

JOÃO GABRIEL SAPUCAHY CHISTE

Estudo comparativo entre roteadores Linux e
FreeBSD para diferenciação de
tráfego

Monografia apresentada como requisito
parcial à obtenção do grau de Bacharel,
Curso de Ciência da Computação,
Universidade Federal de Santa Catarina,
Orientador: Prof. Roberto Willrich

Florianópolis
2005

JOÃO GABRIEL SAPUCAHY CHISTE

Estudo comparativo entre roteadores Linux e FreeBSD para diferenciação de tráfego

Florianópolis
2005

Banca Examinadora:

Roberto Willrich,
Orientador, UFSC/INE

Mário Dantas,
UFSC/INE

João Bosco M. Sobral
UFSC/INE

*“Dedico este trabalho a minha
falecida avó, que sempre acreditou em mim e
sempre me ajudou quando eu precisei, que
descanse em paz”*

AGRADECIMENTOS

À todos eles que acreditaram que eu eu conseguiria terminar o curso e este trabalho.

A minha família que me deu todo o suporte emocional, moral e financeiro para que eu pudesse permanecer aqui.

À todos os meus amigos que me acompanharam nessa jornada, sem a ajuda de vocês eu não conseguiria chegar ao fim.

Aos professores que me passaram o conhecimento e mostraram o caminho para que eu pudesse realizar este trabalho.

Por último e não menos importante ao Christopher Viana, mestrando do LSD que me ajudou muito nesse trabalho, principalmente no manejo do FreeBSD.

RESUMO

O modelo de controle de tráfego na Internet atualmente é baseado no melhor esforço, isso quer dizer que os pacotes são distribuídos na ordem em que chegam. Este trabalho faz um estudo sobre o uso de Qualidade de Serviço como ferramenta para melhorar o controle de tráfego através da diferenciação de serviços, priorizando as aplicações que demandam mais recursos da rede, como VoIP (voz sobre IP) e videoconferência, que são aplicações em tempo real, de forma a garantir um nível mínimo de qualidade pré-estabelecido no SLA (Acordo de nível de Serviço). Neste trabalho foi montada uma rede com as mesmas características da rede local de uma empresa, e utilizando o sistema operacional Linux como roteador e duas máquinas rodando FreeBSD para gerar e receber o tráfego da rede. Foram gerados os tráfegos TCP (simulando mail, web, ftp) e UDP (simulando voip) e através da ferramenta tc, de controle de tráfego do Linux, foram usadas políticas de fila de QoS CBQ para priorizar o tráfego UDP. Os resultados puderam ser avaliados através de algumas métricas de desempenho (delay, jitter, vazão e perda de pacotes) e comparados com os resultados obtidos na mesma rede e com a mesma carga no trabalho feito usando um roteador FreeBSD por Alexandre Bunn. Com os resultados obtidos pode-se concluir que o Linux é uma ferramenta adequada para fazer roteamento com diferenciação de tráfego.

PALAVRAS CHAVE : QoS, Linux, roteamento, FreeBSD

ABSTRACT

The traffic control in the internet nowadays its based in the best effort model, this means that the packets are sent as they came. This work does a study about the use of Quality of Service as a tool to improve the traffic control through the differentiated services prioritizing the applications that demands more resources of the network, like VoIP (voice over IP) and videoconference, that are realtime applications, to insure a minimal level of quality established in SLA (Service Level Agreement). In this work was made a network with the same characteristics of a business local area network, and using the Linux operational system as a router e two other machines running FreeBSD to generate and receive the network traffic. Was generated the TCP (simulating mail, web, ftp) and UDP (simulating voip) traffic and through of the tool tc, traffic controler of Linux, that used queue disciplines of QoS CBQ to prior UDP traffic. The results could be avaliated through some performance meters (delay, jitter, throughput and packet lost) and compared with the results got in the same net and the same workload using a FreeBSD router, done by Alexandre Bunn. With the results got, we can conclude that Linux is a adequated tool to routing with differentiated services.

KEY WORDS : QoS, Linux, routing, FreeBSD

ÍNDICE

LISTA DE FIGURAS	6
LISTA DE TABELAS	7
INTRODUÇÃO	8
1. CONSIDERAÇÕES INICIAIS	8
1.1 Objetivos do trabalho	10
1.2 Estrutura do documento	10
2. ARQUITETURA TCP/IP	12
2.1 Características	12
2.2 Camadas	13
2.2.1 Algoritmos de Roteamento	16
2.2.2 Controle de Congestionamento	17
3. QUALIDADE DE SERVIÇO (QOS)	19
3.1 Princípios	19
3.2 QoS como Mecanismo Gerencial	20
3.2.1. Controle de Tráfego	21
3.3 QoS -Parâmetros	21
3.3.1 Quais Aplicações Necessitam de QoS?	22
3.3.2 Vazão	23
3.3.3 Latência (Atraso)	24
3.3.4 Jitter	27
3.3.5 Perdas	28
3.3.6 Disponibilidade	29
3.4 O Modelo DiffServ	29
3.5 Classificação de tráfego e armazenamento	30
4. QOS NO LINUX	34
4. QOS NO LINUX	35
4.1 Configuração	35
4.2 Políticas de Enfileiramento	37
4.3 Classes	39

4.4 Filtros	47
5. COMPARAÇÃO DOS TESTES LINUX X FREEBSD	49
5.1 Definição da Rede	49
5.2 Resultado dos testes da rede não saturada.....	50
5.2.1 Análise do tempo médio de espera fim-a-fim (delay)	51
5.2.2 Análise do efeito jitter médio	52
5.2.3 Análise da vazão média	54
5.2.4 Análise da taxa de perdas	55
5.3 Análise dos dados com a rede saturada	55
5.3.1 Análise do tempo médio de espera fim-a-fim (delay)	56
5.3.2 Análise do jitter médio	57
5.3.3 Análise da vazão média	59
5.3.4 Análise da taxa de perdas	60
6. CONCLUSÃO E TRABALHOS FUTUROS	62
7. REFERÊNCIAS.....	63
ANEXO 1 - ROTEIRO DE INSTALAÇÃO E CONFIGURAÇÃO DO RUDE & CRUDE.....	67
ANEXO 2 – REGRAS USADAS PARA SIMULAR O TRÁFEGO TCP CONSTANTE	68
ANEXO 3 – SCRIPT PARA A DECODIFICAÇÃO DO TRÁFEGO RECEBIDO ...	69
ANEXO 5 – SINTAXE DOS PROGRAMAS UTILIZADOS	70
ANEXO 6 - ARTIGO.....	71

LISTA DE FIGURAS

FIG. 2-1 - ARQUITETURA TCP/IP	12
FIG. 2.2. DATAGRAMA IP	15
FIG 2.3. PADRÃO DE COMPORTAMENTO NA OCORRÊNCIA DE CONGESTIONAMENTO.	18
FIGURA 3.1 - EFEITO DO JITTER PARA AS APLICAÇÕES	28
FIG 3.2 TCB – BLOCO CONDICIONADOR DE TRÁFEGO DIFFSERV	32
FIGURA 4.1 CONTROLE DE TRÁFEGO DO LINUX.....	36
FIGURA 4.2 FERRAMENTA PARA O DESENVOLVIMENTO DE INTSERV E DIFFSERV.....	37
FIGURA 4.3 UMA FILA EXEMPLO	39
FIGURA 4.4: ESTRUTURA DE FILTROS	48
FIG. 5.2 DELAY MÉDIO REDE NÃO SATURADA LINUX.....	51
FIG. 5.3 DELAY MÉDIO REDE NÃO SATURADA FREEBSD.....	51
FIG. 5.4 JITTER MÉDIO REDE NÃO SATURADA LINUX	52
FIG. 5.5 JITTER MÉDIO REDE NÃO SATURADA FREEBSD	53
FIG. 5.6 VAZÃO MÉDIA REDE NÃO SATURADA LINUX.....	54
FIG. 5.7 VAZÃO MÉDIA REDE NÃO SATURADA FREEBSD	55
FIG. 5.8 DELAY MÉDIO REDE SATURADA LINUX	56
FIG. 5.9 DELAY MÉDIO REDE SATURADA FREEBSD	57
FIG. 5.10 JITTER MÉDIO REDE SATURADA LINUX	58
FIG. 5.11 JITTER MÉDIO REDE SATURADA FREEBSD.....	58
FIG. 5.12 VAZÃO MÉDIA REDE SATURADA LINUX	59
FIG. 5.13 VAZÃO MÉDIA REDE SATURADA FREEBSD.....	59
FIG. 5.14 PERDA DE PACOTES REDE SATURADA LINUX	60
FIG. 5.15 PERDA DE PACOTES REDE SATURADA FREEBSD	60

LISTA DE TABELAS

TAB. 2.1 VETOR_DISTÂNCIA X ESTADO_DE_ENLACE	17
TABELA 3.1 - VAZÃO TÍPICA DE APLICAÇÕES EM REDE.....	24
TABELA 3.2 - ATRASOS DE PROPAGAÇÃO - FIBRAS ÓPTICAS - EXEMPLOS.....	25
TABELA 5.1 ESTATÍSTICAS DELAY MÉDIO REDE NÃO SATURADA	52
TABELA 5.2 JITTER MÉDIO, ESTATÍSTICAS.....	54
TABELA 5.3 VAZÃO MÉDIA, ESTATÍSTICAS.....	55
TABELA 5.4 ESTATÍSTICAS DELAY MÉDIO REDE SATURADA.....	57
TABELA 5.5 ESTATÍSTICAS JITTER MÉDIO REDE SATURADA.....	59
TABELA 5.6 VAZÃO MÉDIA REDE SATURADA, ESTATÍSTICAS.....	60
TABELA 5.7 PERDA DE PACOTE REDE SATURADA, ESTATÍSTICAS.....	61

INTRODUÇÃO

1. Considerações iniciais

Atualmente as redes desempenham um papel fundamental nas empresas, desde as pequenas às grandes corporações, não importando se a empresa tem a informática como foco principal ou apenas a utiliza como ferramenta de trabalho. E nesse cenário, aplicações de missão crítica, tais como as de Planejamento de Recursos Empresariais – ERP (Enterprise Resource Planning), compartilham largura de banda com o tráfego convencional (transferência de arquivos, acesso a páginas WWW) e outras aplicações. As redes, entretanto, convergem para uma infra-estrutura única compartilhada, capaz de suportar outros tipos de dados como áudio e vídeo e, para um melhor aproveitamento, tipos diferentes de tráfego devem ser tratados de modo diferenciado. Algumas aplicações, como as de áudio, têm exigências rígidas quanto ao atraso fim-a-fim, mas podem tolerar perdas mínimas de pacotes, enquanto outras como transferência de arquivos são sensíveis a este último aspecto, mas as exigências quanto ao atraso são pouco severas[Mel01].

As aplicações de missão crítica dão suporte às principais operações de empresas e não podem tolerar atrasos causados por aplicações multimídia. Estas possuem altos requisitos de largura de banda e, potencialmente, podem absorver todos os recursos de rede disponíveis. Em uma instituição financeira, por exemplo, aplicações de missão crítica são aquelas ligadas diretamente ao negócio, como o suporte às operações de saque, emissão de extrato, consulta de saldo e outras desta natureza. Já a empresa que decide implantar uma aplicação de voz sobre IP (VoIP) em sua rede corporativa necessita priorizar o tráfego gerado por essa aplicação em relação as demais, sem entretanto afetá-las significativamente.[Mel01]

Normalmente a Internet trabalha com a filosofia do melhor esforço: cada usuário compartilha largura de banda com outros e, portanto, a transmissão de seus dados concorre com as transmissões dos demais usuários. Os dados empacotados são encaminhados da melhor forma possível, conforme as rotas e banda disponíveis. Quando há congestionamento, os pacotes são descartados sem distinção. Não há

garantia de que o serviço será realizado com sucesso. Entretanto, aplicações como voz sobre IP e videoconferência necessitam de tais garantias.

Com a implantação de qualidade de serviço (*Quality of Service* – QoS), é possível oferecer maior garantia e segurança para aplicações avançadas, uma vez que o tráfego destas aplicações passa a ter prioridade em relação a aplicações tradicionais.[Rnp].

Com uso de QoS os pacotes são marcados para distinguir os tipos de serviços e os roteadores são configurados para criar filas distintas para cada aplicação, de acordo com as prioridades das mesmas. Assim, uma faixa da largura de banda, dentro do canal de comunicação, é reservada para que, no caso de congestionamento, determinados tipos de fluxos de dados ou aplicações tenham prioridade na entrega.[Rnp]

Pesquisas feitas pelo ATM Fórum[Atm] e IETF[Iet] apresentam diversas soluções para prover QoS em redes de computadores: o padrão ATM (Asynchronous Transfer Mode), o protocolo de reserva de recursos, RSVP (Resource ReServation Protocol) [Bra95], principal componente da arquitetura IntServ de serviços integrados [DiffServ_Charter] e a arquitetura DiffServ de serviços diferenciados[IntServ_Charter] são soluções interoperáveis e complementares que buscam resolver este problema.

IntServ é baseado em reserva de recursos, enquanto DiffServ é uma proposta na qual os pacotes são marcados de acordo com classes de serviços pré-determinadas.

No trabalho a ser desenvolvido será utilizada a técnica DiffSev em que os pacotes são encaminhados de forma diferenciada, através do agrupamento destes em categorias de tráfego chamadas classes. Uma classe pode conter um único fluxo, que seria um conjunto de pacotes com o mesmo endereço fonte e destino. Ou a agregação de múltiplas instâncias de um fluxo. Tratamento diferenciado nos pacotes pode ser empregado para criar serviços. Um serviço está associado às necessidades das aplicações como largura de banda, atraso, variação do atraso ou jitter e taxa de perda de pacotes. Conceitualmente, a largura de banda define a capacidade de transmissão de dados de um canal de comunicação, enquanto o atraso é o tempo decorrido entre o envio de uma mensagem e sua recepção no nó destino. Em redes de pacotes, a variação do atraso é uma distorção nos tempos originais de transmissão, enquanto que a taxa de

perda de pacotes representa o percentual de pacotes que foram transmitidos, mas não chegaram ao destino em um determinado período de tempo.

O suporte de QoS requer medição de desempenho para aferir se os níveis de qualidade especificados foram alcançados. Quando um provedor Internet ou uma corporação fornece serviços baseados em QoS para seus usuários, diferentes níveis de medições podem ser realizados:

- na aplicação para adaptação aos níveis de QoS esperados;
- pelos usuários para verificar o serviço fornecido pela rede; e
- pelo provedor para monitorar e validar os serviços.

1.1 Objetivos do trabalho

Avaliar e estudar o sistema operacional Linux como plataforma de gerenciamento de qualidade de serviço (QoS), em especial suas potencialidades de diferenciação de tráfego.

Realizar um testbed composto de duas máquinas FreeBSD e um roteador Linux. Sendo que o roteador estará configurado para realizar diferenciação de tráfego através de QoS. Comparar os resultados dos testes com os resultados obtidos utilizando o FreeBSD como roteador.

1.2 Estrutura do documento

Os 7 capítulos deste trabalho, de forma resumida, apresentam os seguintes conteúdos:

Capítulo 1: Introdução – Neste capítulo são apresentados os objetivos do trabalho e a sua justificativa em termos de necessidades de QoS. Ainda neste capítulo apresenta-se a forma de organização do trabalho.

Capítulo 2: Arquitetura TCP/IP – Neste capítulo é apresentada a arquitetura TCP/IP, descrevendo cada uma de suas 5 camadas com maior ênfase na camada de rede que é a camada onde atuam os roteadores.

Capítulo 3: Qualidade de Serviço (QoS) – Neste capítulo é descrita a definição de QoS, histórico, e a aplicação de QoS no trabalho.

Capítulo 4: Recursos de QoS no Linux – Neste capítulo é apresentada uma visão geral de QoS no Linux e mais especificamente os recursos utilizados no trabalho.

Capítulo 5: Comparação dos testes Linux x FreeBSD – Neste capítulo são apresentados os resultados dos experimentos realizados, assim como a avaliação destes resultados.

Capítulo 6: Conclusão e trabalhos futuros - Aqui é feita a conclusão do trabalho e dos testes realizados. Sugestões para trabalhos futuros são apresentadas.

Capítulo 7: Referências - Neste capítulo é apresentada a bibliografia e as referências bibliográficas utilizadas.

2. ARQUITETURA TCP/IP

A arquitetura básica do TCP/IP é mostrada na figura abaixo, que fazendo relação com o modelo de referência OSI correspondem as camadas da Rede, Transporte e Aplicações.

Como pode-se observar, na camada de rede TCP/IP tem-se o protocolo IP(Internet Protocol) ; na camada de transporte estão dois protocolos, um que oferece serviços sem conexão, que é o protocolo UDP (User Datagram Protocol) e um outro que oferece serviços orientados a conexão, protocolo TCP(Transport Control Protocol). Na camada de aplicações o TCP/IP tem uma variedade de protocolos de aplicação, como SMTP, TELNET, FTP e NSF entre outros.

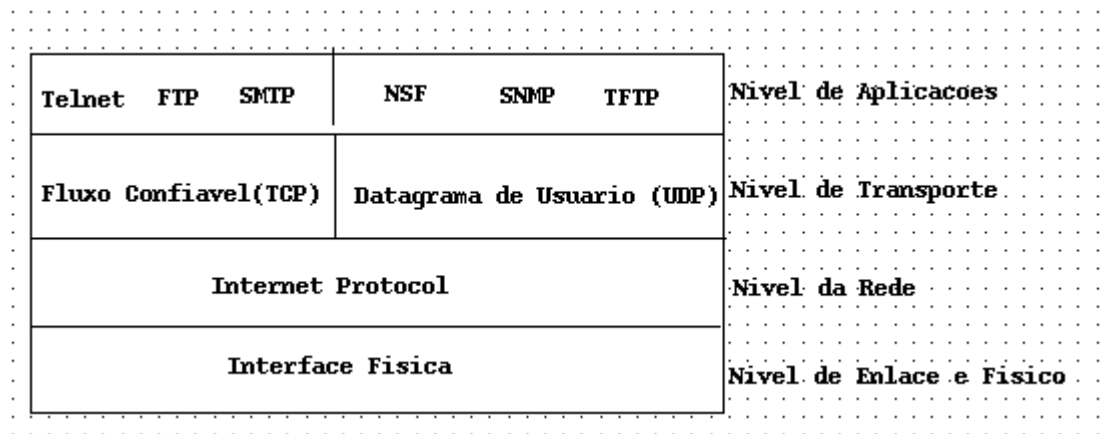


FIG. 2-1 - ARQUITETURA TCP/IP [ART]

2.1 Características

- Padrão de protocolos aberto, não associado a nenhum tipo específico de plataforma de hardware, sistema operacional. ou hardware específico para acesso ao meio físico de transmissão (TCP/IP funciona sobre Ethernet, Token-ring, linha discada, X.25, e qualquer outro tipo de meio de transmissão).
- Esquema de endereçamento comum que permite a identificação única de um elemento da rede (na rede local, ou no planeta).

- Protocolos de alto nível padronizados para disponibilização universal e consistente de serviços aos usuários.
- Documentação ampla acessível na própria rede sob a forma de “Request for Comments” – RFC’s que não sofrem do rigor imposto aos relatórios técnicos formais. As RFC’s contêm as últimas versões das especificações de todos os protocolos TCP/IP padrões.
- Interconexão cooperativa de redes, suportando serviços de comunicação universal (usada em uma rede local ou em uma rede [inter] planetária).
- Utilização de tecnologia adequada às necessidades locais em cada rede.
- Independência de hardware e sistemas operacionais.
- Interconexão de redes se dá por meio de roteadores.

2.2 Camadas

- **Aplicação**

É formada pelos protocolos utilizados pelas diversas aplicações do modelo TCP/IP. Esta camada não possui um padrão comum. O padrão é estabelecido por cada aplicação. Isto é, o FTP possui seu próprio protocolo, assim como o TELNET, SMTP, POP3, DNS, etc...

- **Camada de Transporte**

Camada fim-a-fim, isto é, uma entidade desta camada só se comunica com a sua entidade-par do host destinatário. É nesta camada que se faz o controle da conversação entre as aplicações intercomunicadas da rede. Dois protocolos aqui são usados: o TCP e o UDP. O TCP é orientado à conexão e o UDP não. O acesso das aplicações à camada de transporte é feito através de portas que recebem um número inteiro para cada tipo de aplicação.

Suas funções compreendem:

- Montagem de segmentos para a camada de rede a partir de mensagens vindas da camada de aplicação (fragmentação).

- Montagem de mensagens para a camada de aplicação a partir de segmentos vindos da camada de rede (desfragmentação).
- Transmissão de dados sem conexão e não confiável (protocolo UDP) ou com conexão e confiável (cria circuito virtual através do protocolo TCP).
- Retransmissão temporizada, seqüenciamento de segmentos (máx. de 64 KB cada), controle de fluxo (através de janela deslizante);
- Armazenamento temporário (“buffering”) na recepção e transmissão;
- Identificação de processos origem e destino.

- **Camada de Rede**

Essa camada é a primeira normatizada do modelo. Também conhecida como camada Internet, é responsável pelo endereçamento, roteamento e controle de envio e recepção. Ela não é orientada à conexão, se comunica através de datagramas IP, sobre os quais é baseada a rede TCP/IP, são os “tijolos” de construção da internet. Em geral é implementada através de gerentes (*drivers*) de dispositivos. É a camada onde atuam os roteadores.

Suas funções compreendem:

- Montagem de datagramas IP a partir de segmentos recebidos da camada de transporte (fragmentação de segmentos).
- Montagem de segmentos para a camada de transporte a partir de datagramas IP recebidos da camada de acesso ao meio físico (desfragmentação de segmentos).
- Transmissão de dados sem conexão e não confiável (modo datagrama).
- Identificação de máquina de origem e máquina de destino.
- Encaminhamento (roteamento) de datagramas IP através da rede lógica.
- Integração de diversas redes físicas formando uma única rede.

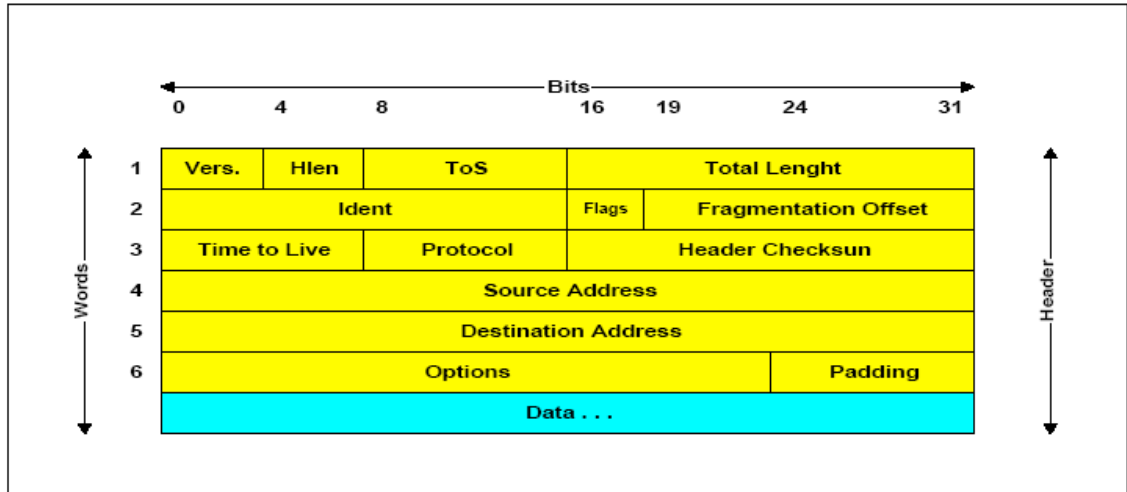


FIG. 2.2. DATAGRAMA IP [GUE02]

- Version: versão do IP (atualmente 4).
- Hlen: tamanho do cabeçalho do datagrama.
- ToS: tipo do serviço (precedência normal/controle, baixo retardo, alta eficiência, alta confiabilidade), sem garantia de cumprimento.
- Total Length: tamanho total do datagrama (máximo de 64 Kbytes).
- Ident: identificação do datagrama (único para cada datagrama).
- Flags: controle de fragmentação (habilita/desabilita fragmentação, marca de fim do datagrama original).
- Fragmentation Offset: deslocamento do fragmento.
- TTL: tempo de vida do datagrama (inicia em N, decrementa a cada passagem por um roteador; chegando em 0 (zero), o datagrama é descartado e é gerada uma mensagem de erro).
- Protocol: protocolo de nível superior (TCP, UDP).
- Header checksum verificação de integridade do datagrama.
- Source Address: endereço origem (máquina emissora).
- Dest Address: endereço destino (máquina receptora).
- Options: opções de teste e depuração.
- Record Route Option: datagrama guarda endereços de roteadores intermediários por onde passou.
- Source Route Option: sistema origem define rota que um datagrama deve seguir.

- Timestamp Option: datagrama guarda informação sobre data e hora que chegou aos roteadores intermediários.
- Padding: preenchimento.
- Data: dados transportados.

- **Camada física**

Camada de abstração de hardware, tem como principal função a interface do modelo TCP/IP com os diversos tipos de redes (X.25, ATM, FDDI, Ethernet, Token Ring, Frame Relay, PPP e SLIP). Por causa da grande variedade de tecnologias de rede, ela não é normatizada pelo modelo, o que provê a possibilidade de interconexão e interoperação de redes heterogêneas

Suas funções compreendem:

- Recebe datagramas IP da camada de rede e os transmite sobre a tecnologia de rede física disponível.
- Faz o encapsulamento de datagramas IP em quadros da rede (eventualmente com fragmentação).
- Faz o mapeamento de endereços lógicos Internet em endereços físicos de equipamentos na rede.

2.2.1 Algoritmos de Roteamento

Roteamento é o mecanismo pelo qual se escolhe o caminho (canal de comunicação) que um pacote deve seguir para atingir seu destino. Em redes sem conexão (datagrama), cada datagrama tem de carregar seu endereço destino e a decisão de roteamento é tomada em cada nó da rede, para cada datagrama que chega.. Em redes com conexão (circuito virtual) os pacotes não precisam carregar o endereço destino (só a identificação da conexão) e a decisão de roteamento é tomada no estabelecimento da conexão, após a qual todo pacote segue sempre o mesmo caminho (roteamento por sessão).

Um algoritmo de roteamento deve oferecer:

- Correticidade (funciona).
- Simplicidade (é fácil de entender/usar).

- Robustez (suporta falhas na rede).
- Estabilidade (fixa caminhos rapidamente).
- Equanimidade (distribui carga de modo justo).
- Optimalidade (proporciona melhor caminho sempre).

Um algoritmo de roteamento pode ser:

- Não adaptativo, quando as decisões de roteamento são definidas antecipadamente (pelo gerente da rede, p. ex.) e colocadas nos roteadores quando estes são ligados - roteamento estático.
- Adaptativo, quando as decisões de roteamento são [re]definidas continuamente, de acordo com a estrutura da rede (topologia, carga, etc.) - roteamento dinâmico.

Nesse último caso, o algoritmo de roteamento pode ser do tipo Vetor Distância ou Estado de Enlace, cujas características são resumidas na tabela a seguir.

VETOR DISTÂNCIA	ESTADO DE ENLACE
Em cada nó da rede, o algoritmo mantém uma tabela com a <u>menor a distância</u> para cada destino	Em cada nó da rede, o algoritmo mantém uma tabela com o <u>melhor caminho</u> para cada destino
Distância é medida pela quantidade de roteadores que o pacote tem de atravessar (<i>hop count</i>) até o destino	Melhor caminho é definido com base em informações da rede (velocidade, atraso, taxa de ocupação de enlaces, etc.)
Roteadores vizinhos trocam entre si informações à respeito de suas tabelas de roteamento (tabela inteira)	Roteadores vizinhos trocam informações entre si à respeito dos seus enlaces (somente o que for alterado desde a última troca de informação)

Tab. 2.1 Vetor_Distância X Estado_de_Enlace

2.2.2 Controle de Congestionamento

Congestionamento ocorre quando a quantidade de pacotes na rede é muito grande (normalmente isso ocorre quando se atinge um patamar da capacidade de carga dos canais de comunicação).

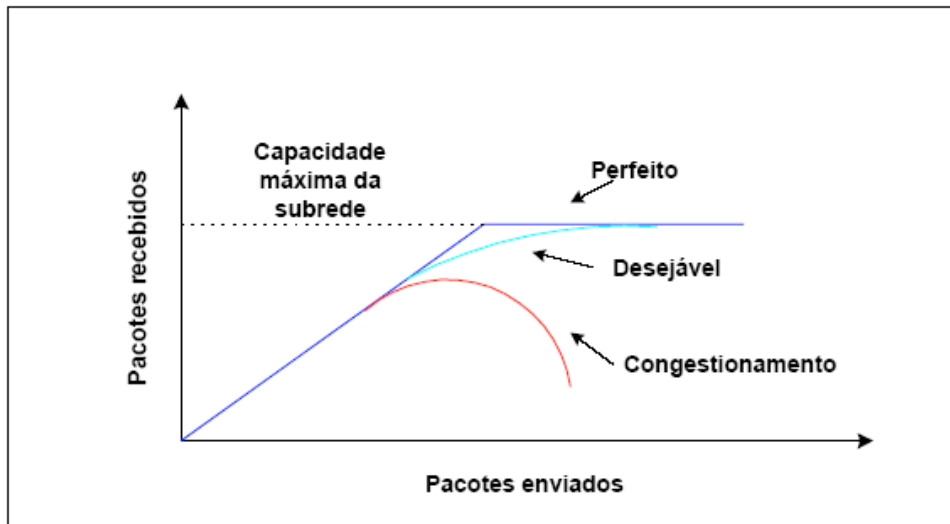


FIG 2.3. PADRÃO DE COMPORTAMENTO NA OCORRÊNCIA DE CONGESTIONAMENTO.

Fatores que ocasionam congestionamento:

Pacotes chegando por canais de comunicação rápidos, tendo de sair por canais mais lentos.

Roteadores lentos ou com pouca memória para armazenar pacotes temporariamente (memória infinita estraga tudo [Tan,96]).

Existem dois modos de controlar o fluxo, o modelo circuito aberto (*open loop*), propõe resolver os problemas na fase de projeto/configuração dos roteadores de modo a (tentar) garantir que não ocorra congestionamento. Para realizar ajustes, há a necessidade de reinicializar tudo. Já o modelo circuito fechado (*closed loop*), propõe a monitoração do sistema para detectar quando e onde o congestionamento ocorre e então passar estas informações para pontos de controle onde alguma ação pode ser tomada. Para corrigir o problema bastam ajustes na operação do sistema em funcionamento. No modelo circuito fechado, o controle pode ser explícito, quando o ponto de congestionamento avisa (de alguma forma) a origem dos pacotes (p.ex. ATM com ABR); ou implícito, quando a origem dos pacotes deduz que há congestionamento fazendo observações localmente (p.ex. pela demora no recebimento de confirmação de entrega de pacotes, p.ex. TCP/IP).

3. QUALIDADE DE SERVIÇO (QoS)

A qualidade de serviço (QoS) nas redes IP é um aspecto operacional fundamental para o desempenho fim-a-fim das novas aplicações (VoIP, multimídia, ...). Assim sendo, é importante o entendimento dos seus princípios, parâmetros, mecanismos, algoritmos e protocolos desenvolvidos e utilizados para a obtenção de uma QoS.

A obtenção de uma QoS adequada é um requisito de operação da rede e suas componentes para viabilizar a operação com qualidade de uma aplicação.

Em seguida discute-se com mais detalhe o que vem a ser o termo “Qualidade de Serviço”.

3.1 *Princípios*

Numa primeira abordagem o termo “Qualidade de Serviço” pode ser entendido da seguinte forma:

Qualidade de Serviço (QoS) é um requisito da(s) aplicação(ões) para a qual exige-se que determinados parâmetros (atrasos, vazão, perdas, ...) estejam dentro de limites bem definidos (valor mínimo, valor máximo).

A QoS é garantida pela rede, suas componentes e equipamentos utilizados. Do ponto de vista dos programas de aplicação, a QoS é tipicamente expressa e solicitada em termos de uma “Solicitação de Serviço” ou “Contrato de Serviço”. A solicitação de QoS da aplicação é denominada tipicamente de SLA (*Service Level Agreement*) [Job99] [Jam98].

A SLA deve definir claramente quais requisitos devem ser garantidos para que a(s) aplicação(ões) possam executar com qualidade. Um exemplo típico de SLA para uma aplicação de voz sobre IP (VoIP - *Voice over IP*) com algumas centenas de canais voz simultâneos numa rede IP WAN poderia ser:

- ◆ Vazão ≥ 2 Mbps;

- ◆ Atraso ≤ 250 msec
- ◆ Disponibilidade $\geq 99,5\%$

Uma vez que a rede garanta este SLA, tem-se como resultado que a aplicação VoIP em questão poderá executar garantindo a qualidade de voz prevista para os seus usuários se comunicando simultaneamente através da rede IP.

Do ponto de vista dos usuários, tem-se normalmente que a qualidade obtida de uma aplicação pode ser variável e, a qualquer momento, pode ser alterada ou ajustada (para melhor qualidade ou pior qualidade). Por exemplo, pode-se assistir um vídeo com uma qualidade de 32 fps (*Frames per Second*) ou 4 fps e, fundamentalmente, isto depende da qualidade de vídeo esperada pelo usuário final. Embora este comportamento possa ser dinâmico do ponto de vista dos usuários finais (seres humanos), do ponto de vista das redes as SLAs são estáticas e, eventualmente, podem ser alteradas. A alteração numa SLA implica, como veremos adiante, normalmente numa nova solicitação de qualidade de serviço à rede em questão.

3.2 QoS como Mecanismo Gerencial

Do ponto de vista de um gerente ou administrador de redes, a percepção da qualidade de serviço é mais orientada no sentido da utilização de mecanismos, algoritmos e protocolos de QoS em benefício de seus clientes e suporte às aplicações. Ou seja, como efetivamente a rede e suas componentes podem garantir as inúmeras SLAs definidas para diversos usuários e aplicações.

Outros aspectos importantes do ponto de vista gerencial são a escalabilidade e flexibilidade da solução implantada.

A escalabilidade dos protocolos, algoritmos e mecanismos de QoS é um assunto de pesquisa (P&D) e se torna particularmente relevante quando consideramos a possibilidade de estender a garantia de QoS através de múltiplos domínios administrativos

3.2.1. Controle de Tráfego

Alguns aspectos fundamentais ao estudo de Controle de Tráfego são necessários quando falamos de performance da rede e atendimento aos requisitos de QoS. Desta forma é possível determinar, para cada tipo de aplicação típica, como voz, vídeo, dados, etc., qual a melhor técnica, ou conjunto de técnicas, a ser aplicada para atendê-la, uma vez que existe um forte relacionamento entre a aplicação utilizada e a categoria de serviço. A essência do Controle de Tráfego é determinar quando uma nova conexão pode ser aceita ou não. Para isto a rede deve avaliar se uma determinada conexão, com um determinado perfil de QoS, pode ser suportado pela rede com os recursos disponíveis, sem prejuízo para as conexões atualmente ativas. Uma vez aceita a conexão, rede e usuário estabelecem um Contrato de Tráfego, onde a rede concorda em suportá-la segundo um conjunto de parâmetros de QoS negociados e o usuário se compromete em não exceder os limites dos parâmetros de tráfego. As funções do Controle de Tráfego estão diretamente relacionadas com esta negociação e o policiamento de QoS pela rede. Seu principal objetivo é evitar o congestionamento, mas, se isso ocorrer, também poderá atuar na recuperação da rede desta situação.

3.2.2. Componentes do controle de tráfego

A partir do controle de parâmetros de uso e de rede, de conformação de tráfego e do controle de prioridades, as diretrizes para a implementação de mecanismos de Controle de Tráfego orientados a provisão de Qualidade de Serviço se concentram nos seguintes componentes:

A flexibilidade dos mecanismos de controle de QoS é um fator determinante na aceitabilidade do mesmos pela comunidade.

3.3 QoS -Parâmetros

Como definido anteriormente, a QoS necessária às aplicações é definida em termos de uma SLA. Na especificação das SLAs são definidos os parâmetros de qualidade de serviço e alguns dos mais comumente utilizados são:

- Vazão (Banda)
- Atraso (Latência)
- *Jitter*
- Taxa de Perdas, Taxa de Erros, etc...
- Disponibilidade
- Outros

Em seguida, discute-se quais aplicações realmente necessitam da garantia de QoS e, em seguida, discute-se os parâmetros básicos de especificação da qualidade de serviço indicados acima.

3.3.1 Quais Aplicações Necessitam de QoS?

Inicialmente, é necessário considerar que não são todas as aplicações que realmente necessitam de garantias fortes e rígidas de qualidade de serviço (QoS) para que seu desempenho seja satisfatório. Dentre as novas aplicações identificadas anteriormente, as aplicações multimídia são, normalmente, aquelas que têm uma maior exigência de QoS.

No mínimo, as aplicações sempre precisam de vazão (banda) e, assim sendo, este é o parâmetro mais básico e certamente mais presente nas especificações de QoS. Este parâmetro da qualidade de serviço é normalmente considerado durante a fase de projeto e implantação da rede e corresponde a um domínio de conhecimento bem discutido e relatado na literatura técnica.

As considerações que seguem tentam identificar as exigências em termos de QoS das aplicações multimídia ilustrando algumas situações práticas.

Uma aplicação multimídia *offline* envolvendo, por exemplo, dados, gráficos e arquivos com animação, não necessita de sincronização e, assim sendo, não necessita de “cuidados especiais” (QoS) da rede. Observe que lida-se com dados correspondentes a uma animação que, em termos práticos, necessita de uma determinada vazão, eventualmente carrega a

rede, mas não exige atrasos, sincronização ou tempo de resposta. Este é um caso típico onde a necessidade de QoS reduz-se a uma necessidade de vazão, normalmente atendida pelo próprio projeto da rede.

Por outro lado, para uma aplicação multimídia de conferência de áudio, garantir apenas a vazão não é suficiente. Neste caso específico, os atrasos de comunicação e as perdas de pacotes influenciam na interatividade dos usuários e na qualidade da aplicação. Considerando números, se esta aplicação gera uma vazão (fluxo de dados) de 64 Kbps, mesmo a utilização de uma LP (Linha Privada) em rede WAN de 256 Kbps pode não ser suficiente. Neste caso, os atrasos e perdas decorrentes da operação podem prejudicar a qualidade da aplicação. Diz-se então que a aplicação exige uma qualidade de serviço da rede.

3.3.2 Vazão

A vazão (banda) é o parâmetro mais básico de QoS e é necessário para a operação adequada de qualquer aplicação.

Em termos práticos as aplicações geram vazões que devem ser atendidas pela rede. A tabela 1 em seguida ilustra a vazão típica de algumas aplicações:

<i>Aplicação</i>	<i>Vazão (Típica)</i>
Aplicações Transacionais	1 Kbps a 50 Kbps
Quadro Branco (<i>Whiteboard</i>)	10 Kbps a 100 Kbps
Voz	10 Kbps a 120 Kbps
Aplicações Web (WWW)	10 Kbps a 500 Kbps
Transferência de Arquivos (Grandes)	10 Kbps a 1 Mbps
Vídeo (<i>Streaming</i>)	100 Kbps a 1 Mbps
Aplicação Conferência	500 Kbps a 1 Mbps

Vídeo MPEG	1 Mbps a 10 Mbps
Aplicação Imagens Médicas	10 Mbps a 100 Mbps
Aplicação Realidade Virtual	80 Mbps a 150 Mbps

Tabela 3.1 - Vazão Típica de Aplicações em Rede

Como discutido, o atendimento do requisito vazão para a qualidade de serviço é um dos aspectos levados em conta no projeto da rede.

3.3.3 Latência (Atraso)

A latência e o atraso são parâmetros importantes para a qualidade de serviço das aplicações. Ambos os termos podem ser utilizados na especificação de QoS, embora o termo “latência” seja convencionalmente mais utilizado para equipamentos e o termo “atraso” seja mais utilizado com as transmissões de dados (P. ex.: atrasos de transmissão, atrasos de propagação, ...).

De maneira geral, a latência da rede pode ser entendida como o somatório dos atrasos impostos pela rede e equipamentos utilizados na comunicação. Do ponto de vista da aplicação, a latência (atrasos) resulta em um tempo de resposta (tempo de entrega da informação - pacotes, ...) para a aplicação.

Os principais fatores que influenciam na latência de uma rede são os seguintes:

- Atraso de propagação (*Propagation Delay*);
- Velocidade de transmissão e
- Processamento nos equipamentos.

O atraso de propagação corresponde ao tempo necessário para a propagação do sinal elétrico ou propagação do sinal óptico no meio sendo utilizado (fibras ópticas, satélite, coaxial, ...) e é um parâmetro imutável onde o gerente de rede não tem nenhuma influência. A tabela 2 em seguida ilustra a título de exemplo alguns valores para o atraso de

propagação entre cidades numa rede WAN utilizando fibras ópticas como meio físico de comunicação.

<i>Trecho (Round Trip Delay)</i>	<i>Atraso de Propagação</i>
Miami a São Paulo	100 mseg
New York a Los Angeles	50 mseg
Los Angeles a Hong Kong	170 mseg

Tabela 3.2 - Atrasos de Propagação - Fibras Ópticas - Exemplos

A velocidade de transmissão é um parâmetro controlado pelo gerente visando normalmente a adequação da rede à qualidade de serviço solicitada. Em se tratando de redes locais (LANs) [Tan96], as velocidades de transmissão são normalmente bastante elevadas, tendendo a ser tipicamente superior à 10 Mbps dedicada por usuário (p. ex.: utilizando *LAN Switches* [Mat97]). Além disso, considere-se também que:

- Num cenário de redes locais (LANs - redes proprietárias confinadas) tem-se apenas custos de investimento e
- Nas LANs não tem-se, pelo menos em termos de equipamentos, custos operacionais mensais.

Em se tratando de redes de longa distância (Redes corporativas estaduais e nacionais, redes metropolitanas, intranets metropolitanas, ...) as velocidades de transmissão são dependentes da escolha de tecnologia de rede WAN (Linhas privadas, *frame relay*, satélite, ATM ,....). Embora exista obviamente a possibilidade de escolha da velocidade adequada para garantia da qualidade de serviço, neste caso há restrições e/ ou limitações nas velocidades utilizadas, tipicamente devido aos custos mensais envolvidos na operação da rede. Além desse fator, existem algumas restrições quanto à disponibilidade da tecnologia e à velocidade de transmissão desejada. Em termos práticos, trabalha-se em WAN tipicamente com vazões da ordem de alguns megabits por segundo (Mbps) para grupos de usuários.

O resultado das considerações discutidas é que a garantia de QoS é certamente mais crítica em redes MAN e WAN pelo somatório de dois fatores, ambos negativos:

- Trabalha-se com velocidades (Vazão) mais baixas e
- A latência (Atrasos) é muito maior quando compara-se com o cenário das redes locais.

O terceiro fator que contribui para a latência da rede é a contribuição de atraso referente ao processamento realizado nos equipamentos. A título de exemplo, numa rede IP os pacotes são processados ao longo do percurso entre origem e destino por:

- Roteadores (comutação de pacotes)
- *LAN Switches* (comutação de quadros)
- Servidores de Acesso Remoto (RAS) (comutação de pacotes, ...)
- *Firewalls* (processamento no nível de pacotes ou no nível de aplicação, ...)
- Outros

Considerando que a latência é um parâmetro fim-a-fim, os equipamentos finais (*hosts*) também têm sua parcela de contribuição para o atraso. No caso dos *hosts*, o atraso depende de uma série de fatores, a saber:

- Capacidade de processamento do processador;
- Disponibilidade de memória;
- Mecanismos de cache;
- Processamento nas camadas de nível superior da rede (Programa de aplicação, camadas acima da camada IP ;
- Etc...

Em resumo, nota-se que os *hosts* são também um fator importante para a qualidade de serviço e, em determinados casos, podem ser um ponto crítico na garantia de QoS. Esta consideração é particularmente válida para equipamentos servidores (*Servers*) que têm a tarefa de atender solicitações simultâneas de clientes em rede.

3.3.4 Jitter

O *jitter* é um outro parâmetro importante para a qualidade de serviço. No caso, o *jitter* é importante para as aplicações executando em rede cuja operação adequada depende de alguma forma da garantia de que as informações (pacotes) devem ser processadas em períodos de tempo bem definidos. Este é o caso, por exemplo, de aplicações de voz e fax sobre IP (VoIP), aplicações de tempo real, etc...

Do ponto de vista de uma rede de computador, o *jitter* pode ser entendido como a variação no tempo e na seqüência de entrega das informações (p.ex.: pacotes) (*Packet-Delay Variation*) devido à variação na latência (atrasos) da rede.

Conforme discutido no item anterior, a rede e seus equipamentos impõem um atraso à informação (p. ex.: pacotes) e este atraso é variável devido a uma série de fatores, a saber:

- Tempos de processamento diferentes nos equipamentos intermediários (roteadores, *switches*, ...);
- Tempos de retenção diferentes impostos pelas redes públicas (*Frame relay*, ATM, X.25, IP, ...) e
- Outros fatores ligados à operação da rede.

A figura 3.1 ilustra o efeito do *jitter* entre a entrega de pacotes na origem e o seu processamento no destino. Observe que o *jitter* causa não somente uma entrega com periodicidade variável (*Packet-Delay Variation*) como também a entrega de pacotes fora de ordem.

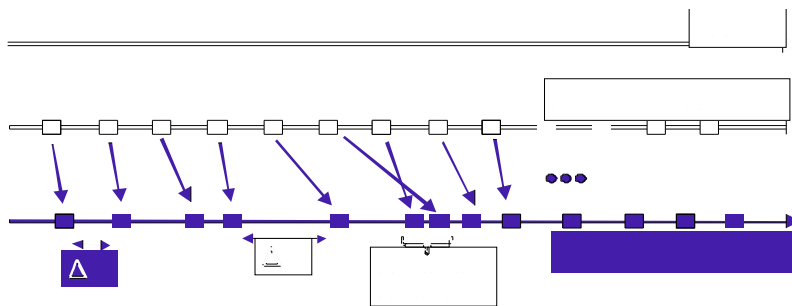


FIGURA 3.1 - EFEITO DO JITTER PARA AS APLICAÇÕES

Em princípio, o problema dos pacotes fora de ordem poderia ser resolvido com o auxílio de um protocolo de transporte como o TCP (*Transmission Control Protocol*) [Ste94] que verifica o seqüenciamento das mensagens e faz as devidas correções. Entretanto, na prática a grande maioria das aplicações multimídia optam por utilizar o UDP (*User Datagram Protocol*) [Ste94] ao invés do TCP pela maior simplicidade e menor *overhead* deste protocolo. Nestes casos, o problema de seqüenciamento deve ser resolvido por protocolos de mais alto nível normalmente incorporados à aplicação como, por exemplo, o RTP (*Real Time Transfer Protocol*) [Mau98].

O *jitter* introduz distorção no processamento da informação na recepção e deve ter mecanismos específicos de compensação e controle que dependem da aplicação em questão. Genericamente, uma das soluções mais comuns para o problema consiste na utilização de *buffers* (Técnica de “*buffering*”).

3.3.5 Perdas

As perdas de pacotes em redes IP ocorrem principalmente em função de fatores tais como:

- Descarte de pacotes nos roteadores e *switch routers* (Erros, congestionamento) e
- Perda de pacotes devido à erros ocorridos na camada 2 (PPP - *Point-to-Point Protocol*, ethernet, *frame relay*, ATM, ...) durante o transporte dos mesmos.

De maneira geral, as perdas de pacotes em redes IP são um problema sério para determinadas aplicações como, por exemplo, a voz sobre IP. Neste caso específico, a perda de pacotes com trechos de voz digitalizada implica numa perda de qualidade eventualmente não aceitável para a aplicação. O que fazer em caso de perdas de pacotes é uma questão específica de cada aplicação em particular.

Do ponto de vista da qualidade de serviço da rede (QoS) a preocupação é normalmente no sentido de especificar e garantir limites razoáveis (Taxas de Perdas) que permitam uma operação adequada da aplicação.

3.3.6 Disponibilidade

A disponibilidade é um aspecto da qualidade de serviço abordada normalmente na fase de projeto da rede.

Em termos práticos, a disponibilidade é uma medida da garantia de execução da aplicação ao longo do tempo e depende de fatores tais como:

- Disponibilidade dos equipamentos utilizados na rede proprietária (Rede do cliente) (LAN, MAN ou WAN) e
- Disponibilidade da rede pública, quando a mesma é utilizada (Operadoras de telecomunicações, *carriers*, ISPs - *Internet Service Providers*, ...).

As empresas dependem cada vez mais das redes de computadores para a viabilização de seus negócios (Comércio eletrônico, *home-banking*, atendimento *online*, transações *online*) e, neste sentido, a disponibilidade é um requisito bastante rígido. A título de exemplo, requisitos de disponibilidade acima de 99% do tempo são comuns para a QoS de aplicações WEB, aplicações cliente/servidor e aplicações de forte interação com o público, dentre outras.

3.4 O Modelo DiffServ

A B-ISDN trouxe a idéia de serviços diferenciados. Pelas razões explicadas antes, redes IP eram melhores do que redes ATM. Dentro do INTSERV WG, foi feita uma tentativa de prover diferenciação de serviços. A limitação desse modelo (escalabilidade, flexibilidade em construir serviços customizados) levantou a necessidade por um modelo novo, e de certa forma complementar. Isso fica evidenciado do primeiro parágrafo do primeiro

documento relatado ao modelo DiffServ [DiffServ_2474]: “ Serviços Diferenciados tem a intenção de habilitar discriminação de serviço escalável na Internet sem a necessidade de estados por fluxo e sinalização a cada hop. A variedade de serviços podem ser construídas de um pequeno conjunto bem definido de blocos que serão abertos nos nodos da rede.” Permitindo escalabilidade e flexibilidade na construção de serviços customizados que são os focos principais do modelo. Os próximos parágrafos focam nos princípios de projeto, da arquitetura DiffServ, os blocos de construção de uma rede DiffServ e um modelo conceitual de um roteador DiffServ.

Em um ambiente de serviços diferenciados provê qualidade de serviço em um domínio de rede aplicando regras para criar tráfego agregado e casando cada um desses tipos agregados com um caminho específico de tratamento no domínio através de um código no cabeçalho IP. O ambiente tenta acompanhar uma lista de requerimentos presentes [DiffServ_2475] e resumido como segue [Kil,99]:

- **Versatilidade:** Uma larga variedade de serviços fim-a-fim deveria ser possível de realizar; serviços de rede deveriam ser independentes de aplicações, e eles deveriam ser diretamente aplicáveis com as aplicações e serviços de rede correntes.
- **Simplicidade:** O sistema em geral ou partes dele não deveria depender de sinalização para fluxos individuais; somente um pequeno conjunto de comportamentos de encaminhamento deveria ser necessário.
- **Custo benefício:** Informação sobre um fluxo individual não deveria ser usada em nós principais. Somente estados de fluxos agregados deveriam ser usados em nodos principais.

3.5 Classificação de tráfego e armazenamento

Anteriormente, nós introduzimos o conceito de Contrato de Nível de Serviço (SLA) como a parte formal que descreve o tipo de serviço a ser fornecido por um provedor de serviço a um consumidor. Esse contrato vislumbra aspectos técnicos (condicionamento de tráfego, avaliabilidade de serviço) e aspectos comerciais (tarifação, regras contratuais). Um acordo de condicionamento de tráfego (TCA) é um subconjunto de SLA e se refere a classificação

de endereços de tráfego e regras de condicionamento a serem aplicadas a um fluxo de dados entrando na rede. Uma nova terminologia tem sido introduzida para descrever esses elementos de SLA e TCA que são contemplados pela DiffServ. Especificação de Nível de Serviço (SLS) descreve um conjunto de parâmetros e seus valores que juntos definem o serviço oferecido para um fluxo de tráfego por um provedor de serviço. Especificação de Condicionamento de Tráfego (TCS) é usado para descrever um conjunto de parâmetros e seus valores que juntos especificam um conjunto de regras classificadoras e um perfil de tráfego.

Classificação de Tráfego é usada para descrever a atividade de dividir uma única entrada de dados em múltiplas saídas. O propósito é conduzir os pacotes de acordo com regras especificadas para um elemento condicionador tráfego para processamento posterior. A seleção é baseada no conteúdo de alguma porção cabeçalho do pacote. Dois tipos de classificadores foram definidos. O classificador de Comportamento Agregado BA (Behaviour Aggregate) classifica os pacotes baseado somente no código DS. O classificador multi campo MF (Multi-Field) seleciona pacotes baseado no valor de uma combinação de um ou mais campos de cabeçalho, tais como endereço fonte, endereço destino, código DS, tipo de protocolo, número da porta fonte e destino, e outras informações como interface de entrada. Condicionamento de tráfego é usado para descrever a atividade de garantir que o tráfego entrando no domínio DiffServ está de acordo com as regras especificadas no TCS, de acordo com o política de segurança do provedor de serviços. Essa atividade é feita através dos seguintes mecanismos:

- **Métricas (*Meter*)**. Elas medem as propriedades temporais do fluxo de pacotes selecionados por um classificador contra um perfil de tráfego especificado em um TCS. Uma métrica passa informação de estado para outras funções de condicionamento para disparar uma ação particular para cada pacote que está dentro ou fora do perfil de disparo.
- **Marcadores (*Marker*)**. Eles marcam o campo DiffServ de um pacote para um código particular adicionando o pacote marcado a um comportamento DiffServ agregado.
- **Modeladores (*Shaper*)**. Eles atrasam alguns ou todos os pacotes em um fluxo de dados para acomodar o fluxo no perfil do tráfego.

• **Descartadores (Droppers).** Eles descartam alguns ou todos os pacotes em um fluxo de dados para acomodar o fluxo no perfil do tráfego. Esse processo é conhecido como “policiamento” do fluxo.

Para facilitar a definição das funções de condicionamento específicas que um fluxo de tráfego pode passar, o Bloco Condicionador de Tráfego foi introduzido. Pode ser visto como uma entidade com uma entrada e uma ou mais saídas e um conjunto de parâmetros de controle.

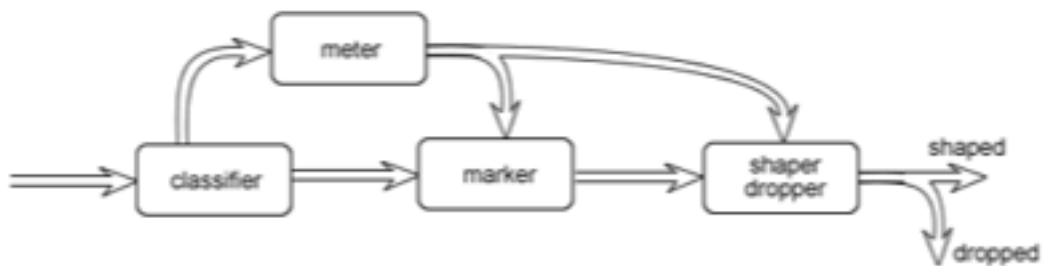


FIG 3.2 TCB – BLOCO CONDICIONADOR DE TRÁFEGO DIFFSERV

3.5.1 *Per-Hop Behaviors* (PHB)

O grupo de trabalho DiffServ definiu o comportamento por nodo (PHB) como o tratamento de encaminhamento externamente observável aplicado por um nodo DiffServ reclamante, em cada nodo da rede, mapeando o DSCP (Differentiated Services Code Point) de qualquer pacote para um PHB em particular é a forma de realizar serviços diferenciados ao longo do caminho de saída de um pacote. PHB's podem ser especificados individualmente, ou como um grupo (um único PHB é um caso especial de um grupo PHB). Um grupo PHB usualmente consiste de um conjunto de dois ou mais PHB's que podem somente ser especificados e implementados simultaneamente, devido a restrições comuns aplicadas a cada PHB dentro do grupo, como uma fila de serviço ou fila de policiamento [DiffServ_2474]. Os seguintes conjuntos de PHB's foram definidos pelo grupo de trabalho DiffServ : PHB Padrão, PHB's Seleccionadores de Classe, Encaminhamento expresso PHB (EF) e Grupos PHB de Encaminhamento Seguro (AF). Uma breve descrição é apresentada nos próximos parágrafos.

- **PHB Padrão**

Cada nodo DS precisa prover um PHB padrão e o código recomendado é 000000. O comportamento de melhor esforço necessita um tratamento externo. Uma implementação razoável seriam disciplinas de fila que enviam pacotes desse agregado até que o link externo não seja solicitado para satisfazer outro PHB. Isso poderia ser garantido por um mecanismo na qual cada nodo reserva alguns mínimos recursos (p.e. buffers, largura de banda) para comportamentos padrões agregados. Isso permite que usuários que não fazem parte da rede de serviços diferenciados continuem a usar a rede da mesma maneira que hoje.

- **PHB Seleccionador de Classe**

Para preservar compatibilidade com qualquer esquema precedente de IP em uso na rede , os valores XXX000 do DSCP tem sido reservados. Esses valores DSCP são chamados Códigos Seleccionadores de Classe e mapeiam em PHB's chamados seleccionadores de classe. Para isso, os requerimentos foram definidos no campo DS para garantir que nodos DS possam coexistir com nodos baseados em precedência de IP.

- ***Expedited Forwarding (EF):***

Provê o maior nível de qualidade de serviço porque emula uma linha dedicada convencional minimizando os atrasos, probabilidade de perda e *jitter* para os pacotes. Os Mecanismos utilizados para isso são *traffic shaping*, buferização (*buffering*) e priorização de filas.

- ***Assured Forwarding (AF):***

Emula um comportamento semelhante a uma rede com pouca carga mesmo durante a ocorrência de congestionamento.

A latência negociada é garantida com um alto grau de probabilidade.

AF define 4 níveis de prioridade de tráfego (Ouro, Prata, Bronze e *Best Effort*)*.

Para cada nível de prioridade são definidos 3 preferências de descarte de pacotes (semelhante ao *Frame Relay*).

Este serviço usa mecanismos de *Traffic Shaping (Token Bucket)* e usa o algoritmo RED (*Randon Early Detection*), durante o congestionamento

4. QoS NO LINUX

Este capítulo comenta o suporte de QoS existente nos kernels de linux mais recentes. O suporte de QoS no *kernel* provê a ferramenta para a implementação de várias tecnologias IP QoS como serviços integrados e serviços diferenciados. Aqui serão discutidos os detalhes de configuração, implementação e uso do suporte de QoS no linux.

4.1 Configuração

O suporte à qualidade de serviço está disponível desde as versões 2.1.90 do kernel. Entretanto, o suporte é mais compreensivo nos kernels mais recentes. Este documento foi escrito com referência à versão 2.6.10.8 do *kernel*. A partir do kernel 2.4 os serviços diferenciados foram integrados ao kernel, somente sendo necessário alterar a configuração padrão. Os últimos kernels do linux podem ser obtidos de <http://www.kernelnotes.org/>.

Para habilitar DiffServ siga os seguintes passos:

1. Faça um 'make xconfig' ou 'make menuconfig' ou 'make config' no diretório /usr/src/linux.
2. Em opções de rede, ponha 'y' para as seguintes opções de *kernel*: *Kernel/User netlink socket*, *Routing messages*, *TCP/IP networking* e *QoS and/or fair queueing*. Ligando a opção *QoS and/or fair queuing*, habilita o CBQ, CSZ, PRIO, RED, SFQ, TEQL, TBF, GRED, DS_MARK, classificador *tcindex*, Classificador de pacotes API, classificador U32 e classificador baseado em tabela de roteamento.
3. Faça um 'make dep; make clean; make bzilo'
4. Reinicie o linux usando a nova imagem do *kernel*.

Tendo discutido a configuração do suporte de QoS no linux, Serão visto os detalhes envolvidos na implementação dessas características. Todos os arquivos de kernel relacionados no documento estão localizados no diretório /usr/src/linux.

Para mais informações sobre parâmetros de QoS no kernel consultar o documento /usr/src/linux/Documentation/networking/ip-sysctl.txt.

O principio básico envolvido na implementação de QoS no linux é mostrado na Figura 4.1. Essa figura mostra como o kernel processa os pacotes que chegam, e como gera os pacotes

a serem enviados para a rede. O de-multiplexer de entrada examina os pacotes de entrada para determinar se os pacotes são destinados para o nodo local. Em caso afirmativo, eles são enviados para a camada mais alta para processamento posterior. Se não, envia os pacotes para o bloco seguinte. O bloco seguinte, que pode também ser recebido localmente gerou pacotes da camada superior, verifica a tabela de roteamento e determina o próximo hop para o pacote. Depois disso, enfileira o pacote a ser transmitido na interface de saída. É nesse ponto que controle de tráfego do linux atua. O controle de tráfego pode ser usado para construir uma complexa combinação de filas, classes e filtros que controlam os pacotes que são enviados à interface de saída. [Qos]

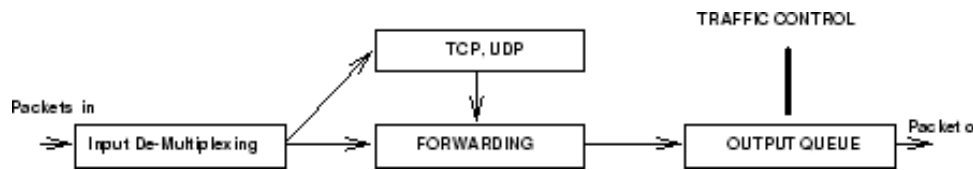


FIGURA 4.1 CONTROLE DE TRÁFEGO DO LINUX

Quando políticas de filas são criadas para um dispositivo, um ponteiro para a fila é mantido na estrutura do dispositivo (em `include/netdevice.h`). A camada IP, depois de adicionar a informação de cabeçalho ao pacote (`in_net/ipv4/ip_output.c`), chama a função `dev_queue_xmit` (em `net/core/dev.c`). A porção desse código é mostrada abaixo.

```

q = dev->qdisc;
if (q->enqueue) {
    q->enqueue(skb, q);
    qdisc_wakeup(dev);
    return 0;
}
...
if (dev->hard_start_xmit(skb, dev) == 0)

```

Essa função mostra que antes de enviar o pacote para a interface de saída (fazendo um `hard_start_xmit`), o pacote é enfileirado na fila mantida pelo dispositivo, se não existe. Então, como mencionado anteriormente, controle de tráfego é implementado pouco antes do pacote ser enviado para o driver.

Como foi mencionado, o mecanismo de controle de tráfego provê a ferramenta básica para o desenvolvimento de serviços integrados e serviços diferenciados no linux. Isto é mostrado na figura 4.2.

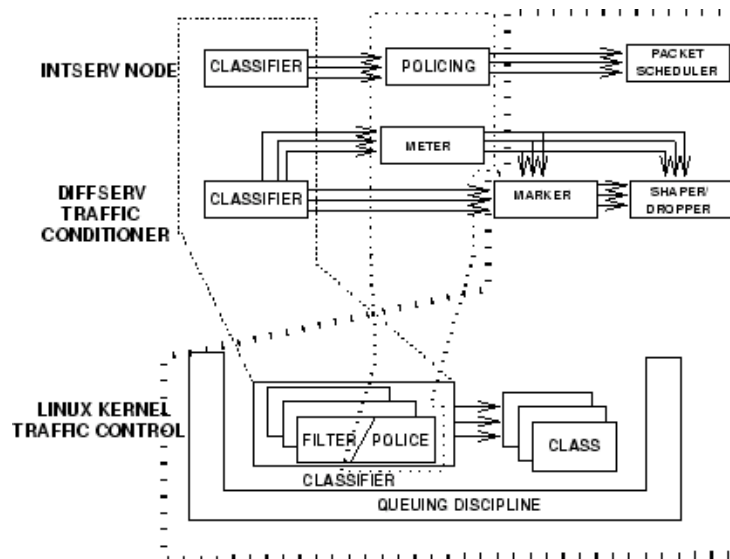


FIGURA 4.2 FERRAMENTA PARA O DESENVOLVIMENTO DE INTSERV E DIFFSERV

Como visto na Figura 4.2, o suporte de QoS no linux consiste dos três blocos básicos de construção, chamados:

- Disciplinas de enfileiramento
- Classes
- Filtros/Policers

4.2 Políticas de Enfileiramento

Essa seção discute as políticas, que formam um bloco de construção básico para o suporte de QoS no linux. Cada dispositivo de rede tem uma fila associada à ele. Existem 11 tipos de políticas de fila que são atualmente suportados no linux, as quais são:

- Class Based Queue (CBQ)
- Token Bucket Flow (TBF)
- Clark-Shenker-Zhang (CSZ)
- First In First Out (FIFO)
- Priority

- Traffic Equalizer (TEQL)
- Stochastic Fair Queuing (SFQ)
- Asynchronous Transfer Mode (ATM)
- Random Early Detection (RED)
- Generalized RED (GRED)
- Diff-Serv Marker (DS_MARK)

Filas são identificadas por um marcador <maior número:menor número>, onde o menor número é zero para filas. Marcadores são usados para associar classes à políticas de enfileiramento.

Disciplinas de enfileiramento e classes estão amarradas uma à outra. A presença de classes e suas semânticas são propriedades fundamentais das políticas de filas. Em contraste, filtros podem ser arbitrariamente combinados com políticas de fila e classes, assim como as filas tem classes. Nem todas as políticas de fila estão associadas com classes. Por exemplo, a Token Bucket Flow (TBF) não possui qualquer classe associada a ela.

A Figura 4.3 mostra uma fila exemplo. Neste exemplo, existem dois tipos de filas, uma para alta prioridade e outra para baixa. O filtro seleciona o escolhido para alta prioridade, enquanto a remanescente é tratada como baixa prioridade. Os pacotes de baixa prioridade são servidos por uma fila FIFO, enquanto os pacotes de alta prioridade são servidos por um algoritmo *Token Bucket Flow Algorithm*. A fila TBF é usada para garantir que os pacotes de baixa prioridade não estão “famintos”.

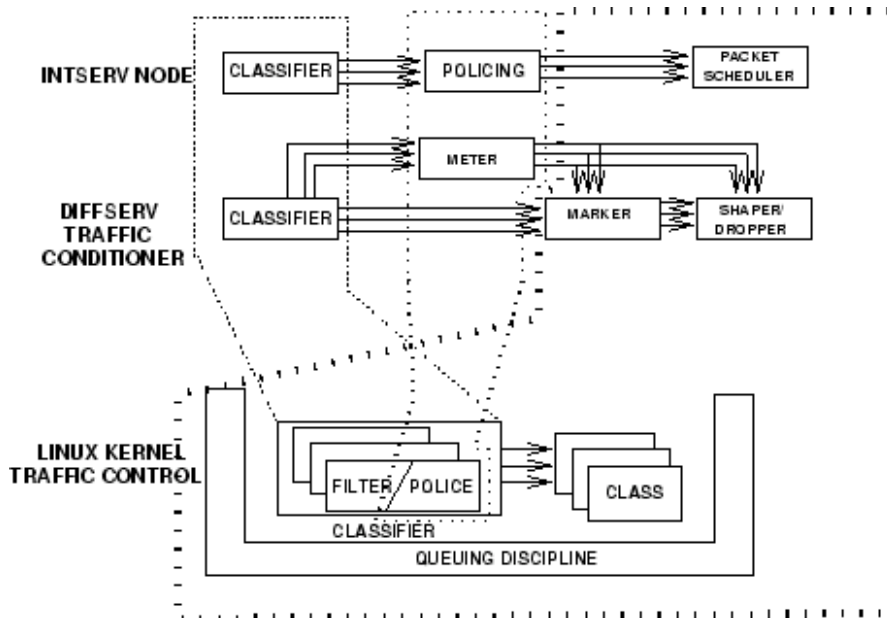


FIGURA 4.3 UMA FILA EXEMPLO

Uma das principais vantagens do suporte de QoS no linux é a flexibilidade na qual a combinação de filas e classes podem ser combinadas. Cada política de enfileiramento pode ter um número de classes. Essas classes não guardam os pacotes por si próprias, usam outras filas para este propósito, as quais, podem ter um número de classes e assim sucessivamente. É essa flexibilidade que faz o suporte de QoS em linux único.

Quando um kernel linux configurado para suporte de QoS é iniciado, a função `net_dev_init` (in `net/core/dev.c`) chama a função `pktsched_init` (em `net/sched/sch_api.c`) para inicializar a unidade de controle de tráfego no kernel do linux. Em `pktsched_init`, as políticas de enfileiramento que tenham sido compiladas no kernel são todas registradas e inicializadas. Os ponteiros para acessar as funções `tc_ctl_qdisc`, `tc_dump_qdisc`, `tc_ctl_tclass` e `tc_dump_tclass`, nas quais são usadas para realizar várias funções em filas e classes também são inicializadas em `pktsched_init`.

4.3 Classes

Como já foi dito, filas e classes estão amarradas uma à outra. Cada classe tem uma fila, que por padrão é do tipo FIFO. Quando a função enfileirar de uma fila é chamada, a política de enfileiramento aplica os filtros para determinar a classe na qual o pacote pertence, que então chama a função enfileirar da fila que pertence a esta classe.

- Graft

A função `graft` de uma classe é usada para associar uma nova disciplina de enfileiramento à classe. Como mencionado na seção anterior, a política padrão associada à classe quando ela é criada é a fila FIFO. Para mudar a política, uma operação `graft` é executada na classe. Abaixo o código da função `cbq_graft` em `net/sched/sch_cbq.c`.

```
static int cbq_graft(struct Qdisc *sch, unsigned long arg,
struct Qdisc *new, struct Qdisc **old)
{
    struct cbq_class *cl = (struct cbq_class*)arg;

    if (cl) {
        if (new == NULL) {
```

```

if ((new = qdisc_create_dflt(sch->dev, &pfifo_qdisc_ops)) == NULL)
    return -ENOBUFFS;
new->classid = cl->classid;
}

if ((*old = xchg(&cl->q, new)) != NULL)
    qdisc_reset(*old);

return 0;
}
return -ENOENT;
}

```

- Get

A função `get` é usada para retornar o ID interno de uma classe, dado seu ID de classe. A função `get/` incrementa a contagem de uso de uma classe. No arquivo `net/sched/sch_cbq.c` pode ser encontrado o código fonte da função abaixo:

```

static unsigned long cbq_get(struct Qdisc *sch, u32 classid)
{
    struct cbq_sched_data *q = (struct cbq_sched_data *)sch->data;
    struct cbq_class *cl = cbq_class_lookup(q, classid);

    if (cl) {
        cl->refcnt++;
        return (unsigned long)cl;
    }
    return 0;
}

static __inline__ struct cbq_class *
cbq_class_lookup(struct cbq_sched_data *q, u32 classid)
{
    struct cbq_class *cl;

    for (cl = q->classes[cbq_hash(classid)]; cl; cl = cl->next)
        if (cl->classid == classid)
            return cl;
    return NULL;
}

```

- Change

A função `change` em uma classe é usada para mudar as propriedades associadas com uma classe. Entretanto, a função `change` também é usada para criar classes as vezes. Como exemplo, a função `cbq_change` em `net/sched/sch_cbq.c` em detalhe. A função `cbq_change` é invocada com a política de enfileiramento, o ID de classe da classe cujas propriedades necessitam ser modificadas ou adicionadas e novas propriedades que necessitam ser associadas à essa classe. Isto pode ser visto abaixo:

```
static int
cbq_change(struct Qdisc *sch, u32 classid, u32 parentid, struct rtattr **tca,
unsigned long *arg)
{
.
.
}
```

Depois de realizar alguns testes iniciais, as propriedades especificadas são associadas à classe uma por uma. Isto pode ser visto na seguinte porção de código:

```
if (tb[TCA_CBQ_WRRROPT-1]) {
    cbq_rmprio(q, cl);
    cbq_set_wrr(cl, RTA_DATA(tb[TCA_CBQ_WRRROPT-1]));
}

if (tb[TCA_CBQ_OVL_STRATEGY-1])
    cbq_set_overlimit(cl, RTA_DATA(tb[TCA_CBQ_OVL_STRATEGY-1]));
```

Esta porção de código ativa as prioridades e parâmetros *round robin* balanceados para uma classe. Também ativa a informação limite. Deve-se notar que essa função pode ser usada não só para mudar as propriedades de uma classe existente, mas também para criar uma nova classe com as propriedades especificadas. Este fato se torna claro na seguinte porção de código.

```
static int
cbq_change(struct Qdisc *sch, u32 classid, u32 parentid, struct rtattr **tca,
unsigned long *arg)
{
    struct cbq_class *cl = (struct cbq_class*)*arg;
.
.
    if (cl) {
        /* Change properties */
```

```

.
.
return 0;
}

/* Cria uma nova classe e designa suas propriedades */
.
.
.
}

```

A função `change` é chamada na função `tc_ctl_tclass` em `net/sched/sch_api.c`. A função `tc_ctl_tclass` é chamada não importando se existe uma tentativa de criar, deletar ou mudar uma classe.

- Delete

A função `delete` é usada para apagar uma classe. Determina o uso de uma classe, checando a contagem de referência, e se zero, desativa e remove a classe. Como exemplo, o código da função `cbq_delete` em `net/sched/sch_cbq.c`.

```

static int cbq_delete(struct Qdisc *sch, unsigned long arg)
{
    struct cbq_sched_data *q = (struct cbq_sched_data *)sch->data;
    struct cbq_class *cl = (struct cbq_class*)arg;

    if (cl->filters || cl->children || cl == &q->link)
        return -EBUSY;

    start_bh_atomic();

    if (cl->next_alive)
        cbq_deactivate_class(cl);

    if (q->tx_borrowed == cl)
        q->tx_borrowed = q->tx_class;
    if (q->tx_class == cl) {
        q->tx_class = NULL;
        q->tx_borrowed = NULL;
    }

    cbq_unlink_class(cl);
    cbq_adjust_levels(cl->tparent);
    cl->defmap = 0;
    cbq_sync_defmap(cl);
}

```

```

cbq_rmprio(q, cl);

if (--cl->refcnt == 0)
    cbq_destroy_class(cl);

end_bh_atomic();

return 0;
}

```

- Walk

A função walk de uma classe é usada para realizar uma iteração sobre todas as classes de uma fila e invoca uma função de retorno para cada uma das classes. Isso é usualmente usado para obter dados diagnósticos para todas as classes de uma fila. Abaixo, o código da função cbq_walk function em net/sched/sch_cbq.c.

```

static void cbq_walk(struct Qdisc *sch, struct qdisc_walker *arg)
{
    struct cbq_sched_data *q = (struct cbq_sched_data *)sch->data;
    unsigned h;
    .
    .

    for (h = 0; h < 16; h++) {
        struct cbq_class *cl;

        for (cl = q->classes[h]; cl; cl = cl->next) {
            if (arg->count < arg->skip) {
                arg->count++;
                continue;
            }

            if (arg->fn(sch, (unsigned long)cl, arg) {
                arg->stop = 1;
                break;
            }
            .
            .
        }
    }
}

```

Essa porção de código mostra que o comando walk percorre todas as classes de uma fila específica e invoca uma função de retorno. A função walk function é chamada da função

tc_dump_tclass function em net/sched/sch_api.c, que por sua vez é invocada quando uma requisição de dump é feita na classe. A porção de código em tc_dump_tclass que faz isso é mostrado abaixo:

```
static int tc_dump_tclass(struct sk_buff *skb, struct netlink_callback *cb)
{
    struct device *dev;
    struct Qdisc *q;
    .
    .
    .
    for (q=dev->qdisc_list, t=0; q; q = q->next, t++) {
        .
        arg.w.fn = qdisc_class_dump;
        .
        q->ops->cl_ops->walk(q, &arg.w);
        .
        .
    }
}
```

A função de retorno mapeia para a função qdisc_class_dump no mesmo arquivo. Em qdisc_class_dump, a função tc_fill_tclass function é invocada, que por sua vez chama a função dump em todas as classes.

- Tcf_chain

A função tcf_chain de uma classe é usada para retornar a ancora para a lista de filtros que são associados à uma classe. Cada classe é associada com um lista filtro que contém a lista de filtros que são usados para identificar os pacotes que pertencem a uma classe em particular. Como já foi mencionado, pacotes com diferentes propriedades podem mapear para a mesma classe. Por exemplo, pacotes de duas fontes diferentes. Como resultado, podem existir múltiplos filtros associados a uma classe. Filtros são discutidos em mais detalhes na próxima seção. A função cbq_find_tcf function em net/sched/sch_cbq.c é mostrada abaixo como exemplo:

```
static struct tcf_proto **cbq_find_tcf(struct Qdisc *sch, unsigned long arg)
{
    struct cbq_sched_data *q = (struct cbq_sched_data *)sch->data;
    struct cbq_class *cl = (struct cbq_class *)arg;
```

```

    if (cl == NULL)
        cl = &q->link;

    return &cl->filter_list;
}

```

A função `cbq_find_tcf` retorna o ponteiro para a lista de filtros. Filtros são mantidos em uma estrutura chamada `tcf_proto` em `include/net/pkt_cls.h`. A função `tcf_chain` é chamada na função `tc_ctl_tfilter` em `net/sched/cls_api.c`. Essa função é chamada quando uma tentativa é feita para criar/modificar/apagar/pegar um filtro nodo.

- `Bind_tcf`

A função `bind_tcf` é usada para associar uma instância de um filtro a uma classe. A função `cbq_bind_filter` em `net/sched/sch_cbq.c` é um exemplo para a função `bind_tcf` de uma classe.

```

static unsigned long cbq_bind_filter(struct Qdisc *sch, u32 classid)
{
    struct cbq_sched_data *q = (struct cbq_sched_data *)sch->data;
    struct cbq_class *cl = cbq_class_lookup(q, classid);

    if (cl) {
        cl->filters++;
        return (unsigned long)cl;
    }
    return 0;
}

```

Como é visto acima, nessa função, quando a função `cbq_bind_filter` é chamada, o contador de filtro para a classe é incrementado. Essa função é quase similar a função `get` de uma classe. A diferença é que uma classe que é apontada por filtros não pode ser apagada sem apagar os filtros primeiro. Isto é, a política de enfileiramento explicitamente nega requisições para apagar uma classe se a classe está em uso. Isso pode ser visto abaixo na porção de código da função `cbq_delete` em `net/sched/sch_cbq.c`

```

static int cbq_delete(struct Qdisc *sch, unsigned long arg)
{
    .
    .
    if (cl->filters || cl->children || cl == &q->link)

```



```

    return -EBUSY;
.
.
.
}

```

Entretanto, existe uma falha na implementação da política de enfileiramento dsmark. Nesse caso, as funções `get` e `bind_tcf` mapeiam para a função `dsmark_get`, O que é incorreto. Então uma classe sendo apontada por um filtro pode ser apagada, O que não é um comportamento desejável. [Qos]

- `Unbind_tcf`

A função `unbind_tcf` é usada para remover uma instância de um filtro associado a uma classe. A função `cbq_unbind_filter` em `net/sched/sch_cbq.c` é um exemplo para a função `unbind_tcf` de uma classe.

```

static void cbq_unbind_filter(struct Qdisc *sch, unsigned long arg)
{
    struct cbq_class *cl = (struct cbq_class*)arg;

    cl->filters--;
}

```

Essa porção de código indica claramente que a função `unbind_tcf` é similar a função `put` de uma classe. O contador de filtros associado a classe é decrementado. Para a classe a ser apagada, essa contagem precisa ser zero.

- `Dump_class`

A função `dump_class` de uma classe é usada para fazer um diagnóstico dos dados de uma classe. Existe uma grande quantidade de dados que são mantidos sobre as classes e a função `dump` é usada para observar esses valores. Não existe nada mais que precisa ser elaborado na função `dump_class`.

Quando a função `enqueue` de uma fila é chamada, a função `tc_classify` em `include/net/pkt_cls.h` é invocada para determinar a classe na qual o pacote pertence. A forma mais simples de classificação possível é a especificação do campo `skb->priority` (em `struct sk_buff` in `include/linux/skbuff.h`). Se `skb->priority` é especificado, então não é feita

nenhuma outra tentativa de classificação. `skb->priority` é setado para `sk->priority` (em `include/net/sock.h`) quando o pacote é criado. O valor de `sk->priority` pode ser especificado com a ajuda da função `setsockopt`. A opção `SO_PRIORITY` na chamada de `setsockopt` necessita ser usada para esse propósito. Entretanto até a versão 2.2.3 do kernel, o valor de `sk->priority` é limitado entre 0 e 7. Entretanto essa forma de classificar pacotes não funciona. É válido mencionar que a esse ponto que `skb->priority` pode conter outros valores como o byte TOS no cabeçalho IP. Todos esses valores são menores de 65536, que é o menor valor válido como número de classe (como o mínimo valor possível para o maior número de uma classe é 1).

4.4 Filtros

Filtros são usados para classificar pacotes baseada em certas propriedades de um pacote, por exemplo, *byte* TOS no cabeçalho IP, endereço IP, números de porta, etc. É invocado quando a função `enqueue` de uma fila é chamada. Políticas de enfileiramento usam filtros para assignar os pacotes que chegam para uma de suas classes.

Filtros podem ser mantidos por classe ou por fila baseado no forma da política de enfileiramento. Como já foi dito, filtros são mantidos em listas de filtro. A lista de filtro é especificada como um `struct tcf_proto`, em `include/net/pkt_cls.h`.

```
struct tcf_proto
{
    /* Fast access part */
    struct tcf_proto    *next;
    void                *root;
    int                 (*classify)(struct sk_buff*, struct tcf_proto*,
                                   struct tcf_result *);
    u32                 protocol;

    /* All the rest */
    u32                 prio;
    u32                 classid;
    struct Qdisc        *q;
    void                *data;
    struct tcf_proto_ops *ops;
};
```

Essa estrutura é usada para representar listas de filtro e é mantida pelas classes e filas. Como exemplo, a estrutura `cbq_class` em `net/sched/sch_cbq.c` mantém a lista de filtro usando a estrutura `tcf_proto`. A função `tcf_chain` em classes, descrita na seção anterior, é usada para retornar a ancora para uma lista de filtro, que pode ser usada para percorrer a lista de filtro. Listas de Filtro são ordenadas por prioridade, em ordem ascendente. As entradas são chaveadas pelo protocolo pela qual elas aplicam, por exemplo, IP, UDP etc. Filtros para o mesmo protocolo na mesma lista podem ter valores de prioridade diferentes. Os números de protocolo são usados em `skb->protocol` e eles são definidos em `include/linux/if_ether.h`.

Filtros também podem ter uma estrutura interna: pode controlar elementos internos, que são referenciados por um *handle*. Esses *handles* são de 32 bits mas são divididos em números maiores e menores como identificadores de classe. *Handle 0* refere ao próprio filtro. Como classes, filtros também possuem um identificador interno, que pode ser obtido com a ajuda da função `get`. A estrutura básica de filtros é mostrada na Figura 4.4.

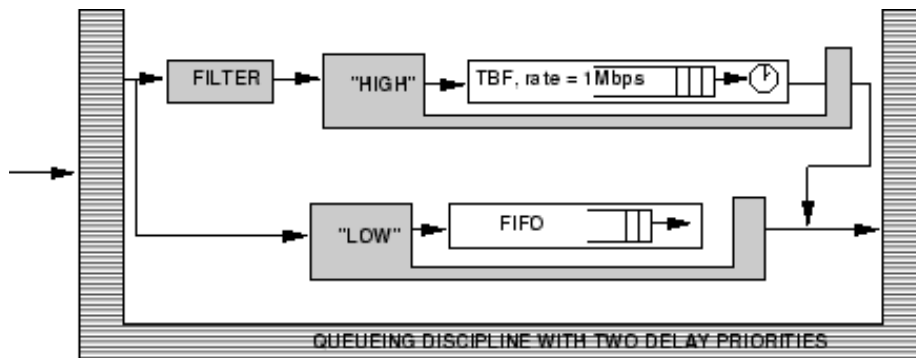


FIGURA 4.4: ESTRUTURA DE FILTROS [QOS]

5. COMPARAÇÃO DOS TESTES LINUX x FREEBSD

Aqui são mostrados os resultados dos testes feitos na rede. No primeiro caso, com a rede utilizando QoS, e no segundo caso sem o QoS.

Para isso, o ambiente de testes foi usado em dois momentos distintos: primeiro momento utilizando o roteamento IP clássico; e em um segundo momento usando roteamento com QoS, ou seja, o roteador irá controlar o fluxo de pacotes no enlace L2 priorizando os pacotes de aplicações VoIP e controlando o fluxo das demais aplicações de forma a não interferirem no uso das aplicações de VoIP mais sensíveis a variações de tráfego na rede. Nestes dois momentos, serão realizados 2 testes, o primeiro enlace não-saturado e o outro com o enlace saturado, visando medir o desempenho e os resultados são confrontados. A duração de cada teste foi de 30s ao contrário do FreeBSD em que os testes duraram 60s. Este valor de 30 segundos foi escolhido por ser um valor suficiente para análise e por facilitar a visualização dos gráficos. Esta análise visa comparar, em situações de tráfego igual e trafegando pelos mesmos nós com as mesmas condições, é revelar se um Roteador Linux é adequado para efetuar o do Controle de Banda de forma a priorizar o tráfego VoIP de uma empresa.

Os gráficos apresentados seguem a ordem Linux depois FreeBSD.

5.1 Definição da Rede

O primeiro passo para a realização da avaliação de desempenho pretendida foi definir a topologia de rede mais adequada ao nosso propósito. E depois a simulação de um link de 1Mbps (Enlace L2 conforme podemos observar na Figura 5.1) alcançado através da uso da disciplina de fila associada as placas de rede do roteador Linux, primeiramente controlando o tráfego e utilizando o mecanismo de enfileiramento de pacotes FIFO e no segundo teste além de controlar o tráfego utilizou-se o mecanismo de enfileiramento CBQ, devido à possibilidade da criação de regras que otimizem o uso da banda ao máximo. A topologia criada está disposta conforme a figura 5.1. O ambiente de testes definido tem 1 roteador e 2 computadores sendo um deles somente para gerar e o outro para gerar e receber tráfego, com exceção do roteador, os outros computadores rodam o Sistema Operacional FreeBSD.

As 6 placas de rede utilizadas no projeto são do modelo Intel Express Pro 100 operando em 10baseT/UTP. Essa placa foi escolhida devido a sua qualidade comprovada em outros experimentos realizados no laboratório anteriormente.

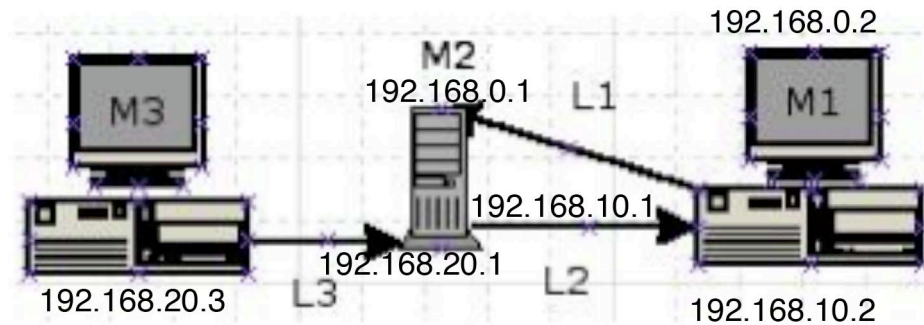


FIG 5.1 CONFIGURAÇÃO DA REDE

5.2 Resultado dos testes da rede não saturada

Os resultados deste teste se referem ao tráfego VoIP simulado com a ferramenta RUDE&CRUDE para FreeBSD simulando 4 ligações simultâneas de 80kbits. Esse tráfego foi gerado pelo RUDE na máquina M1 e coletado pelo CRUDE na mesma máquina, informações sobre os scripts utilizados se encontram nos anexos.

O tráfego de fundo TCP gerado pela máquina M3 foi de 200kbit/s através da ferramenta netperf rodando como cliente em M3 e como servidor em M1 foram usados pipes gerados pelo DUMMYNET do FreeBSD para manter a taxa de bits constante. Mais informações nos anexos.

Com a rede não saturada a utilização de QoS não influenciou muito no resultado.

5.2.1 Análise do tempo médio de espera fim-a-fim (delay)

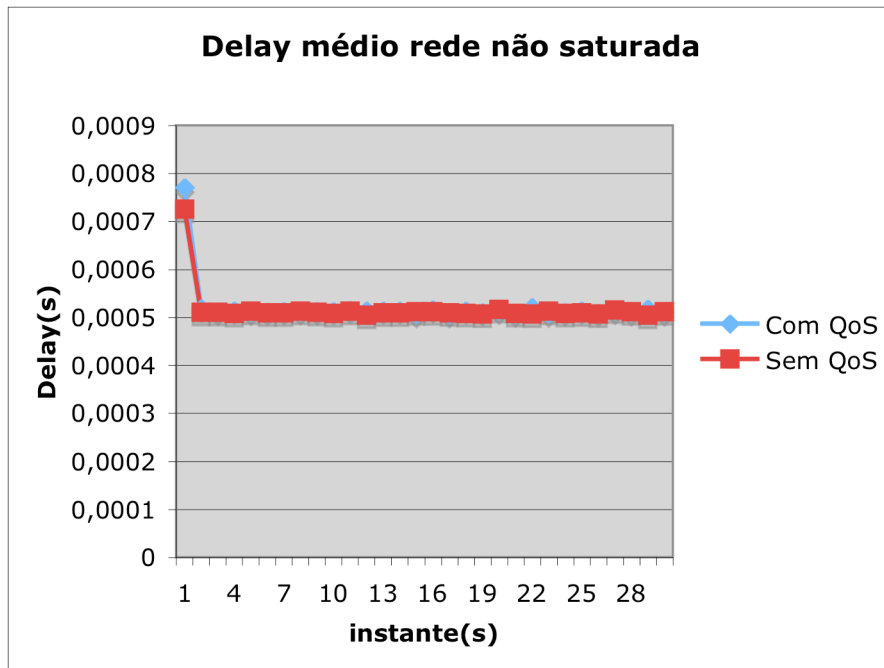


FIG. 5.2 DELAY MÉDIO REDE NÃO SATURADA LINUX

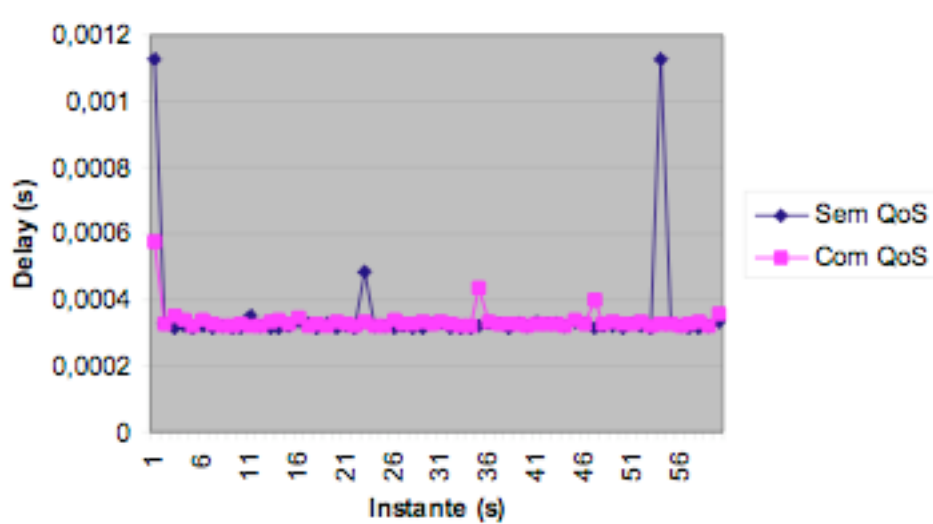


FIG. 5.3 DELAY MÉDIO REDE NÃO SATURADA FREEBSD

Quanto ao delay o FreeBSD mostrou melhores resultados com a rede não saturada, tanto com como sem QoS, mas vale ressaltar que o Linux sem QoS apresentou menores variações.

Linux	QoS (CBQ)	Sem QoS (FIFO)
Máximo	0,516718	0,521277
Mínimo	0,505207	0,507330
Média	0,510447	0,512855

Tabela 5.1 Estatísticas Delay médio rede não saturada

5.2.3 Análise do efeito jitter médio

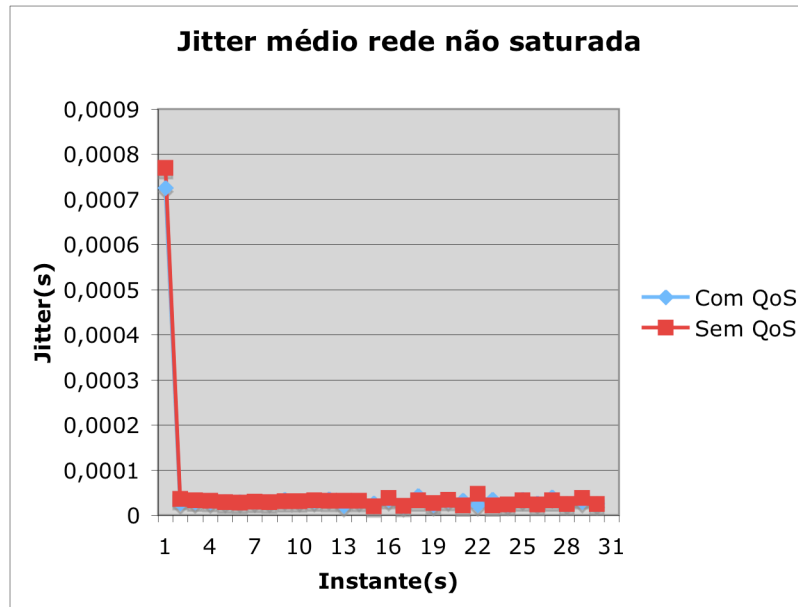


FIG. 5.4 JITTER MÉDIO REDE NÃO SATURADA LINUX

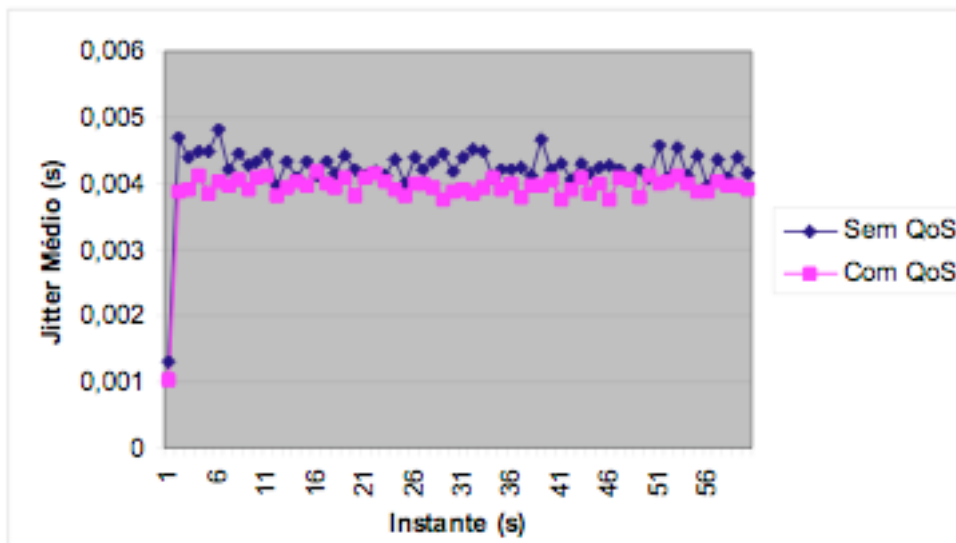


FIG. 5.5 JITTER MÉDIO REDE NÃO SATURADA FREEBSD

Quanto ao jitter com a rede não saturada, os resultados do Linux foram mais satisfatórios tanto quanto à pouca variação quanto ao valor médio, e é importante ressaltar que o jitter é um aspecto muito importante quando se considera aplicações em tempo real como VoIP pois os pacotes devem chegar na mesma ordem em que chegam como já foi dito no capítulo 2.

Linux	QoS (CBQ)	Sem QoS (FIFO)
Max	0,043000ms	0,047000ms
Mínimo	0,021000ms	0,020000ms
Media	0,030137ms	0,030137ms

Tabela 5.2 Jitter médio, estatísticas.

5.2.3 Análise da vazão média

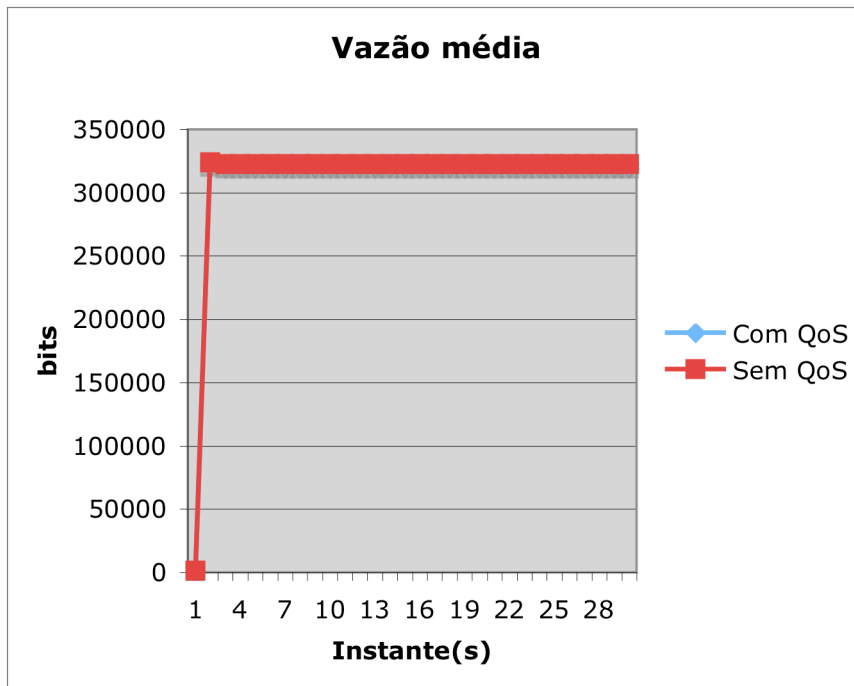


FIG. 5.6 VAZÃO MÉDIA REDE NÃO SATURADA LINUX

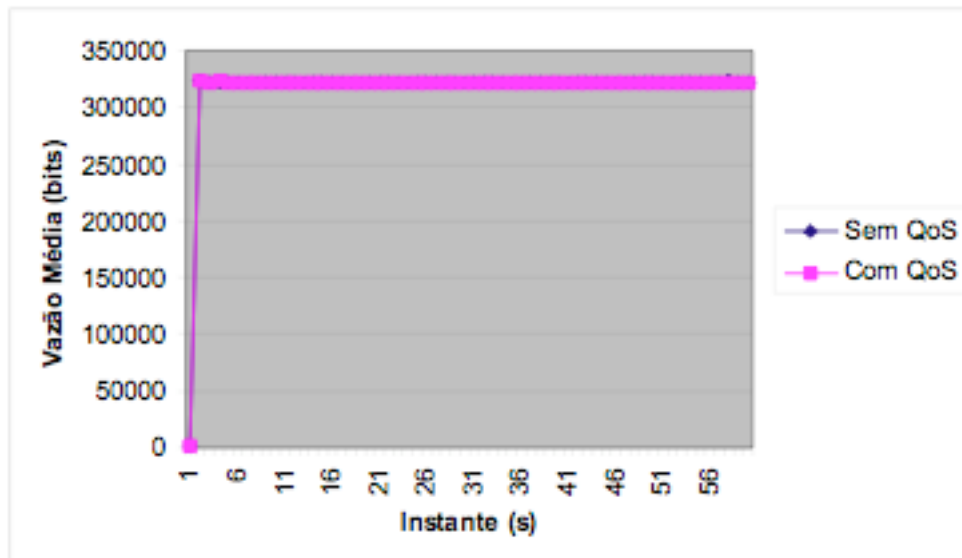


FIG. 5.7 VAZÃO MÉDIA REDE NÃO SATURADA FREEBSD

Aqui os resultados foram bem similares, e idênticos quanto a política de fila adotada. Mantendo sempre a 322560 bps depois de 2 segundos no caso do Linux.

Linux	QoS (CBQ)	Sem QoS (FIFO)
Max	324000	324000
Min	322560	322560
Media	322652,90	322652,90

Tabela 5.3 Vazão média, estatísticas.

5.2.4 Análise da taxa de perdas

Não houve perdas de pacotes com a rede não saturada em ambos os testes.

5.3 Análise dos dados com a rede saturada

Nessa segunda parte do experimento tráfego de fundo TCP gerado pela máquina M3 foi de 800kbit/s através da ferramenta netperf rodando como cliente em M3 e como servidor em

M1. Aqui as diferenças entre o teste com e sem QoS ficaram mais evidentes assim como quanto ao sistema operacional utilizado.

5.3.1 Análise do tempo médio de espera fim-a-fim (delay)

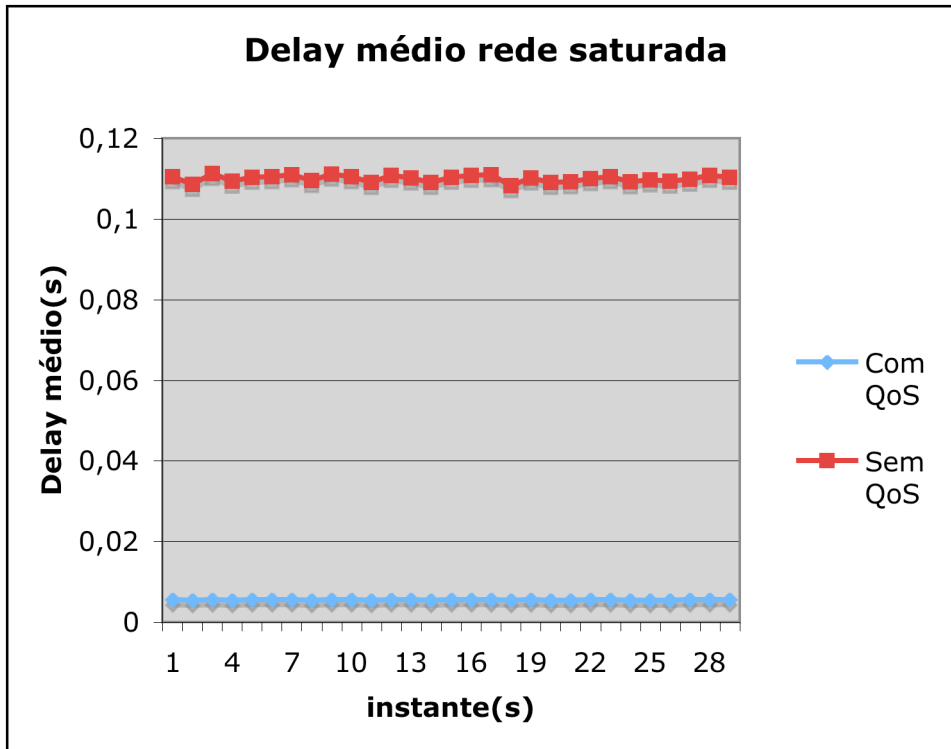


FIG. 5.8 DELAY MÉDIO REDE SATURADA LINUX

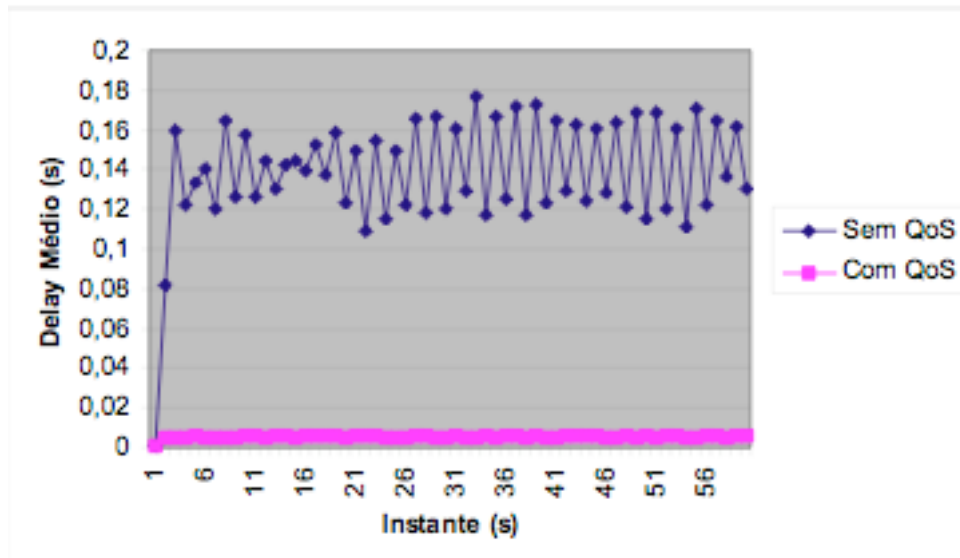


FIG. 5.9 DELAY MÉDIO REDE SATURADA FREEBSD

Analisando os dados com QoS os resultados foram similares com uma média em torno de 5 ms mas no resultado sem QoS o desempenho no Linux foi melhor. Aqui fica evidente que é imprescindível o uso de QoS para garantir baixa latência das aplicações que requerem mais recursos da rede.

Linux	QoS(CBQ)	Sem QoS(FIFO)
Max	5,56473ms	111,294ms
Min	5,41241ms	108,248ms
Média	5,49875ms	109,975ms

Tabela 5.4 Estatísticas Delay médio rede saturada

5.3.2 Análise do jitter médio

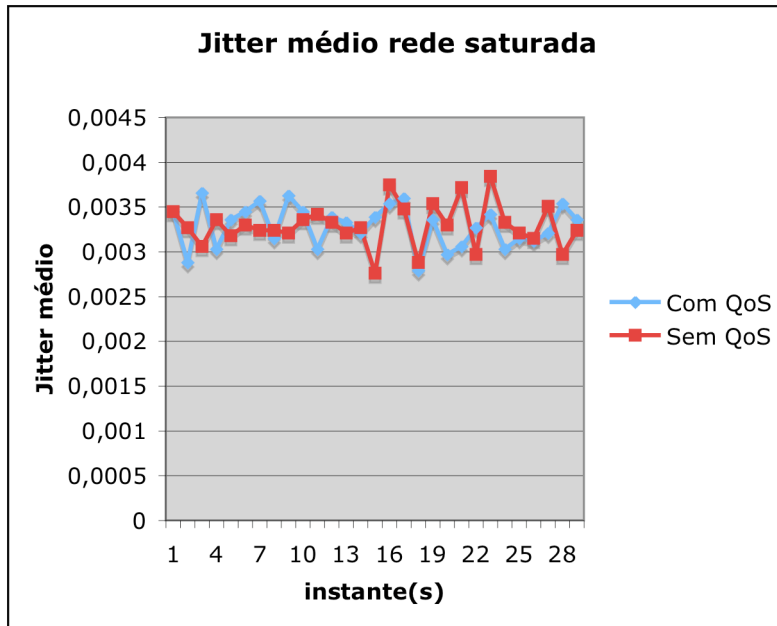


FIG. 5.10 JITTER MÉDIO REDE SATURADA LINUX

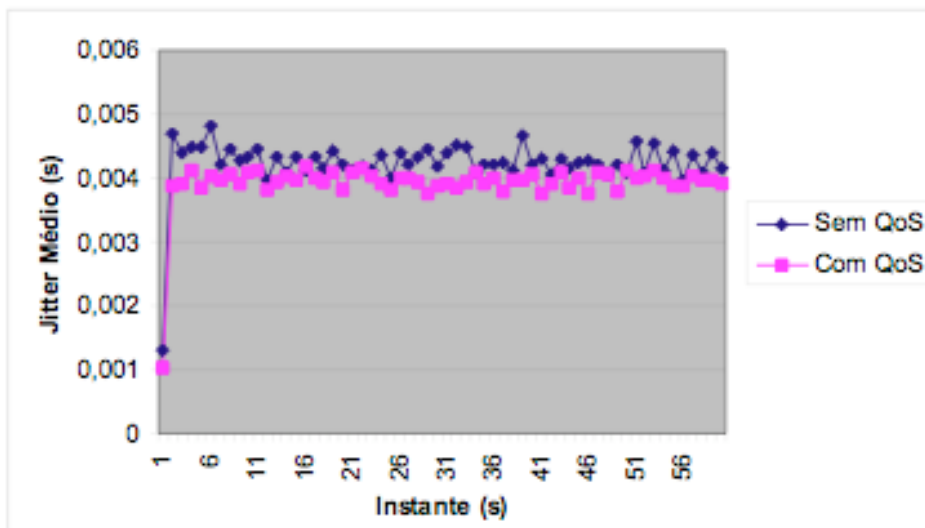


FIG. 5.11 JITTER MÉDIO REDE SATURADA FREEBSD

Aqui podemos notar que o Linux novamente teve a variação do Jitter bem baixa e se saiu melhor neste quesito. O uso de QoS não influenciou muito nesse aspecto.

Linux	QoS(CBQ)	Sem QoS(FIFO)
Max	3,66ms	3,84ms
Min	2,79ms	2,76ms
Média	3,28ms	3,29ms

Tabela 5.5 Estatísticas Jitter médio rede saturada

5.3.3 Análise da vazão média

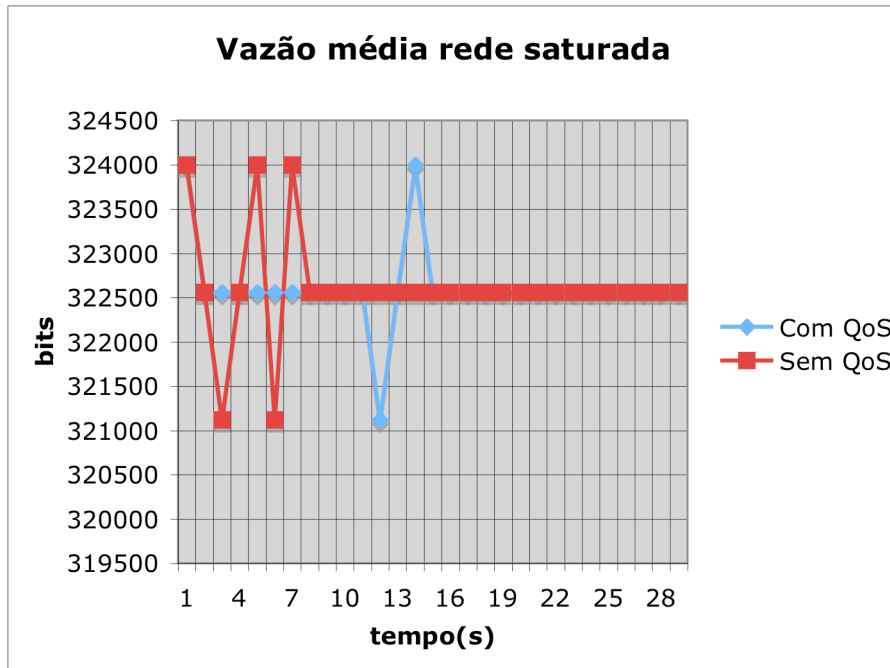


FIG. 5.12 VAZÃO MÉDIA REDE SATURADA LINUX

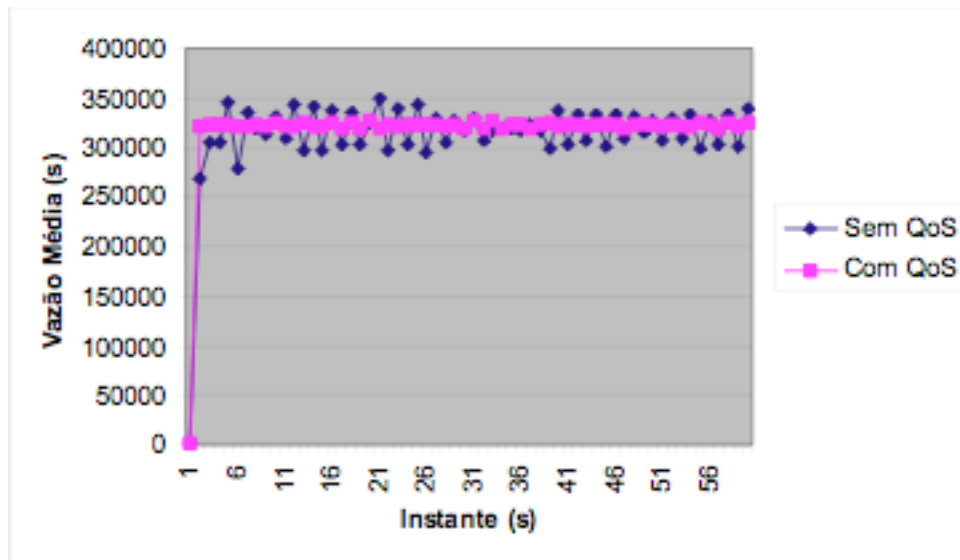


FIG. 5.13 VAZÃO MÉDIA REDE SATURADA FREEBSD

Aqui podemos notar que a vazão oscilou menos no Linux e nunca baixou do nível necessário as aplicações simuladas no teste, que seria 320kbps. E a vazão média foi ligeiramente melhor.

Linux	QoS(CBQ)	Sem QoS(FIFO)
Max	324000	324000
Min	321120	321120
Média	322609,65	322609,65

Tabela 5.6 Vazão média rede saturada, estatísticas.

5.3.4 Análise da taxa de perdas

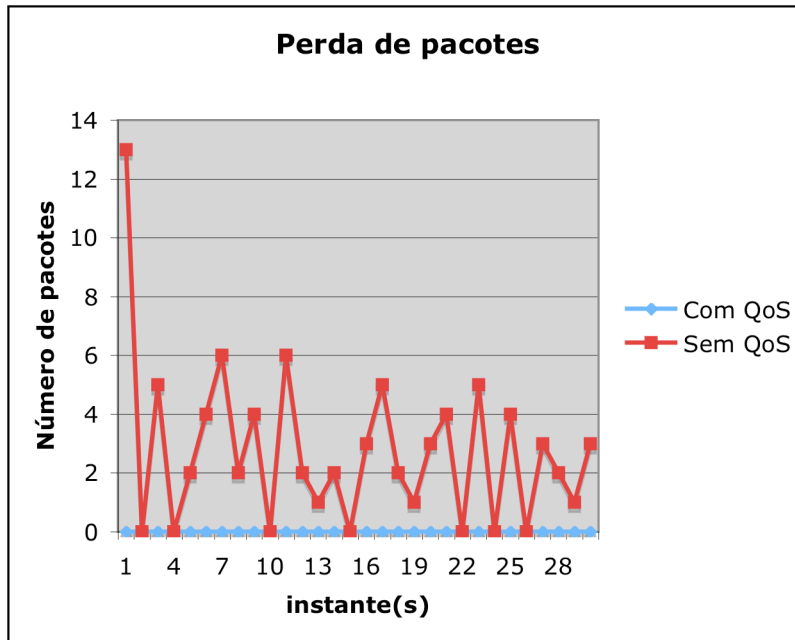


FIG. 5.14 PERDA DE PACOTES REDE SATURADA LINUX

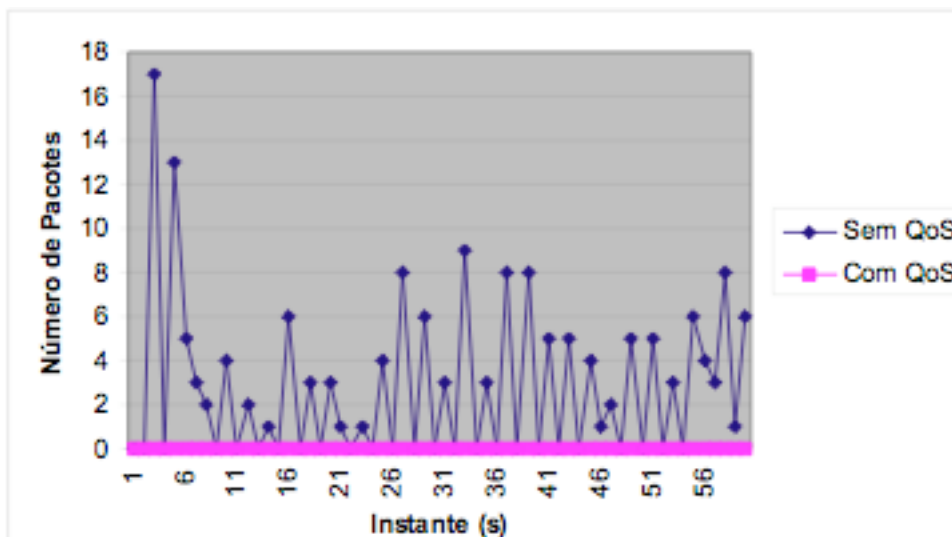


FIG. 5.15 PERDA DE PACOTES REDE SATURADA FREEBSD

Tanto no Linux como no FreeBSD ocorreram algumas perdas de pacotes UDP durante os testes. A perda ocorreu apenas no caso da rede estar saturada e o roteador não efetuando o controle do tráfego (QoS) momento em que os pacotes UDP (referentes ao tráfego VoIP) concorriam, através do mecanismo FIFO, pela banda disponível com os pacotes TCP (referentes às demais aplicações).

Linux	QoS(CBQ)	Sem QoS(FIFO)
Max	0	13
Min	0	0
Média	0	2,413793103

Tabela 5.7 Perda de pacote rede saturada, estatísticas.

6. CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho foi de grande valia, possibilitando um estudo mais aprofundado sobre redes, e principalmente sobre o sistema operacional Linux e sua ferramenta de QoS e controle de tráfego, que se mostrou muito poderosa.

Isso é devido ao fato dela ser integrada ao Kernel, tornando o acesso as suas funções mais rápido e seguro, e as bibliotecas são compiladas de acordo com as características da máquina em questão.

O uso do sistema operacional FreeBSD e suas ferramentas de simulação de tráfego (RUDE&CRUDE, netperf), possibilitou mais um útil aprendizado.

Com base nos resultados obtidos nos experimentos usando FreeBSD e Linux na mesma rede e usando a mesma política de fila, chegamos a conclusão de que o Linux é o sistema operacional mais indicado para realizar roteamento com diferenciação de tráfego numa rede local.

Outro fator a ser destacado é a necessidade de usar QoS para garantir os requerimentos de aplicações que exijam mais recursos da rede como VoIP, videoconferência, streamings de áudio e vídeo em geral. Com isso se pode estabelecer um SLA, de forma a garantir a disponibilidade dos serviços para os clientes e usuários da rede.

Com a realização deste trabalho pode-se afirmar que a prática é fundamental para consolidar os ensinamentos teóricos obtidos ao longo do curso e traz retorno tanto para o aluno quanto para a Universidade na forma de conhecimento.

Como trabalhos futuros ficam as sugestões:

- Usar políticas de filas, classes e de filtros diferentes das utilizadas neste trabalho.
- Realizar uma carga mais pesada de tráfego.
- Utilizar um Ambiente de Alta-Disponibilidade para prover maior confiabilidade ao Sistema.
- Usar uma topologia de rede distinta.

7. REFERÊNCIAS

[And00] Andreozzi, S. Diffserv simulations using the network simulator: requirements, issues and solutions “

[Art] Artola, E. <http://penta.ufrgs.br/Esmilda/arquitet.html>

[Atm] ATM fórum. <http://atmforum.org>

[Bra97] Braden, R., Ed. (1997): Resource Reservation Protocol (RSVP) – Version 1, Functional Specification, Request for Comments 2205. [ftp: //ftp.nic.it/rfc/rfc2205.txt](ftp://ftp.nic.it/rfc/rfc2205.txt)

[Bun04] Bunn, A., (2004), “*Avaliação de Serviço de QoS no FreeBSD*”

[Com95] Douglas E. Comer, “*Internetworking with TCP/IP - Principles, Protocols and Architecture*”, Vol. 1, Prentice-Hall, 1995.

[Dan98] Daniel Minoli and Emma Minoli, “*Delivering Voice over Frame Relay and ATM*”, Wiley Computer Publishing, 1998.

[Dav96] David Ginsburg, “*ATM Solutions for Enterprise Internetworking*”, Addison-Wesley, 1996.

[DiffServ_2474] K. Nichols, S. Blake, F. Baker, D. Black, “*Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*”, RFC 2474, December 1998

[DiffServ_2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, “*An Architecture for Differentiated Services*”, RFC 2475, December 1998

[DiffServ_2597] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, “*Assured Forwarding PHB Group*”, RFC 2597, June 1999

[DiffServ_2598] V. Jacobson, K. Nichols, K. Poduri, “*An Expedited Forwarding PHB*”, RFC 2598, June 1999

[DiffServ_Charter] “IETF *Differentiated Services Working Group*”. Em
<http://www.ietf.org/html.charters/diffser-charter.html> e
<http://www.ietf.org/ids.by.wg.diffserv.html>

[Fer99] Ferguson, P. & Houston, G. (1999): *Quality of Service: Delivering QoS on the Internet and Corporate Networks* – Wiley computer Publishing.

[Gue02]Guerra, André (2002): <http://www.clubedasredes.eti.br/rede0007.htm>

[Hub02] Hubert, Bert(2002): “*Linux Advanced Routing & Traffic Control*” <http://lartc.org>

[Iet_DiffSev] An Informal Management Model for DiffServ Routers
<http://www.ietf.org/internet-drafts/draft-ietf-diffserv-model-06.txt> February 2001
 (proposed standard)

[IntServ_2211] J. Wroclawski, “*Specification of the Controlled-Load Network Element Service*”, RFC 2211, September 1997

[IntServ_2212] S. Shenker, C. Partridge, R. Guerin, “*Specification of Guaranteed Quality of Service*”, RFC 2212, Sept 1997

[IntServ_2215] S. Shenker, J. Wroclawski, “*General Characterization Parameters for Integrated Service Network Elements*”, RFC 2215, September 1997

[IntServ_2216] S. Shenker, J. Wroclawski, “*Network Element Service Specification Template*”, RFC 2216, Sept 1997

[IntServ_Charter] “IETF *Integrated Services Working Group*”. Em <http://www.ietf.org/html.charters/intserv-charter.html> e <http://www.ietf.org/ids.by.wg/intserv.html>

[ISSLL_Charter] “*Integrated Services over Specific Link Layers*”, Em <http://www.ietf.org/html.charters/issll-charter.html>

[Jam98] James D. McCabe, “*Practical Computer Network Analysis and Design*”, Morgan Kaufmann Series in Networking, 1998.

[Job99] Joberto Martins, “Redes Corporativas MultiServiço - Caracterização das Aplicações e Parâmetros Básicos de Operação”, Em <http://www.jsmnet.com/slides/AnaliseRequisitos/index.htm>

[JSMNet] “JSMNet - Estado da Arte e P&D em Redes de Computadores”. Em <http://www.jsmnet.com>

[Kil99] Kilkki, Kalevi *Differentiated Services for the Internet* Macmillan Technology Series, 1999

[Mat97] Mathias Hein and David Griffiths, “*Switching Technology in the Local Network - From LAN to Switched LAN to Virtual LAN*”, Thomson Computer Press, 1997.

[Mau97] Thomas A. Maufer, “*Deploying IP Multicast in the Enterprise*”, Prentice-Hall, 1997.

[Mel01] Melo, Edson (2001): “Qualidade de Serviço em Redes IP com DiffServ: Avaliação através de Medições”.

[Mpls_Charter] IETF “*Multiprotocol Label Switching*” Working Group. Em <http://www.ietf.org/html.charters/mpls-charter.html> e <http://www.ietf.org/ids.by.wg/mpls.html>

[Opa] Leonardo Balliache(2003), “*Differentiated Service on Linux HOWTO*”
<http://www.opalsoft.net/qos/DS.htm>

[Qos] <http://qos.ittc.ukans.edu/howto/node3.html>

[Rnp] <http://www.rnp.br/qos/qos-sobre.html>

[Rsvp_2205] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, “*Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*”, RFC 2205, September 1997

[Sha75] Shannon, Robert E. *System Simulation: The Art and Science* Prentice-Hall 1975

[Sis04] Sisterolli, Marcelo C., “Desenvolvimento De Controle De Admissão Em Modelo Mrsvp/Intserv”

[Ste94] W. Richard Stevens, “*TCP/IP Illustrated - The Protocols*”, Vol. 1, Addison-Wesley, 1994.

[Str03] Strelow, Fábio (2003): “Análise dos Problemas de Provisão de QoS em Redes Sem Fio Utilizando Simulação”.

[Tan96] Andrew Tanenbaum, “*Computer Networks*”, 3rd edition, Prentice-Hall, 1996.

[Tho96] Stephen A. Thomas, “*IPng and the TCP/IP Protocols - Implementing the Next Generation Internet*”, Wiley Computer Publishing, 1996.

ANEXO 1 - Roteiro de Instalação e configuração do Rude & Crude

1 – Atualização do Perl

```
cd /usr/ports/lang/perl5.8
```

```
make
```

```
make install
```

```
use.perl port
```

2 – Instalação do Rude/Crude

```
cd /usr/ports/net/rude
```

```
make
```

```
make install
```

Observação: estes procedimentos foram realizados após a atualização da árvore do ports com o seguinte comando:

```
cvsup -g -L 2 /etc/cvsupfile
```

O arquivo /etc/cvsupfile tem o seguinte conteúdo:

```
*default host=cvsup2.br.FreeBSD.org
```

```
*default base=/usr
```

```
*default prefix=/usr
```

```
*default release=cvs tag=RELENG_5
```

```
*default tag=.
```

```
*default delete use-rel-suffix
```

```
*default compress ports-all
```

ANEXO 2 – Regras usadas para simular o tráfego TCP constante

Regras utilizadas para simular o tráfego constante a 200Kbps entre M3 e M1, estas regras foram colocadas em M3

```
#ipfw add 2 pipe 1 all from any to 192.168.10.2 out via fxp0
#ipfw pipe 1 config bw 200kbit/s
#ipfw add 3 pipe 2 all from 192.168.10.2 to any in via fxp0
#ipfw pipe 2 config bw 200kbit/s
```

Regras utilizadas para simular o tráfego constante a 800Kbps entre M3 e M1, estas regras foram colocadas em M3

```
#ipfw add 2 pipe 1 all from any to 192.168.10.2 out via fxp0
#ipfw pipe 1 config bw 800kbit/s
#ipfw add 3 pipe 2 all from 192.168.1.2 to any in via fxp0
#ipfw pipe 2 config bw 800kbit/s
```

- Sendo que fxp0 é a interface de rede pertencente a M3 e ao enlace L3 e 192.168.1.2 é IP de M1 pertencente ao enlace L2 (vide Figura 5.1).

ANEXO 3 – Script para a decodificação do tráfego recebido

```
#!/usr/local/bin/bash
# decodifica os logs no diretorio atual, com a extensao .log, e os processa.
FILES=`ls -l $1`
PPATH=$PWD
CRUDE="/home/bunn/experimentacao/crude"
SPATH="/home/bunn/experimentacao/scripts"
DELAY="$SPATH/delay"
DELAYM="$SPATH/delay-medio"
JITTER="$SPATH/jitter"
JITTERM="$SPATH/jitter-medio"
PERDAS="$SPATH/perdas"
PERDAST="$SPATH/perdas-tot"
VAZAO="$SPATH/vazao"
mkdir delay jitter vazao perdas
for i in $FILES;
do {
$CRUDE -d "$1/$i" > "$SPATH/$i.tmp"
cat "$i.tmp" | sed "1 D" > "$i.log" } done
rm -f "$SPATH/*.tmp" FILES=`ls -l *.log`
for i in $FILES; do {
cd "$SPATH/vazao"
$VAZAO "$SPATH/$i" > "v-$i"
cd "$SPATH/delay"
$DELAY "$SPATH/$i" > "d-$i"
$DELAYM "$SPATH/$i" > "dm-$i"
cd "$SPATH/jitter"
$JITTER "$SPATH/$i" > "j-$i"
$JITTERM "$SPATH/$i" > "jm-$i"
cd "$SPATH/perdas"
$PERDAS "$SPATH/$i" > "p-$i"
$PERDAST "../$i" >> "p-tot" } done
```


ANEXO 5 – Sintaxe dos programas utilizados

1 – Crude

```
#crude -p 5001 -l arquivo.txt
```

2 – Rude

```
#rude -s rude.conf
```

Onde o conteúdo do arquivo rude.conf é:

```
START NOW
```

```
#Simulando 4 ligações simultâneas
```

```
1000 0010 ON 3001 192.168.10.2:5001 CONSTANT 224 180 31000 0010 OFF
```

Obs.: Os programas devem rodar ao mesmo tempo, através do uso de terminais diferentes.

3 – Netperf – server em M1

```
#netserver
```

4 – Netperf – para gerar tráfego em M3

```
#netperf -H 192.168.10.2 -l 120
```

ANEXO 6 – ARTIGO

Estudo comparativo entre roteadores Linux e FreeBSD para diferenciação de tráfego

Roberto Willrich¹, Mário A. R. Dantas¹, João Bosco M. Sobral¹, João G. S. Chiste¹
¹INE – Departamento de Informática e Estatística da UFSC
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil
{willrich, mario, bosco, [jg](mailto:jg@inf.ufsc.br)}@inf.ufsc.br

RESUMO: O modelo de controle de tráfego na Internet atualmente é baseado no melhor esforço, isso quer dizer que os pacotes são distribuídos na ordem em que chegam. Este trabalho faz um estudo sobre o uso de Qualidade de Serviço como ferramenta para melhorar o controle de tráfego através da diferenciação de serviços, priorizando as aplicações que demandam mais recursos da rede, como VoIP (voz sobre IP) e videoconferência, que são aplicações em tempo real, de forma a garantir um nível mínimo de qualidade pré-estabelecido no SLA (Acordo de nível de Serviço). Neste trabalho foi montada uma rede com as mesmas características da rede local de uma empresa, e utilizando o sistema operacional Linux como roteador e duas máquinas rodando FreeBSD para gerar e receber o tráfego da rede. Foram gerados os tráfegos TCP (simulando mail, web, ftp) e UDP (simulando voip) e através da ferramenta tc, de controle de tráfego do Linux, foram usadas políticas de fila de QoS CBQ para priorizar o tráfego UDP. Os resultados puderam ser avaliados através de algumas métricas de desempenho (delay, jitter, vazão e perda de pacotes) e comparados com os resultados obtidos na mesma rede e com a mesma carga no trabalho feito usando um roteador FreeBSD por Alexandre Bunn. Com os resultados obtidos pode-se concluir que o Linux é uma ferramenta adequada para fazer roteamento com diferenciação de tráfego.

INTRODUÇÃO

1. Considerações iniciais

Normalmente a Internet trabalha com a filosofia do melhor esforço: cada usuário compartilha largura de banda com outros e, portanto, a transmissão de seus dados concorre com as transmissões dos demais usuários. Os dados empacotados são encaminhados da melhor forma possível, conforme as rotas e banda disponíveis. Quando há congestionamento, os pacotes são descartados sem distinção. Não há garantia de que o serviço será realizado com sucesso. Entretanto, aplicações como voz sobre IP e videoconferência necessitam de tais garantias.

Com a implantação de qualidade de serviço (*Quality of Service – QoS*), é possível oferecer maior garantia e segurança para aplicações avançadas, uma vez que o tráfego destas aplicações passa a ter prioridade em relação a aplicações tradicionais.[Rnp].

Com uso de QoS os pacotes são marcados para distinguir os tipos de serviços e os roteadores são configurados para criar filas distintas para cada aplicação, de acordo com as prioridades das mesmas. Assim, uma faixa da largura de banda, dentro do canal de comunicação, é reservada para que, no caso de congestionamento, determinados tipos de fluxos de dados ou aplicações tenham prioridade na entrega.[Rnp]

1.1 Objetivos do trabalho

Avaliar e estudar o sistema operacional Linux como plataforma de gerenciamento de qualidade de serviço (QoS), em especial suas potencialidades de diferenciação de tráfego. Realizar um testbed composto de duas máquinas FreeBSD e um roteador Linux. Sendo que o roteador estará configurado para realizar diferenciação de tráfego através de QoS. Comparar os resultados dos testes com os resultados obtidos utilizando o FreeBSD como roteador.

2. ARQUITETURA TCP/IP

A arquitetura básica do TCP/IP é mostrada no figura abaixo, que fazendo relação com o modelo de referência OSI correspondem as camadas da Rede, Transporte e Aplicações. Como pode-se observar, na camada de rede TCP/IP tem-se o protocolo IP(Internet Protocol) ; na camada de transporte estão dois protocolos, um que oferece serviços sem

conexão, que é o protocolo UDP (User Datagram Protocol) e um outro que oferece serviços orientados a conexão, protocolo TCP (Transport Control Protocol). Na camada de aplicações o TCP/IP tem uma variedade de protocolos de aplicação, como SMTP, TELNET, FTP e NSF entre outros.

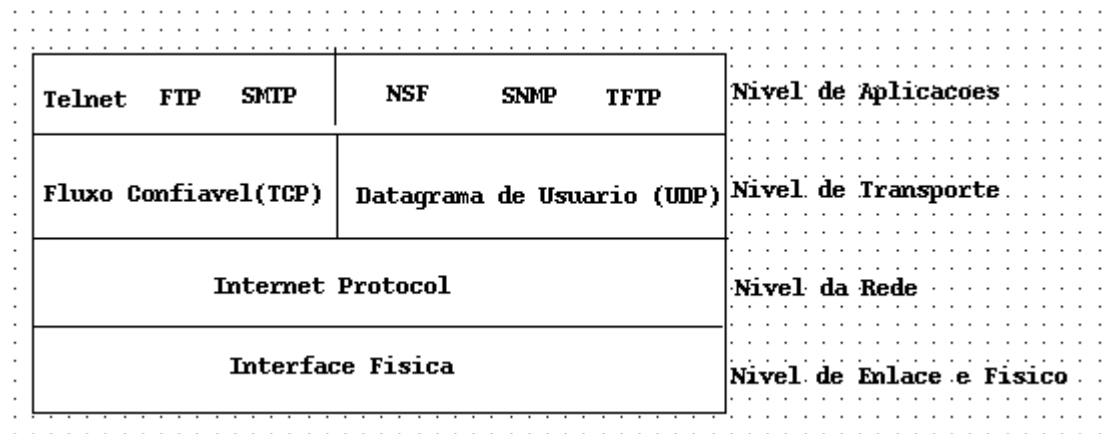


FIG. 2-1 - ARQUITETURA TCP/IP [ART]

2.1 Características

- Padrão de protocolos aberto, não associado a nenhum tipo específico de plataforma de hardware, sistema operacional, ou hardware específico para acesso ao meio físico de transmissão (TCP/IP funciona sobre Ethernet, Token-ring, linha discada, X.25, e qualquer outro tipo de meio de transmissão).
- Esquema de endereçamento comum que permite a identificação única de um elemento da rede (na rede local, ou no planeta).
- Protocolos de alto nível padronizados para disponibilização universal e consistente de serviços aos usuários.
- Documentação ampla acessível na própria rede sob a forma de “Request for Comments” – RFC’s que não sofrem do rigor imposto aos relatórios técnicos formais. As RFC’s contêm as últimas versões das especificações de todos os protocolos TCP/IP padrões.
- Interconexão cooperativa de redes, suportando serviços de comunicação universal (usada em uma rede local ou em uma rede [inter] planetária).
- Utilização de tecnologia adequada às necessidades locais em cada rede.
- Independência de hardware e sistemas operacionais.

- Interconexão de redes se dá por meio de roteadores.

2.2.1 Algoritmos de Roteamento

Roteamento é o mecanismo pelo qual se escolhe o caminho (canal de comunicação) que um pacote deve seguir para atingir seu destino. Em redes sem conexão (datagrama), cada datagrama tem de carregar seu endereço destino e a decisão de roteamento é tomada em cada nó da rede, para cada datagrama que chega.. Em redes com conexão (circuito virtual) os pacotes não precisam carregar o endereço destino (só a identificação da conexão) e a decisão de roteamento é tomada no estabelecimento da conexão, após a qual todo pacote segue sempre o mesmo caminho (roteamento por sessão).

2.2.2 Controle de Congestionamento

Congestionamento ocorre quando a quantidade de pacotes na rede é muito grande (normalmente isso ocorre quando se atinge um patamar da capacidade de carga dos canais de comunicação).

Fatores que ocasionam congestionamento:

Pacotes chegando por canais de comunicação rápidos, tendo de sair por canais mais lentos. Roteadores lentos ou com pouca memória para armazenar pacotes temporariamente (memória infinita estraga tudo [Tan,96]).

3. QUALIDADE DE SERVIÇO (QoS)

A qualidade de serviço (QoS) nas redes IP é um aspecto operacional fundamental para o desempenho fim-a-fim das novas aplicações (VoIP, multimídia, ...). Assim sendo, é importante o entendimento dos seus princípios, parâmetros, mecanismos, algoritmos e protocolos desenvolvidos e utilizados para a obtenção de uma QoS.

A obtenção de uma QoS adequada é um requisito de operação da rede e suas componentes para viabilizar a operação com qualidade de uma aplicação.

3.1 QoS como Mecanismo Gerencial

Do ponto de vista de um gerente ou administrador de redes, a percepção da qualidade de serviço é mais orientada no sentido da utilização de mecanismos, algoritmos e protocolos de QoS em benefício de seus clientes e suporte às aplicações. Ou seja, como efetivamente a rede e suas componentes podem garantir as inúmeras SLAs definidas para diversos usuários e aplicações.

3.1.1. Controle de Tráfego

Alguns aspectos fundamentais ao estudo de Controle de Tráfego são necessários quando falamos de performance da rede e atendimento aos requisitos de QoS. Desta forma é possível determinar, para cada tipo de aplicação típica, como voz, vídeo, dados, etc., qual a melhor técnica, ou conjunto de técnicas, a ser aplicada para atendê-la, uma vez que existe um forte relacionamento entre a aplicação utilizada e a categoria de serviço. A essência do Controle de Tráfego é determinar quando uma nova conexão pode ser aceita ou não. Para isto a rede deve avaliar se uma determinada conexão, com um determinado perfil de QoS, pode ser suportado pela rede com os recursos disponíveis, sem prejuízo para as conexões atualmente ativas. Uma vez aceita a conexão, rede e usuário estabelecem um Contrato de Tráfego, onde a rede concorda em suportá-la segundo um conjunto de parâmetros de QoS negociados e o usuário se compromete em não exceder os limites dos parâmetros de tráfego. As funções do Controle de Tráfego estão diretamente relacionadas com esta negociação e o policiamento de QoS pela rede. Seu principal objetivo é evitar o congestionamento, mas, se isso ocorrer, também poderá atuar na recuperação da rede desta situação.

3.2 QoS -Parâmetros

Como definido anteriormente, a QoS necessária às aplicações é definida em termos de uma SLA. Na especificação das SLAs são definidos os parâmetros de qualidade de serviço e alguns dos mais comumente utilizados são:

- Vazão (Banda)
- Atraso (Latência)
- *Jitter*
- Taxa de Perdas, Taxa de Erros, etc...
- Disponibilidade
- Outros

3.2.1 Vazão

A vazão (banda) é o parâmetro mais básico de QoS e é necessário para a operação adequada de qualquer aplicação.

3.2.2 Latência (Atraso)

A latência e o atraso são parâmetros importantes para a qualidade de serviço das aplicações. Ambos os termos podem ser utilizados na especificação de QoS, embora o termo “latência” seja convencionalmente mais utilizado para equipamentos e o termo “atraso” seja mais utilizado com as transmissões de dados (P. ex.: atrasos de transmissão, atrasos de propagação, ...).

3.2.3 Jitter

O *jitter* é um outro parâmetro importante para a qualidade de serviço. No caso, o *jitter* é importante para as aplicações executando em rede cuja operação adequada depende de alguma forma da garantia de que as informações (pacotes) devem ser processadas em períodos de tempo bem definidos. Este é o caso, por exemplo, de aplicações de voz e fax sobre IP (VoIP), aplicações de tempo real, etc...

Do ponto de vista de uma rede de computador, o *jitter* pode ser entendido como a variação no tempo e na seqüência de entrega das informações (p.ex.: pacotes) (*Packet-Delay Variation*) devido à variação na latência (atrasos) da rede.

3.2.4 Perdas

As perdas de pacotes em redes IP ocorrem principalmente em função de fatores tais como:

- Descarte de pacotes nos roteadores e *switch routers* (Erros, congestionamento) e
- Perda de pacotes devido à erros ocorridos na camada 2 (PPP - *Point-to-Point Protocol*, ethernet, *frame relay*, ATM, ...) durante o transporte dos mesmos.

4. QoS NO LINUX

Este capítulo comenta o suporte de QoS existente nos kernels de linux mais recentes. O suporte de QoS no *kernel* provê a ferramenta para a implementação de várias tecnologias IP QoS como serviços integrados e serviços diferenciados. Aqui serão discutidos os detalhes de configuração, implementação e uso do suporte de QoS no linux.

4.1 Configuração

O suporte à qualidade de serviço está disponível desde as versões 2.1.90 do kernel. Entretanto, o suporte é mais compreensivo nos kernels mais recentes. Este documento foi escrito com referência à versão 2.6.10.8 do *kernel*. A partir do kernel 2.4 os serviços diferenciados foram integrados ao kernel, somente sendo necessário alterar a configuração padrão. Os últimos kernels do linux podem ser obtidos de <http://www.kernelnotes.org/>.

Para habilitar DiffServ siga os seguintes passos:

1. Faça um 'make xconfig' ou 'make menuconfig' ou 'make config' no diretório /usr/src/linux.
2. Em opções de rede, ponha 'y' para as seguintes opções de *kernel*: *Kernel/User netlink socket*, *Routing messages*, *TCP/IP networking* e *QoS and/or fair queueing*. Ligando a opção *QoS and/or fair queueing*, habilita o CBQ, CSZ, PRIO, RED, SFQ, TEQL, TBF, GRED, DS_MARK, classificador *tcindex*, Classificador de pacotes API, classificador U32 e classificador baseado em tabela de roteamento.
3. Faça um 'make dep; make clean; make bzilo'
4. Reinicie o linux usando a nova imagem do *kernel*.

Tendo discutido a configuração do suporte de QoS no linux, Serão visto os detalhes envolvidos na implementação dessas características. Todos os arquivos de kernel relacionados no documento estão localizados no diretório /usr/src/linux.

Para mais informações sobre parâmetros de QoS no kernel consultar o documento /usr/src/linux/Documentation/networking/ip-sysctl.txt.

5. COMPARAÇÃO DOS TESTES LINUX x FREEBSD

Aqui são mostrados os resultados dos testes feitos na rede. No primeiro caso, com a rede utilizando QoS, e no segundo caso sem o QoS.

Para isso, o ambiente de testes foi usado em dois momentos distintos: primeiro momento utilizando o roteamento IP clássico; e em um segundo momento usando roteamento com QoS, ou seja, o roteador irá controlar o fluxo de pacotes no enlace L2 priorizando os pacotes de aplicações VoIP e controlando o fluxo das demais aplicações de forma a não interferirem no uso das aplicações de VoIP mais sensíveis a variações de tráfego na rede. Nestes dois momentos, serão realizados 2 testes, o primeiro enlace não-saturado e o outro com o enlace saturado, visando medir o desempenho e os resultados são confrontados. A duração de cada teste foi de 30s ao contrário do FreeBSD em que os testes duraram 60s. Este valor de 30 segundos foi escolhido por ser um valor suficiente para análise e por facilitar a visualização dos gráficos. Esta análise visa comparar, em situações de tráfego igual e trafegando pelos mesmos nós com as mesmas condições, é revelar se um Roteador Linux é adequado para efetuar o do Controle de Banda de forma a priorizar o tráfego VoIP de uma empresa.

Os gráficos apresentados seguem a ordem Linux depois FreeBSD.

5.1 Definição da Rede

O primeiro passo para a realização da avaliação de desempenho pretendida foi definir a topologia de rede mais adequada ao nosso propósito. E depois a simulação de um link de 1Mbps (Enlace L2 conforme podemos observar na Figura 5.1) alcançado através da uso da disciplina de fila associada as placas de rede do roteador Linux, primeiramente controlando o tráfego e utilizando o mecanismo de enfileiramento de pacotes FIFO e no segundo teste além de controlar o tráfego utilizou-se o mecanismo de enfileiramento CBQ, devido à

possibilidade da criação de regras que otimizem o uso da banda ao máximo. A topologia criada está disposta conforme a figura 5.1. O ambiente de testes definido tem 1 roteador e 2 computadores sendo um deles somente para gerar e o outro para gerar e receber tráfego, com exceção do roteador, os outros computadores rodam o Sistema Operacional FreeBSD.

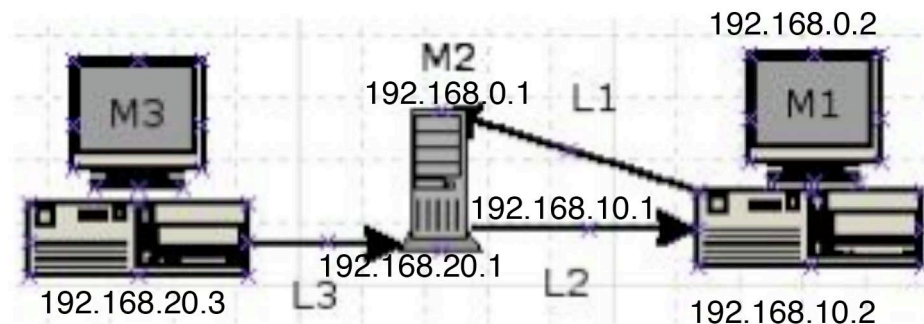


FIG 5.1 CONFIGURAÇÃO DA REDE

5.2 Resultado dos testes da rede não saturada

Os resultados deste teste se referem ao tráfego VoIP simulado com a ferramenta RUDE&CRUDE para FreeBSD simulando 4 ligações simultâneas de 80kbits. Esse tráfego foi gerado pelo RUDE na máquina M1 e coletado pelo CRUDE na mesma máquina, informações sobre os scripts utilizados se encontram nos anexos.

O tráfego de fundo TCP gerado pela máquina M3 foi de 200kbit/s através da ferramenta netperf rodando como cliente em M3 e como servidor em M1 foram usados pipes gerados pelo DUMMYNET do FreeBSD para manter a taxa de bits constante. Mais informações nos anexos.

Com a rede não saturada a utilização de QoS não influenciou muito no resultado.

5.2.1 Análise do tempo médio de espera fim-a-fim (delay)

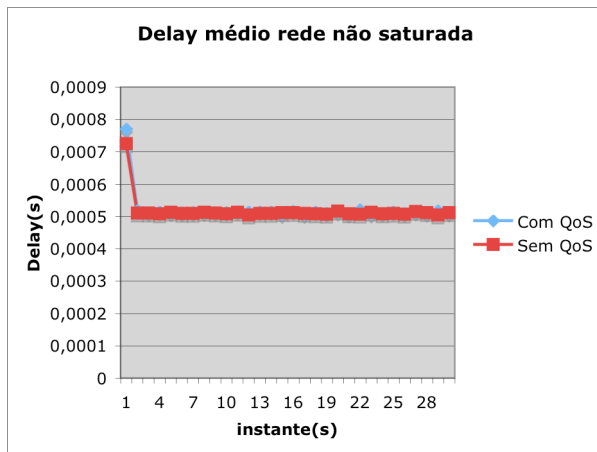


FIG. 5.2 DELAY MÉDIO REDE NÃO SATURADA LINUX

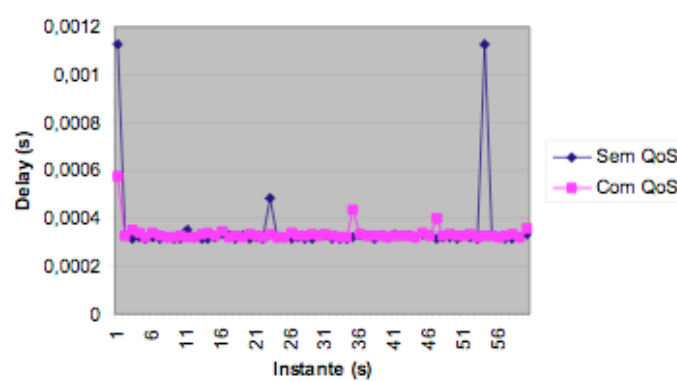


FIG. 5.3 DELAY MÉDIO REDE NÃO SATURADA FREEBSD

Quanto ao delay o FreeBSD mostrou melhores resultados com a rede não saturada, tanto com como sem QoS, mas vale ressaltar que o Linux sem QoS apresentou menores variações.

Linux	QoS (CBQ)	Sem QoS (FIFO)
Máximo	0,516718	0,521277
Mínimo	0,505207	0,507330
Média	0,510447	0,512855

Tabela 5.1 Estatísticas Delay médio rede não saturada

5.2.3 Análise do efeito jitter médio

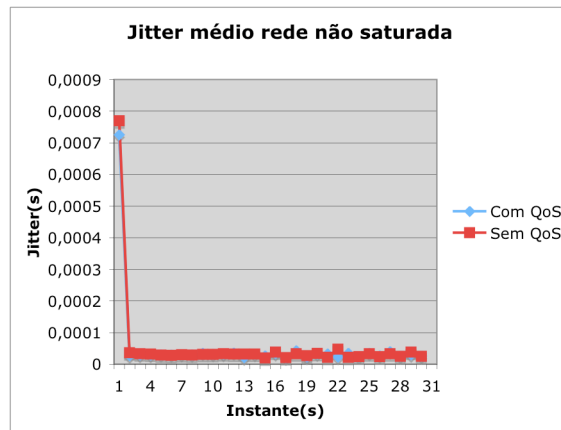


FIG. 5.4 JITTER MÉDIO REDE NÃO SATURADA LINUX

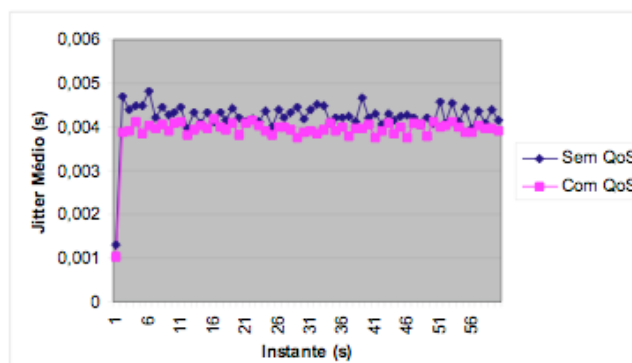


FIG. 5.5 JITTER MÉDIO REDE NÃO SATURADA FREEBSD

Quanto ao jitter com a rede não saturada, os resultados do Linux foram mais satisfatórios tanto quanto à pouca variação quanto ao valor médio, e é importante ressaltar que o jitter é um aspecto muito importante quando se considera aplicações em tempo real como VoIP pois os pacotes devem chegar na mesma ordem em que chegam como já foi dito no capítulo 2.

Linux	QoS (CBQ)	Sem QoS (FIFO)
Max	0,043000ms	0,047000ms
Mínimo	0,021000ms	0,020000ms
Media	0,030137ms	0,030137ms

Tabela 5.2 Jitter médio, estatísticas.

5.2.3 Análise da vazão média

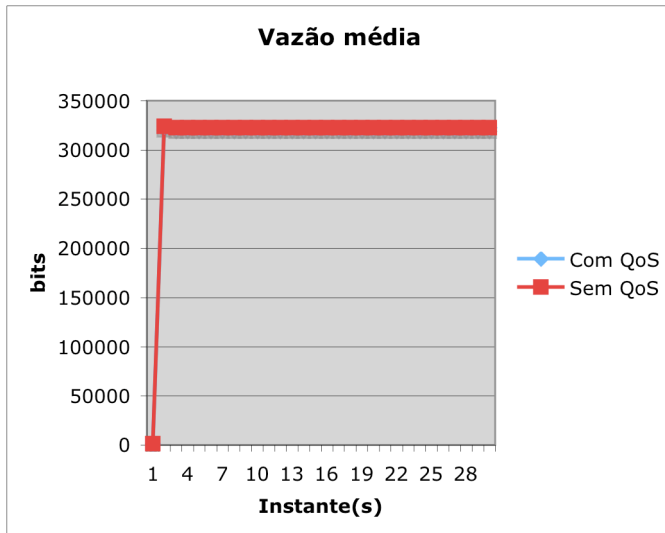


FIG. 5.6 VAZÃO MÉDIA REDE NÃO SATURADA LINUX

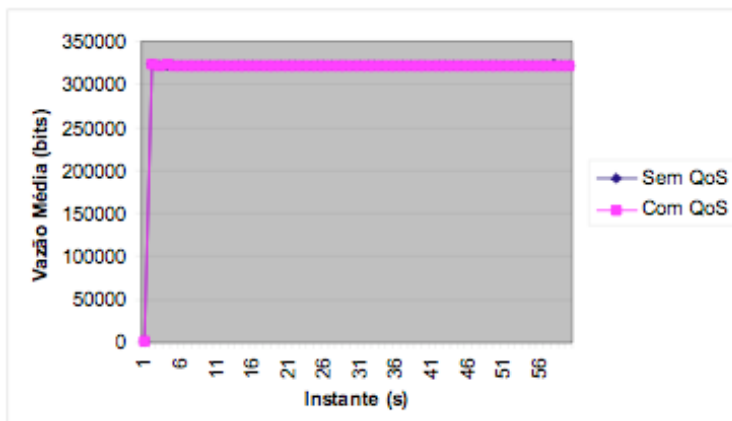


FIG. 5.7 VAZÃO MÉDIA REDE NÃO SATURADA FREEBSD

Aqui os resultados foram bem similares, e idênticos quanto a política de fila adotada. Mantendo sempre a 322560 bps depois de 2 segundos no caso do Linux.

5.2.4 Análise da taxa de perdas

Não houve perdas de pacotes com a rede não saturada em ambos os testes.

5.3 Análise dos dados com a rede saturada

Nessa segunda parte do experimento tráfego de fundo TCP gerado pela máquina M3 foi de 800kbit/s através da ferramenta netperf rodando como cliente em M3 e como servidor em M1. Aqui as diferenças entre o teste com e sem QoS ficaram mais evidentes assim como quanto ao sistema operacional utilizado.

5.3.1 Análise do tempo médio de espera fim-a-fim (delay)

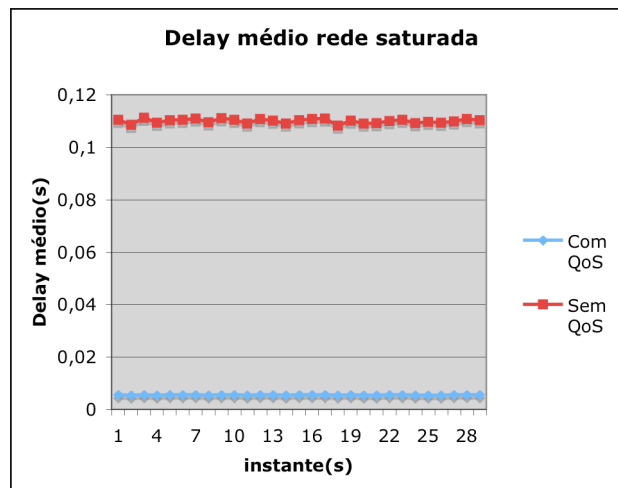


FIG. 5.8 DELAY MÉDIO REDE SATURADA LINUX

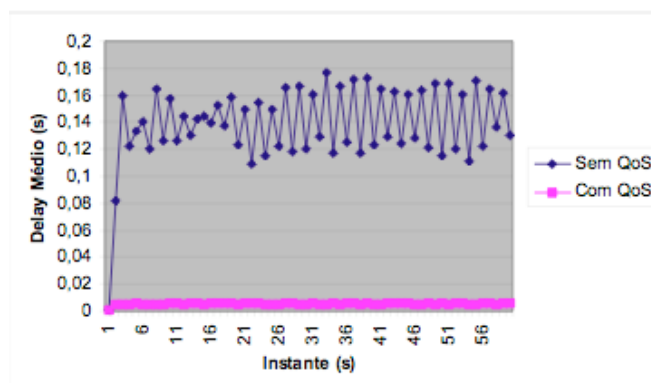


FIG. 5.9 DELAY MÉDIO REDE SATURADA FREEBSD

Analisando os dados com QoS os resultados foram similares com uma média em torno de 5 ms mas no resultado sem QoS o desempenho no Linux foi melhor. Aqui fica evidente que é imprescindível o uso de QoS para garantir baixa latência das aplicações que requerem mais recursos da rede.

Linux	QoS(CBQ)	Sem QoS(FIFO)
Max	5,56473ms	111,294ms
Min	5,41241ms	108,248ms
Média	5,49875ms	109,975ms

Tabela 5.3 Estatísticas Delay médio rede saturada

5.3.2 Análise do jitter médio

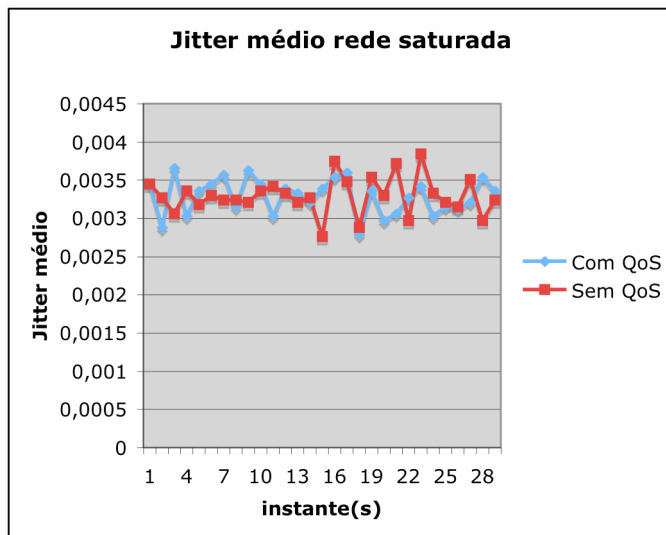


FIG. 5.10 JITTER MÉDIO REDE SATURADA LINUX

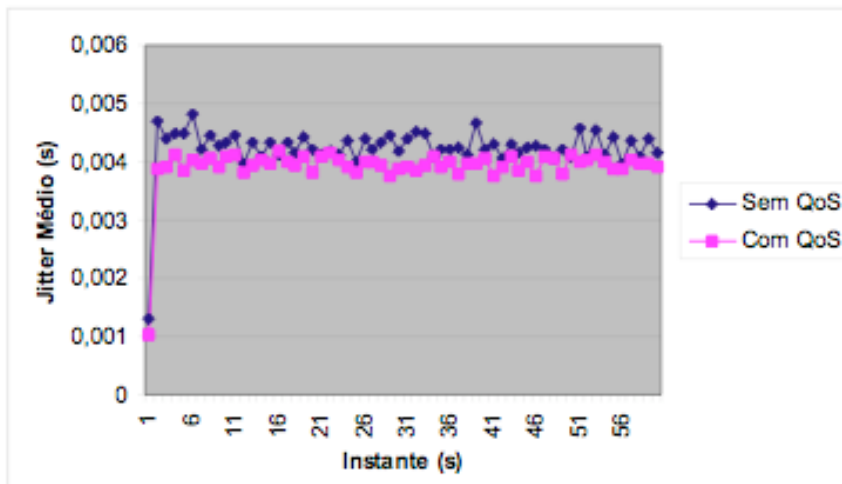


FIG. 5.11 JITTER MÉDIO REDE SATURADA FREEBSD

Aqui podemos notar que o Linux novamente teve a variação do Jitter bem baixa e se saiu melhor neste quesito. O uso de QoS não influenciou muito nesse aspecto.

Linux	QoS(CBQ)	Sem QoS(FIFO)
Max	3,66ms	3,84ms
Min	2,79ms	2,76ms
Média	3,28ms	3,29ms

Tabela 5.6 Estatísticas Jitter médio rede saturada

5.3.3 Análise da vazão média

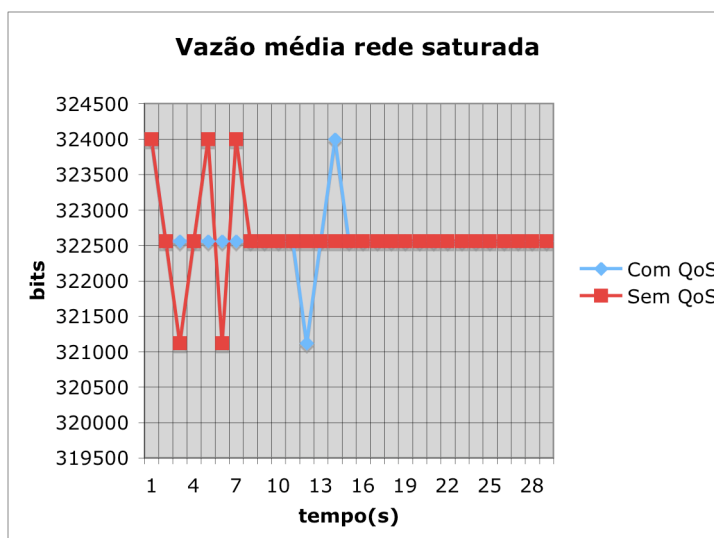


FIG. 5.12 VAZÃO MÉDIA REDE SATURADA LINUX

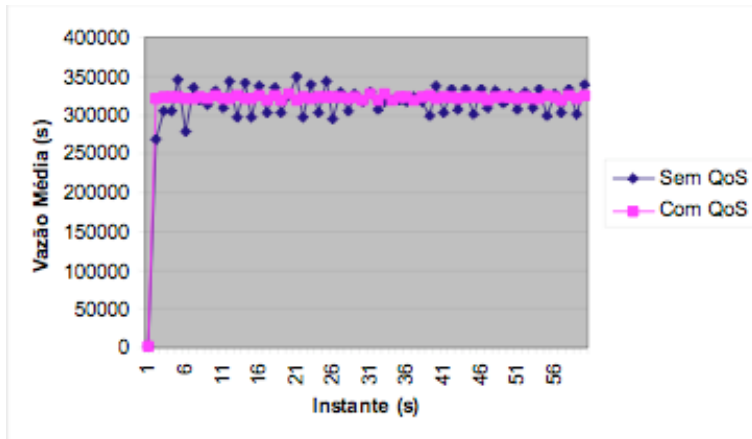


FIG. 5.13 VAZÃO MÉDIA REDE SATURADA FREEBSD

Aqui podemos notar que a vazão oscilou menos no Linux e nunca baixou do nível necessário as aplicações simuladas no teste, que seria 320kbps. E a vazão média foi ligeiramente melhor.

Linux	QoS(CBQ)	Sem QoS(FIFO)
Max	324000	324000
Min	321120	321120
Média	322609,65	322609,65

Tabela 5.6 Vazão média rede saturada, estatísticas.

5.3.4 Análise da taxa de perdas

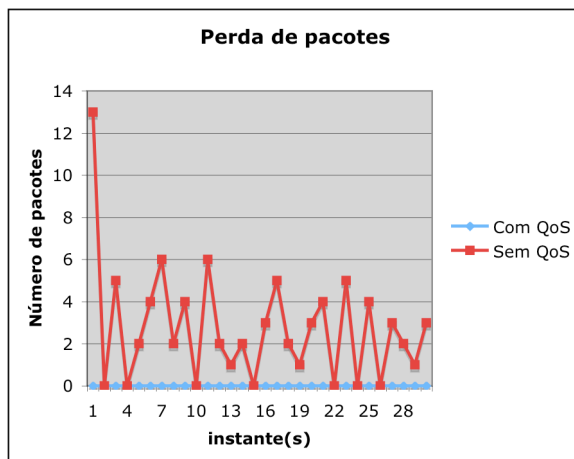


FIG. 5.14 PERDA DE PACOTES REDE SATURADA LINUX

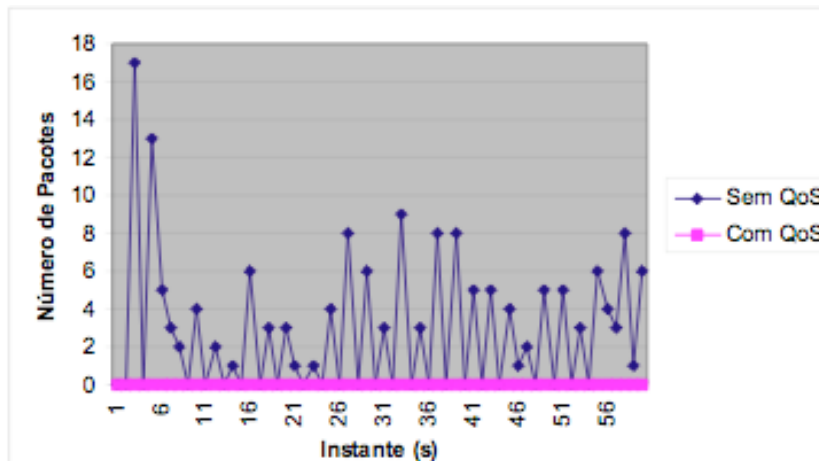


FIG. 5.15 PERDA DE PACOTES REDE SATURADA FREEBSD

Tanto no Linux como no FreeBSD algumas perdas de pacotes UDP durante os testes. A perda ocorreu apenas no caso da rede estar saturada e o roteador não efetuando o controle do tráfego (QoS) momento em que os pacotes UDP (referentes ao tráfego VoIP) concorriam, através do mecanismo FIFO, pela banda disponível com os pacotes TCP (referentes às demais aplicações).

Linux	QoS(CBQ)	Sem QoS(FIFO)
Max	0	13
Min	0	0
Média	0	2,413793103

Tabela 5.7 Perda de pacote rede saturada, estatísticas.

6. Conclusão e trabalhos futuros

Este trabalho foi de grande valia, possibilitando um estudo mais aprofundado sobre redes, e principalmente sobre o sistema operacional Linux e sua ferramenta de QoS e controle de tráfego, que se mostrou muito poderosa.

Isso é devido ao fato dela ser integrada ao Kernel, tornando o acesso as suas funções mais rápido e seguro, e as bibliotecas são compiladas de acordo com as características da máquina em questão.

O uso do sistema operacional FreeBSD e suas ferramentas de simulação de tráfego (RUDE&CRUDE, netperf), possibilitou mais um útil aprendizado.

Com base nos resultados obtidos nos experimentos usando FreeBSD e Linux na mesma rede e usando a mesma política de fila, chegamos a conclusão de que o Linux é o sistema operacional mais indicado para realizar roteamento com diferenciação de tráfego numa rede local.

Outro fator a ser destacado é a necessidade de usar QoS para garantir os requerimentos de aplicações que exijam mais recursos da rede como VoIP, videoconferência, streamings de áudio e vídeo em geral. Com isso se pode estabelecer um SLA, de forma a garantir a disponibilidade dos serviços para os clientes e usuários da rede.

Com a realização deste trabalho pode-se afirmar que a prática é fundamental para consolidar os ensinamentos teóricos obtidos ao longo do curso e traz retorno tanto para o aluno quanto para a Universidade na forma de conhecimento.

Como trabalhos futuros ficam as sugestões:

- Usar políticas de filas, classes e de filtros diferentes das utilizadas neste trabalho.
- Realizar uma carga mais pesada de tráfego.
- Utilizar um Ambiente de Alta-Disponibilidade para prover maior confiabilidade ao Sistema.
- Usar uma topologia de rede distinta.

7. REFERÊNCIAS

[Art] Artola, E. <http://penta.ufrgs.br/Esmilda/arquitet.html>

[Atm] ATM fórum. <http://atmforum.org>

[Bun04] Bunn, A., (2004), “*Avaliação de Serviço de QoS no FreeBSD*”

[Fer99] Ferguson, P. & Houston, G. (1999): *Quality of Service: Delivering QoS on the Internet and Corporate Networks* – Wiley computer Publishing.

[Gue02]Guerra, André (2002): <http://www.clubedasredes.eti.br/rede0007.htm>

[Hub02] Hubert, Bert(2002): “*Linux Advanced Routing & Traffic Control*” <http://lartc.org>

[Job99] Joberto Martins, “Redes Corporativas MultiServiço - Caracterização das Aplicações e Parâmetros Básicos de Operação”, Em <http://www.jsmnet.com/slides/AnaliseRequisitos/index.htm>

[JSMNet] “JSMNet - Estado da Arte e P&D em Redes de Computadores”.
Em <http://www.jsmnet.com>