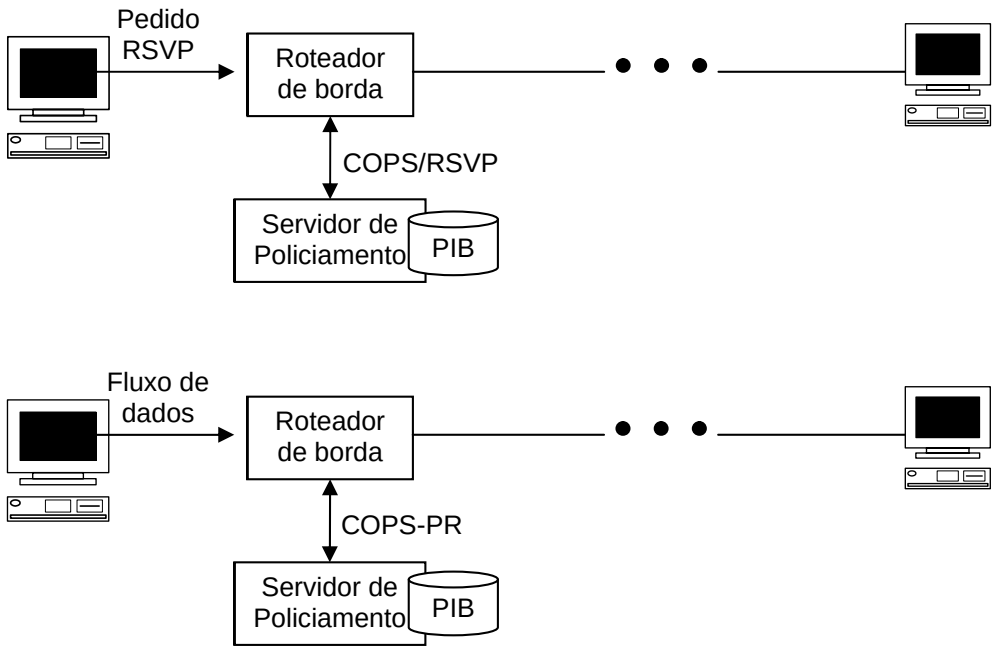


Figura 3.11: Uso do COPS-PR e do COPS/RSVP para aprovisionamento de recursos

O protocolo COPS/RSVP é utilizado quando o processo de solicitação de recursos é realizado de forma explícita pelo protocolo RSVP. Observa-se aqui que a utilização do protocolo RSVP, nesta situação, não se enquadra na arquitetura *IntServ*, sendo que o seu escopo é somente de atuar como um protocolo de sinalização junto ao processo de solicitação de recursos. Nesta situação, o roteador de borda encaminha imediatamente o pedido RSVP ao servidor de policiamento que aceita ou rejeita o pedido, de acordo com a disponibilidade de recursos. Caso o pedido seja aceito, o servidor de policiamento atualiza então o roteador com as informações de políticas e regras do domínio. Estas informações são chamadas de PIB (*Policy Information Base*).

Já o protocolo COPS-PR é utilizado quando alguma ação de aprovisionamento parte diretamente do servidor de policiamento, e não do roteador de borda, como no caso

anterior. Nesta situação, o servidor de policiamento transfere de forma pró-ativa as informações de políticas e regras do domínio ao roteador de borda. Podem ocorrer ainda casos no qual o roteador de borda solicita novas informações de policiamento ao servidor, em virtude de alterações na rede, tais como: aumento de tráfego ou queda de uma interface.

O protocolo COPS define uma arquitetura cliente/servidor composta por duas entidades funcionais, responsáveis pela comunicação entre os planos de dados e de controle, conforme ilustra a Figura 3.12:

- PEP (*Policy Enforcement Point*): Ponto de aplicação do policiamento. É localizado no plano de dados, e é o local onde o policiamento é efetivamente aplicado;
- PDP (*Policy Decision Point*): Ponto central de decisão do policiamento, também chamado de servidor de policiamento. É localizado no plano de controle, e é o local onde as políticas e regras do domínio são armazenadas, conhecido por PIB (*Policy Information Base*).

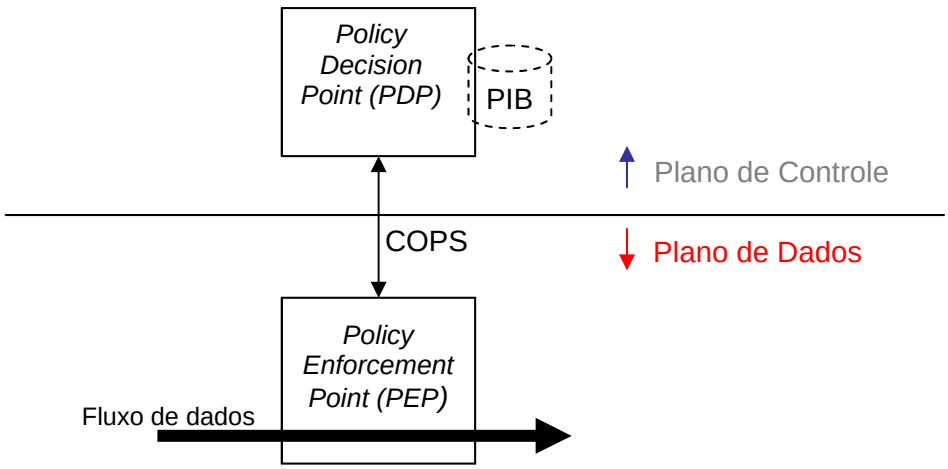


Figura 3.12: Arquitetura padrão COPS

A comunicação entre um PEP e um PDP é realizada através de uma conexão TCP e ocorre principalmente na forma de troca de mensagens de pedidos e respostas. Quando uma

informação de policiamento é instalada com sucesso no PEP, este precisa enviar ainda uma mensagem (*report*) para o PDP confirmando a instalação. A Figura 3.13 ilustra uma seqüência de troca de mensagens entre um PEP e um PDP.

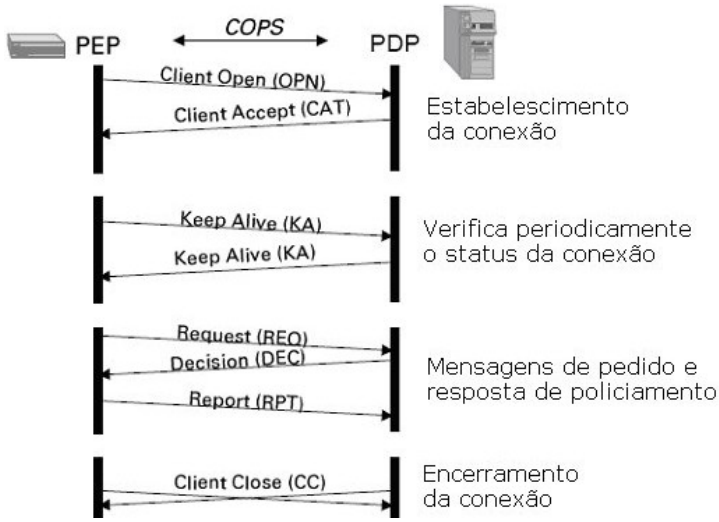


Figura 3.13: Mensagens COPS trocadas entre PEP e PDP

O protocolo COPS utiliza-se de objetos na comunicação entre o PEP e PDP, os quais contém dados para identificar os tipos de pedidos, os contextos dos pedidos, relato de erros, entre outras. A Figura 3.14 apresenta um exemplo de objeto com informações de configuração de um roteador de borda *DiffServ*.

Length = 60 or 88 or 116, etc.		S-Num = 5	S-Type = 1	
Direction	Protocol ID	Flags, defined below	Session Class	
Source IP Address (32-bits)				
Destination IP Address (32-bits)				
Source Port (16-bits)		Destination Port (16-bits)		
DS Field	Reserved	Reserved	Reserved	
Timer T1 value				
Timer T2 value				
Token Bucket Rate [r] (32-bit IEEE floating point number)				Flow spec alt #1
Token Bucket Size [b] (32-bit IEEE floating point number)				
Peak Data Rate (p) (32-bit IEEE floating point number)				
Minimum Policed Unit [m] (32-bit integer)				
Maximum Packet Size [M] (32-bit integer)				
Rate [R] (32-bit IEEE floating point number)				
Slack Term [S] (32-bit integer)				

Figura 3.14: Exemplo de um objeto COPS

A Figura 3.15 ilustra de forma resumida o fluxograma dos principais processos envolvidos no plano de controle.

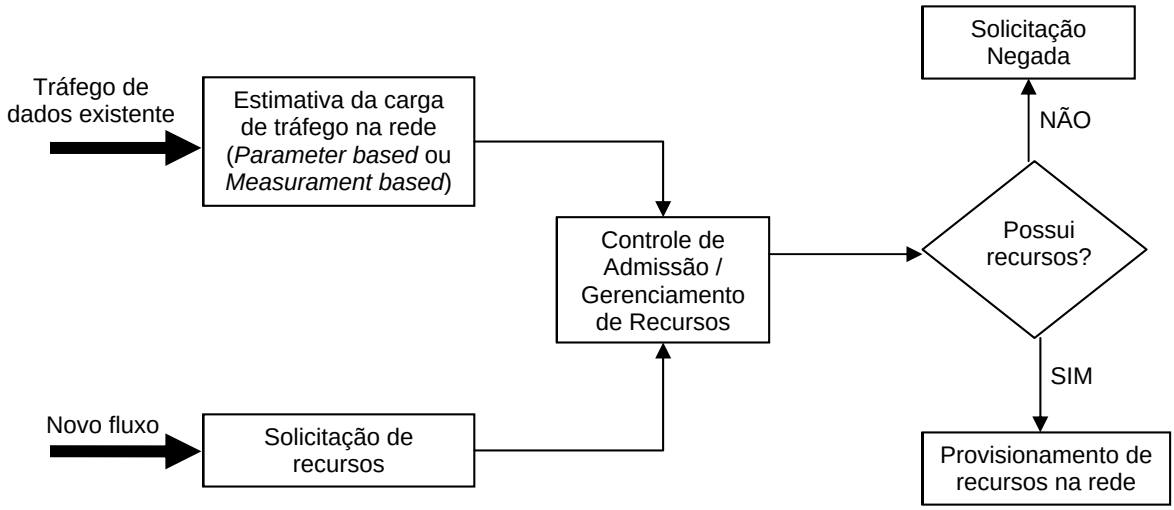


Figura 3.15: Fluxograma do plano de controle

Ressalta-se ainda a interdependência entre as funções de controle de admissão e gerenciamento de recursos. Um controle de admissão severo conduz a um número maior de clientes que têm suas solicitações de recursos negadas, e portanto os recursos da rede são utilizados de forma ineficaz. Por um outro lado, um controle de admissão brando conduz a um número maior de clientes que têm suas solicitações de recursos aceitas, contudo, pode exceder a capacidade de recursos da rede, e consequentemente, produzir congestionamento, afetando a qualidade de serviços da rede.

3.4 Conclusões

Neste capítulo apresentaram-se as características, funções e blocos funcionais dos planos de dados e de controle da arquitetura *DiffServ*. O plano de dados é composto pelos seguintes blocos funcionais: classificadores, marcadores, medidores e condicionadores de

tráfego, gerenciadores de filas e escalonadores. Já o plano de controle é composto pelos seguintes blocos funcionais: gerenciamento de recursos, controle de admissão e provisão de recursos.

Foi apresentado ainda o elemento *Bandwidth Broker* como sendo o agente responsável pelas funções do plano de controle, enquanto que as funções do plano de dados concentram-se principalmente nos roteadores de borda.

O *Bandwidth Broker* tem como tarefa prover uma integração entre os planos de dados e de controle, através de protocolos de sinalização e aprovisionamento, interagindo as políticas e regras do domínio com as necessidades de qualidade de serviços dos clientes. Dessa integração, esperam-se mecanismos que possibilitem a alocação dinâmica de recursos, controle de congestionamento do tipo preventivo e um ponto central de aprovisionamento dos recursos da rede.

Capítulo 4

Bandwidth Broker (BB)

4.1 Introdução

Neste capítulo apresenta-se em detalhes o elemento *Bandwidth Broker*, conforme segue abaixo:

- Na seção 4.2 descrevem-se os módulos do *Bandwidth Broker*, assim como sua principal arquitetura de referência para gerenciamento de recursos em domínios *DiffServ*;
- Na seção 4.3 descrevem-se os principais trabalhos relacionados com a validação da arquitetura apresentada na seção 4.2;
- Já na seção 4.4 são apresentadas arquiteturas alternativas para gerenciamento de recursos em domínios *DiffServ*.

4.2 Arquitetura e módulos do *Bandwidth Broker*

O modelo proposto em [56] é reconhecido como o principal modelo de gerenciamento de recursos em domínios *DiffServ* com a utilização do *Bandwidth Broker*. De acordo com esse modelo, cada domínio, também chamado de sistema autônomo (AS - *Autonomous System*), deve centralizar as funções do plano de controle no BB, seja em um único BB ou em múltiplos BBs. Do ponto de vista da arquitetura *DiffServ*, um BB expande o PHB (*Per Hop Behavior*) de fluxos agregados em PDB (*Per Domain Behavior*).

Um *Bandwidth Broker* é responsável não apenas pelo gerenciamento de recursos intradomínio, mas também pelo interdomínio, através da comunicação ponto-a-ponto entre BBs de domínios adjacentes. Existe ainda o caso de ter-se um BB em um domínio tipo trânsito, no qual os recursos são reservados entre os pontos de ingresso e egresso de outros domínios. Dessa forma, a qualidade de serviço fim-a-fim pode ser alcançada pela concatenação da reserva de recursos intra e inter domínios, conforme mostra a Figura 4.1.

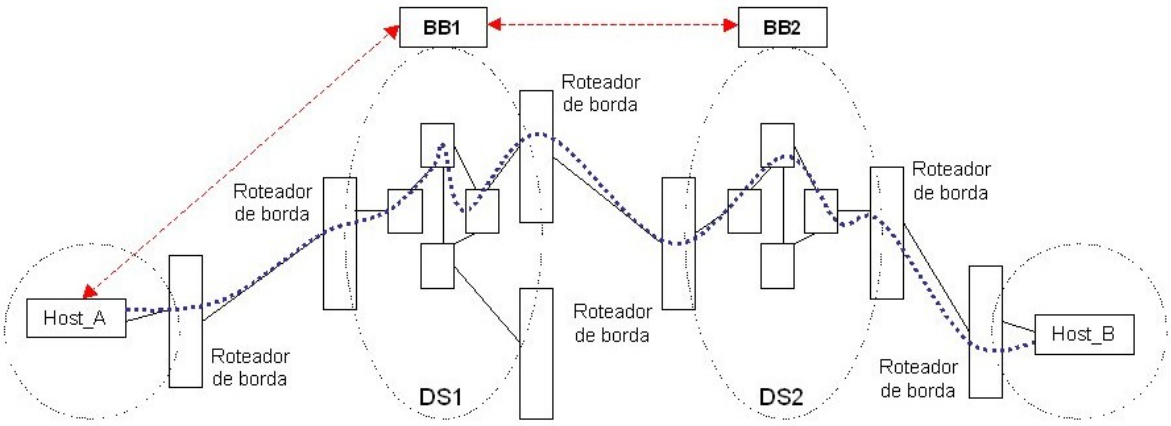


Figura 4.1: Gerenciamento de recursos fim-a-fim

Um *Bandwidth Broker* é composto de vários módulos, como ilustrado na Figura 4.2. Segue abaixo a descrição desses módulos:

- Módulo interdomínio: é utilizado para comunicação com BBs adjacentes. Neste caso, os protocolos de sinalização propostos são: RSVP [17] e SIBB (*Simple Interdomain Bandwidth Broker signaling*) [57];
- Módulo intradomínio: é responsável pela comunicação com os roteadores de borda, enviando informações de aprovisionamento. O protocolo COPS-PR [55] foi escolhido como o protocolo de aprovisionamento, sendo que o BB representa o PDP e os roteadores de borda representam os PEPs;
- Módulo de tabela de roteamento: armazena a topologia e rotas da rede. Este módulo pode ser utilizado para automatizar o processo de engenharia de tráfego;
- Módulo de interface de usuário/aplicação: o escopo desse módulo é permitir que um usuário ou aplicação envie solicitações de recursos diretamente ao BB, através de um protocolo de sinalização;
- Módulo de policiamento: armazena as informações de políticas e regras do domínio, tais como os parâmetros de SLA pré-estabelescidos com os clientes, e informações de rede, como mapeamento dos fluxos em valores DSCP;
- Módulo de gerenciamento da rede: é utilizado como interface de gerência e operação por um operador de rede.

Um BB possui ainda uma base de dados para o armazenamento de informações sobre a carga de tráfego na rede (roteadores, enlaces físicos e circuitos virtuais). A atualização desta base de dados é realizada periodicamente por um protocolo de gerência, tal como o

SNMP (*Simple Network Management Protocol*) [58]. Observa-se que o IETF padronizou uma MIB (*Management Information Base*) [11] específica para as redes *DiffServ*.

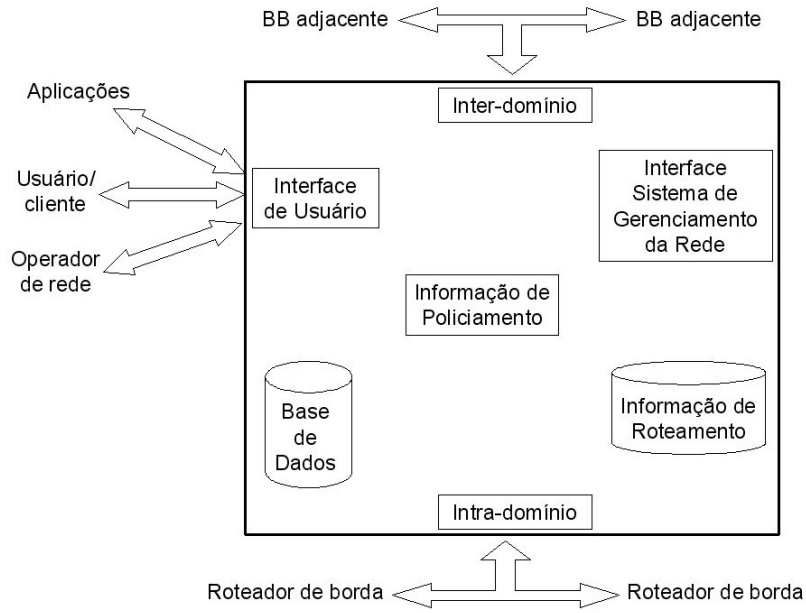


Figura 4.2: Estrutura de um *Bandwidth Broker*

A solicitação de recursos ao BB pelos clientes do domínio ocorre através de mensagens chamadas de RAR (*Resource Allocation Request*). Quando o *Bandwidth Broker* recebe este tipo de mensagem, verifica se a solicitação encontra-se de acordo com os parâmetros de SLA previamente negociados, e ainda se o domínio possui recursos suficientes para proceder com a alocação de recursos. Então, o *Bandwidth Broker* notifica o cliente sobre sua decisão com a mensagem conhecida por RAA (*Request Allocation Answer*).

Caso a solicitação seja aceita, o *Bandwidth Broker* utiliza-se de seu módulo intradomínio para configurar os roteadores de borda pertencentes ao domínio por ele gerenciado.

As mensagens RAR e RAA fazem parte de um protocolo de sinalização. Para as aplicações de telefonia IP tem destacado-se o SIP [47] como protocolo de sinalização.

Sendo que a RFC 3312 [59] apresenta uma proposta de integração entre este protocolo e a função de gerenciamento de recursos em redes IP.

Em contraste com a arquitetura *DiffServ* inicial, onde o SLA entre o cliente e o domínio ocorre de forma pré-estabelecida e definitiva, e consequentemente a alocação de recursos, na arquitetura "*DiffServ* + BB" o cliente pode solicitar recursos diretamente ao BB por meio de seu módulo de interface usuário/aplicação. Neste caso, a alocação de recursos ocorre de forma dinâmica, do ponto de vista de que os recursos são alocados no momento em que ocorre a solicitação (alocação sob demanda).

Contudo, no resultado final ainda existe uma parcela de alocação estática, pois uma vez que os recursos são alocados ao cliente, toda a largura de banda requisitada estará indisponível, do ponto de vista do BB, independente da atual necessidade. Desta forma, os recursos da rede são desperdiçados quando a carga de tráfego não utiliza a largura de banda alocada.

4.3 Principais Projetos

Vários trabalhos têm sido realizados nos últimos anos, utilizando-se de simulações, modelagens, implementação de protocolos e até mesmo no desenvolvimento de *hardwares* específicos, com o objetivo de validar a eficiência, escalabilidade e robustez do modelo proposto em [56].

Os artigos [60], [61] e [62] avaliam, através de simulações, a performance de algumas funcionalidade do *Bandwidth Broker*.

Outros trabalhos também têm sido desenvolvidos na forma de projetos. Normalmente dispõem de uma significativa quantidade de recursos, contam com a participação de universidades e fabricantes de equipamentos, e procuram realizar uma análise detalhada em todos os elementos e funções do modelo proposto.

Dentre os principais projetos pode-se citar: “Internet2 QBone Bandwidth Broker” [63], e “TF-NGN” (*Task Force- Next Generation Networking*) [64] conduzido pela associação TERENA (*Trans-European Research and Education Networking Association*).

Ambos os projetos têm como objetivo principal avaliar a escalabilidade e a garantia de qualidade de serviço fim-a-fim, passando por vários domínios *DiffServ*. A principal diferença entre esses projetos é que enquanto o primeiro é conduzido nos EUA, o segundo é conduzido por membros da comunidade européia.

Destaca-se ainda o projeto “Policy Based Networks & Bandwidth Broker” [65] por apresentar um detalhamento quanto ao gerenciamento de recursos intra-domínio.

No Brasil a UNICAMP e a UFRJ conduziram em conjunto um projeto chamado QUARESMA (**Qua**lidade de Serviço em **Re**des, **S**egurança, **M**obilidade e **A**plicações) [66] que teve como objetivo integrar e interagir projetos relacionados às áreas de rede, *middleware* e aplicações. Dentro da área de *middleware* foi desenvolvido um *Bandwidth Broker* em um domínio *DiffServ*/MPLS utilizando-se CORBA.

4.4 Outros Modelos

Ainda com relação a gerenciamento de recursos em domínios *DiffServ* outros modelos têm sido propostos, dentre estes destacam-se os modelos: SCORE (*Stateless Core*) [67] e

MBAC (*Measurament Based Admission Control*) [68].

Apesar do modelo SCORE também reconhecer o *Bandwidth Broker* como o elemento responsável pelas funções do plano de controle, difere significativamente do anterior, pois sugere adequações radicais no plano de dados, baseadas no conceito chamado de DPS (*Dynamic Packet State*). O artigo [69] apresenta um algoritmo para controle de admissão baseado no modelo SCORE.

O modelo SCORE tem como objetivo garantir uma qualidade de serviço no nível de fluxo individual, similar à arquitetura *IntServ*. Porém, sem exigir que os roteadores mantenham qualquer informação de estado. Entretanto, informações de estado são carregadas nos próprios pacotes IP através do acréscimo de um cabeçalho específico.

Este cabeçalho é inserido e removido pelos roteadores de borda, enquanto que os roteadores de núcleo aplicam o QoS de acordo com as informações de estado, contidas nos cabeçalhos dos pacotes. A Figura 4.3 ilustra o plano de dados e de controle deste modelo.

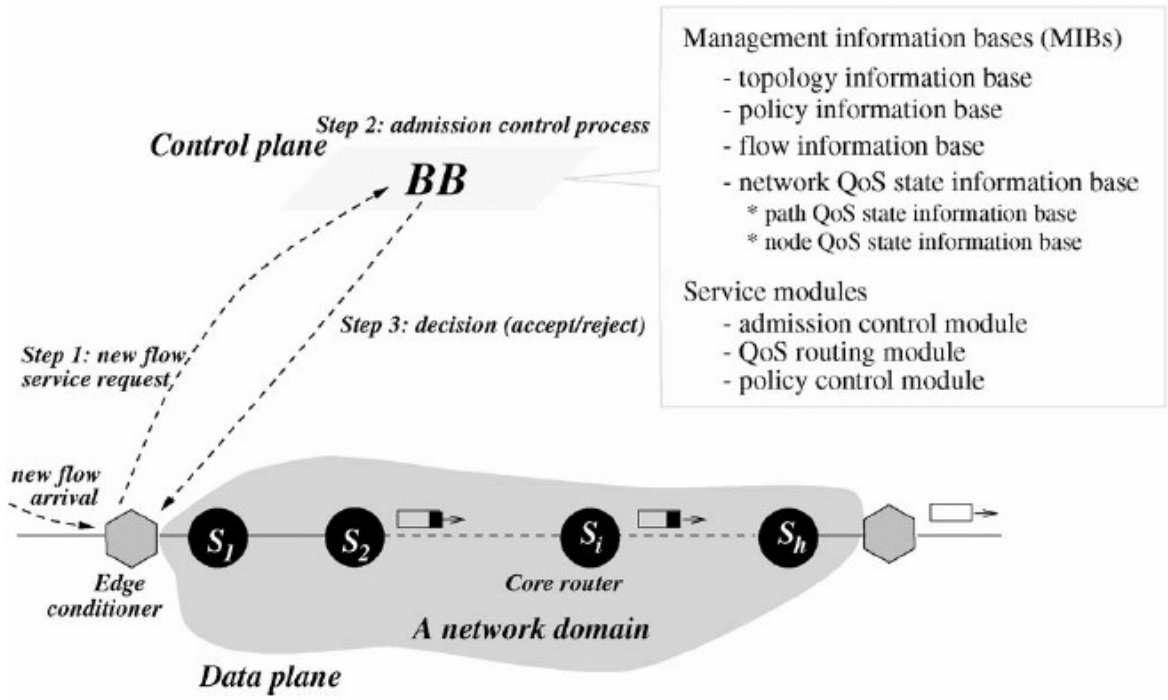


Figura 4.3: Plano de dados e controle utilizando DPS [69]

Já no modelo MBAC não existe o elemento *Bandwidth Broker* e as funções de controle encontram-se no mesmo plano das funções de dados. Contudo, não é necessário qualquer sinalização nó-a-nó, como na arquitetura *IntServ*/RSVP, assim como não é necessária a implementação de funcionalidades adicionais nos roteadores, como no modelo SCORE.

No modelo MBAC um terminal (pode ser um *host* ou um roteador de borda) possui um mecanismo de medição de QoS fim-a-fim, que é capaz de estimar o estado de congestionamento interno da rede para então executar o controle de admissão. Existem dois métodos no MBAC: o passivo e o ativo, sendo que em ambos a qualidade de serviço da rede é medida em intervalos de tempo, utilizando-se de indicadores como taxa de perda de pacotes, taxa média de atraso fim-a-fim e taxa média de *jitter*.

No método passivo os indicadores de qualidade de serviço são extraídos diretamente dos próprios fluxos individuais, enquanto que no método ativo, utiliza-se de um fluxo de teste (também chamado de fluxo de prova), representando um processo de sinalização. A Figura 4.4 ilustra o processo de controle de admissão do método ativo.

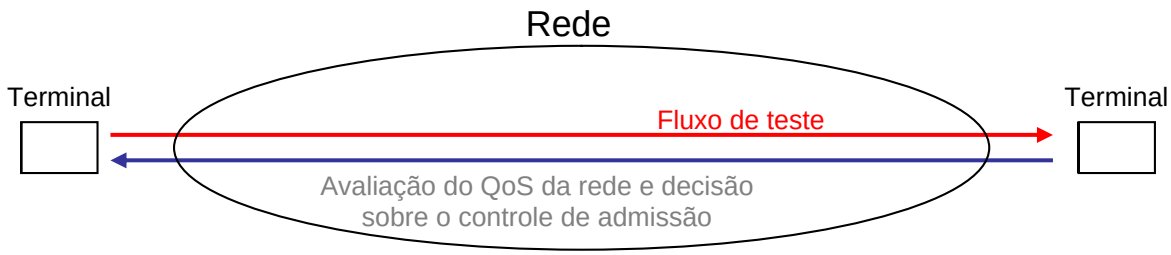


Figura 4.4: Modelo MBAC utilizando-se do método ativo

Uma vantagem do método passivo é que o mesmo não gera tráfego adicional, o qual consome recursos da rede, como no método ativo. Contudo, o controle de admissão do método ativo é mais suave e eficiente do que no método passivo. Os artigos [70] e [71] apresentam esquemas de controle de admissão baseados no modelo MBAC.

4.5 Conclusões

Neste capítulo foram apresentados os módulos que compõem a estrutura de um *Bandwidth Broker*, a principal arquitetura de referência para gerenciamento de recursos em domínios *DiffServ*, assim como os principais trabalhos relacionados com essa arquitetura. Descrevem-se ainda arquiteturas alternativas para o gerenciamento de recursos em domínios *DiffServ*.

O *Bandwidth Broker* é composto dos seguintes módulos: interdomínio, intradomínio, tabela de roteamento, interface de usuário/aplicação, policiamento e gerenciamento da rede.

A principal arquitetura para gerenciamento de recursos é baseada na concatenação de reserva de recursos intra e inter domínios para obter uma qualidade de serviço fim-a-fim. Neste sentido, foram padronizados protocolos específicos para prover os recursos intradomínio e interdomínio, respectivamente.

Foram apresentadas também outras arquiteturas para gerenciamento de recursos. Contudo, questões como as necessidades de realizar alterações significativas no modelo atual da Internet têm feito com que estas arquiteturas tenham um baixo índice de aceitação.

Capítulo 5

Proposta de um Mecanismo para Alocação Otimizada de Recursos em Domínios *DiffServ*

5.1 Introdução

Apesar da arquitetura *DiffServ* ter se mostrado como uma solução escalável para prover QoS nos grandes *backbones* IP, novos blocos funcionais têm sido propostos (mais especificamente relacionados ao plano de controle), com o objetivo de assegurar uma qualidade de serviço mais rígida.

Propõe-se neste trabalho a inclusão de um novo módulo na estrutura do *Bandwidth Broker*, em acordo com a arquitetura descrita no capítulo anterior. Esse módulo realiza ações voltadas ao gerenciamento dinâmico de recursos através de um mecanismo de alocação de recursos da rede.

O módulo tem como objetivo aumentar a eficiência na utilização dos recursos da rede, contudo, respeitando os parâmetros de QoS: perda de pacotes e atraso fim-a-fim.

Na seção 5.2 apresentam-se as limitações do modelo atual. Na seção 5.3 apresenta-se o mecanismo de alocação proposto, e por fim na seção 5.4 apresentam-se os trabalhos relacionados.

5.2 Descrição do Problema

Na arquitetura *DiffServ* os primeiros blocos funcionais definiam que os SLAs fossem pré-estabelecidos entre os clientes e os seus respectivos provedores de serviço, conduzindo a uma alocação estática de recursos por meio do aprovisionamento do serviço pelo provedor. O SLA é baseado em certos parâmetros, tais como: PIR, CIR, CBS, atraso fim-a-fim e *jitter*. Embora este modelo seja escalável e fácil de implementar, os recursos da rede são utilizados de maneira ineficaz, uma vez que os mesmos são alocados de forma estática.

Observa-se ainda que a rede não é capaz de prover um serviço com QoS confiável, pois o policiamento executado pelo roteador de borda junto ao tráfego do cliente não leva em consideração o estado de QoS da rede, sendo averiguado, nas extremidades da rede, somente se o tráfego do cliente encontra-se de acordo com o SLA acordado. Logo, não existe um mecanismo capaz de realizar ações preventivas de congestionamento na rede. Esta questão foi exemplificada na seção 3.3.1.1 (Gerenciamento Estático).

Com o objetivo de alocar os recursos da rede de uma forma mais eficaz, introduziu-se o conceito do *Bandwidth Broker* [9], sendo que recentemente foi elaborado um novo *Internet Draft* [10] alinhado com o trabalho anterior, cujo objetivo é descrever em detalhes os

elementos do plano de controle e de sua arquitetura. A RFC 2905 [72] aborda aspectos quanto a sua implementação.

Já com a utilização do *Bandwidth Broker* a alocação de recursos ocorre de forma dinâmica, do ponto de vista que os recursos são alocados no momento da solicitação (sob demanda).

Contudo, apesar da alocação de recursos ser considerada dinâmica, uma vez que é executada sob demanda, existe ainda uma parcela de alocação estática. Ocorre que, depois de que foram concedidos os recursos a uma determinada solicitação, tais recursos permanecem totalmente alocados, sem levar em consideração o comportamento do tráfego que na maioria dos casos é dinâmico por natureza, como por exemplo, tráfego em rajadas e tráfego tipo VBR (*Variable Bit Rate*). Assim sendo, os recursos da rede são desperdiçados nos casos em que a carga de tráfego não utiliza totalmente a largura de banda alocada. A Figura 5.1 ilustra um exemplo onde é alocada uma largura de banda de 2 Mbps a um determinado fluxo e a largura de banda efetivamente utilizada.

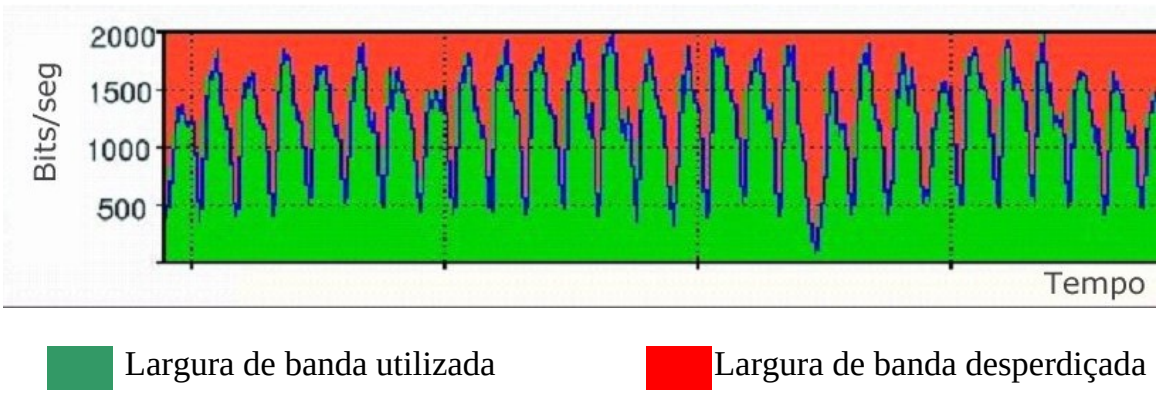


Figura 5.1: Largura de banda alocada, utilizada e desperdiçada

5.3 Solução Proposta

Tendo em vista o problema da sub-utilização de recursos apresentado na seção anterior, neste trabalho propõe-se um módulo de gerenciamento de recursos que opere junto com o módulo de policiamento de um BB. A finalidade desse módulo é realocar periodicamente os recursos da rede, considerando a carga de tráfego instantânea gerada pelas aplicações. Para isso, considera-se que a largura de banda alocada seja determinada através do valor de CIR e, quando a carga de tráfego atual encontra-se abaixo desse valor, é definido um novo e menor valor de CIR. Assim, parte da largura de banda previamente alocada retorna para um *pool* de largura de banda disponível e, desta forma, o BB pode conceder este recurso a novas aplicações, ou mesmo, retorná-lo às aplicações já existentes, caso seja necessário.

A informação sobre os requisitos atuais da carga de tráfego instantânea gerada pelas aplicações advém de medições realizadas pelos roteadores de borda, cujos resultados são reportados periodicamente ao BB através de seu módulo intradomínio. Desta forma, o módulo de gerenciamento de recursos interage com o módulo de policiamento com a finalidade de determinar o novo valor de CIR. Finalmente, o BB comunica-se com os roteadores de borda, configurando os novos valores de CIR. A Figura 5.2 mostra os módulos envolvidos nos elementos *Bandwidth Broker* e roteador de borda, e suas interações.

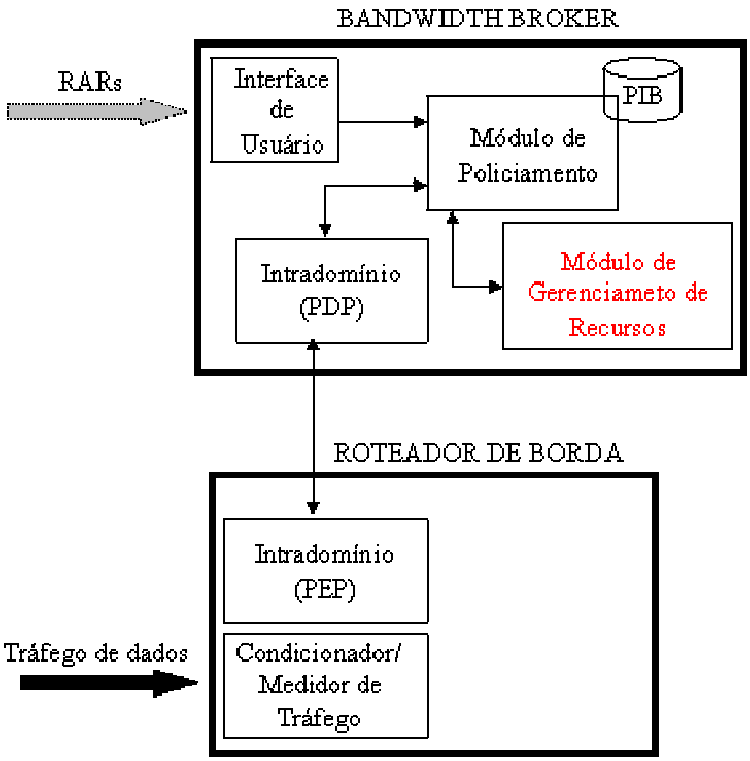


Figura 5.2: Interação entre módulos do *Bandwidth Broker* e do roteador de borda

Os seguintes procedimentos são realizados para determinar o novo valor de CIR:

- A carga de tráfego instantânea de uma aplicação é medida periodicamente;
- Caso esta carga de tráfego instantânea esteja abaixo do valor de CIR, a diferença entre estes valores retorna ao BB como largura de banda disponível e define-se um novo valor de CIR igual à carga de tráfego instantânea medida;
- Caso a carga de tráfego instantânea medida seja maior que o valor de CIR, o BB verifica se o domínio dispõe de recursos e, em caso positivo, o BB utiliza-se do *pool* de largura de banda disponível para aumentar o valor de CIR, definindo-a igual à carga de tráfego instantânea medida. Observa-se que este caso acontece normalmente quando uma parte da largura de banda foi realocada e, agora, o tráfego gerado pela aplicação requer um aumento da largura de banda. Observa-se também

que o valor reajustado de CIR nunca pode ser maior do que o valor inicial do CIR, especificado no SLA. A Figura 5.3 apresenta uma visão simplificada do algoritmo.

```
DO WHILE (t <=TEMPO DE SIMULAÇÃO)
{
        IF (FlowA.Meter(t) <= CIR)
        THEN
                {
                BB.AddBW(CIR - FlowA.Meter(t))
                CIR = FlowA.Meter(t)
                }
        ELSE IF (FlowA.Meter(t) – CIR <= BB.AvailableBW( ))
                THEN
                        {
                        BB.ReleaseBW(FlowA.Meter(t) – CIR)
                        CIR = FlowA.Meter(t)
                        }
t = n * PERÍODO DE MONITORAMENTO    (n = 1, 2, 3, …)
}
```

Figura 5.3: Algoritmo simplificado do módulo de gerenciamento

Um aspecto importante no mecanismo proposto é referente ao período de monitoramento dos recursos da rede, caso o período seja muito pequeno, a alocação de recursos acompanhará de forma fiel a necessidade de largura de banda. Contudo, este processo produz uma grande quantidade de mensagens de sinalização trafegando na rede. Por outro lado, caso o período seja muito grande, o mecanismo proposto torna-se ineficaz.

O *Bandwidth Broker* permanece como o gerenciador de recursos de rede no domínio *DiffServ*. Contudo, o processo de alocação de recursos é melhorado através da capacidade do BB em realizar ações periódicas, definindo novos valores de CIR, e então modelando a largura de banda alocada de acordo com a carga de tráfego gerada pelas aplicações.

O mecanismo proposto neste trabalho evita desperdício de largura de banda e possibilita aumentar o número de usuários que tenham seus acessos concedidos junto ao domínio. Desta forma, torna-se possível um cenário de sobre alocação de recursos, onde a quantidade instantânea de recursos alocados pode exceder a capacidade máxima da rede.

Observa-se ainda que o mecanismo proposto nesta dissertação poderá aumentar a probabilidade de descarte de pacotes, devido à característica periódica na designação de valores de CIR. Pode ocorrer que no instante subsequente à designação de um determinado valor de CIR, tenha-se um volume de tráfego acima desse valor, caracterizando parte do tráfego como *out-of-profile* e, consequentemente alguns pacotes serão marcados com um DSCP de QoS inferior.

De forma a evitar cenários com uma alta probabilidade de descarte de pacotes, torna-se importante observar alguns pontos na configuração dos parâmetros do mecanismo de alocação. Para tanto se deve: definir um fator de alocação moderado, fixar um limite no número de alocações permitidas por classe de serviço (CoS), e reservar uma certa largura de banda ao serviço *Best-Effort* (BE), que em caso de necessidade poderia ser alocada para as outras classes.

5.4 Trabalhos Relacionados

Os trabalhos realizados em [73], [74] e [75] também propõem mecanismos relacionados ao gerenciamento de recursos em domínios *DiffServ* com o objetivo de aumentar a eficiência na utilização dos recursos.

Em [73] e [74] propõe-se um algoritmo adaptativo para gerenciar RARs que solicitam reservas futuras, desta forma o BB conhece previamente a demanda de recursos necessários para os períodos futuros e, pode então decidir quais RARs serão aceitos, de modo que a utilização da rede seja a melhor possível. Observa-se que neste esquema o gerenciamento de recursos foca em um momento antes da decisão do BB quanto à concessão ou não dos recursos solicitados, algo entre a solicitação de alocação de recursos (RAR) e a resposta de alocação de recursos (RAA). Desta forma, este esquema e o mecanismo de alocação proposto neste trabalho podem ser vistos como complementares, uma vez que o mecanismo proposto foca no gerenciamento de recursos após a tomada de decisão pelo BB.

Já os autores em [75] propõem um esquema de gerenciamento ativo de recursos. O módulo de policiamento do BB associa um conjunto de valores de DSCP para cada fluxo, e baseado no monitoramento do tráfego gerado pelo fluxo é designado um destes valores de DSCP. De forma similar ao mecanismo proposto neste trabalho, este esquema também se detêm no gerenciamento de recursos após a tomada de decisão pelo BB. Contudo, a designação de diferentes valores de DSCP reflete diretamente no nível de QoS associado aos fluxos, afetando parâmetros como: vazão, atraso fim-a-fim e *jitter*. Já o mecanismo de alocação proposto neste trabalho é capaz de respeitar o nível de QoS dos fluxos, e manter uma exata informação sobre a utilização dos recursos da rede.

5.5 Conclusões

Neste capítulo apresentou-se o problema da sub-utilização de recursos e uma proposta de solução baseada em um mecanismo para alocação otimizada de recursos em domínios

DiffServ. Foram descritas sua forma de funcionamento, características e contribuições à arquitetura existente. Foi discutido ainda semelhanças e diferenças com alguns trabalhos relacionados.

A premissa do mecanismo proposto neste trabalho consiste no fato de que o tráfego das aplicações é geralmente dinâmico por natureza e, por causa disso, uma vez concedidos os recursos, os mesmos devem ser periodicamente realocados de acordo com a carga instantânea de tráfego. Neste sentido foi proposta nesta dissertação a inclusão de um módulo de gerenciamento de recursos no elemento *Bandwidth Broker* capaz de interagir com os outros módulos, com a finalidade de moldar a alocação de recursos de cada aplicação.

A principal contribuição do mecanismo proposto neste trabalho é quanto ao aumento da eficiência na utilização dos recursos da rede, contudo sem prejudicar a qualidade de serviço existente.

Quanto aos trabalhos relacionados ao gerenciamento de recursos em domínios *DiffServ*, observou-se que existem basicamente duas linhas distintas de abordagem, conforme resumidas abaixo:

- Uma delas procura otimizar os recursos previamente à tomada de decisão quanto à alocação de recursos pelo BB (otimização pré-controle de admissão);
- Já a outra abordagem, procura otimizar os recursos após os recursos terem sido concedidos pelo BB (otimização pós-controle de admissão).

Capítulo 6

Avaliação do Mecanismo Proposto para Alocação de Recursos

6.1 Introdução

O módulo de gerenciamento de recursos apresentado no capítulo anterior deste trabalho tem como objetivo aumentar a eficiência de utilização dos recursos da rede através de um mecanismo avançado de alocação de recursos.

Neste capítulo avalia-se a eficiência do mecanismo de alocação proposto submetido a 2 cenários distintos:

- No 1º cenário, chamado de "Alocação Total", a soma dos RARs corresponde à capacidade máxima da rede. Compara-se a performance de QoS da rede entre um domínio *DiffServ* com um BB tradicional, chamado de "*DiffServ* + BB", e um

domínio *DiffServ* com um BB utilizando o mecanismo de alocação proposto, chamado de "*DiffServ* + BB_*Enhanced*".

- No 2º cenário, chamado de "Sobre-Alocação", configura-se um fator de sobre-alocação de 25% utilizando-se do mecanismo proposto, de forma a permitir que novos usuários tenham acesso ao domínio. Comparam-se, então, os resultados obtidos nos 2 cenários, em termos de utilização de largura de banda, perda de pacotes e atraso fim-a-fim.

6.2 Modelagem e Simulação dos Cenários

6.2.1 Rede de Referência

O desenvolvimento do mecanismo proposto neste trabalho teve como referência a arquitetura NGN, ilustrada de forma simplificada na Figura 6.1.

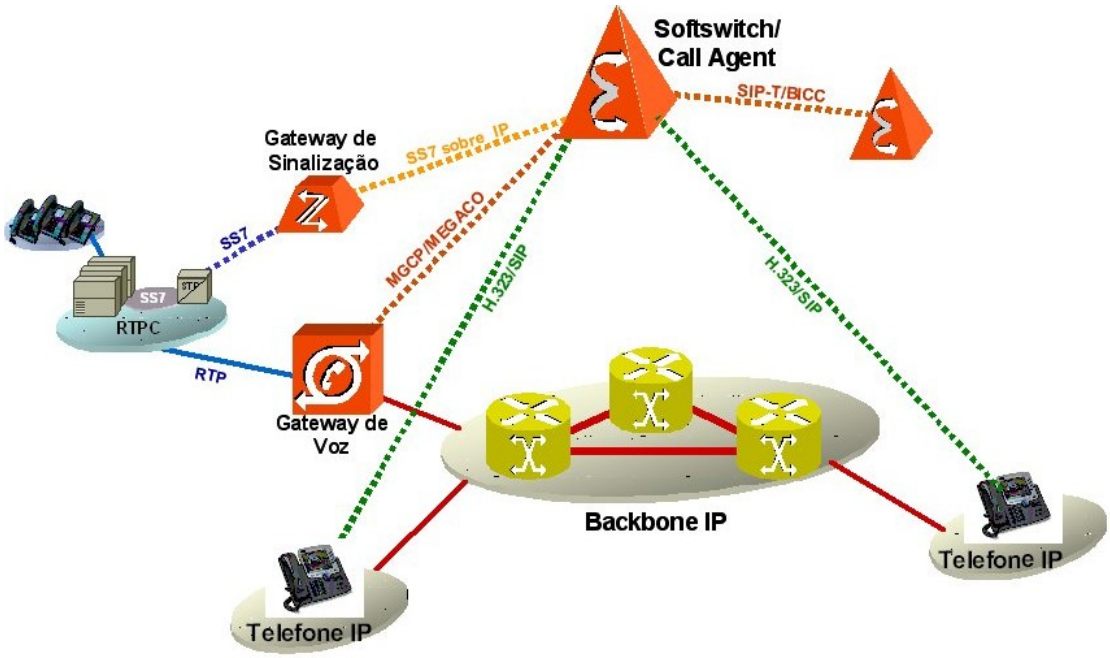


Figura 6.1: Arquitetura NGN

As principais aplicações suportadas por uma rede NGN são telefonia IP e vídeo-telefonia IP, e suas características são listadas abaixo:

- Comunicação em tempo real;
- Interatividade;
- UDP (*User Datagram Protocol*) como protocolo de transporte;
- Altamente sensível a atrasos fim-a-fim;
- Suporta pequenas perdas de pacotes;
- Largura de banda pequena para telefonia IP e média para vídeo telefonia;
- Fontes de tráfego tipo VBR.

A arquitetura de uma rede NGN é composta basicamente dos seguintes elementos:

- Uma rede *backbone IP* composta por roteadores de borda e de núcleo;
- Terminais IP (telefones IP, computadores, *handsets*, etc) ou mesmo telefones convencionais, responsáveis pelo acesso junto aos clientes;
- *Gateways* de mídia e sinalização responsáveis pela conexão do *backbone IP* à rede de telefonia convencional, baseada em comutação por circuito;
- Um *Softswitch* responsável por diversas funções, tais como: controle de chamada (estabelecimento e encerramento), lógica do serviço, controle de admissão, gerenciamento de recursos, provisão de recursos, autenticação, autorização, bilhetagem, etc.

Observa-se que o *softswitch* é um elemento bastante complexo, uma vez que incorpora uma série de funcionalidades, inclusive funções de um *Bandwidth Broker*.

Neste contexto, o ETSI (*European Telecommunications Standard Institute*) padronizou a arquitetura funcional das redes NGN no modelo conhecido por TISPAN (***T**elecoms & **I**nternet converged **S**ervices & **P**rotocols for **A**dvanced **N**etworks*) [76]. A Figura 6.2

apresenta todos os blocos funcionais do *softswitch* de acordo com o modelo TISPAN. Destacam-se os blocos funcionais CSCF (*Call Session Control Function*) e MRFC (*Multimedia Resource Function Controller*) como sendo os blocos responsáveis pelas funções de controle de admissão e gerenciamento de recursos, respectivamente.

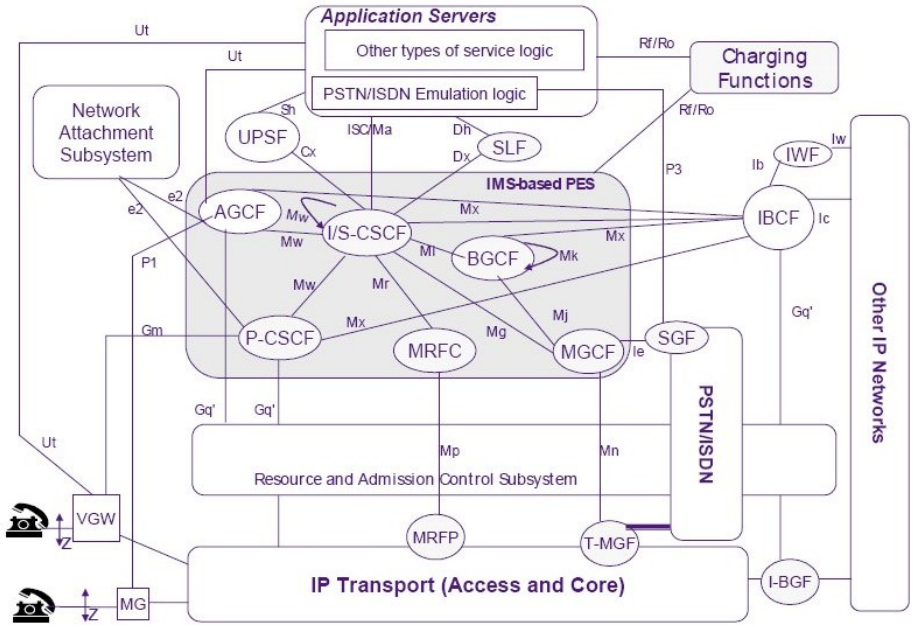


Figura 6.2: Arquitetura Funcional TISPAN [76]

6.2.2 Sobre a Ferramenta de Simulação

As simulações foram realizadas utilizando o *software* NS (*Network Simulator*) [77], que é um *software* gratuito de código aberto, e disponível para vários sistemas operacionais, tais como: FreeBSD, Linux, SunOS, Solaris e Windows. O NS é uma poderosa ferramenta para simulação que é capaz de simular redes fixas, móveis, e de satélites, o usuário pode definir topologias arbitrárias de rede, consistindo de nós, enlaces, filas e aplicações. Devido à característica de código aberto, os pesquisadores podem projetar novos protocolos, e então testar suas funcionalidades e performance.

O *software* NS é escrito em C++ e utiliza OTcl como interface de comando. O NS

suporta protocolos de rede: TCP e UDP, tipos de fontes de tráfego: FTP, Telnet, Web, CBR e VBR, algoritmos de roteamento: *Dijkstra* e *Bellman-Ford*. NS é um simulador orientado a eventos o qual recebe *scripts* TCL como dados de entrada para execução, e apresenta os dados de saída em diversas formas, tais como: gráficos, textos alfanuméricos e arquivos de dados brutos para pós-processamento.

Neste trabalho foi utilizado o NS versão 2.1b6a, o módulo embutido NAM (*Network Amination*), responsável por realizar animações gráficas de toda a rede (pacotes, filas, nós, enlaces, topologia, etc.), e a ferramenta adicional TRACEGRAPH [78], responsável pelo pós-processamento de arquivos gerados pelo NS, capaz de produzir gráficos e medições estatísticas de vários parâmetros de QoS, tais como: *jitter*, atraso fim a fim, largura de banda, RTT (*Round Trip Times*), pacotes gerados, pacotes descartados, etc.

6.2.3 Configuração Utilizada no NS para este Trabalho

Para a realização dos experimentos no *software* NS foram utilizados módulos adicionais, com o objetivo de implementar as funções do plano de dados e controle em domínios *DiffServ*.

Para as funções do plano de dados foi utilizado o módulo *DiffServ* desenvolvido pela *Nortel Networks* [79]. Já para as funções do plano de controle foi utilizado como referência o módulo *Bandwidth Broker* desenvolvido pela *Rutgers University* [80].

O módulo para o plano de dados dispõe de 4 filas físicas, sendo que a primeira utiliza do escalonador PQ e as demais do escalonador WRR. Para cada fila física existem 3 filas virtuais, representando diferentes níveis de prioridade de descarte. Existem ainda os seguintes condicionadores de tráfego: TSW3CM, TSW2CM e *Token Bucket*.

Já o módulo para o plano de controle dispõe dos elementos básicos de um *Bandwidth Broker*, são eles:

- Interface de usuário: Provê um meio pelo qual os usuários solicitam recursos, seja através de um protocolo ou mesmo através de um operador da rede. Neste módulo o processo de negociação de recursos entre o BB e os usuários é realizado diretamente por *scripts* TCL, uma vez que não faz parte do escopo deste trabalho analisar a performance de um protocolo de sinalização;
- Módulo de policiamento: Responsável pelo armazenamento de informações relacionadas aos SLAs dos usuários (CIR, PIR, CBS, valores de DSCPs, etc.). O módulo de policiamento no BB executa a função de um servidor de policiamento (PDP), enquanto que o módulo de policiamento nos roteadores de borda é conhecido por ponto de aplicação de policiamento (PEP).

A partir desse módulo foram realizadas alterações com a finalidade de suportar a inclusão do módulo de gerenciamento de recursos dentro da estrutura do *Bandwidth Broker*. Desenvolveram-se ainda rotinas para a comunicação entre os módulos do plano de dados e do plano de controle. No anexo deste documento encontram-se os códigos fontes do módulo para o plano de controle com as respectivas adequações.

6.3 Alocação Total – 1º Cenário

Neste cenário considera-se a topologia de simulação ilustrada na Figura 6.3, que consiste dos seguintes elementos: 9 nós de rede (S1 a S6 representam as fontes de tráfego enquanto D1 a D3 representam os hosts de destino), 2 roteadores de borda (*Edge1* e *Edge2*), roteadores de núcleo (*Core*) representando o núcleo da rede e um *Bandwidth*

Broker. Ambos os enlaces *Link1* e *Link2* possuem 4 Mbps de largura de banda, que representa a capacidade máxima da rede.

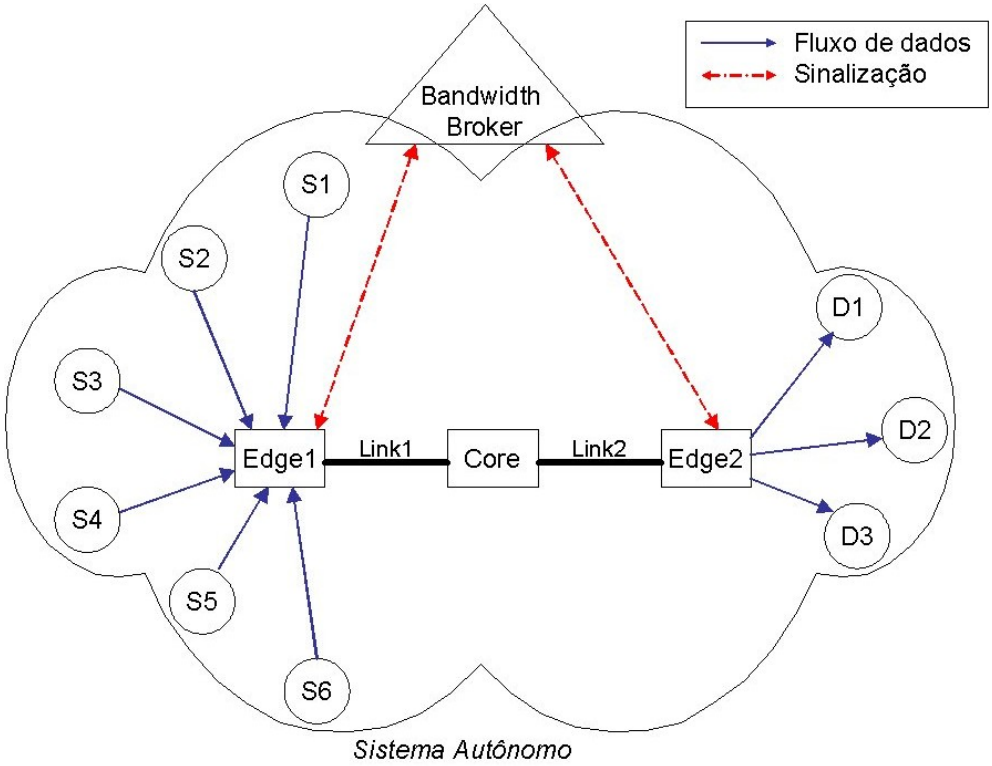


Figura 6.3: Topologia de simulação – 1º Cenário

A Tabela 6.1 lista as fontes de tráfegos e os seus respectivos parâmetros. As fontes de tráfegos S1 a S5 começam a transmitir imediatamente no início da simulação (t = 0 seg.), e representam aplicações de telefonia IP e vídeo-telefonia IP, as quais apresentam requisitos específicos de QoS. Já a fonte de tráfego S6 representa um tráfego de *background* da Internet e não apresenta requisitos de QoS.

Existem várias propostas para a modelagem matemática do tráfego de voz e vídeo na Internet, porém as mais utilizadas consideram esses tráfegos como um tráfego ON-OFF, (também conhecido como EXPONENCIAL). Durante os períodos ON ocorre a efetiva transmissão da informação, enquanto que os períodos OFF são intervalos de inatividade.

Apesar de não ser a modelagem mais usual, as fontes de tráfego S1 e S2 foram caracterizadas como um tráfego CBR com a finalidade de analisar o mecanismo proposto.

Tabela 6.1: Parâmetros das fontes de tráfego

Fonte de Tráfego	Parâmetros		
	Tipo	Pacote	Taxa
S1	CBR/UDP	210 Bytes	450 Kbps
S2	CBR/UDP	80 Bytes	250 Kbps
S3	Exponencial/UDP. Tempo rajada/ocioso: 300/300ms	210 Bytes	775 Kbps
S4	Exponencial/UDP. Tempo rajada/ocioso: 500/300ms	300 Bytes	950 Kbps
S5	Exponencial/UDP. Tempo rajada/ocioso: 300/500ms.	500 Bytes	775 Kbps
S6	FTP/TCP. 7 períodos de rajadas (t = 15; 25; 35; 45; 55; 65; 75) transmitindo 1000 pacotes de 1 Kbytes em cada período.		

A Tabela 6.2 apresenta os parâmetros de SLA previamente acordados entre os usuários e o domínio: CIR, PIR, CoS e valores de DSCP, sendo que o segundo e o terceiro valores são utilizados quando o tráfego excede o valor de CIR (tráfego *out-of-profile*).

Tabela 6.2: Parâmetros de SLA

Fonte	CIR (Kbps)	PIR (Kbps)	CoS	DSCPs
S1	450	450	EF	10 / 0
S2	250	250	EF	11 / 0
S3	775	1000	AF	15 / 16 / 0
S4	950	1500	AF	17 / 18 / 0
S5	775	-	AF	19 / 20
S6	800	-	BE	13 / 14
TOTAL	4000			

A largura de banda exigida no domínio é calculada somando-se os valores de CIR de cada fonte de tráfego. Ressalta-se ainda que a largura de banda previamente reservada[2] para

[2] Os recursos para a classe *best effort* são previamente provisionados e não participam do processo de controle de admissão.

cada fonte é capaz de atender suas respectivas cargas de tráfego, desta forma os recursos solicitados são aceitos, e então alocados pelo BB.

Nos roteadores *DiffServ*, a disciplina PQ é utilizada para a classe EF, e a disciplina WRR é utilizada para as classes AF e *Best Effort*, conforme é apresentado na Figura 6.4.

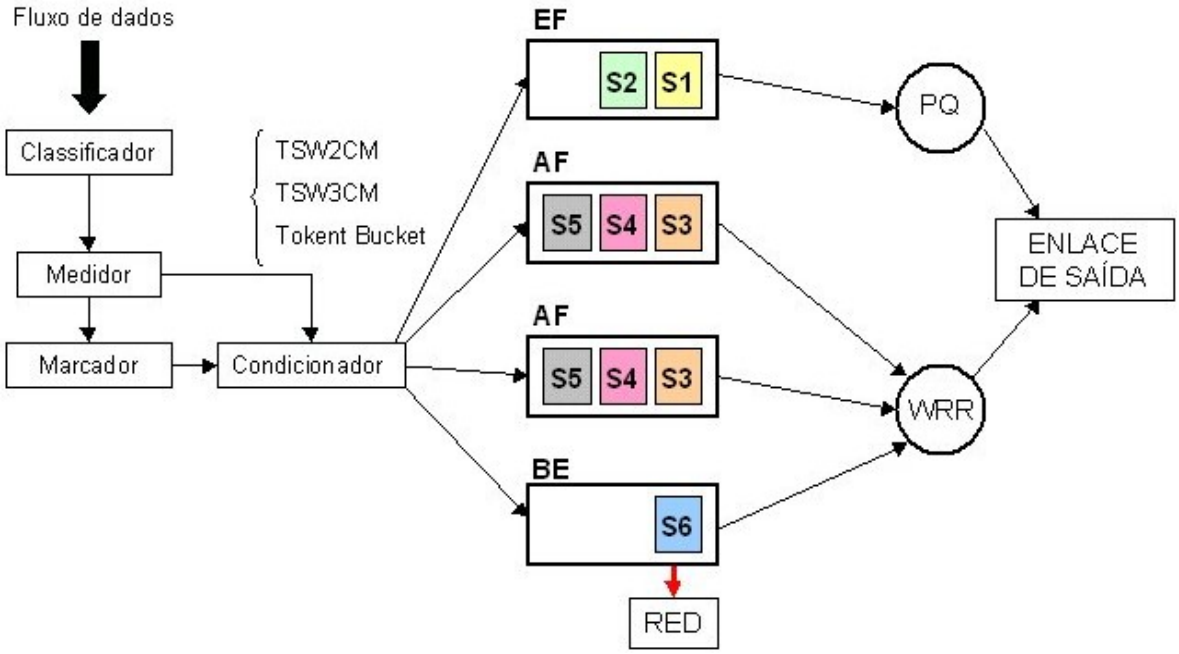


Figura 6.4: Configuração do plano de dados

Nos resultados de simulação foram considerados intervalos de confiança com 95% de confiabilidade, sendo realizadas 5 medições. A seguir são apresentados os resultados para a solução "*DiffServ* + BB".

A Tabela 6.3 apresenta os resultados com relação ao número de pacotes descartados, através das seguintes informações: DSCP, TotPkts (número de pacotes gerados), TxPkts (número de pacotes transmitidos), L_drops (número de pacotes descartados por exceder a capacidade máxima do *buffer* do roteador) e E_drops (quantidade de pacotes descartados preventivamente pelo gerenciador de fila RED).

Tabela 6.3: Pacotes transmitidos e descartados – 1º Cenário "*DiffServ* + BB"

Fonte	DSCP	TotPkts	TxPkts	L-drops	E-drops
S1	10	22767	22767	0	0
S2	11	33202	33202	0	0
S6	13	2757	2756	0	1
	14	4258	4250	0	8
S3	15	19318	19318	0	0
S4	17	19976	19976	0	0
S5	19	6296	6296	0	0
Total		108574	108565	0	9

A Figura 6.5 ilustra o atraso fim-a-fim ao longo da simulação e a Figura 6.6 ilustra a função de distribuição acumulativa do atraso fim-a-fim, ambas para as classes EF, AF e BE. Já a Tabela 6.4 apresenta os atrasos fim-a-fim médio e máximo, e seus respectivos intervalos de confiança com 95% de confiabilidade.

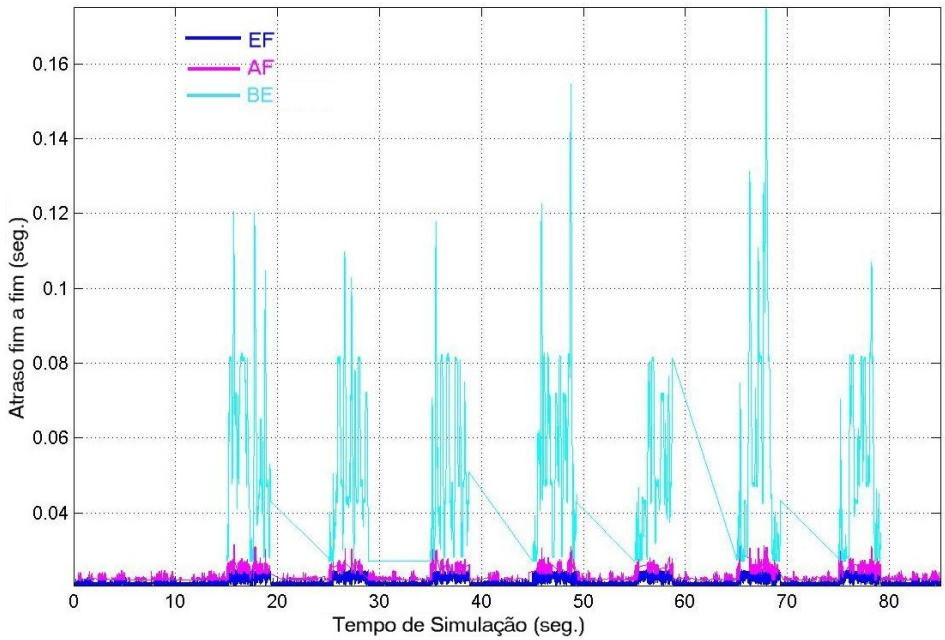


Figura 6.5: Atraso fim-a-fim para as classes EF, AF e BE – 1º cenário "*DiffServ*+BB"

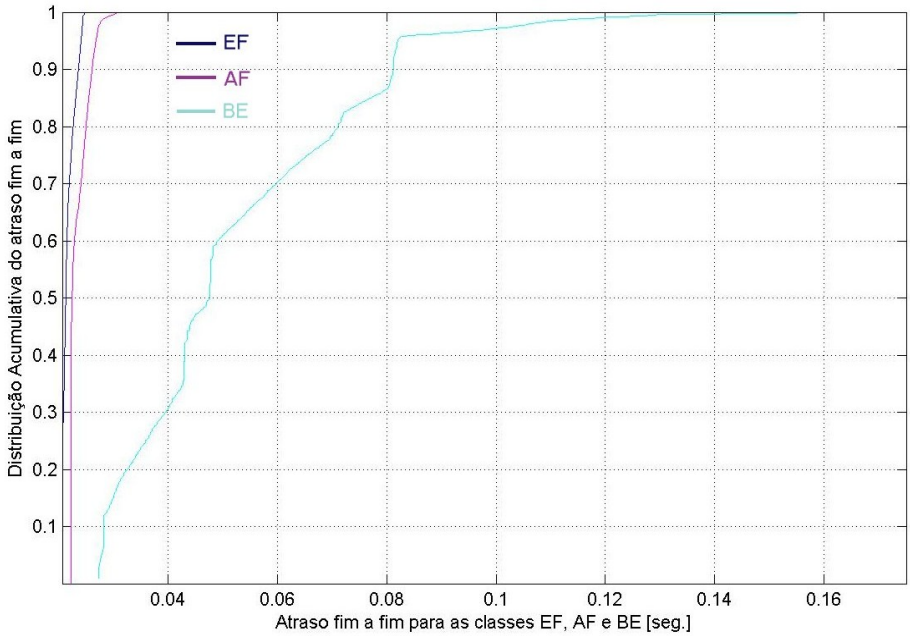


Figura 6.6: Função de distribuição acumulativa do atraso – 1º cenário "*DiffServ*+BB"

Tabela 6.4: Atraso fim-a-fim médio e máximo – 1º Cenário "*DiffServ* + BB"

CoS	Atraso fim-a-fim (mseg)		Intervalo de Confiança (95%)	
	Médio	Máximo	Médio	Máximo
EF	21,6	25,0	20,95 - 22,34	24,37 - 25,87
AF	23,3	31,3	22,56 -24,62	30,03 - 32,41
BE	51,8	175,1	50,62 - 53,12	173,94 - 176,56

Ainda neste mesmo cenário, observam-se a seguir os resultados obtidos utilizando-se do mecanismo de alocação proposto neste trabalho, solução "*DiffServ*+ BB_*Enhanced*". Novamente, nos resultados de simulação foram considerados intervalos de confiança com 95% de confiabilidade, sendo realizadas 5 medições.

Enquanto que na solução "*DiffServ* + BB" os valores de CIR das fontes de tráfego são mantidos constantes durante toda a simulação, observa-se que na solução "*DiffServ*+BB_*Enhanced*" os valores de CIR são modificados ao longo da simulação, acompanhando a necessidade de largura de banda das fontes de tráfego. Configurou-se no mecanismo de alocação um período de monitoramento de 10 segundos. A Figura 6.7 apresenta a alocação da largura de banda por fonte de tráfego ao longo da simulação.

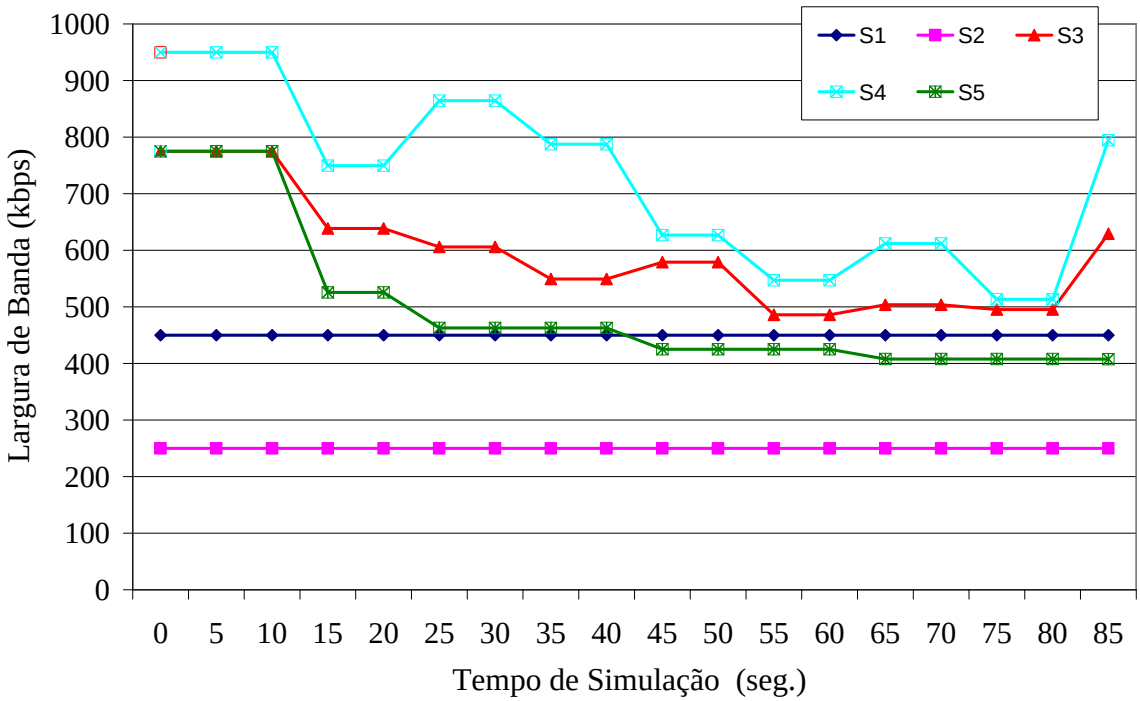


Figura 6.7: Alocação da largura de banda por fonte de tráfego – 1º cenário "*DiffServ*+BB_*Enhanced*"

Como já era esperado, a eficiência do mecanismo de alocação proposto nesta dissertação está relacionada com o perfil de tráfego da fonte. Conforme mostra a Figura 6.7, os valores de CIR das fontes S1 e S2 não foram modificados devido à característica do tráfego destas fontes que transmitem a uma taxa constante durante toda a simulação.

A Tabela 6.5 mostra que em algumas fontes, parte do tráfego comporta-se como *out-of-profile* em determinados momentos, pois tem pacotes marcados com valores de DSCP de menor prioridade. Isto ocorre devido à característica dinâmica do SLA utilizado no mecanismo proposto. Contudo, observa-se que o número de pacotes descartados para cada classe de serviço é praticamente o mesmo da solução "*DiffServ* + BB".

Intencionalmente, não é apresentado o resultado para o atraso fim-a-fim para a solução "*DiffServ*+BB_*Enhanced*". O atraso fim-a-fim neste caso pode ser considerado o mesmo daquele apresentado na solução "*DiffServ* + BB", sob uma mesma carga de tráfego na rede.

Isto se deve ao fato de que o mecanismo proposto neste trabalho reflete diretamente somente no aumento da probabilidade de descarte de pacotes.

Tabela 6.5: Pacotes transmitidos e descartados – 1º Cenário "*DiffServ*+ BB*_Enhanced*"

Fonte	DSCP	TotPkts	TxPkts	L-drops	E-drops
S1	10	22767	22767	0	0
S2	11	33202	33202	0	0
S6	13	2845	2842	0	3
	14	4167	4162	0	5
S3	15	18220	18220	0	0
	16	345	345	0	0
S4	17	18107	18107	0	0
	18	1578	1578	0	0
S5	19	6510	6510	0	0
	20	427	427	0	0
Total		108168	108160	0	8

A Figura 6.8 ilustra a alocação da largura de banda total dentro do domínio, ao longo da simulação. Como pode ser visto, alcançou-se uma economia de até 25% em certos momentos.

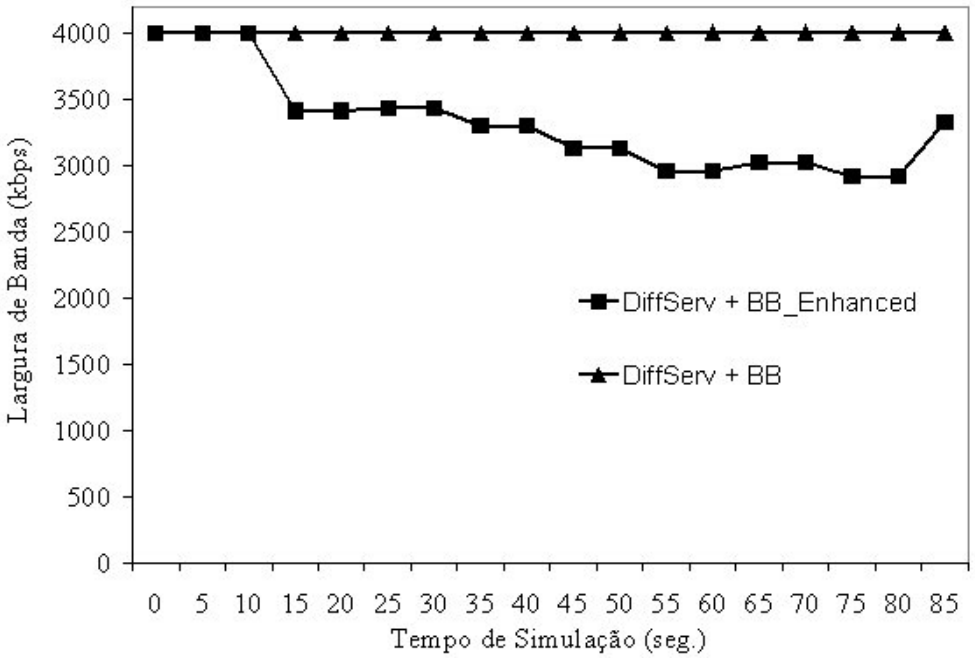


Figura 6.8: Alocação da largura de banda no domínio – 1º cenário "*DiffServ*+BB" *versus* "*DiffServ*+BB*_Enhanced*"

6.4 Sobre Alocação – 2º Cenário

Neste cenário os recursos alocados excedem a capacidade máxima da rede. Para tanto, adicionam-se mais 3 fontes de tráfego (S7, S8 e S9) ao domínio, conectadas ao roteador de borda *Edge1*, conforme ilustra a Figura 6.9.

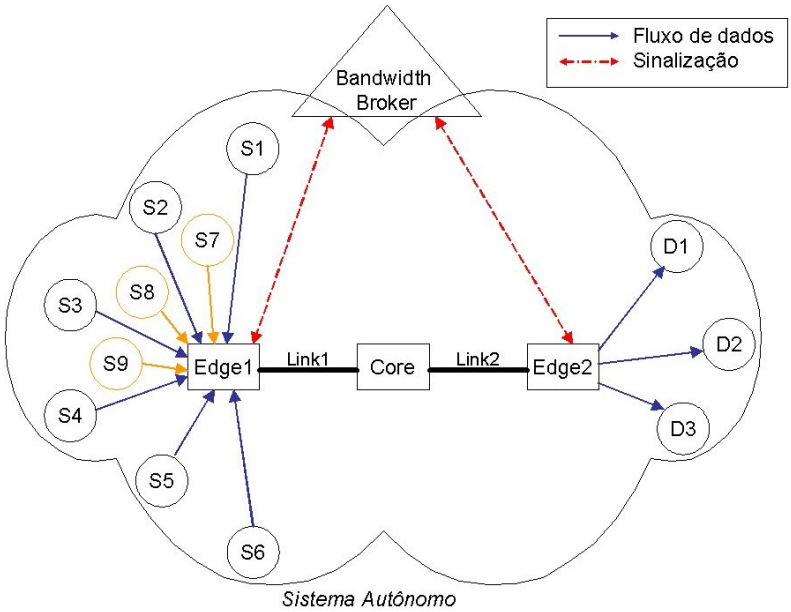


Figura 6.9: Topologia de simulação – 2º Cenário

A Tabela 6.6 lista os parâmetros destas 3 novas fontes de tráfego. Já a Tabela 6.7 apresenta os parâmetros de SLA pré-estabelecido com o domínio.

Tabela 6.6 - Parâmetros das fontes de tráfego

Fonte de Tráfego	Parâmetros		
	Tipo	Pacote	Taxa
S7	Exponencial/UDP. Tempo rajada/ocioso: 300/300ms	210 Bytes	225 Kbps
S8	Exponencial/UDP. Tempo rajada/ocioso: 500/300ms	300 Bytes	325 Kbps
S9	Exponencial/UDP. Tempo rajada/ocioso: 300/500ms.	500 Bytes	450 Kbps

Tabela 6.7 - Parâmetros de SLA

Fonte	CIR (Kbps)	PIR (Kbps)	CoS	DSCPs
S7	225	525	AF	21 / 22 / 0
S8	325	450	AF	23 / 24 / 0
S9	450	-	AF	25 / 26
TOTAL	1000			

Observa-se que na solução "*DiffServ* + BB" o acesso ao domínio destas 3 novas fontes de tráfego teriam sido negados, uma vez que a soma de seus RARs excederiam a capacidade máxima de largura de banda em 1 Mbps. Contudo, na solução "*DiffServ* + BB_*Enhanced*" configurou-se um fator de sobre alocação de 25%, com a finalidade de que o BB aceitasse todos os RARs. Observa-se que a escolha do fator de sobre alocação deu-se em virtude da análise de performance no cenário anterior.

Nos resultados de simulação foram considerados intervalos de confiança com 95% de confiabilidade, sendo realizadas 5 medições.

A Figura 6.10 mostra que novamente o mecanismo de alocação proposto neste trabalho acompanha a necessidade de largura de banda das fontes de tráfego.

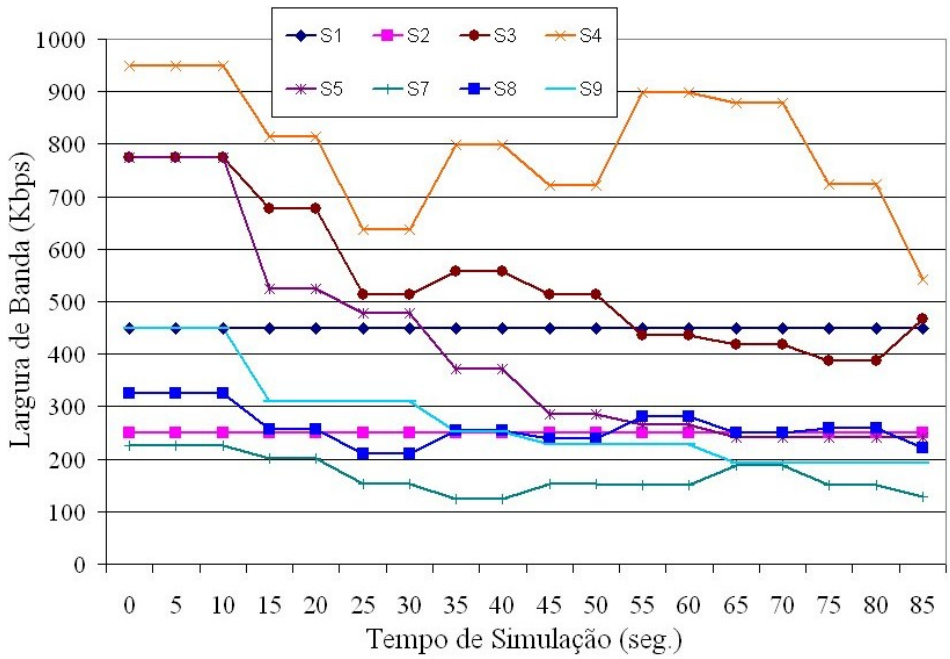


Figura 6.10: Alocação da largura de banda por fonte de tráfego – 2º cenário "*DiffServ*+BB_*Enhanced*"

A Tabela 6.8 mostra que mesmo com o aumento do número de fontes de tráfego, o número de pacotes descartados para as classes EF e AF é preservado, enquanto ocorre um pequeno aumento do número de pacotes descartados para a classe BE, comparando-se com o 1º cenário.

Tabela 6.8 - Pacotes transmitidos e descartados – 2º Cenário "*DiffServ*+ BB*_Enhanced*"

Fonte	DSCP	TotPkts	TxPkts	L-drops	E-drops
S1	10	22767	22767	0	0
S2	11	33202	33202	0	0
S6	13	3482	3472	0	10
	14	3543	3534	0	9
S3	15	16493	16493	0	0
	16	466	466	0	0
S4	17	20868	20868	0	0
	18	1059	1059	0	0
S5	19	4802	4802	0	0
	20	854	854	0	0
S7	21	5537	5537	0	0
	22	181	181	0	0
S8	23	6967	6967	0	0
	24	190	190	0	0
S9	25	3378	3378	0	0
	26	495	495	0	0
Total		124284	124265	0	19

A Figura 6.11 ilustra o atraso fim-a-fim ao longo da simulação e a Figura 6.12 ilustra a função de distribuição acumulativa do atraso fim-a-fim, ambas para as classes EF, AF e BE. Já a Tabela 6.9 apresenta os atrasos fim-a-fim médio e máximo, e seus respectivos intervalos de confiança com 95% de nível de confiança.

Tabela 6.9: Atraso fim-a-fim médio e máximo – 2º Cenário "*DiffServ*+ BB*_Enhanced*"

CoS	Atraso fim-a-fim (mseg)		Intervalo de Confiança (95%)	
	Médio	Máximo	Médio	Máximo
EF	21,7	25,0	21,01 - 22,44	24,33 - 25,92
AF	24,3	42,1	23,23 - 25,11	40,95 - 43,08
BE	68,5	350,9	67,24 - 69,95	349,78 – 352,16

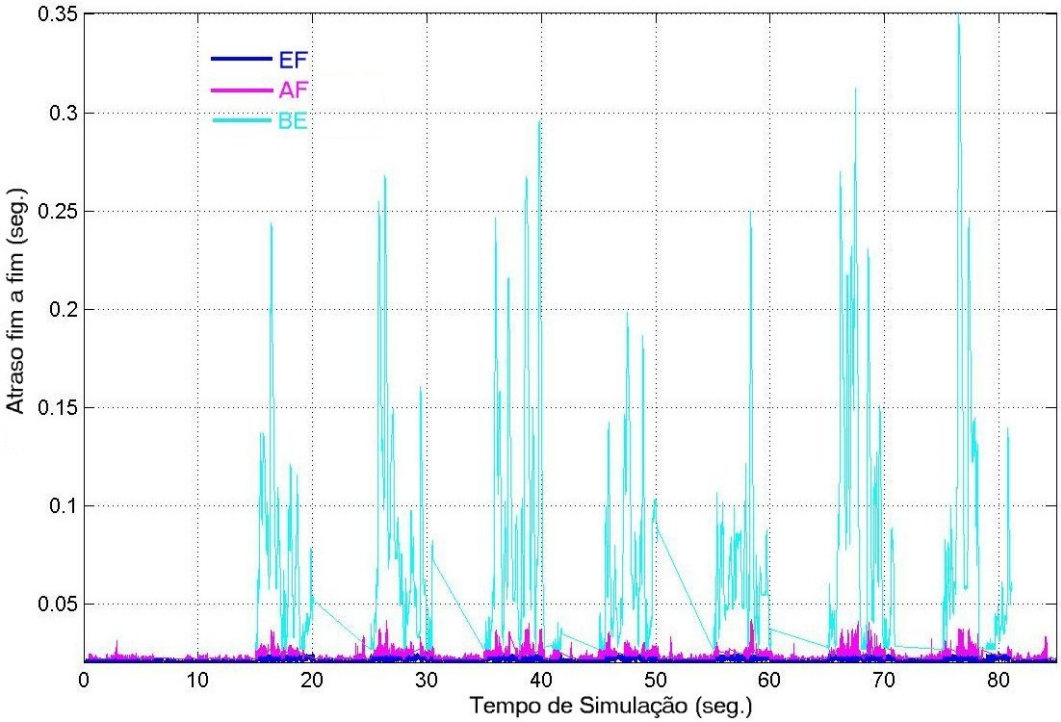


Figura 6.11: Atraso fim-a-fim para as classes EF, AF e BE – 2º cenário "*DiffServ*+BB_*Enhanced*"

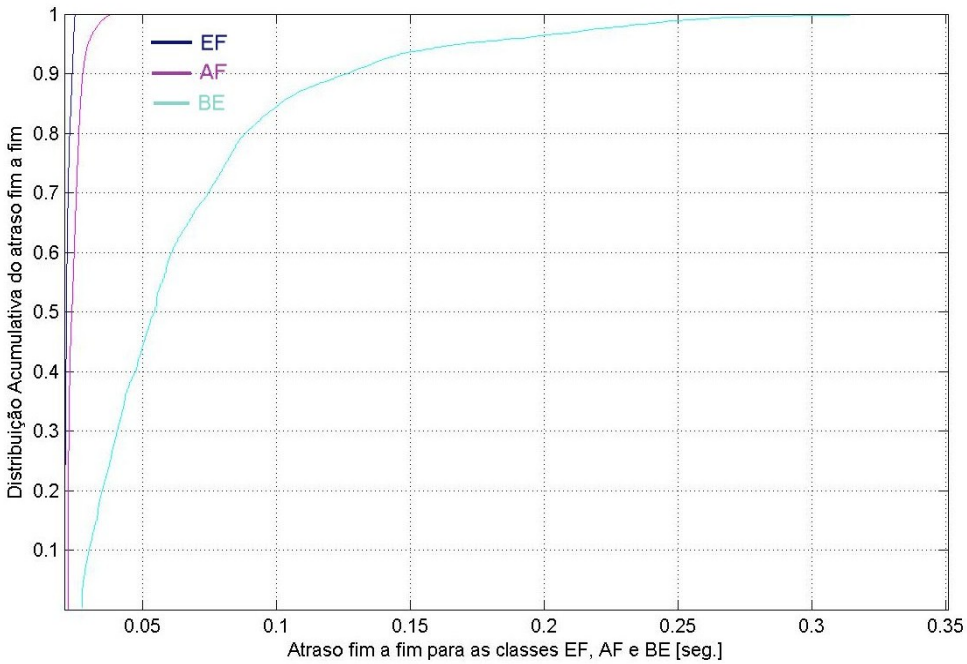


Figura 6.12: Função de distribuição acumulativa do atraso fim-a-fim – 2º cenário "*DiffServ*+BB_*Enhanced*"

A Figura 6.13 compara os resultados de atraso fim-a-fim entre o 1º e o 2º cenário. Observa-se que mesmo com o aumento do número de fontes de tráfego na rede, o atraso fim-a-fim para as classes EF e AF é preservado, enquanto aumenta para a classe BE.

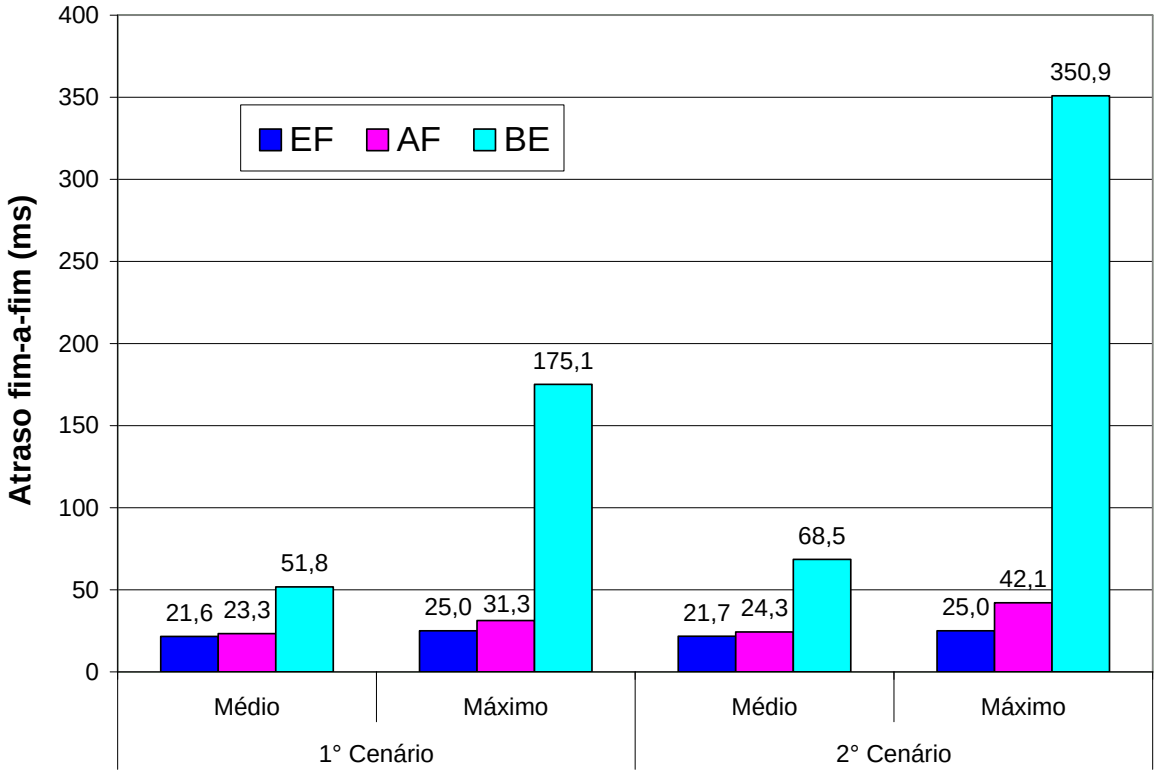


Figura 6.13: Comparação do atraso fim-a-fim entre o 1º e 2º cenário

6.5 Conclusões

Neste capítulo avaliou-se, por meio de modelagem e simulações, a performance do mecanismo de alocação proposto no capítulo anterior em 2 cenários distintos. No 1º cenário a soma dos recursos solicitados corresponde à máxima capacidade da rede, enquanto que no 2º cenário os recursos são sobre alocados. Foram avaliados então os parâmetros de QoS: perda de pacotes e atraso fim-a-fim em um domínio *DiffServ*

utilizando-se de um *Bandwidth Broker* tradicional e em um domínio *DiffServ* utilizando-se de um *Bandwidth Broker* com o mecanismo de alocação proposto.

No 1º cenário foram observados os seguintes resultados:

- O mecanismo de alocação proposto foi capaz de moldar a alocação de recursos, conforme a carga de tráfego dos clientes, resultando em uma economia de até 25% de largura de banda;
- Embora o mecanismo proposto faça com que seja produzido um maior número de pacotes *out-of-profile*, o número de pacotes descartados permaneceu praticamente o mesmo.

Já no 2º cenário foram observados os seguintes resultados:

- Devido à economia alcançada pelo mecanismo proposto foi possível sobre alocar 1 Mbps em recursos da rede. Este mesmo cenário não é possível em um domínio *DiffServ* utilizando-se de um *Bandwidth Broker* tradicional;
- Novamente o mecanismo de alocação proposto foi capaz de moldar a alocação de recursos da rede, conforme a carga de tráfego das fontes;
- Os parâmetros de QoS perda de pacotes e atraso fim-a-fim foram mantidos para as classes EF e AF, às custas de degradação para a classe BE, comparando-se com o 1º cenário.

Capítulo 7

Conclusões Finais, Contribuições e Trabalhos Futuros

A Internet tem passado por grandes avanços nas últimas décadas, tornando-se algo intrínseco ao estilo de vida da sociedade moderna. A Internet é utilizada não apenas como uma ferramenta para distribuição de informações, mas também como um meio de comunicação entre as pessoas e/ou as empresas, entretenimento e lazer, vigilância e monitoramento, transações financeiras, entre outros.

Diante deste contexto um número enorme de serviços e aplicações têm surgido desde então, cada um com características específicas de funcionamento, e específicos requisitos de recursos pela rede. Neste sentido o provimento de qualidade de serviço pelos provedores de serviço à Internet tornou-se um fator fundamental para o desenvolvimento da própria Internet.

Inicialmente, apresentou-se neste trabalho a evolução das arquiteturas para provimento de qualidade de serviço, ressaltando os princípios fundamentais em que foram baseadas, e suas características. Ao longo desta evolução vários modelos foram propostos até chegar a um modelo orientado ao pacote IP e ao domínio administrativo do provedor de serviço. Este modelo é reconhecido pela arquitetura *DiffServ*.

Descreveu-se que uma das principais razões que tornaram a arquitetura *DiffServ* reconhecida como um modelo escalável, e principalmente um modelo reconhecido de fato pelo mercado para o provimento de qualidade de serviço nos *backbones* IP, foi a clara separação entre as funções de controle e de encaminhamento de pacotes em planos distintos, respectivamente em um plano de controle e um plano de dados.

O plano de dados é o plano onde efetivamente ocorre o encaminhamento dos pacotes IP, e é composto pelos seguintes blocos funcionais: classificadores, marcadores, medidores e condicionadores de tráfego, gerenciadores de filas e escalonadores. Já o plano de controle é responsável por coordenar, monitorar e executar ações junto ao plano de dados, e é composto pelos seguintes blocos funcionais: gerenciamento de recursos, controle de admissão e provisão de recursos.

Na seqüência foram apresentadas as características, principais funções e forma de funcionamento para cada um destes blocos funcionais.

Apesar do sucesso da arquitetura *DiffServ* existem ainda alguns desafios a serem superados, sendo que a maioria destes estão relacionados às funções do plano de controle. Neste sentido o IETF e outros órgãos de padronização, tais como o 3GPP e *Packet Cable*, têm elaborado novos documentos com o objetivo de prover uma especificação mais detalhada dos processos envolvidos no plano de controle. Dentre os modelos já propostos, o principal modelo especifica um agente chamado de *Bandwidth Broker* como sendo o

responsável pelas funções do plano de controle. Especifica-se ainda que a qualidade de serviço fim-a-fim é alcançada através do gerenciamento de recursos em cada um dos domínios *DiffServ* envolvidos na comunicação entre um *host* origem e um *host* destino.

Dentre as várias facilidades e benefícios advindos da utilização de um *Bandwidth Broker* em um domínio *DiffServ*, ressalta-se a capacidade de alocar os recursos da rede de uma forma dinâmica. Desta forma, os recursos são alocados sob demanda, a partir de uma solicitação de recursos pelo cliente.

Contudo, apesar da alocação de recursos ser considerada dinâmica, uma vez que é executada sob demanda, existe ainda uma parcela de alocação estática. Isto porque uma vez que os recursos foram concedidos, os mesmos permanecem totalmente alocados, sem levar em consideração o comportamento do tráfego. Assim, os recursos da rede são desperdiçados nos casos em que a carga de tráfego não utiliza a largura de banda alocada.

Neste trabalho propõe-se um mecanismo para alocação de recursos em domínios *DiffServ* capaz de otimizar a utilização dos recursos da rede. O *Bandwidth Broker* permanece como o gerenciador de recursos em um domínio *DiffServ*, contudo é acrescentada a facilidade de realizar ações periódicas junto à alocação de largura de banda na rede, modelando então a alocação de largura de banda conforme a carga instantânea de tráfego.

Portanto, quando a carga instantânea de tráfego na rede requer largura de banda abaixo do valor alocado, uma porção da largura de banda não utilizada retorna para um *pool* de largura de banda disponível. Resultados de simulação mostraram que o mecanismo proposto possibilitou uma economia de até 25% de largura de banda na rede.

Considerando esta economia, demonstrou-se que é possível sobre-alocar recursos na rede. Logo, um número maior de usuários pode ter suas solicitações de acesso ao domínio

concedido pelo *Bandwidth Broker*, aumentando a eficiência de utilização dos recursos da rede.

Observou-se ainda que mesmo com os recursos da rede sobre-alocados, os parâmetros de QoS perda de pacotes e atraso fim-a-fim são preservados para as classes EF e AF às custas da degradação de QoS para a classe BE.

Finalmente, para futuros trabalhos sugerem-se algumas alterações nos processos de controle de admissão e gerenciamento de recursos, de forma que se reduza a freqüência de interações entre os planos de dados e de controle, uma vez que as iterações consomem recursos dos planos, tais como: capacidade de processamento e largura de banda. As interações entre os planos ocorrem principalmente nas etapas de solicitação de recursos ao domínio pelo cliente ou aplicação, provisionamento de recursos pelo domínio, e nas medições periódicas dos recursos da rede pelos roteadores de borda.

Basicamente a alteração consiste na descentralização do processo de controle de admissão. O *Bandwidth Broker* permanece como o elemento no plano de controle responsável pelo controle de admissão, contudo o *Bandwidth Broker* delega aos roteadores de borda a tarefa de executar o controle de admissão sobre uma parcela da capacidade total de largura de banda do domínio. Assim, a comunicação entre os roteadores de borda e o *Bandwidth Broker* ocorreria quando a utilização da largura de banda atingisse um limiar máximo ou mínimo, de forma que o roteador de borda solicitaria ao *Bandwidth Broker* um aumento da largura de banda sobre sua administração, ou o roteador de borda retornaria ao *Bandwidth Broker* uma parte da largura de banda sob sua administração.

Referências Bibliográficas

[1] MIRAS, D. A Survey on Network QoS Needs of Advanced Internet Applications, Computer Science Department, University College London, Website: <http://www.internet2.edu/qos/wg/apps/fellowship/Docs/Internet2AppsQoSNeeds.pdf>, Novembro, 2002.

[2] The Internet Engineering Task Force - Website: <www.ietf.org>

[3] BARDEN, R., CLARK, D., AND SHENKER, S. Integrated Services in the Internet Architecture: An Overview, IETF RFC 1633, Junho, 1994.

[4] BLAKE, S., BLACK, D. L., CARLSON, M., DAVIES, E., WANG, Z., AND WEISS, W. An Architecture for Differentiated Services, IETF RFC 2475, Dezembro, 1998.

[5] KUROSE, J. F., ROSS, K. W. Redes de Computadores e a Internet: Uma Nova Abordagem, Addison Wesley, 1 ed., São Paulo, 2003.

[6] EL-GENDY, M. A., BOSE, A., SHIN, K. G. Evolution of the Internet QoS and Support for Soft Real-Time Application, In proceeding of IEEE, vol. 91, pag. 1086-1104, Julho, 2003.

[7] LI, B., HAMDI, M., JIANG, D., HOU, Y., CAO, X. QoS-enabled Voice Support in the Next-Generation Internet: Issues, Existing Approaches and Challenges, IEEE Communications Magazine, vol. 38, pag. 54-61, Abril, 2000.

[8] PUJOLLE, G., KORNER, U., PERROS, H., Resource Allocation in the New Fixed and Mobile Internet Generation, International Journal of Network Management, vol.13, pag. 181-185, 2003.

[9] NICHOLS, K., JACOBSON, V., ZHANG, L. A Two-bit Differentiated Services Architecture for the Internet, IETF RFC 2638, Julho, 1999.

[10] NICHOLS, K., SAMPSON, L., BARRIOS, R., ADAMS, K., PULLIAN, J., KIM, J. A Strawman Architecture for DiffServ Control Plane Elements, IETF Internet Draft, draft-nichols-dcpel-strawman-arch-00.txt, Outubro, 2005.

[11] BAKER, F., CHAN, K., AND SMITH, A. Management Information Base for the Differentiated Services Architecture, IETF RFC 3289, Maio, 2002.

[12] FINE, M., MCCLOGHRIE, K., SELIGSON, J., CHAN, K., HAHN, S., BELL, C., SMITH, A., AND REICHMEYERS, F. Differentiated Services Quality of Service Policy Information Base, IETF Internet-draft, draft-ietf-diffserv-pib-09.txt, Junho, 2002.

[13] ROSEN, A., VISWANATHAN, A., AND CALLON R. Multiprotocol Label Switching Architecture, IETF RFC 3031, Janeiro, 2001.

[14] AWDUCHE, D., MALCOLM, J., AGOGBUA, J., O´DELL., M., MCMANUS, J. Requirements for Traffic Engineering over MPLS, IETF RFC 2702, Setembro, 2001.

[15] POSTEL, J., Internet Protocol - STD 5, IETF RFC 791, Setembro, 1981.

[16] ALMQUIST, P., Type of Service in the Internet Protocol Suite, IETF RFC 1349, Julho, 1992.

[17] BRADEN, R., ZHANG, L., BERSON, S., HERZOG, S., AND JAMIN, S. Resource Reservation Protocol (RSVP) - Version 1 Functional Specification, IETF RFC 2205, Setembro, 1997.

[18] WROCLAWSKI, J. Specification of the Controlled-Load Network Element Service, IETF RFC 2211, Setembro 1997.

[19] SHENKER, S., PARTRIDGE, C., GUERIN, R. Specification of Guaranteed Quality of Service, IETF RFC 2212, Setembro 1997.

[20] NICHOLS, K., BLAKE, S., BAKER, F., BLACK, D. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, IETF RFC 2474, Dezembro, 1998.

[21] JACOBSON, V., NICHOLS, K., PODURI, K. An Expedited Forwarding PHB, IETF RFC 2598, Junho, 1999.

[22] DAVIE, B., CHARNY, A., BENNETT, J., BENSON, K., LE BOUDEC, J. An Expedited Forwarding PHB (Per-Hop Behavior), IETF RFC 3246, Março, 2002.

[23] HEINANEN, J., BAKER, F., WEISS, W., WROCLAWSKI, J. Assured Forwarding PHB Group, IETF RFC 2597, Junho, 1999.

[24] ANDERSSON, L., DOOLAN, P., FELDMAN, N., FREDETTE, A., THOMAS, B., LDP Specification, IETF RFC 3036, Janeiro, 2001.

[25] LE FAUCHEUR, F., WU, L., DAVIE, B., DAVARI, S., VAANANEN, P., KRISHNAN, R., CHEVAL, P., HEINANEN, J. MultiProtocol Label Switching (MPLS) Support of Differentiated Service, IETF RFC 3270, Maio, 2002.

[26] JAMOUSSI, B., ANDERSSON, L., CALLON, R., DANTU, R., DOOLAN, P., WORSTER, T., FELDMAN, N. Constraint-Based LSP Setup using LDP, IETF RFC 3212, Janeiro, 2002.

[27] SRISURESH, P., JOSEPH, P., An Experimental Extension to OSPF for Traffic Engineering OSPF-TE, IETF Internet-draft, draft-srisuresh-ospf-te-07.txt, Dezembro, 2004.

[28] CARPENTER, B., NICHOLS, K. Differentiated Services in the Internet, In Proceeding of the IEEE, vol. 90, no. 9, pag. 1479 – 1494, Setembro, 2002.

[29] WELZL, M., MUHLHAUSER, M. Scalability and Quality of Service: A Trade-off ?, IEEE Communications Magazine, pag. 32-36, Junho, 2003.

[30] CHRISTIN, N., LIEBEHERR, J. A QoS Architecture for Quantitative Service Differentiations, IEEE Communications Magazine, pag. 38-45, Junho, 2003.

[31] CHIU, J., HUANG, Z., LO, C., HWANG, W., SHIEH, C. Supporting End-to-End QoS in DiffServ/MPLS Networks, IEEE Communications Magazine, pag. 261-266, Junho, 2003.

[32] SUN, W., BHANIRAMKA, P., JAIN, R. Quality of Service using Traffic Engineering over MPLS: An Analysis, IEEE Communications Magazine, pag. 238-241, 2000.

[33] ROSENBAUM, G., JHA, S., HASSAN, M. Empirical Study of Traffic Trunking in Linux-based MPLS test-bed, International Journal of Network Management, no. 13, pag. 277-288, 2003.

[34] ANDERSON, T. Requirements for Separation of IP Control and Forwarding, Internet Draft, Fevereiro, 2002.

[35] ZHANG, Z., DUAN, Z., GAO, L., HOU, Y.T. Decoupling QoS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services, In Proceedings of ACM SIGCOMM_00, Estocolmo, Suécia, pag. 71–83, Agosto, 2000.

[36] FANG, W., SEDDIGH, N. A Time Sliding Window Three Colour Marker (TSWTCM), IETF RFC 2859, Junho, 2000.

[37] SHENKER, S., PARTRIDGE, C., GUERIN, R. Specification of Guaranteed Quality of Service, IETF RFC 2212, Setembro, 1997.

[38] FLOYD, S., JACOBSON, V. Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, vol. 1, nro. 4, pag. 397-413, Agosto, 1993.

[39] TERZIS, A., WANG, L., OGAWA, J., ZHANG, L. A Two-tier Resource Management Model for the Internet, In Proceedings of IEEE GLOBECOM_99, Rio de Janeiro, Brasil, pag. 1779–1791, Dezembro, 1999.

[40] ANJALI, T., SCOGLIO, C., UHL, G. A New Scheme for Traffic Estimation and Resource Allocation for Bandwidth Brokers, Computer Networks 41, pag. 761–777, 2003.

[41] WANG, F., MOHAPATRA, P., MUKHERJEE, S., BUSHMITCH, D. An Efficient Bandwidth Management Scheme for Real-Time Internet Application, Computer Communications 25, pag. 1596-1605, 2002.

[42] BAKIRAS, S., LI, V. A Scalable Architecture for End-to-End QoS Provisioning, Computer Communications 27, pag. 1330-1340, 2004.

[43] 3GPP (3rd Generation Partnership Project), 3GPP TS23.207, <http://www.3gpp.org/>

[44] PacketCable, Dynamic Quality of Services Specifications (PKT-SP-DQOS), <http://www.packetcable.com/>

[45] MultiService Switching Forum, Quality of Service for Next Generation Voice Over IP Networks (MSF-TR-QoS), <http://www.msforum.org/>

[46] INTERNATIONAL TELECOMMUNICATION UNION, H.323: Packet-based Multimedia Communications Systems, Recommendation H.323, Julho, 2003

[47] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., SCHOOLER, E. SIP: Session Initiation Protocol, IETF RFC 3261, Junho, 2000

[48] ARANGO, M., DUGAN, A., ELLIOTT, I., HUITEMA, C., PICKETT, S. Media Gateway Control Protocol (MGCP) Version 1.0, IETF RFC 2705, Outubro, 1999.

[49] CUERVO, F., GREENE, N., RAYHAN, A., HUITEMA, C., ROSEN, B., SEGERS, J. Megaco Protocol Version 1.0, IETF RFC 3015, Novembro, 2000.

[50] QIU, J., KNIGHTLY, E. W., Measurement-Based Admission Control with Aggregate Traffic Envelopes, IEEE/ACM Transactions On Networking, vol. 9, no. 2, Abril, 2001.

[51] FIDLER, M., SANDER, V. A Parameter Based Admission Control for Differentiated Services Networks, Computer Networks 44, pag. 463–479, 2004.

[52] MANTAR, H., HWANG, J., OKUMUS, I., CHAPIN, S. Edge-to-Edge Resource Provisioning and Admission Control in DiffServ Networks, Syracuse University.

[53] DURHAM, D., BOYLE, J., COHEN, R., HERZOG, S., RAJAN, R., SASTRY, A. The COPS (Common Open Policy Service) Protocol, IETF RFC 2748, Janeiro, 2000.

[54] HERZOG, S., BOYLE, J., COHEN, R., DURHAM, D., RAJAN, R., SASTRY, A. COPS usage for RSVP, IETF RFC 2749, Janeiro, 2000.

[55] CHAN, K., SELIGSON, J., DURHAM, D., GAI, S., MCCLOGHRIE, K., HERZOG, S., REICHMEYER, F., YAVATKAR, R., SMITH, A. COPS Usage for Policy Provisioning (COPS-PR), IETF RFC3084, Março, 2001.

[56] REICHMEYER, F., ONG, L., TERZIS, A., ZHANG, L., YAVATKAR, R. A Two-Tier Resource Management Model for Differentiated Services Networks, Internet Draft, Novembro, 1998.

[57] TEITELBAUM, B., CHIMENTO, P. QBone Signaling Design Team - Internet 2 Document, disponível em: <http://qbone.internet2.edu/bb>, acesso em Novembro, 2004.

[58] CASE, J., FEDOR, M., SCHOFFSTALL, M., DAVIN, J. Simple Network Management Protocol (SNMP), IETF RFC1157, Maio, 1990.

[59] CAMARILLO, G., MARSHALL, W., ROSENBERG, J. Integration of Resource Management and Session Initiation Protocol (SIP), IETF RFC 3312, Outubro, 2002.

[60] GUNTER, M., BRAUN, T. Evaluation of Bandwidth Broker Signaling, Swiss National Science Foundation project, Maio, 1999

[61] PEDATELLA, R., MADEIRA, E., MAGALHÃES, M. Um Framework para Obtenção de QoS Fim-a-Fim na Internet, 23º Simpósio Brasileiro de Rede de Computadores, Maio, 2003.

[62] BOURAS, C., PRIMPAS, D., STAMOS, K., STATHIS, N. Design and Implementation of a Bandwidth Broker in a Simulation Environment, 7th International Symposium on Communications Interworking – INTERWORKING, Ottawa, Canada, December, 2004.

[63] TEITELBAUM, B., CHIMENTO, P. Chimento QBone Bandwidth Broker Architecture, <http://qbone.internet2.edu/bb/bboutline2.html>, acesso em Novembro, 2004.

[64] TF-NGN Website - <http://www.terena.nl/tech/task-forces/tf-ngn/>, acesso em Dezembro, 2004.

[65] KIVIMAKI, P. Policy Based Networks & Bandwidth Broker, Tampere University of Technology, Telecommunication Laboratory, Agosto, 2000.

[66] QUARESMA Website - <http://www.gta.ufrj.br/quaresma>, acesso em Junho, 2005.

[67] STOICA, I. Stateless Core: A Scalable Approach for Quality of Service in the Internet, Tese Ph.D., Carnegie Mellon University, Pittsburg, Dezembro, 2000.

[68] OOTTAMAKORN, C., BUSHMITCH, D. A DiffServ Measurement-Based Admission Control Utilizing Effective Envelopes and Services Curves, ICC 2001, 2001.

[69] DUAN, Z., ZHANG, Z., HOU, Y., GAO, L. A Core Stateless Bandwidth Broker Architecture for Scalable Support of Guaranteed Services, IEEE Trans. On Parallel and Distributed Systems, vol. 15, no. 1, Janeiro, 2004.

[70] MASE, K., KOBAYASHI, H. An Efficient End-to-End Measurement-Based Admission Control for VoIP Networks, ICC 2004, 2004.

[71] MASE, K., Toward Scalable Admission Control for VoIP Networks, IEEE Comunications Magazine, Julho, 2004.

[72] VOLLBRECHT, J., CALHOUN, P., FARREL, S., GOMMANS, L., GROSS, G., BRUIJIN, B., LAAT, C., HOLDREGE, M., SPENCE, D. AAA Authorization Application Examples, IETF RFC 2905.

[73] BOURAS, C., STAMOS, K. An Adaptive Admission Control Algorithm for Bandwidth Brokers, Proc. IEEE International Symposium on Network Computing and Applications (NCA04), pag. 243-250, 2004.

[74] YI, D., KIM, J. Dynamic Resource Management Technique with Advance Reservation over QoS-provisioned Networks, Proceedings of SPIE, vol. 4925, pag. 146-155, Outubro, 2002.

[75] MAHAJAN, M., RAMANATHAN, A., PARASHAR, M. Active Resource Management for the Differentiated Services Environment, International Journal of Network Management , vol. 14, pag. 149-165, 2004.

[76] ETSI Standard, Telecommunications and Internet Converged Services and Protocols for Advanced Networks (TISPAN) – Functional Architecture – PSTN/ISDN Emulation Subsystem, Draft ETSI TS 02030 v<1.2.7>, December, 2005.

[77] Network Simulator (NS-2) Webpage - <http://www.isi.edu/nsnam/ns>, acesso em Julho, 2003.

[78] Tracegraph Webpage - <http://www.geocities.com/tracegraph/>, acesso em Junho, 2005.

[79] PIEDA, P., ETHRIDGE, J., BAINES, M., SHALLWANI, F. A Network Simulator Differentiated Services Implementation – Open IP, Nortel Networks, Julho, 2000.

[80] TASSL (The Applied Software System Laboratory) Webpage- <http://www.caip.rutgers.edu/TASSL/>, acesso em Dezembro, 2004.

Trabalhos Publicados pelo Autor:

[81] REIS, L. G. e GUARDIEIRO, P. R**.** Dynamic Resource Management with Enhanced Allocation Mechanism for DiffServ Domains, 4th International Information and Telecommunication Technologies Symposium, I2TS'2006, Florianópolis, Santa Catarina, Brazil, December, 2005.

[82] REIS, L. G. e GUARDIEIRO, P. R. Bandwidth Broker: Providing Guaranteed Services in DiffServ Domains, Accepted for publication in Advanced International Conference on Telecommunications, AICT'06, Guadeloupe, French Caribbean, February, 2006.

[83] REIS, L. G. e GUARDIEIRO, P. R**.** Dynamic Resource Management Mechanism for DiffServ Networks, Accepted for publication in International Conference on Internet and Web Applications and Services, ICIW'06, Guadeloupe, French Caribbean, February, 2006.

[84] REIS, L. G. e GUARDIEIRO, P. R**.** Real Time Resource Management Mechanism for DiffServ Networks, Accepted for publication in Networks and Communication Systems, NCS'2006, Chiang Mai, Thailand, March, 2006.

[85] REIS, L. G. e GUARDIEIRO, P. R**.** A Resource Allocation Mechanism for Dynamic Management in DiffServ Domains, Accepted for publication in International Network Conference, INC'2006, Plymouth, UK, July, 2006.

[86] REIS, L. G. e GUARDIEIRO, P. R**.** An Enhanced Resource Allocation Mechanism for DiffServ Domains, Accepted for publication in 13th International Conference on Telecommunications, ICT'2006, Madeira Island, Portugal, May, 2006.

[87] REIS, L. G. e GUARDIEIRO, P. R. An Enhanced Allocation Resource Mechanism for DiffServ Domains, International Conference on Networking and Services, ICNS'2006, Silicon Valley, USA, July, 2006.

[88] REIS, L. G. e GUARDIEIRO, P. R. Dynamic Resource Management Mechanism for DiffServ Domains, Accepted for publication in ANDESCON 2006, Quito, Ecuador, November, 2006.

Anexos

Códigos Fonte

DsManager.cc

```
#include <stdio.h>
#include "dsManager.h"
#include <string.h>
#include <stdlib.h>


/*-----------------------------------------------------------------------
---------------------------
class DsManagerClass : public TclClass
Instantiates a DsManager in a new simulation.
-----------------------------------------------------------------------
--------------------------*/

static class DsManagerClass : public TclClass {
public:
        DsManagerClass() : TclClass("DsManager") {}
        TclObject* create(int, const char*const*) {
                return (new DsManager);
        }
} class_dsmanager;



/*-------------------------------------------------------------
Manager() constructor
-------------------------------------------------------------*/

DsManager::DsManager()
{
  numEdgePolicyObj = 0;           // numPolicyObj defined to enter client
requests through the BB
  numCorePolicyObj = 0;
  reallocTableSize = 0;
```

```
  codepointTableSize = 0;
  TOTAL_BW_USED = 0;           // calculated using CIR, so bw_used is
defined in bytes per second
  AF_BW_USED = 0;                 // Bandwidth reserved for assured
forwarding.
  EF_BW_USED = 0;                 // Bandwidth reserved for expedited
forwarding.
  BE_BW_USED = 0;                 // Bandwidth reserved for best effort.
  NumQ = 0;
  AF1 = 0;
  AF2 = 0;
}

/*-----------------------------------------------------------------------
----------------
int DsManager::command(int argc, const char*const* argv)
Takes care of interface with Tcl
-----------------------------------------------------------------------
---------------*/

int DsManager::command(int argc, const char*const* argv)
{
  policerType policer;
  nsaddr_t SNode, DNode;


  if(strcmp(argv[1], "setMaxBw") == 0)
    {
      setMaxBw(atoi(argv[2]),atoi(argv[3]),atoi(argv[4]));
      return TCL_OK;
    }
  else if(strcmp(argv[1], "setNumQueues") == 0)
    {
      setNumQueues(atoi(argv[2]));
      return TCL_OK;
    }

  else if(strcmp(argv[1], "addNewPolicy") == 0)              // New
Policy Entry
    {
      SNode =  atoi(argv[2]);
      DNode = atoi(argv[3]);
      int codePt = atoi(argv[5]);
      double cir = 0;
      double cbs = 0;
      double ebs = 0;
      double pir = 0;
      double pbs = 0;

      if(strcmp(argv[4], "EF") == 0)
      {
        policer =  EF;
        cir = atoi(argv[6])/8.0;
        pir = atoi(argv[9])/8.0;
      } else if(strcmp(argv[4], "TSW2CM") == 0)
        {
          policer = TSW2CM;
```

```
          cir = atoi(argv[6])/8.0;
        } else if(strcmp(argv[4], "TSW3CM") == 0)
          {
            policer = TSW3CM;
            cir = atoi(argv[6])/8.0;
            pir = atoi(argv[9])/8.0;
          } else if(strcmp(argv[4], "TokenBucket") == 0)
            {
            policer = tokenBucketPolicer;
            cir = atoi(argv[6])/8.0;
            cbs = atoi(argv[7]);
            } else if(strcmp(argv[4], "srTCM") == 0)
            {
              policer = srTCMPolicer;
              cir = atoi(argv[6])/8.0;
              cbs = atoi(argv[7]);
              ebs = atoi(argv[8]);
            } else if(strcmp(argv[4], "trTCM") == 0)
              {
                policer = trTCMPolicer;
                cir = atoi(argv[6])/8.0;
                cbs = atoi(argv[7]);
                pir = atoi(argv[9])/8.0;
                pbs = atoi(argv[10]);
              }

      addNewPolicy(SNode, DNode, policer, codePt, cir, cbs, ebs, pir,
pbs);
      return TCL_OK;
    }
  else if(strcmp(argv[1], "dynamicAllocation") == 0) {
    nsaddr_t sourceNode = atoi(argv[2]);
    nsaddr_t destNode = atoi(argv[3]);
    dynamicAllocation(sourceNode, destNode);
    return TCL_OK;
  }
  else if(strcmp(argv[1], "addNewPHB") == 0) {
    //policer = atoi(argv[2]);
    policer = policerType(atoi(argv[2]));
    addNewPHB(policer);
    return TCL_OK;
  }
  else if(strcmp(argv[1], "addEdgePolicyObj") == 0) {
    for (int i = 2; i < argc; i++) {
      edgeQueue* ep = (edgeQueue*)TclObject::lookup(argv[i]);
      addEdgePolicyObj(ep);
    }
    return TCL_OK;
  }
  else if(strcmp(argv[1], "addCorePolicyObj") == 0) {
    for (int i = 2; i < argc; i++) {
      coreQueue* cp = (coreQueue*)TclObject::lookup(argv[i]);
      addCorePolicyObj(cp);
    }
    return TCL_OK;
  }
  else if(strcmp(argv[1], "addReallocTableEntry") == 0) {
```

```
    int cp = atoi(argv[2]);
    double initialCir = atoi(argv[3])/8.0;
    double firstChange = atoi(argv[4])/8.0;
    double secondChange = atoi(argv[5])/8.0;
    addReallocTableEntry(cp, initialCir, firstChange, secondChange);
    return TCL_OK;
  }
  else if (strcmp(argv[1], "endFlow") == 0) {
   nsaddr_t SNode = atoi(argv[2]);
   nsaddr_t DNode = atoi(argv[3]);
   endFlow(SNode, DNode);
   return TCL_OK;
  }
  else if (strcmp(argv[1], "ShowBwStatus") == 0) {
   ShowBwStatus(atoi(argv[2]));
   return TCL_OK;
  }

}


/*-----------------------------------------------------------------
void DsManager::addNewPolicy(nsaddr_t SNode, nsaddr_t DNode, policerType
policer, int cp, double cir, double cbs, double ebs, double pir, double
pbs)
method to add a new policy to the dsPolicy table

EF            InitialCodePoint  CIR  PIR
TSW2CM        InitialCodePoint  CIR
TSW3CM        InitialCodePoint  CIR  PIR
TokenBucket   InitialCodePoint  CIR  CBS
srTCM         InitialCodePoint  CIR  CBS  EBS
trTCM         InitialCodePoint  CIR  CBS  PIR  PBS

-----------------------------------------------------------------*/

void DsManager::addNewPolicy(nsaddr_t SNode, nsaddr_t DNode, policerType
policer, int cp, double cir, double cbs, double ebs, double pir, double
pbs){

  if (MAX_BW > TOTAL_BW_USED)
    {
      Tcl& tcl = Tcl::instance();

      switch(policer)
      {
      case EF:
        if (pir > (MAX_EF_BW - EF_BW_USED))
          {
            printf("ERROR: NO BANDWIDTH TO ALLOCATE\n");
            Tcl_SetVar(tcl.interp(), "allocate", "0",TCL_GLOBAL_ONLY );
          } else
            {
            Tcl_SetVar(tcl.interp(), "allocate", "1", TCL_GLOBAL_ONLY);
            EF_BW_USED += pir;
            }
        break;
```

```
      default:
        if (cir > (MAX_AF_BW - AF_BW_USED))
            {
            printf("ERROR: NO BANDWIDTH TO ALLOCATE\n");
            Tcl_SetVar(tcl.interp(), "allocate", "0",TCL_GLOBAL_ONLY );
            } else
            {
              Tcl_SetVar(tcl.interp(), "allocate", "1 ",
TCL_GLOBAL_ONLY);
              AF_BW_USED += cir;
            }
        break;
      }
      TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;
    } else
      {
      printf("NO MORE ALLOCATIONS POSSIBLE \n");
      }
  return;
}



/*-------------------------------------------------------------------------
--------------

void addNewPHB()
Assigns codepoints to queues depending on QStatus

-------------------------------------------------------------------------
------------*/

void DsManager::addNewPHB(policerType policer)
{
  Tcl& tcl = Tcl::instance();

  if (NumQ == 4)
    {
      if (AF1 < AF2)
      {
        switch(policer)
          {
          case EF:
            Tcl_SetVar(tcl.interp(), "setPHB", "0", TCL_GLOBAL_ONLY);
            break;
          default:
            Tcl_SetVar(tcl.interp(), "setPHB", "1", TCL_GLOBAL_ONLY);
            AF1++;
            break;
          }
      }
      else
      {
        switch(policer)
          {
          case EF:
            Tcl_SetVar(tcl.interp(), "setPHB", "0", TCL_GLOBAL_ONLY);
```

```
            break;
          default:
            Tcl_SetVar(tcl.interp(), "setPHB", "2", TCL_GLOBAL_ONLY);
            AF2++;
            break;
          }
      }
      return;
    }
  else
    {
      return;
    }
}

/*-----------------------------------------------------------------------
--------------------------------

addEdgePolicyObj(): Stores the various links between the edge and the
core routers
that need to be broker managed.

-----------------------------------------------------------------------
--------------------------------*/

void DsManager::addEdgePolicyObj(edgeQueue* edgePol)
{
  if (numEdgePolicyObj > 9)
    {
      printf("ERROR: No more policy objects can be added\n");
    } else
      {
      edgePolicyObj[numEdgePolicyObj] =edgePol;
      numEdgePolicyObj++;
      }
  return;
}

/*-----------------------------------------------------------------------
--------------------------------

addCorePolicyObj(): Stores the various links between the core and the
edge routers
that need to be broker managed.

-----------------------------------------------------------------------
--------------------------------*/

void DsManager::addCorePolicyObj(coreQueue* corePol)
{
  if (numCorePolicyObj > 9)
    {
      printf("ERROR: No more policy objects can be added\n");
    } else
      {
      corePolicyObj[numCorePolicyObj] =corePol;
      numCorePolicyObj++;
```

```
      }
  //  printf("Successfully added policy Objects\n");
  return;
}


#if 0
/*-----------------------------------------------------------------------
--------------------------------

addPolicyObj(): Stores the various links between the edge and the core
routers
that need to be broker managed.

-----------------------------------------------------------------------
-------------------------------*/

void DsManager::addPolicyObj(edgeQueue* pol)
{
  if (numPolicyObj > 9)
    {
      printf("ERROR: No more policy objects can be added\n");
    } else
      {
      policyObj[numPolicyObj] =pol;
      numPolicyObj++;
      }
  return;
}



/*-----------------------------------------------------------------------
--------------------------------
a method to add policys to all the policy objects

-----------------------------------------------------------------------
-------------------------------*/

void DsManager::policyObjEntries(nsaddr_t SNode, nsaddr_t DNode,
policerType policer, int cp, double cir, double cbs, double ebs, double
pir, double pbs)
{
  int first, second, third, fourth;

  for(int i=0; i<numPolicyObj; i++)
    {
      policyObj[i]->addNewPolicyEntry(SNode, DNode, policer, cp, cir,
cbs, ebs, pir, pbs);
      printf("The policies have been added\n");
      policyObj[i]->addNewPolicer(policer, first, second, third, fourth);
      printf("The Policer has been added\n");

    }

  return;
}
```

```
#endif

/*-----------------------------------------------------------------------
------------
setMaxBw() sets the maximum bandwidth available to allocate.
need to check on the sense behind converting it between bits and bytes???
-----------------------------------------------------------------------
-----------*/

void DsManager::setMaxBw(int bw, int ef_bw, int af_bw)
{
  MAX_BW = (double) bw/8.0;

  MAX_EF_BW = (double) ef_bw/8.0;

  //MAX_AF_BW = MAX_BW*(5.0/8.0);
  MAX_AF_BW = af_bw/8.0;

  MAX_BE_BW = MAX_BW - (MAX_EF_BW + MAX_AF_BW);
  //MAX_BE_BW = MAX_BW*(1.0/5.0);

  //MAX_EF_BW = MAX_BW - (MAX_BE_BW + MAX_AF_BW);

  printf("MAX_BW    is %f\nMAX_EF_BW is %f\nMAX_AF_BW is %f\nMAX_BE_BW is
%f\n", MAX_BW*8.0, MAX_EF_BW*8.0, MAX_AF_BW*8.0, MAX_BE_BW*8.0);
  return;
}

/*-----------------------------------------------------------------------
----------*/

void DsManager::ShowBwStatus(int t)
{
  printf("Tempo-> %d:\n", t);
  printf("MAX_BW    is: %f and TOTAL_BW_USED is: %f\n", MAX_BW*8.0,
TOTAL_BW_USED*8.0);
  printf("MAX_EF_BW    is: %f and EF_BW_USED is: %f\n", MAX_EF_BW*8.0,
EF_BW_USED*8.0);
  printf("MAX_AF_BW    is: %f and AF_BW_USED is: %f\n", MAX_AF_BW*8.0,
AF_BW_USED*8.0);
  printf("MAX_BE_BW    is: %f and BE_BW_USED is: %f\n", MAX_BE_BW*8.0,
BE_BW_USED*8.0);
  printf("\n");
}

/*-----------------------------------------------------------------------
----------*/


/*-----------------------------------------------------------------------
---------------
setNumQueues(NumQ)
sets the Number of queues in the diffserv domain.
This lets the BB decide based on the number of queues tp which queue the
next
codepoint should be assigned.
```

```
------------------------------------------------------------------------
-------------*/

void DsManager::setNumQueues(int nQ)
{
  NumQ = nQ;
  return;
}


/*------------------------------------------------------------------------
-----------------------------
  void DsManager::dynamicAllocation()

method provides for dynamic allocation. checks the meter, depending on
the type of policer and the
code points, decides the new codepoint to be used.

------------------------------------------------------------------------
-----------------------------*/

void DsManager::dynamicAllocation(nsaddr_t source, nsaddr_t dest)
{
  int i;

  for(i = 0; i < numEdgePolicyObj; i++) {
    tableEntry = edgePolicyObj[i]->getPolicyTableEntry(source, dest);
    if(tableEntry != NULL) break;
  }

  if(i == numEdgePolicyObj) {
    printf("ERROR!!! No whatever found from source %d to destination
%d\n", source, dest);
    return;
  }

  switch(tableEntry->policer)
    {
    case EF:
      if (tableEntry->avgRate <= ((tableEntry->pir)/2) + 1000.0)
      {
        if (tableEntry->cir <= ((tableEntry->pir)*2/3 + 500.0))
          {
          }
        else if (tableEntry->cir <= ((tableEntry->pir)*3/4 + 500.0))
          {
            upgradeTwo(tableEntry);
            EF_BW_USED = EF_BW_USED - (tableEntry->pir)/12;
          }
        else
          {
            upgradeTwo(tableEntry);
            EF_BW_USED = EF_BW_USED - (tableEntry->pir)/3;
          }
      }
      else if (tableEntry->avgRate <= ((tableEntry->pir)*2/3) + 1000.0)
      {
```

```
        if (tableEntry->cir <= ((tableEntry->pir)*2/3 + 500.0))
          {
            if (MAX_EF_BW >= EF_BW_USED + (tableEntry->pir)/12) {
            downgradeOne(tableEntry);
            EF_BW_USED = EF_BW_USED + (tableEntry->pir)/12;
            }
            else {
            }
          }
        else if (tableEntry->cir <= ((tableEntry->pir)*3/4 + 500.0))
          {
          }
        else
          {
            upgradeOne(tableEntry);
            EF_BW_USED = EF_BW_USED - (tableEntry->pir)/4;
          }
      }
      else
      {
        if (tableEntry->cir <= ((tableEntry->pir)*2/3 + 500.0))
          {
            downgradeTwo(tableEntry);
            EF_BW_USED = EF_BW_USED + (tableEntry->pir)/3;
          }
        else if (tableEntry->cir <= ((tableEntry->pir)*3/4 + 500.0))
          {
            if (MAX_EF_BW >= EF_BW_USED + (tableEntry->pir)/4) {
            downgradeTwo(tableEntry);
            EF_BW_USED = EF_BW_USED + (tableEntry->pir)/4;
            }
            else {
            }
          }
        else
          {
          }
      } break;
    default:
      if (tableEntry->avgRate < tableEntry->cir)
      {
        AF_BW_USED = AF_BW_USED - (tableEntry->cir - tableEntry-
>avgRate)/2.0;
        tableEntry->cir = tableEntry->cir - (tableEntry->cir -
tableEntry->avgRate)/2.0;
      }
      else if (tableEntry->avgRate < tableEntry->pir)
      {
        if (MAX_AF_BW >= AF_BW_USED + (tableEntry->avgRate - tableEntry-
>cir))
          {
            AF_BW_USED = AF_BW_USED + (tableEntry->avgRate - tableEntry-
>cir);
            tableEntry->cir = tableEntry->avgRate;
          }
        else
          {
```

```
          }
      }
      else
      {
      }
      break;
    }
  TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;
  return;
}

/*-------------------------------------------------------------------------
----------------------------

void DsManager::addReallocTableEntry(int cp, double cir)

This method changes the policy marked in a a pkt during runtime
to regain bw.

-------------------------------------------------------------------------
---------------------------*/

void DsManager::addReallocTableEntry(int cp, double initialCir, double
first, double second)
{
  if (reallocTableSize == MAX_REALLOC)
    {
      printf("No more policies accepted\n");
    }
  else
    {
      reallocTable[reallocTableSize].codePt = cp;
      reallocTable[reallocTableSize].initialCir = initialCir;
      reallocTable[reallocTableSize].firstChange = first;
      reallocTable[reallocTableSize].secondChange = second;
      reallocTableSize++;
    }
  return;
}

/*-------------------------------------------------------------------------

void DsManager::upgradeOne()

-------------------------------------------------------------------------*/

void DsManager::upgradeOne(policyTableEntry* tableEntry)
{
  for (int i=0; i < reallocTableSize; i++)
    {
      if (reallocTable[i].codePt == tableEntry->codePt)
      tableEntry->cir = reallocTable[i].firstChange;
    }
  return;
}

/*-------------------------------------------------------------------------
```

```
void DsManager::upgradeTwo()

------------------------------------------------------------------------*/

void DsManager::upgradeTwo(policyTableEntry* tableEntry)
{
  for (int i=0; i < reallocTableSize; i++)
    {
      if (reallocTable[i].codePt == tableEntry->codePt)
      tableEntry->cir = reallocTable[i].secondChange;
    }
  return;
}

/*------------------------------------------------------------------------

void DsManager::downgradeTwo()

------------------------------------------------------------------------*/

void DsManager::downgradeTwo(policyTableEntry* tableEntry)
{
  for (int i=0; i < reallocTableSize; i++)
    {
      if (reallocTable[i].codePt == tableEntry->codePt)
      tableEntry->cir = reallocTable[i].initialCir;
    }
  return;
}

/*------------------------------------------------------------------------

void DsManager::downgradeOne()

------------------------------------------------------------------------*/

void DsManager::downgradeOne(policyTableEntry* tableEntry)
{
  for (int i=0; i < reallocTableSize; i++)
    {
      if (reallocTable[i].codePt == tableEntry->codePt)
      tableEntry->cir = reallocTable[i].firstChange;
    }
  return;
}

/*------------------------------------------------------------------------
---

void DsManager::endFlow()

------------------------------------------------------------------------
-*/

void DsManager::endFlow(nsaddr_t SNode, nsaddr_t DNode)
{
```

```
  double bw;
  policerType policer;

  for(int i = 0; i < numEdgePolicyObj; i++) {
    tableEntry = edgePolicyObj[i]->getPolicyTableEntry(SNode, DNode);
    bw = tableEntry->cir;

    policer = tableEntry->policer;
    edgePolicyObj[i]->removePolicyTableEntry(SNode, DNode);

    switch(policer)
      {
      case EF:
      EF_BW_USED = EF_BW_USED - bw;
      default:
      AF_BW_USED = AF_BW_USED - bw;
      }

    TOTAL_BW_USED = EF_BW_USED + AF_BW_USED + BE_BW_USED;

    if(tableEntry != NULL) break;
  }
  return;
}
```

DsManager.h

```
/*
 *This class acts as a manager agent for one ds domain.
 *It works on the policy table in the bwBroker class, and decides
 *on a good allocation scheme for the requested bandwidth.
 */


#ifndef dsManager_h
#define dsManager_h
#include "dsPolicy.h"
#include "dsred.h"
#include <string.h>
#include "edge.h"
#include "core.h"

#define MAX_REALLOC 40


/*---------------------------------------------------------------------
structure of reallocation table entry
---------------------------------------------------------------------*/

struct reallocTableEntry
{
  int codePt;
  double initialCir;
  double firstChange;
```

```
  double secondChange;
};


class DsManager : public TclObject
{

 private:
  edgeQueue* edgePolicyObj[10];
  coreQueue* corePolicyObj[10];

  int numEdgePolicyObj;
  int numCorePolicyObj;
  int codepointTableSize;
  int reallocTableSize;
  double TOTAL_BW_USED;
  double MAX_BW;
  double MAX_EF_BW, EF_BW_USED;
  double MAX_AF_BW, AF_BW_USED;
  double MAX_BE_BW, BE_BW_USED;
  int NumQ, cp, AF1, AF2;
  double cir, cbs, ebs, pir, pbs;

  Tcl_Interp *interp;
  Policy* polObj;
  int allocate;
  double avgRate;
  policyTableEntry* tableEntry;
  reallocTableEntry reallocTable[MAX_REALLOC];


 public:
  DsManager();
  int command(int argc, const char*const* argv);
  inline void addNewPolicy(nsaddr_t SNode, nsaddr_t DNode, policerType
policer, int cp, double cir, double cbs, double ebs, double pir, double
pbs);
  inline void dynamicAllocation(nsaddr_t source, nsaddr_t dest); //
performing dynamic allocation on the policies that have been defined
  inline void setMaxBw(int bw, int ef_bw, int af_bw);
  void addEdgePolicyObj(edgeQueue* edgePol);
  void addCorePolicyObj(coreQueue* corePol);
  void setNumQueues(int numQ);
  void policyObjEntries(nsaddr_t SNode, nsaddr_t DNode, policerType
policer, int cp, double cir, double cbs, double ebs, double pir, double
pbs);
  void addNewPHB(policerType policer);
  void addReallocTableEntry(int cp, double intialCir, double first,
double second);
  void downgradeOne(policyTableEntry* tableEntry);
  void downgradeTwo(policyTableEntry* tableEntry);
  void upgradeOne(policyTableEntry* tableEntry);
  void upgradeTwo(policyTableEntry* tableEntry);
  void endFlow(nsaddr_t SNode, nsaddr_t DNode);
  void ShowBwStatus(int tempo);
};
#endif /* dsManager_h   */
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo