UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UM ESTUDO COMPARATIVO DOS ALGORITMOS TCP RENO E TCP VEGAS SOBRE UMA REDE DE SERVIÇOS DIFERENCIADOS, EM TERMOS DE DISTRIBUIÇÃO DE BANDA PASSANTE

RUY DE OLIVEIRA

FEVEREIRO

2001

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UM ESTUDO COMPARATIVO DOS ALGORITMOS TCP RENO E TCP VEGAS SOBRE UMA REDE DE SERVIÇOS DIFERENCIADOS, EM TERMOS DE DISTRIBUIÇÃO DE BANDA PASSANTE

Dissertação apresentada por Ruy de Oliveira à Universidade Federal de Uberlândia para obtenção do título de Mestre em Engenharia Elétrica aprovada em 07/02/2001 pela Banca examinadora:

Professor Paulo Roberto Guardieiro, Dr. (UFU) – Orientador

Professor Shusaburo Motoyama, PhD. (Unicamp)

Professor Jamil Salem Barbar, Dr. (UFU)

UM ESTUDO COMPARATIVO DOS ALGORITMOS TCP RENO E TCP VEGAS SOBRE UMA REDE DE SERVIÇOS DIFERENCIADOS, EM TERMOS DE DISTRIBUIÇÃO DE BANDA PASSANTE

RUY DE OLIVERIA

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

Prof. Dr. Paulo Roberto Guardieiro

Orientador

Prof. Dr. Luís Carlos de Freitas

Coordenador do Curso de Pós-Graduação

DEDICATÓRIA

Ao meus queridos pais, Antônio e Adolfina, às minhas amada esposa Cristiane e querida filha Rebeca, pela compreensão, paciência e amor que sempre demonstraram.

"Feliz é o homem que acha sabedoria, e o homem que adquire entendimento; pois melhor é o lucro que ela dá do que o lucro da prata, e a sua renda do que o ouro. Mais preciosa é do que as jóias, e nada do que possas desejar é comparável a ela." (Prov. 3: 13-15).

AGRADECIMENTOS

À Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia pelos recursos oferecidos para a execução deste estudo.

Ao meu orientador Prof. Dr. Paulo Roberto Guardieiro por sua orientação e dedicação na realização deste trabalho.

À Escola Técnica Federal de Mato Grosso pelo desprendimento e iniciativa de apoiar seu corpo docente na obtenção de aperfeiçoamento para uma melhor contribuição ao ensino médio no país.

À CAPES Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior pela ajuda financeira recebida no decurso do meu trabalho de pós-graduação.

Aos colegas e amigos do Laboratório de Pesquisa em Redes, Clidenor, Daniela, Eliane, Haroldo, Leandra, Johann e também ao amigo Valmir, mestrando pelo laboratório de automação, pelo apoio e incentivo que sempre me dispensaram.

Aos técnicos administrativos Joana, Gonçalo e José Maria pela presteza com que sempre me atenderam.

Aos professores do Curso de Pós-graduação em Engenharia Elétrica, em especial aos professores Edgar e Camacho, o meu muito obrigado.

RESUMO

Um Estudo Comparativo dos Algoritmos TCP Reno e TCP Vegas sobre uma Rede de Serviços Diferenciados, em termos de Distribuição de Banda Passante

Apresenta-se um estudo por intermédio de simulações sobre a influência dos parâmetros α e β do algoritmo TCP Vegas, bem como dos parâmetros do algoritmo RIO, no atributo justiça, em termos de distribuição de banda passante proporcional ao serviço contratado por cada usuário, de uma rede de Serviços Diferenciados (DiffServ). O algoritmo TCP Reno é simulado de maneira que os resultados obtidos para o TCP Vegas possam ser comparados com os daquele. Avalia-se também a interoperação entre estes dois algoritmos TCP e a eficiência dos mesmos diante da agressividade do UDP. Dois modelos de simulação são empregados, um com apenas duas fontes e outro com dez fontes. Avalia-se três conjuntos de valores para o algoritmo RIO e quatro para os parâmetros α e β. A análise dos resultados revelou que o problema de injustiça demostrado em pesquisas recentes a respeito do TCP Vegas para uma rede IP convencional também ocorre num ambiente DiffServ, e que o ajuste adequado dos parâmetros mencionados pode minimizar tais problemas. Os melhores resultados foram conseguidos com o modelo composto por duas fontes, utilizando-se $\alpha=\beta$ e valores "extremamente pequenos" para o RIO. Neste caso, conseguiu-se, ainda uma boa interoperação entre os dois algoritmos TCP estudados. Com o modelo de dez fontes não se observou vantagem significativa no ajuste de tais parâmetros e o TCP Reno superou o TCP Vegas na maioria dos casos. O tráfego UDP se mostrou agressivo diante de ambos os algoritmos TCP, independentemente dos ajuste empregados.

Palavras-chave: TCP Vegas, TCP Reno, RIO.

ABSTRACT

A Comparative Study about TCP Reno and TCP Vegas in a Differentiated Services Network, in terms of Bandwidth Distribution

It is presented a study, based on simulations, about the influence of α and β parameters included in TCP Vegas algorithm as well as the influence of the RIO algorithm parameters, on fairness of a Differentiated Services (DiffServ) network. Fairness is treated here as the capacity of the network in sharing its bandwidth proportionally to the service requested for each user. The TCP Reno algorithm is simulated in a way that allows its results to be compared with the ones from TCP Vegas. It is also evaluated the interoperation between these two TCP algorithms and the efficiency of each one with regards to the UDP aggressiveness. Two simulation models are employed, the first one with only two sources and the other one with ten sources. Three sets of values for the RIO algorithm and four sets of values for α and β parameters are evaluated too. The analysis of the results revealed that the unfairness problem demonstrated in recent researches related to the TCP Vegas in conventional IP networks, also happens in a DiffServ environment, and that the adequate adjustment for the mentioned parameters can minimize such problems. The best results were obtained from the model which consists of two sources, by using $\alpha=\beta$ and "extremely small" values for RIO. In this case, a good interoperation was gotten between both TCP algorithms. Regarding to the ten-sources model, no important advantage was observed when adjusting those parameters and in most cases TCP Reno surpassed TCP Vegas. The UDP traffic showed an aggressive behavior against both TCP algorithms, regardless of the employed adjustments.

Keywords: TCP Vegas, TCP Reno, RIO.

UM ESTUDO COMPARATIVO DOS ALGORITMOS TCP RENO E TCP VEGAS SOBRE UMA REDE DE SERVIÇOS DIFERENCIADOS, EM TERMOS DE DISTRIBUIÇÃO DE BANDA PASSANTE

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SÍMBOLOS

ABR	Available Bit Rate
ACK	Acknowledge
AF	Assured Forwarding
AS	Assured Service
ATM	Asynchronous Transfer Mode
BA	Behavior Aggregate
BB	Bandwidth Broker
BE	Best Effort
BGP	Border Gateway Protocol
CAC	Connection Admission Control
CBR	Constant Bit Rate
CBS	Committed Burst Size
CDVpp	peak-to-peak Cell Delay Variation
CDVT	Cell Delay Variation Tolerance
CER	Cell Error Rate
CIR	Committed Information Rate
CLIP	Classical IP
CLP	Cell Loss Priority
CLR	Cell Loss Rate
CLS	Controlled Load Service
CMR	Cell Misinsertion rate
COS	Class of Service

CTDmáx	maximum Cell Transfer Delay
CU	Currently Unused
CWND	Congestion Window
DiffServ	Differentiated Services
DS	Differentiated Services
DSCP	Differentiated Services Codepoint
EBS	Excess Burst Size
ECN	Explicit Congestion Notification
ED	Edge Device
EF	Expedited Forwarding
ER	Edge Router
FEC	Forward Equivalence Class
FIFO	First In First Out
FTP	File Transfer Protocol
GFR	Guaranteed Frame Rate
GS	Guaranteed Service
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IntServ	Integrated Services
IP	Internet Protocol
IPX	Internetwork Packet Exchange
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
ITU-T	International Telecommunications Union-Telecommunication
LAN	Local Area Network

LANE	LAN Emulation
LDP	Label Distribution Protocol
LSP	Label Switched Path
LSR	Label Switched Router
MBS	Maximum Burst Size
MCR	Minimum Cell Rate
MF	Multi Field
MFS	Maximum Frame Size
MPLS	Multiprotocol Label Switching
MPOA	Multiprotocol over ATM
nrt-VBR	Non-Real-Time Variable Bit Rate
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PBS	Peak Burst Size
PCR	Peak Cell Rate
PHB	Per Hop Behavior
PHOP	Previous Hop
PIR	Peak Information Rate
PNNI	Private Network to Network Interface
PRR	Priority Round Robin
PS	Premium Service
PWFQ	Priority Weighted Fair Queue
QoS	Quality of Service
RED	Random Early Detection
REM	Random Early Marking

RIO	RED with IN and OUT
RM	Resource Management
RR	Round Robin
RSVP	Resource ReSerVation Protocol
RTT	Round Trip Time
rt-VBR	Real-Time Variable Bit Rate
SCR	Sustainable Cell Rate
SECBR	Severely Errored Cell Block Ratio
SLA	Service Level Agreement
SrTCM	Single Rate Three Color Marker
SSTHRESH	Slow Start Threshold
TB	Token Bucket
TCP	Transmission Control Protocol
ToS	Type of Service
UBR	Unspecified Bit Rate
UDP	User Network Interface
VBR	Variable Bit Rate
VCC	Virtual Channel Circuit
VCI	Virtual circuit Identifier
VLL	Virtual Leased Line
VoD	Video on Demand
VoIP	Video over IP
VPI	Virtual Path Identifier
VPN	Virtual Private Network
WAN	Wide Area Network

WFQ	Weighted Fair Queue
WRED	Weighted RED
WRR	Weighted Round Robin
WWW	World Wide Web

UM ESTUDO COMPARATIVO DOS ALGORITMOS TCP RENO E TCP VEGAS SOBRE UMA REDE DE SERVIÇOS DIFERENCIADOS, EM TERMOS DE DISTRIBUIÇÃO DE BANDA PASSANTE

CAPÍTULO I

INTRODUÇÃO

O crescimento fenomenal da Internet observado nesta última década tornou obsoleto o serviço padrão empregado nesta rede, denominado de serviço de "melhor esforço". Este serviço proporciona o mesmo nível de prioridade a todos os fluxos submetidos a esta rede. No entanto, as aplicações emergentes, tais como vídeoconferência e telefonia IP dentre outras aplicações interativas sobre a Internet, demandam níveis de Qualidade de Serviço (QoS – *Quality of Service*) diferenciados e confiáveis e, isso, não pode ser oferecido pelo serviço mencionado.

Nesse sentido, a IETF (*Internet Engineering Task Force*) tem direcionado esforços rumo ao desenvolvimento de novas tecnologias que possam prover QoS na Internet atual. Dentre as tecnologias mais promissoras, visto o crescente interesse pela mesma observado nos últimos tempos, está a arquitetura "Serviços Diferenciados" (DiffServ) [15,45,29], a qual é

objeto de estudo deste trabalho. Esta arquitetura é avaliada aqui sob o ponto de vista da capacidade da mesma em oferecer "justiça" (*fairness*) na distribuição de banda passante aos seus usuários, empregando-se o algoritmo TCP Vegas [35,36,43] e comparando-o com o TCP Reno [30,31,42,44].

Num ambiente DiffServ, a justiça representa um parâmetro de fundamental importância, uma vez que os usuários nesta rede esperam obter proporções diferenciadas dos recursos da rede, relativamente aos seus serviços contratados, independentemente do estado de ocupação em que se encontra essa rede. Muitas pesquisas têm abordado esse assunto, tais como [30,31,42], mas todas elas consideraram apenas o algoritmo TCP Reno como mecanismo de controle de fluxo dos *hosts*. Por outro lado, o algoritmo TCP Vegas vem sendo apontado em pesquisas recentes [34,35,37,49,52] como um mecanismo eficiente no que diz respeito ao uso de banda passante da rede, embora o mesmo tenha apresentado problemas de injustiça em termos de distribuição de banda passante, sob certas circunstâncias.

Dessa maneira, com base no estudo apresentado em [34,49], no qual o problema de "injustiça" do TCP Vegas pôde ser minimizado via ajustes nos parâmetros do mecanismo RED (*Random Early Detection*) [28] e, considerando-se que a rede DiffServ possui dois REDs internamente, espera-se neste estudo determinar a viabilidade dessa associação (DiffServ e TCP Vegas), incluindo-se também a interoperação do TCP Reno com o TCP Vegas nesse ambiente, e diferentes ajustes para os parâmetros α e β do TCP Vegas, os quais serão vistos mais adiante.

A arquitetura DiffServ é uma tecnologia relativamente recente e, em conseqüência disso, existem muitas propostas de padronização em estudo pela IETF. Dessa forma, embora se apresente neste estudo diferentes opções de implementação para a mesma, restringe-se aqui a uma análise de desempenho do modelo formado pelo mecanismo RIO (RED with In and Out) [44] e serviço AF (*Assured Forwarding*) [20], visto que estes representam o foco das

pesquisas atuais sobre a arquitetura DiffServ por serem, em princípio, as opções mais viáveis a esta tecnologia.

A proposta da arquitetura DiffServ é oferecer QoS de uma maneira flexível, no que diz respeito à escalabilidade, utilizando-se para isso a técnica de manuseio de tráfego em fluxos agregados. Esta arquitetura se fundamenta no princípio de se concentrar na borda da rede a complexidade das operações envolvidas no tratamento do tráfego, de modo que o núcleo da rede permaneça tão simples quanto possível. Assim, os roteadores de borda desta arquitetura são responsáveis por monitorar, classificar e marcar os pacotes a serem inseridos na rede.

Para isso, define-se um perfil de tráfego para cada fluxo de dados a ser admitido pela rede, de modo que os pacotes desses fluxos que estiverem de acordo com seus respectivos perfis são marcados como pacotes "em conformidade" e caso contrário como pacotes "não em conformidade". A classificação, mencionada no parágrafo anterior, refere-se à agregação dos fluxos individuais em "agregados de fluxos" os quais são associadas a um pequeno número de classes de serviço. No núcleo da rede os roteadores tratam os pacotes de forma diferenciada em função da classe de serviço à qual esses pertencem e da marcação realizada na borda da rede.

O mecanismo RIO é o responsável pelo descarte de pacotes de maneira diferenciada quando ocorre o congestionamento no interior da rede. Observa-se que este possui dois mecanismos RED, um para os pacotes em conformidade e outro para os não em conformidade; então, dois conjuntos de parâmetros de ajustes são necessários ao mesmo. Portanto, é de se esperar que os valores utilizados no ajuste destes parâmetros possam permitir um certo controle sobre o comportamento da rede. Porém, até o presente não consta da literatura estudada nada de expressivo a esse respeito; tal fato contribuiu para a motivação desta pesquisa.

Tratando-se do algoritmo TCP Vegas, este foi proposto por Brakmo e Peterson [36,43] como um aprimoramento do TCP Reno. O TCP Vegas pode interoperar com qualquer outra opção de algoritmo TCP válida e é ressaltado em [43] que o mesmo consegue obter entre 37% e 71% melhor vazão do que o Reno, retransmitir apenas 20% a 50% da quantidade de pacotes retransmitida pelo Reno e, ainda, utilizar mais eficientemente a banda passante da rede.

O TCP Vegas emprega um algoritmo diferente do empregado no TCP Reno, que causa perdas na rede para detectar congestionamentos na rede. Tal mecanismo monitora as variações sofridas pelos valores de RTT (*Round Trip Time*) dos segmentos enviados anteriormente pela conexão a fim de controlar o tamanho da janela de congestionamento (CWND) da fonte TCP. Dessa forma, se os valores de RTT aumentarem, o TCP Vegas deduz que a rede começou a ficar congestionada e, em conseqüência, o mesmo diminui o valor da CWND da conexão. Por outro lado, se os valores de RTT diminuirem, a CWND é incrementada em resposta ao entendimento de que a rede está livre de congestionamento. O algoritmo TCP Vegas define dois importantes parâmetros denominados de α e β, os quais determinam a faixa de variação dos RTTs (que correspondem a uma faixa de variação na taxa de fluxo) dentro da qual o tamanho da CWND permanece inalterada.

Conforme mencionado anteriormente, pesquisas recentes mostraram que o TCP Vegas embora apresente alta eficiência quanto à utilização da banda passante da rede, o mesmo não consegue prover justiça satisfatória às conexões que estejam compartilhando os recursos dessa rede. Nesse sentido, outro grupo de pesquisas atuais [33,52] determinou que os parâmetros α e β podem influenciar de forma decisiva a capacidade de justiça desse algoritmo.

Portanto, o estudo apresentado neste trabalho analisa, por meio de simulações, a influência que os parâmetros α e β do TCP Vegas e os parâmetros de ajuste do RED associado aos pacotes não em conformidade do algoritmo RIO têm sobre a justiça do serviço AF (*Assured Forwarding*) numa rede DiffServ. Com este estudo pretende-se verificar se os

resultados obtidos nas pesquisas mencionadas no parágrafo anterior, nas quais se considerou o TCP Vegas apenas numa rede convencional (baseada no serviço de "melhor esforço"), incluindo-se aí a interoperação do TCP Reno com o TCP Vegas, são válidos também para uma rede DiffServ. A agressividade do algoritmo UDP sobre os dois algoritmos TCP mencionados é também avaliada a fim de se ter uma análise mais realística, visto que o UDP se encontra presente na Internet atual de forma expressiva.

Então, com base nessas avaliações poder-se-á determinar a viabilidade da interoperação do TCP Vegas com a rede DiffServ, comparativamente ao TCP Reno, o que representa o objetivo principal desta pesquisa.

Para isso, empregam-se nas simulações três conjuntos de valores distintos para os referidos parâmetros do algoritmo RIO e quatro conjuntos de valores para os parâmetros α e β do TCP Vegas. Dessa maneira limita-se a avaliação a esses valores a fim de se viabilizar o estudo.

Assim, espera-se dar uma contribuição original à área, uma vez que se aborda aqui a interoperação entre dois importantes e atuais tópicos, DiffServ e TCP Vegas, até o presente não tratada na literatura pesquisada.

Apresenta-se, a seguir, a organização de cada um dos capítulos que formam o restante deste trabalho.

O capítulo II apresenta uma introdução ao tópico qualidade de serviço, de modo que os principais conceitos e mecanismos envolvidos com este assunto sejam vistos. O conteúdo deste capítulo possibilita ainda a compreensão do motivo pelo qual a QoS só começou a ser tratada de forma expressiva recentemente e as dificuldades encontradas nesse sentido.

Com relação ao capítulo III, este retrata as principais tecnologias de QoS envolvidas com a Internet atual, descrevendo o princípio de funcionamento de cada uma e a interoperação entre as mesmas, ressaltando a adequação de uso de cada uma destas. Por fim,

estas tecnologias são comparadas de uma maneira que possibilite uma visão geral sobre o aspecto QoS na Internet da atualidade.

A arquitetura DiffServ é estudada de forma detalhada no capítulo IV. Neste capítulo esta arquitetura é estudada em nível dos componentes que a formam com o propósito de mostrar as opções possíveis de serem implementadas nas redes DiffServ do futuro.

No capítulo V faz-se uma revisão sobre o princípio de funcionamento dos mecanismos RIO, TCP Reno e TCP Vegas e apresenta-se a proposta de contribuição deste trabalho de forma mais detalhada.

O Capítulo VI detalha todos os procedimentos que foram seguidos na realização das simulações, incluindo os valores utilizados para os parâmetros de cada algoritmo envolvido; os resultados obtidos são apresentados e avaliados. Com isso, este capítulo permite uma visão mais detalhada do funcionamento de cada um dos algoritmos de controle de tráfego aqui estudados, em relação aos parâmetros empregados para os algoritmos RIO e TCP Vegas mencionados anteriormente.

Finalmente, no capítulo VII apresentam-se as conclusões gerais relativas a este trabalho, incluindo algumas sugestões para futuras pesquisas relacionadas ao tema aqui estudado.

CAPÍTULO II

QUALIDADE DE SERVIÇO (QoS)

2.1 INTRODUÇÃO

Apesar da velocidade astronômica na qual as redes de computadores estão se desenvolvendo atualmente, ainda é notável o problema de congestionamentos nas mesmas e a tendência é que tal situação deve persistir ainda por um longo tempo, sobretudo, devido às aplicações multimídia emergentes que estão cada vez mais exigentes em relação à banda passante das redes.

A inconveniência do congestionamento consiste no fato de que o mesmo impossibilita ou ao menos dificulta o provimento de QoS por parte dos provedores de serviços, conhecidos por ISPs (*Internet Service Providers*), aos seus usuários. Por isso, muitos desses provedores estão investindo em enlaces mais velozes (usualmente fibras óticas) na tentativa de solucionar ou ao menos minimizar o problema da sobrecarga em suas redes.

Por outro lado, o que se nota é que enlaces mais velozes em todos os tipos de redes de computadores é algo que, seguramente, não deve ocorrer a curto prazo. Se por um lado os provedores estão adequando suas redes, o mesmo não acontece, de forma generalizada e no mesmo ritmo, nas redes corporativas e nas conexões dos usuários domésticos.

Nesse sentido, faz-se necessário estudar os aspectos mais relevantes relacionados com a QoS, bem como analisar a interação da mesma com outros conceitos aplicados às redes de computadores, a fim de que se possa prover métodos alternativos de se conseguir QoS tão

eficientes quanto possível utilizando-se enlaces que nem sempre possuem banda passante abundante.

2.2 DEFINIÇÃO DE QUALIDADE DE SERVIÇO

Definir QoS com precisão não é uma tarefa fácil, dada a variedade de interpretações existentes entre a comunidade científica da atualidade. Contudo, um ponto de convergência acerca do assunto é o fato de que as redes atuais não oferecem serviços compatíveis com a necessidade de seus usuários e assim surge a necessidade de se procurar formas alternativas que visem mudar esse quadro.

De uma forma simplificada, pode-se definir a QoS como sendo "o efeito coletivo do desempenho do serviço que determina a satisfação do usuário deste serviço" [59]. Assim, sob o ponto de vista dos parâmetros de desempenho de uma rede, os principais objetivos da QoS são: banda passante dedicada, retardo (latência) e variação de retardo (*jitter*) controlados e controle eficiente sobre a perda de pacotes. Os mecanismos de QoS se baseiam no princípio da necessidade da otimização dos recursos da rede em oposição à simples adição contínua de capacidade à mesma.

A questão é que a maioria das aplicações geram tráfego a taxas variáveis e geralmente necessitam que as redes sejam capazes de transportar tais dados na mesma cadência, embora normalmente as aplicações tolerem um certo nível de atraso e *jitter* na transmissão do fluxo de tráfego e algumas suportem inclusive perdas de dados controladas; porém, isso depende da aplicação. Numa situação ideal, os recursos da rede deveriam ser infinitos e, nesse caso, todos os fluxos de tráfegos poderiam ser transmitidos com atraso e perda iguais a zero. Contudo, como essa não é a situação real, em algumas partes da rede pode acontecer de os recursos não serem suficientes para garantir a QoS exigida.

As redes são formadas por meio da interligação de elementos que as constituem, tais como *switches*, roteadores, enlaces e *hosts*, os quais se comunicam através de interfaces

empregadas no encaminhamento do tráfego. Dessa forma, quando a taxa de fluxo de tráfego que chega à uma determinada interface de entrada é maior do que a taxa de fluxo de tráfego que sai da mesma (em direção a outro nó da rede), então pode ocorrer o congestionamento. Portanto, a capacidade de uma interface em enviar dados constitui um recurso fundamental das redes de computadores e, os mecanismos de QoS operam justamente sobre este recurso a fim de oferecer prioridade a alguns tráfegos fundamentais em detrimento de outros tráfegos de menor importância.

Dessa forma, para que se possa oferecer diferenciação na distribuição dos recursos da rede às suas correspondentes aplicações, torna-se necessário definir as "Classes de Serviços (ou classes de tráfegos) Diferenciadas" [8]. Aplicando-se esse conceito, o fluxo de tráfego ao chegar em um determinado nó da rede deve ser separado em diferentes tipos de fluxos via um processo denominado de classificação de pacotes. Estes fluxos, por sua vez, são armazenados em filas de armazenamento (*buffers*) distintas, localizadas nas interfaces de encaminhamento de cada elemento da rede, as quais serão atendidas segundo um algoritmo de prioridade pré-estabelecido, o qual determinará a taxa com que cada fluxo será enviado.

Portanto, um pressuposto básico no sentido de se oferecer QoS nas redes de computadores refere-se à implementação e configuração dos seguintes mecanismos nos elementos da rede:

a) Informação de classificação, a qual possibilita aos equipamentos de rede separar os tráfegos em fluxos distintos.
b) Filas de armazenamento e correspondentes algoritmos de atendimento, os quais manuseiam os fluxos de cada fila.

Uma forma alternativa de se prover garantia de QoS refere-se à "Reserva de Recursos" [57], na qual os recursos da rede são reservados nos nós intermediários (nós internos) da mesma de acordo com a solicitação de QoS das aplicações e segundo uma política de

gerenciamento de banda passante, via um protocolo de sinalização. Nesse caso, a diferenciação no tratamento do tráfego ocorre mediante a quantidade de recursos reservados previamente ao envio do tráfego e, assim, não há necessidade de disputa entre os tráfegos por preferência na ordem de envio dentro dos nós da rede.

Cabe ainda salientar que ambas as formas de provimento de QoS, "Classes de Serviços Diferenciadas" e "Reserva de Recursos", apresentam características próprias que as fazem adequadas ao uso em redes WANs de grande porte e em redes de acesso, respectivamente. Maiores detalhes serão apresentados no próximo capítulo.

2.3 FATORES QUE DIFICULTAM O EMPREGO DE QoS EM LARGA ESCALA

Tradicionalmente os provedores de serviço não têm dedicado atenção notória à questão de proverem QoS "inteligente" em seus enlaces, via disponibilização de serviços em classes diferenciadas. Antes, porém, os mesmos têm se concentrado apenas em oferecer banda passante suficiente à demanda de tráfego de seus usuários. Esse quadro se deve a duas razões principais [8].

A primeira diz respeito aos mecanismos de QoS empregados, os quais têm se mostrado ineficientes, uma vez que os mesmos ao tentarem prover QoS, involuntariamente introduzem perturbações na rede, culminando na degradação de desempenho da mesma quanto ao encaminhamento de pacotes. O que ocorre é que a complexidade dos algoritmos de manipulação de filas e de reordenamento de pacotes nos nós da rede, responsáveis pela diferenciação em classes de serviços baseada em fila, causam um atraso considerável ao tráfego, sobretudo nas redes de alta velocidade onde o volume de tráfego é acentuado.

A segunda refere-se à carência de mecanismos de medida de QoS confiáveis que assegurem que uma classe de tráfego esteja realmente recebendo tratamento prioritário sobre as demais classes. Esta característica é fundamental à QoS, visto que, se o provedor de serviço não for hábil para demonstrar ao usuário que seu tráfego está recebendo algum nível de

prioridade e, este por sua vez não puder constatar a autenticidade de tal informação, então, neste caso, a QoS não tem valor.

Em relação às redes corporativas e LANs em geral, a principal razão que justifica o desinteresse em se implementar mecanismos de QoS nesses ambientes é o fato de que normalmente essas redes contam com quantidades de banda passante suficientes (banda passante apresenta um custo relativamente baixo nesses cenários) para evitarem o congestionamento e, consequentemente, não enfrentam problemas relevantes de degradação de desempenho. E, ainda, possíveis ampliações de banda passante nessas redes representam uma operação relativamente simples, uma vez que as mesmas são normalmente de pequeno porte.

Portanto, nesses ambientes de dimensões limitadas, o sobre-dimensionamento da rede (banda passante abundante) pode representar uma solução com relação custo/benefício viável, visto que a implementação de mecanismos de QoS implica em custos e, quando dimensionados inadequadamente, podem também provocar degradação de desempenho da rede.

Por outro lado, nas redes corporativas formadas por diversas localidades geograficamente separadas e conectadas via uma WAN, poderá ser necessário oferecer prioridade aos tráfegos das aplicações críticas (aplicações que devem receber tratamento com prioridade máxima em função da importância de suas informações).

De qualquer modo, a situação atual indica que a preocupação com a QoS em larga escala está se tornando cada vez maior e, por isso, os mecanismos de QoS estão sendo extensivamente aprimorados no sentido de se mudar esse quadro, de pouca importância dispensada à QoS, com brevidade.

2.4 MEDIDA DA QUALIDADE DE SERVIÇO

A medida da QoS representa um dos assuntos de grande importância relacionado com as redes de computadores. Isso se deve, sobretudo, a duas razões. A primeira delas é a necessidade de se medir a QoS do tráfego que recebe tratamento prioritário (classes de tráfego) em relação ao serviço tradicional de melhor esforço da Internet e que, portanto, deve ser tarifado de forma diferenciada. A segunda diz respeito à necessidade que os provedores de serviços têm de medir continuamente a eficiência de seus respectivos enlaces, quanto ao provimento de QoS, a fim de que possam garantir um serviço confiável aos seus usuários.

A medição da QoS de uma rede pode ser vista como um caso especial de medida de desempenho em um ambiente de rede e, por isso, muitos dos avanços já obtidos nesta última podem ser aproveitados na medida da QoS. Resumidamente existem dois métodos que podem ser utilizados para se proceder à medida da QoS de uma rede [8]: o método passivo e o método ativo. O princípio dessas abordagens é o seguinte:

Método Passivo

Este método preconiza a medida da QoS por intermédio da observação da taxa de recebimento de pacotes nos equipamentos terminais (*hosts*), cujos valores possibilitam a dedução do estado em que se encontra a rede. Assim, a eficiência da QoS é deduzida com base somente na observação do comportamento da rede, sem que nenhuma interferência no funcionamento normal da mesma seja imposta.

O método passivo requer o perfeito conhecimento das características da aplicação que está gerando o tráfego sob observação, de modo que seja possível distinguir as alterações sofridas pelo tráfego que sejam provenientes da fonte daquelas causadas pela rede. Nesse sentido, se a medida da QoS for aplicada em apenas um dos lados da rede, então esse método pode não oferecer resultados satisfatórios.

Método Ativo

Esta abordagem se fundamenta no princípio da inserção de pacotes na rede e subseqüente retirada desses mesmos pacotes (ou de pacotes enviados por aplicações remotas em resposta a um pacote originariamente inserido), de forma que se possa proceder à análise e conseqüente determinação do estado da rede.

O comando PING do protocolo ICMP da Internet é um típico exemplo simplificado do método ativo de medida de QoS. Enviando-se seqüências de pacotes PING a intervalos constantes, uma estação de medida pode medir parâmetros tais como: acessibilidade, tempo de ciclo RTT (*Round Trip Time*) referente a um nó remoto e perda de pacotes esperada em relação ao tempo de ciclo de uma rota específica (*Round-Trip Path*). De posse desses dados e conhecendo-se (ou aproximando-se) o comportamento das filas de armazenamento pertencentes aos *switches*, pode-se estimar a banda passante disponível entre dois pontos da rede, bem como o nível de congestionamento correspondente [8].

O método ativo, portanto, possibilita a medida da QoS em apenas um dos lados da rede, sem apresentar o inconveniente citado anteriormente para o método passivo. Entretanto, esse método apresenta a desvantagem de inserir *overhead* na rede e isso às vezes pode mascarar o resultado da medida dependendo das condições em que se encontra a rede.

Resumindo, a medida da QoS em uma rede é um assunto relativamente recente e que ainda tem muito a se desenvolver, mas o fato é que tanto os ISPs quanto os clientes estão cada vez mais interessados pelo tema. O provedores por precisarem medir adequadamente a utilização dos seus recursos (*backbones*) a fim de que possam oferecer QoS sem, no entanto, tomar prejuízo em função de tarifação inadequada; e os clientes por necessitarem de comparativos entre mecanismos de medidas QoS, já que os mesmos serão cobrados com tarifas mais altas por esse serviço e, assim, precisam optar por serviços que ofereçam melhor custo/benefício.

2.5 PRINCÍPIOS E MECANISMOS BÁSICOS DE QoS

Atualmente diversos são os caminhos trilhados na tentativa de se prover QoS na Internet em todo o mundo. Porém, os princípios e mecanismos básicos mais relevantes que têm sido empregados com o objetivo de se oferecer diferenciação ao tráfego nesta rede, de uma forma geral, são abordados nesta sub-seção, os quais são classificados da seguinte forma:

- Caminhos alternativos
- Política de admissão de tráfego
- Algoritmos de enfileiramento
- Mecanismos de moldagem de tráfego
- Controle de congestionamento TCP (*hosts*)
- Controle de congestionamento RED, WRED e RIO (nós intermediários)

2.5.1 Caminhos Alternativos

A prática de se utilizar diversos caminhos físicos em uma rede é normalmente utilizada com o intuído de redundância e *backup*; entretanto, tal artifício pode também ser utilizado a fim de se implementar diferenciação de serviços, desde que os diferentes enlaces possuam características de QoS distintas. Por exemplo, o serviço de melhor esforço poderia ser transmitido por um enlace de baixa velocidade (ex. via satélite) enquanto que o tráfego de maior prioridade seria conduzido pelo caminho de maior velocidade (ex. enlace terrestre de alta velocidade). Obviamente essa abordagem é uma forma muito primitiva de se oferecer QoS e, em princípio, restrita a situações particulares.

Os dois maiores inconvenientes dessa prática na Internet, referem-se aos fatos de que o endereçamento nesta rede é realizado com base no endereço de destino contido no cabeçalho do pacote IP, de modo que a escolha da rota de envio em cada nó da rede é realizada em função do melhor caminho, segundo uma métrica pré-estabelecida, para cada rede de destino

e, nessas condições, um roteamento com base na QoS não é simples de ser implementado [19].

O segundo inconveniente diz respeito à possível assimetria que pode se formar entre o fluxo TCP direto e as mensagens de reconhecimento ou ACK, o que significa que ambos os tráfegos percorrem caminhos com atributos de QoS diferentes. Assim, como o transmissor TCP adapta a sua taxa de transmissão segundo a sinalização ACK, a qual carrega também o RTT do pacote envolvido, a QoS resultante pode sofrer influência do caminho de menor nível de QoS.

Apesar desses inconvenientes, esforços têm sido despendidos na tentativa de se prover mecanismos inteligentes nessas redes que sejam capazes de proceder ao roteamento dinâmico nos nós intermediários da rede levando-se em conta as exigências de QoS de cada pacote. Essa abordagem recebe o nome de "QoS-based routing" e é retratada em [19].

2.5.2 Política de Admissão de Tráfego

A definição mais genérica e básica possível para a política de admissão de tráfego é a que estabelece que a admissão de um tráfego na rede deve ser controlada por uma política de restrição em que, os tráfegos que estejam em conformidade com a política pré-estabelecida, entre usuário e rede, tenham acesso à rede e aos demais tráfegos seja negada essa possibilidade. Ou então, alguns tráfegos podem ter permissão de acesso sob certas condições e serem desabilitados quando tais condições mudarem.

E, numa terceira situação, a admissão à rede pode ser implementada via a utilização de um mecanismo de autenticação, na qual somente os tráfegos "autorizados" têm acesso à rede. Finalmente, o que deve ser ressaltado é o fato de que é imprescindível que os fluxos individuais possam ser distinguidos uns dos outros, por exemplo por meio de protocolos específicos, tipo de fonte ou destino, etc. Caso contrário, uma política de controle de admissão de tráfego não pode ser implementada.

O controle sobre o tráfego a ser admitido pela rede é algo fundamental em qualquer implementação de QoS, uma vez que a falta desse controle efetivo inviabiliza o controle de congestionamento da rede. Isso ocorre porque, em tal situação, a tarefa de controle sobre o tráfego na rede deve ser exercida exclusivamente pelos mecanismos de prevenção e controle de congestionamento da rede. Entretanto, essa não é uma situação ideal, dada a possibilidade de que algumas fontes de tráfego podem introduzir congestionamento severo dentro da rede e dessa forma inviabilizar a eficiência da QoS nesse sistema, que normalmente é projetado para um limite máximo de tráfego.

2.5.3 Algoritmos de Enfileiramento

Os algoritmos de enfileiramento constituem uma das áreas de grande interesse no estudo da QoS, principalmente porque estes facilitam a implementação de mecanismos de diferenciação de QoS. A capacidade das filas de armazenamento, normalmente define a quantidade de pacotes a ser descartada na rede em situações de congestionamentos e, portanto, influenciam diretamente a QoS a ser oferecida pelo sistema.

De uma forma generalizada, os enfileiramentos podem ser classificados em dois tipos básicos, o enfileiramento do tipo FIFO e o enfileiramento com prioridades (nas suas diversas formas de implementação). O primeiro oferece um tratamento com o mesmo nível de prioridade a todos os fluxos, enquanto que o segundo oferece tratamento diferenciado a classes de tráfegos distintas e, portanto, possibilita à rede proporcionar algum nível de QoS aos seus usuários.

O enfileiramento do tipo FIFO não oferece possibilidade de se prover QoS satisfatória, este é o caso do serviço de melhor esforço tradicional da Internet (na sua forma mais simplificada). Dessa forma, a fim de se obter níveis de serviços diferenciados com esse tipo de enfileiramento, torna-se necessário a introdução de mecanismos adicionais.

Uma possível alteração nesse sistema seria a divisão do tráfego em algumas categorias com reserva de recursos proporcional para essas categorias por meio de uma estrutura de reserva pré-estabelecida. Entretanto, ainda que esta seja uma alternativa à diferenciação de prioridade ao tráfego, a mesma não oferece bom desempenho, primordialmente porque o tempo de varredura total da fila continua constante e, assim, ao se reduzir o tempo de enfileiramento de uma categoria se estará automaticamente aumentado o tempo das demais.

Visto que a técnica FIFO não oferece flexibilidade na diferenciação de QoS, então surgiu a necessidade de se implementar as filas com prioridades simples. A mudança básica em relação à estrutura FIFO refere-se à criação de filas distintas, com respectivo nível de prioridade, para cada interface. Essa abordagem proporciona retardo mínimo ao tráfego de maior prioridade, mas os demais tráfegos, com prioridades menores, podem sofrer com retardo excessivo e até mesmo descarte, caso a fila prioritária permaneça ocupada por muito tempo. Portanto, para assegurar que todos os tráfegos recebam um nível mínimo de serviço, torna-se necessário que a rede realize o controle de admissão de tráfego, conforme citado anteriormente, ou ainda que os mecanismos que realizam a moldagem do tráfego (vistos na próxima sub-seção) sejam ajustados para tal fim.

Na tentativa de se eliminar a desvantagem do enfileiramento com prioridades simples citado, quanto à possibilidade de uma fonte prioritária monopolizar o enlace, alguns aprimoramentos foram acrescidos a essa forma de enfileiramento resultando em outros algoritmos com características peculiares para a solução de problemas encontrados nos algoritmos antecessores. Os mais relevantes atualmente são: *Priority Round Robin* (PRR), *Weighted Round Robin* (WRR), *Weighted Fair Queue* (WFQ) e *Priority Weighted Fair Queue* (PWFQ). O princípio de funcionamento de cada um desses algoritmos é apresentado na seção 4.3.5 do capítulo IV, onde os mesmos são tratados mais especificamente como mecanismos de esvaziamento de fila.

2.5.4 Mecanismos de Moldagem de Tráfego

Moldagem de tráfego refere-se à ação tomada pelos mecanismos na borda da rede a fim de controlar o volume de tráfego enviado à mesma e assim determinar a taxa com a qual tal tráfego será transmitido para dentro da rede. Tais mecanismos podem também controlar os tráfegos de forma diferenciada desde que haja possibilidade de identificação de classes de tráfegos distintas. Nesse sentido, infere-se que a política de admissão de tráfego pode fazer uso desses mecanismos de modo a adaptar tráfegos que porventura estejam fora do padrão de admissão (tráfegos não em conformidade).

Existe predominantemente dois métodos utilizados na moldagem de tráfego: o *Leaky-Bucket* e o *Token-Bucket*. Ambos possuem propriedades distintas e são utilizados para fins diferenciados, embora exista a possibilidade de uso em comum dos mesmos. A seguir apresenta-se uma explanação a respeito do princípio de funcionamento de cada um desses métodos.

Leaky-Bucket

O *Leaky-Bucket* é utilizado para controle da taxa do tráfego que é enviado à rede. Esse mecanismo possibilita a moldagem do tráfego em rajadas ou surtivo (*bursty traffic*), de modo que os nós da rede recebam um tráfego com taxa constante e, consequentemente, menos susceptível a perdas, uma vez que o mesmo torna-se previsível e controlável. Esse algoritmo foi desenvolvido para uso nas redes ATM, mas também está sendo usado nas redes IP. Na figura 2.1 é feita uma analogia desse mecanismo com um "balde furado" mostrando a semelhança de funcionamento entre os mesmos, a qual originou o nome desse mecanismo.

O funcionamento do *Leaky-Bucket* se baseia no princípio do enfileiramento FIFO, em que o tráfego ao ser recebido pela fila é armazenado, desde que haja espaço, e independentemente da sua taxa de recepção, o tráfego será sempre enviado numa taxa constante. Assim, quando rajadas forem transmitidas à rede, as mesmas serão armazenadas

nas filas e em seguida enviadas com taxa fixa na saída da fila. Se a quantidade de tráfego que chega para ser armazenada for maior do que a capacidade de armazenamento disponível na fila FIFO naquele momento, então o tráfego excedente será descartado.

O *Leaky-Bucket* representa um mecanismo de moldagem de tráfego de grande importância no controle de reserva de recursos no núcleo da rede. No entanto, o fato deste não possibilitar a moldagem de tráfego para as fontes com taxas de transmissão variáveis (rajadas), representa uma grande deficiência do mesmo. Também pode ser visto como uma deficiência o fato do mesmo não aproveitar eficientemente os recursos da rede em circunstâncias em que o fluxo dentro da rede é baixo (oferecendo banda passante disponível), visto que a taxa de fluxo na saída do *Leaky-Bucket* é sempre fixa.

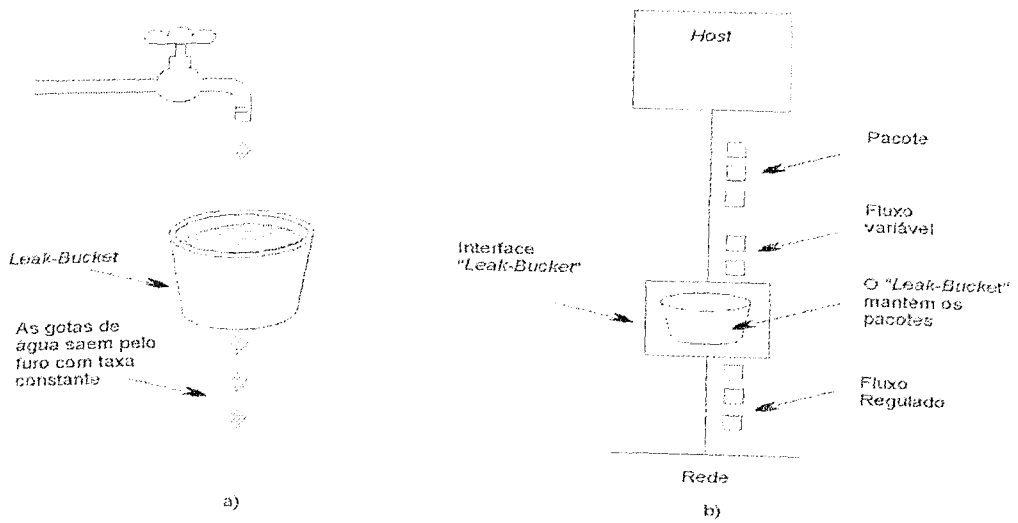

Figura 2.1 – Ilustração do mecanismo de moldagem de tráfego *Leak-Bucket*:
a) "*Leak-Bucket*" com água
b) "*Leak-Bucket*" com pacotes

Token-Bucket

O *Token-Bucket* é outro método utilizado para moldagem de tráfego e controle de taxa de fluxo na borda da rede. A diferença deste método em relação ao *Leaky-Bucket* é que este

último é preenchido somente com o tráfego, que é transmitido numa taxa fixa na sua saída, enquanto que o *Token-Bucket* transporta também permissões (*tokens*) entre o tráfego.

Com o *Token Bucket* (figura 2.2) só há transmissão quando há permissões no "balde" (*bucket*). O balde recebe permissões enviadas periodicamente por um relógio do sistema. O mecanismo *Token-Bucket* admite ainda que tráfegos em rajadas possam ser transmitidos até um determinado limite pré-ajustado e, portanto, provê melhor aproveitamento dos recursos da rede.

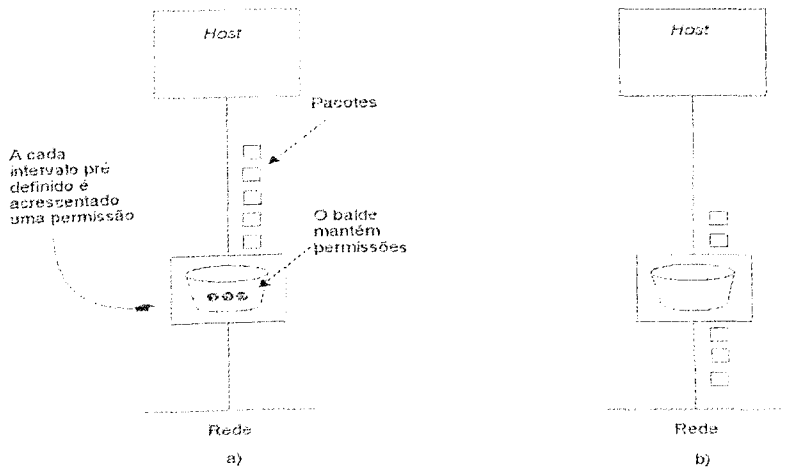

Figura 2.2 – Algoritmo *Token-Bucket*. a) antes. b) depois

O *Token-Bucket* trabalha com permissões que representam quantidades de bytes (cada permissão pode representar um ou mais bytes, isso depende da implementação). Um determinado tráfego só pode ser transmitido se houver uma quantidade mínima específica de permissões no balde, caso contrário os pacotes desse tráfego não podem ser enviados e, a cada pacote enviado (essa quantidade depende da aplicação) uma permissão é retirada do balde. Dessa forma, o tráfego em rajadas, até o limite da taxa de pico de rajada, pode ser transmitido, desde que haja quantidade de permissões suficientes no balde e a duração da rajada tenha sido configurada adequadamente.

2.5.5 Controle de Congestionamento TCP (*Hosts*)

Como grande parte do tráfego na Internet atual é transportado sobre o protocolo TCP e este possui um algoritmo de controle de tráfego que visa evitar congestionamentos dentro da rede, então, torna-se muito importante a análise desse algoritmo sob o ponto de vista da QoS.

O princípio desse algoritmo é manter a carga na rede tão estável quanto possível, de modo que uma situação ideal seja aproximada, na qual as fontes deveriam enviar dados para dentro da rede praticamente no mesmo instante em que os *hosts* de destino estivessem retirando dados da rede. A fim de conseguir realizar tal tarefa, o controle de congestionamento utilizado pelo TCP emprega dois algoritmos básicos, denominados de *Slow Start* e *Congestion Avoidance* [58]. Apresenta-se a seguir o funcionamento destes algoritmos.

Slow Start

Nas primeiras implementações do algoritmo TCP, a fonte TCP podia começar uma transmissão enviando múltiplos segmentos para dentro da rede até o limite estabelecido pela janela de recepção do receptor TCP. Embora este princípio funcione satisfatoriamente quando os dois *hosts* envolvidos estão na mesma LAN, o mesmo pode não ocorrer nos casos em que se tem roteadores e enlaces lentos entre estes *hosts*. O problema é que nestas circunstâncias alguns roteadores intermediários precisam enfileirar os pacotes, e pode ocorrer dos mesmos ficarem sem capacidade de armazenamento por excesso de pacotes a eles submetidos, os quais serão descartados; essa situação pode reduzir a vazão da conexão TCP de forma drástica.

O algoritmo *Slow Start* surgiu exatamente para evitar tal deficiência. Este algoritmo se baseia no princípio de que a taxa com a qual os pacotes devem ser injetados na rede deve corresponder à taxa na qual os sinais de reconhecimento ACK são recebidos da outra extremidade da rede.

Este mecanismo define uma janela para o transmissor TCP, denominada janela de congestionamento e referenciada por CWND; quando uma nova conexão é estabelecida esta janela é ajustada em um segmento (janela inicial) e a cada ACK recebido a mesma é incrementada de um segmento. Assim, a quantidade de bytes enviados pelo transmissor é limitada pelo menor tamanho observado na comparação entre a janela de congestionamento e a janela de recepção. A janela de congestionamento está associada ao controle de fluxo no transmissor, o qual se baseia na avaliação do congestionamento observado na rede; no receptor este controle está relacionado com a janela de recepção e se baseia na quantidade de memória (*buffer*) disponível para a conexão.

Dessa maneira, o transmissor começa transmitindo um segmento e espera pelo seu ACK correspondente; quando este é recebido, a CWND é incrementada de um para dois segmentos e, assim, dois segmentos podem ser enviados. Após o recebimento do ACK de ambos os segmentos a CWND é aumentada para quatro. Tal procedimento faz com que a janela de congestionamento, dependendo dos ACKs recebidos, aumente de forma exponencial.

Esse ritmo continua até que a capacidade da rede seja alcançada, quando então os pacotes começam a ser descartados pelos roteadores no interior da rede. Isso indica ao transmissor que o tamanho da sua janela alcançou um valor muito alto.

Congestion Avoidance

Este algoritmo trata da prevenção de congestionamento na rede, que pode ocorrer quando os dados provenientes de uma rede com grande capacidade de banda passante (LAN rápida) são direcionados a uma rede de menor capacidade (WAN lenta), ou ainda quando múltiplos fluxos de dados chegam a um determinado roteador cuja capacidade de envio é menor do que a soma dos fluxos individuais a ele submetido.

O algoritmo *Congestion Avoidance* se baseia no princípio de que a perda de pacotes por danos causados pelo meio de transmissão é muito pequena (muito menor do que 1%) e, portanto, a perda de um pacote sinaliza o congestionamento em algum ponto da rede entre o transmissor e o receptor. Nesse sentido, este mecanismo emprega duas formas de dectecção de perda de pacotes: a ocorrência de *timeout* e o recebimento de ACK duplicados.

Embora os algoritmos *Congestion Avoidance* e *Slow Start* sejam independentes, quando ocorre o congestionamento, o TCP deve diminuir a taxa de transmissão de pacotes enviados pela sua fonte à rede e invocar o *Slow Start* de modo a reiniciar o processo de transmissão. Portanto, na prática eles são implementados juntos. Dessa forma, além da janela de congestionamento, CWND, estes mecanismos requerem que uma outra variável de limiar do *Slow Start*, SSTHRESH (*Slow Start Threshold*), seja definida para cada conexão. Assim, o funcionamento combinado destes algoritmos ocorre da seguinte forma:

1 O estabelecimento de uma determinada conexão ajusta a CWND em um segmento e o SSTHRESH em 65535 bytes.

2 A rotina de saída do TCP jamais envia mais dados do que o especificado pelo menor tamanho observado na comparação entre a janela de congestionamento e a janela de recepção.

3 Quando ocorre o congestionamento (indicado por um *timeout* ou pela recepção de um ACK duplicado) o SSTHRESH é ajustado com o valor igual ao tamanho da CWND corrente dividido por dois. Adicionalmente, se o congestionamento for indicado por um *timeout*, a CWND é ajustada em um segmento.

4 Quando novos dados são reconhecidos pelo outro lado da conexão, a CWND é aumentada, mas a maneira como a mesma é incrementada depende do fato do TCP estar executando a fase correspondente ao *Slow Start* ou ao *Congestion Avoidance*.

Então, se a CWND contiver um valor menor ou igual ao do SSTHRESH, o TCP está na fase *Slow Start*, caso contrário a fase *Congestion Avoidance* está sendo executada. A fase *Slow Start* continua até que a CWND ultrapasse o valor de SSTHRESH.

Conforme mencionado anteriormente, na fase *Slow Start* a CWND é incrementada de forma exponencial; na fase *Congestion Avoidance* esse crescimento se dá de forma linear, em que a CWND é atualizada da forma: segsize*segsize/CWND, a cada ACK recebido. Onde segsize representa o tamanho do segmento e a CWND é medida em bytes. Na fase *Congestion Avoidance*, o incremento da CWND deve ser limitado a no máximo um segmento por RTT, independentemente da quantidade de ACKs recebidos dentro desse RTT. Diferentemente, portanto, do *Slow Start* que incrementa a CWND em função da quantidade de ACKs recebidos em cada RTT.

Apresenta-se a seguir um exemplo [32] a fim de melhor elucidar o funcionamento dos mecanismos de congestionamento citados (figura 2.3). Neste exemplo, o tamanho de um segmento é definido como sendo de 1 KB (1024 bytes). Inicialmente o SSTHRESH é de 64 KB e após ocorrer um *timeout*, o SSTHRESH é ajustado em 32 KB (64/2 KB) e a CWND em 1 KB. A CWND cresce exponencialmente (*Slow Start*) até alcançar o limiar (32 KB), quando então começa a fase de crescimento linear (*Congestion Avoidance*).

Na transmissão de número 13 ocorre o *timeout*. Então, o SSTHRESH é ajustado para a metade do valor da janela corrente e, por isso, recebe o valor 20 KB (40/2 KB), a CWND é ajustada novamente em 1 KB e a fase *Slow Start* é reiniciada. Desse modo, o crescimento exponencial da CWND continua até a transmissão de número 18 quando então a CWND alcança o SSTHRESH e é iniciada a fase linear novamente. Nessa situação, se não ocorrer *timeout*, a janela de congestionamento continuará crescendo até o tamanho da janela do

receptor quando então a mesma se manterá fixa desde que não ocorra *timeout*, nem o receptor mude o tamanho de sua janela de recepção.

Tais algoritmos funcionam de forma mais eficiente quando a perda de pacotes é detectada antes que as filas de armazenamento da rede fiquem totalmente preenchidas. Por isso, é utilizada a prática de envio de pacotes ACK duplicados de modo que a fonte tenha condição de realizar uma rápida retransmissão e deixar a janela de congestionamento no nível do limiar (CWND = SSTHRESH). Essa técnica de atuação deu origem a novas implementações TCP, conforme será visto no capítulo IV.

Na falta desses procedimentos, se ocorrer o preenchimento total da fila (*queue exhaustion*), a sessão TCP pára de receber sinalização ACK e a retransmissão irá ocorrer somente quando acontecer o *timeout*, quando então a janela de congestionamento é ajustada em um (CWND = 1) e o mecanismo *Slow Start* é reiniciado.

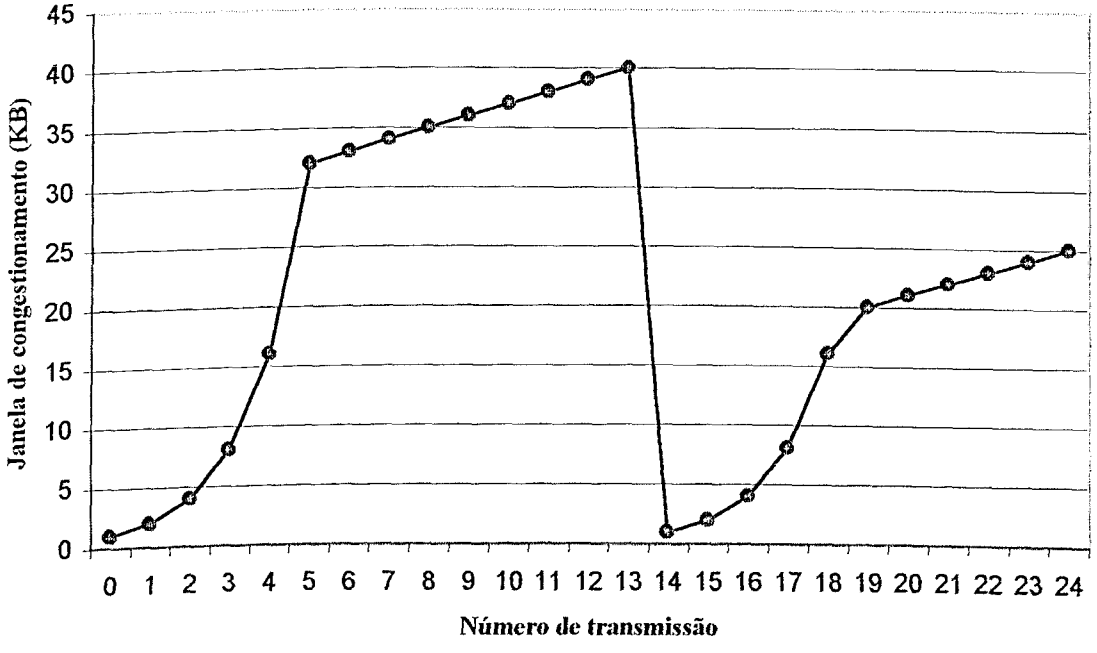

Fig. 2.3 – Controle de congestionamento TCP com os algoritmos *Slow Start* e *Congestion Avoidance*.

No que diz respeito às redes IP de grande capacidade, um dos maiores problemas encontrados nessas redes são os possíveis "enlaces de estrangulamento" (*bottleneck links*), i.e.

enlaces com capacidade de vazão insuficiente para a demanda de tráfego, que podem existir e que causam congestionamento. Se esse congestionamento não for controlado eficientemente, o desempenho da rede pode sofrer degradação acentuada, chegando ao limite do colapso total. Quando um alto volume de tráfego TCP está ativo ao mesmo tempo e ocorre congestionamento num determinado enlace de estrangulamento da rede, cada fluxo de tráfego pode sofrer perda de dados aproximadamente no mesmo instante, caracterizando um fenômeno conhecido por "Sincronização Global".

A Sincronização Global ocorre quando uma grande quantidade de fluxos diminuem suas respectivas taxas de transmissão e entram na fase *Slow Start* aproximadamente ao mesmo tempo (sincronizadas). Cada fonte TCP detecta a perda de pacotes e reage de forma adequada, comutando para a fase *Slow Start,* diminuindo sua respectiva janela de transmissão, parando por um momento e então tentando retransmitir novamente. Se o congestionamento ainda persistir o processo se repete, resultando numa severa degradação de desempenho da rede [3].

Por este motivo, o congestionamento não controlado representa um sério problema para a rede, uma vez que o comportamento da mesma se torna imprevisível, os *buffers* ficam cheios, os pacotes são descartados e, como conseqüência, ocorre um grande número de retransmissões, o que, por fim, pode causar uma situação de colapso completo no congestionamento. Portanto, a Sincronização Global deve ser evitada a todo custo.

2.5.6 Controle de Congestionamento RED, WRED E RIO (nós intermediários)

O mecanismo RED (*Random Early Detection*) [28] opera no sentido de evitar que ocorra a situação em que todos os fluxos TCP sofram congestionamento ao mesmo tempo com a conseqüente perda de dados, ou seja, evita que ocorra a Sincronização Global na rede. Esse mecanismo atua no descarte aleatório de pacotes provenientes de fluxos escolhidos arbitrariamente com o objetivo de minimizar assim a possibilidade de colapso na rede. O RED também tenta forçar a sinalização utilizada pelo TCP, que se baseia no envio de ACK em

duplicata por parte do *host* receptor ao transmissor, de modo a evitar a ocorrência de *timeout* e, consequentemente, prover uma forma menos drástica de diminuição do fluxo TCP.

Para isso, o RED monitora a quantidade média de bytes na fila de armazenamento de modo que quando a mesma ultrapassa um determinado limiar ou nível pré-ajustado (normalmente escolhido pelo administrador da rede), este mecanismo começa a selecionar fluxos TCP aleatóriamente para descartar pacotes e assim sinalizar aos *hosts* transmissores para diminuírem suas respectivas taxas de transmissão. Dessa forma, quanto maior for a quantidade de dados a chegar na fila, maior será também o número de descartes e, consequentemente, mais fontes reduzirão suas respectivas taxas, resultando, portanto, numa prevenção de congestionamento mais eficiente.

Quando comparado com os algoritmos de enfileiramento com prioridade, em todas as suas variações, a abordagem RED oferece a vantagem de exigir muito menos poder computacional dos elementos da rede, uma vez que a mesma não necessita realizar re-ordenamento tampouco gerenciamento do enfileiramento.

O fato do RED descartar pacotes aleatoriamente implica que este oferece o mesmo nível de prioridade a todos os tráfegos. Tal característica confere a esse algoritmo o controle eqüitativo sobre todas as fontes de tráfego e assim uma contribuição indiscutível à qualidade de serviço. Entretanto, a fim de que se possa oferecer níveis diferenciados de QoS, faz-se necessário implementar alguma forma de diferenciação nesse algoritmo para que o mesmo possa descartar algumas classes de tráfego prioritariamente sobre as demais.

A esta modificação no comportamento do RED dá-se o nome de WRED (*Weighted* RED). Obviamente, a implementação do WRED exige que alguma forma de marcação de tráfego seja realizada na borda da rede a fim de que os algoritmos WRED nos nós intermediários da rede possam proceder à identificação e correspondente processamento dos

pacotes a eles destinados. Essa marcação pode ser realizada através do uso dos bits ToS (*Type of Service*) do cabeçalho IP, por exemplo.

Uma forma de implementação WRED de grande destaque nas redes atuais é o algoritmo RIO (RED *with IN and OUT*) [44]. Esse algoritmo se baseia no algoritmo RED e utiliza dois conjuntos de parâmetros para dois tipos de pacotes distintos, os pacotes "em conformidade" e "não em conformidade" (pacotes *in-profile* e *out-of-profile*, respectivamente). O algoritmo RIO pressupõe que os mecanismos de policiamento na borda da rede (com base na política de admissão de tráfego) marquem os pacotes que estão entrando na mesma como um dos dois tipos de pacotes mencionados.

Para a realização de tal tarefa, define-se um perfil de serviço, ou perfil de tráfego, para cada usuário (em função da utilização de banda passante solicitada pelo usuário) de forma que os pacotes que estiverem de acordo com tal perfil são marcados como pacotes em conformidade e os demais como pacotes não em conformidade. No núcleo da rede, esses dois tipos de pacotes são tratados com prioridades diferentes pelo mecanismo RIO, de forma que os pacotes não em conformidade são descartados prioritariamente. O algoritmo RIO é parte integrante das redes de "Serviços Diferenciados" e será tratado no capítulo V.

2.6 CONCLUSÕES

A QoS tem por objetivo principal manter os parâmetros de desempenho da rede: banda passante, retardo, *jitter* e perda de pacotes dentro de limites aceitáveis pelas aplicações; por meio do uso eficiente dos recursos da rede ao invés da simples adição de banda passante à mesma.

Os mecanismos básicos empregados pelas redes atuais com o objetivo de prover QoS aos seus usuários, se fundamentam no princípio da proteção da rede contra possíveis colapsos por sobrecarga excessiva (*overload*) sobre a mesma. Para isso, tais mecanismos devem limitar (moldar) as características do tráfego que entra na rede e também atuar sobre o tráfego no

interior da mesma com o propósito de garantir um serviço fim-a-fim aceitável nesse ambiente. Nesse sentido, o controle final exercido sobre a QoS ocorre tanto nos *hosts* como no interior da rede.

A implementação de tais mecanismos otimiza a utilização dos recursos da rede, uma vez que os mesmos operam no sentido de manter a estabilidade dentro da mesma e, por conseguinte, menor quantidade de descarte de pacotes é esperada. Dessa forma, o conceito de QoS possibilita vantagens não somente ao usuário, mas também aos ISPs.

Em relação às pesquisas afins, um dos grandes desafios relacionados à QoS é o desenvolvimento de métodos eficazes para se medir a sua eficácia, visto que até o presente não se tem registro de um mecanismo eficiente e geral neste sentido. Isso ocorre, sobretudo, em decorrência da elevada complexidade envolvida neste assunto.

Por fim, cabe salientar que QoS trata-se de uma abordagem relativamente recente (ao menos o interesse pela mesma) e que, portanto, está apenas começando a ser explorada, tanto em termos de pesquisas quanto de comercialização. Todavia, nota-se nesses setores uma preocupação geral com o tema nos últimos anos, a prova é que os mecanismos QoS não estão mais restritos aos centros de pesquisas, mas também sendo incorporados aos equipamentos de muitos fabricantes da área. Essa tendência certamente implicará em novas aplicações compatíveis com os conceitos de QoS num futuro próximo.

CAPÍTULO III

TECNOLOGIAS DE QUALIDADE DE SERVIÇO NA INTERNET

3.1 INTRODUÇÃO

Este capítulo aborda as principais tecnologias que têm sido desenvolvidas nos últimos anos com o propósito de se oferecer QoS na Internet. Conforme mencionado anteriormente, a busca por QoS efetiva (fim-a-fim) na Internet tem se acentuado ultimamente, em função principalmente das exigências impostas pelas aplicações emergentes (VoIP, multimídia, etc) que tornam o mecanismo de melhor esforço, tradicionalmente empregado nesta rede, inadequado a essas novas aplicações. Por isso, muitos esforços têm sido despendidos recentemente pela comunidade Internet, mediante o desenvolvimento de novas tecnologias, na tentativa de se viabilizar a operação de tais aplicações [2,3,4,5,8].

As principais tecnologias relacionadas com a QoS na Internet atualmente são: o ATM (*Asynchronous Transfer Mode*), o IntServ (*Integrated Services*) associado ao RSVP (*Resource ReSerVation Protocol*), o DiffServ (*Differentiated Services*) e o MPLS (*Multiprotocol Label Switching*). O ATM permite operações com taxas elevadas de transmissão e oferece suporte à garantia de QoS fim-a-fim; entretanto, em função, principalmente de seu alto custo, o uso dessa tecnologia tem se limitado às redes *backbones*, onde, normalmente, os seus mecanismos de QoS nativos não são explorados na sua totalidade.

O modelo IntServ, juntamente com o protocolo RSVP, tem como característica principal possibilitar a reserva de recursos nos nós intermediários da rede com o objetivo de

oferecer QoS ao fluxo de dados com certa flexibilidade, quanto a possibilidade de escolha da quantidade de recurso a ser reservada. O modelo DiffServ agrega os fluxos que entram na rede e os associa a classes de tráfego diferenciadas, permitindo, assim, atribuir QoS e, por conseguinte, tarifação diferenciada a esses fluxos agregados. O MPLS, por sua vez, é uma tecnologia de encaminhamento (*Forwarding*) que possibilita roteamento simplificado e escalabilidade na rede e, por isso, simplifica a operação do IP sobre o ATM.

Dessa forma, pretende-se neste capítulo fornecer uma visão geral do "estado da arte" em QoS na Internet atual, mediante a análise, sob o aspecto QoS, de cada uma das abordagens mencionadas. Para tanto, estas tecnologias são comparadas quanto à adequação de uso, ressaltando-se os pontos mais relevantes a serem considerados na interoperação das mesmas.

3.2 QoS NAS REDES ATM

O ATM [1,2,3] caracteriza-se como uma tecnologia desenvolvida com base nos fundamentos da QoS, visto que este possui recursos de QoS nativos, os quais possibilitam efetivamente a garantia de QoS fim-a-fim nessas redes ATM.

Em princípio, esta tecnologia foi desenvolvida para operar em todos os tipos de rede, desde as LANs até as WANs de longas distâncias; mas para que isso se efetivasse, seria necessária a criação de novas aplicações direcionada à tecnologia ATM. No entanto, o alto custo do ATM tem desencorajado o desenvolvimento expressivo de tais aplicações e, o que se nota hoje é a concentração do ATM nas redes *backbones*, onde apenas a sua capacidade de transmissão a taxas elevadas e com baixas perdas, tem sido aproveitada; embora, haja disponível, conforme citado, todo um potencial para o manuseio eficiente da QoS [6].

Por outro lado, muitas pesquisas estão sendo desenvolvidas no sentido de se aproveitar os recursos de QoS nativos do ATM por intermédio do mapeamento de outras tecnologias de QoS para o ATM. Nesse sentido, pode-se dizer que o sucesso das redes ATM está associado à possibilidade das mesmas "interoperarem" eficientemente com as demais tecnologias em uso

atualmente na Internet, tais como Intserv, DiffServ e MPLS [47]. Por isso, é muito importante conhecer-se os mecanismos utilizados pelas redes ATM para provimento de QoS, bem como os fatores que dificultam a sua obtenção, de modo que se possa analisar corretamente a sua interoperação com as demais tecnologias citadas.

3.2.1 Categorias de Serviço ATM e suas aplicações

As redes ATM podem operar, atualmente, com até seis categorias de serviço, conforme a especificação af-tm-0121.000 do ATM Forum [1]. O ITU-T também define categorias similares às categorias de serviço especificadas pelo ATM Forum, às quais é atribuído o nome de capacidades de transferência ATM (ATM *transfer capabilities*), descritas na recomendação I.371 do ITU-T [59].

Cada uma das categorias de serviço apresenta suas características de funcionamento apropriadas a aplicações específicas. Isto viabiliza, sobretudo, o provimento de qualidade de serviço, mediante o isolamento de tráfegos distintos em filas separadas e, ainda, possibilita maior simplificação no gerenciamento da rede (o tráfego tem que se adequar a uma das categorias de serviço oferecidas). A atribuição da categoria de serviço ao tráfego que entra na rede é realizada pelo equipamento de borda (*Edge Device*), o qual interliga o usuário à rede. E, essas categorias de serviço ou simplesmente serviços ATM são divididas da seguinte forma:

- **CBR** *Constant Bit Rate*
- **rt-VBR** *Real-Time Variable Bit Rate*
- **nrt-VBR** *Non-Real-Time Variable Bit Rate*
- **UBR** *Unspecified Bit Rate*
- **ABR** *Available Bit Rate*
- **GFR** *Guaranteed Frame Rate*

Estas categorias de serviço apresentam características de tráfego e exigências de QoS associadas ao comportamento da rede. Funções tais como: roteamento, Controle de Admissão de Conexões (CAC) e reserva de recursos são estruturadas de forma diferenciada para cada uma dessas categorias de serviço. A tabela 3.1 apresenta os atributos suportados por cada uma das categorias de serviço. Pode-se observar a existência de atributos distintos para os parâmetros de tráfego e parâmetros de QoS.

Parâmetros de tráfego

- **PCR** *Peak Cell Rate* (Taxa de Pico de Células)
- **CDVT** *Cell Delay Variation Tolerance* (Tolerância na Variação de Retardo de Células)
- **SCR** *Sustainable Cell Rate* (Taxa Média de Células)
- **MBS** *Maximum Burst Size* (Duração Máxima de Rajada)
- **MFS** *Maximum Frame Size* (Tamanho Máximo de Quadro)
- **MCR** *Minimum Cell Rate* (Taxa Mínima de Células)

Parâmetros de QoS

- **CDVpp** *peak-to-peak Cell Delay Variation* (Variação pico-a-pico do Retardo de Células)
- **CTDmáx** *maximum Cell Transfer delay* (Retardo Máximo de Transferência de Células)
- **CLR** *Cell Loss Ratio* (Taxa de Perda de Células)

Os parâmetros de tráfego definem as características gerais do tráfego que está sendo transmitido, enquanto que os parâmetros de QoS definem o comportamento que o tráfego deve seguir a fim de se obter a QoS desejada.

Atributos	Categorias de Serviço ATM					
	CBR	**rt-VBR**	**nrt-VBR**	**UBR**	**ABR**	**GFR**
Parâmetros de tráfego						
PCR e CDVT	Especificado					
SCR e CDVT	n/a	Especificado		n/a		Apenas CDVT para o MCR
MBS	n/a	Especificado		n/a		Especificado para o MCR
MFS	Sem especificação					Especificado
MCR	n/a				Especificado	
Parâmetros de QoS						
CDVpp	Especificado		Sem especificação			
CTDmáx	Especificado		Sem especificação			
CLR	Especificado			Sem especificação	Baixo para os dados em conformidade (c/ c.t.)	
Outros atributos						
Feedback	Sem especificação				Especificado	Sem especificação

n/a = não aplicável c.t. = contrato de tráfego

Tabela 3.1 – Categorias de serviço.

Faz-se a seguir um estudo das principais características de cada uma das categorias de serviço citadas, com base nos três parâmetros de QoS mencionados anteriormente. Os mecanismos utilizados por cada uma dessas categorias de serviço visando a obtenção de QoS podem ser encontrados em [2].

3.2.1.1 Categoria de serviço CBR

Aplicada às conexões que exigem disponibilidade de banda passante fixa durante todo o tempo de existência da conexão. A quantidade de banda passante é caracterizada pelo valor do parâmetro PCR. Nesse caso, se o usuário transmitir a taxas que não ultrapassem o limite PCR, então, a rede se compromete a garantir a qualidade de serviço "negociada" no instante do estabelecimento da conexão.

O serviço CBR é designado às aplicações em tempo real que não admitem variações significativas no retardo de transferência de células (exigem baixo CDVpp). Porém, o mesmo não se limita a essas aplicações. Exemplos de aplicações que podem utilizar o CBR são: voz, vídeo e emulação de circuitos.

3.2.1.2 Categoria de serviço rt-VBR

Como o CBR, o serviço rt-VBR também destina-se às aplicações em tempo real que necessitam de baixos valores de retardo e de variação de retardo. A diferença entre os dois reside no fato de que, o primeiro é caracterizado somente pelo parâmetro PCR, enquanto que o segundo envolve três parâmetros, que são: PCR, SCR e MBS. Os outros dois parâmetros que são acrescido ao serviço rt-VBR, se fazem necessários em função da taxa de tráfego ser variável, ou seja, a fonte geradora de informação (equipamento transmissor) pode transmitir a taxas variáveis, o que eqüivale a dizer que a fonte é surtiva (ou que transmite em rajadas).

Esse serviço pode suportar multiplexação estatística de fontes em tempo real e, ainda, atribui menor relevância às células que sofrem retardos acima do valor especificado pelo parâmetro CTDmax. Exemplos de aplicações que podem trabalhar com o rt-VBR são: voz e vídeo comprimidos.

3.2.1.3 Categoria de serviço nrt-VBR

Destina-se às aplicações não em tempo real e que apresentam tráfego surtivo e, também, são caracterizadas em termos dos parâmetros PCR, SCR e MBS. Nesta categoria de serviço nenhuma limitação de retardo é tratada e, para as aplicações que estiverem transmitindo dentro do limite do contrato de tráfego, espera-se baixa taxa de perda de células (baixa CLR).

Esta categoria de serviço também suporta multiplexação estatística de conexões. Exemplos típicos de aplicação para essa categoria de serviço são os processamentos de transações em que o tempo de resposta é crítico, por exemplo, reserva de passagens em companhias aéreas, transações bancárias e monitoração de processos.

3.2.1.4 Categoria de serviço UBR

Também destinada às aplicações não em tempo real. Essa categoria de serviço não provê garantias quanto ao retardo e perda de célula. O serviço fornecido é do tipo "melhor esforço" e não há justiça ou isolamento entre as conexões. Onde o termo justiça refere-se à eqüidade para todas as conexões e isolamento significa inexistência de interferência entre as conexões quanto aos parâmetros de QoS.

O exemplo mais típico de aplicação desta categoria de serviço é o tráfego de dados tradicional tal como transferência de arquivos e correio eletrônico.

3.2.1.5 Categoria de serviço ABR

Destinada à aplicações não em tempo real e, portanto, também não oferece garantia de retardo. Contudo, oferece garantia de taxa mínima de células e provê baixa taxa de perda de células através de um mecanismo de controle de fluxo.

O mecanismo de controle de fluxo sinaliza às fontes as mudanças que ocorrem com a carga da rede através de células de controle específicas denominadas "Células de Gerenciamento de Recursos" (*Resource Management Cells*) ou, simplesmente, células RM e, assim, as fontes ajustam suas taxas de transmissão de acordo com o solicitado pela rede.

As fontes compartilham a banda passante disponível com eqüidade (todas as fontes têm direitos iguais sobre a banda passante) e, jamais são solicitadas a transmitir numa taxa abaixo de suas respectivas taxas mínimas (MCR).

3.2.1.6 Categoria de serviço GFR

O serviço GFR, como os serviços UBR e ABR, também destina-se às aplicações não em tempo real. Tais aplicações podem requerer garantia de taxa de transmissão mínima e, ainda, se beneficiarem de acesso a banda passante adicional disponibilizada dinamicamente na rede.

De outra maneira, pode-se dizer que o serviço GFR destina-se a usuários que não podem especificar todos os parâmetros que são necessários numa solicitação de serviço VBR e não possuem equipamentos terminais que cumpram com as exigências de comportamento solicitadas pelo serviço ABR. Embora tais usuários possam recorrer às conexões UBR, as mesmas não oferecem garantia de QoS alguma.

Outra característica importante do serviço GFR é que o mesmo requer pouca interação entre os usuários e a rede, além da garantia de taxa mínima de células e baixa taxa de perda de células, para os quadros em conformidade com os limites da categoria.

3.2.2 Fatores que afetam os parâmetros de QoS do ATM

São apresentados nesta sub-seção os principais fatores a serem considerados no ajuste dos parâmetros de QoS. Esta abordagem é dependente das características encontradas tanto nas redes públicas quanto nas redes privadas ou, ainda, nas redes mistas que integram estes dois tipos de redes.

Os fatores indicados abaixo representam as principais causas de degradação da QoS nas redes ATM. Maior detalhamento pode ser encontrado em [1].

- Retardo de propagação
- Retardo por erro médio estatístico
- Retardo dependente da arquitetura do *switch*
- Retardo por limitação de capacidade de *buffer*
- Carga de tráfego
- Número de nós na rota
- Alocação de recursos
- Falhas

A tabela 3.2 a seguir, relaciona tais fatores aos parâmentos de QoS. Cabe observar que a quantidade de nós na "rota" utilizada pelas conexões é o atributo que mais influencia os parâmetros de QoS, enquanto que o efeito do retardo de propagação é sentido apenas pelo parâmetro CDTmáx. Maiores detalhes podem ser vistos em [1].

Atributo	CER	SECBR	CLR	CMR	CTDmáx	CDVpp
Retardo de propagação					X	
Estatística de erro médio	X	X	X	X		
Arquitetura do *switch*			X		X	X
Capacidade de *buffer*		X	X		X	X
Número de nós na rota	X	X	X	X	X	X
Carga de tráfego			X	X	X	X
Falhas	X	X	X			
Reserva de recursos			X		X	X

CER Taxa de erro de célula
CLR Taxa de perda de célula
CTDmáx Retardo máximo de transferência de célula
SECBR Taxa de bloco de células com erro crítico
CMR Taxa de inserção inadequada de célula
CDVpp Variação pico-a-pico do retardo de célula

Tabela 3.2 – Resumo da influência sofrida pelos parâmetros de QoS devido às várias fontes de degradação.

3.3 SERVIÇOS INTEGRADOS (IntServ) E O RSVP

Em função da necessidade de se oferecer QoS na Internet às aplicações multimídia sensíveis a atraso e variação de atraso, tais como: vídeoconferência, telefonia sobre IP, Vídeo sob demanda (VoD), etc, a comunidade Internet foi obrigada a desenvolver uma nova arquitetura para a Internet que fosse capaz de oferecer QoS satisfatória a essas aplicações, além do tradicional serviço de "melhor esforço" (*Best Effort* - BE). Daí o surgimento da arquitetura "Serviços Integrados" (*Integrated Services Architecture*) [10,12].

A arquitetura IntServ define dois modelos de serviço: o *Guaranteed Service* - GS e o *Controlled-Load Service* - CLS. O GS oferece garantia de retardo e limite máximo de *jitter*, destinando-se, portanto, a aplicações em tempo real, enquanto que o CLS provê confiabilidade e melhoria em relação ao BE, o que o faz adequado a serviços críticos tais como transações bancárias e telemedicina. O principal mecanismo utilizado por esta

arquitetura é o protocolo de sinalização RSVP (*Resource ReSerVation Protocol*) [11], o qual é utilizado para reserva de recursos na rede.

A figura 3.1 ilustra o princípio de funcionamento do RSVP. O emissor envia ao receptor uma mensagem PATH, a qual segue a rota estabelecida pelo protocolo de roteamento (BGP, OSPF,...) e sinaliza a todos os roteadores nesse caminho informações referentes às características do tráfego que será gerado pelo emissor, além do endereço do roteador imediatamente anterior na rota (*Previous HOP* – PHOP). Os roteadores, por sua vez, armazenam tais informações. O receptor ao receber uma mensagem PATH responde com uma mensagem RESV, a qual é enviada pela mesma rota anteriormente seguida pela mensagem PATH e, ao passar por cada roteador informa ao mesmo a quantidade de recursos (banda passante e *buffer*) que deve ser reservada a fim de que a QoS solicitada seja eficientemente oferecida pela rede e, ainda, os roteadores também armazenam informações de estado de tráfego por fluxo (*flow state*), necessárias ao controle dos recursos reservados.

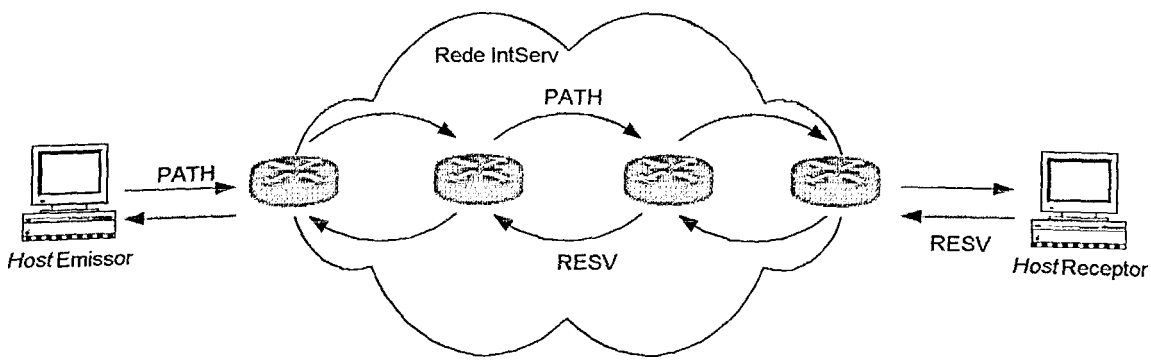

Figura 3.1 – Sinalização do protocolo RSVP na rede IntServ.

No envio da mensagem RESV, se algum dos roteadores não puder disponibilizar a quantidade de recursos solicitada, uma mensagem de erro é enviada ao receptor e o processo de sinalização é finalizado. Por outro lado, os roteadores que não têm implementado em seus respectivos algoritmos o RSVP, simplesmente encaminham as mensagens RSVP aos

roteadores seguintes na rota e, dessa forma, pode-se ter a interoperação dos dois tipos de roteadores (com RSVP e sem RSVP). Os roteadores sem RSVP oferecerão simplesmente o serviço BE. O RSVP opera com um mecanismo denominado de *soft state*, o qual faz com que os recursos reservados expirem após um determinado intervalo sem informações de *refresh* para atualizá-los.

O principal problema do IntServ é a escalabilidade, uma vez que a quantidade de informações de estado de tráfego nos roteadores cresce proporcionalmente com o número de fluxo e, por isso, essa arquitetura não oferece bom desempenho em redes WANs de grande porte, principalmente em *backbones* ATM, onde geralmente as taxas são muito altas e consequentemente a quantidade de informações de estado de tráfego nos nós da rede torna proibitiva a implementação dessa arquitetura.

Quando o RSVP é implementado sobre o ATM em ambientes LANE, CLIP ou MPOA [13,16,48], algumas pesquisas têm mostrado que pode-se minorar a deficiência citada anteriormente, valendo-se, para isso, do fato de que o ATM é orientado a conexão e, por conseguinte, possibilita a montagem de rotas de atalho (*shortcut*) fixas [9] a fim de que o IntServ possa explorar toda a capacidade do ATM de oferecer QoS. Nesse caso, porém, o RSVP se restringe a sinalização no nível do IP e a QoS fim-a-fim fica a cargo do ATM.

3.4 SERVIÇOS DIFERENCIADOS (DiffServ)

A arquitetura "Serviços Diferenciados" (*Differentiated Services*) [15,29] encontra-se em fase de desenvolvimento, com muitos trabalhos relacionados ainda em fase de padronização pela IETF (*Internet Engineering Task Force*), e foi desenvolvida, primordialmente, para suprir a deficiência do serviço IntServ em redes WANs de grande porte, como também permitir tarifações diferenciadas por tipo de serviço oferecido, por parte dos ISPs. A idéia central dessa arquitetura é agregar os fluxos provenientes das aplicações (normalmente sub-redes LANs, Frame Relays ou ISDNs) interligadas aos seus roteadores de

borda (*Edge Router* - ER) em classes de tráfego pré-definidas. Dessa forma, minimiza-se o processamento nos roteadores da rede, uma vez que as informações deixam de ser manuseadas por fluxo simples e passam a ser tratadas por agregados de fluxos.

Assim, conforme ilustra a figura 3.2, o tráfego ao chegar ao ER é classificado e se necessário moldado e, então, é atribuído ao mesmo uma "classe de tratamento de encaminhamento agregado" (AF, EF, BE,..., analisados mais adiante), mediante a inserção de um código (*codepoint*) no campo DS (original TOS do IPv4 e *Traffic Class* do IPv6 renomeado para DS no DiffServ) do cabeçalho do pacote IP [14]. No núcleo da rede DiffServ, os roteadores encaminharão os pacotes (agregados de fluxos) com base na classe de tratamento de encaminhamento, denominada de PHB (*Per Hop Behavior*), que vem indicada no campo DS do cabeçalho de cada pacote. Desse modo, níveis diferenciados de classes de serviço podem ser implementados.

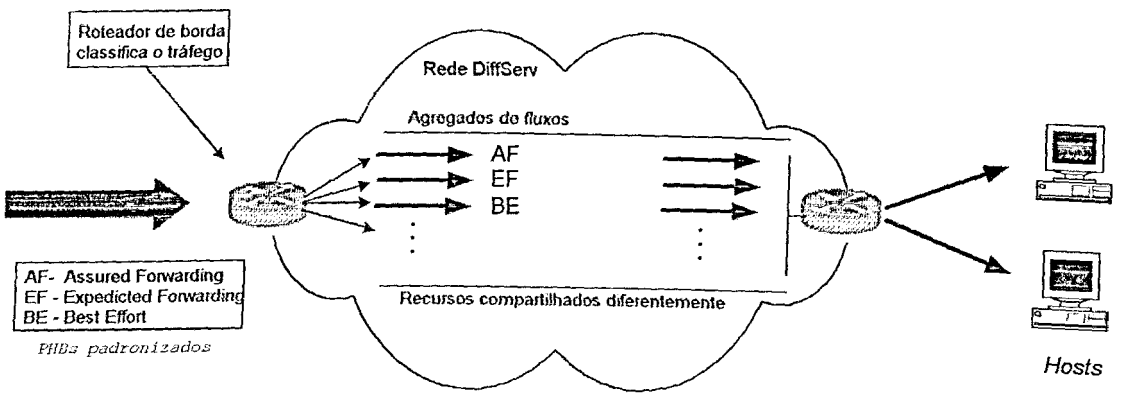

Figura 3.2 – Arquitetura Serviços Diferenciados.

Diferentemente do IntServ, o DiffServ armazena nos roteadores somente informações de estado por classe de serviço ao invés de por fluxo e, como há somente um número limitado de classes, definido pelo campo DS, a quantidade de informações de estado armazenadas é mínima. E, ainda, o trabalho dos roteadores, no núcleo da rede, é simplificado, uma vez que estes se limitam a executar as tarefas de classificação e envio definidas nos PHBs, deixando as demais tarefas, como: classificação sofisticada, marcação (indicação do PHB no campo

DS), policiamento e moldagem, para serem executadas pelos ERs. Estas características conferem à arquitetura DiffServ maior simplicidade de implementação e escalabilidade.

Uma aplicação importante do modelo DiffServ refere-se às redes dos ISPs. E, neste cenário, há a necessidade de se estabelecer, entre o usuário e o ISP, um acordo quanto ao nível do serviço que será oferecido (contrato de tráfego), a fim de que o usuário receba o serviço com a qualidade desejada e o provedor possa monitorar o usuário quanto a extrapolação desse acordo. O referido acordo é denominado de SLA (*Service Level Agreement*). Este conceito também se aplica na interligação de ISPs, inclusive com outras arquiteturas em uso. Em ambos os casos, os ERs de entrada da rede DiffServ (*ingress* ER) devem proceder a: classificação, policiamento e possível moldagem, do tráfego a eles enviados, baseados nos respectivos SLAs.

Em [23] é apresentada uma arquitetura, a qual define as classes de QoS "serviço assegurado" e "serviço prêmio" em complemento ao serviço de melhor esforço (*Assured Service* - AS, *Premium Service* - PS e *Best Effort* – BE, respectivamente). A primeira se destina às aplicações que necessitam de retardo e *jitter* pequenos, enquanto a segunda às que precisam de maior confiabilidade que a oferecida pela última.

Analogamente, o grupo de trabalho "DiffServ WG" da IETF, define duas classes de serviço (PHBs) em complemento ao BE:

- *Assured Forwarding* – AF

Definido em [20], essa classe de serviço oferece alta probabilidade de entrega aos pacotes que estejam de acordo com o contrato de tráfego, denominados pacotes em conformidade, e ainda admite que o tráfego em excesso, relacionado aos pacotes não em conformidade, possa também ser entregue, porém com baixa probabilidade. Os ERs marcam de forma diferenciada esses dois tipos de tráfego, através do campo DS (existe proposta para

se ter mais do que dois níveis de prioridade de descarte), de forma que em situações de congestionamento na rede os pacotes de baixa prioridade são descartados prioritariamente. O AF suporta tráfegos em rajadas e esses podem ser multiplexados estatisticamente de forma análoga à categoria de serviço VBR do ATM.

- *Expedited Forwarding* – EF

Definido em [21], essa classe de serviço garante valores pequenos de retardo, *jitter*, e perda de pacotes, bem como banda passante e QoS fim-a-fim dentro do domínio DiffServ. Para o usuário, este serviço se comporta como se fosse uma conexão ponto-a-ponto ou uma VPN (*Virtual Private Network*), similar ao serviço CBR do ATM. Desse modo, consegue-se oferecer QoS sobre uma rede IP compartilhada (normalmente rede DiffServ pública) e, consequentemente, reduz-se significativamente o custo para o usuário.

Em função do DiffServ ser apropriado a redes WANs *backbones*, nas quais se nota o crescente emprego do ATM, é importante que se faça uma análise do mapeamento entre essas duas tecnologias. Em princípio esta tarefa não é simples, por exemplo, enquanto o ATM possui apenas dois níveis de prioridades de descarte, especificados pelo bit CLP (*Cell Loss Priority*) e, várias filas para cada categoria de serviço, o serviço DiffServ AF, por exemplo, pode trabalhar com múltiplos níveis de prioridade e classes. O mapeamento é realizado mediante a translação dos PHBs nas categorias de serviço ATM. Em [2,53] são realizados estudos que abordam esse assunto com maiores detalhes.

3.5 MPLS (*MULTIPROTOCOL LABEL SWITCHING*)

O MPLS também é uma tecnologia recente, que se encontra em fase de padronização pela IETF [17] e, teve como origem o protocolo proprietário, *Tag Switching*, desenvolvido pela empresa CISCO, o qual inicialmente foi idealizado, sobretudo, para melhorar o desempenho do roteamento na Internet (nas WANs). Entretanto, como a eficiência global das

redes, empregando esse protocolo, vem apresentando resultados expressivos, a padronização de um protocolo genérico e aberto se fez necessária a fim de que se mantenha no futuro, entre os diversos fabricantes, a interoperabilidade.

O MPLS tem como característica intrínsica o fato de poder operar com múltiplos protocolos tanto na camada de rede como na camada de enlace, definindo-o como um protocolo de encapsulamento (opera entre as camadas 2 e 3 do modelo OSI da ISO). Entretanto, o uso mais expressivo do mesmo tem sido com o protocolo IP na camada de rede e o ATM nas camadas inferiores (como rede de transporte). O MPLS integra essas duas tecnologias de forma muito eficaz, combinando a flexibilidade de roteamento do IP com a eficiência das técnicas de comutação de células do ATM.

Com o MPLS, o roteamento IP tradicional é modificado de modo que os roteadores não necessitem processar os valores de suas respectivas tabelas de roteamento, para cada pacote que chega, a fim de encontrar o endereço de envio. De fato, para cada valor na tabela de rotas é associado um único rótulo (*label*), o qual designa a rota de saída. Dessa forma, cada pacote carrega um rótulo e o encaminhamento é realizado com base nesse rótulo. Se o MPLS trabalhar sobre o ATM, então o rótulo pode ser implementado pelos campos VCI/VPI.

Os roteadores que implementam o MPLS são chamados de LSR (*Label-Switched Router*) e os caminhos virtuais estabelecidos para envio de dados são chamados de LSP (*Label-Switched Path*). O protocolo utilizado para distribuição de rótulos no estabelecimento dos LSPs é o LDP (*Label Distribution Protocol*), embora o protocolo de sinalização "RSVP estendido" ou a técnica *piggybacking* também possam ser usados [17].

No estabelecimento dos LSPs, os LSRs negociam a semântica dos rótulos que será utilizada com os seus respectivos parceiros (*peers*), a qual se refere à forma como os pacotes devem ser tratados em relação aos seus rótulos. O estabelecimento dos LSPs pode ser implementado de forma *control-driven* (realizado pelo mecanismo de controle de tráfego,

análogo à forma como ocorre a atualização de rotas) ou *data-driven* (realizado mediante a demanda de fluxo). A determinação de uma LSP pode ocorrer por meio do roteamento IP *(hop-by-hop)*, ou valendo-se da técnica de rota explicita (*Explicit Route*) .

A rota explícita é uma ferramenta muito poderosa que pode ser utilizada pela "Engenharia de Tráfego" (na escolha de rotas menos sobrecarregadas) no sentido de se obter melhoria significativa no desempenho da rede. Essa técnica quando implementada com o datagrama IP puro não oferece resultados expressivos devido ao *overhead* excessivo no processamento dos endereços IP em cada roteador. O MPLS, porém, não apresenta essa deficiência, uma vez que a especificação da rota explicita só precisa ser realizada no momento da atribuição dos rótulos, realizada pelo *ingress LSR* (LSR da borda de entrada da rede). Portanto, o MPLS facilita a implementação da abordagem ER e, por conseguinte, o trabalho da Engenharia de Tráfego.

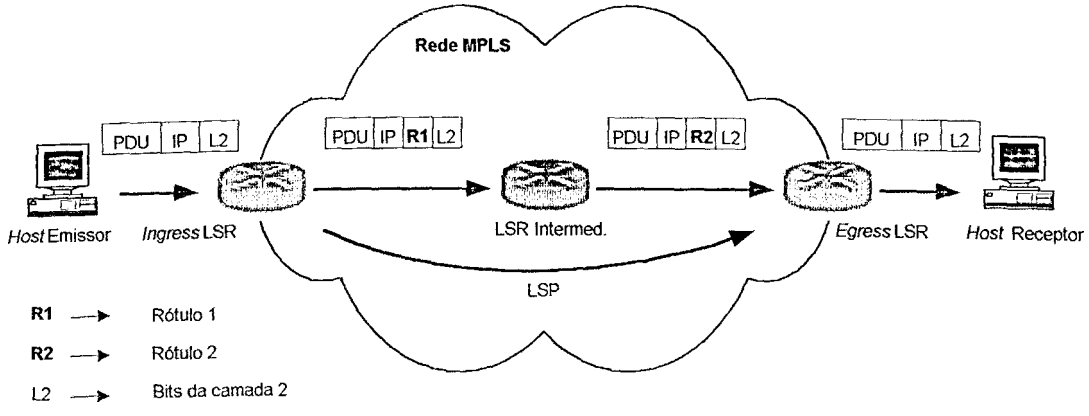

Figura 3.3 – Roteamento no MPLS.

O funcionamento do MPLS pode ser observado na figura 3.3. Ao entrar na rede MPLS (domínio MPLS) o *ingress LSR* classifica, roteia e insere o cabeçalho MPLS aos pacotes. Os LSRs dentro da rede, ao receberem um pacote com rótulo, verificam em suas respectivas tabelas de rótulos a correspondência com o rótulo do pacote e, em seguida, encaminham os pacotes. Dessa forma, o tempo gasto no roteamento diminui sensivelmente quando comparado

com o tradicional roteamento IP (com tabelas de roteamento), pela mesma razão mencionada anteriormente. Os campos *label* e COS do cabeçalho de cada pacote, dentro da rede MPLS, determinam o encaminhamento, a classificação e os serviços de QoS que os pacotes receberão. No *egress LSR* (LSR da borda de saída) da rede MPLS o cabeçalho anteriormente inserido é retirado. Utilizando-se esse procedimento obtém-se maior simplicidade no núcleo da rede MPLS.

Resumindo as principais vantagens do MPLS, tem-se:

- Escalabilidade e suporte à QoS
- Função de roteamento simplificada (redução de *overhead* e latência nos roteadores)
- Suporte à Engenharia de Tráfego e roteamento baseado em QoS
- Eficiente mecanismo de tunelamento
- Independência quanto ao protocolo de rede (IP, IPX, Appletalk,...)

Por oferecer possibilidade de agregação de fluxos associados a classes de QoS diferenciadas, denominadas FEC (*Forward Equivalence Class*), o MPLS pode ser considerado uma tecnologia de serviços diferenciados.

Em relação ao ATM, pode-se dizer que o MPLS utiliza o mesmo conceito de circuito virtual, com reserva de banda passante bem como retardo e *jitter* pequenos. Em função dessa característica, quando o MPLS é implementado sobre o ATM o mesmo dispensa o uso de um protocolo de controle extra para o estabelecimento de VCCs (ex., o PNNI no ATM), mediante o fato de que essa tarefa é executada pelo próprio protocolo MPLS na montagem de seus LSPs. Assim, o uso do MPLS sobre o ATM oferece simplificação de implementação em relação ao ATM puro.

3.6 CENÁRIO ATUAL DE INTEGRAÇÃO DA QoS NA INTERNET

Conforme apresentado nas sub-seções anteriores, cada arquitetura e protocolos relacionados apresenta características que as fazem adequadas a aplicações específicas. De

uma forma geral, pode-se classificá-las como adequadas à LANs ou WANs, embora a tendência seja desaparecer essa distinção, uma vez que essas tecnologias estão sendo aprimoradas, tanto do ponto de vista tecnológico quanto de redução de custos, a fim de que no futuro possa se falar em arquiteturas *All Area Network*.

Entretanto, o estado da arte em QoS na Internet atual engloba o IntServ, o DiffServ e o MPLS, como tecnologias diretamente desenvolvidas para o IP e o ATM como uma nova filosofia de rede que apresenta enorme potencial para QoS fim-a-fim, (apesar de ainda ser pouco explorada quanto a esse aspecto). A integração dessas arquiteturas tem como fundamento, sobretudo, o aproveitamento da enorme infra-estrutura desenvolvida às aplicações IP nos últimos anos, garantido, dessa forma, uma migração para novas tecnologias de forma planejada e eficiente. Assim, fica evidente a importância de se analisar a interoperação entre as mesmas.

O ATM, de acordo com o que foi citado anteriormente, apesar de possuir mecanismos eficientes para oferecer QoS, está sendo utilizado atualmente, primordialmente, por causa da sua alta velocidade de transmissão, o que o faz ideal para as redes *backbones* (LANs e WANs) e, por isso, é a tecnologia dominante nos *backbones* dos ISPs. O IntServ é implementado nas redes de acesso (LAN, *Frame relay*, ISDN, etc), onde o problema de escalabilidade não é crítico (tráfego menor). O serviço DiffServ é empregado nas WANs de grande porte (redes *backbones*) , uma vez que nessas redes a escalabilidade é fundamental, dada as altas taxas de transmissão e o tráfego acentuado inerente a esse ambiente. E, por último, o MPLS está sendo empregado em conjunto com o ATM, nas redes *backbones*. A figura 3.4 ilustra esse cenário.

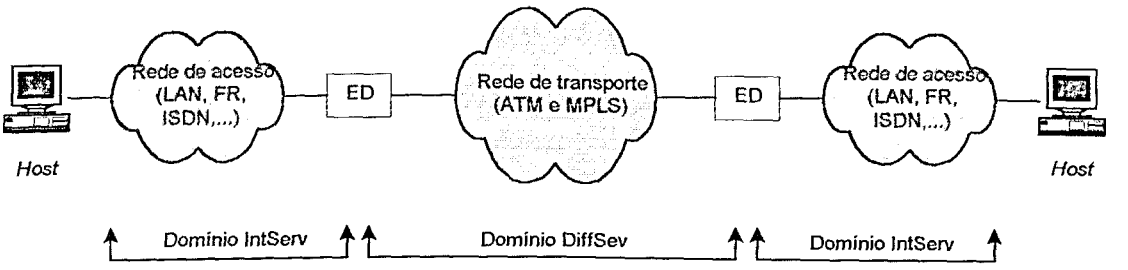

Figura 3.4 – Cenário atual das tecnologias de QoS com alto desempenho na Internet.

Nessa concepção, há a necessidade de se proceder ao mapeamento entre os serviços integrados e diferenciados. Em [7] é feito um estudo em que se trata desse assunto e, uma proposta de padronização acerca do mesmo encontra-se em fase de padronização pela IETF [18].

Conforme pode-se observar pela figura 3.4, o mapeamento entre essas duas tecnologias é realizado pelos EDs (*Edge Devices*) que interligam as redes relacionadas com cada uma delas (podem também ser implementados nos próprios EDs de cada rede envolvida). Dessa maneira, os fluxos IntServ provenientes da rede de acesso são mapeados nos agregados fluxos do DiffServ e, assim, o RSVP é submetido a um tunelamento no núcleo da rede DiffServ, de modo que a sua funcionalidade fique limitada às redes de acesso.

Com essa interoperação obtém-se: escalabilidade fim-a-fim para o IntServ; reserva explicita de recursos para as redes de acesso, que podem ter limitação de banda passante e, por último, flexibilidade de acesso ao serviço DiffServ no núcleo da rede, com QoS individualizada por fluxos, ao invés da configuração estática (de agregados) oferecida pelo DiffServ [7].

Atributos	IntServ	DiffServ	MPLS	ATM
Camada do modelo OSI	Transporte	Transporte	Rede/Enlace	Enlace/Fisico
Aplicação principal	Redes de acesso	WANs	WANs	*Backbones*
Vantagem principal	QoS individualizada	Diferenciação de perfil e preços	Roteamento rápido	Mecanismo de QoS nativos
Escalabilidade	Fraca	Boa	Boa	Boa
Requisitos dos *hosts*	Médio	Alto	Baixo	Baixo
Requisitos dos nós intermediários	Alto	Baixo	Médio	Médio
Baseado em circuito virtual	Sim	Não	Sim	Sim
Protocolo de sinalização	RSVP	RSVP	LDP	PNNI
Serviço em tempo real	GS	EF	n/a	CBR/rt-VBR

n/a - não aplicável

Tabela 3.3 – Comparação entre as tecnologias de QoS.

Quanto às redes *backbones* dos provedores de serviço, de acordo com o que foi dito na sub-seção anterior, o uso do MPLS sobre o ATM é a opção mais eficiente a ser implementada nessas redes, sobretudo, porque o MPLS é um protocolo de encapsulamento e o ATM oferece taxas de transmissão excepcionais. A tabela 3.3 resume as principais características de cada uma das tecnologias de QoS estudas nesse capítulo.

Por fim, aliado a todas essas tecnologias aqui estudadas, também é emergente o uso da Engenharia de Tráfego [22]. Essa abordagem diz respeito ao gerenciamento das rotas da rede de modo que essas sejam estabelecidas visando sempre a reserva eficiente e justa (distribuição de tráfego com eqüidade) de recursos na rede e, desse modo, evitar que alguns pontos da rede fiquem mais sobrecarregados que outros por longos intervalos de tempo. Para isso, a Engenharia de Tráfego faz uso da técnica de roteamento baseado na QoS (*QoS routing*) [19],

a qual realiza a seleção das rotas com base nos parâmetros de QoS. Diferentemente, portanto, dos protocolos de roteamento IP (BGP, OSPF, etc) que procuram sempre pela menor métrica possível.

3.7 CONCLUSÕES

Cada uma das tecnologias estudadas nesse capítulo possui características que as faz adequadas a situações específicas, de modo que a tendência atual é a integração de todas essas tecnologias. Por isso, fica estabelecido que uma única tecnologia, isoladamente, não é capaz de solucionar o problema de QoS na Internet dos dias atuais, em função, sobretudo, da heterogeneidade das sub-redes (associadas às suas aplicações), que compõem esse ambiente.

Por outro lado, a interoperação entre as tecnologias citadas demanda um mapeamento adequado entre as mesmas, o que representa uma complexidade a parte a ser superada pelos desenvolvedores dessas tecnologias. Além do mais, esses mapeamentos não devem interferir de forma expressiva no desempenho global da rede.

Portanto, o provimento de QoS fim-a-fim às aplicações emergentes na Internet atual ainda é algo a ser conquistado, visto que os mapeamentos citados estão sendo extensivamente aprimorados de modo que possam vir a oferecer os requisitos mínimos requeridos por tais aplicações num futuro próximo.

CAPÍTULO IV

REDE DE SERVIÇOS DIFERENCIADOS

4.1 INTRODUÇÃO

Este capítulo trata da funcionalidade da arquitetura de rede Serviços Diferenciados, ou simplesmente modelo DiffServ, levando-se em conta as várias propostas em desenvolvimento na atualidade [26]. Nesse sentido, os principais blocos que constituem o DiffServ são apresentados e discutidos.

Empregando-se os elementos que constituem o DiffServ pode-se prover uma grande variedade de serviços. As redes DiffServ oferecem um comportamento diferenciado aos pacotes marcados, por meio de um *codepoint* [14], em seus respectivos cabeçalhos. Os elementos da rede, tais como roteadores e *gateways* provêem um tratamento de qualidade superior aos pacotes associados ao DiffServ em relação aos pacotes provenientes do serviço BE. Com isso, eles possibilitam que os pacotes DiffServ sejam protegidos com maior eficácia contra os demais fluxos, especialmente os fluxos UDP agressivos (sem controle), e também possibilitam a adequação de uso de banda passante com base nesse esquema de prioridade.

Conforme mencionado no capítulo anterior, as propostas de serviço diferenciado padronizadas atualmente pela IETF, mas ainda em desenvolvimento, são os serviços AF e EF. O tráfego remanescente é enviado como tráfego BE, o qual corresponde ao tráfego tradicional da Internet em que os dados são enviados de forma tão rápida e confiável quanto possível, porém, sem nenhum compromisso de desempenho mínimo.

Nessas redes, cada roteador tem seu próprio conjunto de funcionalidades DiffServ e que deve ser tão simples quanto possível. Os roteadores de borda, devem possuir um conjunto

completo de funcionalidades DiffServ a fim de que possam funcionar em função do SLA, via os "negociadores de banda passante" (*Bandwidth Brokers* - BB) [5], presentes na borda das redes dos ISPs ou das VPNs. Por outro lado, os roteadores no interior da rede devem possuir um conjunto menor dessas funcionalidades e também filas de armazenamento menores, uma vez que todo o tráfego enviado a estes deve seguir um SLA específico e possivelmente já tenha sido moldado na borda da rede.

A funcionalidade DiffServ completa é obtida mediante o condicionamento de tráfego, o qual envolve um conjunto de mecanismos empregados com a finalidade de garantir um tratamento diferenciado aos pacotes. As funções desempenhadas por estes mecanismos consistem em: classificação, monitoração, marcação, descarte, enfileiramento, retirada das filas e moldagem dos pacotes [15]. Os pacotes não em conformidade, em relação ao SLA, podem ser enfileirados até que os mesmos estejam em conformidade (pelo processo de moldagem); ou então serem descartados, marcados com um novo *codepoint* (remarcados) ou encaminhados sem nenhuma alteração, desde que algum procedimento de supervisão seja acionado.

Este capítulo restringe-se à descrição do comportamento dos blocos funcionais do DiffServ, uma vez que implementações práticas deverão ser desenvolvidas pelos diversos fabricantes relacionados com o assunto. Nesse sentido, muitas das propostas para essa arquitetura são discutidas e comparadas ao longo deste capítulo.

4.2 ESTRUTURA DO MODELO DiffServ

A estrutura do DiffServ é implementada de tal forma que os pacotes ao entrarem nessa rede devem atravessar alguns blocos (a quantidade e existência desses blocos pode variar de uma implementação de roteador para outra) antes de serem enviados ao próximo nó da rede (*next hop router*). A figura 4.1 apresenta um diagrama de blocos geral de como o modelo DiffServ pode ser encontrado num roteador.

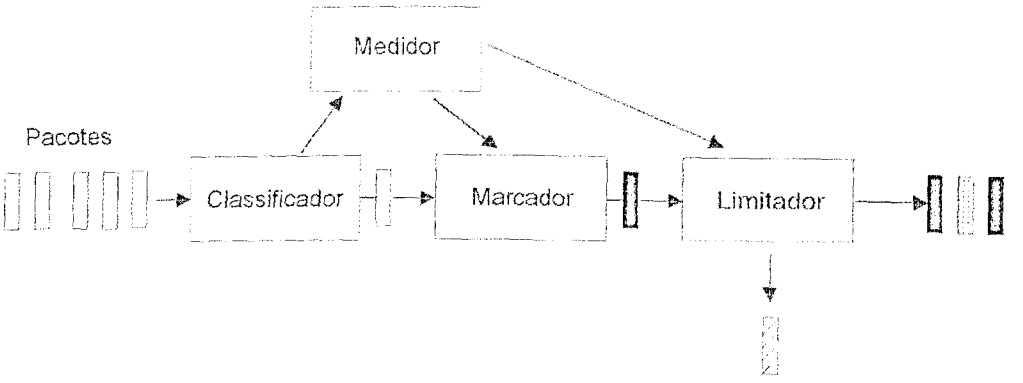

Figura 4.1 – Blocos que formam um roteador DiffServ.

Os pacotes ao chegarem na rede são primeiramente classificados, devido ao fato de que quando os pacotes alcançam o roteador de borda, ou o "roteador do primeiro salto" (*first hop router*) em alguns casos, eles não possuem ainda uma classificação definida. Então, este roteador aplica uma classificação a tais pacotes de forma que a partir desse instante os mesmos passem a pertencer a certas classes de tráfegos distintas. Uma vez classificado, cada pacote é enviado ao bloco Medidor. Cabe observar que a classe de tráfego de cada pacote é marcada na primeira seção do byte DS (que transporta o *codepoint*) denominada de seletor de classe [14] e, uma vez que essa marcação tenha sido atribuída a um pacote, a mesma não pode mais ser alterada por nenhum outro roteador.

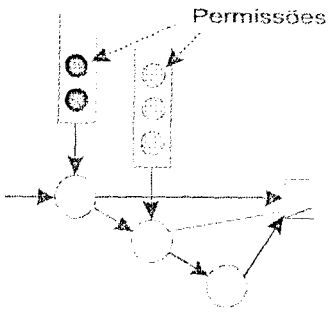

Figura 4.2 – *Token Buckets.*

No bloco Medidor, o pacote é comparado com um perfil de tráfego a fim de verificação quanto aos requisitos das especificações do SLA. Em função do resultado dessa comparação o pacote é marcado com uma prioridade de descarte igual ou maior à sua

prioridade de descarte inicial. A função de medição desempenhada pelo Medidor é realizada como o auxílio dos mecanismos *Token Buckets* (TBs) [8].

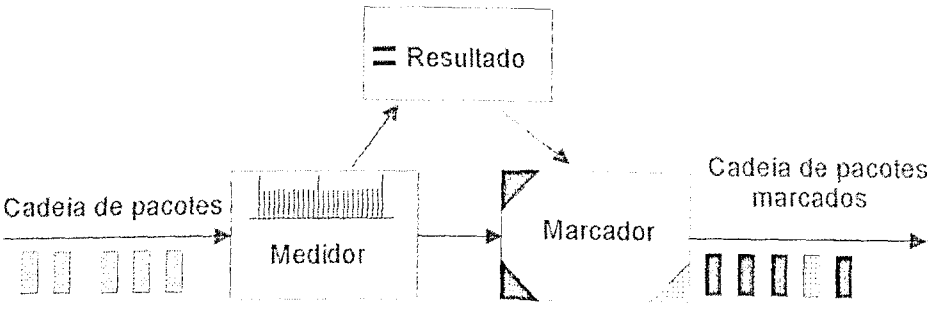

Figura 4.3 – Arquitetura do Medidor/Marcador do DiffServ.

De acordo com o que foi apresentado na seção 2.5.4 do capítulo II, a respeito do funcionamento dos TBs, os "baldes" destes mecanismos são preenchidos com "permissões" emitidas periodicamente pelo sistema (figura 4.2) de maneira que, quando um determinado pacote é recebido pelo Medidor, este o compara com a quantidade de permissões disponível em seu TB (permissões essas que correspondem a certas quantidades de bytes ou pacotes).

Em seguida, o pacote é marcado adequadamente e a quantidade correspondente de permissões é removida do balde. O Marcador altera apenas a prioridade de descarte do pacote (a segunda parte do *codepoint*) e jamais a classe deste. A prioridade de descarte possibilita aos roteadores subseqüentes, bem como aos receptores, saberem do nível de congestionamento dentro da rede, ou ainda, se há ou não congestionamento dentro da mesma. A figura 4.3 resume o que foi explanado até este ponto.

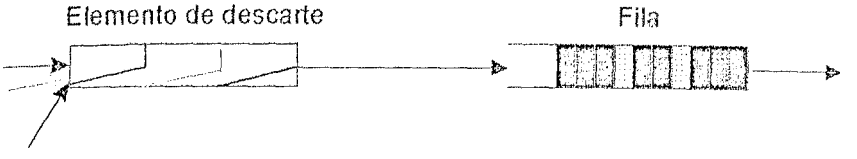

Figura 4.4 – Descarte e enfileiramento (de uma classe de tráfego apenas).

Dependendo da prioridade de descarte e do tamanho da fila de armazenamento, o pacote pode ser enfileirado ou descartado. Os pacotes que não são descartados são armazenados (enfileirados) em filas diferentes, de acordo com a classe de tráfego a qual cada um desses pacotes pertence (figura 4.4). Este procedimento é chamado de enfileiramento baseado em classes (*Class-Based Queueing*) [8].

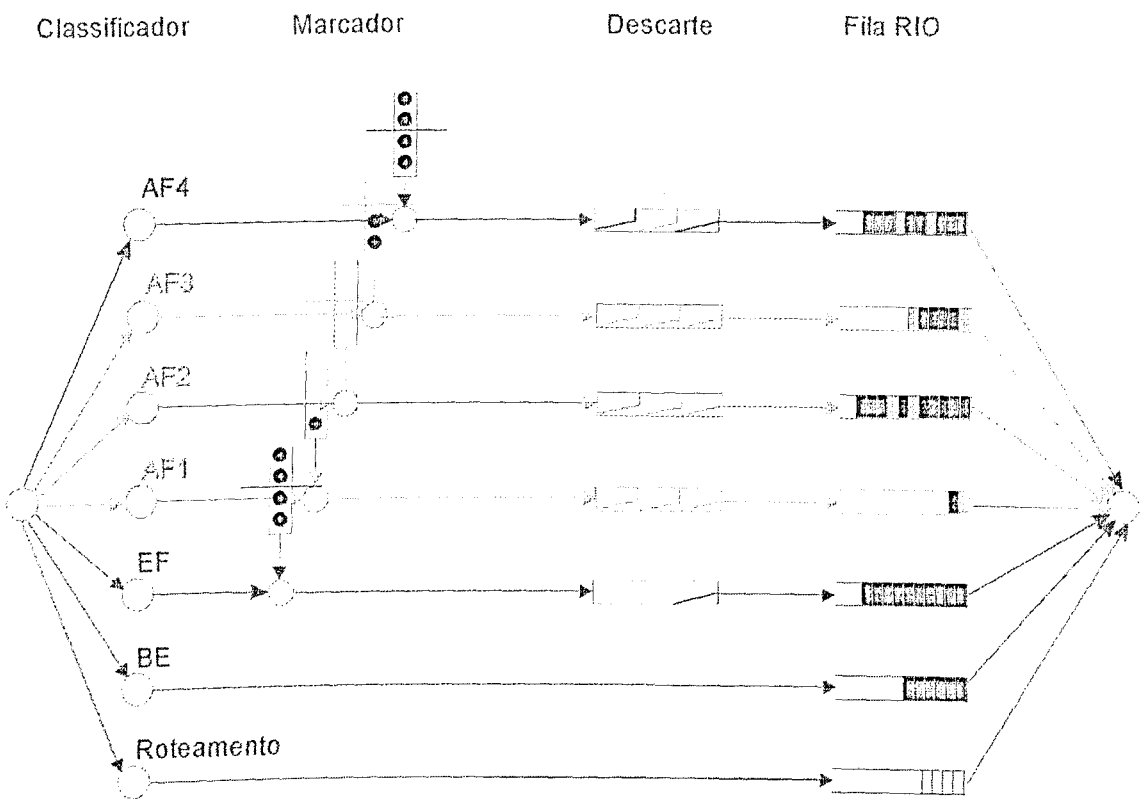

Figura 4.5 – Arquitetura DiffServ com um *Token Bucket*.

Para enviar pacotes ao roteador subseqüente da rota (*next hop router*), o algoritmo de esvaziamento de filas, denominado de *dequeueing*, necessita determinar a fila correta da qual deve ser retirado o pacote. Essa tarefa geralmente é complexa, devido ao fato de que o pacote deve ser escolhido em função do seu tamanho, da banda passante da classe de tráfego a qual o mesmo pertence e talvez também em função do mecanismo de moldagem empregado. Por outro lado, o algoritmo *dequeueing* é vital à estrutura DiffServ, visto que o mesmo é responsável pelo controle de reserva de banda passante das classes de tráfego.

É importante observar que os procedimentos de classificação, medição (com os TBs), marcação, descarte, enfileiramento, *dequeueing* e moldagem, são realizados independentemente para cada classe de tráfego. As quais são divididas da seguinte forma:

- 4 Classes AF
- 1 Classe EF
- 1 Classe BE
- 1 Classe de controle de tráfego

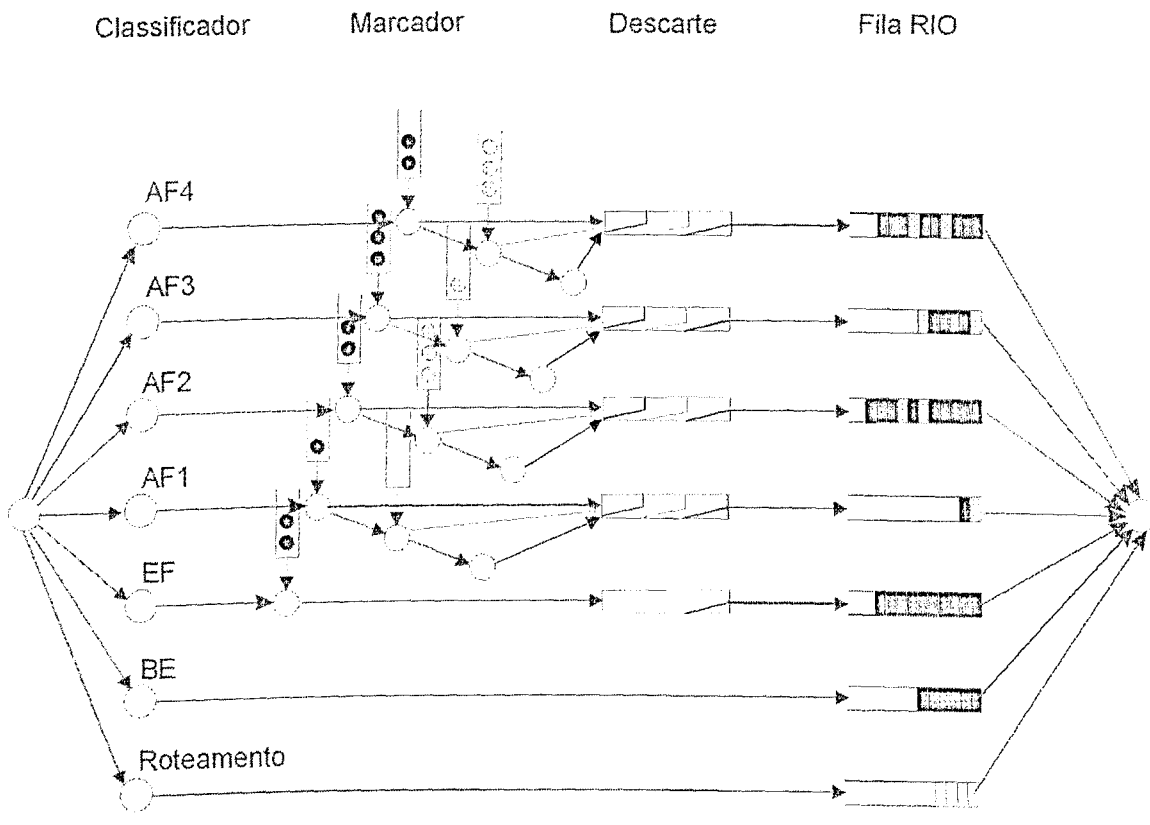

Figura 4.6 – Arquitetura DiffServ com dois *Token Buckets*.

As figuras 4.5 e 4.6 apresentam o conjunto completo dos elementos que formam o DiffServ e que devem fazer parte de um roteador de borda, empregando-se um e dois TBs por classe AF, respectivamente. Pode-se observar que a classe EF possui apenas um TB em ambos os casos, o que ocorre porque o tráfego EF não apresenta surtos (ou rajadas) e, por isso, não há níveis de prioridade para esta classe, ao contrário os pacotes são verificados

apenas quanto à conformidade com o SLA e, de acordo com o resultado, são aceitos pela rede ou descartados (não há marcação).

A classe AF se subdivide em quatro classes de tráfego. Os pacotes são coloridos no desenho a fim de ilustrar melhor os diferentes níveis de prioridades de descarte e, as subclasses de 2 a 4 (AF2, AF3 e AF4) aparecem com uma tonalidade mais fraca nas figuras 4.5 e 4.6 por não estarem sendo implementadas atualmente.

4.3 ELEMENTOS DO MODELO DiffServ

Realiza-se nesta sub-seção um estudo mais detalhado de cada um dos elementos discutidos anteriormente, de modo que a estrutura do DiffServ seja melhor visualizada e assim seja facilitada a compreensão da mesma. Cabe ainda lembrar que a arquitetura DiffServ encontra-se em fase de desenvolvimento e que, por isso, muitas das idéias aqui apresentadas estão sujeitas a alterações em face de adequações que se fizerem necessárias nas implementações práticas.

4.3.1 Classificador (*Classifier*)

Os pacotes IP são classificados nos roteadores de borda [24] e, esta classificação, realizada com base no princípio do comportamento agregado (*Behavior Aggregate* - BA), em que são consideradas apenas classes de tráfego ao invés de fluxos isolados, utiliza apenas o *codepoint* DS. Essa característica representa uma avanço em relação à classificação com base em múltiplos campos (*Multi-Field* - MF), utilizada nas redes IP convencionais, a qual classifica os pacotes por meio da combinação de alguns dos campos do cabeçalho do pacote IP, tais como endereço de origem, endereço de destino, campo DS, porta de origem e de destino e outras informações tais como interface de entrada. A vantagem da classificação baseada no BA é que nesta abordagem os pacotes são classificados no roteador de borda e, portanto, não precisam ser alterados em suas fontes de origem.

4.3.2 Medidor/Marcador (*Meter/Marker*)

Embora as funções de medição e marcação representem duas ações distintas, as mesmas são geralmente tratadas em conjunto, visto que ambas estão muito relacionadas. O Medidor tem por função comparar os pacotes a ele enviados com um perfil de tráfego, enquanto que o Marcador realiza a marcação dos pacotes com os *codepoints* apropriados, de acordo como o resultado da comparação do Medidor [15]. Em outras palavras, diz-se que o Marcador realiza a escrita dos *codepoints* no byte DS e, consequentemente, adiciona os pacotes marcados aos correspondentes comportamentos agregados do DS (DS *behavior aggregate*) ou classes de tráfego.

Nesse sentido, o resultado da análise do Medidor em relação a um determinado pacote (quanto à conformidade com o perfil a ele associado), pode afetar a ação a ser tomada pelos mecanismos de marcação, descarte e moldagem. Referindo-se à implementação, todas as propostas em estudo para o DiffServ sugerem o uso de TBs para auxiliarem a função de medição.

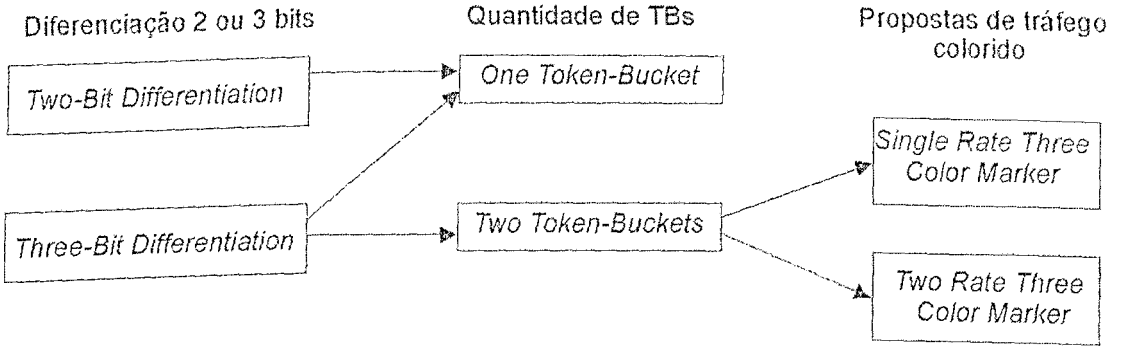

Figura 4.7 – Dependências entre tipos de diferenciação, *Token Buckets* e propostas de tráfego colorido.

Com respeito à marcação dos pacotes, existem duas possibilidades: marcação (figura 4.7) com dois bits de diferenciação (*Two Bit Differentiation*), em que os pacotes são marcados apenas como em conformidade ou não em conformidade com o perfil; e marcação com três bits (*Three Bit Differentiation*), na qual os pacotes são marcados com três níveis de prioridade

de descarte distintos, sendo estes: baixo, médio e alto, e designados também pelas cores verde, amarelo e vermelho (tráfego colorido), respectivamente.

A figura 4.7 apresenta uma visão geral da dependência entre as diferenciações citadas, os *Tokens Buckets* e as propostas de tráfego colorido. A seguir é apresentada uma análise mais detalhada de cada um desses mecanismos.

4.3.2.1 *Two Bit Differentiation*

Esta proposta se baseia na existência dos serviços EF, AF e BE. Nesta abordagem os pacotes EF são marcados utilizando-se o bit denominado P-bit e os pacotes AF empregando-se o bit A-bit. Nesse tipo de marcação emprega-se apenas um TB para cada classe EF e AF. Dessa forma, o serviço AF possui apenas uma classe. Considerando-se a prioridade, o tráfego EF é enviado com prioridade alta (P-bit), o tráfego AF com prioridade baixa (A-bit) e o tráfego BE sem nenhuma prioridade [23].

De acordo com o exposto, sempre que um pacote EF é recebido pelo bloco Marcador, é averiguada a disponibilidade de permissões no mesmo, caso não haja permissões suficientes, o pacote é descartado.

Por outro lado, o pacote AF é testado em relação a um perfil de tráfego, de modo que se este estiver de acordo com o mesmo (em conformidade), então, o pacote é marcado como AF; caso contrário (não em conformidade), o pacote é enviado simplesmente como BE.

Por fim, cabe salientar que esta proposta foi a pioneira no desenvolvimento da arquitetura DiffServ e talvez por isso, como normalmente ocorre com todo início de desenvolvimento de uma tecnologia, ela apresenta deficiências que devem ser superadas por novas propostas. A principal desvantagem desta proposta é o fato de que a mesma não consegue proteger eficientemente os segmentos TCP em conformidade dos segmentos UDP não em conformidade que são inseridos de forma agressiva para dentro da rede (sem

controle). No entanto, esta proposta está sendo tratada aqui para fins de comparação com os avanços conseguidos na proposta *Three Bit Differentiation*.

4.3.2.2 *Three Bit Differentiation*

De forma análoga à proposta anterior, esta proposta também se baseia na existência dos serviços EF, AF e BE; entretanto, o serviço AF possui aqui quatro classes de serviço. Por isso, um *codepoint* é reservado ao serviço EF e este não pode ser alterado no cabeçalho do pacote, mas se necessário o pacote deve ser descartado, enquanto que o *codepoint* do serviço AF pode ser alterado em função do nível de prioridade a ele atribuído em cada roteador [20,21].

Desse modo, a grande diferença desta proposta *Three Bit Differentiation* frente à proposta *Two Bit Differentiation* apresentada anteriormente, refere-se à especificação do serviço AF. Nesta, o *codepoint* AF pode ter, para cada classe AF, três níveis de prioridade de descarte, de forma que os pacotes com prioridades mais altas são descartados prioritariamente em relação aos pacotes com prioridades mais baixas.

No que diz respeito à marcação, deve ser observado que a parte do *codepoint* que se refere à classe de tráfego, no cabeçalho do pacote, é sempre mantida inalterada, a única alteração possível é na parte responsável pela determinação do nível de prioridade de descarte.

Considerando-se o aspecto implementação, a proposta *Three Bit Differentiation* pode ser implementada com um ou dois TBs. Entretanto, os parâmetros de ajustes necessários na implementação com um TB são expressivamente mais difíceis de serem manuseados do que os exigidos com dois TBs.

Por último, cabe mencionar que esta abordagem, *Three Bit Differentiation*, possibilita melhor discriminação entre os dois tipos de tráfego mencionados na proposta anterior (TCP em conformidade e UDP não em conformidade). Assim, fica claro que esta proposta

representa uma evolução da proposta *Two Bit Differentiation* e, por isso, é a opção mais propensa de ser escolhida nas redes DiffServ futuras.

4.3.2.3 *One Token Bucket*

A conformidade de um pacote empregando-se o mecanismo *Two Bit Differentiation*, com um TB, é verificada da seguinte forma:

1. O pacote está em conformidade quando a quantidade de permissões dentro do balde (do TB) é maior ou igual ao tamanho do pacote. Caso positivo, essa quantidade de permissões (equivalente ao tamanho do pacote) é retirada do balde.
2. O pacote está não em conformidade quando não há permissões no balde ou essa quantidade não é suficiente, comparada com o tamanho do pacote. Nesse caso, as permissões são mantidas no balde.

Por outro lado, a conformidade de um pacote dentro do mecanismo *Three Bit Differentiation*, utilizando também um TB, é averiguada mediante o uso da seguinte fórmula:

Número de permissões no balde – tamanho do pacote = X

1. O pacote está em conformidade quando o valor de **X** está acima do limiar pré-estabelecido (figura 4.8) e certamente abaixo da capacidade máxima da fila. Essa quantidade de permissões é removida do balde.
2. O pacote está não em conformidade quando o valor de **X** está entre zero e o limiar. Esta quantidade de permissões é removida do balde.

3. O pacote é descartado quando o valor de **X** está abaixo de zero (negativo). Nesse caso, os permissões remanescentes são deixadas no balde.

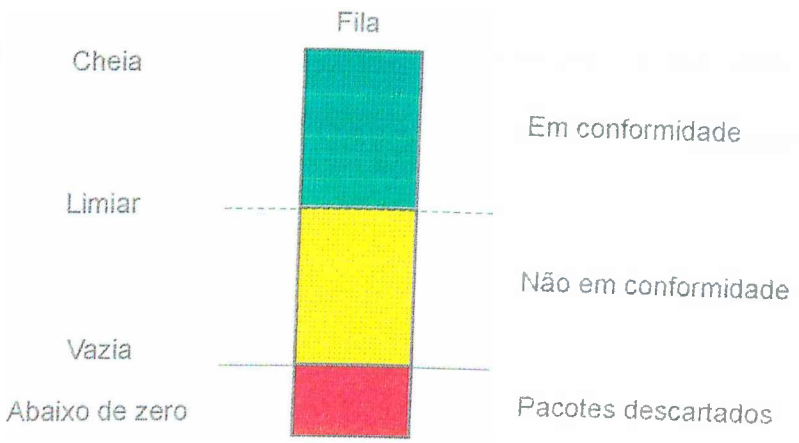

Figura 4.8 – Arquitetura com um *Token Bucket* e *Three Bit Differentiation*.

4.3.2.4 *Two* Token *Buckets*

Num Medidor com dois TBs, cada TB tem uma tarefa distinta. O primeiro compara o tráfego regular (não em rajadas) com a banda passante máxima, enquanto que o segundo verifica o quanto de rajada o sistema pode absorver.

Os mecanismos empregados para a medida de tráfego com dois TBs são diferentes de uma proposta para outra. Entretanto, em linhas gerais esses mecanismos atuam da seguinte forma (figura 4.9):

1. Quando o pacote recebido tiver sido marcado inicialmente com prioridade baixa de descarte e houver quantidade suficiente de permissões no primeiro balde, então, o pacote é marcado com prioridade baixa de descarte.

2. Caso contrário, o pacote oriundo do primeiro passo ou o pacote que tiver sido marcado inicialmente com prioridade média de descarte, é submetido a uma verificação no segundo TB. Se houver uma quantidade suficiente de permissões no segundo balde, então, o pacote é marcado com prioridade média de descarte.
3. Por último, o pacote proveniente dos dois passos anteriores ou que tenha sido marcado inicialmente com prioridade alta de descarte, recebe uma marcação de prioridade alta de descarte.

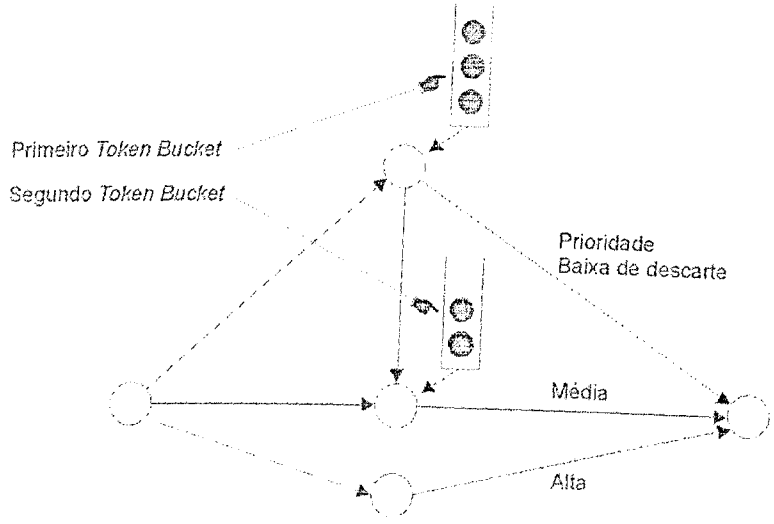

Figura 4.9 – Arquitetura com dois *Token Buckets*.

Deve ser observado que a taxa de atualização de permissões para ambos os TBs é a mesma, e que a quantidade de tráfego regular e de tráfego em rajada pode ser ajustada por intermédio do tamanho de cada TB.

4.3.2.5 *Three Color Marker*

As duas propostas existentes para esses marcadores de três cores, *single rate three color marker* e *two rate three color marker* (de acordo com a figura 4.7), podem ser utilizadas

em um condicionador de tráfego da rede DiffServ e, ambas têm por base o mecanismo *three bit differentiation* com dois TBs [54].

A marcação em três cores propõe uma diferenciação de tráfego com base em três cores: verde, amarelo e vermelho; as quais correspondem às prioridades de descarte baixa, média e alta, respectivamente. Esta proposta apresenta uma nova possibilidade de ajuste para o tamanho e taxa de atualização dos TBs, com o objetivo de proteger de forma mais eficaz o tráfego TCP do tráfego UDP agressivo.

A principal diferença entre as duas propostas refere-se à forma como as permissões dos TBs são atualizadas. Enquanto que no *single rate three color marker* ambos os TBs são atualizados com a mesma taxa de inserção de permissões, no *two rate three color marker* cada TB possui uma taxa de atualização diferente. Por outro lado, esses dois Medidores podem operar em dois modos: *color-blind* e *color-aware*. No primeiro caso, o Medidor considera que o pacote não é colorido e, no segundo, é considerado que o pacote já tenha sido previamente colorido.

4.3.2.6 *Single Rate Three Color Marker* (srTCM)

O srTCM [55] mede um determinado fluxo de tráfego e procede à marcação dos pacotes desse fluxo com base numa informação de comprometimento de taxa, denominada de CIR (*Committed Information Rate*), e dois tamanhos de baldes associados, o CBS (*Committed Burst Size*) e o EBS (*Excess Burst Size*). Nesse sentido, um pacote é marcado como verde se o mesmo não exceder o CBS, ou amarelo se este exceder o CBS mas não exceder o EBS, ou vermelho nos outros casos. O srTCM é útil no policiamento de um serviço no qual somente o comprimento de uma rajada (e não a sua taxa de pico) é utilizado para determinar a elegibilidade de um serviço.

Conforme pode ser observado pela figura 4.10, a quantidade máxima de permissões para o balde C é o CBS e para o balde E é o EBS. Ambos os baldes são atualizados, com novas permissões, CIR vezes por segundo de acordo com o seguinte procedimento. Se a quantidade de permissões no balde C é menor do que o CBS, então, uma permissão é inserida em C, senão, se o número de permissões em E é menor do que o EBS uma permissão é acrescida ao balde E.

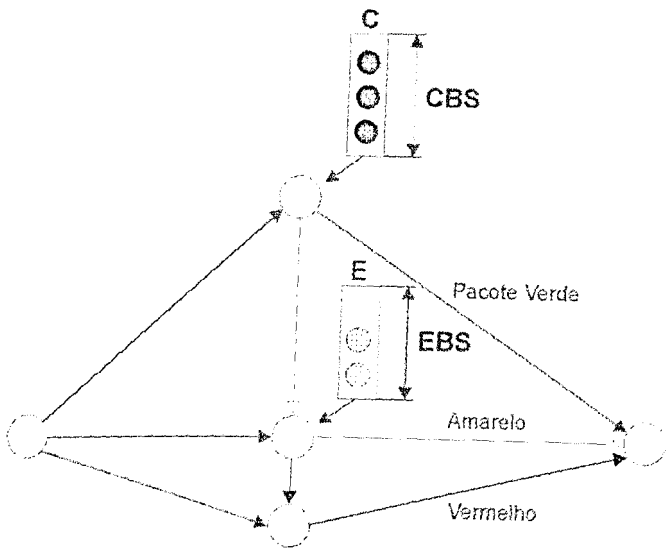

Figura 4.10 – *Single Rate Three Color Marker.*

Ao receber um pacote de tamanho **X** (em bytes), se o srTCM tiver sido configurado para operar no modo *color-aware*, então, as seguintes ações devem acontecer:

1. Se o pacote tiver sido previamente colorido como verde e a verificação de permissões C-**X** ≥ 0 ocorrer, então, o pacote é marcado como verde e **X** permissões são removidas do balde C, senão,
2. Se o pacote tiver sido previamente colorido como verde (já testado no passo 1) ou amarelo e E-**X** ≥ 0, então, o pacote é marcado como amarelo e **X** permissões são retiradas do balde, senão,
3. O pacote é marcado como vermelho e nenhuma permissão é retirada dos baldes.

No modo *color-blind*, o srTCM opera da mesma forma, porém, não é considerado que o pacote tenha sido previamente colorido.

Então, de acordo com estas regras, a marcação de um pacote com uma dada cor requer que haja permissões suficientes para aquela cor a fim de acomodar o pacote por inteiro. Graças a essa característica, o volume de pacotes verdes no Medidor nunca é inferior aos valores determinados pelos parâmetros CIR e CBS, o que indica que permissões referentes a uma determinada cor somente são gastos com pacotes dessa mesma cor. Portanto, o resultado dessa medida determina o valor do *codepoint* que o Marcador coloca no byte DS do cabeçalho do pacote.

4.3.2.7 *Two Rate Three Color Marker* (trTCM)

O trTCM [56] procede à medida e marcação de um fluxo de pacotes IP levando em conta duas taxas, a PIR (*Peak Information Rate*) e a CIR (*Committed Information Rate*) e ainda os tamanhos de rajadas associados a cada uma destas taxas, o PBS (*Peak Burst Size*) e o CBS (*Committed Burst Size*). Desse modo, um determinado pacote é marcado como vermelho se o mesmo exceder a PIR; caso contrário, o mesmo pode ser marcado como amarelo ou verde, isso depende do fato do mesmo exceder ou não a CIR.

Esse tipo de Marcador é útil em um serviço de policiamento em que uma taxa de pico precisa ser oferecida separadamente da CIR. Ainda, é recomendado que o PBS e o CBS sejam configurados de modo que esses sejam maiores ou iguais ao tamanho do maior pacote IP possível no fluxo, enquanto que a PIR deve ser maior ou igual a CIR.

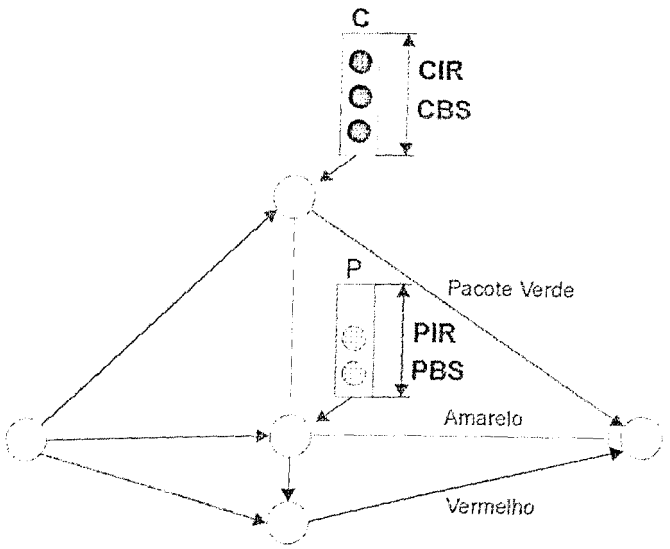

Figura 4.11 – Two *Rate Three Color Marker*

De acordo com a figura 4.11, a quantidade máxima de permissões no balde C é CBS e no balde P é PBS. O balde C é atualizado CIR vezes por segundo e, se o número de permissões no mesmo for menor do que o CBS, então, uma permissão é inserida a este. Analogamente, o balde P é atualizado PIR vezes por segundo e, se o número de permissões no mesmo for menor que o PBS, então, uma permissão é adicionada ao mesmo.

Ao receber um pacote de tamanho **X**, se o srTCM estiver configurado para operar no modo *color-aware*, então, o seguinte procedimento deve acontecer:

1. Se o pacote tiver sido previamente colorido como vermelho e a verificação de permissões $P\text{-}\mathbf{X} \leq 0$ for verdadeira, então, o pacote é marcado como vermelho, senão,
2. Se o pacote tiver sido previamente colorido como amarelo ou se a verificação de permissões $C\text{-}\mathbf{X} \leq 0$ ocorrer, então o pacote é marcado como amarelo e **X** permissões são retiradas do balde P, senão,
3. O pacote é marcado como verde e **X** permissões são removidas dos baldes P e C.

No modo *color-blind*, o trTCM opera da mesma forma, porém, não é considerado que o pacote tenha sido previamente colorido.

Portanto, de acordo com essas regras, pode-se notar que, analogamente ao caso do srTCM, a marcação de um pacote com uma dada cor requer que haja permissões suficientes para essa mesma cor a fim de acomodar o pacote por inteiro. E, da mesma forma como mencionada anteriormente, o resultado dessa medida determina o valor do *codepoint* que o Marcador coloca no byte DS no cabeçalho do pacote.

4.3.3 Mecanismo de descarte (*Dropper*)

Este mecanismo é o responsável pelo descarte de pacotes de cada fluxo de tráfego a fim de mantê-los em conformidade com um determinado perfil de tráfego. Ainda, para evitar a ocorrência de congestionamentos longos e permitir a transmissão de rajadas de curta duração, emprega-se um algoritmo de gerenciamento ativo. Atualmente os mecanismos de descarte padrão são o RED [28] e o RIO [44], os quais representam configurações já bem conhecidas. Nas implementações com três níveis de prioridade, o RIO, que opera com dois níveis de prioridade, necessita ser adequado.

O descarte de um pacote depende do tamanho corrente da fila e também do nível de prioridade de descarte do pacote. Os pacotes com prioridades alta e média de descarte são descartados primeiramente em relação aos de prioridade baixa. Dessa forma, a fila é protegida contra a possibilidade de sobrecarga (*overload*), o que poderia causar retardo excessivo ou mesmo descarte de pacotes pertencentes a fluxos de comportamento adequado (em conformidade).

Com o objetivo de permitir o transporte de rajadas de tráfego, calcula-se uma média do nível de enchimento da fila, de modo que vários pacotes com o mesmo nível de prioridade possam ser inseridos na fila em curtos intervalos de tempo, considerando-se que os mesmos

correspondam a uma rajada de pacotes. Assim, as rajadas podem ser enfileiradas e transmitidas enquanto que os fluxos não em conformidade são na sua maioria descartados.

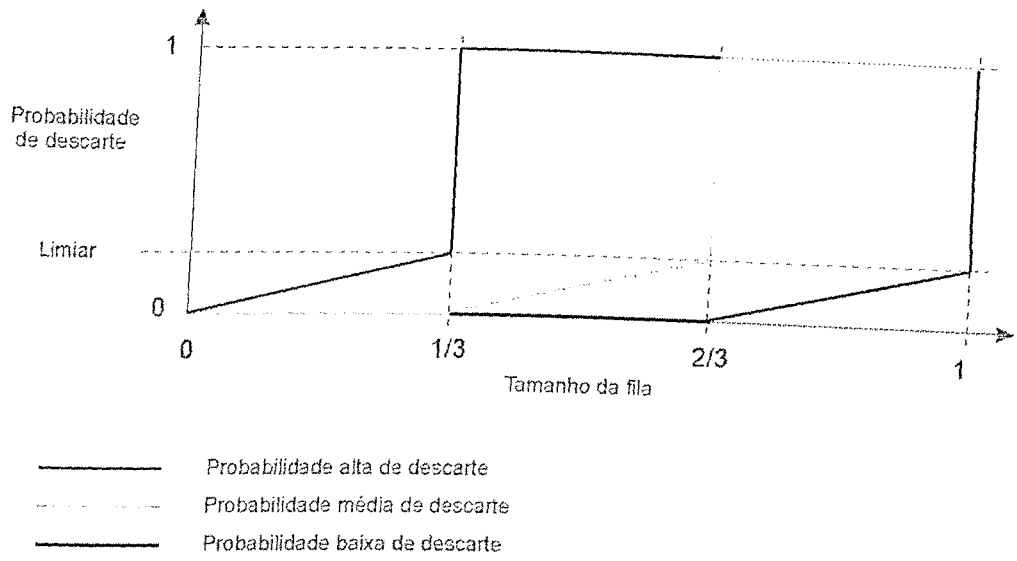

Figura 4.12 – Probabilidade de descarte em função do tamanho da fila para três níveis de prioridade.

É importante observar também que um pacote classificado como sendo de prioridade média ou alta jamais será classificado novamente com prioridade baixa. Este mecanismo indica aos roteadores subseqüentes ou aos receptores a ocorrência de congestionamento na rede. Nesse sentido, a reação a tal informação pode ser a diminuição da taxa de transmissão da fonte TCP até que todos os pacotes estejam em conformidade novamente.

4.3.4 Fila

Após o pacote ter passado por todas as etapas anteriores sem que tenha sido descartado, o mesmo é inserido em uma fila de armazenamento. Essa fila pode ser única, na qual todas as classes de tráfego são mantidas, ou podem haver várias filas, uma para cada classe de tráfego distinta. No caso de apenas uma fila, há a necessidade de um mecanismo inteligente que possa acessar os pacotes dentro da fila ou que proceda à ordenação da mesma. Nesse sentido, uma fila única, com a sua funcionalidade correspondente, demanda custo de

implementação elevado. Além do mais, esses mecanismos são normalmente lentos e difíceis de serem mantidos.

Por tudo isso, é preferível ter-se uma fila independente para cada classe de tráfego, visto que dessa forma os procedimentos de esvaziamento das filas e o manuseio das classes de tráfego são mais simples. A forma mais viável de implementação é ter-se: cada classe AF numa fila distinta, uma fila para o tráfego EF e outra para o tráfego BE (figuras 4.5 e 4.6). Como opção, pode-se tratar os tráfegos relacionados com o roteamento e o gerenciamento, em uma fila separada; porém, o que parece mais lógico é enfileirar esses tráfegos na mesma fila usada pelo serviço EF (alta prioridade).

4.3.5 Esvaziamento de filas/Moldagem (*Dequeue/Shaper*)

Os mecanismos que realizam a moldagem do tráfego (Limitadores), procedem ao atraso de alguns pacotes do fluxo de tráfego forçando o mesmo a entrar em conformidade com um perfil de tráfego. Os pacotes (atrasados) enfileirados no Limitador podem ser descartados se não houver espaço de memória suficiente para acomodá-los. A operação de moldagem é realizada normalmente em conjunto com o procedimento de esvaziamento das filas, o *dequeueing*.

Conforme mencionado anteriormente, na seção referente ao enfileiramento, é preferível ter-se várias filas, uma para cada classe de tráfego, ao invés de uma fila única, onde a complexidade de se separar os pacotes por classe é elevada. Uma fila por classe de tráfego facilita o *dequeueing* e a reserva de banda passante para cada classe de tráfego.

Conforme mencionado na sub-seção 2.5.3 do capítulo II, existem vários algoritmos de *dequeueing*, porém serão abordados a seguir somente os mais relevantes à arquitetura DiffServ.

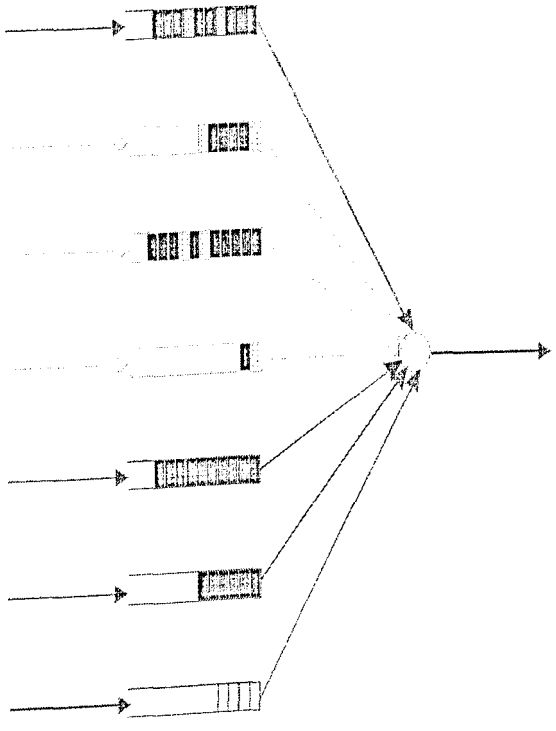

Figura 4.13 – Atendimento (esvaziamento) das filas.

4.3.5.1 *Round Robin* (RR)

Este mecanismo de *dequeueing* procede à varredura de todas as filas ciclicamente (figura 4.13) de modo que todas sejam verificadas dentro de um ciclo quanto à presença de pacotes. Assim, quando uma determinada fila não contiver pacote, a fila subseqüente é verificada e assim sucessivamente. Apesar da simplicidade, este algoritmo garante que todos os pacotes enfileirados sejam retirados de suas respectivas filas. Porém, o mesmo não é adequado à rede DiffServ, visto que o mesmo oferece tratamento eqüitativo a todos os pacotes, independente da prioridade dos mesmos.

Como exemplo, considere-se as duas classes de tráfego EF e AF. Quando os pacotes destes serviços estiverem enfileirados, os mesmos serão retirados segundo a vez de cada um

dentro do ciclo de varredura do RR e, portanto, não será considerada a urgência do pacote EF. Nesta situação, a distribuição de banda passante é de 50% para o EF e 50% para o BE, o que contraria o princípio do modelo DiffServ (justiça com proporcionalidade).

4.3.5.2 *Priority Round Robin* (PRR)

O algoritmo PRR prioriza o esvaziamento da fila EF, em seguida vêm as filas AF e então a fila BE. Dessa forma, contrariamente ao algoritmo RR, o PRR garante prioridade efetiva ao tráfego de maior prioridade (EF). Todavia, esse algoritmo apresenta a desvantagem de permitir que o tráfego EF monopolize os serviços da rede se porventura o mesmo tiver grandes quantidades de pacotes a serem enviados. Tal procedimento certamente não é compatível com os requisitos da rede DiffServ, onde os serviços BE e AF devem receber um nível mínimo de banda passante disponível na rede.

Portanto, no caso de se empregar esse algoritmo numa rede DiffServ, a banda passante da rede deve ser gerenciada em função do SLA, a fim de se evitar a desvantagem citada.

4.3.5.3 *Weighted Round Robin* (WRR)

O algoritmo WRR esvazia as filas ciclicamente tal como o RR, porém, o WRR retira de cada fila um ou mais pacotes por vez, de acordo com o compartilhamento pré-estabelecido de pacotes. Por exemplo, o algoritmo WRR pode retirar cinco pacotes EF, em seguida três pacotes AF e por último dois pacotes BE, então recomeçar com cinco pacotes EF e assim por diante. Este exemplo corresponde a um compartilhamento de banda passante de 50% para o tráfego EF, 30% para o AF e 20% para o BE.

Este princípio funciona adequadamente quando todos os pacotes têm o mesmo tamanho ou a média do tamanho dos mesmos é conhecida a priori, uma vez que isso possibilita que os parâmetros para compartilhamento da banda passante (neste exemplo 5, 3 e 2 para os serviços EF, AF e BE, respectivamente) da rede possam ser escolhidos

corretamente. Então, a solução é realizar o cálculo do tamanho médio de pacote de um determinado tráfego e ajustar os parâmetros citados de acordo com este. Entretanto, encontrar um valor ótimo para esses parâmetros é algo extremamente complexo, sobretudo devido ao fato de que o algoritmo tem de adequar a sua distribuição de banda passante em função do tamanho dos pacotes enviados e recebidos.

4.3.5.4 *Weighted Fair Queue* (WFQ)

Este algoritmo realiza a comparação entre o compartilhamento de banda passante configurado e o compartilhamento de fato utilizado, com o objetivo de retirar um pacote da fila que apresentar a maior diferença positiva. Dessa forma, assegura-se que nenhum serviço receba mais banda passante, na média, do que a previamente configurada. Por outro lado, quando houver banda passante suficiente disponível e não houver nenhum outro serviço enfileirado, então o serviço presente (que está sendo atendido) pode fazer uso de toda a banda passante disponibilizada. Ainda, se houver outros serviços enfileirados, a banda passante excedente é distribuída de forma justa entre os mesmos.

Utilizando-se uma janela de pacotes que registre os últimos X pacotes enfileirados, suas respectivas classe de tráfego e tamanho, uma distribuição exata de banda passante pode ser conseguida. Porém, o tamanho da janela de pacotes representa um parâmetro crítico, visto que se escolhido muito pequeno o algoritmo *dequeueing* não consegue diferenciar eficientemente as classes de tráfego e, como conseqüência, o resultado é um comportamento similar ao do algoritmo RR.

Contrariamente, se o tamanho da janela escolhido for muito grande, o último tráfego que estiver sendo transmitido pode fazê-lo até que uma quantidade elevada de classes de tráfego tenham sido transmitidas e, assim, esse tráfego estará utilizando toda a banda passante disponível por um curto intervalo de tempo e, simultaneamente, impedindo os demais tráfegos regulares (que não apresentam rajadas) de serem encaminhados. Tal problema é

particularmente importante considerando-se as exigências de tempo real do serviço EF. Portanto, o tamanho da janela de pacotes deve ser escolhido com muito critério.

Por fim, pode-se dizer que com o WFQ uma operação de moldagem pode ser obtida eficientemente, visto que as rajadas de um determinado serviço são recebidas e enfileiradas mas não são enviadas como rajadas. No entanto, o WFQ tem de ser configurado de acordo com o enfileiramento (ajustes do TB) e com o SLA.

4.3.5.5 *Priority Weighted Fair Queue* (PWFQ)

O algoritmo PWFQ corresponde a uma combinação dos algoritmos anteriores PRR e WFQ. Nesse algoritmo, o tráfego EF é retirado de sua fila com a mais alta prioridade, independentemente se outros tráfegos estão presentes ou não, de modo que o mesmo possa suportar as exigências impostas pelo tráfego em tempo real. Quanto às quatro classes AF, a classe BE e eventualmente o tráfego de controle da rede, estes são atendidos com o algoritmo WFQ. Dessa forma, apesar do serviço EF possuir prioridade máxima, o mesmo não pode utilizar mais banda passante do que a quantidade que lhe tenha sido previamente reservada, por meio de parâmetros de configuração.

Graças a este procedimento, o PWFQ permite que os outros tráfegos, além do EF, possam transmitir seus tráfegos regularmente. Adicionalmente, o mesmo têm se mostrado, por intermédio de simulações [45], muito eficiente na separação de classes de tráfego e também na distribuição de banda passante a essas classes.

4.4 ARQUITETURA DA REDE DE SERVIÇOS DIFERENCIADOS

Conforme mencionado anteriormente (capítulo III, sub-seção 3.4), a rede DiffServ obtém escalabilidade por meio da agregação de tráfego na borda da rede, realizada mediante a marcação dos pacotes IP, via o campo DS no cabeçalho dos mesmos. Em relação ao transporte desse tráfego agregado, é importante observar que as redes DiffServ normalmente

são formadas por vários domínios (figura 4.14), cada um pertencente a um ISP e, por isso, cada ISP deve garantir que a sua rede não seja sobrecarregada por tráfegos, enviados pelos usuários ou por outros ISPs, que não estejam em conformidade com seus respectivos SLAs.

Nesse sentido, cada ISP deve possuir um roteador de borda que molde o tráfego, a ser admitido pela sua rede, de acordo com o SLA estabelecido, de maneira que os pacotes não em conformidade sejam marcados e possam ser descartados no interior da rede, se necessário, a fim de se evitar o congestionamento dentro da mesma. Assim, se houver disponibilidade de banda passante, o tráfego excedente não em conformidade pode ser transmitido e o usuário será cobrado por esse excedente correspondentemente.

No que diz respeito ao tráfego em rajadas, ao entrar em um domínio DiffServ, comumente referenciado por "domínio DS", o mesmo é limitado a uma taxa máxima de pacotes e, assim, os roteadores no interior da rede não ficam submetidos a rajadas agressivas, tampouco a fluxos com taxas acentuadas. Graças a essa operação de moldagem na borda de um domínio DS, os roteadores no interior da rede DiffServ dispensam funcionalidades tais como TBs e filas de grande capacidade. Dessa forma, o tráfego que entra num domínio DS é encaminhado à saída desse domínio com prioridade alta.

Em princípio, todos os nós a serem atravessados por um pacote da rede DiffServ deveriam ser "DS compatíveis", ou seja, possuírem os mecanismos necessários ao manuseio dos pacotes DiffServ. Entretanto, alguns nós dentro do domínio DS ou entre dois domínios DS, podem ser "não DS compatíveis". Neste caso, se tais nós estiverem dentro do domínio DS o problema não é tão crítico, visto que o tráfego nesse ambiente já deve ter sido previamente moldado e, assim, está menos propenso ao descarte. Porém, se os nós não DS compatíveis estiverem fora do domínio DS, aí então a funcionalidade da rede DiffServ fim-a-fim, quanto ao encaminhamento de classes de tráfego, deixará de fazer sentido e todo o tráfego será transportado como sendo pertencente à classe BE.

Conforme mencionado anteriormente, a arquitetura DiffServ emprega os BBs para realizarem o gerenciamento da banda passante requerida pelos fluxos de tráfego DiffServ entre os ISPs. Dessa forma, quando uma determinada rota não puder ser utilizada por insuficiência de banda passante, o sistema deve detectar outra rota que possua recursos suficientes, sobretudo banda passante e retardo máximo, com base nas informações dos BBs. Essas rotas são determinadas em função das classes de tráfego DiffServ e não em função de um fluxo de tráfego particular. Na troca de informações entre os ISPs, deve ser levado em conta o fato de que entre domínios DS uma única classificação de *codepoint* deve ser utilizada, embora cada domínio DS tenha a possibilidade de empregar internamente os seus próprios mecanismos e *codepoints* de classificação.

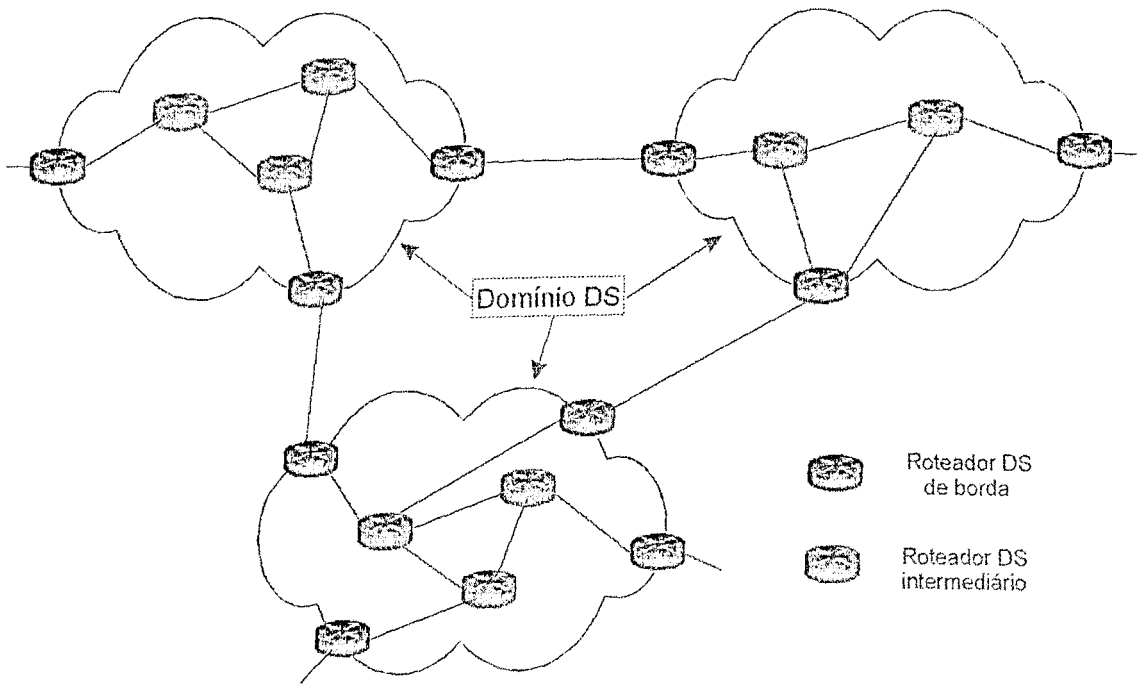

Figura 4.14 – Rede de Serviços Diferenciados com três domínios distintos.

Nas redes DiffServ as linhas de transmissão existentes podem ser divididas virtualmente em várias linhas, denominadas de VLLs (*Virtual Leased Lines*), de modo que uma VLL seja totalmente independente da outra (analogamente à categoria de serviço CBR do ATM). Assim, fazendo-se uso dessa funcionalidade, os ISPs podem prover suporte às

chamadas VPNs (*Virtual Private Networks*), as quais visam garantir requisitos fundamentais aos serviços dos usuários, tais como disponibilidade de acesso à linha de transmissão, banda passante e retardo adequados. Nesse sentido, esse tipo de implementação poderá vir a substituir, com muito mais eficiência, as atuais linhas dedicadas a um único usuário, uma vez que as corporações usuárias da Internet se beneficiarão não apenas de custos mais baixos mas também do fato de somente serem taxadas quando efetivamente utilizarem tais linhas.

Outro assunto de grande importância relacionado ao uso das VPNs é a segurança, embora o mesmo não esteja ainda sendo abordado de forma relevante pela comunidade cientifica envolvida. Por isso, a solução que têm sido empregada usualmente é a codificação dos dados na fonte e a correspondente decodificação no receptor.

4.4.1 O *codepoint*

O *codepoint* é armazenado nos campos ToS e CoS do cabeçalho dos pacotes IPv4 e IPv6, respectivamente, os quais são divididos em um campo denominado de DSCP (6 bits) e outro campo não utilizado atualmente, o CU (*Currently Unused*), como indicado na figura 4.15.

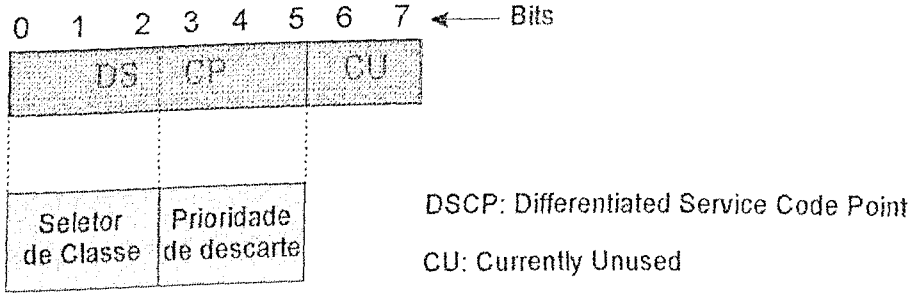

Figura 4.15 – Definição dos campos do byte usado para a Diferenciação dos Serviços.

O *codepoint* DSCP é dividido em duas partes de 3 bits cada. A primeira parte é chamada de Seletor de Classe e representa a classe de tráfego (EF, AF ou BE) do pacote e, por isso, a mesma não pode ser alterada por nenhum roteador no interior da rede. A segunda

parte, denominada de Prioridade de Descarte, define se o pacote será descartado com prioridade alta, média ou baixa. Por fim, cabe lembrar que o tráfego não é diferenciado em termos de fluxos de tráfego simples, mas sim considerando agregados de fluxos divididos em classes de tráfegos.

4.4.2 *Expedited Forwarding*

O serviço EF, também denominado de grupo PHB EF, pode ser utilizado na implementação de um domínio DS que apresente as seguintes características: valores pequenos de perda, retardo e *jitter*; banda passante garantida e serviço fim-a-fim. Desse modo, esses serviços são vistos pelos clientes, nas extremidades da rede, como uma conexão ponto-a-ponto ou uma VLL. Graças a isso, o EF destina-se às aplicações em tempo real que apresentem uma taxa máxima ou constante. Em função de suas características, este serviço foi denominado inicialmente de *Premium Service* [20].

O fato de se prover valores baixos de perda, retardo e *jitter* com o serviço EF, implica que com este serviço o tráfego não é submetido a enfileiramentos dentro da rede, ou então o mesmo ocorre de forma insignificante. O enfileiramento aumenta quando a taxa do tráfego que entra em um determinado nó excede a taxa de transmissão do mesmo. Por isso, a fim de garantir que os pacotes EF sejam entregues em tempo hábil, considerando-se um tráfego em tempo real, o grupo PHB EF é definido de forma que a taxa de envio de pacotes de qualquer nó DiffServ deve ser igual ou maior à sua taxa configurada.

O *codepoint* recomendado pela IETF para o grupo PHB EF é mostrado na tabela 4.1. Este *codepoint* não sobrepõe nenhum outro grupo de PHB.

Em condições normais de operação, considera-se que não há congestionamento para o tráfego EF. Contudo, os mecanismos do serviço EF são projetados para descartar os pacotes em excesso.

O serviço EF é designado para formar uma VLL a fim de substituir a tecnologia de linhas alugadas dedicadas existentes atualmente. De acordo com o que foi mencionado anteriormente, com o EF uma linha alugada poderia ser dividida em diversas VLLs e ser compartilhada por várias companhias usuárias da Internet. Exemplos de categorias de aplicativos para o serviço EF são: voz sobre IP (telefonia) e vídeo sobre IP (teleconferência).

	PHB
Expedited Forwarding	101110

Tabela. 4.1 – *Codepoint* do serviço EF.

4.4.3 *Assured Forwarding*

O grupo PHB AF [21] representa uma maneira pela qual um provedor de serviço pode oferecer diferentes níveis de garantia de encaminhamento aos pacotes IP recebidos de um usuário conectado a um domínio DS. Assim, são definidas quatro classes de serviço, e cada classe possui uma determinada quantidade de recursos, tais como tamanho de *buffer* e banda passante, destinados ao envio dos seus respectivos pacotes.

Considerando-se as ações de condicionamento de tráfego que podem ser impostas pelo serviço AF, tem-se: a moldagem de tráfego, o descarte de pacotes e a elevação do nível de prioridade de descarte dos pacotes. Para cada classe de tráfego AF são possíveis três níveis de prioridade de descarte (tabela 4.2), de modo que os nós, sob congestionamento, procurem proteger os pacotes com valores de prioridade de descarte menores em detrimento dos pacotes que possuem valores de prioridades de descarte maiores. Dessa forma, uma implementação AF deve tratar os congestionamentos de longa duração por meio do descarte de pacotes e as de curta duração (rajadas) com o enfileiramento.

Portanto, a forma como os nós de uma rede DiffServ encaminham os pacotes AF depende dos seguintes fatores:

1. Quantidade de recursos para encaminhamento reservada à classe AF
2. Carga corrente da classe AF e, se houver congestionamento dentro dessa classe
3. Nível de prioridade de descarte do pacote

Os *codepoints* recomendados para o PHB AF são mostrados na tabela 4.2 e, estes não sobrepõem nenhum outro grupo de PHB.

	Classe 1	Classe 2	Classe 3	Classe 4
Prioridade Baixa de descarte	001010	010010	011010	100010
Prioridade Média de descarte	001100	010100	011100	100100
Prioridade Alta de descarte	001110	010110	011110	100110

Tabela. 4.2 – *Codepoints* do serviço AF.

Uma aplicação típica do serviço AF é representada por uma empresa que utiliza a Internet para interconectar suas diversas filiais distribuídas geograficamente e deseja obter uma garantia de que seus pacotes IP sejam transmitidos com prioridade alta, desde que o tráfego agregado de cada uma das localidades envolvidas não exceda o seu respectivo perfil de tráfego. Adicionalmente, é desejável que essas localidades possam exceder os seus respectivos perfis de tráfego, ainda que esse tráfego em excesso não seja transmitido com prioridade alta.

Também é importante o fato de que a rede não deve reordenar os pacotes que pertençam ao mesmo fluxo, independentemente dos mesmos estarem ou não de acordo com o perfil de tráfego. Exemplos de aplicações desse serviço são aquelas sensíveis ao tempo de resposta, tais como transações bancárias e acesso a bancos de dados.

4.4.4 *Best Effort*

O serviço BE é conhecido como o serviço tradicional das redes IP e que não oferece nenhuma garantia de encaminhamento de pacotes, tais como retardo e perda mínimos. Nesse sentido, os pacotes DiffServ que não são marcados com *codepoints* EF ou AF, são enviados como tráfego BE. A tabela 4.3 apresenta o *codepoint* utilizado pelo PHB BE.

	PHB
Best Effort	000000

Tabela. 4.3 – *Codepoint* do serviço BE.

Portanto, de acordo com o exposto, o tráfego BE não recebe nenhuma das vantagens oferecidas pela arquitetura DiffServ, exceto que o mesmo é enfileirado numa fila exclusiva dos demais tráfegos; porém, quando tal fila está cheia, os pacotes são descartados. Quanto às aplicações relacionadas a este serviço, as mais utilizadas são o FTP (*File Transfer Protocol*) e a WWW (*World Wide Web*).

4.5 CONCLUSÕES

A rede DiffServ representa um marco considerável no modo de se prover QoS na Internet, visto que a mesma preconiza a máxima simplificação nos nós intermediários (internos) da rede, o que facilita a implementação da mesma em larga escala. Adicionalmente, essa arquitetura possibilita a interoperação com nós não DS compatíveis e, desse modo, permite que a migração para essa tecnologia possa ser implementada de forma gradativa e sem maiores complicações econômicas.

Nesse sentido, os roteadores na borda da rede devem proceder ao policiamento e correspondente condicionamento do tráfego que entra na rede, com o objetivo de garantir que no interior da mesma só haja tráfego que já tenha sido moldado de acordo com os seus respectivos SLAs.

Os elementos que formam a arquitetura DiffServ podem ser implementados de várias formas. No entanto, a codificação a ser empregada, via *codepoint,* para fins de comunicação entre os domínios DS diferentes deve ser preservada. Para isso, a IETF define os *codepoints* a serem empregados nesses casos. Por outro lado, as informações referentes à quantidade de banda passante disponível em cada domínio DS devem ser fornecidas aos demais domínios DS por meio dos *Bandwidth Brokers.*

Cabe frisar que a possibilidade de compartilhar uma linha de transmissão entre várias VLLs dentro do ambiente DiffServ, seguramente é atrativo às empresas que empregam a Internet como via para suas Intranets. As VLLs dispensam manutenção própria por parte dessas empresas e, ainda, permitem o compartilhamento das redes IPs públicas, o que significa melhor aproveitamento do meio de transmissão e, por conseguinte, menor custo.

No entanto, o nível de QoS oferecido pelos ISPs depende do tipo de serviço que está sendo requisitado pelo usuário. O qual, por sua vez, depende fundamentalmente das características do tráfego que suas aplicações estão gerando. Por outro lado, as redes DiffServ vislumbram a possibilidade de oferecerem um nível mínimo de QoS a todos os tipos de tráfegos submetido à mesma e, por isso, caracterizam-se como uma tecnologia promissora a ser empregada na Internet do futuro.

CAPÍTULO V

ALGORITMOS TCP Reno E TCP Vegas NUMA REDE DE SERVIÇOS DIFERENCIADOS

5.1 INTRODUÇÃO

De acordo com o que foi apresentado nos capítulos anteriores, o algoritmo RIO é caracterizado como um mecanismo de controle de congestionamento que atua no interior da rede DiffServ. Em contrapartida, os algoritmos TCP realizam o controle de fluxo nos *hosts*, ou seja, nas extremidades da rede. Por isso, o controle de congestionamento completo nessas redes é resultado da interação entre esses dois mecanismos [39]. Nesse sentido, esse capítulo se dedica inicialmente ao estudo do mecanismo RIO e de dois importantes algoritmos TCP, o TCP Reno e o TCP Vegas e, em seguida é abordada a proposta de estudo dessa dissertação, a qual trata da avaliação de desempenho desses dois algoritmos TCP num ambiente DiffServ.

Considerando-se os mecanismos de controle de fluxo implementados no protocolo TCP tem-se que, o mais popular deles é o Tahoe que foi desenvolvido por Van Jacobson em 1988 [38] como uma evolução do TCP padrão apresentado na sub-seção 2.5.5 do capítulo II. Dois anos depois de sua criação, o Tahoe foi aprimorado, com a inclusão do algoritmo *Fast Recovery* [40], originando assim o TCP Reno [46]. Por isso, o TCP Reno apresenta melhor desempenho quanto à utilização de banda passante e ainda retransmite menos dados que o seu predecessor. No entanto, em função do fato de que o mesmo causa perda de dados na rede com o propósito de medir a disponibilidade de banda passante da mesma, o TCP Reno provoca valores elevados e variáveis de RTTs, além de agressividade na rede (TCP Reno tende a obter mais banda passante do que outras opções do TCP, considerando-se uma

situação de interoperação) [37]. Tais características podem degradar significativamente o desempenho da rede em situações adversas.

Nesse sentido, Brakmo e Peterson [36,43] propuseram um novo mecanismo de controle de fluxo para o TCP, denominado de TCP Vegas, com base em algumas modificações no comportamento do transmissor do TCP Reno. Este outro mecanismo é capaz de interoperar com qualquer outra implementação válida do TCP e, é ressaltado em [43] que o mesmo consegue obter entre 37% e 71% melhor vazão do que o TCP Reno, retransmitir apenas 20% a 50% da quantidade de pacotes retransmitida pelo TCP Reno e, ainda, utilizar mais eficientemente a banda passante da rede.

Dessa forma, a fim de fundamentar os conceitos a serem tratados nesse trabalho, retrata-se nas três sub-seções subseqüentes o funcionamento do algoritmo RIO e dos algoritmos TCP Reno e TCP Vegas, respectivamente. Maior detalhamento é empregado na descrição do TCP Vegas, visto que esse algoritmo apresenta vários mecanismos novos em relação ao TCP Reno, que por vez já foi exaustivamente estudado em diversas pesquisas anteriores.

5.2 ALGORITMO RIO

Este algoritmo se baseia no algoritmo RED e possui dois conjuntos de parâmetros destinados a dois tipos de pacotes: pacotes em conformidade e não em conformidade. Tais conjuntos de parâmetros são denotados por $(min\_in, max\_in, P_{max\_in})$ e $(min\_out, max\_out, P_{max\_out})$. Sendo min_in e max_in os limiares mínimo e máximo, respectivamente, para os pacotes em conformidade, enquanto que P_{max_in} refere-se à máxima probabilidade com a qual esses pacotes podem ser descartados. Analogamente, min_out e max_out representam os limiares mínimo e máximo, respectivamente, para os pacotes não em conformidade, e P_{max_out} refere-se à máxima probabilidade com a qual esses pacotes podem ser descartados [44]. Com base nisso, o funcionamento do algoritmo RIO é apresentado a seguir.

Ao receber um pacote, o algoritmo RIO estima o valor de dois parâmetros: *avg_in_q* e *avg_q*. Os quais indicam o tamanho médio da fila para os pacotes em conformidade e o tamanho médio da fila total, respectivamente. Assim, o recebimento de um pacote em conformidade irá contribuir para a estimação dos dois parâmetros citados, enquanto que o recebimento de um pacote não em conformidade contribuirá somente com a estimação do parâmetro *avg_q*.

A probabilidade de descarte de cada pacote recebido (p) é calculada em função dos valores correntes de *avg_in_q* e *avg_q* de modo que para os pacotes em conformidade, essa probabilidade é calculada como indicado na equação 5.1.

$$p = P_{\max\_in} * \frac{(avg\_in\_q - \min\_in)}{(\max\_in - \min\_in)} \qquad (5.1)$$

A idéia principal da equação 5.1 é a de que os pacotes em conformidade devem receber prioridade sobre os pacotes não em conformidade. Portanto, conforme pode ser observado nesta equação, os pacotes em conformidade serão enfileirados ou descartados em função somente da quantidade de pacotes em conformidade recebidos recentemente pelo algoritmo RIO e, assim, não serão afetados pelos pacotes não em conformidade, tampouco pelo número total de pacotes na fila (pacotes em conformidade e não em conformidade). Cabe observar que valores negativos para a equação 5.1 implicam em não descarte de pacotes.

Em relação aos pacotes não em conformidade, a probabilidade de descarte é calculada de acordo com a equação 5.2. Tal equação leva em conta o fato de que os pacotes não em conformidade representam o tráfego de menor prioridade e, por isso, os mesmos são dependentes dos pacotes em conformidade no que diz respeito ao enfileiramento. Portanto, a probabilidade de descarte dos pacotes não em conformidade depende não apenas dos outros pacotes não em conformidade na fila, mas também da quantidade de pacotes em

conformidade enfileirados. Tal fato justifica o uso do parâmetro *avg_q* (tamanho médio da fila total) no cálculo da probabilidade de descarte dos pacotes não em conformidade, como mostrado na equação 5.2.

$$p = P_{\max\_out} * \frac{(avg\_q - \min\_out)}{(\max\_out - \min\_out)} \tag{5.2}$$

Com o propósito de elucidar melhor o princípio de funcionamento do algoritmo RIO, apresenta-se a seguir um ilustração gráfica das fases de operação do mesmo, por meio da figura 5.1, em que são apresentadas as quatro fases de estado de congestionamento desse algoritmo em função do comprimento médio da fila (avg_q). Faz-se a seguir uma descrição de cada uma das fases envolvidas.

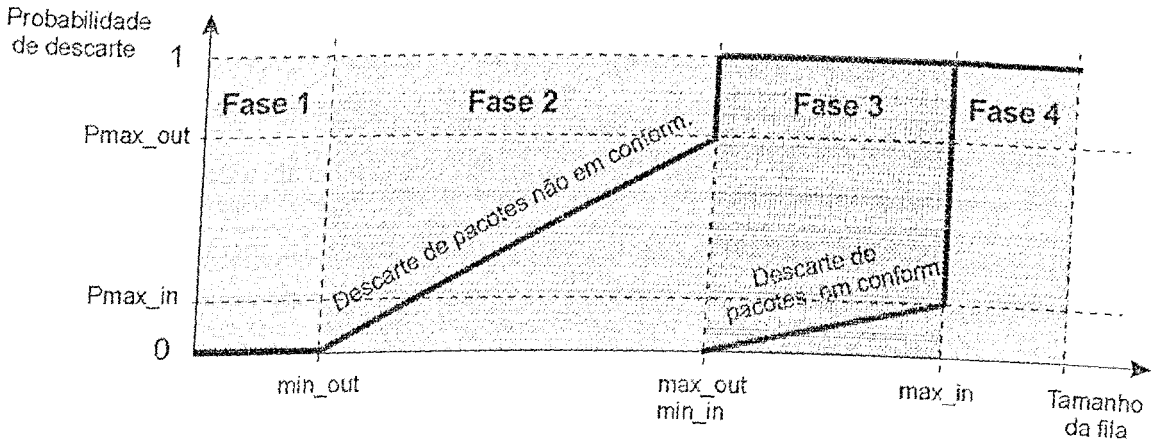

Figura 5.1 – Algoritmo RIO.

Fase livre de congestionamento (Fase 1)

Nesta fase, o mecanismo RIO está operando de forma confortável, uma vez que as quantidades de pacotes em conformidade e não em conformidade estão bem abaixo da sua capacidade máxima. Nessa situação o mecanismo RIO vê apenas valores muito pequenos de tamanho instantâneo e médio de fila e, graças a isso, nenhum pacote é descartado.

Fase de previsão de congestionamento (Fase 2)

Nesta fase, o mecanismo suspeita de que a fila poderia ficar cheia e, então, o mesmo começa a descartar pacotes como um sinal de que a rede está congestionada; contudo, apenas os pacotes não em conformidade são descartados. Durante essa fase, os pacotes em conformidade apenas vêem uma fila instantânea relativamente longa, mas jamais são descartados.

Fase de alarme de congestionamento (Fase 3)

Nesta fase, todos os pacotes não em conformidade são descartados e, ainda, os pacotes em conformidade também começam a ser descartados com o objetivo de manter o tamanho da fila num patamar razoável. Esta é uma fase indesejável pelos ISPs, porque a mesma compromete a garantia dos contratos de tráfego (SLAs) dos mesmos, visto que nessa fase os pacotes em conformidade não são protegidos contra o descarte.

Fase de controle de congestionamento (Fase 4)

Nesta fase, o sistema está congestionado e, por isso, o mecanismo RIO descarta os dois tipos de pacotes com probabilidade máxima (p=1). Nessas circunstâncias, esse mecanismo não consegue mais cumprir o seu objetivo principal que se refere à diferenciação entre os dois tipos de pacotes (pacotes em conformidade e não em conformidade) a fim de controlar o congestionamento na rede. Como conseqüência, o mecanismo RIO degrada o seu funcionamento ao nível de uma fila simples, FIFO, o que causa outros desdobramentos indesejáveis, tais como: o descarte de múltiplos pacotes oriundos da mesma fonte, o efeito da sincronização global (Capítulo II, sub-seção 2.5.5), etc. Portanto, se tal mecanismo operar continuamente nessa fase, pode-se assegurar que o sistema está sub-dimensionado ou que os parâmetros dos mecanismos condicionador e/ou RIO, pertencentes à rede DiffServ, não estão ajustados de forma adequada.

Dentre todas essas fases, a fase 2 é a desejável para um roteador operar, visto que na mesma tanto o tamanho instantâneo da fila como o seu tamanho médio são pequenos e, ainda, a utilização do enlace (canal de transmissão) é alta. Nessa fase apenas os pacotes não em conformidade são descartados, o que não compromete a garantia dos SLAs por parte dos ISPs.

Por outro lado, quando o roteador está operando na fase 1, nota-se que embora um pequeno congestionamento seja visto pelo mesmo, a capacidade do enlace não é aproveitada adequadamente.

Portanto, quando o tráfego submetido à rede puder ser predito, os ISPs devem configurar os seus respectivos sistemas de forma que estes evitem a operação nas fases 3 e 4 e, consequentemente, operem na maioria dos casos nas fases 1 e 2.

5.3 TCP Reno

Com o controle de congestionamento TCP Reno o tamanho da janela CWND muda continuamente. A CWND cresce até que ocorra perda de um segmento (ou pacote) e quando isso ocorre, o TCP infere que a rede está congestionada e, assim, o mesmo reduz a CWND ao tamanho de um segmento (janela inicial). O TCP Reno apresenta duas fases em que a CWND é incrementada: fase *Slow Start* e fase *Congestion Avoidance*. Dessa forma, quando um segmento ACK é recebido pelo TCP transmissor no instante $t+t_{\Delta}$, a $CWND(t+t_{\Delta})$ é atualizada, levando em consideração a CWND(t) (CWND no instante t), da seguinte forma:

$$CWND(t+t_{\Delta}) = \begin{cases} (Slow\ Start) \\ CWND(t)+1, \ se \ \ CWND(t) < SSTH; \\ (Congestion\ Avoidance) \\ CWND(t)+\dfrac{1}{CWND(t)}, \ se \ \ CWND(t) \geq SSTH; \end{cases} \tag{5.3}$$

Onde SSTH (SSTHRESH) representa o valor de limiar no qual o TCP muda da fase *Slow Start* para fase *Congestion Avoidance* . O que determina se a CWND será incrementada a partir da fase *Slow Start* ou a partir da fase *Congestion Avoidance* é a forma como a perda de um segmento é detectada. Essa detecção pode ocorrer por *timeout* (tempo limite para recepção de um ACK) ou pelo mecanismo *Fast Retransmit* apresentado na sub-seção seguinte.

Dessa forma, quando é detectada a perda de um segmento por *timeout*, os parâmetros CWND(t) e SSTH são atualizados da seguinte forma:

$$SSTH = \frac{CWND}{2}, \quad CWND(t) = 1 \tag{5.4}$$

As equações mostradas em 5.4 indicam que o TCP Reno entra novamente na fase *Slow Start* quando ocorre perdas por *timeout*. Entretanto, nem sempre essa é a melhor opção de retransmissão para o TCP e, por isso, os mecanismos *Fast Retransmit* e *Fast Recovery* foram implementados no mesmo.

5.3.1 Mecanismos *Fast Retransmit* e *Fast Recovery*

A fim de minimizar o tempo gasto na detecção de perdas de segmentos (também na detecção de segmentos fora de seqüência), e na recuperação da taxa de transmissão desses segmentos, o TCP Reno emprega os mecanismos *Fast Retransmit e Fast Recovery* [40]. Com estes mecanismos, as perdas de segmentos, desde que não ocorram em seqüência longas, podem ser recuperadas sem a necessidade de espera pelo tempo de *timeout*.

O mecanismo *Fast Retransmit* funciona da seguinte forma: toda vez que o receptor TCP detecta a perda de um segmento (ou segmento fora de seqüência), este repete o último ACK enviado a cada novo segmento recebido. Assim, quando o receptor detectar o "n-ésimo"

(usualmente o terceiro) ACK duplicado, o mesmo entra na fase *Fast Recovery*, a qual implica em atualizar a CWND e o SSTH da seguinte forma:

$$SSTH = \frac{CWND}{2}, \quad CWND(t) = SSTH \tag{5.5}$$

Conforme pode ser observado na figura 5.2a, o mecanismo de controle de fluxo do TCP Reno causa uma oscilação periódica no tamanho da CWND, em função da atualização contínua aplicada à mesma. Essa oscilação faz com que os RTTs dos segmentos também oscilem, o que provoca valores de *jitter* elevados além de uso ineficiente da banda passante da rede em função do grande número de retransmissões de segmentos repetidos subseqüentemente à ocorrência das perdas

Portanto, a fase *Fast Recovery* significa entrar direto na fase *Congestion Avoidance* após uma retransmissão. A figura 5.2a apresenta um exemplo típico do comportamento da CWND(t) no TCP Reno.

Outro exemplo que mostra mais detalhadamente a atuação dos mecanismos *Fast Retransmit* e *Fast Recovery* é apresentado na figura 5.3. Por esta figura pode-se observar que esses mecanismos podem causar uma recuperação ineficiente, quanto à taxa de transmissão, em situações em que há perdas de múltiplos pacotes. Nesse exemplo são representadas as perdas de três pacotes fim-a-fim. Após a segunda ocorrência da fase *Fast Retransmit*, o transmissor TCP não recebe mais nenhum pacote de modo que ative a contagem dos pacotes ACK duplicados. Por isso, a retransmissão é forçada por meio do *timeout*, o qual ocorre com um tamanho de janela muito pequeno e, por conseqüência, a recuperação na fase *Congestion Avoidance* ocorre de forma muito lenta, especialmente em redes com retardos elevados.

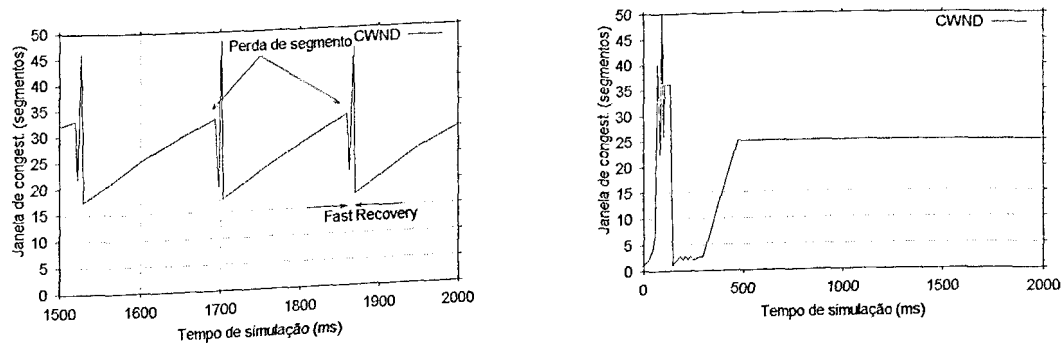

Figura 5.2 – Variação da Janela de congestionamento (CWND) para os algoritmos TCP: a) TCP Reno e b) TCP Vegas.

Portanto, embora o algoritmo TCP Reno possa oferecer um desempenho razoável na maioria das situações, existem casos particulares nos quais o mesmo apresenta sérias deficiências e, por isso, conforme citado anteriormente, muitos estudos têm sido desenvolvidos no sentido de se solucionar tais deficiências desse algoritmo.

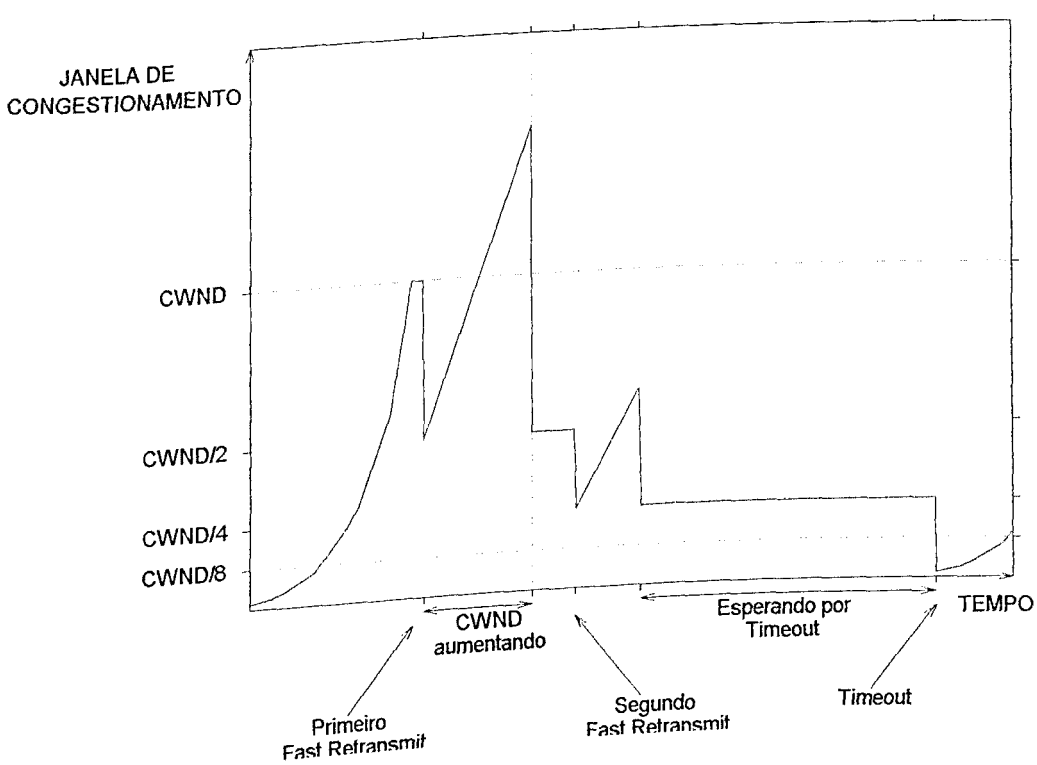

Figura 5.3 – Mecanismos *Fast Retransmit* e *Fast Recovery*.

5.4 TCP Vegas

De acordo com o que foi apresentado anteriormente, com o TCP Reno, o tamanho da janela CWND é incrementado até que ocorram perdas de segmentos por congestionamento na rede e, só então o referido tamanho é decrementado. A desvantagem desse algoritmo é que o mesmo provoca degradação na vazão da conexão, visto que as próprias fontes podem causar congestionamento na rede ao aumentarem "indefinidamente" suas respectivas CWNDs. Com base nesse fato, o TCP Vegas foi proposto com o intuito de manter o tamanho da CWND estável e, assim, evitar perdas excessivas na rede.

O TCP Vegas emprega um mecanismo diferente para detectar congestionamentos na rede. Com esse mecanismo, o controle da CWND é realizado por meio da monitoração das variações sofridas pelos RTTs dos segmentos enviados anteriormente pela conexão. Dessa forma, se os RTTs aumentarem, o TCP Vegas deduz que a rede começou a ficar congestionada e, em conseqüência, o mesmo diminui o valor da CWND da conexão. Por outro lado, se os RTTs diminuirem, a CWND é incrementada em resposta ao entendimento de que a rede está livre de congestionamento. Portanto, numa situação ideal a CWND converge para um valor apropriado como indicado na figura 5.2b, e a vazão da conexão não sofre degradação considerável, como ocorre com o TCP Reno (figura 5.2a).

Nesse sentido, com o intuito de aumentar a vazão na rede e ao mesmo tempo diminuir as perdas, o TCP Vegas faz uso de três técnicas fundamentas, as quais apresentam as seguintes características:

- A primeira, possibilita maior precisão na determinação do instante em que os pacotes descartados (perdidos) devem ser retransmitidos.
- A segunda, oferece ao TCP a condição de prever congestionamentos na rede e, então, ajustar a sua taxa de transmissão adequadamente.

- A terceira, modifica o mecanismo *Slow Start* do TCP de forma a evitar perdas de pacotes enquanto o mesmo tenta encontrar a banda passante disponível na rede (via aumento da CWND).

Com o propósito de elucidar o funcionamento do TCP Vegas, e também estabelecer as diferenças do mesmo em relação ao TCP Reno, é apresentado a seguir um estudo pormenorizado de cada uma dessas técnicas.

5.4.1 Novo Mecanismo de Retransmissão

No algoritmo TCP Reno, o RTT é medido por meio de um temporizador de baixa resolução (*coarse-grained*), usualmente de 500 ms, o que faz com que o cálculo do RTT estimado [51] não seja muito preciso. E, ainda, essa baixa resolução influencia também a freqüência com que o TCP verifica o *timeout* para os segmentos.

Recordando o que foi dito anteriormente, o TCP Reno retransmite um segmento sempre que ocorre um *timeout* de baixa resolução, ou quando o mesmo recebe n (normalmente 3) segmentos ACK duplicados (*Fast Retransmit*). O *host* receptor do TCP Reno envia um ACK duplicado sempre que o mesmo recebe um segmento que não pode ser reconhecido porque os dados anteriores ainda não foram todos recebidos. Tal situação ocorre quando há perda de segmentos entre o transmissor e o receptor, ou recebimento de pacote fora de seqüência.

Por exemplo, se o TCP Reno recebe o pacote de número 2 e o pacote 3 é perdido; então o mesmo gera um ACK duplicado para o pacote 2 no instante em que o pacote 4 é recebido, novamente o mesmo ACK é enviado quando o pacote 5 é recebido e assim por diante. Dessa forma, quando o transmissor receber o terceiro ACK duplicado para o pacote 2 (enviado no recebimento do pacote 6), o mesmo retransmite o pacote 3.

Alternativamente, o TCP Vegas modifica o mecanismo de retransmissão do TCP Reno de modo que o mesmo funcione como descrito a seguir.

O algoritmo TCP Vegas calcula o RTT de cada segmento quando o mesmo recebe o ACK de cada um. Para isso, ele utiliza o relógio do sistema (*clock system*) e também o *timestamp* (marca de tempo) do referido segmento. Da comparação entre ambos resulta o RTT, o qual é mais preciso do que o obtido pelo TCP Reno, porque é calculado a intervalos menores. Assim, o TCP Vegas utiliza esse RTT, na realidade um valor estimado de RTT [51], a fim de verificar a necessidade de retransmissão de um segmento nas duas situações que se seguem:

a) Quando um ACK duplicado é recebido, o TCP Vegas verifica se a diferença entre o tempo corrente e o *timestamp* do referido segmento é maior do que o valor de *timeout* de alta resolução (calculado com base nos RTTs de cada segmento). Caso positivo, então o TCP Vegas retransmite o segmento sem precisar esperar por n (usualmente 3) ACKs duplicados.

b) Quando um ACK não duplicado é recebido, se este for o primeiro ou o segundo após uma retransmissão (os quais possuem alta probabilidade de serem os primeiros ACKs de uma seqüência de ACKs duplicados a serem recebidos em função de segmentos perdidos); o TCP Vegas verifica novamente se o seu RTT é maior do que o valor de *timeout* (de alta resolução). Se isso ocorrer, então o TCP Vegas retransmite o segmento. Esse procedimento assegura que qualquer segmento que tenha sido perdido antes da retransmissão, seja retransmitido rapidamente sem a necessidade de espera por um ACK duplicado.

Comparando-se a abordagem do item (a) como o mecanismo de retransmissão do TCP Reno tem-se que, em muitos casos, as perdas são tão elevadas ou a CWND é tão pequena que o transmissor jamais recebe três ACKs duplicados e, nesse caso, o TCP Reno só pode contar com o *timeout* de baixa resolução mencionado anteriormente, o que pode degradar sensivelmente o desempenho da rede.

Em relação ao TCP Vegas, nota-se que o mesmo trata o recebimento de alguns ACKs como um disparo para a verificação da condição que indica se deveria ocorrer um *timeout* (de alta resolução) ou não. Por outro lado, o TCP Vegas ainda mantém o *timeout* de baixa resolução para os casos em que esse algoritmo possa falhar. É importante observar ainda que a CWND deve ser reduzida somente em função de perdas que ocorrem na taxa de transmissão corrente e não devido a perdas anteriores que tenham ocorrido a taxas de transmissão maiores. As perdas que ocorreram antes do último decremento da CWND não indicam necessariamente que a rede está congestionada para o tamanho de CWND corrente e, portanto, não significam que a CWND deve ser reduzida novamente.

No TCP Reno é possível ocorrer mais de um decremento da CWND durante um intervalo de RTT. Contrariamente, o TCP Vegas somente decrementa a CWND se o segmento retransmitido foi previamente enviado após o último decremento da mesma. Tal mudança é necessária em função do fato de que o TCP Vegas detecta as perdas muito mais rapidamente do que o TCP Reno.

5.4.2 Mecanismo *Congestion Avoidance*

Em conformidade com o que foi mencionado anteriormente, a abordagem utilizada pelo TCP Vegas opera com base na observação das alterações sofridas pela taxa de vazão da conexão. Esse algoritmo compara continuamente a taxa de vazão medida ou real com a esperada. O TCP Vegas explora a simples idéia de que a quantidade de bytes em trânsito é diretamente proporcional à vazão esperada e, portanto, quando a CWND aumenta, ocasionando aumento da quantidade de bytes em trânsito, a vazão da conexão também deveria aumentar.

Desse modo, o TCP Vegas funciona com base no principio de se medir e controlar a quantidade de dados extras que a conexão tem em trânsito. Onde dados extras dizem respeito aos dados que não teriam sido enviados se a banda passante da conexão correspondesse

exatamente à banda passante disponível na rede. Enfim, o objetivo do TCP Vegas é manter a quantidade "correta" de dados extras na rede, uma vez que se a conexão (de fato a fonte dessa conexão) enviar muitos dados extras, ela causará congestionamento na rede; e no caso oposto, ela não conseguirá responder rápido o suficiente aos transientes de aumento de banda passante disponíveis na rede.

Portanto, as ações realizadas pelo mecanismo *Congestion Avoidance* do TCP Vegas se baseiam nas alterações observadas na quantidade estimada de dados extras na rede, e não somente nas perdas de segmentos como o faz o TCP Reno. A seguir esse algoritmo é apresentado em detalhes.

As três principais fases de operação desse mecanismo são:

a) Define-se, primeiramente, o parâmetro *base_rtt* para a conexão, o qual representa o RTT de um segmento quando a conexão não está congestionada. Na prática o TCP Vegas ajusta o *base_rtt* para ser o valor mínimo dentre todos os RTTs medidos, o que corresponde normalmente ao RTT do primeiro segmento enviado pela conexão, antes das filas dos roteadores aumentarem em função do tráfego gerado pela própria conexão. Assim, desde que a conexão não esteja sob congestionamento, a vazão esperada (VE) para a mesma é dada por:

$$VE = \frac{CWND}{base\_rtt} \tag{5.6}$$

Onde, a CWND é considerada aqui igual à quantidade de bytes em trânsito a fim de facilitar a explicação.

b) Calcula-se, a seguir, a vazão real (VR) da conexão. Para isso, o TCP Vegas registra o instante em que o segmento é enviado e a quantidade de bytes transmitidos até o recebimento do ACK do mesmo, quando então o TCP Vegas determina o seu RTT.

Graças a essas informações medidas, o TCP Vegas pode então determinar a vazão real da conexão, que é calculada por meio da divisão da quantidade de bytes enviados pelo RTT do segmento. Este cálculo é realizado uma vez por RTT, e pode ser resumido na equação 5.7 a seguir.

$$VR = \frac{CWND(t)}{rtt} \tag{5.7}$$

c) Por fim, compara-se as vazões real e esperada, calculando-se Diff = VE – VR, e atualizando-se a CWND adequadamente. Diff é sempre maior ou igual a zero, visto que a condição VR>VE implica na atualização do parâmetro *base_rtt* com o valor do último RTT medido, o que faz *base_rtt*= rtt e, consequentemente, VE=VR (equações 5.6 e 5.7). O TCP Vegas define também dois parâmetros de limiar, α e β (sendo $\alpha<\beta$), que correspondem às situações de se ter muitos e poucos dados extras na rede, respectivamente. Usualmente esses parâmetros são especificados em bytes. Assim, nos instantes em que Diff<α/*base_rtt*, o TCP Vegas aumenta a CWND linearmente durante o próximo RTT; e quando Diff>β/*base_rtt*, a CWND é decrementada linearmente durante o próximo RTT. Por fim, quando o Diff está entre esses dois extremos, o TCP Vegas mantém a CWND inalterada.

Resumindo-se o que foi dito, na fase *Congestion Avoidance* o tamanho da janela CWND, no intante $t+t_{\Delta}$, é atualizado da seguinte forma:

$$CWND(t+t_{\Delta}) = \begin{cases} CWND(t)+1, & se \quad Diff < \dfrac{\alpha}{base\_rtt}; \\ CWND(t), & se \quad \dfrac{\alpha}{base\_rtt} \le Diff \le \dfrac{\beta}{base\_rtt}; \\ CWND(t)-1, & se \quad \dfrac{\beta}{base\_rtt} < Diff; \end{cases} \tag{5.8}$$

Sendo,

$$Diff = VE - VR = (\frac{CWND(t)}{base\_rtt}) - (\frac{CWND(t)}{rtt}) \quad (5.9)$$

A Equação 5.8 utilizada pelo TCP Vegas possibilita constatar que se os RTTs dos segmentos permanecerem estáveis, de forma que Diff oscile dentro dos limites especificados pelos parâmetros α e β (divididos por *base_rtt*), o tamanho da janela se mantém inalterado. Isto pode ser visto na figura 5.2b, onde a CWND converge efetivamente para um valor em torno de 25 segmentos.

Desse modo, quanto mais distantes as vazões real e esperada estiverem uma da outra, mais congestionada estará a rede, o que indica que o transmissor deve reduzir a sua taxa de transmissão. Essa redução é ativada pelo limiar β/*base_rtt*. Por outro lado, quanto mais próximas uma da outra estiverem essas vazões, a conexão está sob risco de não estar utilizando toda a banda passante disponível na rede e, então, o transmissor precisa aumentar a sua vazão. Esse aumento é ativado pelo limiar α/*base_rtt*. O objetivo final é manter a quantidade de bytes extras na rede entre os limiares α e β.

5.4.3 Mecanismo *Slow Start* modificado

Mais uma diferença do TCP Vegas em relação ao TCP Reno diz respeito ao mecanismo *Slow Start*. O TCP Reno dobra o tamanho da CWND a cada RTT até que haja perdas na rede, na tentativa de encontrar a banda passante disponível na rede e, isso, causa perdas da ordem de metade do tamanho da CWND corrente. Diferentemente, o TCP Vegas somente admite o crescimento exponencial da CWND em RTTs alternados, de forma que a CWND fique fixa por mais tempo, possibilitando, assim, a validação da comparação entre as vazões esperada e real citadas anteriormente. Medir a vazão real com a CWND fixada é

necessário para garantir que a vazão medida represente realmente a banda passante disponibilizada à conexão naquele instante.

Dessa forma, quando a vazão real cai abaixo da esperada por um determinado valor de limiar denominado de γ, o TCP Vegas muda da fase *Slow Start* para a fase linear de incremento/decremento. Portanto, em função da alternância no crescimento da CWND empregada pelo TCP Vegas, pode-se dizer que a quantidade de dados enviados por esse algoritmo corresponde à quantidade de dados reconhecidos pelos ACKs (o TCP Reno envia dois segmentos para cada ACK recebido, na fase *Slow Start*, naturalmente).

Os inventores do TCP Vegas argumentam que a detecção de congestionamento na fase *Slow Start* acrescentada ao TCP Vegas é importante e se tornará ainda mais importante à medida que a banda passante das redes aumentarem. No entanto, os mesmos reconhecem que o TCP Vegas implementa apenas um controle inicial e muito simples e que, por isso, precisará ser aprimorado no futuro [36].

5.5 ESTUDO COMPARATIVO

Com base na explanação apresentada no capítulo anterior sobre a arquitetura Serviços Diferenciados, não é difícil de se imaginar que, devido à importância do tema e também por ser uma tecnologia relativamente recente, muitas pesquisas relacionadas a esta arquitetura estão sendo desenvolvidas atualmente pela comunidade científica envolvida com a Internet. Dessa forma, várias propostas de mecanismos e ou arquiteturas visando aperfeiçoar o desempenho do DiffServ têm sido apresentadas. Entretanto, cabe salientar que em geral cada proposta possui características que a faz adequada a certa aplicação e ou condição e, às vezes em situações adversas a mesma pode não oferecer desempenho mínimo aceitável ou mesmo não funcionar como esperado.

Conforme mencionado anteriormente, um dos temas de grande interesse atualmente relacionado com a QoS na Internet é a capacidade dessa rede em garantir banda passante

previamente reservada aos seus usuários e de forma justa, ou seja, capacidade de oferecer justiça entre os tráfegos que estão sendo transportados. E, essa característica é a base da arquitetura DiffServ. Entretanto, o mecanismo padrão de controle de congestionamento empregado por essa arquitetura (o RIO) não oferece uma garantia realmente eficaz em todas as situações [30], sobretudo, porque o mesmo depende da interação com o protocolo de transporte TCP empregado, conforme mencionado anteriormente. Portanto, a capacidade da rede DiffServ em garantir QoS aos seus usuários é altamente dependente da interoperação entre os mecanismos RIO e TCP, os quais realizam o controle de congestionamento e de fluxo, respectivamente nesse ambiente, conforme mostrado na figura 5.4.

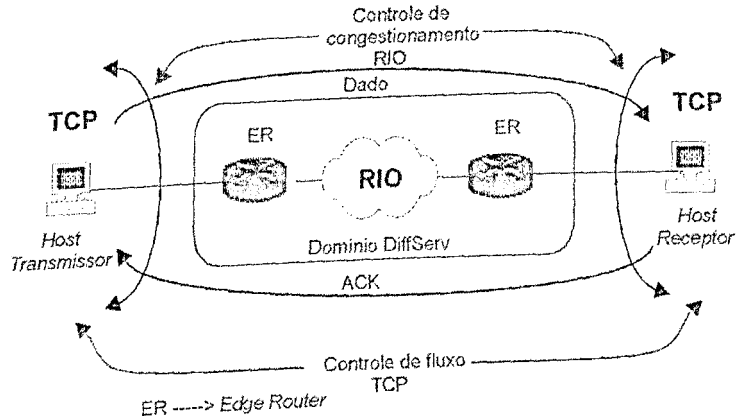

Figura 5.4 – Interação entre os mecanismos TCP e o RIO no controle de tráfego numa rede DiffServ.

Nesse sentido, muitas pesquisas abordando essa interação têm sido publicadas [30,31,44] e quase a totalidade das mesmas abordam apenas o algoritmo TCP Reno como opção de protocolo de transporte para a rede DiffServ. Por outro lado, o TCP Vegas tem sido abordado em pesquisas recentes como um protocolo promissor, no que diz respeito à estabilidade e utilização de banda passante [33,34,35,37], embora ajustes se façam necessários a fim de que o mesmo possa vir a ser implementado em larga escala, conforme mostrado em [41,50].

As maiores deficiências do TCP Vegas parecem ser a sua deficiência em garantir justiça na distribuição de banda passante [33,34] e a sua inatividade diante da agressividade do TCP Reno, o que o faz pouco viável numa rede heterogênea composta por fontes TCP Reno e TCP Vegas [34,37].

Por outro lado, de acordo com alguns estudos anteriores, tais como os desenvolvidos em [34,49], o mecanismo RED, se bem configurado, pode melhorar significativamente a performance do TCP Vegas em relação aos problemas citados.

Nesse sentido, como a rede DiffServ opera com o RIO, o qual possui dois REDs (um para os pacotes em conformidade e outro para pacotes não em conformidade) e, portanto, possui parâmetros que possibilitam controlar o comportamento do tráfego no interior da rede, a mesma se mostra, em princípio, uma boa opção para operar com o TCP Vegas. Com essa associação poderia-se vislumbrar boa utilização da banda passante da rede e justiça adequada, considerando tanto os ambientes compostos por *hosts* exclusivamente TCP Vegas como os intercalados com *hosts* TCP Reno.

Dessa forma, motivado pela relevância do assunto, e aliado ao fato de que até o presente nenhum estudo nesse sentido foi desenvolvido, pelo menos não publicado nos meios científicos de maior relevância; a proposta dessa dissertação tem por objetivo principal o estudo comparativo da performance dos algoritmos TCP Vegas e TCP Reno numa rede DiffServ.

Portanto, realiza-se nesse trabalho um estudo por meio de simulações a fim de se levantar o comportamento do TCP Vegas comparativamente ao TCP Reno, e da interação dos mesmos dentro do cenário DiffServ, sob o aspecto justiça na distribuição de banda passante. O principal ponto analisado é a interação dos mecanismos da classe de serviço AF do DiffServ com o protocolo de transporte TCP (TCP Reno e TCP Vegas), uma vez que essa interação possibilita várias combinações de comportamento possíveis. Também são

considerados os aspectos relacionados com a interoperação desses algoritmos TCP com o tráfego agressivo do UDP, visto que essa interoperação é o que realmente se constata numa implementação real.

As simulações são realizadas enfatizando-se a influência dos parâmetros do algoritmo RIO, e também dos parâmetros α e β do algoritmo TCP Vegas, na vazão das conexões. São consideradas duas situações distintas. Na primeira, são empregadas apenas duas conexões de modo que seja facilitada a análise dos resultados do TCP Vegas sobre a rede DiffServ. Na segunda, são consideradas múltiplas conexões compartilhando a banda passante da rede, em que o TCP Reno e o TCP Vegas são comparados. Os resultados obtidos são apresentados no próximo capítulo.

Assim, o estudo desenvolvido nessa pesquisa determina, sob o aspecto justiça, o perfil de funcionamento do TCP Vegas num ambiente DiffServ e, por conseguinte, estabelece condições que possibilitam a análise da viabilidade dessa associação.

5.6 CONCLUSÕES

O algoritmo TCP Vegas provê a manutenção da estabilidade na rede, uma vez que o mesmo tende a manter a CWND das conexões o mais estável possível (limitadas em função de α e β), com o propósito geral de diminuir as perdas nesse ambiente e, assim, melhorar a utilização da banda passante da rede. No entanto, essa característica do TCP Vegas tem se mostrado ineficiente, perante os resultados de trabalhos anteriores, quanto à justiça no compartilhando de banda passante por várias conexões sobre um enlace em comum.

O controle de tráfego na rede DiffServ depende tanto do RIO no interior da rede como do TCP nos *hosts*. O algoritmo RIO possui parâmetros de ajustes que possibilitam o controle de congestionamento no interior da rede. Assim, o uso associado do TCP Vegas com o RIO sobre uma rede DiffServ pode possibilitar melhorias significativas no que diz respeito às desvantagens do TCP Vegas citadas no parágrafo anterior.

Portanto, valendo-se do fato de que o TCP Vegas pode ser uma opção viável às redes DiffServ e que até o presente nenhuma pesquisa abordou esse assunto; a proposta dessa dissertação é verificar a viabilidade dessa associação diante da combinação do TCP Reno com o DiffServ, no que diz respeito à garantia de banda passante aos usuários dessa rede. Os resultados desse estudo se fundamentam em simulações.

Finalmente, pretende-se também com esse trabalho prover subsídio ao estudo de outros algoritmos TCP (tais como: new TCP Reno, Sack, Fack, etc) como opções à arquitetura DiffServ, visto que até o presente somente o TCP Reno tem sido considerado. A escolha do TCP Vegas nesse trabalho se fundamentou, sobretudo, na possibilidade de se minimizar o seu problema de injustiça, quanto à distribuição de banda passante, por intermédio de ajustes dos parâmetros do algoritmo RIO presente no DiffServ.

CAPÍTULO VI

AVALIAÇÃO DE DESEMPENHO DOS PROTOCOLOS TCP Reno E TCP Vegas NUMA REDE DiffServ

6.1 INTRODUÇÃO

Seguindo a fundamentação teórica apresentada nos capítulos anteriores, faz-se neste capítulo a abordagem quantitativa e comparativa dos protocolos TCP Reno e TCP Vegas numa rede DiffServ. O foco deste estudo é a avaliação da capacidade do serviço AF (*Assured Service*) da arquitetura DiffServ em distribuir banda passante de forma justa aos seus usuários, mediante o uso destes algoritmos de controle de fluxo (ou de congestionamento) TCP.

Para isso, lança-se mão do estudo por intermédio de simulações de forma que seja viabilizada a análise desses mecanismos sob condições diferenciadas e, por conseguinte, se estabeleça maior extensão dos resultados encontrados neste estudo.

Tomando por base os trabalhos realizados em [33,52] em que os parâmetros α e β do protocolo TCP Vegas são revelados como sendo de fundamental importância ao comportamento deste protocolo e também considerando os resultados obtidos em [34,49], nos quais os parâmetros do algoritmo RED são mostrados como tendo influência significativa no funcionamento do TCP Vegas juntamente como o TCP Reno, este trabalho se concentra na investigação da influência desses parâmetros, em termos de justiça (*fairness*), no desempenho de uma rede DiffServ.

De acordo com o que foi mencionado no capítulo anterior, o termo justiça aqui empregado, refere-se à capacidade da rede DiffServ em garantir aos seus usuários banda passante proporcional às suas respectivas taxas de transmissão configuradas. Nesse sentido, um nível de justiça adequado é sempre desejado, visto que os usuários de uma rede DiffServ são tratados de forma diferenciada em função dos seus respectivos SLAs e, por isso, esperam obter proporções diferenciadas de banda passante da rede, independentemente do fato da rede estar congestionada ou não. Portanto, justiça numa rede DiffServ é fundamental.

Neste estudo são abordados basicamente dois cenários, um em que se tem apenas duas fontes transmitindo dados, e outro no qual dez (múltiplas) fontes são consideradas. No primeiro caso, ressalta-se o funcionamento do algoritmo TCP Vegas em nível de detalhes que permitem compreender o comportamento do mesmo no ambiente simulado. No segundo caso, concentra-se apenas na influência dos parâmetros citados anteriormente sobre a justiça da rede simulada. Dessa forma os principais aspectos do TCP Vegas, no contexto deste trabalho, são cobertos.

6.2 DEFINIÇÃO DOS PARÂMETROS PARA OS EXPERIMENTOS COM OS SIMULADORES

Faz-se aqui uma abordagem descritiva dos dados que foram empregados nas simulações de uma forma geral. Nesse sentido, todos os resultados de simulações apresentados aqui se baseiam em um dos dois modelos de simulação empregados neste estudo, um com apenas duas e outro com dez (múltiplas) fontes transmitindo dados sobre uma rede DiffServ. O propósito de se utilizar esses dois modelos é o de não somente obter-se resultados mais abrangentes, mas também o de se ter os mesmos cenários que foram empregados em algumas das pesquisas citadas na seção anterior, de forma que seja viabilizada a comparação dos resultados aqui obtidos com os daquelas referências.

Em relação ao tráfego gerado pelas fontes tem-se que todas as fontes submetidas ao algoritmo TCP (denominadas fontes TCP) geram tráfego do tipo "FTP infinito", i.e. transmitem um arquivo de tamanho "infinito" a uma taxa suficientemente alta de modo que a taxa de transmissão ou banda passante configurada de cada fonte seja determinada somente pelos ajustes realizados nos roteadores de borda (mediante o ajuste para o tráfego em conformidade). Analogamente, as fontes que utilizam o algoritmo UDP, denominadas fontes UDP, geram tráfego CBR.

Os parâmetros usados para o algoritmo RIO, estão indicados na tabela 6.1. Conforme mencionado anteriormente, nos capítulos 2 e 4, o RIO possui dois REDs, um para os pacotes em conformidade e outro para os pacotes não em conformidade[44]. A tabela 6.1 indica que os parâmetros relacionados aos pacotes em conformidade são mantidos fixos para todas as simulações, enquanto que os parâmetros de limiar min_out e max_out dos pacotes não em conformidade, referenciados genericamente aqui como Min_{th} e Max_{th} para ambos os tipos de pacotes, são variados entre os limites indicados na referida tabela.

	Em conformidade	Não em conformidade
Min_{th} (pacotes)	20	3/5/7
Max_{th} (pacotes)	40	6/10/14
Max_p	0,02	0,1

Tabela 6.1 – Valores dos parâmetros dos REDs do RIO.

Tal procedimento foi seguido, sobretudo, com o intuito de viabilizar o desenvolvimento deste estudo, uma vez que a variação de todos esses parâmetros acarretaria um número muito grande de situações o que inviabilizaria este trabalho. Da mesma forma, restringiu-se os valores utilizados para a análise das variações dos parâmetros α e β.

Nesse sentido, os conjuntos de valores utilizados nas simulações para os parâmetros de limiar Min_{th} e Max_{th} associados aos pacotes não em conformidade, e representados da forma RIO(out): Min_{th} - Max_{th}, são: RIO(out): 3-6, RIO(out): 5-10 e RIO(out): 7-14. Estes valores foram escolhidos por representarem bem a faixa em que a maioria das pesquisas relacionadas

tem trabalhado, como por exemplo [34]. Observa-se que Min_{th} é o dobro de Max_{th} nos três casos e isso se deve ao fato de que essa é uma recomendação feita em [28]. Por questão de simplificação, os valores atribuídos a esses parâmetros serão daqui por diante mencionados simplesmente como RIO(out): a-b, em que a e b correspondem aos respectivos parâmetros citados.

Nesse mesmo princípio, quatro conjuntos de valores (em bytes) para os parâmetros α e β do TCP Vegas são tratados, os quais são: α=1 e β=3, α=β=1, α=β=2 e α=β=3. Também por simplificação, esses parâmetros serão referenciados simplesmente como parâmetros α e β.

Todos os simuladores foram implementados por meio do programa de simulação "*NS network simulator*" [27]. Dessa forma, para todos os demais parâmetros não mencionados explicitamente neste trabalho devem ser considerados os valores padrões definidos nesse programa de simulação.

6.3 JUSTIÇA PARA DUAS FONTES TCP NUMA REDE DiffServ

O propósito principal desta seção é estabelecer os procedimentos indispensáveis à correta operação do TCP Vegas conjuntamente com a rede DiffServ, visto que essa associação demanda ajustes específicos nos parâmetros do algoritmo RIO bem como nos parâmetros α e β. Também são comparados os algoritmos TCP Vegas e TCP Reno e avaliada a interoperação dos mesmos.

6.3.1 MODELO DE SIMULAÇÃO

O modelo de rede empregado em todas as simulações tratadas na seção 6.3 (duas fontes) é o apresentado na figura 6.1. De acordo com a figura, há duas fontes (S1 e S2) enviando dados a dois destinos (D1 e D2, respectivamente), dois roteadores de borda interligados por um enlace DiffServ que possui uma fila do tipo RIO, e quatro enlaces com filas do tipo FIFO interligando as fontes e os destinos aos roteadores. Conforme mostra a

figura 6.1, o enlace DiffServ tem capacidade de 3 Mbps e retardo de 1 ms e cada enlace FIFO possui capacidade de 10 Mbps e retardo de 1 ms.

Esse cenário estabelece duas conexões, uma entre S1 e D1 e outra entre S2 e D2, as quais serão referenciadas daqui por diante simplesmente como conexão da fonte S1 e conexão da fonte S2, respectivamente (ou ainda conexão associada, ou relacionada, à S1 e à S2, respectivamente). A velocidade de transmissão (configurada) de cada fonte, para o tráfego em conformidade, pode variar de uma simulação para outra e, por isso, as mesmas são representadas por X e Y Mbps na figura 6.1 para as fontes S1 e S2, respectivamente.

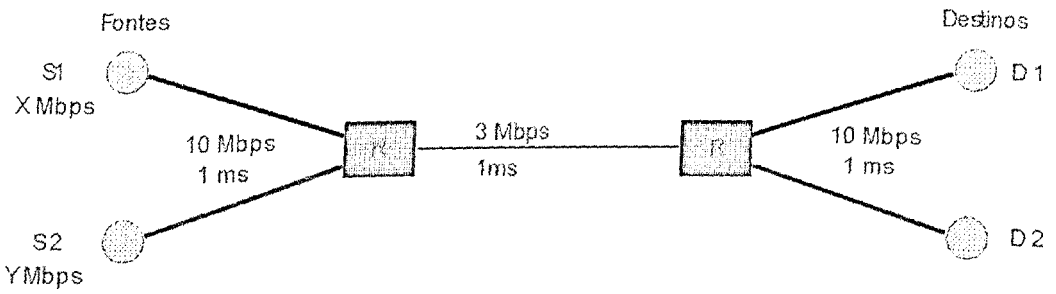

Figura 6.1 – Modelo de rede simulada.

6.3.2 Influência dos parâmetros α e β

O estudo desenvolvido nesta sub-seção tem por objetivo mostrar a influência que os parâmetros α e β têm sobre a distribuição de banda passante numa rede DiffServ. Nos simuladores aqui apresentados, o conjunto de parâmetros RIO(out) são mantidos fixos para que apenas as variações causadas pelos dois referidos parâmetros possam ser observadas.

O modelo simulado corresponde exatamente àquele da figura. 6.1, considerando-se duas fontes TCP Vegas com taxa de transmissão configurada de 1 Mbps cada uma. As fontes S1 e S2 começam a transmitir nos instantes $t_1 = 0$ s e $t_2 = 5$ s, respectivamente; quanto ao tempo de simulação, foram utilizados dois intervalos: de 0 a 30 s e de 0 a 100 s, os quais podem ser identificados pelas figuras correspondentes a cada simulação. A finalidade destes dois intervalos é facilitar a visualização da extensão dos resultados obtidos em cada caso.

Dessa forma, foram realizadas quatro simulações, nas quais manteve-se o RIO(out): 3-6 e variou-se os parâmetros α e β de acordo com os valores citados na seção 6.2. Os resultados obtidos são mostrados pelas figuras 6.2, 6.3, 6.4 e 6.5. Estas figuras apresentam: (a) banda passante alcançada pelas conexões associadas às fontes, (b) janela de congestionamento (CWND) de cada conexão e (c) tempo de ciclo medido para cada conexão. O item (d) de cada uma dessas figuras corresponde a uma ampliação do item (c) no instante de interesse de análise em que a fonte S2 começa a transmitir (na vizinhança de t = 5 s). A seguir, faz-se a avaliação dos resultados obtidos para essas quatro simulações citadas.

6.3.2.1 Avaliação para α=1 e β=3

Analisando-se o resultado obtido para α=1 e β=3 mostrado na figura 6.2a, pode-se observar que as conexões das fontes S1 e S2 não obtiveram valores de banda passante iguais conforme esperado, uma vez que a rede não está congestionada e ambas as conexões são idênticas. Esse resultado está de acordo com o estudo realizado em [33] no qual é mostrado que esse problema de injustiça, quanto à distribuição de banda passante, é causado pelo fato de que com $\alpha<\beta$ as CWNDs das conexões se mantêm fixas, como indicado na figura 6.2b (a partir de t = 7 s) e, em conseqüência, a banda passante obtida por cada conexão se estabiliza num valor diferente uma da outra. Isso acontece porque os RTTs das conexões somente sofrem variações nos instantes em que as fontes começam a enviar dados.

Portanto, conforme pode ser observado pelas figuras 6.2c e 6.2d, no instante em que a segunda fonte entra em operação (t = 5 s), os RTTs de ambas as conexões começam a variar e se mantêm assim até aproximadamente t = 6,7 s, o que faz com que a CWND da fonte S1 decresça até o tamanho de 3 segmentos (em função do aumento em RTT1) e a CWND de S2 atinja 4 segmentos. A partir desse instante o RTT de ambas as conexões se estabilizam, em torno de 18 ms, e então o algoritmo Vegas entende que não há congestionamento na rede e, por conseguinte, não altera mais o valor das CWNDs (capítulo V, seção 5.4).

O fato da segunda conexão obter maior banda passante em relação à primeira também pode ser explicado com base no princípio de funcionamento do TCP Vegas. De acordo com o

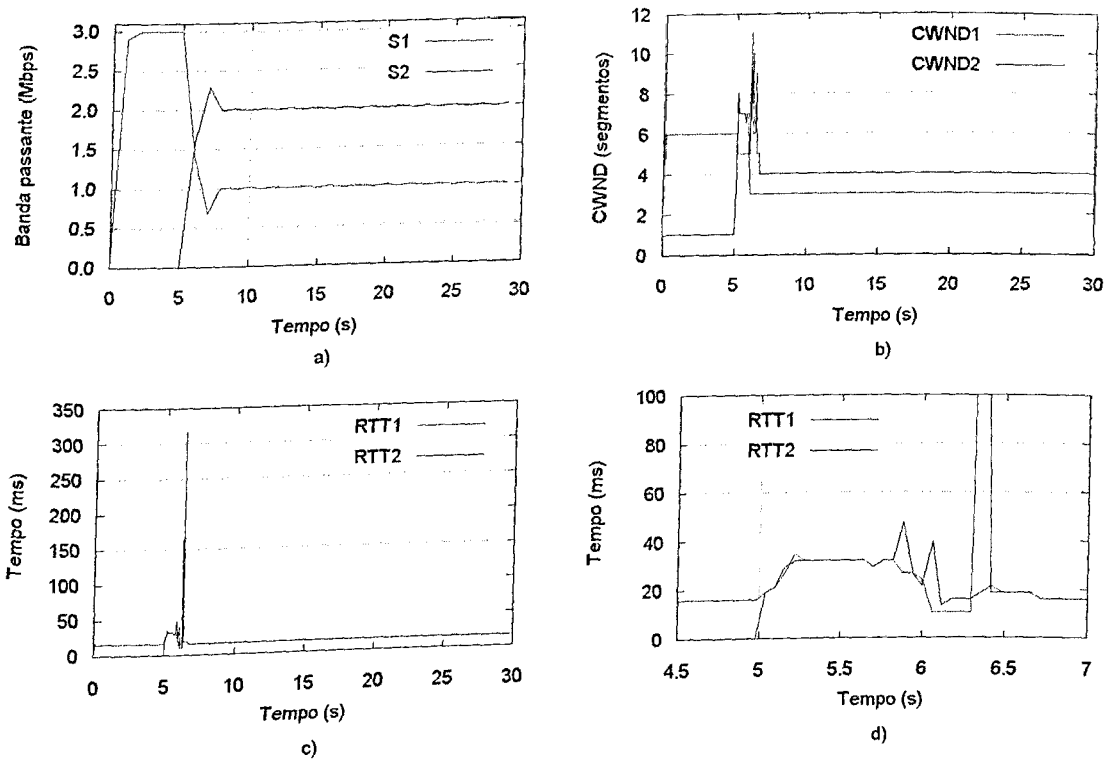

Figura 6.2 – TCP Vegas com α =1 e β=3:
a) Banda passante obtida pelas conexões das fontes S1 e S2
b) Janela de congestionamento das conexões associadas a S1 e S2
c) RTTs das conexões
d) RTTs no instante em que as duas conexões começam a funcionar conjuntamente

que foi apresentado no capítulo anterior, no instante em que uma conexão TCP Vegas começa a transmitir, a mesma mede o RTT do primeiro segmento enviado a fim de determinar se a sua CWND pode ser aumentada.

Dessa forma, como no instante em que S2 entra em operação a rede só possui a fonte S1 transmitindo, o valor de RTT medido para a conexão da fonte S2 (RTT2) deve estar próximo ao medido para a conexão da fonte S1 (RTT1), figuras 6.2c e 6.2d. Assim, a fonte S2 começa a transmitir e à medida que a mesma insere dados na rede, o RTT1 começa a aumentar e, em resposta, o algoritmo TCP Vegas de S1 diminui a CWND1 causando uma redução na taxa de transmissão de S1. Nesse ritmo, a CWND1 diminui até que o RTT de cada

conexão atinja o equilíbrio a partir de t = 6,7 s. Deste ponto em diante o algoritmo TCP Vegas infere que as CWNDs das conexões não devem ser alteradas e, assim, uma situação de estabilidade se instaura culminando no problema de injustiça citado anteriormente.

Portanto, nessas condições pode-se dizer que o algoritmo TCP Vegas original, com os parâmetros α=1 e β=3, não provê justiça às conexões desta rede DiffServ. Em [33,52] os autores ressaltam que o TCP Vegas pode prover justiça adequada numa rede convencional, baseada no serviço de melhor esforço, fazendo-se α=β. Nesse sentido, foram simulados aqui três conjuntos de parâmetros α=β a fim de se verificar a eficácia desta abordagem num ambiente DiffServ. As figuras 6.3, 6.4 e 6.5 apresentam os resultados para α=β=1, α=β=2 e α=β=3, respectivamente; a avaliação de cada um destes casos são apresentadas nas sub-seções seguintes.

6.3.2.2 Justiça para α=β

Fazer α=β, implica em introduzir oscilações constantes na CWND das conexões a fim de se obter a mesma banda passante para todas as conexões que compartilham a banda passante da rede, considerando-se fontes idênticas, como é o caso. No entanto, para que isso efetivamente ocorra, o RTT de cada conexão deve variar continuamente, visto que o TCP Vegas depende diretamente desse parâmetro para ajustar a CWND de cada conexão. Essa explanação pode ser melhor entendida analisando-se a equações 6.1 e 6.2 que mostram como a CWND de uma conexão é atualizada pelo TCP Vegas. Estas equações são as mesmas apresentadas no capítulo anterior (equações 5.6 e 5.7) e são novamente apresentadas aqui simplesmente com o objetivo de facilitar a alusão às mesmas.

$$CWND(t+t\Delta) = \begin{cases} CWND(t)+1, \ se \ \ Diff < \dfrac{\alpha}{base\_rtt}; \\ CWND(t), \quad se \quad \dfrac{\alpha}{base\_rtt} \le Diff \le \dfrac{\beta}{base\_rtt}; \\ CWND(t)-1, \ se \ \ \dfrac{\beta}{base\_rtt} < Diff; \end{cases} \tag{6.1}$$

Sendo,

$$Diff = VE - VR = (\frac{CWND(t)}{base\_rtt}) - (\frac{CWND(t)}{rtt}) \tag{6.2}$$

Observando-se a equação 6.1, nota-se que se $\alpha=\beta$, então a mesma se transforma na equação 6.3. E, por esta equação pode-se observar que desde que Diff $\neq$ ($\alpha=\beta$)/base_rtt, a CWND varia continuamente.

$$CWND(t+t\Delta) = \begin{cases} CWND(t)+1, \ se \ \ Diff < \dfrac{\alpha=\beta}{base\_rtt}; \\ CWND(t), \quad se \quad Diff = \dfrac{\alpha=\beta}{base\_rtt}; \\ CWND(t)-1, \ se \ \ Diff > \dfrac{\alpha=\beta}{base\_rtt}; \end{cases} \tag{6.3}$$

A seguir são apresentados os resultados das simulações em que foram utilizados os conjuntos de valores $\alpha=\beta$ citados anteriormente.

6.3.2.3 Avaliação para $\alpha=\beta=1$

Com base no exposto na sub-seção anterior pode-se explicar o resultado obtido para $\alpha=\beta=1$, o qual é apresentado na figura 6.3. A figura 6.3a mostra que as conexões referentes às fontes S1 e S2 não receberam a mesma banda passante, como era em princípio esperado. A

figura 6.3b mostra que a CWND1 oscila continuamente entre 4 e 7 segmentos até o instante em que S2 entra em operação. A partir de então ambas as CWNDs se mantêm estáveis, e a CWND2 se estabiliza no valor mínimo 2 (janela de reinicio).

Isto ocorre porque a partir do instante em que S2 começa a transmitir (t = 5 s), o RTT1 aumenta de um patamar médio ao redor de 14 ms até o pico de 27 ms, quando então se estabiliza em 17 ms (figuras 6.3c e 6.3d). Com isto, de acordo com a equação 6.3, a CWND1 é decrementada até o valor mínimo de 2 segmentos e não volta a aumentar, uma vez que o seu RTT não volta a cair. Por outro lado, a CWND2 se estabiliza em 4 segmentos, o que indica que a igualdade da equação 6.3 foi alcançada, e como os RTTs de ambas as conexões se mantêm iguais e estáveis a partir deste transitório, uma situação de estabilidade é instaurada para as conexões e, analogamente ao caso anterior, ainda persiste o problema de injustiça na distribuição de banda passante.

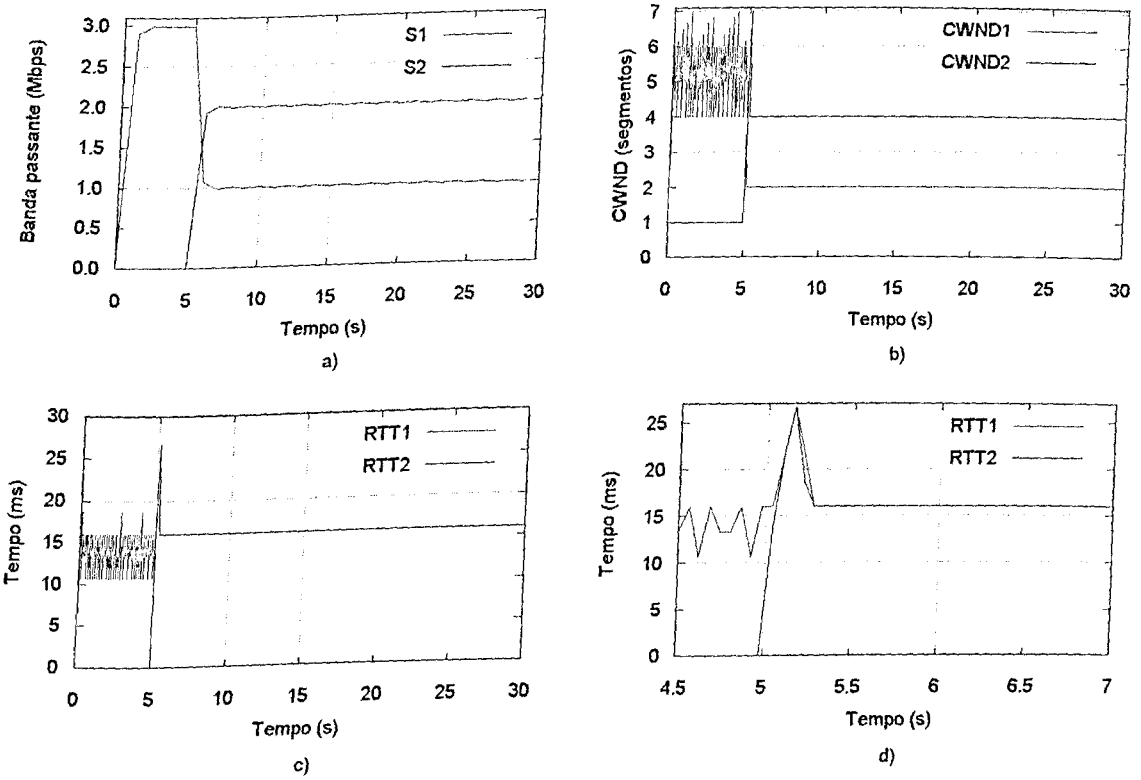

Figura 6.3 – TCP Vegas com α =1 e β=1:
a) Banda passante obtida pelas conexões das fontes S1 e S2
b) Janela de congestionamento das conexões associadas a S1 e S2
c) RTTs das conexões
d) RTTs no instante em que as duas conexões começam a funcionar conjuntamente

6.3.2.4 Avaliação para $\alpha=\beta=2$

Os resultados da simulação realizada com $\alpha=\beta=2$ são apresentados na figura 6.4. Pode-se notar que nesta simulação ambas as conexões competiram por banda passante de forma justa, visto que a banda passante média alcançada pelas mesmas possui o mesmo valor de 1,5 Mbps (figura 6.4a). Portanto, com este valor de parâmetro para α e β, pode-se dizer que o modelo simulado fornece uma justiça adequada.

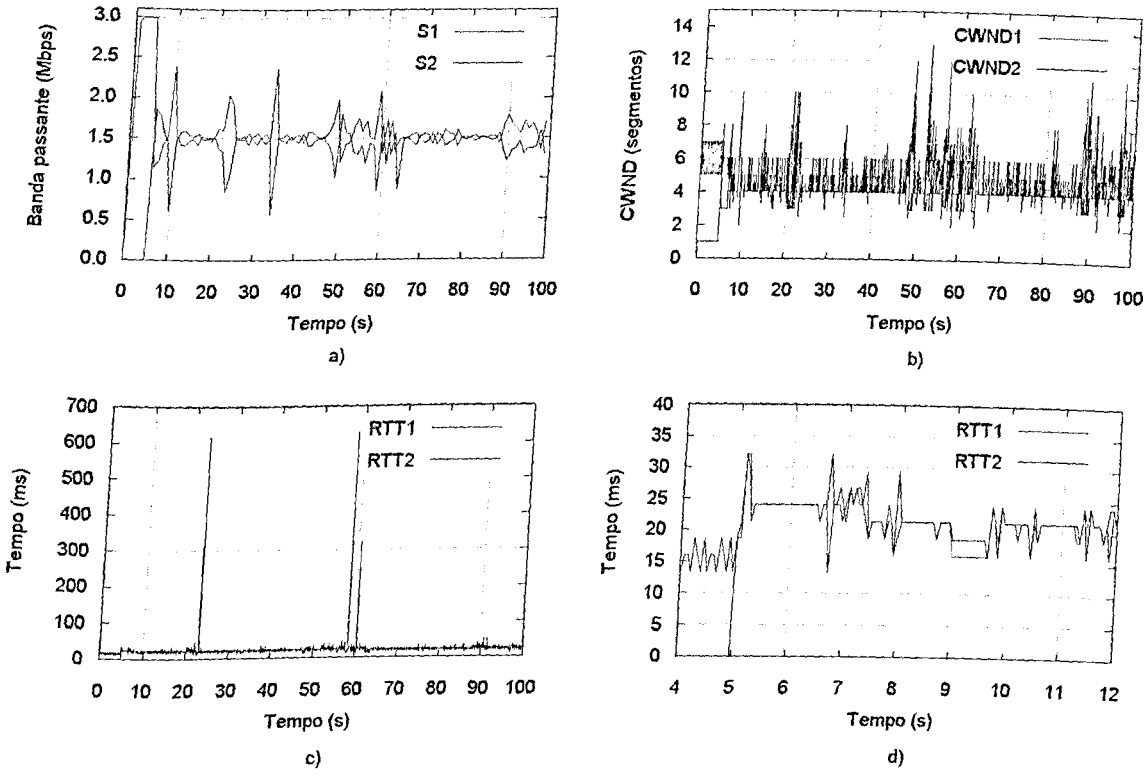

Figura 6.4 – TCP Vegas com α =2 e β=2:
a) Banda passante obtida pelas conexões das fontes S1 e S2
b) Janela de congestionamento das conexões associadas a S1 e S2
c) RTTs das conexões
d) RTTs no instante em que as duas conexões começam a funcionar conjuntamente

Comparando-se a figura 6.4 com a 6.3, nota-se que a diferença fundamental entre as mesmas é o fato de que na figura 6.4 a CWND das conexões varia continuamente em função das variações correspondentes de seus respectivos RTTs.

Tal comportamento decorre do fato de que com um valor de $\alpha=\beta$ maior, em relação ao caso anterior ($\alpha=\beta=1$), a relação ($\alpha=\beta$)/base_rtt presente na equação 6.3 também torna-se maior e, por conseguinte, o valor de Diff, equação 6.2, ao ser incrementado, leva mais tempo para alcançar o valor desta relação. Nessa tendência, o valor da CWND atinge valores maiores, em relação ao caso anterior, antes de começar a decrescer e, isso, afeta o RTT das conexões (figuras 6.4c e 6.4d), visto que as mesmas tornam-se mais agressivas quanto ao envio de tráfego à rede.

Com base nisso é que se justifica o comportamento oscilatório das CWNDs e dos RTTs observados nas figuras 6.4b e 6.4c, respectivamente. Pode-se observar que o nível máximo atingido pela CWND2 na figura 6.4b é ligeiramente maior do que o da figura 6.3b (13 e 7 segmentos, respectivamente). De forma análoga, entre os instantes t = 4 s e t = 12 s (transitório), o RTT2 para a figura 6.4d atinge um máximo de 32 ms enquanto que este mesmo parâmetro na figura 6.3d não passa dos 27 ms. Portanto, isso comprova que um pressuposto fundamental para se conseguir justiça com o TCP Vegas, considerando-se o modelo simulado, é a variação contínua no tamanho da janela de congestionamento das conexões [33].

6.3.2.5 Avaliação para $\alpha=\beta=3$

Utilizando-se $\alpha=\beta=3$, também se conseguiu justiça para as duas fontes em questão, conforme se pode constatar pelos resultados apresentados na figura 6.5. A banda passante obtida por cada uma das conexões na figura 6.5a corresponde a 1,5 Mbps. No entanto, como esperado pela análise feita anteriormente, o nível de oscilação das CWNDs das conexões torna-se maior (figura 6.5b), em função das variações dos RTTs correspondentes (figuras 6.5c e 6.5d), comparativamente ao caso anterior em que se utilizou $\alpha=\beta=2$; isso provoca maior instabilidade na rede, conforme pode ser observado pela figura 6.5a. Valores ainda maiores do

que 3 tendem a causar mal funcionamento da rede, uma vez que as perdas tendem a aumentar por excesso de agressividade das conexões, ou pode ocorrer de somente umas das conexões obter banda passante em detrimento da outra que é anulada.

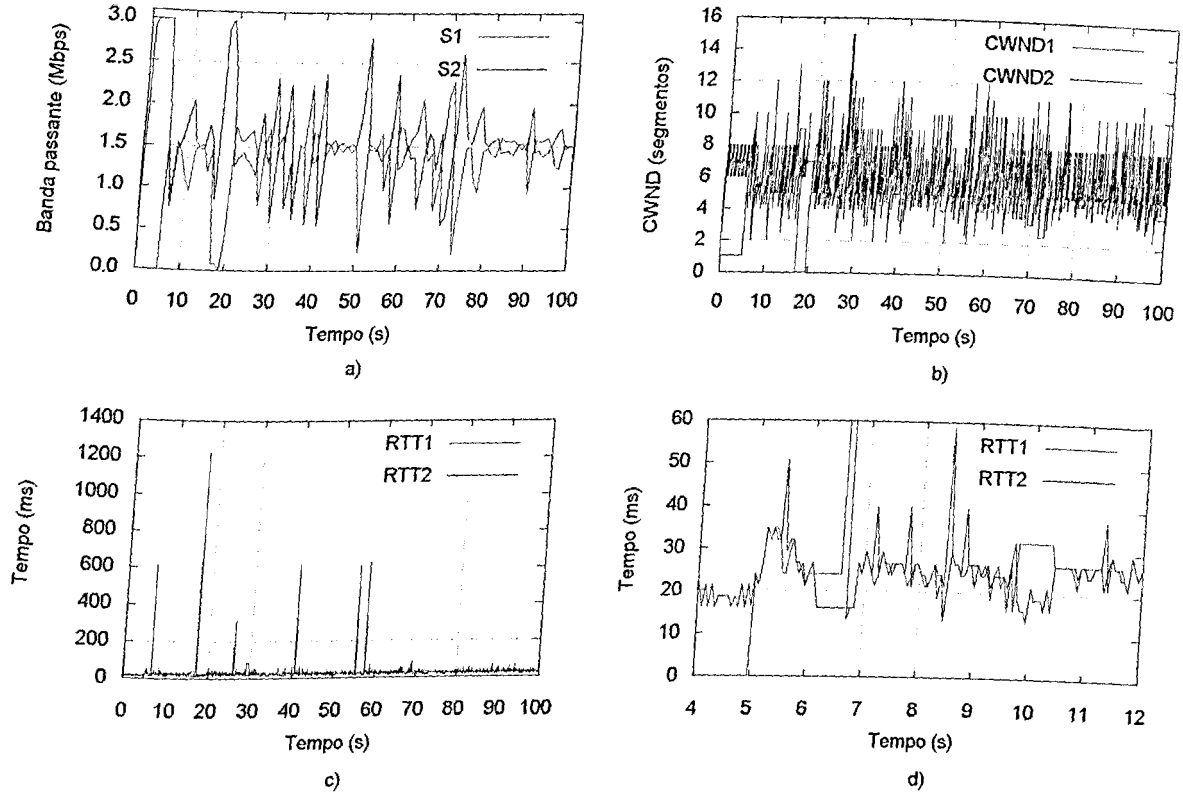

Figura 6.5 – TCP Vegas com α =3 e β=3:
a) Banda passante obtida pelas conexões das fontes S1 e S2
b) Janela de congestionamento das conexões associadas a S1 e S2
c) RTTs das conexões
d) RTTs no instante em que as duas conexões começam a funcionar conjuntamente

Portanto, nas condições simuladas e empregando-se o modelo da figura 6.1, o melhor resultado foi obtido para $\alpha=\beta=2$. Com $\alpha=1$ e $\beta=3$ e $\alpha=\beta=1$, a estabilidade do TCP Vegas não possibilitou uma justiça aceitável e, com $\alpha=\beta=3$, embora se tenha conseguido justiça adequada a rede se mostrou muito instável.

6.3.3 Influência dos parâmetros do algoritmo RIO

Na sub-seção anterior os parâmetros RIO(out) foram fixados em 3-6 de modo que se pudesse analisar apenas o efeito dos parâmetros α e β sobre a distribuição de banda passante

na rede DiffServ. Nesta sub-seção o objetivo é determinar como a banda passante das conexões são afetadas pela variação dos parâmetros RIO(out) nesse ambiente.

O modelo de rede simulado é o mesmo da figura 6.1, sendo as fontes S1 e S2 TCP Vegas idênticas com taxa de transmissão configurada de 1 Mbps cada uma. Os parâmetros α e β foram ajustados em 2 (α=β=2) e dois conjuntos de parâmetros RIO(out) foram simulados: 5-10 e 7-14.

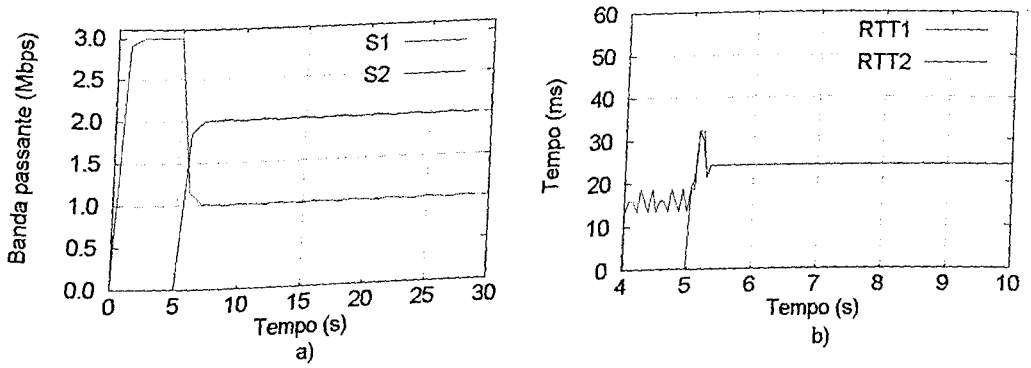

Figura 6.6 – TCP Vegas com RIO(out):5-10 e α=β=2: a) Banda passante b) RTT

Os resultados das simulações para RIO(out): 5-10 e RIO(out): 7-14 são apresentados nas figuras 6.6 e 6.7, respectivamente. Conforme pode-se observar, os resultados para ambas as simulações mostraram-se muito próximos e não possibilitaram justiça adequada às conexões.

Verificando-se o RTT de ambas as simulações, figuras 6.6 e 6.7, nota-se que os mesmos não sofrem oscilação alguma após o transitório imposto pela inclusão do tráfego da fonte S2 (entre os instantes t = 5 s e t = 5,4 s). Isso se deve, sobretudo, ao fato de que valores maiores para os parâmetros RIO(out) implicam em maior quantidade de tráfego na rede, uma vez que menos segmentos são descartados nesta situação. Com isso, o tempo de enfileiramento maior sofrido pelos pacotes faz com que os RTTs dos mesmos se tornem maiores, em relação aos valores obtidos para o RIO(out): 3-6 apresentados na sub-seção

anterior. Dessa forma, o resultado aqui obtido é muito parecido àquele mostrado na figura 6.3, pelas mesmas razões mencionadas naquele tópico.

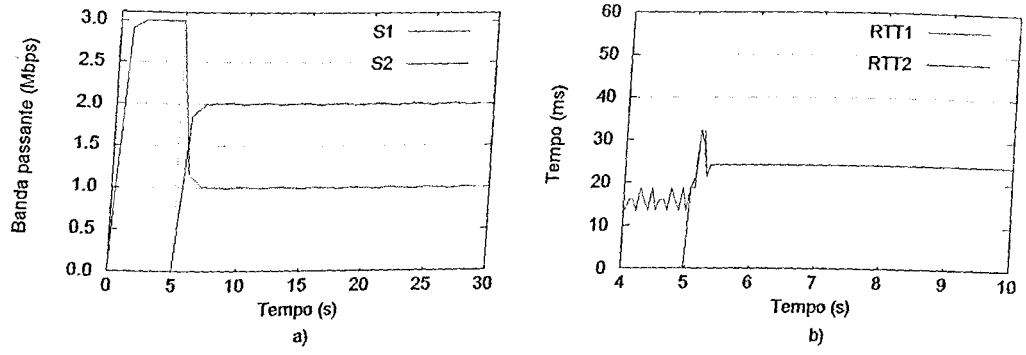

Figura 6.7 – TCP Vegas com RIO(out): 7-14 e α=β=2: a) Banda passante b) RTT

Desse modo, conclui-se que a rede DiffServ somente é capaz de oferecer justiça adequada aos seus usuários, considerando-se o cenário estudado, se forem utilizados valores extremamente pequenos para os parâmetros RIO(out) e valores iguais, entre 2 e 3, para os parâmetros α e β.

6.3.4 Fontes com taxas de transmissão configuradas diferentes

Todos os casos estudados anteriormente consideraram uma rede não congestionada, ou super dimensionada, em que as duas fontes envolvidas eram idênticas. Nesta sub-seção, aborda-se a avaliação da rede DiffServ em termos de distribuição de banda passante, considerando-se tais fontes com taxas de transmissão diferentes uma da outra, nas situações de congestionamento e sem congestionamento na rede.

O modelo de rede empregado é o mesmo da figura 6.1, sendo S1 e S2 fontes TCP Vegas. Porém, nesta simulação a taxa de transmissão configurada da fonte S2 foi variada de 1 a 3 Mbps (Y Mbps), conforme apresentado na figura 6.8, enquanto que a da fonte S1 foi mantida em 1 Mbps. A finalidade desse procedimento é simular as situações nas quais a rede se encontra congestionada e não congestionada.

Foram empregados os valores de parâmetros $\alpha=\beta=2$ e RIO(out): 3-6. O tempo de simulação analisado corresponde a 100 s e as vazões medidas correspondem aos 80 s finais de simulação (rede estabilizada). Analisando-se o gráfico da figura 6.8, nota-se que até o ponto em que a rede se encontra sem congestionamento (Y = 2 Mbps), ambas as conexões alcançam suas respectivas banda passante configuradas. A partir desse ponto, a conexão de S1, que começa a funcionar primeiramente, mantém a sua banda passante configurada (1 Mbps), e a conexão associada a S2 não consegue passar de 2 Mbps, embora a taxa de transmissão configurada de S2 continue aumentado até 3 Mbps.

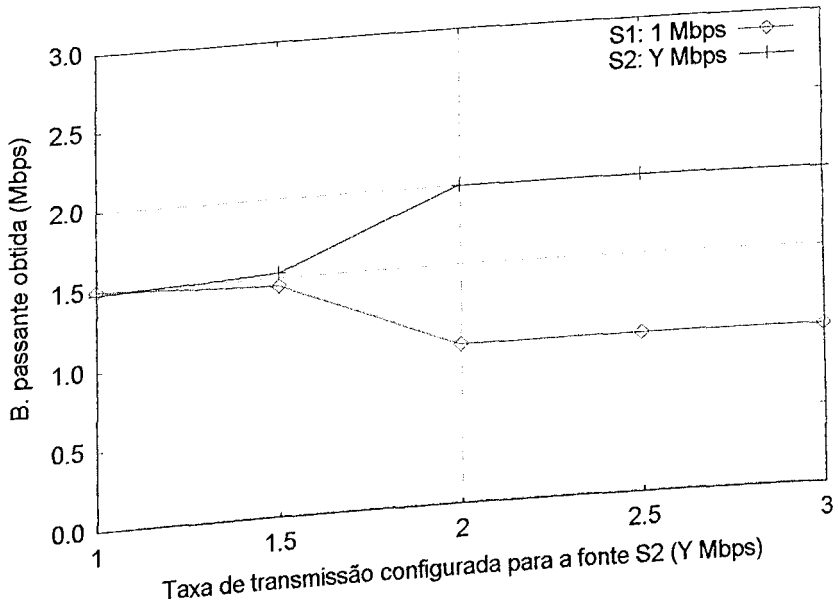

Figura 6.8 – Capacidade do TCP Vegas em distribuir banda passante de forma justa.

Esse comportamento na distribuição de banda passante é atribuído ao fato de que na iminência de congestionamento na rede, o algoritmo TCP Vegas faz com que ambas as fontes diminuam as suas respectivas taxas de transmissão (diminuindo as suas CWNDs relacionadas) de forma a evitar o congestionamento efetivo dentro da rede. Por isso, quando a conexão associada à fonte S2 ultrapassa o limite de 2 Mbps, a mesma causa congestionamento na rede, o qual é detectado pelo aumento significativo em seu RTT medido, quando então a mesma

diminui a sua CWND, levando o RTT de ambas as conexões a uma situação de estabilidade e, por conseguinte, a banda passante obtida por cada fonte.

A desvantagem desta característica de funcionamento do TCP Vegas perante a rede DiffServ é que a mesma não possibilita, em situações de congestionamento, um compartilhamento proporcional de banda passante entre os usuários dessa rede, considerando-se a taxa de transmissão configurada de cada usuário.

6.3.5 TCP Reno e TCP Vegas

O objetivo desta sub-seção é avaliar o desempenho da rede DiffServ, em termos de justiça, quando operando com uma fonte TCP Reno e outra TCP Vegas (interoperação). Analogamente à abordagem anterior em que as fontes TCP Vegas foram estudadas sob as condições de rede congestionada e não congestionada, a interação entre os dois mecanismos TCP aqui mencionados é também realizada para essas duas condições da rede.

6.3.5.1 Fontes com taxas de transmissão iguais numa rede não congestionada

Nessas simulações o modelo utilizado é o mesmo da figura 6.1. No entanto, neste caso as fontes S1e S2 empregam o TCP Reno e o TCP Vegas como mecanismo de controle de congestionamento, respectivamente, e as fontes iniciam as suas transmissões simultaneamente no instante $t = 0$ s. Os parâmetros $\alpha=\beta=2$ foram utilizados para a fonte TCP Vegas (S2) e, com respeito aos parâmetros RIO(out), foram simulados os mesmos três conjuntos de valores empregados anteriormente (3-6, 5-10 e 7-14), conforme pode ser visto na figura 6.9. O tempo de simulação é de 100 s e como ambas as fontes possuem uma taxa de transmissão configurada de 1 Mbps, totalizando 2 Mbps sobre um enlace de 3 Mbps (66% de utilização), a rede é dita não congestionada.

Os resultados obtidos indicam que somente com o RIO(out): 3-6, figura 6.9a, foi possível obter-se igualdade na distribuição de banda passante (mesma banda passante média)

para as conexões relacionadas às fontes TCP Reno e TCP Vegas. À medida que os parâmetros RIO(out) foram aumentados, figuras 6.9b e 6.9c, acentuou-se a diferença na banda passante alcançada por essas conexões.

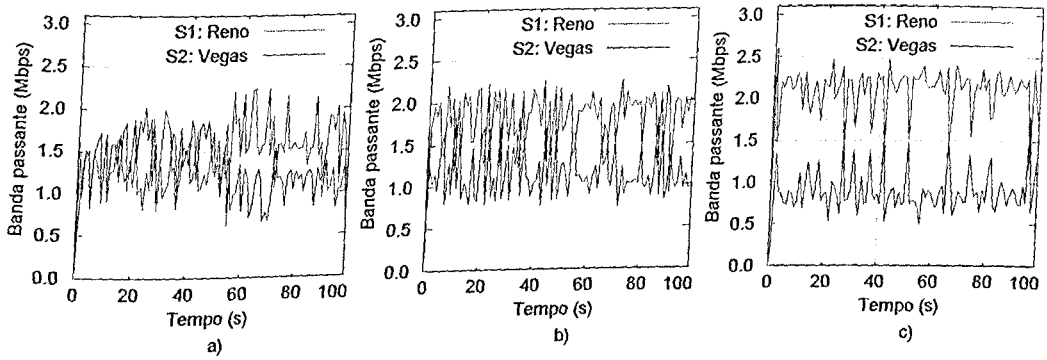

Figura 6.9 – Efeito dos parâmetros RIO(out) sobre os tráfegos TCP Reno e TCP Vegas com rede não congestionada:
a) RIO(out): 3-6 b) RIO(out): 5-10 c) RIO(out): 7-14

Isso ocorre porque, diminuindo-se os valores RIO(out), limita-se a quantidade de pacotes não em conformidade dentro da rede. Nesse sentido, como esses pacotes são gerados em maioria pela fonte TCP Reno, em função da sua característica de aumentar "indefinidamente" a sua CWND até que ocorram perdas na rede, diminui-se a agressividade dessa fonte diante da fonte TCP Vegas e, por conseguinte, melhora-se a justiça obtida nesse ambiente.

6.3.5.2 Fontes com taxas de transmissão iguais numa rede congestionada

As simulações apresentadas nesta sub-seção são análogas as da sub-seção anterior, a única diferença é o fato de que neste caso a rede se encontra congestionada. Para isso, as taxas de transmissão de cada fonte são configuradas em 2 Mbps, resultando num total de 4 Mbps sendo enviado sobre um enlace de 3 Mbps, o que corresponde à uma "carga" de 133% sobre este enlace.

Os resultados são apresentados na figura 6.10, na qual se observa que o comportamento da rede se aproximou ao da situação anterior (rede não congestionada), em que a igualdade na distribuição de banda passante somente foi obtida com o menor valor de RIO(out), para o qual cada conexão obteve uma banda passante média de 1,5 Mbps. No entanto, pode-se notar que neste caso, para o valor RIO(out): 3-6 (figura 6.10a), as variações sofridas pela banda passante de cada conexão ficaram mais próximas uma da outra.

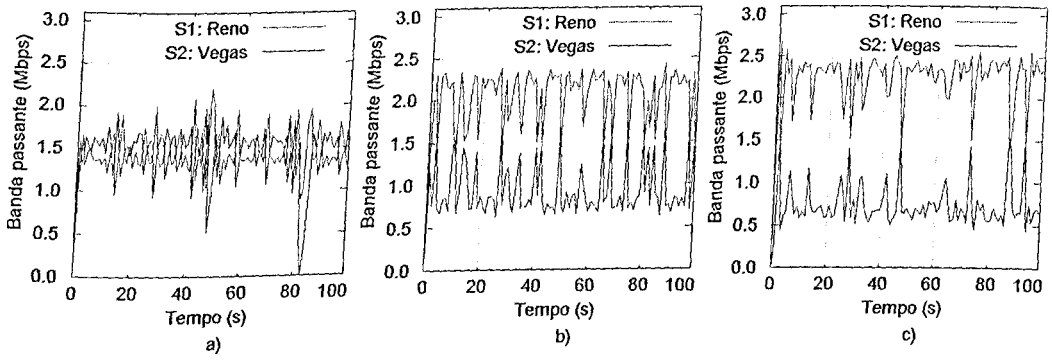

Figura 6.10 – Efeito dos parâmetros RIO(out) sobre os tráfegos TCP Reno e TCP Vegas com rede congestionada:
a) RIO(out): 3-6 b) RIO(out): 5-10 c) RIO(out): 7-14

Isso se deve ao fato de que sob congestionamento a rede possui maior quantidade de pacotes em trânsito, sobretudo pacotes oriundos da fonte S1 (Reno) em função da sua agressividade mencionada anteriormente, o que consequentemente causa mais perdas dentro da rede e impede que a fonte S1 sobreponha a fonte S2 em intervalos intercalados como acontece no caso anterior (figura 6.9a). Em relação aos resultados obtidos com os parâmetros RIO (out) 5-10 e 7-14, conforme se pode notar pelas figuras 6.10b e 6.10c, respectivamente, houve uma ligeira acentuação na diferença entre as banda passantes alcançadas por ambas as conexões.

Portanto, conclui-se que com o RIO(out): 3-6 obtém-se justiça adequada entre as fontes TCP Reno e TCP Vegas, tanto para a rede congestionada como não congestionada. Por

outro lado, valores maiores para esse parâmetro não garantem justiça alguma, mas sim a superação da fonte TCP Reno sobre a fonte TCP Vegas.

6.3.5.3 Fontes com taxas de transmissão diferentes

Os resultados das simulações apresentadas nesta sub-seção mostram o quanto a agressividade do TCP Reno pode influenciar a capacidade do TCP Vegas em conseguir obter a banda passante solicitada pelo usuário. Para isso, utilizou-se o mesmo modelo de simulação da figura 6.1 e procedeu-se a duas simulações. Na primeira a fonte S1 (Reno) foi mantida com a taxa de transmissão configurada fixa em 1 Mbps enquanto que a da fonte S2 sofreu variações de 1 a 3 Mbps (Y Mbps). Na segunda simulação a situação foi invertida, conforme pode ser observado pelas figuras 6.11a e 6.11b. Ambas as fontes começam a transmitir no instante t = 0 s.

De forma análoga ao procedimento adotado na sub-seção 5.3.4, utilizou-se nestas simulações os valores de parâmetros α=β=2 e RIO(out): 3-6. O tempo de simulação utilizado para cada valor de banda passante analisado corresponde a 100 s e as vazões medidas correspondem aos 80 s finais de simulação (rede estabilizada).

Analisando-se os resultados apresentados nas figuras 6.11a e 6.11b, em que são consideradas as situações de rede congestionada e não congestionada, nota-se que a agressividade do TCP Reno o torna superior diante do TCP Vegas. Na figura 6.11a é mostrado que apesar da fonte S2 (Vegas) possuir uma taxa de transmissão configurada crescente frente à fonte S1 (Reno) que é fixada em 1 Mbps, a conexão da mesma não consegue obter aumento na sua banda passante alcançada a partir de 2 Mbps. Por outro lado, na situação mostrada na figura 6.11b, a conexão de S1 (Reno) consegue aumentar continuamente a sua banda passante em detrimento da conexão de S2 (Vegas).

Portanto, de uma forma geral pode-se dizer que os parâmetros α=β, bem como os parâmetros RIO(out) podem afetar expressivamente o funcionamento de uma rede DiffServ,

operando com o mecanismo de controle de fluxo TCP Vegas. Contudo, a interoperação do TCP Vegas com o TCP Reno ainda necessita ser estudada com mais profundidade, visto que o primeiro, em certas circunstâncias, não consegue resultados expressivos na presença do segundo, por mais restritivos que os parâmetros citados sejam ajustados.

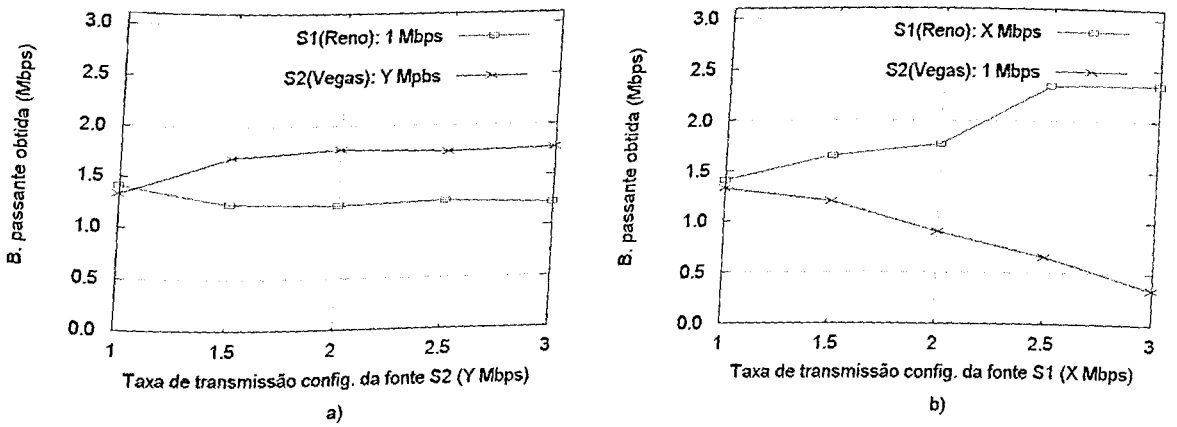

Figura 6.11 – Comparação entre TCP Reno e TCP Vegas em termos de garantia de banda passante:
a) Taxa de transmissão configurada do TCP Reno fixa em 1 Mbps e do TCP Vegas variável de 0 a 3 Mbps
b) Taxa de transmissão configurada do TCP Vegas fixa em 1 Mbps e do TCP Reno variável de 0 a 3 Mbps

Em relação aos valores utilizados para esses parâmetros, conclui-se, com base nos resultados obtidos pelas simulações, que empregando-se o modelo utilizado neste estudo (figura 6.1), os valores que possibilitam o melhor nível de justiça são: $\alpha=\beta=2$ e RIO(out): 3-6.

Por outro lado, com o intuito de se verificar a extensão dos resultados aqui obtidos, considerando-se apenas duas fontes, realiza-se nas sub-seções seguintes o estudo da influência dos parâmetros α e β e RIO(out) numa rede DiffServ com múltiplas fontes.

6.4 JUSTIÇA PARA MÚLTIPLAS FONTES TCP NUMA REDE DiffServ

Considerando-se o parâmetro justiça, o comportamento da rede DiffServ quando operando com múltiplas fontes pode diferenciar substancialmente daquele observado na seção 6.3 em que apenas duas fontes foram analisadas. O estudo abordado nesta seção objetiva

analisar a justiça dessa rede considerando os mecanismos TCP Vegas e TCP Reno e o relacionamento destes com o mecanismo RIO numa rede DiffServ com múltiplas fontes. Também são avaliadas as interoperações entre cada um dos mecanismos TCP citados como o mecanismo UDP. Ainda, para cada uma das simulações, analisa-se as situações de rede congestionada e não congestionada, de modo que os resultados possam ser os mais representativos possíveis.

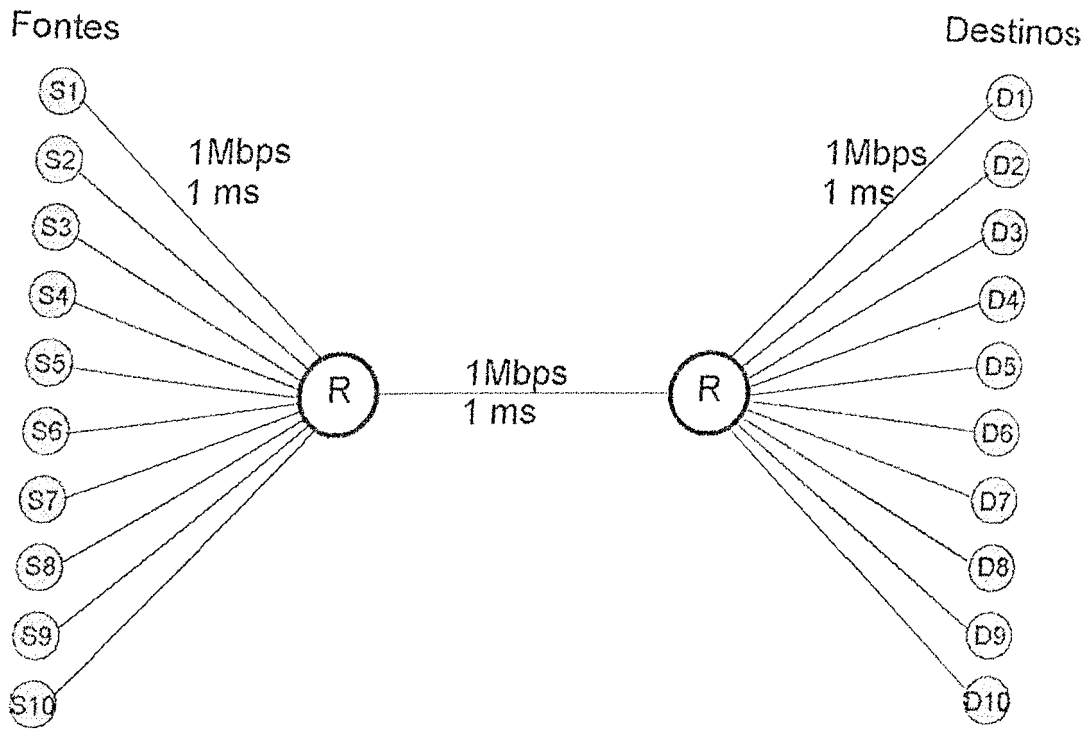

Figura 6.12 – Modelo de rede simulada (múltiplas fontes).

6.4.1 Modelo de simulação

Todas as simulações realizadas neste estudo, com múltiplas fontes, utilizaram o modelo representado na figura 6.12, na qual se observa que há dez fontes (S1, S2,...,S10) enviando dados para dez destinos (D1, D2, ...,D10), totalizando 10 conexões neste cenário. Analogamente ao caso anterior, no qual se tinha apenas duas fontes, referencia-se aqui cada conexão mencionando-se apenas a fonte a ela associada a fim de facilitar a alusão às mesmas.

Desse modo, a conexão da fonte S1 refere-se à conexão entre S1 e D1, a conexão da fonte S2 indica a conexão entre S2 e D2, e assim por diante.

A taxa de transmissão e o retardo de cada enlace são mostrados na referida figura, e as fontes TCP e UDP empregadas, de acordo com cada simulação, geram o mesmo tipo de tráfego citado na seção 6.1. Do mesmo modo, os parâmetros do algoritmo RIO para os pacotes em conformidade, conforme citado anteriormente, são fixados nos valores indicados na tabela 6.1.

A taxa de transmissão configurada de cada fonte, levando-se em consideração os casos de rede congestionada e não congestionada, 225% e 68% de carga, respectivamente, em relação à capacidade de utilização do enlace principal de 1 Mbps (figura 6.12), são apresentadas na tabela 6.2, exceto para as simulações da sub-seção 6.4.6 onde se trata da interoperação do TCP Vegas com o TCP Reno.

Fonte	Taxa de transmissão configurada (Kbps)	
	Rede congestinada	Rede não congestionada
S1	0	0
S2	50	15
S3	100	30
S4	150	45
S5	200	60
S6	250	75
S7	300	90
S8	350	105
S9	400	120
S10	450	135
Total	2250	675
Carga	225%	68%

Tabela 6.2 – Taxa de transmissão configurada de cada fonte.

Todas as simulações tiveram uma duração de 2000 s e os resultados apresentados aqui referem-se à banda passante obtida pelas conexões nos últimos 300 s de simulação. As fontes iniciam suas transmissões de forma seqüencial e distanciadas de 50 s uma da outra, de modo que a fonte S1 entra em operação em $t = 0$ s, a fonte S2 em $t = 50$ s, e assim por diante.

6.4.2 Fontes TCP Reno

Os resultados das simulações apresentados nesta sub-seção mostram o comportamento do algoritmo TCP Reno numa rede DiffServ em termos de distribuição de banda passante. Para isso, foram levados em conta os mesmos três conjuntos de valores distintos para o mecanismo RIO utilizados nas simulações apresentadas anteriormente. Esse mesmo estudo foi realizado em [31], sem considerar as influências dos parâmetros do mecanismo RIO.

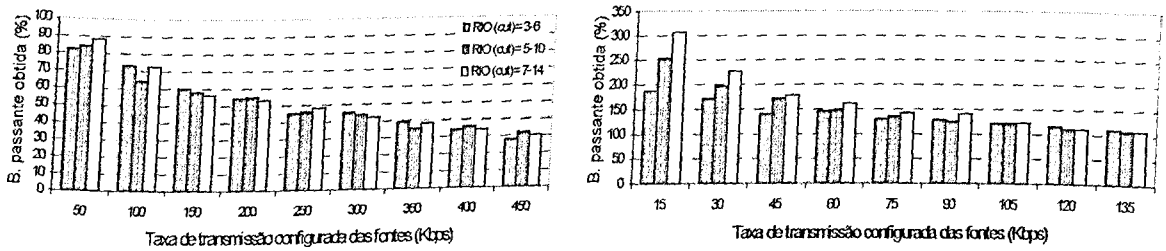

Figura 6.13 – Banda passante para as conexões referentes às fontes TCP Reno:
a) Rede congestionada
b) Rede não congestionada

A figura 6.13 apresenta os resultados das simulações para as conexões referentes às fontes que possuem taxa de transmissão diferente de zero (S2,...,S10), e a tabela 6.3 mostra os valores numéricos obtidos por essas conexões, em porcentagem da taxa de transmissão configurada de cada fonte relacionada. Os resultados para a conexão da fonte S1 nesta tabela são mostrados em Kbps, visto que esta fonte não possui taxa de transmissão configurada e, portanto, o seu fluxo é tratado simplesmente como pertencente à classe de serviço de melhor esforço (BE), ou seja, sem prioridade alguma. O mesmo procedimento é adotado nas sub-seções que se seguem.

De acordo com o que foi obtido em [31], observa-se que em ambos os casos, rede congestionada e não congestionada, figuras 6.13a e 6.13b, respectivamente, as conexões das fontes que possuem taxas de transmissão configuradas menores obtiveram maior banda passante percentual, do que as conexões das fontes com taxas de transmissão configuradas

maiores. Isso se deve, sobretudo, ao fato de que o TCP Reno aumenta a capacidade de transmissão oferecida a cada fonte de forma idêntica (incrementado as suas CWNDs durante a fase *Slow Start* ou *Congestion Avoidance*), independentemente da taxa de transmissão configurada de cada uma. Por isso, as conexões das fontes com menor expectativa de banda passante tendem a alcançar mais rapidamente as suas respectivas taxas configuradas.

Pela figura 6.13b, que se refere à condição de rede não congestionada, pode-se notar que o uso do TCP Reno possibilitou à rede DiffServ garantir 100% da banda passante configurada a todas as conexões, embora a banda passante excedente não tenha sido compartilhada de forma justa entre as mesmas.

Fonte	Banda passante obtida (%)*					
	Rede congestionada RIO(out)			Rede não congestionada RIO(out)		
	3--6	5--10	7--14	3--6	5--10	7--14
S1	0	0	0	1	1	0
S2	82	84	88	186	253	306
S3	73	64	72	170	196	226
S4	60	58	56	140	171	177
S5	54	55	53	145	146	161
S6	45	46	48	129	134	141
S7	45	44	42	127	126	140
S8	39	35	38	121	121	122
S9	34	36	34	115	110	110
S10	28	32	30	111	108	107

** Fonte S1 em Kbps*

Tabela 6.3 – Banda passante obtida pelas conexões TCP Reno.

Quanto aos parâmetros RIO(out), observa-se que os mesmos não impõem variações significativas aos resultados obtidos. Apenas no caso de rede congestionada, figura 6.13b, é que se nota que as conexões das fontes com taxas de transmissão configuradas menores sofreram influências significativas desses parâmetros. Pode-se observar que os valores RIO(out) maiores possibilitaram a essas conexões obter maior quantidade de banda passante. Tal comportamento se justifica pelo fato de que quanto maior forem esses parâmetros, menos pacotes não em conformidade serão descartados pelo mecanismo RIO e, consequentemente,

as conexões, sobretudo as das fontes com taxas de transmissão configuradas menores, obtêm maior banda passante.

6.4.3 Fontes TCP Vegas

Neste estudo o objetivo é verificar a influência dos parâmetros α e β do TCP Vegas e também dos parâmetros RIO(out) sobre a capacidade da rede DiffServ em termos de distribuição de banda passante aos seus usuários, diante de uma rede congestionada e não congestionada. O modelo utilizado nas simulações é aquele apresentado na figura 6.12. Os resultados obtidos para a rede congestionada são mostrados na figura 6.14 e tabela 6.4, e para a rede não congestionada esses resultados são apresentados na figura 6.15 e tabela 6.5.

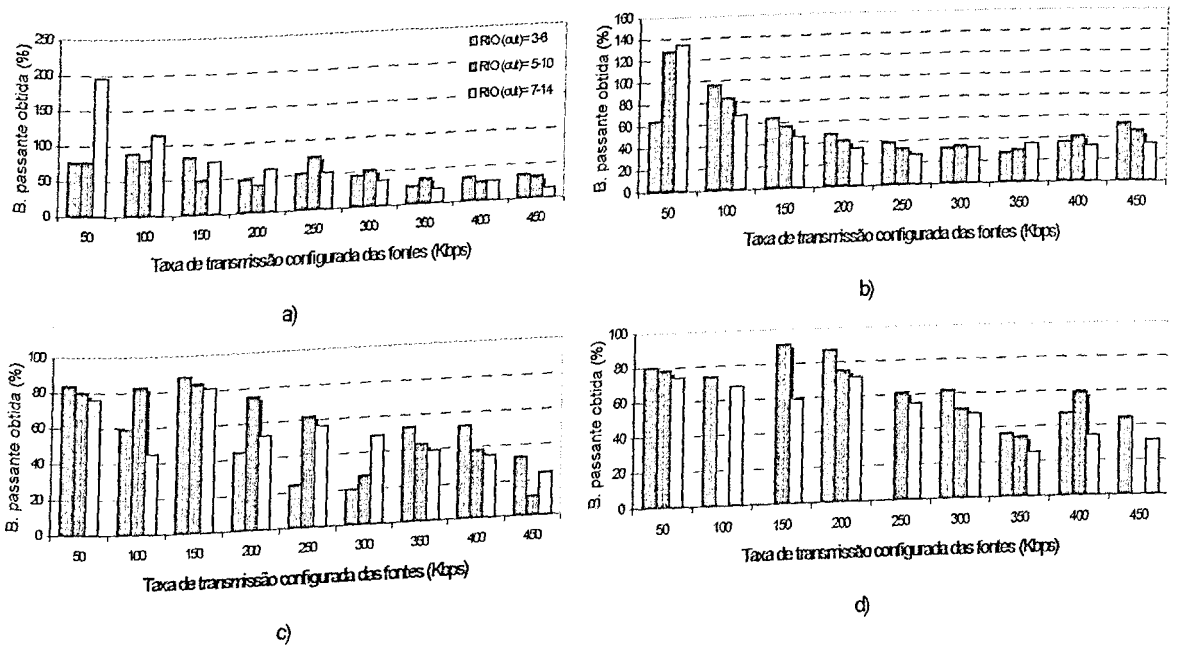

Figura 6.14 – Banda passante para as conexões das fontes TCP Vegas com congestionamento:
a) Parâmetros α=1 e β=3
b) Parâmetros α=β=1
c) Parâmetros α=β=2
d) Parâmetros α=β=3

A figura 6.14 mostra que os parâmetros RIO(out), neste caso, não determinam um comportamento determinístico que possa ser relacionado à quantidade de banda passante configurada para cada conexão. Entretanto, cabe salientar que nas simulações em que se

utilizou valores elevados para os parâmetros α=β, figura 6.14d por exemplo, algumas conexões não obtiveram banda passante alguma com os valores RIO(out) menores e, nessas circunstâncias, diz-se que essas conexões foram anuladas (0 Kbps).

Isso pode ser notado pela figura 6.14d, em que as conexões das fontes S3 e S10, com taxas de transmissão configuradas de 100 e 450 Kbps, respectivamente, não obtiveram banda passante alguma com o RIO(out): 5-10 e, o mesmo pode ser observado para as conexões das fontes S4 e S6, 150 e 250 Kbps, respectivamente, simuladas com o RIO(out): 3-6.

Fonte	Taxa configurada (Kbps)	Banda passante obtida (%)*											
		Alfa=1 Beta=3			Alfa=1 Beta=1			Alfa=2 Beta=2			Alfa=3 Beta=3		
		3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14
S1	0	0	0	69	1	12	149	1	1	0	0	0	0
S2	50	76	76	196	64	128	134	84	80	76	80	78	74
S3	100	88	79	114	96	84	68	59	83	45	74	0	68
S4	150	82	48	74	64	56	46	88	84	81	0	91	59
S5	200	47	38	60	48	42	34	44	75	52	87	75	71
S6	250	52	75	52	38	33	27	24	62	56	0	61	55
S7	300	44	52	35	32	34	32	20	27	49	62	51	48
S8	350	25	36	19	27	29	35	53	43	39	36	34	25
S9	400	34	27	27	36	41	32	52	38	35	47	59	34
S10	450	34	31	14	53	46	34	33	11	24	44	0	31

* *Fonte S1 em Kbps*

Tabela 6.4 – Banda passante obtida pelas conexões TCP Vegas com rede congestionada.

Em relação aos parâmetros α e β, nota-se que, diferentemente do que foi observado no estudo anterior com apenas duas fontes, os valores de α=1 e β=3, possibilitam um comportamento similar ao comportamento obtido com o TCP Reno. Por outro lado, a agressividade de algumas conexões em detrimento de outras, observada no estudo anterior para valores maiores de α=β, é também observada aqui. A figura 6.14d mostra que para que as conexões citadas no parágrafo anterior obtenham alguma banda passante é necessário que se utilize valores de RIO(out) de no mínimo 7-14.

A principal causa dessa diferença de comportamento do TCP Vegas em relação ao uso dos parâmetros α e β com valores diferentes (α=1 e β=3), comparativamente ao estudo

anterior no qual se considerou apenas duas fontes, é a variação contínua sofrida pelos RTTs dos pacotes. E, isso é uma conseqüência do elevado número de fluxos (múltiplas fontes) competindo pela banda passante da rede nesta circunstância.

Nesse mesmo princípio, considerando-se a rede não congestionada pode-se notar também neste caso, por meio da figura 6.15, que o desempenho da rede , quanto à justiça, diminuiu sensivelmente para valores pequenos de RIO(out) e tornou-se ainda pior à medida que os parâmetros α=β aumentaram. As figuras 6.15c e 6.15d mostram que com o RIO(out): 3-6 somente duas conexões, referentes à S4 e S8 (45 e 105 Kbps, respectivamente), obtiveram banda passante, enquanto as demais receberam 0 Kbps (foram anuladas).

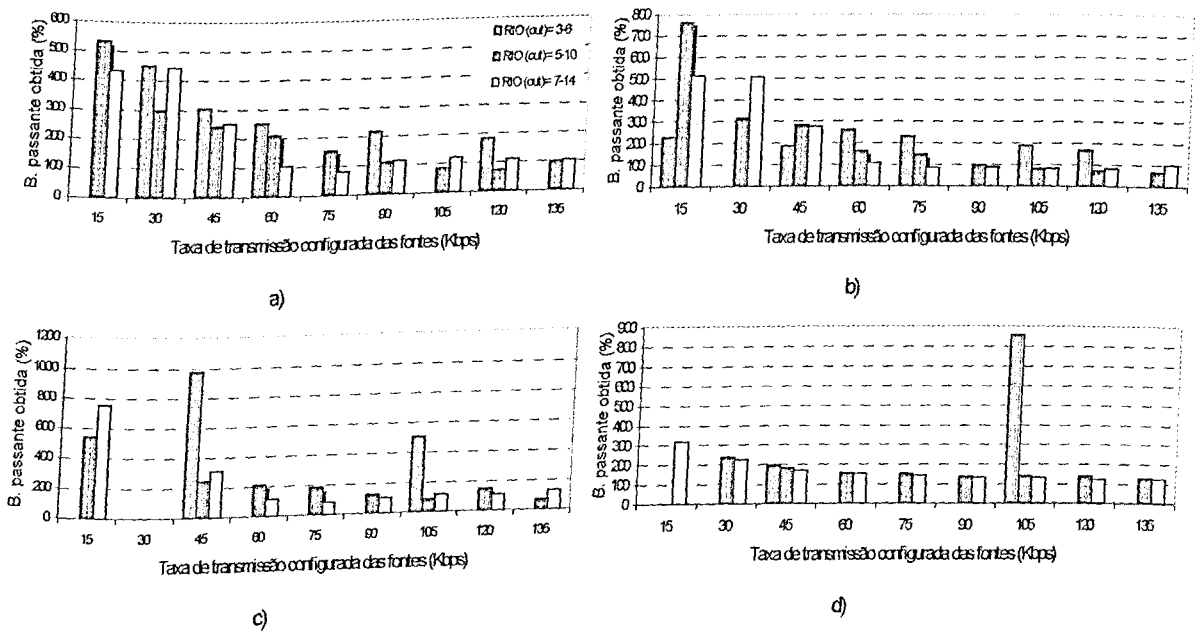

Figura 6.15 – Banda passante para as conexões das fontes TCP Vegas sem congestionamento:
a) Parâmetros α=1 e β=3
b) Parâmetros α=β=1
c) Parâmetros α=β=2
d) Parâmetros α=β=3

A análise da tabela 6.5 permite constatar-se que a conexão da fonte S1, com 0 Kbps de banda passante configurada, também obteve alguma banda passante para α=1 e β=3. Isto sugere que o uso destes valores para α e β, faz com que as conexões das fontes que possuem taxa de transmissão configurada diferente de zero sejam menos agressivas.

De uma forma geral pode-se notar pela figura 6.15 que o RIO(out): 7-14 provê melhor justiça, embora a figura 6.15c mostre que a conexão da fonte S3 (30 Kbps) não consegue obter banda passante alguma.

Fonte	Taxa configurada (Kbps)	Banda passante obtida (%)*											
		Alfa=1 Beta=3			Alfa=1 Beta=1			Alfa=2 Beta=2			Alfa=3 Beta=3		
		3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14
S1	0	44	62	67	74	101	117	0	64	92	0	9	18
S2	15	0	533	433	226	760	513	0	546	753	0	0	320
S3	30	450	293	443	0	310	506	0	0	0	0	240	226
S4	45	304	242	248	184	280	275	973	242	306	195	184	168
S5	60	250	208	101	258	160	106	0	208	110	0	156	153
S6	75	0	152	81	230	146	85	0	180	81	0	152	142
S7	90	215	108	113	0	97	87	0	123	103	0	136	131
S8	105	0	83	118	186	83	80	501	85	114	854	138	131
S9	120	180	73	109	167	73	82	0	148	109	0	138	124
S10	135	0	98	101	0	64	100	0	66	131	0	128	125

* *Fonte S1 em Kbps*

Tabela 6.5 – Banda passante obtida pelas conexões TCP Vegas com rede não congestionada.

Portanto, o TCP Vegas com múltiplas fontes numa rede DiffServ não oferece justiça adequada para valores elevados de α=β, tampouco para valores muito pequenos de RIO(out). Por outro lado, valores de $\alpha \neq \beta$, nesse caso α=1 e β=3, podem oferecer uma justiça aceitável comparativamente aos resultados obtidos para o TCP Reno.

6.4.4 TCP Reno com tráfego UDP

Trata-se aqui da interoperação do TCP Reno com o algoritmo UDP sobre uma rede DiffServ a fim de se estudar a capacidade dessa rede em garantir a banda passante configurada às conexões empregadas pelas suas fontes, em situações de congestionamento e de não congestionamento. O modelo que foi empregado nas simulações é o apresentado na figura 6.11, considerando-se cinco fontes TCP Reno (S1, S3, S5, S7, S9) e cinco fontes UDP (S2, S4, S6, S8, S10), com as suas respectivas taxas de transmissão configuradas indicadas na figura 6.16.

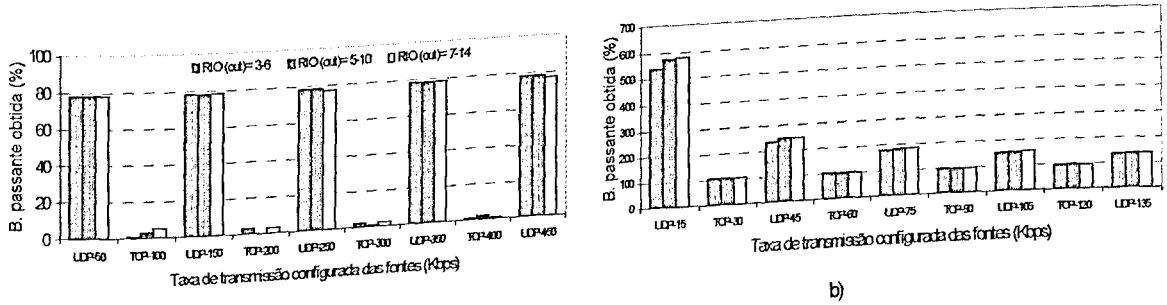

Figura 6.16 – Banda passante para as conexões das fontes TCP Reno mais UDP:
a) Rede congestionada
b) Rede não congestionada

Os resultados obtidos pelas simulações são mostrados na figura 6.16 e na tabela 6.6. Estes resultados estão de acordo com os obtidos em [31] e, conforme mostra a figura 6.16a, sob congestionamento as conexões TCP Reno praticamente não obtém banda passante alguma diante das conexões UDPs, as quais compartilham a banda passante disponível da rede de forma justa, em que se tem aproximadamente 80 Kbps (tabela 6.6) para cada conexão.

Fonte	Tráfego	Banda passante obtida (%)*					
		Rede congestionada RIO(out)			Rede não congestionada RIO(out)		
		3--6	5--10	7--14	3--6	5--10	7--14
S1	TCP	0	0	0	0	0	0
S2	UDP	78	78	78	540	573	580
S3	TCP	1	3	5	106	106	106
S4	UDP	78	78	78	235	246	248
S5	TCP	3	0	3	103	101	100
S6	UDP	78	78	77	176	181	182
S7	TCP	2	1	2	97	98	96
S8	UDP	78	78	78	149	153	154
S9	TCP	1	2	1	96	98	95
S10	UDP	78	78	77	134	137	138

* *Fonte S1 em Kbps*

Tabela 6.6 – Banda passante obtida pelas conexões TCP Vegas com rede não congestionada.

Isso ocorre em função da atuação do mecanismo de controle de fluxo do TCP Reno, o qual faz com que as conexões das fontes TCP Reno recebam uma banda passante menor diante da presença do tráfego UDP, o qual é desprovido de mecanismo de controle de fluxo.

Daí dizer-se que o algoritmo UDP proporciona um tráfego agressivo nas redes de comunicações em que trafega.

No caso da figura 6.16b, na qual são apresentados os resultados da simulação considerando-se a rede livre de congestionamento, nota-se que o tráfego UDP ainda supera o TCP. No entanto, nesse caso a rede DiffServ garante a banda passante configurada para cada conexão (100% de banda passante). E, o tráfego UDP nesse caso não é distribuído igualmente entre todas as conexões UDP, como ocorreu no caso anterior.

Em ambos os casos, rede congestionada e não congestionada, observa-se que os parâmetros RIO(out) não oferecem influência significativa sobre a banda passante alcançada pelas conexões.

6.4.5 TCP Vegas com tráfego UDP

Analogamente ao caso da sub-seção anterior, a intenção deste estudo é a análise do comportamento da rede DiffServ sob tráfego TCP Vegas juntamente com tráfego UDP, considerando-se a influência dos parâmetros α e β e dos parâmetros RIO(out) sobre o desempenho dessa rede quando em congestionamento e não em congestionamento.

O modelo de simulação utilizado é o mesmo da figura 6.12 em que as fontes S1, S3, S5, S7 e S9 são TCP Vegas, enquanto que as demais são UDPs. Os resultados obtidos são apresentados na figura 6.17 e tabela 6.7 para a rede congestionada e na figura 6.18 e tabela 6.8 para o caso de rede não congestionada.

No caso da rede congestionada observa-se um comportamento muito próximo ao obtido no estudo do TCP Reno, no qual as conexões das fontes UDP predominam totalmente sobre as conexões associadas às fontes TCP. Observa-se também que, como no caso anterior, os parâmetros RIO(out) não produzem efeito expressivo sobre a justiça dessa rede. Identicamente, os valores usados para os parâmetros α e β são indiferentes. Tais resultados já

eram esperados, visto que o TCP Vegas atua de forma similar ao TCP Reno diante de tráfegos agressivos como o UDP, no que diz respeito ao controle do seu fluxo transmitido.

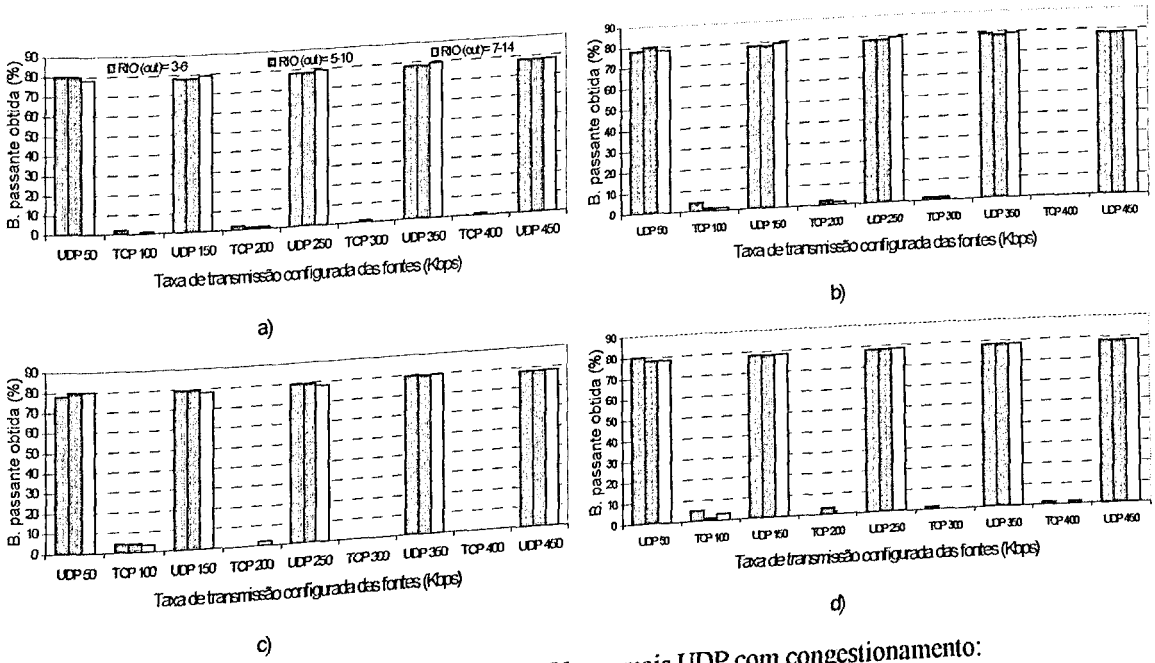

Figura 6.17 – Banda passante para as conexões TCP Vegas mais UDP com congestionamento:

a) Parâmetros α=1 e β=3
b) Parâmetros α=β=1
c) Parâmetros α=β=2
d) Parâmetros α=β=3

Por outro lado, os resultados obtidos com a rede congestionada indicam que apesar de se assemelharem aos obtidos para o TCP Reno, em que a rede DiffServ garante a banda passante configurada às conexões das fontes TCP (100%) e compartilha a banda excedente de forma desproporcional entre as conexões das fontes UDP, tais resultados dependem significativamente dos valores usados para os parâmetros α e β e RIO(out).

As figuras 6.18c e 6.18d mostram que a agressividade dos valores mais altos para os parâmetros α=β, nesse caso α=β=2 e α=β=3, fazem com que a conexão da fonte S9 (120 kbps) não obtenha banda passante alguma, para nenhum valor de RIO(out) utilizado. E, se for considerado o RIO(out): 3-6, nota-se pela figura 6.18d que dentre todas as conexões

associadas às fontes TCP Vegas, somente a conexão da fonte S7 (90 Kbps) obteve banda passante, as demais conexões TCP foram anuladas.

Fonte	Tráfego	Taxa configurada (Kbps)	Banda passante obtida (%)*											
			Alfa=1 Beta=3			Alfa=1 Beta=1			Alfa=2 Beta=2			Alfa=3 Beta=3		
			3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14
S1	TCP	0	40	40	39	0	0	0	0	0	0	0	0	0
S2	UDP	50	80	80	78	78	80	78	78	80	80	80	78	78
S3	TCP	100	2	0	1	4	1	1	4	4	3	5	1	3
S4	UDP	150	78	78	79	78	78	79	79	79	78	78	78	78
S5	TCP	200	2	1	1	0	2	1	0	0	2	0	3	0
S6	UDP	250	78	78	79	78	78	79	79	79	78	78	78	78
S7	TCP	300	0	1	0	1	1	0	0	0	0	1	0	0
S8	UDP	350	78	78	79	79	78	79	79	79	79	78	78	78
S9	TCP	400	0	1	0	0	0	0	0	0	0	1	0	1
S10	UDP	450	78	78	78	78	78	78	78	78	78	78	78	78

* *Fonte S1 em Kbps*

Tabela 6.7 – Banda passante obtida pelas conexões TCP Vegas e UDP com rede congestionada.

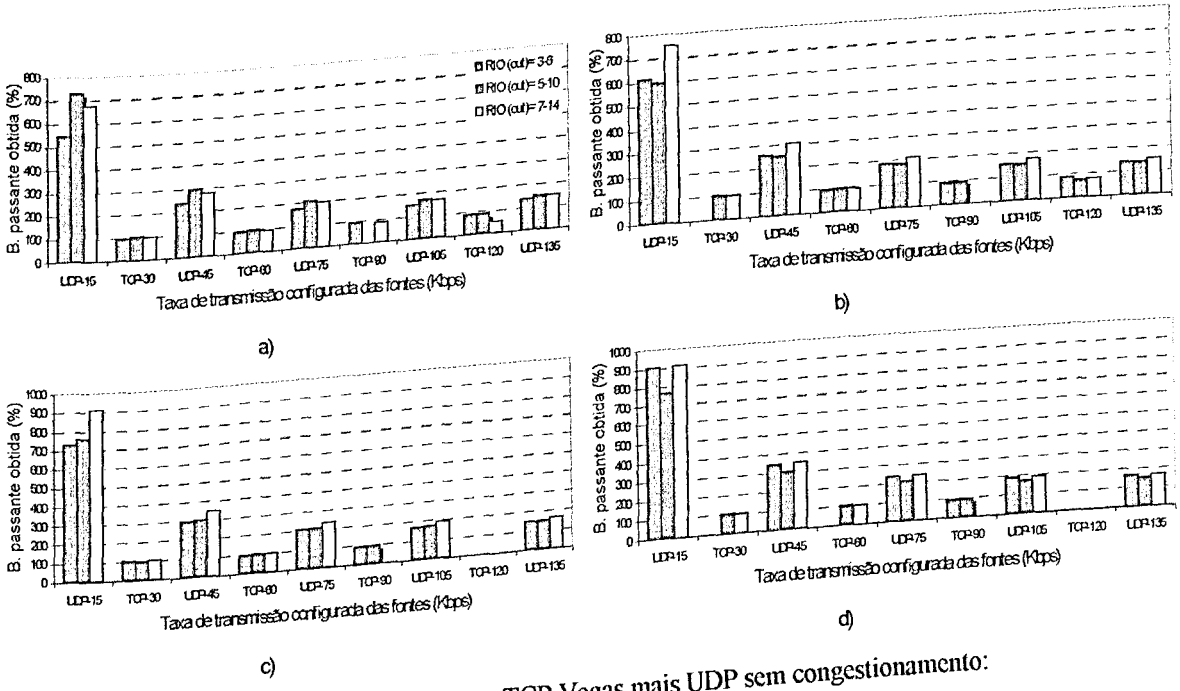

Figura 6.18 – Banda passante para as conexões TCP Vegas mais UDP sem congestionamento:

a) Parâmetros α=1 e β=3
b) Parâmetros α=β=1
c) Parâmetros α=β=2
d) Parâmetros α=β=3

De um modo direto, pode-se dizer que a escolha dos valores a serem utilizados para os parâmetros RIO(out) determinam quais conexões serão anuladas, o que também depende do

tráfego na rede. Os resultados obtidos nesta sub-seção, considerando-se as quatro simulações da figura 6.18, apresentadas de forma numérica na tabela 6.8, permitem constatar-se que com os conjuntos de valores RIO(out): 3-6 e 7-14, cinco conexões TCP foram anuladas, enquanto que com o RIO(out): 5-10, apenas três delas receberam 0 Kbps (desconsiderando-se a fonte S1, que é submetida ao serviço BE).

Fonte	Tráfego	Taxa configurada (Kbps)	Banda passante obtida (%)*											
			Alfa=1 Beta=3			Alfa=1 Beta=1			Alfa=2 Beta=2			Alfa=3 Beta=3		
			3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14
S1	TCP	0	0	0	0	0	0	0	0	0	0	0	0	0
S2	UDP	15	546	733	673	613	593	753	733	760	913	906	766	913
S3	TCP	30	96	103	100	0	103	100	103	100	103	0	106	103
S4	UDP	45	237	295	277	257	251	302	297	304	348	346	302	348
S5	TCP	60	95	98	93	96	98	95	96	100	100	0	101	100
S6	UDP	75	176	208	198	188	184	212	209	213	238	237	212	238
S7	TCP	90	94	0	87	93	94	0	93	94	0	94	93	0
S8	UDP	105	150	171	165	157	155	174	172	174	191	191	174	191
S9	TCP	120	88	86	50	87	77	79	0	0	0	0	0	0
S10	UDP	135	135	151	146	141	140	152	151	153	165	164	152	165

* *Fonte S1 em Kbps*

Tabela 6.8 – Banda passante obtida pelas conexões TCP Vegas e UDP com rede não congestionada.

Portanto, conclui-se que com essa configuração de simulação é inevitável que algumas conexões relacionadas às fontes TCP Vegas sejam anuladas, embora haja banda passante disponível para que todas recebam suas respectivas banda passante configuradas.

6.4.6 TCP Reno e TCP Vegas

O estudo desenvolvido nesta sub-seção tem por objetivo avaliar a interoperação entre os algoritmos TCP Reno e TCP Vegas numa rede DiffServ com múltiplas fontes. Para esse fim, utilizou-se o mesmo modelo de simulação apresentado na figura 6.12, considerando-se as fontes ímpares como sendo TCP Reno e as pares como sendo TCP Vegas.

Entretanto, neste caso, diferentemente das simulações anteriores, a taxa de transmissão configurada de cada fonte, são aquelas indicas nas tabelas 6.9 e 6.10, nas quais pode-se notar

Fonte	Algoritmo TCP	Taxa configurada (Kbps)	Banda passante obtida (%)*											
			Alfa=1 Beta=3 RIO(out)			Alfa=1 Beta=1 RIO(out)			Alfa=2 Beta=2 RIO(out)			Alfa=1 Beta=3 RIO(out)		
			3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14
S1	Reno	70	85	85	85	88	91	88	84	84	84	81	84	81
S2	Vegas		80	85	58	85	87	87	75	81	74	77	90	78
S3	Reno	140	65	65	62	70	69	68	60	68	65	63	65	64
S4	Vegas		57	60	62	49	48	45	69	58	41	0	71	72
S5	Reno	210	60	58	56	63	63	64	50	60	55	53	57	58
S6	Vegas		45	37	45	29	29	36	51	35	52	53	57	64
S7	Reno	280	49	53	51	58	57	56	45	52	53	46	49	49
S8	Vegas		33	30	29	24	23	24	38	38	36	48	50	0
S9	Reno	350	45	49	46	57	58	55	40	40	41	44	44	44
S10	Vegas		24	24	30	18	20	20	29	27	30	40	0	38

Tabela 6.9 – Banda passante do tráfego composto por fontes TCP Vegas e Reno com rede congestionada.

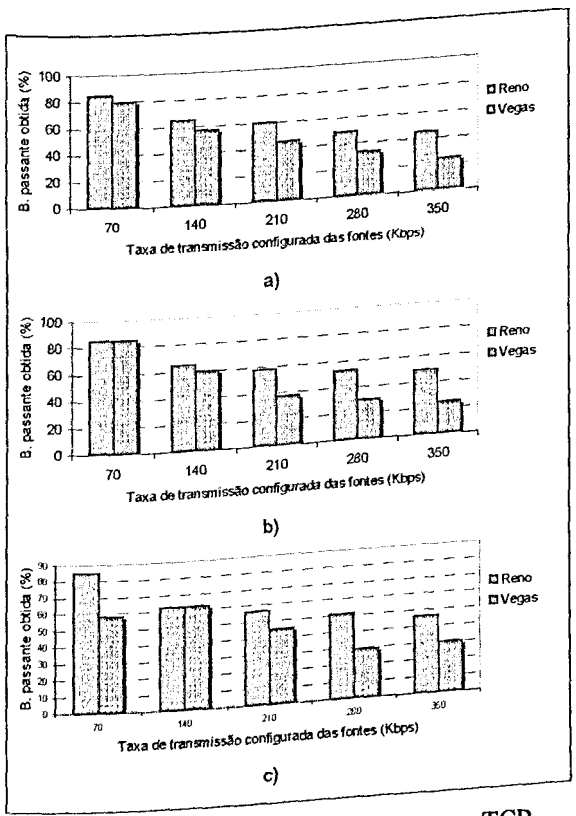

Figura 6.19 – Banda passante para as fontes TCP Reno e TCP Vegas. Parâmetros α=1 e β=3 com congestionamento:
a) RIO(out): 3-6
b) RIO(out): 5-10
c) RIO(out): 7-14

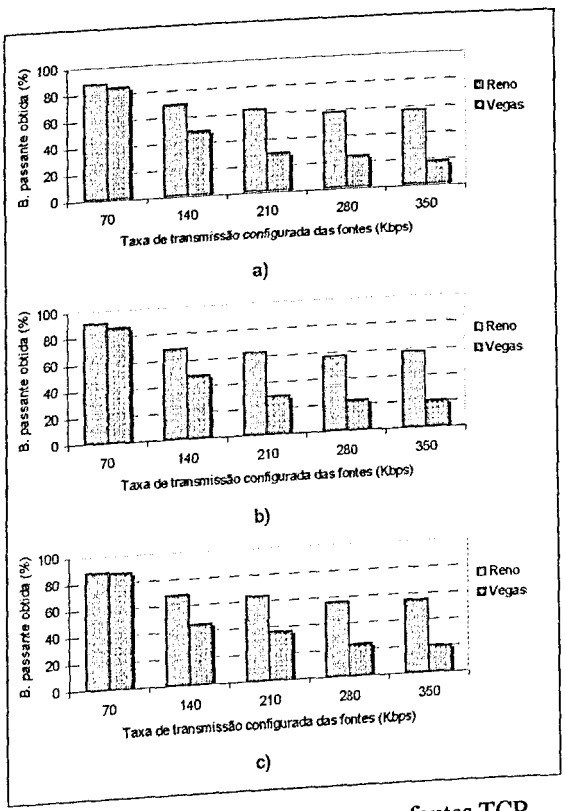

Figura 6.20 – Banda passante para as fontes TCP Reno e TCP Vegas. Parâmetros α=1 e β=1 com congestionamento:
a) RIO(out): 3-6
b) RIO(out): 5-10
c) RIO(out): 7-14

que há uma fonte TCP Reno e outra TCP Vegas, para cada taxa configurada. A finalidade desta distribuição é facilitar a comparação citada. As fontes S1 e S2 iniciam a transmitir no

instante t = 0 s, S3 e S4 em t = 50 s e assim sucessivamente a cada 50 s. Os demais dados das simulações, são os mesmos descritos na seção 6.2.

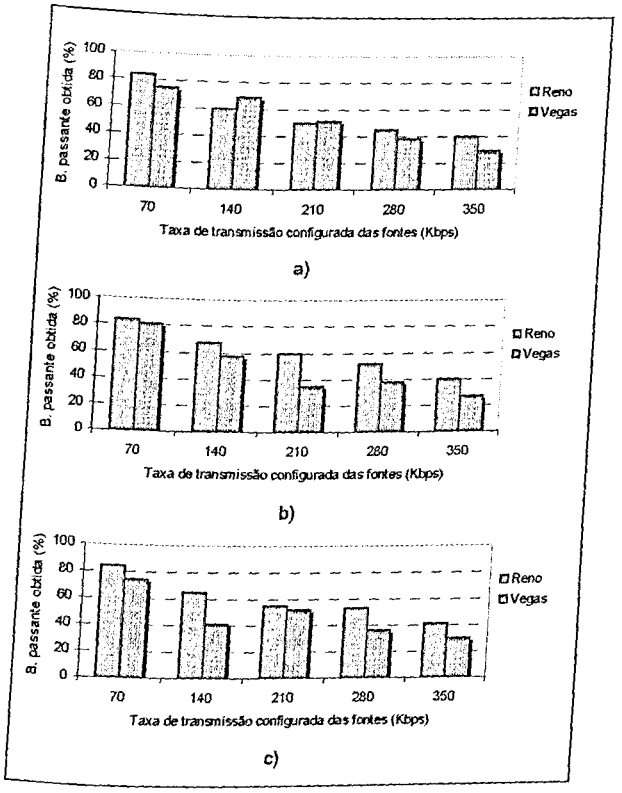

Figura 6.21 – Banda passante para as fontes TCP Reno e TCP Vegas. Parâmetros α=2 e β=2 com congestionamento:
a) RIO(out): 3-6
b) RIO(out): 5-10
c) RIO(out): 7-14

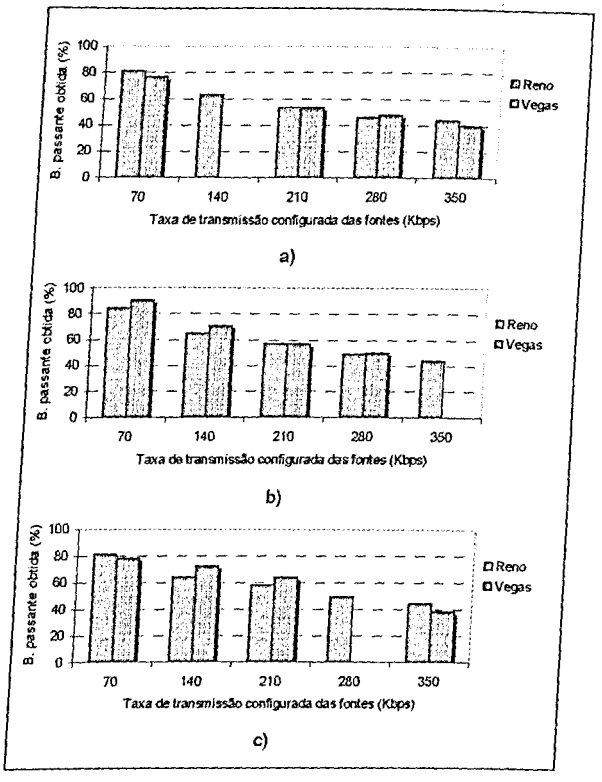

Figura 6.22– Banda passante para as fontes TCP Reno e TCP Vegas. Parâmetros α=3 e β=3 com congestionamento:
a) RIO(out): 3-6
b) RIO(out): 5-10
c) RIO(out): 7-14

Analogamente aos casos anteriores, foram consideradas as mesmas variações para os parâmetros α e β e RIO(out), como também as situações de rede congestionada e não congestionada (210% e 60% de carga). As figuras 6.19, 6.20, 6.21 e 6.22 e tabela 6.9 apresentam os resultados para a rede congestionada e para a rede não congestionada os resultados encontram-se nas figuras 6.23, 6.24, 6.25 e 6.26 e tabela 6.10. Analisando-se os resultados para a rede congestionada, por intermédio das figuras 6.19, 6.20, 6.21 e 6.22, em que os parâmetros α=1 e β=3, α=β=1, α=β=2, α=β=3, respectivamente, foram utilizados, pode-se notar que em geral o TCP Reno supera o TCP Vegas, o que constata a característica

agressiva do TCP Reno frente ao TCP Vegas, como também ocorreu no estudo anterior empregando-se apenas duas fontes.

Fonte	Algoritmo TCP	Taxa configurada (Kbps)	Banda passante obtida (%)*											
			Alfa=1 Beta=3 RIO(out)			Alfa=1 Beta=1 RIO(out)			Alfa=2 Beta=2 RIO(out)			Alfa=1 Beta=3 RIO(out)		
			3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14	3--6	5--10	7--14
S1	Reno	20	220	295	360	355	280	340	430	320	285	610	535	450
S2	Vegas		205	275	300	0	250	250	375	285	255	0	0	0
S3	Reno	40	177	232	227	267	202	205	262	220	195	370	317	262
S4	Vegas		152	212	0	215	155	177	0	145	170	0	0	0
S5	Reno	60	151	191	170	213	153	166	220	181	160	293	250	211
S6	Vegas		136	161	170	0	136	156	0	170	155	0	0	0
S7	Reno	80	152	157	151	200	138	133	187	161	140	258	210	177
S8	Vegas		115	145	156	0	132	145	170	0	143	0	228	178
S9	Reno	100	135	147	135	178	127	135	171	147	140	226	184	159
S10	Vegas		103	0	146	141	132	136	0	139	135	0	0	169

Tabela 6.10 – Banda passante do tráfego composto por fontes TCP Vegas e Reno com rede não congestionada.

Conforme citado anteriormente, isso ocorre devido à característica do TCP Reno de aumentar a sua janela de congestionamento até que ocorram perdas na rede a fim de detectar congestionamento na mesma, o que lhe confere agressividade de comportamento quando operando em conjunto com o TCP Vegas. Cabe salientar que esse mesmo resultado também foi encontrado em *[34, 49], para uma rede convencional.*

Em relação aos parâmetros α e β, nota-se a mesma tendência observada nas simulações anteriores, nas quais $\alpha=\beta=3$ causa maior agressividade por parte de algumas conexões, o que faz com que outras sejam anuladas. Isso pode ser visto pela figura 6.22, nas quais todas as simulações apresentaram pelos menos uma conexão TCP Vegas com banda passante nula. Também pode-se observar que neste caso os valores $\alpha=1$ e $\beta=3$, figura 6.19, possibilita um funcionamento compatível com os obtidos com $\alpha=\beta=1$ e $\alpha=\beta=2$.

Analisando-se os parâmetros RIO(out), observa-se pela figura 6.21a, que o melhor desempenho do TCP Vegas em relação ao TCP Reno, no que diz respeito à mínima diferença

entre ambas as vazões (para cada par de conexão, uma Reno e outra Vegas), foi obtido para o RIO(out): 3-6 e $\alpha=\beta=2$ (figura 6.21).

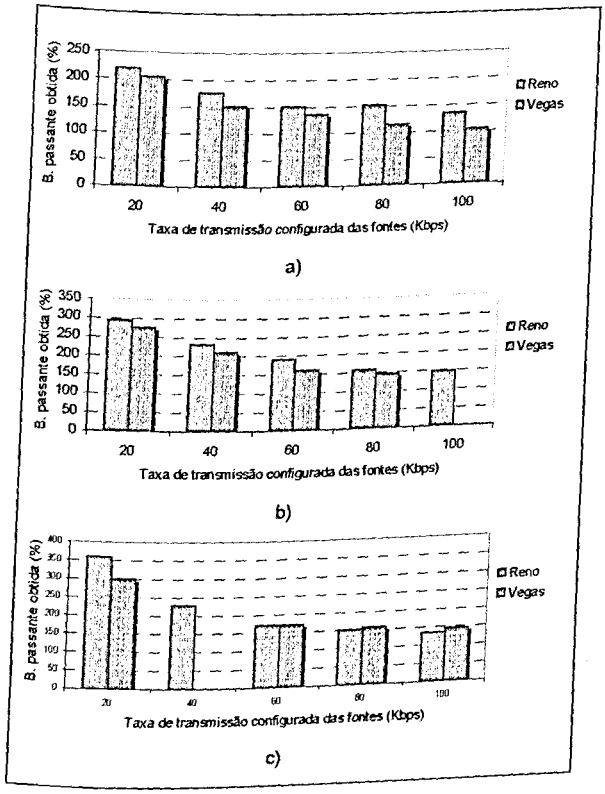

Figura 6.23 – Banda passante para as fontes TCP Reno e TCP Vegas. Parâmetros $\alpha=1$ e $\beta=3$ sem congestionamento:
a) RIO(out): 3-6
b) RIO(out): 5-10
c) RIO(out): 7-14

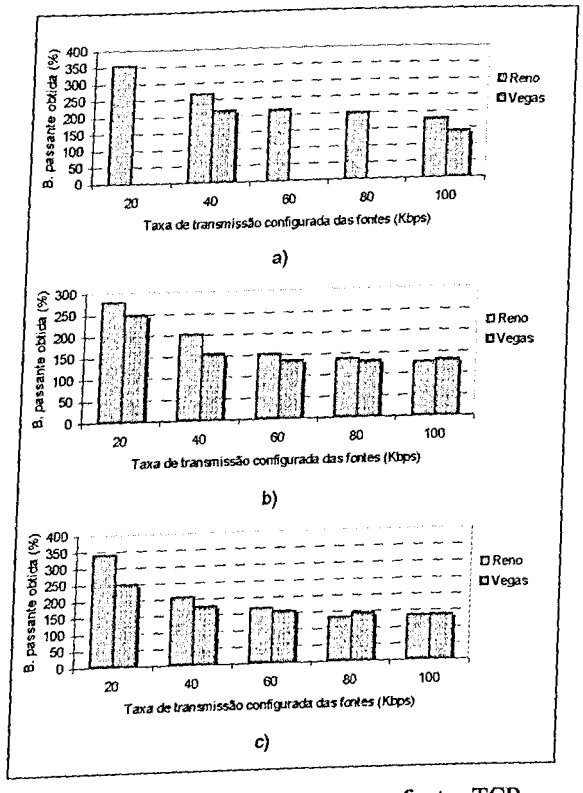

Figura 6.24 – Banda passante para as fontes TCP Reno e TCP Vegas. Parâmetros $\alpha=1$ e $\beta=1$ sem congestionamento:
a) RIO(out): 3-6
b) RIO(out): 5-10
c) RIO(out): 7-14

Enfim, os resultados sugerem que as situações nas quais se obtém melhor desempenho da rede, em relação à minimização da agressividade do TCP Reno sobre o TCP Vegas, são conseguidas com valores pequenos para o RIO(out) e para $\alpha=\beta$ menor do que 3, ou mesmo $\alpha=1$ e $\beta=3$.

Considerando-se agora os resultados obtidos para a rede não congestionada, pode-se notar pelas figuras 6.23, 6.24, 6.25 e 6.26, que à medida que se aumentou o valor dos parâmetros α e β, as conexões das fontes TCP Vegas foram perdendo a capacidade de obter

banda passante, chegando ao ponto extremo de nenhuma delas conseguir transmitir dados, como mostrado pela figura 6.26a, a qual foi obtida empregando-se $\alpha=\beta=3$ e RIO(out): 3-6.

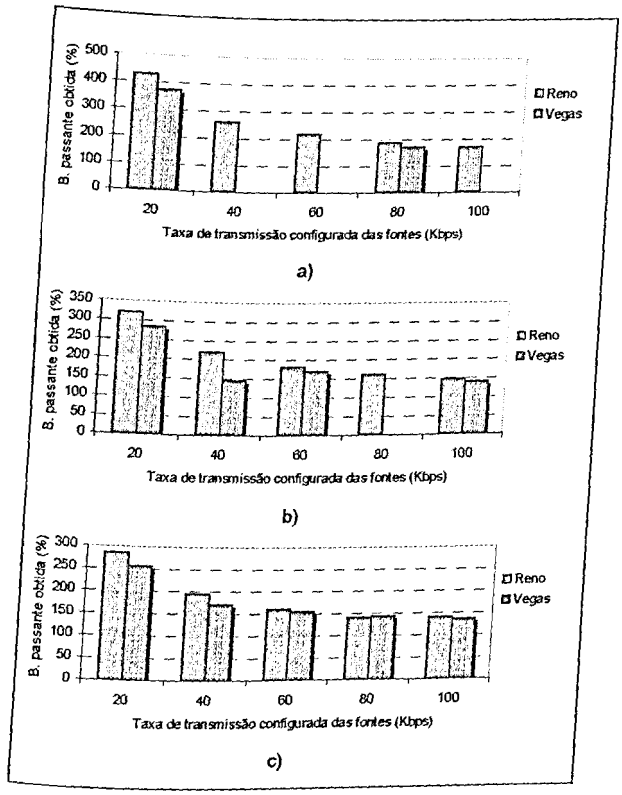

Figura 6.25 – Banda passante para as fontes TCP Reno e TCP Vegas. Parâmetros $\alpha=2$ e $\beta=2$ sem congestionamento:
a) RIO(out): 3-6
b) RIO(out): 5-10
c) RIO(out): 7-14

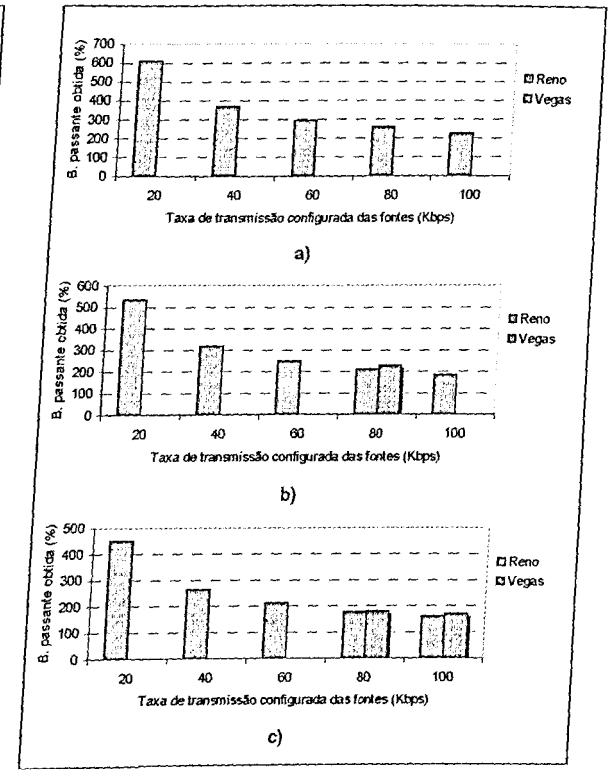

Figura 6.26 – Banda passante para as fontes TCP Reno e TCP Vegas. Parâmetros $\alpha=3$ e $\beta=3$ sem congestionamento:
a) RIO(out): 3-6
b) RIO(out): 5-10
c) RIO(out): 7-14

Em relação à influência dos parâmetros RIO(out) sobre a justiça da rede, observa-se pelas mesmas figuras citadas no parágrafo anterior que, dentre as simulações que foram realizadas empregando-se $\alpha=1$ e $\beta=3$, o melhor resultado foi obtido naquela em que se utilizou o RIO(out): 3-6, fig. 6.23a, visto que neste caso nenhuma conexão foi anulada. Em contrapartida, as simulações realizadas com $\alpha=\beta$, figuras 6.24, 6.25 e 6.26, mostram que os conjuntos de valores RIO(out) maiores possibilitaram resultados melhores. A figura 6.25c apresenta o melhor resultado, o qual corresponde aos parâmetros RIO(out): 7-14 e $\alpha=\beta=2$.

Ainda considerando-se os parâmetros RIO(out), pode-se verificar pela tabela 6.10 que para todas as simulações realizadas a quantidade de conexões que foram anuladas para os conjuntos de valores 3-6, 5-10 e 7-14, foram 11, 6 e 3 fontes, respectivamente. O que mostra que realmente nessas condições a dupla 7-14 funciona melhor.

De uma forma geral pode-se dizer que na operação do DiffServ com múltiplas fontes, os parâmetros RIO(out) muito pequenos não funcionam muito bem e os parâmetros $\alpha=\beta$ maiores do que 3 também provocam menor desempenho nessas redes. Por outro lado, os parâmetros $\alpha=1$ e $\beta=3$ mostram-se muito mais praticáveis nesse caso do que no estudo anterior em que se utilizou apenas duas fontes.

6.5 CONCLUSÕES

O estudo apresentado neste capítulo possibilitou inicialmente constatar-se a extensão de alguns resultados obtidos em pesquisas anteriores [33, 34, 49,52] a respeito do TCP Vegas numa rede convencional, para a rede DiffServ. Adicionalmente também foi possível a determinação de algumas características exclusivas de um ambiente DiffServ operando com o TCP Vegas, até então não estudadas.

Os resultados obtidos permitem concluir que nas situações em que se tem apenas duas fontes TCP Vegas competindo por banda passante sobre uma rede DiffServ, é imprescindível que se utilize os parâmetros $\alpha=\beta$ para o TCP Vegas e valores extremamente pequenos para parâmetros do mecanismo RED associado aos pacotes não em conformidade do algoritmo RIO (RIO(out)), a fim de se conseguir uma justiça adequada nesse ambiente. Esses mesmos valores permitem também minimizar significativamente a já conhecida agressividade do TCP Reno sobre o TCP Vegas nas circunstâncias citadas.

Em relação ao uso de múltiplas fontes sobre uma rede DiffServ, os resultados mostraram que os parâmetros $\alpha\neq\beta$ (usualmente $\alpha=1$ e $\beta=3$) para o TCP Vegas também podem ser utilizados e fornecem resultados próximos aos obtidos com o TCP Reno. Quanto aos

parâmetros do algoritmo RIO citados, esses não devem ser muito pequenos como no caso em que se utiliza apenas duas fontes, sob pena de anularem muitas conexões. No uso conjunto de fontes TCP Reno e TCP Vegas nota-se dificuldades em se conseguir equilíbrio entre as conexões dessas fontes por meio do ajuste dos parâmetros mencionados, de modo que as conexões das fontes TCP Reno tendem a superar as fontes TCP Vegas na maioria dos casos.

Nas simulações com em que se considerou o tráfego UDP, observou-se que este afeta os dois algoritmos TCP aqui estudados de maneira muito similar, em que estes são penalizados pela agressividade do UDP. Isto sugere a necessidade de policiamento na borda da rede com o propósito de moldar tal tráfego.

Em geral, considerando-se todos os casos estudados, observa-se que os parâmetros α=β não devem ter valor elevado, visto que isso causa instabilidade na rede e tende a anular algumas conexões. O melhor valor a ser empregado em todos os casos parece ser α=β=2. Por outro lado, os parâmetros do algoritmo RIO estudados neste trabalho mostraram não ter influência significativa sobre as simulações em que foram empregadas apenas fontes TCP Reno.

Nesse sentido, conclui-se que o TCP Vegas, comparativamente ao TCP Reno, embora possa oferecer uma justiça razoável para o caso em que se tem duas fontes sobre uma rede DiffServ, o mesmo apresenta uma grande deficiência nesse sentido no caso em que se tem múltiplas fontes. Isso sugere que o uso do TCP Vegas num ambiente DiffServ necessita de mecanismos adicionais ou mesmo alterações em seu algoritmo a fim de melhorar o seu desempenho no contexto estudado nesta pesquisa.

Em princípio, dois mecanismos são sugeridos como possíveis soluções em se tratando de viabilizar o uso do TCP Vegas com a rede DiffServ: o mecanismo ECN (*Explicit Congestion Notification*), de acordo com o que foi proposto em [44] e o mecanismo REM

(*Random Early Marking*) como apresentado em [35]. No entanto, a efetiva viabilidade dessas implementações demanda estudos específicos e isso fica sugerido a pesquisas futuras.

CAPÍTULO VII

CONCLUSÕES GERAIS

O ritmo elevado no qual o tráfego na Internet vem crescendo nos últimos anos se deve não apenas ao aumento no número de *hosts* nesta rede, mas também ao surgimento das aplicações multimídias (voz e vídeo) emergentes. Nesse sentido, como tais aplicações demandam rigorosos padrões de QoS, os quais implicam na necessidade de se utilizar a banda passante das redes com a máxima eficiência possível, novos mecanismos e princípios relacionados a esta abordagem têm sido desenvolvidos nesta última década.

Dentre as principais tecnologias desenvolvidas com o propósito de prover QoS na Internet nestes últimos anos tem-se: o IntServ, o DiffServ e o MPLS. A tecnologia de QoS ATM apesar de ter sido elaborada para funcionar de forma independente do IP (o qual caracteriza a Internet), uma vez que a mesma possui mecanismos de QoS nativos capazes de proporcionar garantia fim-a-fim de QoS, as mesmas não estão sendo exploradas neste sentido, sobretudo por carência de aplicações compatíveis.

A explicação para esta situação vem do fato de que o uso do ATM fim-a-fim implica na necessidade de se desenvolver aplicações compatíveis com esta tecnologia, e isso vem sendo inviabilizado diante da crescente aceitação que o IP tem apresentado desde a sua criação, o que resultou numa enorme infra-estrutura criada para este protocolo e que deve ser aproveitada. Por outro lado, o ATM está sendo empregado massivamente nas redes *backbones* de altas velocidades na Internet, onde as mesmas proporcionam alta confiabilidade além da

possibilidade de uso futuro da sua funcionalidade de QoS, o que, em princípio, já é uma contribuição à QoS na Internet.

Portanto, diante deste cenário e de acordo com o que foi apresentado neste trabalho, fica evidente que a interoperação de todas essas tecnologias de QoS se faz necessária com a finalidade de que se obtenha QoS fim-a-fim na Internet atual. E, isso implica em mapeamentos entre as mesmas, o que está sendo extensivamente pesquisado em todo o mundo.

Considerando-se as quatro tecnologias de QoS para a Internet mencionadas, as que se encontram em franco desenvolvimento, em função de serem muito recentes, são o MPLS e o DiffServ. Portanto, neste trabalho optou-se por estudar a arquitetura DiffServ, sobretudo em função do futuro promissor que se espera desta tecnologia.

Conforme apresentado neste trabalho a arquitetura DiffServ apresenta a capacidade de possibilitar tarifações diferenciadas por tipo de serviço oferecido na Internet atual. Tal capacidade é algo extremamente almejado pelos ISPs a fim de que estes possam oferecer QoS aos seus usuários e, ao mesmo tempo, receber as tarifas devidas em relação ao serviço oferecido, uma vez que QoS normalmente implica em maior demanda de recurso da rede envolvida.

Esta arquitetura possibilita ainda o uso das VLLs, de modo que as grandes corporações podem interligar as suas Intranets, relativas às suas várias filiais, utilizando a rede pública Internet. Este procedimento é muito atrativo a tais empresas, uma vez que o mesmo proporciona uma redução de custos significativa às mesmas.

Tratando-se do controle de congestionamento numa rede DiffServ, este é realizado por dois mecanismos distintos, um localizado no interior da rede e outro nos *hosts*. Dentro da rede o controle é realizado pelo mecanismo RIO (embora existam outras opções em estudo), o qual monitora a quantidade de pacotes enfileirados e, se necessário, os descarta de forma aleatória

com o propósito de evitar o fenômeno conhecido por "Sincronização Global", em que se tem o colapso na rede.

Nos *hosts*, o controle é realizado pelo mecanismo TCP, o qual limita a quantidade de pacotes que a fonte insere na rede. Portanto, o resultado final do controle de congestionamento nesse ambiente depende da interação entre os dois controles citados e, de acordo com os resultados obtidos, os parâmetros envolvidos nestes dois mecanismos podem determinar o comportamento da rede.

O mecanismo TCP apresenta atualmente várias opções de algoritmo possíveis. Dentre essas, o TCP Reno representa o mais popular algoritmo TCP usado atualmente na Internet. Contudo, este algoritmo apresenta a grande desvantagem de causar perdas na rede com o intuito medir o congestionamento na mesma. Por isso, novos algoritmos TCP têm sido propostos nos últimos e, dentre os quais, há o TCP Vegas.

O algoritmo TCP Vegas vem sendo abordado pela literatura como um protocolo promissor, visto que o mesmo pode proporcionar uma vazão muito superior a do TCP Reno e ainda retransmitir muito menos segmentos do que este último. Entretanto, problemas de injustiça, quanto à distribuição de banda passante, têm sido revelados por pesquisas recentes, tais como [34,35,37], nas quais se considerou apenas uma rede convencional (baseada no serviço BE).

Desse modo, o estudo desenvolvido neste trabalho abordou a avaliação de desempenho da rede DiffServ, em termos de justiça na distribuição de banda passante aos seus usuários, considerando os algoritmos TCP Reno e TCP Vegas. Nesse sentido, empregou-se diferentes conjuntos de valores para os parâmetros α e β do algoritmo TCP Vegas e para o RED associado aos pacotes não em conformidade do algoritmo RIO, chamado neste trabalho de RIO(out). Dois modelos de rede foram simulados, um com duas fontes e outro com dez

fontes de modo que fossem avaliadas as mais abrangentes situações. A influência do tráfego UDP sobre o tráfego TCP Vegas e TCP Reno foi também avaliada em algumas simulações.

De uma forma ampla, os resultados obtidos neste estudo revelaram que algumas das conclusões acerca do TCP Vegas obtidas em trabalhos anteriores como [34,35,37], nos quais se considerou apenas uma rede convencional, são válidas também para a arquitetura DiffServ. Por exemplo, o problema de injustiça do TCP Vegas e a capacidade que o algoritmo RED (Neste caso RIO(out)) tem de minimizar este problema, mediante ajuste adequado de seus parâmetros, são também observados para uma rede DiffServ. Outro exemplo é a influência que os parâmetros α e β do algoritmo TCP Vegas têm sobre a justiça da rede, conforme mostrado em [33,52], que também se comprovou aqui.

Outra revelação importante é o fato de que o comportamento da rede DiffServ pode variar de forma expressiva quando operando com apenas duas fontes daquele observado quando operando com múltiplas fontes (neste caso dez), considerando-se o uso do TCP Vegas. Os resultados mostraram que no caso em que se tem duas fontes sobre uma rede DiffServ é imprescindível que se utilize o mesmo valor para ambos os parâmetros do TCP Vegas ($\alpha=\beta$) e valores extremamente pequenos para os parâmetros de gatilho Min_{th} e Max_{th} do algoritmo RIO(out), a fim de que se consiga justiça adequada nessa rede.

Por outro lado, no caso em que se tem múltiplas fontes, é possível obter-se justiça mesmo com os ajustes nos quais se tem valores diferentes para os parâmetros do TCP Vegas ($\alpha\neq\beta$), e ainda neste caso os parâmetros citados para o RIO(out) não devem ser muito pequenos, sob pena de anularem muitas fontes (impondo 0 Kbps de banda passante às suas conexões).

Contudo, em ambos os casos (duas ou dez fontes), observou-se que não se deve empregar valores elevados para o ajuste do parâmetro $\alpha=\beta$, já que isso causa instabilidade na rede e também pode resultar em anulação de muitas fontes.

No que tange à avaliação da interoperação dos algoritmos TCP Reno e TCP Vegas, os resultados mostraram que nos casos em que se tem apenas duas fontes é possível se conseguir justiça adequada seguindo os ajustes mencionados anteriormente ($\alpha=\beta$ e ajustes estritamente pequenos para RIO(out)). Para múltiplas fontes, porém, o mesmo não ocorre e o que se observa é que em geral o TCP Reno supera o TCP Vegas.

Tratando-se do UDP, os resultados mostraram que o ajuste dos parâmetros estudados neste trabalho não é suficiente para minimizar a agressividade do mesmo e, por isso, mecanismos de controle de admissão de fluxo se fazem necessários nesse caso, independentemente de se estar utilizando o TCP Reno ou o TCP Vegas. E, com relação ao TCP Reno, os parâmetros do algoritmo RIO não mostraram influência expressiva sobre o funcionamento da rede DiffServ quando operando com este algoritmo.

Portanto, analisando-se os resultados obtidos como um todo conclui-se que o mecanismo TCP Vegas isoladamente pode oferecer uma justiça equivalente a oferecida com o TCP Reno, desde que os seus parâmetros α e β, juntamente com os mencionados parâmetros do RIO, sejam adequadamente ajustados. No entanto, no caso de se ter a interoperação destes dois mecanismos o TCP Vegas é, em geral, penalizado pela agressividade do TCP Reno.

Assim sendo, embora o TCP Vegas ofereça melhor aproveitamento da banda passante da rede do que o TCP Reno [34,36,37,43], uma possível migração de forma gradativa deste último para o primeiro seria comprometida pelo mencionado problema de injustiça na interoperação entre os mesmos. Sugere-se como futuros trabalhos, no final da seção 6.4 deste estudo, a análise de viabilidade de duas abordagens (mecanismos ECN e REM) na solução deste problema.

Finalizando, cabe mencionar que o objetivo principal proposto inicialmente por este trabalho foi concretizado, o qual se referia à análise da viabilidade de uso do TCP Vegas num ambiente DiffServ, comparativamente ao TCP Reno. Não menos importante é o fato de que,

apesar da sua relevância, este assunto não foi abordado em nenhum estudo anterior, dentre o bibliografia pesquisada. Sugere-se como futuros trabalhos, além daqueles citados no parágrafo anterior, o estudo de outros importantes algoritmos TCP, tais como: New Reno, FACK, SACK, etc, num ambiente DiffServ, visto que só assim poder-se-á determinar qual é a melhor opção a ser adotada por esta arquitetura nas implementações futuras.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] The ATM Forum Technical Committe, Traffic Management Specification, version 4.1 AF-TM-0121.000, March 1999, (available at http://atmforum.com/atmforum/specs/approved.html).

[2] FAHMY, S. et al., Quality of Service for Internet Traffic over ATM Service Categories, Computer Communications, vol. 22, Issue 14, pp. 1307-1320, September 1999.

[3] FERGUSON, P., HUSTON, G., Quality of Service in the Internet: Fact, Fiction, or Compromise?, (available at http://www.employees.org/~ferguson/inet-qos.htm).

[4] CISCO SYSTEMS., Quality of Service (QoS) Networking, (available at http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm).

[5] XIAO, X., LIONEL, M., Internet QoS: A Big Picture, IEEE Network Magazine, pp. 8-18, March/April 1999.

[6] VANDALORE, B. et al., QoS and Multipoint Support for Multimedia Applications over the ATM ABR Service, IEEE Communications Magazine, pp. 53-57, January 1999.

[7] EICHLER, G. et al., Implementing Integrated and Differentiated Services for the Internet with ATM Networks: A Practical Approach, IEEE Communications Magazine, pp. 132-140, January 2000.

[8] FERGUSON, P., HUSTON, G., Quality of Service, Delivering QoS on the Internet and in Corporate Networks, John Wiley & Sons, Inc. USA.226p. 1998.

[9] COCCA, R., SALSAMO, S., LISTANTI, M., Internet Integrated Service over ATM: A solution for Shortcut QoS Virtual Channels, IEEE Communications Magazine, pp. 98-104, December 1999.

[10] FERRARI, T., QoS Support for Integrated Networks, PhD thesis, Faculty of Engineering, University of Bologna, Italy, 1998. 114p.

[11] BRADEN, R. et al., Resouce ReSerVation Protocol (RSVP), – Version 1 – Functional Specification, IETF RFC 2205, September 1997.

[12] CRAWLEY, E. et al., A framework for Integrated Services and RSVP over ATM, IETF RFC 2382, August 1998.

[13] GARRET, M. and BORDEN, M., Interoperation of Controlled-Load and Guaranteed Service with ATM, IETF RFC 2381, August 1998.

[14] NICHOLS, K. et al., Definition of the Differentiated Services Field (DS Field) in the IPv4 and Ipv6 Headers, IETF RFC 2474, December 1998.

[15] BLAKE, S. et al., An Architecture for Differentiated Services, IETF RFC 2475, December 1998.

[16] BERGER, L., RSVP over ATM Implementation Guidelines, IETF RFC 2379, August 1998.

[17] CALLON, R., A Framework for Multiprotocol Label Switching, IETF Draft draft-ietf-mpls-framework-05.txt, September 1999.

[18] BERNET, Y. et al., Integrated Services Operation Over Diffserv Networks, IETF Draft draft-ietf-issll-intserv-rsvp-02.txt, June 1999.

[19] CRAWLEY, E., A Framework for QoS-based Routing in the Internet, IETF RFC 2386, August 1998.

[20] HEINANEN, J., Assured Forwarding PHB Group, IETF RFC 2597, June 1999.

[21] JACOBSON, V., An Expedited Forwarding PHB, IETF RFC 2598, June 1999.

[22] AWDUCHE, D. et al., Requirements for Traffic Engineering over MPLS, IETF DRAFT draft- ietf-mpls-traffiv-eng.00.txt, Octuber 1998.

[23] NICHOLS, K., A Two-bit Differentiated Services Architecture for the Internet, IETF RFC 2638, July 1999.

[24] CLARK, D., FANG, W., Explicit Allocation of Best Effort Paket Delivery Service, (available at http://diffserv.lcs.mit.edu/Papers/exp-alloc-ddc-wf.pdf).

[25] WANG, Z., User-Share Differentiatin – Scalable Service Allocation for the Internet, Internet Draft, November 1997.

[26] BASU, A., WANG, Z., A Comparative Study of Shemes for Differentiated Services, (available at http://www.iam.unibe.ch/~baumgart/diffserv.html).

[27] NS Simulator. University of Californy at Berkeley, CA. (available at http://www.nrg. ee.lbl.gov/ns).

[28] FLOYD, S., JACOBSON, V. Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, 1(4):397--413, July 1993.

[29] JACOBSON, V., Differentiated Services Architecture, Talk in the Int-Serv WG at the Munich IETF, August 1997.

[30] YEON, I., REDDY, A. L. N., Realizing throughput guarantees in a differentiated services network, (available at http://computer.org/proceedings/icmcs /0253/ volume %202/02530372abs.htm).

[31] BAUMGARTNER, F., BRAUN, T., SIEBEL, C., Fairness ofAssured Service, (available at http://www.iam.unibe.ch/~rvs/publications/).

[32] TANENBAUM, A. S., Computer Networks, Prentice Hall PTR, third edition, 1996.

[33] HASEGWA, G., MURATA, M., MIYAHARA, H., Fairness and

Stability of Congestion Control Mecahnisms of TCP, in Proceedings of INFOCOM'99, pp. 1329-1336, March 1999.

[34] MO, J. et al., Analysis and Comparation of TCP Reno and Vegas, in Proceedings of INFOCOM'99, pp 1556-1563, March 1999.

[35] PETERSON, L., WANG, L., Understanding TCP Vegas: Theory and Practice, (available at http://www.cs.princeton.edu/nsg/publications.html).

[36] BRAKMO, L. S., O'MALLEY, S. W., PETERSON, L., TCP Vegas: New Techniques for Congestion Detection and Avoidance, in Proceedings of ACM SIGCOMM'94, pp. 24-35, London, Octber 1994.

[37] HENGARTEHER, U., BOLLIGER, J. and GROSS, Th., TCP Vegas Revisited, (available at http://www.cs.inf.ethz.ch/~bolliger/papers/infocom00.txt.).

[38] STEVENS, W. Richard, TCP/IP illustrated: The Protocols, vol. I Addison - Wesley, Reading, Massachusetts, 1994.

[39] JACOBSON, V., Congestion avoidance and control, ACM SIGCOMM 88, pp. 273-288, 1988.

[40] JACOBSON, V., Modified TCP Congestion avoidance Algorithm, mailling list, end2end-interest, April 1990.

[41] AHN, J. S. et al, Experience with TCP Vegas: Emulation and experiment, Proc. SIGCOMM'95 Symp., August 1995.

[42] SEDDIGH, N., NANDY, B., PIEDA, P., Bandwidth assurance Issues for TCP flows in a Differentiated Services Network, in Globecom'99, pp. 1792-1798, Dec. 1999.

[43] BRAKMO, L. S. and PETERSON, L. L., TCP Vegas: End to End Congestion Avoidance on a Global Internet, IEEE JSAC, Vol. 13, pp. 1465-140, 1995.

[44] WENJIA F., TCP mechanisms for Diff-Serv Architecture, (available at http://www.cs.princeton.edu/~wfang/papers.html.).

[45] DOBREFF, A., Comparison of Simulation and Real Functionality for the Mapping of Differentiated Services to ATM, Diploma Thesis, Faculty of philosophy an science of nature, university of Berne, Switzerland, November 119p.(available at http://www.iam.unibe.ch/~rvs/publications/ dobreff.Diplom.pdf)

[46] FALL, K., FLOYD, S., Simulation-based Comparisons of Tahoe, Reno, and SACK TCP, computer communication Review, July 1996.

[47] IP QoS over ATM, (available at http://www.netlab.ohio-state.edu/~jain/cis788-99/ip_qos_atm/index.html).

[48] CRAWLEY, E. et al., A framework for Integrated Services and RSVP over ATM, IETF RFC 2382, August 1998.

[49] RAGHAVENDRA, A. M., KINICKI, R. E., A Simulation Performance Study of TCP Vegas and Random Early Detection, in Proceedings of 18th Int. Performance Computing communications Conference, pp. 169-176, 1999.

[50] ZHANG, Y., YAN, E., DAO, S. K., A Measurement of TCP over Long-Delay Network, In Proceedings of 6th Intl. Conf. On Telecommunications Systems, pp. 498-504, March 1998.

[51] KARN, P., PARTRIDGE, C., Improving Round-Trip Time Estimates in Reliable Transport Protocols, (available at http://www.acm.org/sigcomm/ccr/archive/1995/jan95/ccr-9501-partridge87.pdf).

[52] BOUTREMANS, C., BOUDEC, J., A Note on the fairness of TCP Vegas, in Proceedings of International Zurich Seminar on Broadband Communications, pp. 163-170, Zurich, Switzerland, February 2000.

[53] ROGERS, G. et al., Matching Differentiated Services PHBs to ATM Service Categories – the AF to VBR Case, (available at http://www-networks.tip.csiro.au/~minhazu/dif_serv.html.).

[54] HEINANEN, J. et. al., A Three Color Marker, IETF DRAFT draft-heinanen-idffserv-tcm-01.txt, Febuary 1999.

[55] HEINANEN, J. et al., A Single Rate Three Color Marker, IETF RFC 2697, September 1999.

[56] HEINANEN, J. et. al., A Two Rate Three Color Marker, IETF RFC 2698, September 1999.

[57] QoS Forum, The Need for QoS, White paper, 1999, (available at http://www.qosforum.com).

[58] STEVENS, W., TCP Slow Start, Congestion Avoidance, Fast Retransmite, and Fast Recovery Algorithm, IETF RFC 2001, January 1997.

[59] ITU-T Recommendation I.371, Traffic Control and congestion in B-ISDN, ITU-T Study Group 13, Geneva, August 1996.

Trabalhos Publicados pelo Autor:

[60] OLIVEIRA, R., GUARDIEIRO, P. R., Qualidade de Serviço na Internet: Um Estudo de Tecnologias e Arquiteturas, Anais do III Seminário de Estudos em Engenharia Elétrica, III SEEE, Universidade Federal de Uberlândia, 23 a 26 de Outubro de 2000.

[61] OLIVEIRA, R., GUARDIEIRO, P. R., A Comparative Study of TCP Reno and TCP Vegas in a Differentiated Services Network, accepted for publication in CONFTELE 2001, Figueira da Foz, Portugal, April 23 - 24, 2001.

[62] OLIVEIRA, R., GUARDIEIRO, P. R., Performance evaluation of TCP Vegas in a Differentiated Services Network, submetido para publicação na Revista Ciência e Engenharia, Universidade Federal de Uberlândia, 2001.

[63] OLIVEIRA, R., GUARDIEIRO, P. R., An valuation of TCP Vegas in a Differentiated Services Network in terms of Fairness, submitted to the 19th Brazilian Symposium on Computer, Santa Catarina, Brazil, 2001.