Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Proposta e Análise de Desempenho de um Comutador de Pacotes com Enfileiramentos na Entrada e na Saída

Autor: Carlos Roberto dos Santos

Orientador: Prof. Dr. Shusaburo Motoyama

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: **Telecomunicações e Telemática.**

Banca Examinadora

Anilton Salles Garcia, Dr.....INF/Ufes, ES
Antônio Marcos Alberti, Dr.....DTE/Inatel, MG
Akebo Yamakami, Dr..... DT/FEEC/Unicamp, SP
Dalton Soares Arantes, Dr.....DECOM/FEEC/Unicamp, SP
Ivanil Sebastião Bonatti, Dr..... DT/FEEC/Unicamp, SP
Shusaburo Motoyama, Dr. .....DT/FEEC/Unicamp, SP

Campinas, SP

Abril/2006

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Santos, Carlos Roberto dos
Sa59p Proposta e análise de desempenho de um comutador de pacotes com enfileiramentos na entrada e na saída / Carlos Roberto dos Santos. --Campinas, SP: [s.n.], 2006.

Orientador: Shusaburo Motoyama
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Telecomunicações – Sistemas de comutação. 2. Telecomunicações. I. Shusaburo, Motoyama. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Titulo em Inglês: Proposal and Performance Analysis of a Combined Input and Output Queuing Packet Switch
Palavras-chave em Inglês: High speed switching, Quality of service, Switch performance
Área de concentração: Telecomunicações e Telemática.
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Anilton Salles Garcia, Antonio Alberti, Akeko Yamakami, Dalton Soares Arantes e Ivanil Sebastião Bonati
Data da defesa: 27/04/2006

Aos meus saudosos avós, Artur Rodrigues de Aguiar e Aurelina Soares da Rocha.

É graça divina começar bem. Graça maior persistir na caminhada certa. Mas a graça das graças é não desistir nunca. (Helder Câmara)

Resumo

Neste trabalho é proposto um comutador de pacotes baseado em uma estrutura *crossbar* com *m* enlaces paralelos internos, denominado comutador CEP (Comutador *Crossbar* de Enlaces Paralelos), e com facilidade para prover qualidade de serviço (QoS – *Quality of Service*). O comutador proposto utiliza uma combinação de filas na entrada e na saída. Os pacotes são transferidos dos *buffers* de entrada para os *buffers* de saída através de *m*xN linhas internas. Como as linhas internas e externas operam com a mesma velocidade, não há necessidade de aumentar a velocidade do *clock* interno, fazendo com que a estrutura proposta seja apropriada para comutadores de alta velocidade. O desempenho do comutador CEP é analisado admitindo pacotes de tamanho fixo (célula ATM) e pacotes de tamanho variável. O tempo médio de atraso dos pacotes e o tamanho médio das filas de entrada e de saída são avaliados por simulação e/ou por modelos analíticos, utilizando teoria de filas.

Palavras-Chave: Comutação em alta velocidade, Qualidade de serviço, Desempenho de Comutadores.

Abstract

A QoS (Quality of Service) provisioned CIOQ (Combined Input Output Queuing) switch using crossbar structure with m parallel lines per output port is proposed in this work. The packets at input buffers are transferred to the output buffers by means of *m*xN internal lines. Since all internal lines have the same speed as external links, no internal clock speedup is required so that the proposed structure is suited for high-speed switches. Switch models for analysis are proposed for both fixed and variable packet lengths and their performances, in terms of average packet waiting time and average queue size for both input and output buffers, are evaluated by simulation and/or analytically by means of queuing theory. The proposed switch also presents a feature that facilitates the choice of scheduler in order to satisfy the QoS of each class of service.

Keywords: High speed switching, Quality of Service, Switch Performance.

Agradecimentos

Ao meu amigo e orientador, Prof. Shusaburo Motoyama, a minha gratidão pela orientação, dedicação e suporte ao longo de todo o estudo.

Aos meus filhos, Bárbara, Carlos Jr. e Pedro Henrique, que compreenderam as ausências físicas e espirituais ao longo deste trabalho, e de modo muito especial à minha esposa Rosana que jamais deixou de me apoiar.

Aos meus pais Sebastião e Ione, a quem sou eternamente grato pela minha formação.

A minha sogra Dona Neusa pelas incansáveis orações.

Ao Inatel, que viabilizou e apoiou a realização deste trabalho através de seu Programa de Capacitação Docente.

Aos meus amigos Leandro Marques, Maria Helena Brusamolin e todos aqueles que, de uma forma ou de outra, contribuíram para a realização deste trabalho.

A Deus, pelo dom da vida. Obrigado pela saúde e oportunidade de concluir este doutorado.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ABR - Available Bit Rate

ARIS - Aggregate Route-Based IP Switching

ATM - Asynchronous Tranfer Mode

BP - Back-Pressure

CBR - Constant Bit Rate

CEP - Crossbar de Enlaces Paralelos

CICQ - Combined Input Crossbar Queuing

CSR - Cell Switch Router

FCFS - First Come First Served

FIFO - First In First Out

FIRO - First In Randon Out

GFR - Guaranteed Frame Rate

HOLB - Head Of Line Blocking

HTML - Hyper Text Markup Language

HTTP - Hypertext Transfer Protocol

IETF - Internet Engineering Task Force

IP - Internet Protocol

IRRM-MC - Iterative Round-Robin Matching with Multiple Class

ITU-T - International Telecommunication Union – Telecommunication Standardization sector

LCFS - Last Come First Served

MPLS - MultiProtocol Label Switching

MTU - Maximum Transmission Unit

PIM - Parallel Iterative Matching

PTM - Packet Transfer Mode – Modo de Transferência Pacote

QL - Queue Loss

QoS - Quality of Service

RAM - Random Access Memory

RDSI-FL - Redes Digitais de Serviços Integrados em Faixa Larga

SPIM - Simplified PIM

STM - Synchronous Transfer Mode

TCP - Transmission Control Protocol

TCP/UDP - Transport Control Protocol / User Datagram Protocol

UBR - Unspecified Bit Rate

VBR - Variable Bit Rate

VBR-nrt - Variable Bit Rate – non real time

VBR-rt - Variable Bit Rate - real time

VOQ - Virtual Output Queuing

WDM - Wavelength Division Multiplexing

Lista de Símbolos

B - Velocidade de operação do enlace externo

B - Probabilidade de bloqueio de células.

B_{av} - Probabilidade de bloqueio de células.

C - Capacidade do enlace.

E{N} - Número médio de pacotes no sistema.

E{Nq} - Número médio de pacotes na fila.

E{T} - Tempo médio de espera de pacotes no sistema.

E{Tq} - Tempo médio de espera de pacotes na fila.

fr - Fluxo de pacotes de classe r.

L - Tamanho médio dos pacotes em bits.

P_j - Probabilidade de j células serem enviadas a uma saída particular.

P(m) - Probabilidade de até *m* células serem transferidas para uma determinada saída.

S_i - Percentagem de tráfego de classe i.

λ_i - Taxa de chegada dos pacotes de classe i.

μ - Taxa de atendimento de pacotes.

ρ - Carregamento de Tráfego.

τc - Soma do tempo que o pacote ACK leva para propagar-se do cliente para o servidor e do tempo que o segmento de dados TCP leva para propagar-se do servidor para o roteador.

τ_l - Tempo de transmissão do segmento de dados TCP do roteador para o cliente.

τ_{total} - Tempo total para transmissão do segmento de dados TCP.

Capítulo 1

Introdução

Ao longo do tempo, várias estruturas de comutação têm sido desenvolvidas para diferentes aplicações como voz e dados, baseadas em modos de transferência como STM (*Synchronous Transfer Mode* – Modo de Transferência Síncrono) e PTM (*Packet Transfer Mode* – Modo de Transferência por Pacote), também denominados comutação de circuitos e comutação de pacotes, respectivamente. Estas estruturas de comutação têm sido adaptadas com o passar do tempo em função dos avanços tecnológicos.

A comutação de circuito tradicional foi concebida como uma forma de lidar e transportar tráfego do tipo *stream* como voz e vídeo. Nesta técnica, uma conexão com banda fixa é estabelecida, a qual provê vazão fixa e atraso constante. Apesar de ser simples e muito familiar, esta técnica é muito ineficiente para suportar serviços com diferentes taxas de bits. Mesmo em sistemas de comutação de circuito multi-taxas que permitem alocação de taxa igual a um múltiplo inteiro de uma taxa básica, como mostra [1], a escolha da taxa básica constitui uma tarefa difícil no mecanismo de decisão.

A comutação de pacotes foi desenvolvida primeiramente como uma maneira eficiente de transportar o tráfego de comunicação de dados. Suas principais características são o armazenamento intermediário (temporário), a multiplexação estatística e atrasos variáveis que são conseqüências do compartilhamento dinâmico dos recursos de comunicações, a fim de melhorar a utilização. Protocolos muito sofisticados têm sido desenvolvidos incorporando funções complexas como recuperação de erro, controle de fluxo, roteamento e controle de sessão. As competências desses comutadores de pacotes são muito atrativas para aplicações que requerem baixa vazão e baixo atraso, ou aquelas que requerem alta vazão, mas podem tolerar altos atrasos. Entretanto, essas competências não são suficientes para as aplicações em tempo real, tais como tráfegos de voz e vídeo.

Por outro lado, a demanda global por largura de faixa está crescendo a uma taxa exponencial, devido principalmente à proliferação da Internet e das aplicações multimídias emergentes,

demandando da rede comutadores de pacotes de alta velocidade. As tecnologias de transmissão óptica como WDM (*Wavelength Division Multiplexing* – multiplexação por divisão de comprimento de onda), têm suportado este crescimento no que diz respeito à transmissão, multiplicando a largura de faixa disponível na fibra óptica. Assim, os nós de comutação e os de roteamento das redes, implementados com tecnologia eletrônica, poderão ser, em pouco tempo, os gargalos da rede, caso não acompanhem este crescimento.

Os comutadores ópticos são soluções promissoras para se conseguir altas taxas de vazão, mas ainda apresentam duas desvantagens claras: primeiro, a granularidade é pobre, o que dificulta a comutação por pacotes, e segundo, a implementação prática de *buffer* óptico ainda é um grande desafio. Desta forma, espera-se que os comutadores de pacotes com tecnologia eletrônica continuem exercendo o seu papel por um longo tempo.

Tradicionalmente, os nós de encaminhamento de pacotes na rede Internet são baseados em roteadores, cujo encaminhamento dos pacotes IP (*Internet Protocol*) se baseia em mecanismos de camada 3 do modelo de referência OSI (*Open Systems Interconnection*). O desempenho desses roteadores é usualmente limitado, uma vez que o roteamento dos pacotes IP de comprimento variável é feito por *software*, um processo relativamente lento. As novas aplicações multimídias, como áudio e videoconferência, vieram impor novos requisitos de tráfego, fazendo com que a infra-estrutura da rede Internet tivesse que se adaptar.

Novas tecnologias têm sido desenvolvidas. Entre elas, uma das mais importantes é a tecnologia ATM (*Asynchronous Transfer Mode* – Modo de Transferência Assíncrono). Desde que o ITU-T (*International Telecommunication Union – Telecommunication Standardization Sector*) adotou o ATM como o padrão para a RDSI-FL (Redes Digitais de Serviços Integrados em Faixa Larga), ele vem sendo adotado pelas operadoras de telecomunicações como solução para implementação de seus *backbones*. Uma das principais vantagens do ATM é a possibilidade de se transportar tráfegos de diferentes perfis (voz, dados e vídeo) com qualidade de serviço (*Quality of Service* – QoS) garantida.

O ATM é uma tecnologia baseada na transmissão de pequenas unidades de informação (pacotes) de tamanho fixo e formato padronizado denominadas células. Células são transmitidas através de conexões com circuitos virtuais, com base em informações contidas no cabeçalho das mesmas. Esta tecnologia é capaz de suportar diferentes serviços com diferentes tipos de tráfego.

Uma célula ATM é um pequeno pacote de tamanho fixo, contendo 53 bytes, dos quais cinco bytes formam o cabeçalho e 48 bytes são a carga útil que pode ser voz, vídeo ou dados. Uma parte do cabeçalho é um identificador de conexão que é utilizado pelo comutador para encaminhar a célula a uma determinada porta de saída. O encaminhamento da célula pode ser feito praticamente em *hardware*, permitindo assim a comutação em alta velocidade.

Nas redes ATM as aplicações negociam um contrato de tráfego com a rede para cada conexão virtual estabelecida. O contrato de tráfego inclui os seguintes componentes: a categoria de serviço, a qualidade de serviço requerida, as características do tráfego da conexão e a definição de como o tráfego deve se comportar (definição de conformidade)[2].

As categorias de serviço previstas são: CBR (*Constant Bit Rate*), VBR (*Variable Bit Rate*), ABR (*Available Bit Rate*), GFR (*Guaranteed Frame Rate*) e UBR (*Unspecified Bit Rate*). A categoria VBR é dividida em duas sub-categorias, VBR-rt (*VBR - real time*) e VBR-nrt (*VBR – non real time*).

A técnica de comutação ATM pode ser aplicada para se conseguir sistemas de comutação IP de alta velocidade para a Internet. O sistema de comutação ATM possui algumas vantagens em termos de preço/desempenho em relação ao encaminhamento de pacote para uso na Internet. Entretanto, a integração da tecnologia ATM e o encaminhamento de pacotes IP apresentam alguns desafios. Uma questão importante é que o comprimento do pacote IP é variável, enquanto que o de uma célula ATM é fixo. Assim, os pacotes IP que chegam aos comutadores são segmentados e comutados como células ATM. Nas saídas os pacotes são remontados a partir das células e encaminhados ao próximo nó.

Uma abordagem mais eficiente é comutar os pacotes IP diretamente, ou seja, sem a necessidade de segmentá-los em células ATM. Para acompanhar a crescente demanda imposta pelas redes IP, novas tecnologias ou mecanismos de roteamento, denominados de comutação IP, têm sido propostos a fim de aumentar o desempenho dos comutadores IP. As tecnologias mais populares são, IP *switching* da Ipsilon [3], *Tag switching* da Cisco [4], ARIS (*Aggregate Route-Based IP switching*) da IBM [5] e CSR (*Cell Switch Router*) da Toshiba [6]. Essas tecnologias utilizam basicamente o mesmo mecanismo de comutação baseado em rótulo ou etiqueta que combina a velocidade de comutação da camada 2 com a flexibilidade de roteamento da camada 3 do modelo OSI, mas diferem na forma de associar o rótulo ou etiqueta a um determinado fluxo. O

IETF (*Internet Engineering Task Force*) [7] padronizou a idéia sob o nome MPLS (*MultiProtocol Label Switching* – comutação de rótulos multiprotocolos) [8]. Assim, um comutador IP integra a flexibilidade do roteamento IP com a velocidade de comutação ATM. Algumas pesquisas na área de comutação IP incluem projeto de estruturas, especificações de protocolo, estudos com simulação e laboratório, e algoritmos para classificação de fluxos [9].

Conceitualmente, um comutador IP consiste de um comutador ATM com célula de comprimento variável (*hardware*) e um controlador de comutação IP (*software*). O comutador ATM executa a comutação dos fluxos selecionados enquanto o controlador de comutação IP executa a classificação dos fluxos, bem como o encaminhamento dos fluxos ainda não selecionados. Um fluxo IP consiste de uma série de pacotes IP que compartilham de propriedades comuns como prefixo de endereço IP ou pares de endereços IPs (mesmos endereços de origem e destino) e eventualmente também pares de portas TCP/UDP (*Transport Control Protocol / User Datagram Protocol*).

Várias estruturas de comutação IP têm sido propostas na literatura, juntamente com as respectivas análises de desempenho [3] – [5], [10] e [11]. Em [3], [5] e [10], as estruturas de comutação IP propostas são baseadas em uma combinação da camada 2 do ATM com roteamento IP. Nestas estruturas os pacotes que chegam aos comutadores são segmentados e comutados como células ATM. Nas saídas, os pacotes são remontados a partir das células e transmitidos para o próximo nó. Nas estruturas propostas em [4] e [11], um rótulo com finalidade de comutação é acrescentado ao pacote. Tal rótulo tem função de comutação de camada 2 e o pacote é comutado sem segmentação.

O objetivo desta tese é propor uma estrutura de comutação de pacotes para aplicações em redes ATM e Internet. A estrutura proposta é denominada de Comutador *Crossbar* de Enlaces Paralelos – CEP. O comutador CEP possui características de comutar pacotes em alta velocidade, de ter alto desempenho, de garantir equidade entre as entradas e entre as saídas, de dar suporte a aplicações com qualidade de serviço e de ter robustez quando submetido a tráfego real. O comutador CEP é baseado em uma estrutura com enfileiramentos nas entradas e nas saídas, e além de permitir a transferência paralela de pacotes das entradas para as saídas, pode suportar qualquer algoritmo de escalonamento para obter altas taxas de comutação mesmo com a utilização de escalonadores simples.

Organização da Tese

Esta tese está estruturada da seguinte forma: este capítulo (Capítulo 1) é uma introdução geral contendo um breve histórico e perspectivas dos sistemas de comutação, abordando o crescimento e mudanças nos requisitos de tráfego em face das novas aplicações multimídias emergentes. Uma visão geral sobre as novas tecnologias de comutação existentes é feita com a finalidade de introduzir e caracterizar o domínio do problema abordado nesta tese, bem como a solução proposta.

No Capítulo 2, além de uma revisão bibliográfica, é feito um resumo das estruturas de comutadores de pacotes existentes, com base nas três principais classes de estruturas do ponto de vista da localização dos *buffers* e disciplina de enfileiramento: *buffer* na entrada, *buffer* na saída e *buffer* compartilhado. São revistas também as variações mais recentemente utilizadas, como fila virtual de entrada e filas combinadas nas entradas e nas saídas. As vantagens e desvantagens de cada uma das classes são evidenciadas. No final do capítulo, é apresentada a idéia da estrutura do comutador proposto.

A estrutura do comutador CEP proposto é apresentada no Capítulo 3. É feita uma explicação do princípio da estrutura, bem como um detalhamento da sua operação, considerando pacotes de tamanho fixo (célula ATM), utilizando escalonamentos *Round-robin* e por prioridade. O desempenho da estrutura, sob certas condições de tráfego, é analisado matematicamente e por simulação, em termos de comportamento das filas de entrada. Os vários resultados obtidos são apresentados e discutidos.

No Capítulo 4 é avaliado o desempenho do comutador CEP operando com pacotes de tamanho variável. O novo princípio básico de operação é explicado admitindo escalonamentos *Round-robin* e por prioridade. Considerando um tráfego do tipo Poisson, o desempenho é analisado matematicamente e por simulação, tanto para as filas de entrada quanto para as filas de saída. A fim de verificar o desempenho do comutador proposto quando submetido a uma fonte de tráfego real, o mesmo foi submetido a um modelo de fonte HTTP (*Hyper Text Transfer Protocol*) e o seu desempenho avaliado por simulação. Características do tráfego *web* e detalhes da simulação da fonte HTTP são mostrados nesse capítulo.

Finalmente no Capítulo 5 é feito um resumo das conclusões e indicações de continuidade para futuros trabalhos.

Resumo das Contribuições e Publicações Geradas

A seguir é apresentado um resumo das principais contribuições desta tese, bem como das publicações geradas.

No Capítulo 3 é proposta uma estrutura de um comutador ATM que utiliza um único *buffer* do tipo FIFO na entrada, *m buffers* do tipo FIFO em cada saída e *m* enlaces internos conectando os *buffers* de entrada aos *m buffers* de saída. Desta forma, até *m* células poderão ser transferidas das entradas para uma mesma saída. A análise de desempenho é feita considerando três cenários: o primeiro considera que, entre as entradas, o atendimento é do tipo *Round-robin*, e no *buffer* a disciplina de atendimento é do tipo FIFO. O segundo considera que o atendimento entre as entradas é do tipo *Round-robin*, mas a disciplina no *buffer* é do tipo aleatória. No terceiro cenário o atendimento entre as entradas é por prioridade e a disciplina no *buffer* é do tipo FIFO. Os resultados apresentados neste capítulo foram publicados nos seguintes artigos:

- Motoyama, S., Santos, C. R., *A Combined Input-Output Queuing ATM Switch With m Internal Links for Cell Transfer*, International Telecommunications Symposium (ITS), Natal, Brazil, September 2002, 142-146.
- Motoyama, S., Santos, C. R., *Performance Analysis of a CIOQ ATM Switch With m Internal Links for Cell Transfer*, 10th International Conference on Software, Telecommunications & Computer Networks – SoftCom 2002, Venice/Ancona, Italy, Split/Dubrovnik, Croatia, November 2002, 636 – 640.
- Santos, C. R., Motoyama, S., *A QoS Provisioned CIOQ ATM Switch with m Internal Links*, 11th International Conference on Telecommunications – ICT 2004, Fortaleza, Brazil, August 2004, 698 – 703.

No Capítulo 4 a estrutura proposta é submetida a um tráfego de pacotes de tamanho variável. O seu desempenho é verificado admitindo um tráfego do tipo Poisson e sob uma fonte de tráfego HTTP. Para fonte de tráfego do tipo Poisson foi analisado o desempenho considerando atendimentos *Round-robin* e por prioridade. Os resultados apresentados nessa análise foram publicados no seguinte artigo:

- Santos, C. R., Motoyama, S.: "A QoS Provisioned CIOQ Packet Switch Using Crossbar Structure with *m* Internal Links", International Conference on High Speed Network – ICHSN - 2005, Montreal, Canada, August 2005, 241 – 247.

Uma outra importante contribuição desta tese foi que ela serviu de inspiração como tema de dissertação de mestrado apresentada em [12]. Os principais objetivos da dissertação eram avaliar os resultados até então obtidos nesta tese utilizando uma outra ferramenta de simulação, e analisar outros parâmetros como comprimento máximo do *buffer* de entrada e desempenho do *buffer* de saída. Esta dissertação também gerou o seguinte artigo:

- Alberti, A. M, Aguiar Filho, S. R, Garcia, A. S.: "Modeling Simulation and Performance Evaluation for a CIOQ Switch Architecture", Annual Simulation Symposium - ANSS 2006, Hunstville, Alabama, EUA, April 2006.

A principal contribuição no primeiro artigo foi apresentar uma estrutura de comutação ATM, relativamente simples, capaz de transferir simultaneamente até *m* células das entradas para as saídas, sem a necessidade de *speed-up* interno, utilizando algoritmo de escalonamento simples. No segundo artigo, a principal contribuição foi no sentido de mostrar que a solução utilizando apenas dois enlaces internos e *buffer* do tipo FIFO nas entradas apresenta um desempenho melhor, em termos de diminuição do HOLB (*Head Of Line Blocking* – bloqueio na cabeça da fila), do que a solução tradicional com *buffer* randômico nas entradas. Além disto, mostrou-se também que não há aumento no desempenho quando se adota mais de um enlace interno e *buffer* randômico, simultaneamente. No terceiro artigo, a contribuição foi apresentar a estrutura do comutador ATM, modificada para atender aplicações que requerem qualidade de serviço. No quarto artigo, mostrou-se que a estrutura proposta inicialmente para comutar pacotes de tamanho fixo (células ATM) pode também ser utilizada para comutar pacotes de tamanho variável, bastando uma pequena alteração no algoritmo de escalonamento. Adicionalmente, em cada artigo é apresentado a respectiva análise de desempenho.

Capítulo 2

Estruturas de Comutadores de Pacotes

2.1 Introdução

Neste capítulo é feita uma breve revisão de alguns conceitos básicos e dos principais tipos de estruturas de comutadores de pacotes do ponto de vista de estratégia de localização dos *buffers*, com o objetivo de levantar as vantagens e desvantagens relevantes de cada uma. Um estudo mais completo pode ser encontrado em [13] - [17]. Em [18] é dado um foco especial nas estruturas com divisão espacial.

Além das estruturas clássicas com *buffer* na entrada, *buffer* na saída e *buffer* compartilhado, recentemente duas outras estruturas de comutadores têm recebido muita atenção: fila virtual de saída, e filas de entrada e de saída combinadas. A estrutura com fila virtual de saída é um esquema de enfileiramento usado para resolver o problema de bloqueio na cabeça da fila, associado a filas do tipo FIFO (*First In First Out*) ou para prover qualidade de serviço. Nesta técnica, cada porta de entrada mantém uma fila virtual separada para cada porta de saída. A estrutura com filas de entrada e de saída combinadas é um esquema que combina enfileiramento na entrada e na saída, sendo uma boa solução de compromisso entre desempenho e escalabilidade.

2.2 Classificação das Estruturas de Comutadores

Na literatura diversas publicações usam diferentes formas para classificar as estruturas de comutadores em diversas categorias. Alguns critérios utilizados são: com bloqueio e sem bloqueio, estratégia de localização dos *buffers* (*buffer* na entrada, *buffer* na saída, *buffer* compartilhado, etc.), com perda e sem perda, único estágio ou vários estágios, com divisão no tempo ou no espaço.

Uma estrutura de comutação é denominada sem bloqueio quando ela garante que, um pacote que esteja chegando com destino a uma determinada porta de saída possa ser encaminhado imediatamente, caso já não exista nenhum outro pacote endereçado à mesma saída. Desta forma,

uma saída será sempre servida se existir pelo menos um pacote endereçado a ela. Este tipo de comutador é denominado *work conserving*.

Duas formas de bloqueio podem ser distinguidas. A primeira, em função da natureza estatística da chegada dos pacotes ao comutador, pode acontecer de se ter mais de um pacote chegando simultaneamente nas entradas, todos eles endereçados à mesma saída. Se apenas um desses pacotes pode ser encaminhado à saída, os outros terão que esperar. Assim, o comutador deverá ser equipado com *buffers* para acomodar os pacotes que não podem ser encaminhados imediatamente. Esta forma de bloqueio é denominada de contenção de saída. A segunda forma de bloqueio está relacionada à estrutura de comutação baseada em multi-estágio, quando a contenção pode ocorrer na malha de comutação interna mesmo entre pacotes destinados a diferentes portas de saída. As estruturas que exibem este comportamento, como Banyan, são denominadas de com bloqueio.

Uma outra característica importante dos comutadores é a necessidade ou não de se preservar a ordem com que os pacotes deixam o comutador. Certas tecnologias, como o ATM (*Asynchronous Transfer Mode* - modo de transferência assíncrono), requerem que os pacotes pertencentes ao mesmo fluxo ou à mesma conexão virtual sejam entregues na mesma ordem em que chegaram. Isto significa que o comutador deve manter a seqüência dos pacotes que pertençam a uma dada combinação de porta de entrada, porta de saída e enlace virtual.

Estruturas de comutação com um único estágio possuem desempenho superior às estruturas multi-estágios, mas não são inerentemente escaláveis. Para redes de grandes dimensões, do ponto de vista de custos, a abordagem por multi-estágio é mais interessante do que por único estágio. As métricas normalmente utilizadas para verificar desempenho são:

- Vazão média: definido como o número médio de pacotes que deixam o comutador durante um ciclo de pacote dividido pelo número de portas de saída do comutador. Equivale à fração do tempo em que as linhas de saída estão sendo utilizadas.
- Atraso médio de pacotes: tempo médio encontrado pelo pacote para atravessar o comutador.
- Taxa média de perda de pacotes: percentual de pacotes que chegam e são descartados pelo comutador devido à falta de espaço no *buffer*.
- Variação média do atraso de pacotes (*jitter*): representa a distribuição dos atrasos de pacotes em torno do atraso médio.

A Fig. 2.1 ilustra uma malha de comutação *N* x *N* (*N* entradas e *N* saídas) de três estágios. Neste caso, o primeiro e o terceiro estágios possuem *N/n* elementos básicos de comutação *n* x *k*, enquanto o segundo estágio possui *k* elementos básicos de comutação *N/n x N/n*. Nesta estrutura, o número de matrizes básicas é dado por *2N/n+k,* e se o número de matrizes básicas no segundo estágio (*k*) for igual ao número de entradas em cada bloco básico (n) das entradas menos um, (*k=2n-1*), a malha de comutação será sem bloqueio.

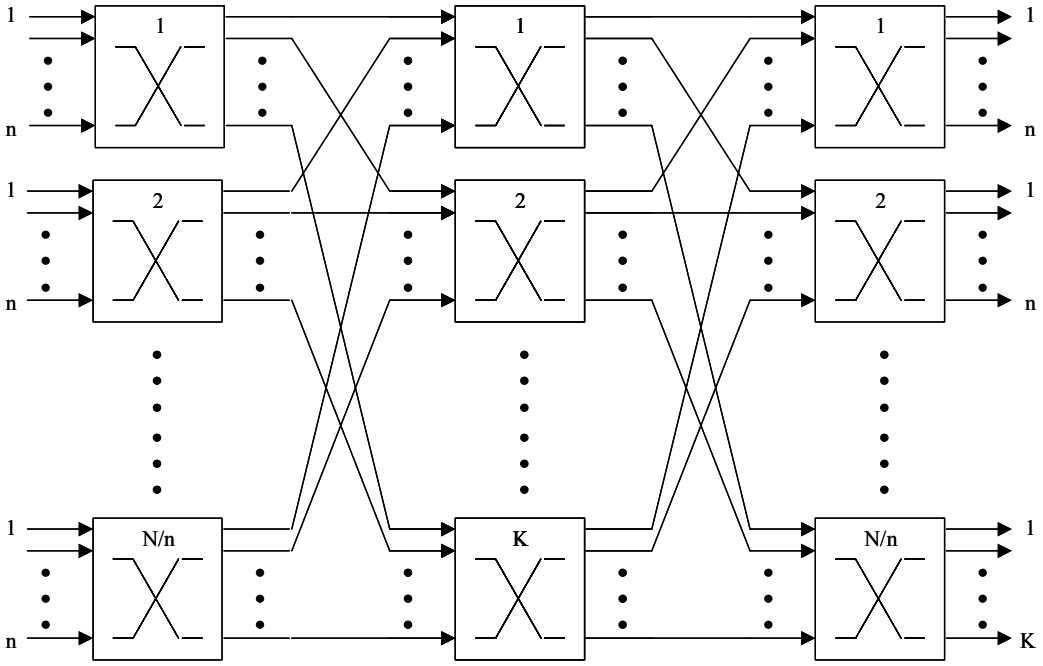

Figura 2.1- Malha de comutação *N* x *N* com 03 estágios.

Em função da estrutura proposta nesta tese, apenas as estruturas com um único estágio serão abordadas com maior ênfase neste capítulo. Um estudo mais completo sobre estruturas do tipo multi-estágio é encontrado em [19].

2.3 Estrutura com Filas nas Entradas

Formalmente, um comutador pertence à classe de comutadores com fila na entrada se a função de armazenamento dos pacotes é executada antes da função de comutação. A colocação de *buffers* na entrada representa uma solução clássica e simples para resolver o problema da contenção de saída. Os pacotes que chegam às portas de entrada e não podem ser encaminhados imediatamente são armazenados nesses *buffers*. A razão deste tipo de estrutura continuar sendo utilizada se deve principalmente ao fato de não exigir velocidade de operação muito alta para

escrita ou leitura dos pacotes quando comparada à velocidade de operação do enlace em cada porta. Assim, a velocidade de operação em cada *buffer* é proporcional à velocidade de cada porta e não do número de portas. Apenas uma operação de escrita e uma de leitura são requeridas por ciclo de pacote em cada *buffer*. Se a velocidade de operação de cada porta é *B* bits por segundo (bps), a velocidade de operação requerida em cada *buffer* será 2.*B*, enquanto a velocidade agregada será de 2.*N*.*B* para um comutador de tamanho *N* x *N*.

A contenção de saída (quando múltiplas entradas possuem pacotes endereçados a uma mesma saída) é resolvida por uma unidade de controle denominada árbitro. Em cada ciclo o árbitro seleciona uma entrada que tenha um pacote presente na cabeça da fila para ser transmitido para uma das saídas. Cada porta de entrada poderá encaminhar no máximo um pacote para uma determinada saída. Adicionalmente, o árbitro deverá prover uma política de seleção de tal forma que haja, por exemplo, eqüidade ou justiça entre as entradas. Uma vez feita a seleção das entradas, os pacotes são removidos dos seus *buffers* e encaminhados às portas de saída através da malha de comutação.

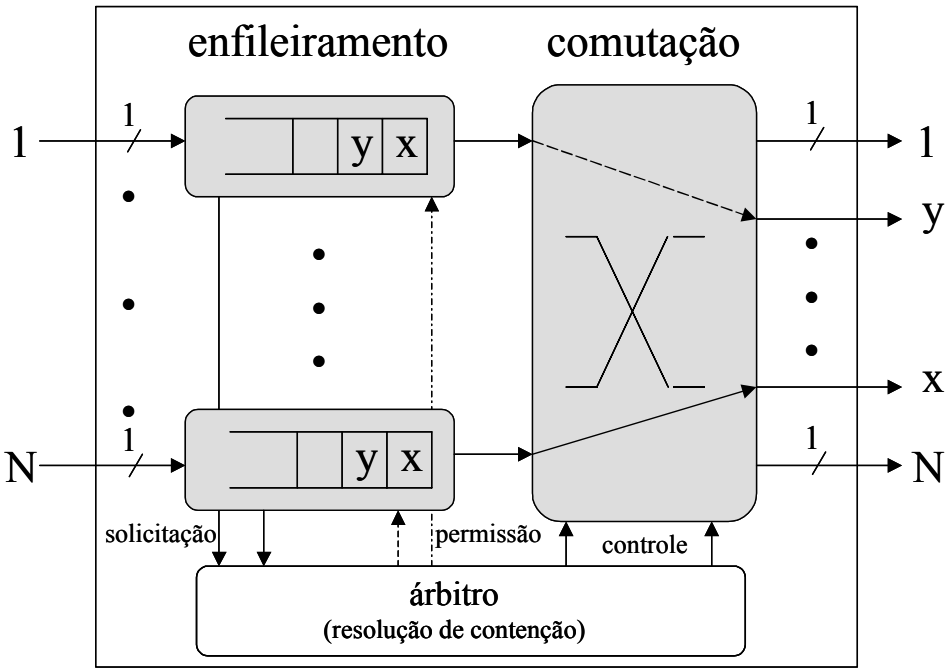

Figura 2.2- Estrutura de um comutador com filas do tipo FIFO nas entradas.

Em um comutador com filas na entrada do tipo FIFO (*First In First Out* – o primeiro que entra é o primeiro que sai), como mostra a Fig. 2.2, quando mais de uma entrada possuem na cabeça da fila pacotes endereçados a uma mesma saída, somente uma delas recebe permissão para

transmitir, de acordo com as regras estabelecidas. Vários algoritmos de seleção são propostos na literatura, e uma revisão detalhada dos principais pode ser encontrada em [18]. Os principais algoritmos de seleção usados são:

- Seleção aleatória: uma entrada é selecionada aleatoriamente entre aquelas que possuem pacotes endereçados a uma mesma saída;
- Seleção *Round-robin*: a próxima entrada selecionada para ser servida será aquela depois de uma outra que foi servida por último. Neste caso, cada saída mantém um ponteiro para a última entrada servida;
- Seleção baseada em ponderação: neste caso existem várias alternativas onde a seleção pode levar em consideração alguns parâmetros da fila. Por exemplo, a fila com o maior número de pacotes esperando, a fila que menos vezes foi atendida, etc.

Em [20] é mostrado que o esquema de fila do tipo FIFO na entrada limita a vazão máxima em $2-\sqrt{2} \approx 58{,}6\%$, supondo fontes de tráfego uniformemente distribuídos e independentes do tipo Bernoulli. Um dos motivos da vazão máxima ser baixa se deve ao fenômeno conhecido como HOLB (*Head Of Line Blocking* - bloqueio na cabeça da fila), isto é, quando um pacote na cabeça da fila impede que outros pacotes localizados atrás dele, endereçados a saídas livres, possam ser transmitidos.

A partir do problema de HOLB, pode-se concluir que este tipo de comutador com fila na entrada não é do tipo *work conserving*. No exemplo da Fig. 2.2 existe um pacote endereçado à saída y mas ele não poderá ser servido, uma vez que está sendo bloqueado pelo pacote endereçado à saída x. É importante notar que determinados requisitos, como por exemplo eqüidade ou justiça, podem conflitar com a condição de *work conserving*.

Vários esquemas de escalonamento têm sido propostos no sentido de amenizar ou solucionar o problema do HOLB e consequentemente aumentar a vazão. Em [21] é proposto um mecanismo de resolução de contenção distribuído que aloca dinamicamente tempos de transmissão para os pacotes armazenados nos *buffers* de entrada. Uma análise aproximada e simulações mostram, para um tráfego i.i.d. uniforme, uma vazão máxima em torno de 76%, considerando um sistema com 16 entradas e 16 saídas (16 x 16).

Em [22] é proposto um esquema baseado em reserva de *timeslot*. Este procedimento requer o uso de *buffers* do tipo FIRO (*First In Random Out*), ou seja, escrita seqüencial e leitura aleatória,

uma vez que o algoritmo executa uma busca em cada *buffer* de entrada até uma determinada profundidade *d*, para encontrar um pacote que possa ocupar um *timeslot* não reservado naquele ciclo. Para uma busca com profundidade $d = 16$ e um sistema 16 x 16, a vazão máxima é em torno de 90%, para tráfego do tipo i.i.d. uniforme.

Um estudo comparativo do desempenho dos algoritmos de escalonamentos IRRM-MC (*Iterative Round-Robin Matching with Multiple Class*), PIM (*Parallel Iterative Matching*), SPIM (*Simplified PIM*) e SLIP-IRRMC para um comutador com classes de serviços, é feito em [23] e [24]. Para uma chegada do tipo Bernoulli, os resultados mostram que o atraso dos pacotes pode ser mantido baixo para os quatro algoritmos analisados. Com cinco iterações, os algoritmos PIM, SPIM e IRRC-MC apresentam uma vazão em torno de 98%. O algoritmo SLIP-IRRMC apresenta uma vazão praticamente igual a 100% com uma única iteração.

Ainda com relação ao HOLB, várias referências são citadas em [25]. Algumas abordam a solução por variações no escalonamento e outras por variações na estrutura, como aumento da velocidade interna em relação à velocidade dos enlaces externos, uso de múltiplas malhas de comutação em paralelo, etc.

2.4 Estrutura com Filas nas Saídas

Formalmente, um comutador pertence à classe de comutadores com fila na saída se a função de armazenamento dos pacotes é executada depois da função de comutação. Teoricamente, um comutador com *buffer* na saída apresenta um desempenho máximo ideal, uma vez que não apresenta o problema de HOLB e a contenção é reduzida ao mínimo, ou seja, existe apenas a contenção de saída que é inevitável por causa da natureza estatística do tráfego de pacotes. Um comutador com filas na saída com *buffer* de tamanho infinito apresenta melhor desempenho, em termos de vazão e atraso dos pacotes, quando comparado a um comutador com filas nas entradas. De fato, este tipo de comutador é o único tipo verdadeiramente *work conserving* no sentido da definição. A Fig. 2.3 ilustra uma estrutura de um comutador com filas nas saídas.

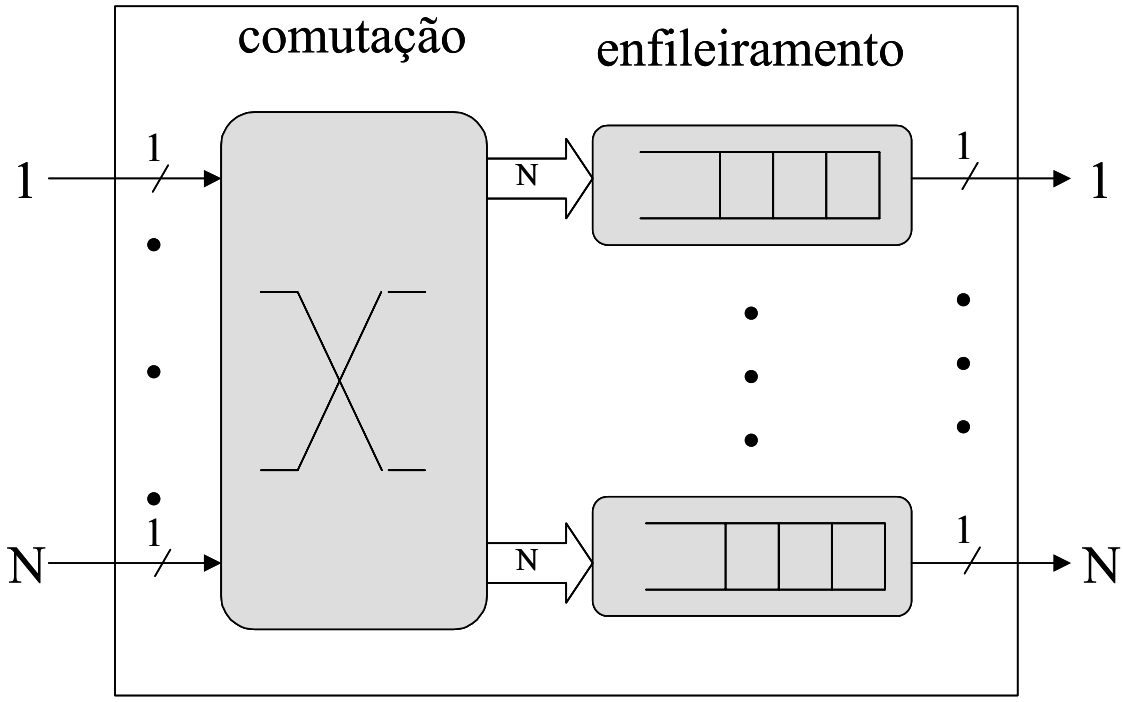

Figura 2.3- Estrutura de um comutador com filas nas saídas.

O comutador com filas na saída requer uma velocidade de operação de escrita/leitura nos *buffers* maior que o comutador com filas nas entradas. A velocidade requerida em um único *buffer* é proporcional à velocidade de operação dos enlaces externos e do número de portas, uma vez que em um único ciclo de pacote pode-se ter até N pacotes chegando com destino à mesma saída. Logo serão necessárias N escritas (N pacotes chegando) e uma leitura (um pacote partindo). Sendo B a velocidade do enlace externo, a velocidade de operação requerida em cada porta de saída será $(N + 1)B$, enquanto a taxa agregada será $N(N+1)B$.

Como não há *buffer* na entrada, é necessário que os pacotes que chegam às entradas endereçadas a uma mesma saída sejam todos transmitidos no mesmo ciclo, ou seja, a velocidade de operação interna deverá ser, no limite, N vezes maior que a velocidade de operação externa. Este fenômeno é denominado de *speed-up factor* (fator de aceleração). Com a velocidade dos meios de transmissão crescendo a cada dia, esta condição não deixa de ser uma desvantagem, fazendo com que este tipo de comutador tenha problemas de escalabilidade, uma vez que o fator de aceleração aumenta com o aumento do número de portas.

2.5 Estrutura com Filas nos Cruzamentos

A Fig. 2.4 ilustra um comutador com *buffer* nos cruzamentos, implementado pela Fujitsu e descrito por [26] como matriz de comutação por barramento. Alguns autores definem este tipo de estrutura como sendo com filas na saída, uma vez que, quando o pacote é armazenado no *buffer* de cruzamento, ele já chegou à porta de saída de destino. A função de comutação, executada pelos filtros de endereços AF (*Address Filter*), ocorre antes do armazenamento.

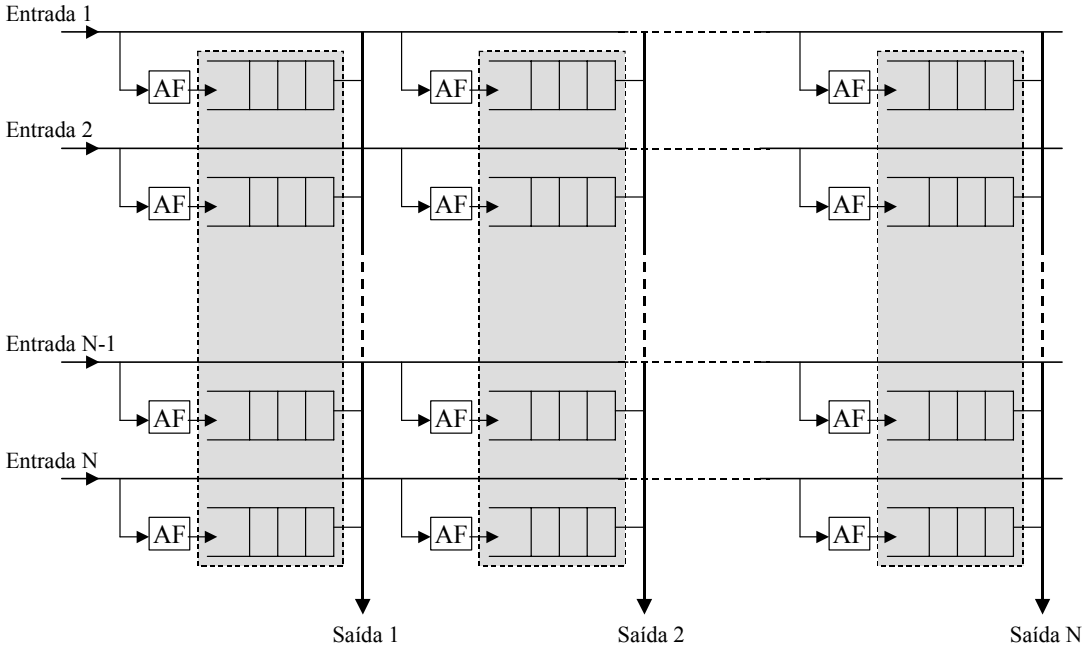

Figura 2.4- Estrutura de um comutador com filas nos cruzamentos.

Os *buffers* nos cruzamentos podem ser físicos ou lógicos, ou seja, cada saída pode possuir filas distribuídas ou cada saída pode ter um único *buffer* compartilhado por várias entradas (sombreado na Fig. 2.4). No primeiro caso, uma estrutura N x N teria N^2 *buffers*, e no segundo haveria a necessidade de operar com aceleração da velocidade interna, uma vez que pode-se ter N escritas e uma leitura em um ciclo. Os dois casos são estudados analiticamente em [27].

Uma estrutura que tem aparecido na literatura e que pode ser derivada tanto da estrutura com *buffer* na entrada quanto da estrutura com *buffer* nos cruzamentos é denominada de estrutura com enfileiramentos na entrada e nos cruzamentos (CICQ – *Combined Input Crossbar Queuing*). Em [11] é apresentada uma estrutura de comutador de pacotes que utiliza *buffer* na entrada e um único *buffer* em cada cruzamento. O seu desempenho é analisado considerando pacotes IP com classes de fluxos.

2.6 Estrutura com Filas Compartilhadas

As estruturas de comutação com filas nas entradas ou nas saídas tratadas até agora utilizam uma quantidade fixa de *buffer* dedicada a cada entrada ou a cada saída, podendo levar a uma utilização não otimizada desses recursos. A estrutura com filas compartilhadas foi proposta com o objetivo de utilizar melhor os recursos de *buffer*. Neste caso, todas as saídas, ou entradas, ou grupos de entradas ou saídas, compartilham os mesmos recursos de *buffer*, com o objetivo de se ter melhor eficiência na utilização.

Embora o conceito de compartilhamento tenha provado uma redução efetiva na perda de pacotes, as referências [28] e [29] mostram que isto pode levar a uma iniqüidade, como por exemplo, pacotes de uma determinada entrada, ou destinados a uma determinada saída, monopolizarem os recursos compartilhados, causando uma degradação no sistema.

A Fig. 2.5 ilustra um esquema de comutador com buffer na entrada onde um grupo de k entradas são agrupadas para compartilhar um mesmo buffer, proposto por [30].

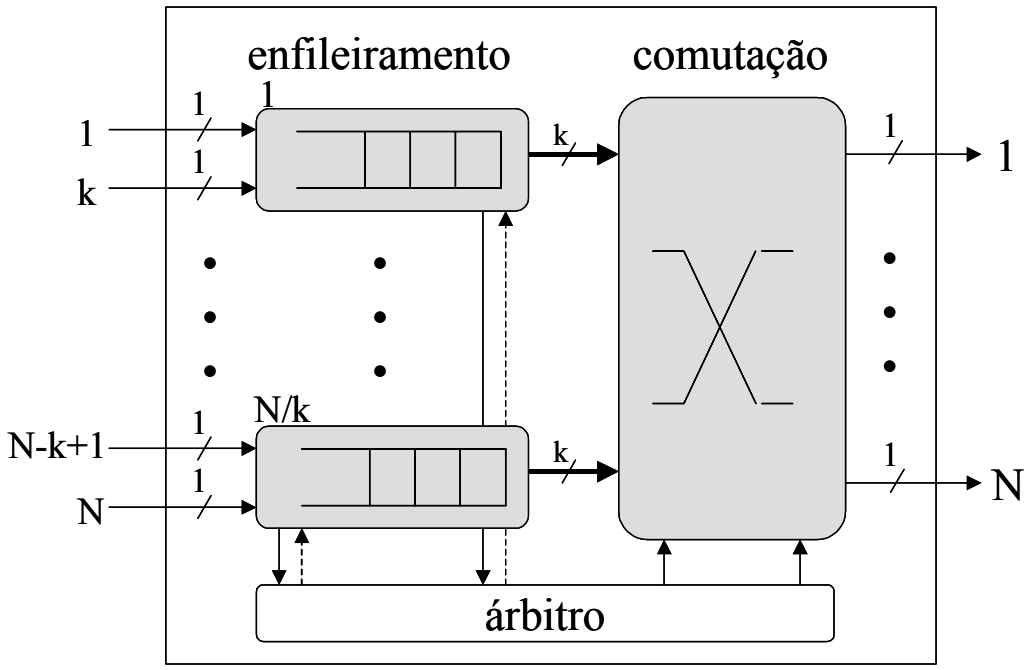

Figura 2.5- Estrutura de um comutador com *buffer* de entrada e entradas agrupadas.

Em cada ciclo, até k pacotes são selecionados em cada grupo. Neste caso, a disciplina de fila não pode ser FIFO. Os pacotes a serem transmitidos são selecionados da seguinte forma:

- Um grupo é escolhido aleatoriamente;

- Deste grupo, até k pacotes com diferentes destinos são escolhidos, marcando as portas de saídas selecionadas como ocupadas;
- Repete-se o passo anterior para o próximo grupo até todos os grupos serem processados ou não haver mais portas de saídas livres.

Como explicitado em [25], este algoritmo de seleção não é muito eficiente, pois no pior caso, em *N* interações, todo o agrupamento de filas deverá ser pesquisado.

A maioria das implementações práticas de comutadores com filas na saída utilizam *buffer* compartilhado, onde todas as filas de saída compartilham uma mesma memória, como mostra a Fig. 2.6.

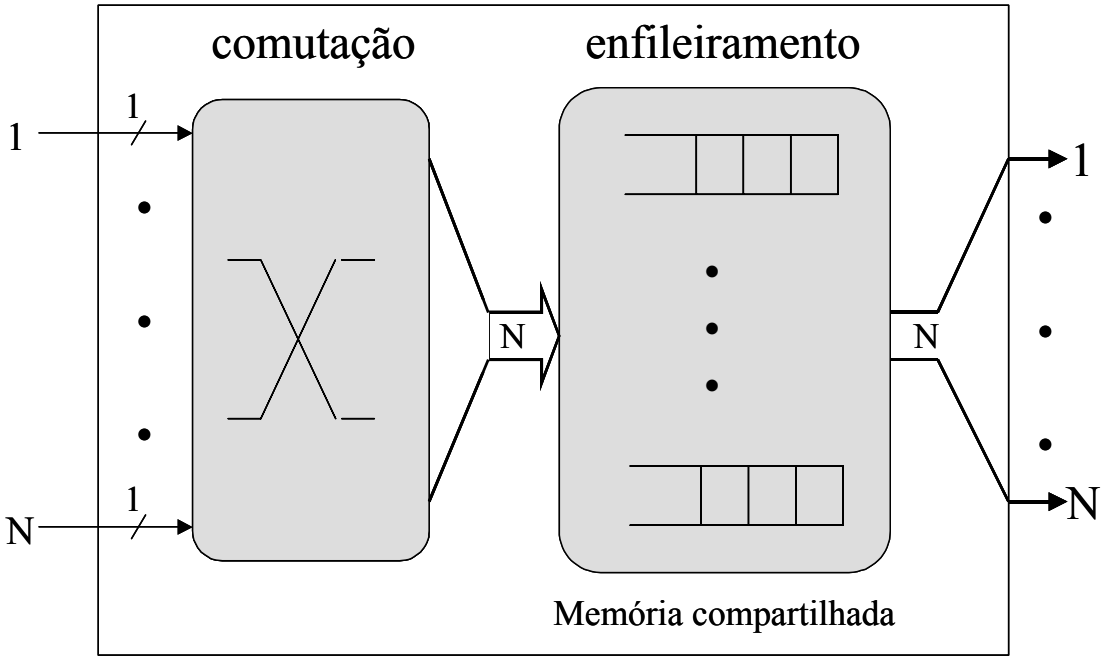

Figura 2.6- Estrutura lógica de um comutador com *buffer* de saída compartilhado.

Do ponto de vista de implementação, esta abordagem apresenta uma vantagem significativa sobre filas dedicadas para cada saída, uma vez que a velocidade de operação agregada é igual a $2.N.B$, ou seja, igual à taxa agregada do comutador com filas na entrada, em vez de $N(N+1)B$. Adicionalmente, as referências [28], [29], [31], [32] mostram que se tem uma eficiência maior na utilização da memória quando o espaço de memória disponível é compartilhado entre todas as saídas. Os resultados mostram uma melhoria significativa na taxa média de perda de pacotes, uma vez que, com o compartilhamento, um pacote só será perdido quando todo espaço da memória compartilhada estiver ocupado.

Embora esta abordagem reduza a taxa agregada para 2.*N.B* quando comparada com filas dedicadas por saída, a sua implementação é um grande desafio. No caso da estrutura com *buffer* na entrada, e que possui a mesma taxa agregada 2.*N.B*, a memória é distribuída entre as *N* entradas separadas fisicamente, enquanto que, na estrutura com *buffer* compartilhado na saída, toda a taxa é afunilada para uma única memória. Algumas soluções de implementações são referenciadas em [25].

2.7 Estrutura com Fila Virtual de Saída

Devido aos problemas associados às estruturas com *buffer* compartilhado e com *buffer* na saída, em particular a escalabilidade por causa da taxa de operação, uma outra estrutura tem recebido muita atenção ultimamente. Ela é denominada de VOQ (*Virtual Output Queuing* - Fila Virtual de Saida), ilustrada na Fig. 2.7.

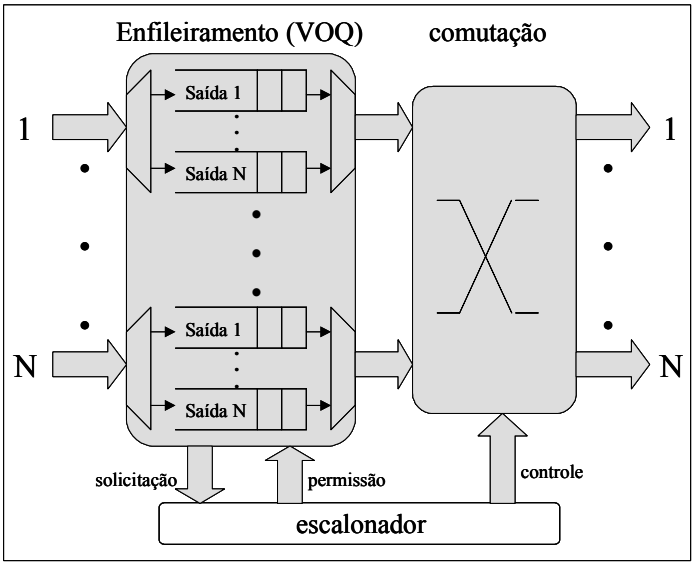

Figura 2.7- Estrutura lógica de um comutador com filas virtuais de saída.

Neste caso, cada fila de entrada mantém uma fila separada para cada saída. Assim, para um comutador *N* x *N*, cada entrada possui *N* filas lógicas do tipo FIFO, nas quais são armazenados os pacotes destinados a uma das *N* portas de saída.

Nas entradas existem N^2 VOQs, sendo normalmente a malha de comutação do tipo *crossbar*. Em cada ciclo o escalonador coleta as solicitações, executa algum algoritmo para decidir qual fila virtual deve encaminhar seu pacote, retorna com as respostas às solicitações e configura a matriz

crossbar. Uma vez que o principal objetivo da unidade de escalonamento é decidir, em cada ciclo, quais entradas deverão transmitir seus pacotes para quais saídas, ela deve encontrar o máximo de combinações entre entradas e saídas, o que pode levar a um problema de restrição de tempo devido à velocidade sempre crescente dos enlaces externos.

Como agora existem, no limite, até N^2 pacotes na cabeça da fila ao invés de N, a seleção dos pacotes que deverão ser transmitidos se torna um problema muito mais complexo. O desempenho desta estrutura será determinado pelo tipo de algoritmo utilizado. Em [25] é apresentada uma revisão bastante abrangente sobre as técnicas e algoritmos propostos na literatura até o momento, abordando desempenho, eqüidade e complexidade de implementação.

2.8 Estrutura com Filas Combinadas nas Entradas e nas Saídas

As estruturas apresentadas até agora neste capítulo utilizam somente *buffer* na entrada ou na saída. A próxima estrutura combina as duas concepções e é comumente referida como CIOQ (*Combined Input Output Queuing* - Filas Combinadas na Entrada e na Saída). Formalmente, um comutador pertence à classe de comutadores com filas nas entradas e nas saídas se a função de enfileiramento é executada antes e depois da função de comutação. O CIOQ representa portanto, um esquema de enfileiramento que combina filas nas entradas e nas saídas, como ilustra a Fig. 2.8.

Esta estrutura apresenta uma boa solução de compromisso entre desempenho e escalabilidade em relação aos comutadores só com *buffer* na entrada ou só com *buffer* na saída. Para os comutadores clássicos com *buffer* na entrada, apenas um pacote pode ser transmitido de uma determinada entrada para as portas de saída em uma unidade de tempo. Para os comutadores clássicos com *buffer* na saída, até N pacotes poderão ser entregues a uma determinada porta de saída em uma unidade de tempo. Usando CIOQ, ao invés de escolher um desses dois extremos, tem-se a opção de escolher valores mais adequados. Na Fig. 2.8, o termo S_i representa o fator de aceleração de entrada, ou seja, o número de pacotes que pode ser transmitido de uma única entrada em um ciclo, enquanto o termo S_o representa o fator de aceleração de saída, ou seja, o número de pacotes que pode ser entregue a uma única saída em um ciclo. Os termos S_i e S_o podem assumir qualquer valor entre 1 e N. $S_i = 1$ e $S_o = 1$ equivale ao comutador clássico com *buffer* na entrada e não há necessidade de se ter *buffer* na saída, enquanto $S_i = 1$ e $S_o = N$ é equivalente ao comutador clássico com *buffer* de saída.

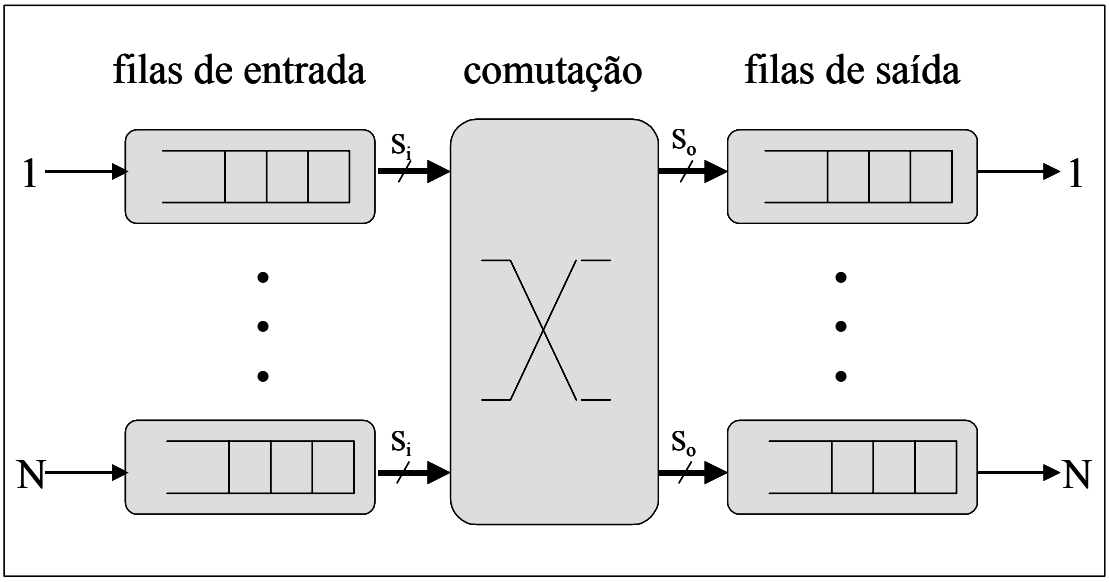

Figura 2.8- Estrutura lógica de um comutador que combina *buffer* na entrada e *buffer* na saída.

As pesquisas chegaram a esta solução híbrida através de dois caminhos: primeiro, adicionando *buffer* na saída em um comutador com *buffer* na entrada, com um modesto fator de aceleração, para melhorar o desempenho, principalmente com relação à vazão; segundo, adicionando *buffer* na entrada em um comutador com *buffer* de saída, para melhorar a taxa de perda de pacotes.

As estruturas de comutadores CIOQ podem ser classificadas com base nos seguintes critérios:

- Organização da fila de entrada: FIFO ou não FIFO, por exemplo, VOQ;
- Fator de aceleração interno: aceleração completa $S = N$, ou aceleração parcial, $1 < S < N$;
- Tamanho do *buffer* de saída: infinito ou finito. No caso de tamanho finito, os *buffers* de saída podem ser dedicados ou compartilhados. Além disto, a estrutura poderá ser com ou sem perdas, onde, no último caso, algum mecanismo de controle de fluxo interno deverá ser utilizado.

Algumas estruturas CIOQ têm sido propostas na literatura. Em [13] é analisado um modelo analítico de um sistema CIOQ com *buffer* limitado tanto na entrada quanto na saída e um fator de aceleração *S*. Dois modos de operação denominados QL (*Queue Loss* - perdas na fila) e BP (*Back-Pressure* -contrapressão) são analisados. O modo QL diz respeito à perda de pacotes nos *buffers* de saída, onde o excesso de pacotes é descartado. O modo BP é uma prevenção contra a perda de pacotes, não permitindo que mais pacotes sejam encaminhados para a fila de saída se não houver espaço disponível. Trata-se portanto, de um mecanismo de controle de fluxo interno

que, no momento em que uma fila de saída se torna completa, ela instantaneamente sinaliza as entradas, a fim de que pacotes não sejam perdidos devido ao transbordamento da fila. Neste caso, o excesso de pacotes deve ficar esperando nos *buffers* de entrada.

Uma estrutura CIOQ com *buffer* de saída de tamanho finito também é analisada em [33]. O sistema consiste de *N buffers* de saída de tamanho *b*, *N buffers* de entrada e uma estrutura de conexão entre eles para rotear os pacotes com um fator de aceleração $S = \min(b,N)$. As filas de entrada e de saída são sempre servidas no esquema FIFO. O impacto no desempenho é estudado em função dos critérios de seleção FCFS (*First Come First Served* – primeiro a chegar primeiro a ser servido), LCFS (*Last Come First Served* – último a chegar primeiro a ser servido) e aleatório. Conclui-se que a vazão não depende do critério de seleção utilizado, mas somente do tamanho do *buffer* de saída e da distribuição do tamanho do pacote. Quando a política FCFS é adotada, obtém-se o limite inferior com relação ao atraso, enquanto que, com a política LCFS, obtém-se o limite superior do atraso. Todos os resultados obtidos foram baseados em chegadas independentes e distribuição uniforme (tráfego do tipo Bernoulli).

Embora esta estrutura amenize os principais problemas encontrados nas estruturas com fila na entrada e com fila na saída, no primeiro caso com relação ao bloqueio na cabeça da fila, e no segundo com relação ao fator de aceleração total, ainda assim existe a necessidade de um fator de aceleração parcial. Vários algoritmos têm sido propostos na literatura, como mostram as referências [34] – [38], onde consegue-se um desempenho, em termos de vazão, bem próximo ao de uma estrutura com fila na saída, utilizando fator de aceleração menor que *N* ($S < N$). Todos eles são de difícil implementação prática, uma vez que o algoritmo proposto deverá operar *S* vezes mais rápido e, dependendo do algoritmo, requerendo várias interações.

A estrutura apresentada em [39] usa uma combinação de fila virtual de saída e filas de entrada e de saída combinadas. Cada porta de saída possui N^2 *buffers*, além de escalonadores na estrada e na saída. Os árbitros das entradas executam resolução de contenção nas entradas, enquanto os elementos comutadores de saída executam a resolução clássica de contenção nas saídas. Os dois maiores problemas apresentados por esta estrutura são a complexidade de interconexão da memória compartilhada e o número de acessos do sistema de fila de saída por ciclo de pacote.

A necessidade de se ter a velocidade dos enlaces internos maior do que a velocidade dos enlaces externos para adaptar portas de entrada e de saída em uma estrutura que usa filas de

entrada e de saída combinadas é apresentada em [40]. Em [41] é proposta e analisada uma estrutura com aceleração dos enlaces internos baseado na multiplexação por divisão de espaço.

2.9 Idéia da Estrutura Proposta

Neste trabalho está sendo proposta uma estrutura de um comutador *Crossbar* de Enlaces Paralelos (Comutador CEP) que utiliza filas nas entradas e nas saídas (CIOQ), e malha de comutação baseada em uma estrutura *crossbar*. Cada porta de entrada se conecta a cada porta de saída através de m linhas internas. Desta forma, os pacotes são transferidos dos *buffers* das entradas para os *buffers* das saídas através de m x N linhas internas. Como as linhas internas e externas operam com a mesma velocidade, não há necessidade de se acelerar a velocidade do relógio interno, ou seja, o fator de aceleração é igual a um ($S = 1$). Apesar de qualquer tipo de escalonamento poder ser utilizado, são propostos dois algoritmos de fácil implementação: *Round-robin* e de prioridade, operando com pacotes de tamanho fixo, como por exemplo, células ATM, e com pacotes de tamanho variável, como por exemplo, pacotes IP (*Internet Protocol*).

2.10 Conclusão

Neste capítulo foi feita uma revisão das principais estruturas de comutadores de pacotes, considerando principalmente a localização dos *buffers*. Foram apresentadas as principais vantagens e desvantagens de cada uma delas no âmbito do escopo deste trabalho. Um número significativo de referências foi apresentado, onde um estudo mais detalhado de cada uma das estruturas apresentadas pode ser encontrado. Adicionalmente, a referência [45] apresenta um elaborado estudo de desempenho das principais estruturas aqui apresentadas. Também alguns conceitos básicos relacionados a comutadores de pacotes foram revistos com o objetivo de não mais defini-los no restante deste trabalho. A idéia da estrutura proposta também foi apresentada. Pelo número de artigos publicados recentemente sobre estruturas com enfileiramento nas entradas e nas saídas, parece claro que este tipo de abordagem tem obtido uma aceitação muito grande na comunidade de pesquisa como sendo a melhor solução para os comutadores de pacotes de alta velocidade. As soluções apresentadas operam com *speed-up* o que, dependendo da velocidade do enlaces externos, pode se tornar impraticável. Nesta tese é proposta uma estrutura que utiliza enfileiramento nas entradas e nas saídas, mas sem a necessidade de acelerar a velocidade do relógio interno.

Capítulo 3

O Comutador Crossbar de Enlaces Paralelos – CEP – para Pacotes de Comprimento Fixo

3.1 Introdução

Neste capítulo é apresentada a estrutura proposta do comutador. O princípio básico de funcionamento da estrutura é apresentado considerando os pacotes com comprimento fixo, denominados neste capítulo de células ATM. A estrutura proposta inicialmente é modificada a fim de prover atendimento com Qualidade de Serviço (QoS – Quality of Service) às classes ATM. Assim, dois tipos de escalonamento são analisados: *Round-robin* e por prioridade, considerando o tráfego com distribuição de Bernoulli. Um estudo comparativo utilizando *buffers* com esquemas FIFO e *random* na estrutura proposta também é apresentado. Os resultados obtidos, analiticamente ou através de simulações, do desempenho em termos de bloqueio e tamanho médio da fila são mostrados e discutidos.

3.2 Estrutura com Escalonamento *Round-robin*

A Fig. 3.1 ilustra o comutador proposto que se baseia em uma estrutura do tipo *crossbar* e possui as seguintes características: cada entrada possui um único *buffer* do tipo FIFO enquanto cada saída possui um conjunto de *m buffers,* também do tipo FIFO, onde *m* representa o número de enlaces internos ou canais. Cada *buffer* de entrada é conectado aos *m buffers* de cada uma das saídas através de *m* enlaces físicos verticais em paralelo que operam com a mesma velocidade dos enlaces externos. Assim, até *m* células provenientes das entradas podem ser transferidas simultaneamente para uma mesma saída, sem necessidade de aceleração da velocidade interna. Cada entrada possui uma Unidade de Controle (CRT) enquanto cada saída possui um Escalonador (SCH). Uma linha reservada de um único bit conecta cada CRT a cada SCH com finalidade de Requisição ou solicitação de transmissão (Barr. REQ). Da mesma forma, uma linha reservada também de um bit conecta cada SCH a cada CRT com finalidade de Confirmação ou

permissão de transmissão (Barr. ACK). Qualquer tipo de algoritmo de escalonamento poderia ser implementado no Escalonador.

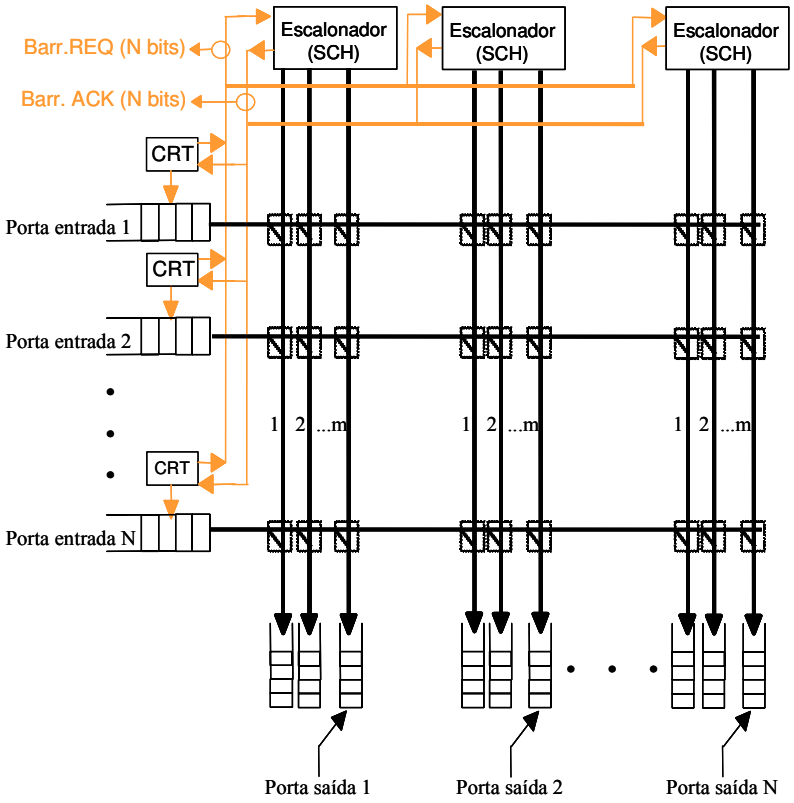

Figura 3.2- Estrutura proposta do comutador.

3.2.1 Princípio Básico de Operação

Inicialmente, em cada unidade de tempo de uma célula (*time-slot*), cada Unidade de Controle (CRT) verifica se existe alguma célula esperando para ser transmitida em seu *buffer*. Caso exista, com base no endereço da porta de saída, a CRT envia um sinal de solicitação de transmissão utilizando a linha Barr. REQ ao referido Escalonador (SCH). Cada entrada pode enviar apenas um único sinal REQ em cada *time-slot*. Caso existam mais do que *m* solicitações, provenientes de diferentes entradas, para um mesmo SCH em um mesmo *time-slot*, o SCH irá selecionar *m* entradas com base no esquema *Round-robin* e enviará os *m* sinais ACKs utilizando a linha Barr. ACK às CRTs escolhidas, sinalizando que cada CRT pode transferir sua célula para a saída durante aquele *time-slot*. A fim de garantir eqüidade entre todas as entradas, no próximo ciclo o SCH começará servindo as entradas que não foram servidas no ciclo anterior.

O SCH possui também a responsabilidade de gerenciar os *buffers* de saída a fim de evitar que uma célula que chegou depois, proveniente de uma mesma entrada, possa ser transmitida antes de uma que chegou antes, ou seja, evitar que células sejam transmitidas fora de seqüência. Isto pode ser facilmente implementado usando um esquema do tipo *Round-robin* que armazena as células na forma seqüencial nos *buffers* da saída gerenciada por aquele SCH.

Para ilustrar o princípio básico de funcionamento, seja um exemplo onde no primeiro ciclo as entradas i_1, i_2, i_3, i_4 e i_5 possuem células na cabeça de suas filas a serem transferidas para a mesma saída de endereço j. Vamos supor que a estrutura apresenta apenas três enlaces internos (m=3), como ilustra a Fig. 3.2. Após estas cinco entradas terem enviado o sinal REQ ao Escalonador j, apenas as entradas i_1, i_2 e i_3 irão receber o sinal ACK. Simultaneamente o Escalonador j irá escolher os enlaces internos 1, 2 e 3 e irá habilitar os respectivos contatos nos pontos de cruzamento, permitindo a transferência das células. As entradas i_4 e i_5 serão servidas no próximo ciclo (*time-slot*) usando os enlaces internos 1 e 2 respectivamente. Note que em cada ciclo apenas uma única célula é transmitida dos *buffers* de uma determinada saída para o enlace externo. A Fig. 3.2 ilustra a operação básica descrita anteriormente.

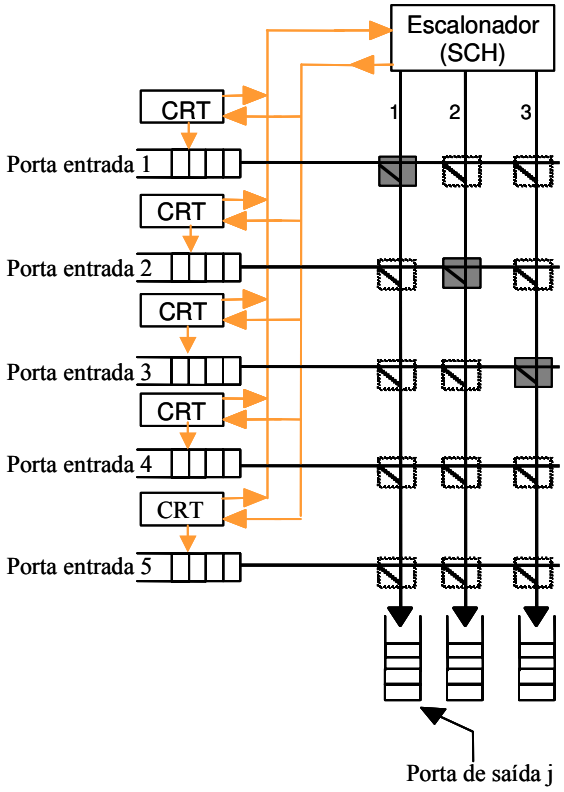

Figura 3.2 – Operação básica da estrutura proposta.

3.2.2 Análise de Desempenho

Como mostrado na seção anterior, o comutador proposto é capaz de transferir, em um mesmo *timeslot*, até *m* células das entradas para uma mesma saída, devido aos *m* enlaces internos que conectam os *buffers* de entrada aos *m buffers* de cada saída. Caso haja nos *buffers* de entrada, no mesmo *timeslot*, mais do que *m* células endereçadas a uma mesma porta de saída, algumas células só serão enviadas no ciclo seguinte, ou seja, elas terão que esperar nos *buffers* de entrada. A este fenômeno denomina-se bloqueio. Nesta seção é analisado o desempenho da estrutura proposta em termos da probabilidade de bloqueio e tamanho médio da fila nos *buffers* de entrada. Para a probabilidade de bloqueio foi proposto um modelo matemático que foi validado depois por simulação. Os resultados referentes ao comportamento da fila foram obtidos através de simulações.

3.2.2.1 Cálculo da Probabilidade de Bloqueio

Para análise da probabilidade de bloqueio foram admitidas as seguintes condições: o número de entradas e saídas é igual a N, ou seja, a estrutura possui um tamanho N x N. As chegadas das células nas entradas obedecem a um processo de Bernoulli, sendo independentes e identicamente distribuídas (i.i.d.). A probabilidade de chegada de uma célula em um *slot* é ρ e *m* é o número de enlaces internos que conecta cada *buffer* de entrada aos *m buffers* de saída. A probabilidade que uma célula seja enviada a uma particular porta de saída será ρ/N. Assim, a probabilidade que j células sejam enviadas a uma saída particular pode ser calculada por:

$$P_j = \binom{N}{j}\left(\frac{\rho}{N}\right)^j\left(1-\frac{\rho}{N}\right)^{N-j} \tag{3.1}$$

A probabilidade de que até *m* células sejam transferidas para uma determinada saída será:

$$P(m) = \sum_{j=1}^{m}\binom{N}{j}\left(\frac{\rho}{N}\right)^j\left(1-\frac{\rho}{N}\right)^{N-j} \tag{3.2}$$

O bloqueio irá ocorrer quando mais do que *m* entradas solicitarem, a uma mesma saída, permissão para transmissão. A probabilidade de bloqueio pode ser calculada por:

$$B_{av} = \sum_{j=m+1}^{N}(j-m)\binom{N}{j}\left(\frac{\rho}{N}\right)^j\left(1-\frac{\rho}{N}\right)^{N-j} \tag{3.3}$$

A partir da Eq. 3.3 foram realizados alguns exemplos numéricos. A Fig. 3.3 mostra os resultados obtidos para uma carga ρ=0.9 em cada porta de entrada para vários tamanhos da estrutura proposta. Pode-se observar que, para *m* maior do que 5, o valor da probabilidade de bloqueio é menor do que 10^{-4}. É importante salientar que, para *m*=1, a estrutura proposta opera como um comutador com *buffer* FIFO na entrada e os *buffers* de saída são desnecessários. Da Fig. 3.3, para uma estrutura com tamanho 128 x 128, os valores da probabilidade de bloqueio são $3.0528\text{x}10^{-1}$ para *m*=1, $7.7893\text{x}10^{-2}$ para *m*=2 e $2.5822\text{x}10^{-3}$ para *m*=4. Logo, a estrutura proposta apresenta, quando comparada a estrutura com *buffer* FIFO na entrada, uma probabilidade de bloqueio em torno de 4 vezes menor para *m*=2 e 120 vezes menor para *m*=4.

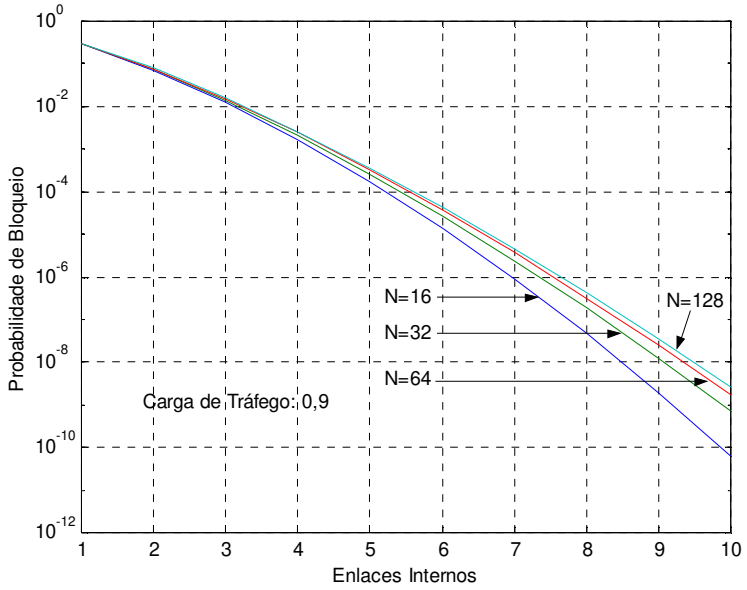

Figura 3.3 – Probabilidade de bloqueio em função do número de enlaces internos e do tamanho da estrutura.

A probabilidade de bloqueio em função do número de enlaces internos é mostrada na Fig. 3.4 para vários valores de carga de tráfego ρ, assumindo uma estrutura com N=64. Os resultados analíticos mostram que a estrutura proposta comporta-se muito bem mesmo sob uma carga intensa de tráfego (ρ=1).

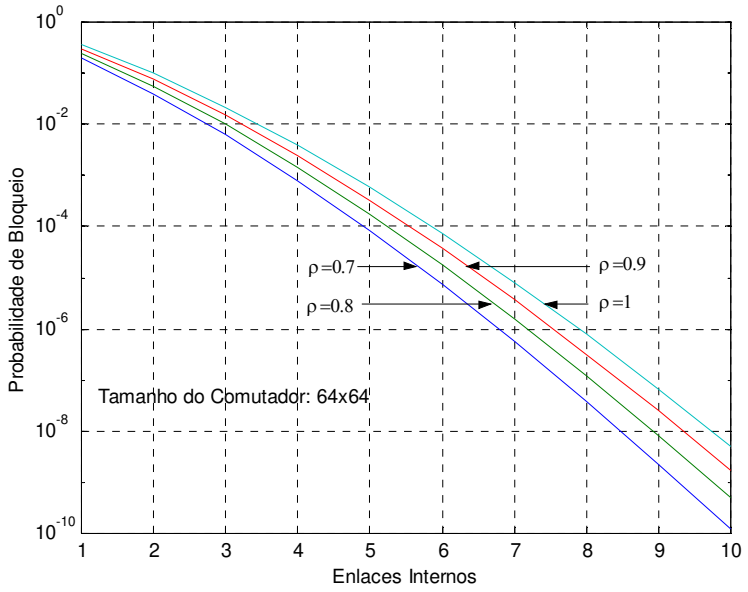

Figura 3.4 – Probabilidade de bloqueio em função do número de enlaces internos e da carga de tráfego.

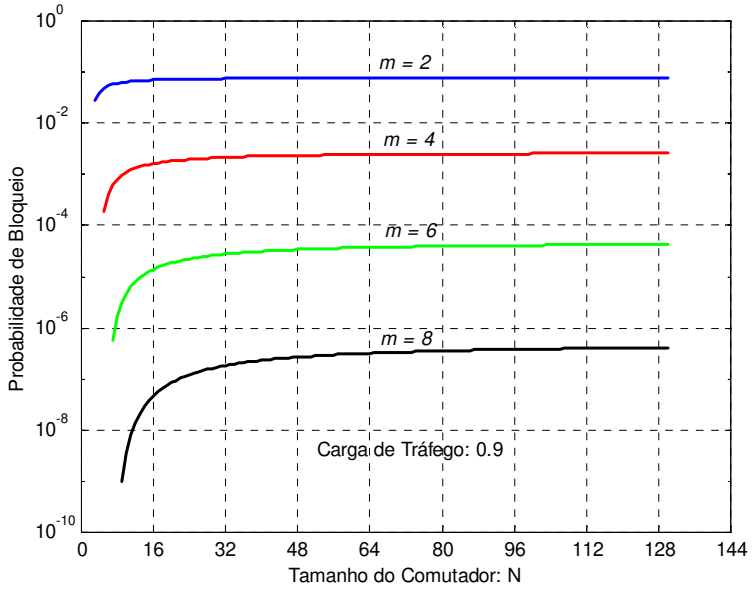

Figura 3.5 – Probabilidade de bloqueio em função do número de entradas.

A probabilidade de bloqueio aumenta com o aumento do número de portas de entrada e saída para um mesmo número de enlaces internos, como mostra a Fig. 3.5. Entretanto, ela praticamente satura a partir de N=32, supondo um tráfego randômico ρ=0.9 em cada entrada. Tal saturação já

era esperada uma vez que, para um número grande de entradas N, o tráfego do tipo binomial torna-se um tráfego do tipo Poisson, e a taxa de chegada de células é praticamente constante.

Simulação

Um estudo com base em simulação foi feito visando principalmente dois objetivos: validar o modelo matemático proposto para o cálculo da probabilidade de bloqueio e avaliar o comportamento da fila nos *buffers* de entrada, em termos de comprimento médio da fila e taxa média de perdas de célula.

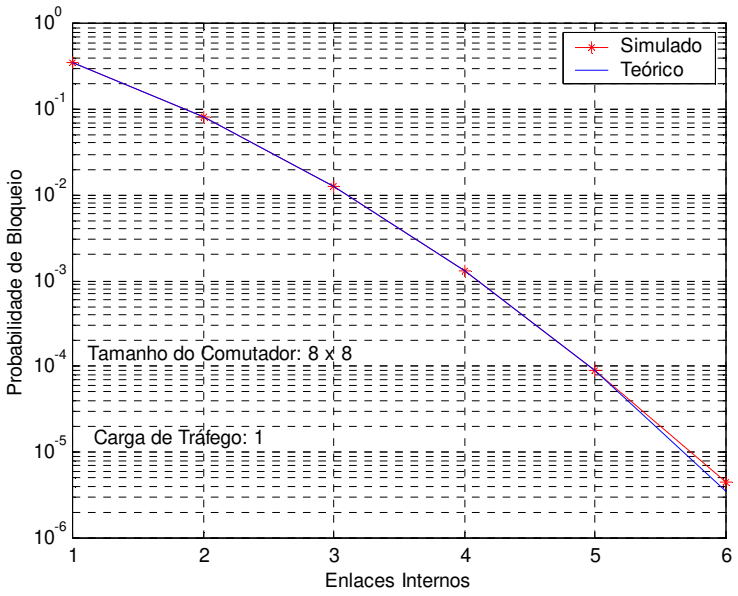

Figura 3.6 – Comparação entre os resultados do modelo matemático e da simulação.

Para executar o estudo com simulação, um programa computacional foi desenvolvido e escrito utilizando a ferramenta MATLAB. As seguintes variáveis foram utilizadas: o número de portas de entradas e saídas (N), o número de enlaces internos ($1 \le m < N$) e a carga de tráfego ($0 < \rho \le 1$). Em cada período de célula, para cada uma das entradas, o programa de simulação gera uma célula com probabilidade ρ e com um endereço de saída entre 1 e N, e armazena-a no *buffer* de entrada. O programa então transfere as células presentes na cabeça dos *buffers* de entrada para os *buffers* de saída de acordo com o número de enlaces internos, verifica o total de células da cabeça dos *buffers* que não foram transferidas para computar o bloqueio e o total de células que permanece nos *buffers* de entrada para computar o tamanho médio da fila. Este procedimento é executado a cada período de célula várias vezes, assumindo *buffers* de tamanho infinito em cada

entrada. A Fig. 3.6 mostra uma comparação entre os resultados obtidos com o modelo matemático e os obtidos com a simulação. Como pode ser observado, os resultados obtidos são praticamente os mesmos, comprovando a consistência do modelo. Devido aos pequenos valores da probabilidade de bloqueio, há necessidade de longos tempos de simulação que aumenta com a quantidade de portas e com a diminuição do fator de carga. Desse modo foram consideradas para a simulação uma estrutura 8 x 8 e carga de tráfego ρ=1.

Análise da Fila nos *Buffers* de Entrada

A Fig. 3.7 mostra os resultados da simulação para o número médio de células nas filas, ou seja, o tamanho médio da fila considerando a estrutura com N=8 e uma carga de tráfego ρ=0.9. Observa-se que para m>2 praticamente não existem células esperando nos *buffers* de entrada. Mesmo para m=2 o número de células na fila é relativamente pequeno, em média aproximadamente quatro células. Entretanto, os resultados da simulação mostraram que, para m=1 e carga ρ=0.9, o número de células na fila é muito grande devido ao problema do HOLB.

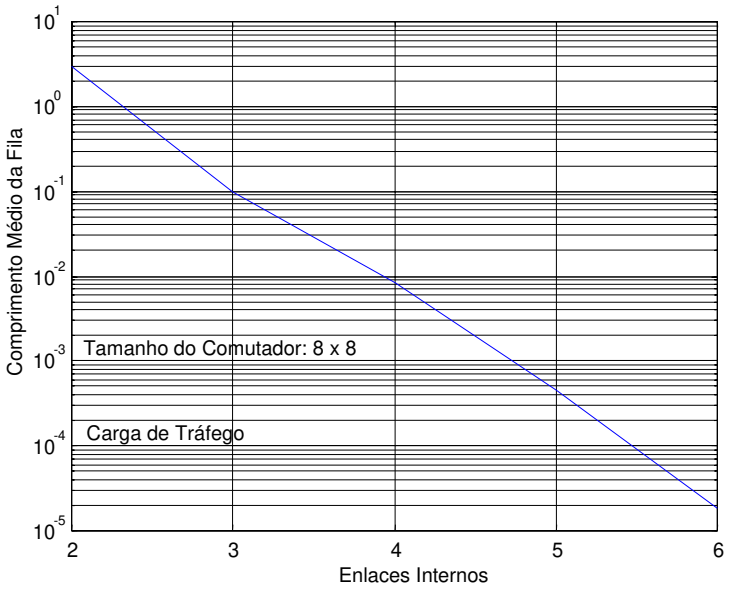

Figura 3.7 – Comprimento médio da fila nos *buffers* de entrada.

Para estimar a probabilidade de perda de células, durante a simulação foi utilizado um *buffer* finito de tamanho igual a 10, ou seja, com capacidade para 10 células em cada entrada. Os resultados da simulação mostraram que, para m=3, a probabilidade de perda de células ficou em torno de 10^{-8}, supondo uma carga ρ=0.9 e uma estrutura 64 x 64.

FIFO Buffer Versus Random Buffer

Até agora a estrutura proposta foi estudada admitindo filas do tipo FIFO nas entradas. Esquema do tipo FIFO, apesar de simples, é muito limitado, uma vez que apenas a primeira célula da fila é examinada. Podem ocorrer situações onde a segunda célula ou as células subseqüentes na fila poderiam ser encaminhadas para uma saída livre, mas se a primeira célula não foi ainda servida elas terão que esperar. Utilizando *buffer* do tipo RAM (*Random Access Memory*) em vez de *buffer* do tipo FIFO, qualquer célula poderia ser lida da memória. Este tipo de memória é denominado de *Random buffer* ou *buffer* de acesso randômico.

A estrutura proposta foi simulada utilizando *buffer* do tipo *random* em cada entrada para verificar o desempenho da mesma em termos de comprimento médio da fila e também comparar com aquela utilizando *buffer* do tipo FIFO. Na simulação cada *buffer* de entrada, com base no esquema *Round-robin*, é pesquisado na sua profundidade total, ou seja, até encontrar uma célula com destino a uma saída livre.

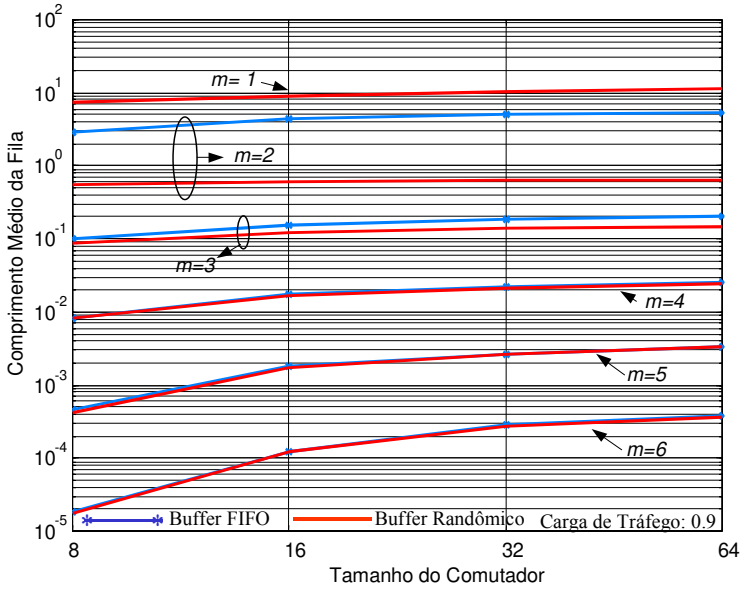

Figura 3.8 – Comparação dos resultados do comprimento médio da fila para os esquemas FIFO e *random*.

A Fig. 3.8 mostra os resultados da simulação em termos do comprimento médio da fila para a estrutura com esquema FIFO *buffer* e *random buffer*, para uma carga de tráfego ρ=0.9 e vários tamanhos de N. Para *m*=1, apenas o desempenho do esquema *random* foi mostrado no gráfico, uma vez que no esquema FIFO o tamanho da fila é muito grande. Observa-se que para *m*=1 o

esquema *random* apresenta desempenho muito bom, pois o número de células esperando na fila é muito pequeno. Em média têm-se oito células para N=8 e onze células para N=64. Para $m \geq 2$ não existe praticamente células esperando nos *buffers* de entrada. Para $m=2$ o número médio de células nos *buffers* de entrada é em torno de cinco para o esquema FIFO e 0.6 para o esquema *random*. Para $m \geq 3$ os resultados são os mesmos da Fig. 3.7, ou seja, as estruturas com esquema FIFO e *Random* apresentam os mesmos resultados. Portanto, para $m \geq 3$ praticamente não compensa utilizar o esquema *random* associado à estrutura com enlaces internos.

Como abordado no capítulo 2, várias estruturas utilizando esquema *random* têm sido propostas na literatura com o objetivo de resolver o problema do HOLB. O maior problema nessas estruturas é implementar um escalonador que encontre, em cada *time-slot*, o máximo de combinações entre as portas de entrada e saída, o que representa um grande desafio devido à crescente velocidade dos enlaces externos. Observando a Fig. 3.8, conclui-se que a estrutura proposta com esquema FIFO apresenta um desempenho melhor, mesmo com $m=2$, do que uma estrutura com *buffer* na entrada com esquema randômico ($m=1$). Além disso, a estrutura proposta apresenta um esquema de escalonamento bastante simples.

3.3 Estrutura com Escalonamento por Prioridade

Na seção 3.2 foi apresentada a estrutura proposta que utiliza um único *buffer* do tipo FIFO em cada entrada, um conjunto de *m buffers* em cada porta de saída e *m* enlaces internos que conecta cada *buffer* de entrada aos *m buffers* de cada saída. Nesta seção a estrutura proposta anteriormente é modificada a fim de prover Qualidade de Serviço (QoS – *Quality of Service*). A Fig. 3.9 mostra a nova estrutura proposta e que apresenta as seguintes características: Cada porta de entrada utiliza cinco *buffers* do tipo FIFO, uma para cada classe de serviço ATM (CBR, rtVBR, nrtVBR, ABR e UBR) e uma unidade de controle (CRT). Cada porta de saída utiliza um conjunto de 5x*m buffers*, onde *m* é o número de enlaces internos e um escalonador (SCH). É importante notar que cada enlace interno necessita de um *buffer* físico, mas a separação das classes de serviços pode ser feita em *buffers* lógicos ou memória compartilhada. Um conjunto de linhas conecta as CRTs aos SCH para transportar as solicitações de transmissão (REQ) e um outro conjunto conecta os SCH as CRTs para transportar as autorizações de transmissão (ACK). Para que o SCH possa identificar qual tipo de classe de serviço está solicitando transmissão seriam necessárias n ($n = \log_2 r$, onde r é a quantidade de classes) linhas separadas ou bits. Para a

nova estrutura proposta *r*=5 e *n*=3. As células que chegam às portas de entrada são discriminadas de acordo com a classe de serviço e armazenadas nos respectivos *buffers*. Também neste caso qualquer tipo de algoritmo de escalonamento poderia ser implementado no escalonador.

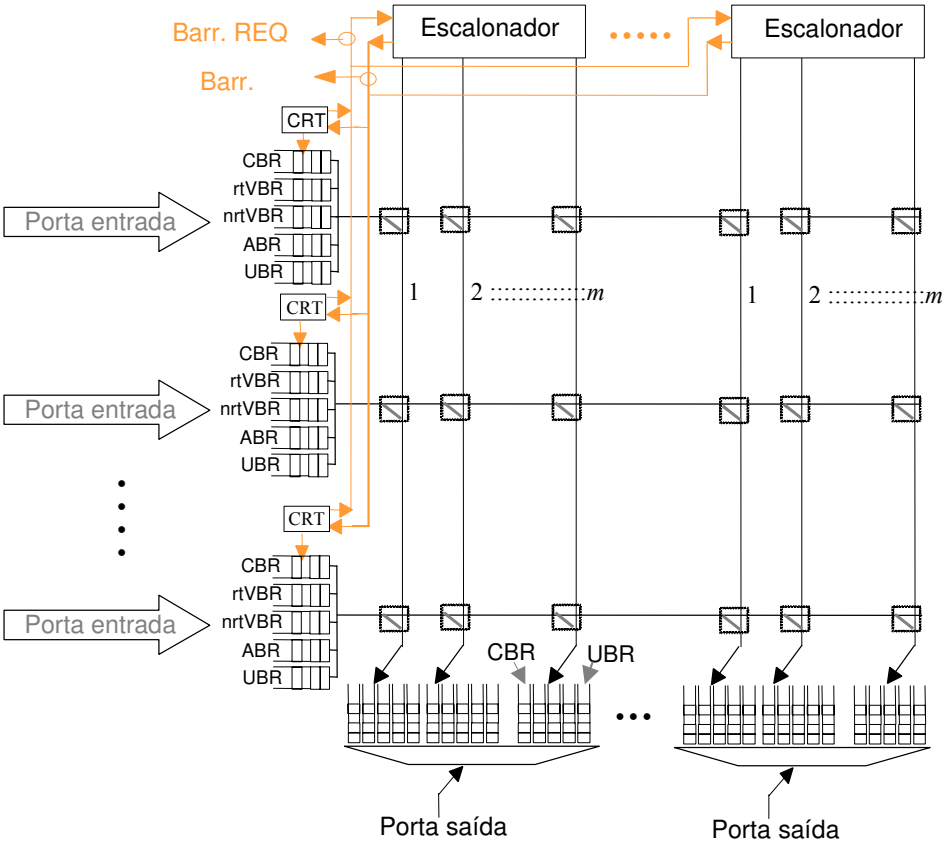

Figura 3.9 – Estrutura proposta modificada para prover Qualidade de Serviço.

3.3.1 Princípio Básico de Operação

A operação da nova estrutura proposta depende do tipo de algoritmo de escalonamento utilizado no escalonador. Existem vários algoritmos propostos na literatura para diferentes aplicações. Neste trabalho é adotado um algoritmo simples de prioridade sem preempção, a fim de ilustrar o funcionamento básico.

No esquema de prioridade, as células que chegam em cada porta de entrada são separadas e armazenadas nas filas apropriadas de cada classe de serviço. A classe de serviço CBR possui a prioridade mais alta, seguida pela classe rtVBR, enquanto a classe UBR possui a prioridade mais baixa. Primeiramente, a CRT de cada entrada verifica se existe alguma célula no *buffer* da classe CBR a ser transmitida. Se alguma célula estiver esperando na fila da classe CBR, a CRT envia ao SCH da saída a qual a célula deverá ser encaminhada um sinal código CBR (três bits, por

exemplo, 111) usando a linha REQ. Se existirem mais do que *m* solicitações para a classe CBR, a um mesmo SCH, no mesmo *time-slot*, o SCH irá escolher *m* entradas com base no esquema *Round-robin* e enviará o sinal de autorização através da linha ACK (um único bit) as CRTs escolhidas. Se não existir célula no *buffer* da classe CBR, a CRT examina então o *buffer* da classe rtVBR. Caso exista alguma célula esperando nesse *buffer*, o CRT enviará um sinal código rtVBR (por exemplo, 110) ao SCH correspondente utilizando a linha REQ. Da mesma forma, se existirem mais do que *m* solicitações no mesmo *time-slot,* a um mesmo SCH, apenas *m* entradas serão servidas, escolhidas com base no esquema *Round-robin*. Como esta seqüência é executada em cada porta de entrada, a classe de serviço UBR só será servida se não existir nenhuma outra classe a ser servida. Se um escalonador receber mais do que *m* solicitações de transmissão de diferentes classes de serviços, em um mesmo *time-slot*, ele irá servir apenas *m* entradas obedecendo à disciplina de prioridade das classes de serviços.

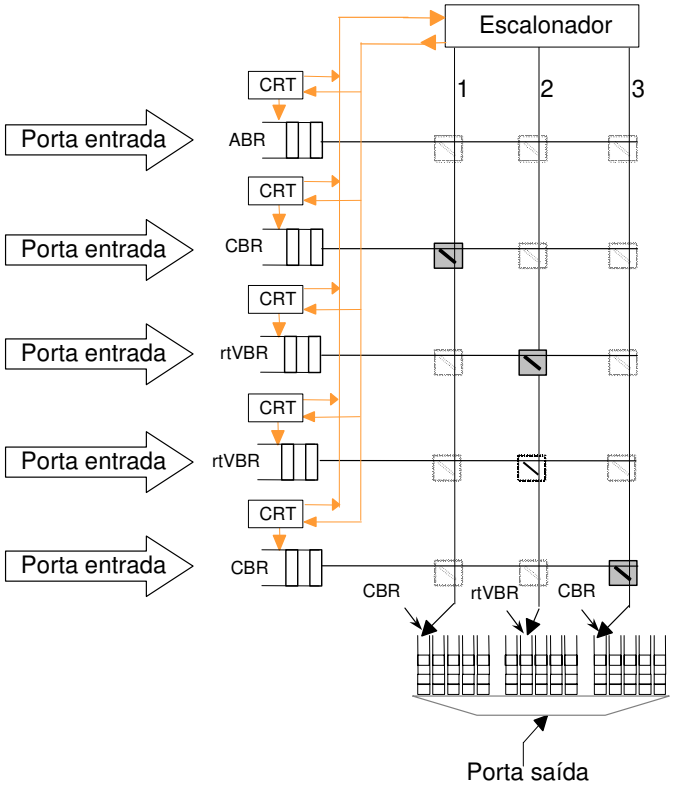

Figura 3.10 – Operação básica da estrutura modificada para prover QoS.

Como exemplo de operação, vamos supor que as portas de entrada i_1, i_2, i_3, i_4 e i_5 possuem em seus *buffers* células na cabeça da fila com classes de serviços ABR, CBR, rtVBR e CBR, respectivamente, endereçadas à mesma porta de saída j. Vamos supor ainda que o número de enlaces internos seja três (*m*=3). Após as CRTs destas entradas terem enviado os respectivos

códigos das classes de serviços ao escalonador da porta de saída j, através das linhas REQ, o SCH j irá selecionar as entradas com base nos códigos das classes recebidos nas solicitações de transmissão, e apenas as entradas i_1, i_3 e i_5 irão receber o sinal de permissão ACK. Esta operação está ilustrada na Fig. 3.10.

3.3.2 Análise de Desempenho do Comprimento da Fila

Para estudar o desempenho da nova estrutura proposta, em relação ao comprimento médio das filas para cada classe de serviço, tanto nos *buffers* de entrada quanto nos de saída, foi desenvolvido e escrito um programa utilizando o ambiente MATLAB.

Modelo de Simulação

Devido à simetria da nova estrutura proposta, o modelo de simulação pode ser simplificado com base em uma única saída genérica, como ilustra a Fig. 3.11.

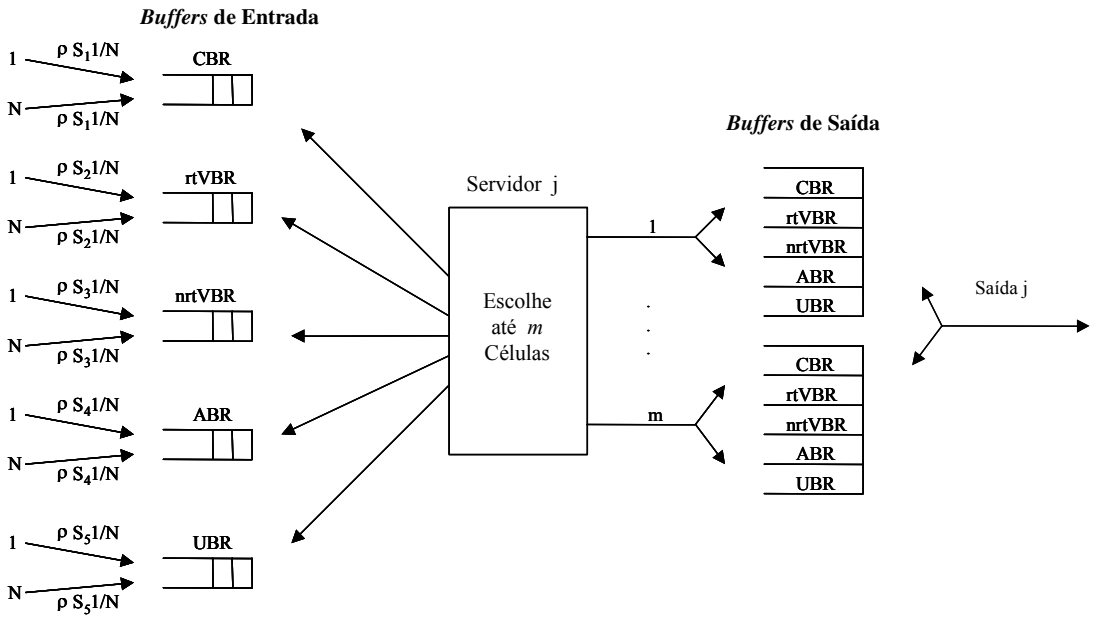

Figura 3.11 – Modelo de simulação para a estrutura com classe de serviço.

No modelo proposto são considerados cinco *buffers* de entrada, um para cada classe de serviço. Em cada *buffer* de classe de serviço as células chegam das N portas de entrada com uma taxa λ_i. Para a simulação foram supostas as seguintes condições: as chegadas das células nas entradas obedecem à distribuição de Bernoulli (i.i.d.), a probabilidade de chegada de uma célula em um *slot* é ρ, o percentual de tráfego da classe de serviço i é S_i (i=1,2...5 e $\Sigma_i S_i = 1$) e a

probabilidade de que uma célula seja enviada a uma porta de saída em particular é 1/N. Assim, a taxa λ_i em cada *buffer* de classe de serviço será $\lambda_i = N.\rho.S_i.\frac{1}{N}$.

Em cada *time-slot* o servidor j procura inicialmente por uma célula para transmitir no *buffer* de classe CBR, que é o de mais alta prioridade, e pode servir até *m* células simultaneamente. Se não existir nenhuma célula ou se a quantidade de células for menor do que *m* no *buffer* de classe CBR, o servidor irá verificar o próximo *buffer* ou *buffers* na seqüência rtVBR, nrtVBR, ABR e UBR, até o limite de *m* células.

Os *buffers* de saída em cada um dos *m* enlaces internos são divididos em cinco classes de serviço e as células são transmitidas para a porta de saída j obedecendo à ordem de prioridade. Primeiro, todas as células do *buffer* de classe CBR dos *m buffers* de saída são transmitidas. Depois transmite todas as células da classe rtVBR dos *m buffers* de saída, e assim sucessivamente.

Desempenho no *Buffer* de Entrada

O resultados obtidos através da simulação para o número médio de células na fila nos *buffers* de entrada, para cada uma das classes de serviços em função da quantidade de enlaces internos são mostrados na Fig. 3.12, considerando uma estrutura 16 x 16 e uma carga de tráfego ρ=0.9.

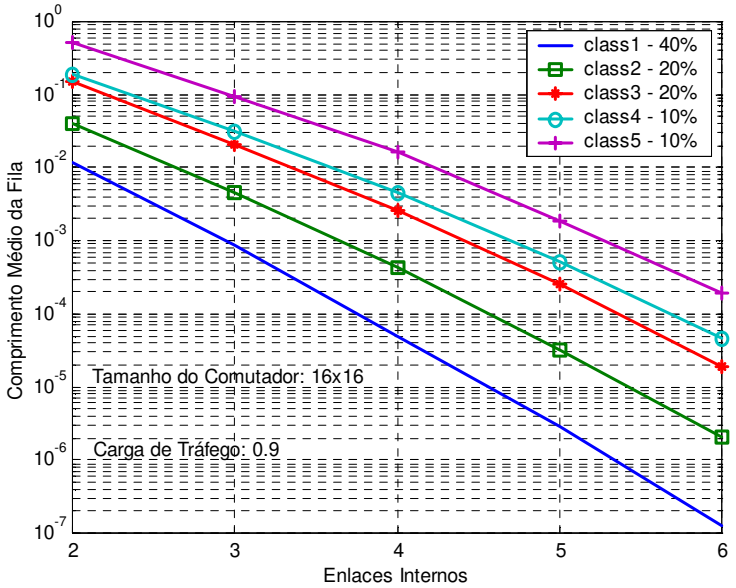

Figura 3.12 – Comprimento médio da fila para cada classe de serviço nos *buffers* de entrada

O percentual de tráfego das classes de serviços são 40%, 20%, 20%, 10% e 10% para as classes CBR (Class1), rtVBR (Class2), nrtVBR (Class3), ABR (Class4) e UBR (Class5), respectivamente. Como já era esperado, os resultados da simulação mostram que o número médio de células em espera em cada *buffer* de classe de serviço é muito pequeno. Para m≥3 praticamente não existem células esperando nos *buffers* de entrada independente da classe de serviço. Mesmo para *m*=2 a classe UBR, que é a de mais baixa prioridade, apresenta um número pequeno de células na fila. Em média menos do que uma célula em espera.

Desempenho no *Buffer* de Saída

Nos *buffers* de saída o desempenho em relação ao número médio de células para cada classe de serviço é avaliado em termos do número de enlaces internos e também em função da carga ρ. Os resultados da simulação são mostrados na Fig. 3.13, considerando uma estrutura 16 x 16 e carga ρ=0.9. Observa-se que apenas a classe de menor prioridade UBR apresenta um número médio de células em espera maior do que uma célula, em média 2.5 células. Nota-se ainda que aumentando o número de enlaces internos *m* não há praticamente aumento no número médio de células esperando no *buffer*. Para as classes 1 a 4, o número médio de células em cada *buffer* de saída é muito pequeno, menos de uma célula em média. Pode-se concluir, para o tipo de tráfego em questão, que três enlaces internos (*m*=3) são suficientes para uma transferência e transmissão rápida de células.

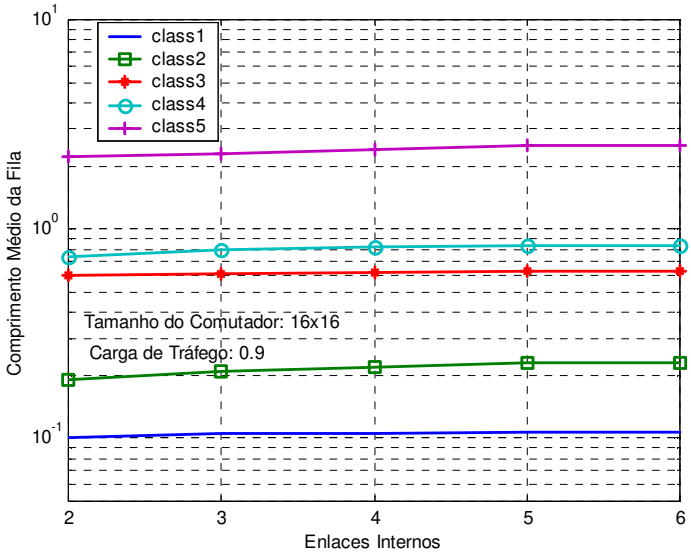

Figura 3.13 – Comprimento médio da fila para cada classe de serviço nos *buffers* de saída.

Na Fig. 3.14 o desempenho em relação ao comprimento médio da fila nos *buffers* de saída para as classes CBR (S1), rtVBR (S2), nrtVBR (S3), ABR (S4) e UBR (S5) é avaliado em função da variação da carga ρ, considerando uma estrutura 16 x 16 e um número de enlaces internos *m*=6, que representa o pior caso em termos de células esperando nos *buffers* de saída, uma vez que mais células são transferidas simultaneamente. Praticamente não existem células nos *buffers* de entrada e quase todas as células estão esperando nos *buffers* de saída. Os resultados da simulação mostram que, para uma carga de tráfego de até 80%, em média tem-se menos de uma célula esperando na fila. Para uma carga acima de 80%, apenas a classe UBR apresenta alguma instabilidade.

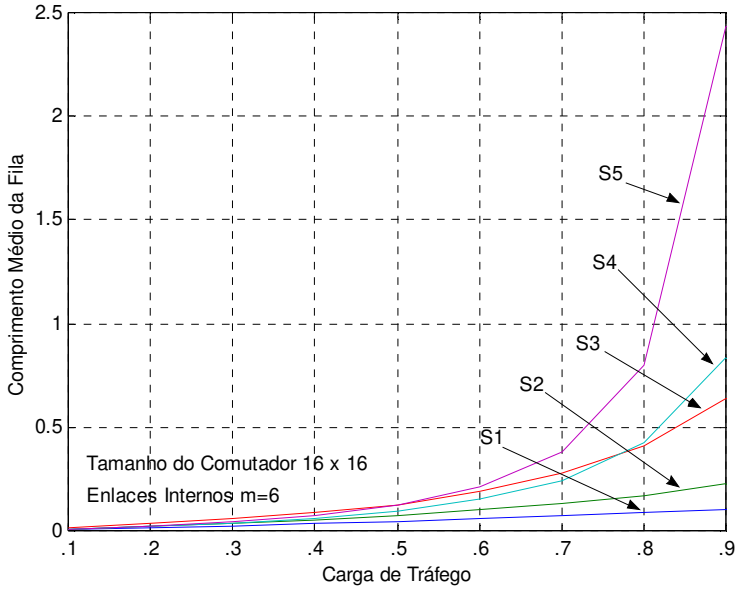

Figura 3.14 – Comprimento médio da fila para cada classe de serviço em função da carga ρ.

3.4 Conclusão

Na primeira parte deste capítulo foi apresentada a estrutura proposta do comutador de pacotes de tamanho fixo (ATM) que utiliza uma combinação de *buffers* na entrada e na saída. Cada entrada possui um único *buffer* do tipo FIFO enquanto cada saída possui *m buffers*. Conectando cada *buffer* de entrada aos *buffers* de saída existem *m* enlaces físicos internos. Devido ao uso de *m* enlaces internos, mesmo utilizando *buffers* do tipo FIFO nas portas de entrada há aumento da vazão por causa da diminuição da probabilidade de bloqueio na cabeça da fila. Os resultados obtidos com a simulação mostraram que, para *m*=4, são necessários *buffers* com apenas 10

comprimentos de célula para conseguir uma probabilidade de perda de célula muito baixa, admitindo uma carga de tráfego ρ=0.9, e independente da quantidade de portas do comutador. Adicionalmente, o esquema utilizado no escalonamento não é complicado, mostrando que a estrutura proposta com $m \geq 3$ é mais eficaz na redução do problema do HOLB do que as estruturas com esquema de escalonamento do tipo *random*.

Na segunda parte deste capítulo a estrutura proposta inicialmente foi modificada para prover qualidade de serviço às classes de serviço ATM. Cada entrada possui um conjunto de cinco *buffers* do tipo FIFO, enquanto cada saída possui um conjunto de mx5 *buffers*. Na estrutura modificada, as células que chegam nas entradas são discriminadas com base nas classes de serviço e armazenadas nos *buffers* apropriados. Essas células são transferidas para os *buffers* de saída por um conjunto *m*xN enlaces internos. Utilizando um esquema simples de escalonamento com prioridade, os resultados obtidos com a simulação mostram que apenas três enlaces internos (m=3) para cada saída são suficientes para transferir rapidamente as células dos *buffers* de entrada para os de saída. Os resultados mostram ainda que o comprimento médio da fila de células nos *buffers* de saída pode ser mantido muito pequeno para qualquer classe de serviço, com base no tipo de tráfego apresentado.

Todos os resultados mostrados neste capítulo foram também comprovados e confirmados no trabalho de dissertação de mestrado da referência [12], utilizando o *software* de simulação de eventos discretos ARENA 5.0 (*Rockell Software*). Além desses resultados, foi analisado o desempenho da fila de saída para as estruturas com escalonamento *Round-robin* e por prioridade. Outros dados estatísticos como variância, desvio padrão, comprimento máximo, etc, foram obtidos para as filas de entrada e de saída nas estruturas com os dois tipos de escalonamento.

Tanto a estrutura proposta inicialmente, quanto a estrutura modificada para atender aplicações com qualidade de serviço, utilizam *m* enlaces internos que operam à mesma velocidade dos enlaces externos, não havendo necessidade de aceleração da velocidade interna, fazendo com que estas estruturas sejam adequadas às aplicações de comutadores de alta velocidade.

Capítulo 4

O Comutador Crossbar de Enlaces Paralelos – CEP – para Pacotes de Comprimento Variável

4.1 Introdução

No capítulo anterior, além de apresentar a estrutura proposta do comutador, foi feita uma análise do seu desempenho, considerando pacotes de tamanho fixo, ou seja, células ATM. A análise foi feita considerando os esquemas de escalonamento *Round-robin* e com prioridade, onde a estrutura proposta inicialmente é modificada para prover qualidade de serviço.

Neste capítulo é estudado o desempenho das estruturas anteriores considerando agora pacotes de tamanho variável. No esquema de escalonamento *Round-robin* é analisado o desempenho para dois tipos de tráfegos, Poisson e HTTP. No esquema de escalonamento com prioridade, apenas o tráfego do tipo Poisson é tratado. Os resultados obtidos, analiticamente utilizando teoria de filas ou através de simulações, em termos de tempo médio de espera e tamanho médio da fila, são também mostrados e discutidos.

4.2 Estrutura com Escalonamento *Round-robin*

A Fig. 4.1 é a mesma figura apresentada no capítulo anterior (Fig. 3.1), e repetida aqui para facilitar o acompanhamento das explicações. No caso de pacotes de tamanho fixo, as operações de solicitação de transmissão de um pacote da entrada para a saída, permissão para transmissão e a própria transmissão ocorriam no intervalo de um *slot*, ou seja, existiam instantes bem definidos para a solicitação, permissão, início e término da transmissão do pacote, caracterizando um sincronismo. Os escalonadores sabiam exatamente quando os enlaces internos ficavam livres. No caso de pacotes de tamanho variável e aleatório tal sincronismo deixa de existir pois as solicitações, as permissões, início e término das transmissões podem ocorrer a qualquer instante. Desta forma, além das informações de solicitação e permissão, uma outra informação adicional de controle será necessária para que o escalonador fique sabendo quando é que um determinado

enlace interno tornou-se disponível. A própria linha REQ poderá ser utilizada para este fim, como mostrado a seguir.

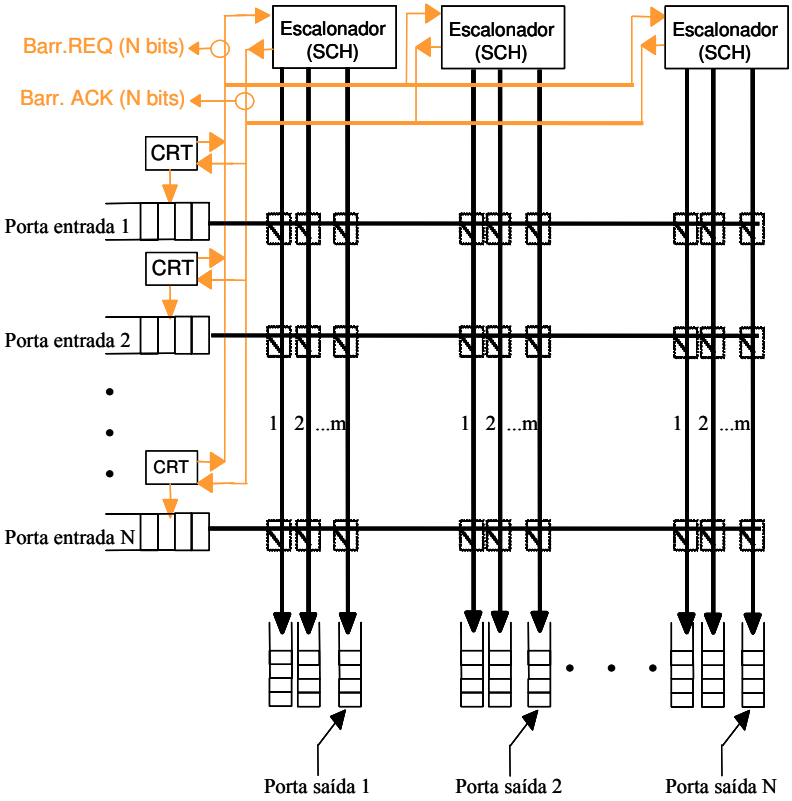

Figura 4.3 - Estrutura proposta do comutador.

Novamente, cada porta de entrada possui um único *buffer* do tipo FIFO, enquanto cada porta de saída possui um conjunto de *m buffers* também do tipo FIFO. Cada *buffer* de entrada é conectado aos *m buffers* de saída através de *m* enlaces físicos. Cada porta de entrada possui uma unidade de controle (CRT), enquanto cada porta de saída possui um escalonador (SCH) com finalidades de controle. Cada CRT é conectado a cada SCH através de uma linha reservada para transportar a informação de solicitação (REQ). A informação REQ é enviada do CRT ao correspondente SCH quando um novo pacote tem que ser transportado do *buffer* de entrada para uma saída. O escalonador, ao receber os pedidos de REQs dos CRTs, utiliza um algoritmo de agendamento para escolher até *m* CRTs para transmitir os seus pacotes aos *m buffers* de saída. Aos CRTs escolhidos o SCH envia um sinal de confirmação (ACK) através de uma linha reservada. Ao terminar a transmissão do seu pacote, o CRT envia um outro sinal de informação

de fim de transmissão (EOT) ao SCH. Este sinal pode ser enviado usando as mesmas linhas usadas para transmitir o sinal REQ.

4.2.1 Princípio Básico de Operação

Para exemplificar a operação do comutador proposto, vamos supor que, no mesmo instante, as portas de entradas i_1, i_2, i_3, i_4 e i_5 possuem em seus *buffers* pacotes endereçados à mesma porta de saída de endereço j. Vamos supor ainda que o número de enlaces internos seja igual a três (m=3). Usando, por exemplo, o esquema de agendamento do tipo *Round-robin*, após as entradas anteriores terem enviado o sinal REQ ao escalonador j, apenas as entradas i_1, i_2 e i_3 receberão o sinal ACK. Simultaneamente, o escalonador j irá escolher os enlaces internos 1, 2 e 3 e habilitará os respectivos pontos de cruzamentos. Assim que uma das portas i_1, i_2 ou i_3 terminar a sua transmissão e enviar o sinal de fim de transmissão (EOT) para o SCH, a entrada i_4 receberá o sinal ACK e irá ser servida usando um enlace interno livre. A Fig. 4.2 mostra o princípio básico de operação descrito acima.

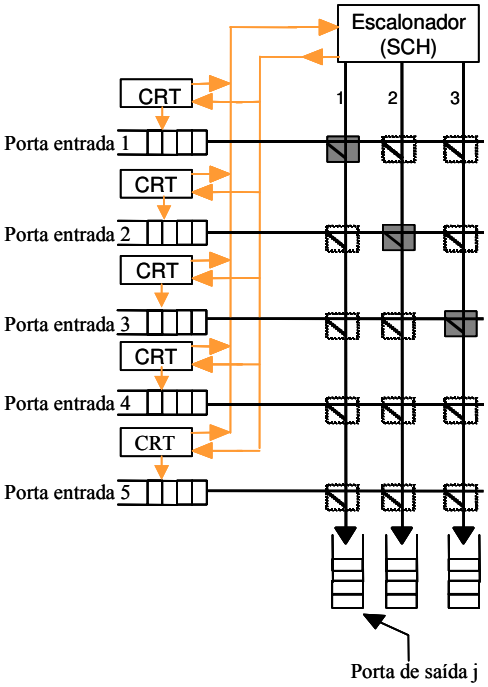

Figura 4.2 – Operação básica da estrutura proposta.

O algoritmo de escalonamento usado no *buffer* das portas de saída pode ser independente do algoritmo de escalonamento presente no escalonador SCH, mas combinando ambos escalonadores é possível obter algoritmos de agendamento complexos e flexíveis a fim de garantir qualidade de serviço para cada serviço.

4.3 Estrutura com Escalonamento por Prioridade

A Fig. 4.3 mostra a estrutura do comutador, usando escalonamento por prioridade, como um exemplo para satisfazer qualidade de serviço. Qualquer tipo de algoritmo pode ser utilizado no escalonador, mas devido à simplicidade foi escolhido um escalonamento por prioridade sem preempção.

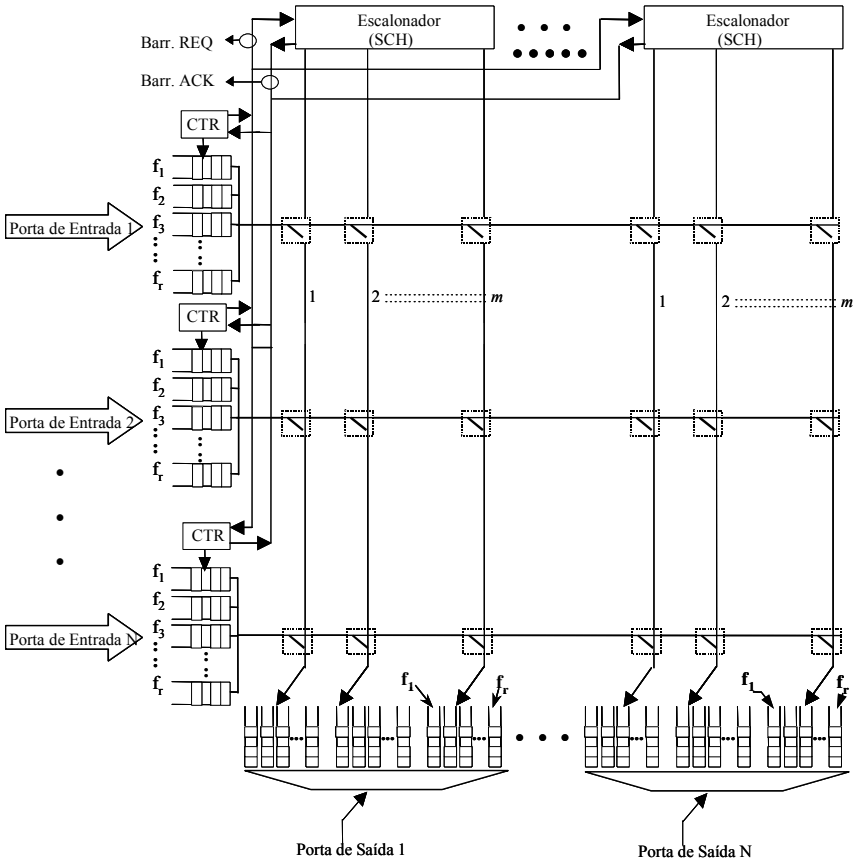

Figura 4.3 – Estrutura proposta do comutador de pacotes para atender QoS.

Cada porta de entrada possui r *buffers* do tipo FIFO. Linhas REQ e ACK conectam unidades de controle (CRT) e escalonadores (SCH) para transportarem informações de solicitação e confirmação. Cada porta de saída necessita de rx*m buffers*, onde *m* representa o número de enlaces internos. É importante notar que cada enlace interno necessita de um *buffer* físico, mas a separação das classes de fluxos pode ser feita por *buffer* lógico (memória compartilhada). A figura mostra ainda que cada porta de saída necessita de um escalonador por prioridade.

Os pacotes que chegam às portas de entrada são discriminados e armazenados nas filas apropriadas de cada classe de fluxo. A classe de fluxo f_1 possui a maior prioridade, seguida pela classe f_2 e assim sucessivamente. A classe f_r é a que possui a menor prioridade. Cada CRT envia

um código apropriado, de acordo com a classe de fluxo para o SCH, usando as linhas REQ. Desta forma, o SCH possui todas as informações necessárias para escolher, de acordo com a ordem de prioridade, o pacote a ser transmitido. Se existirem mais do que *m* solicitações simultâneas, o SCH irá selecionar *m* pacotes por ordem de prioridade e enviará o sinal ACK para os CRTs escolhidos, a fim de que eles possam transmitir os seus pacotes para o *buffer* de saída. Em cada porta de saída um escalonador transmite os pacotes na ordem de prioridade.

4.3.1 Princípio Básico de Operação

Como um exemplo de operação, vamos supor que as portas de entrada i_1, i_2, i_3, i_4 e i_5 possuem em seus *buffers* pacotes na cabeça da fila com classes de fluxos f_4, f_1, f_2, f_3 e f_1 respectivamente, endereçados à mesma porta de saída de endereço j. Suponhamos ainda que o número de enlaces internos seja igual a três (*m*=3). Após receber o sinal REQ das portas i_1 a i_5, pelo critério de prioridade o SCH da saída j irá selecionar as portas de entrada i_2, i_3 e i_5 e enviará o sinal ACK a elas. Esta operação é ilustrada na Fig. 4.4.

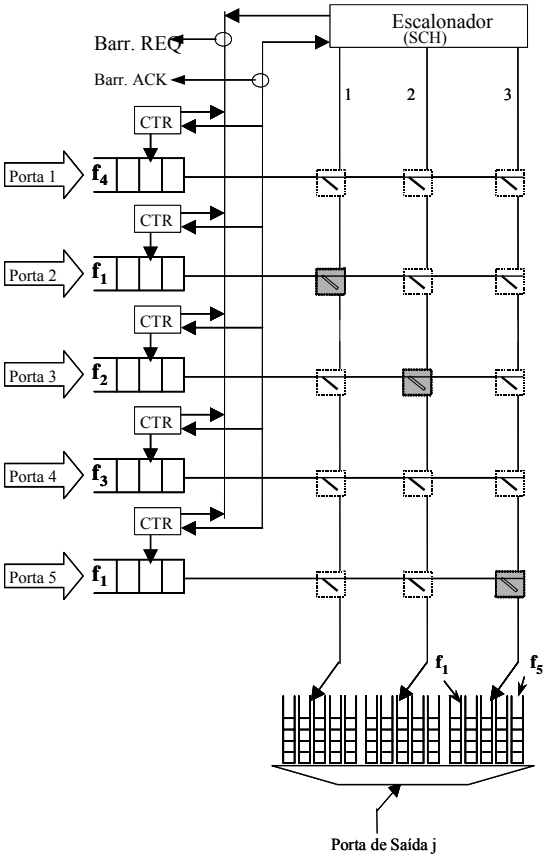

Figura 4.4 – Princípio básico de operação do comutador de pacotes com QoS.

4.4 Análise de Desempenho para Fonte de Tráfego do Tipo Poisson

Nesta seção são analisados o desempenho da estrutura do comutador com escalonamento *Round-robin* e por prioridade, em relação ao comportamento da fila em termos de número médio de pacotes (comprimento médio da fila) e tempo médio de espera, considerando fontes de tráfego do tipo Poisson. São propostos modelos para as estruturas com escalonamento *Round-robin* e por prioridade. A análise de desempenho é feita analiticamente para os dois tipos de escalonamento e também por simulação para o escalonamento *Round-robin*.

4.4.1 Modelo da Estrutura do Comutador com Escalonamento *Round-robin* para Análise de Desempenho

Como a estrutura *crossbar* é simétrica, para a modelagem da estrutura do comutador podemos considerar apenas uma saída genérica j, como mostra a Fig. 4.2. Para a porta de saída j, as filas de entrada são filas distribuídas e podem ser modeladas como sendo uma única fila e os *m* enlaces como sendo *m* servidores. Na porta de saída, o conjunto das *m* filas pode ser também considerado como uma única fila. Desta forma, o modelo completo do comutador pode ser representado por uma rede aberta de filas, como mostra a Fig. 4.5.

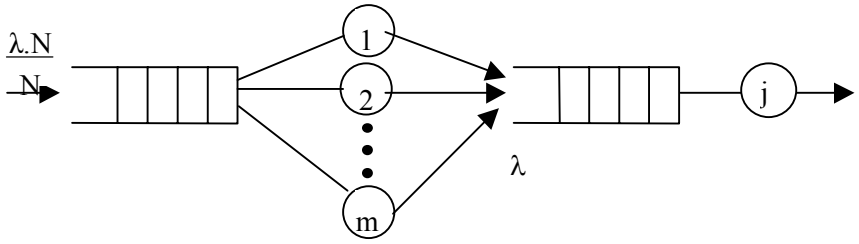

Figura 4.5 – Modelo da estrutura do comutador para análise de desempenho.

4.4.1.1 Análise das Filas

Para a análise foram feitas as seguintes considerações: o número de enlaces de entrada ou de saída é N, a chegada dos pacotes em cada enlace de entrada obedece a uma distribuição de Poisson com taxa média de chegada igual a λ, e o comprimento dos pacotes é exponencialmente distribuído com média $1/\mu$. A probabilidade de um pacote em uma entrada ser encaminhado a uma determinada saída é 1/N. Os pacotes são servidos em um esquema FIFO. Assumindo as considerações anteriores, a rede aberta de filas da Fig. 4.5 pode ser modelada como uma fila

M/M/*m* na entrada, e como uma fila M/M/1 na saída. Utilizando o teorema de Jackson para uma rede aberta de filas, a fila de entrada pode ser considerada independente da fila de saída.

Para um sistema de fila M/M/*m*, o cálculo do tamanho médio da fila E{Nq} e o tempo médio de espera na fila E{Tq} são bem conhecidos. O tamanho médio da fila é dado por:

$$E(Nq) = \frac{\rho^{m+1} / m}{m!(1-\rho/m)^2}.p_0 \qquad (4.1)$$

onde $\rho = \frac{\lambda}{\mu}$ representa a carga por enlace de entrada ou de saída e

$$p_0 = \left[\sum_{n=0}^{m-1} \frac{\rho^n}{n!} + \frac{m.\rho^m}{m!(m-\rho)}\right]^{-1}$$

O tempo médio de espera na fila E{Tq} pode ser calculado pelo teorema de Little.

$$E(Tq) = \frac{E(Nq)}{\lambda} \qquad (4.2)$$

O número médio de pacotes no sistema, E{N} e o tempo médio de espera no sistema (fila + atendimento) E{T} são dados por:

$$E[N] = \rho + E\{Nq\} = \rho + \frac{\rho^{m+1} / m}{m!(1-\rho/m)^2}.p_0, \qquad \text{e}$$

$$E\{T\} = \frac{E\{N\}}{\lambda} = \frac{1}{\mu} + E\{Tq\} \qquad (4.3)$$

Para um sistema de fila M/M/1, que é um caso particular do sistema M/M/*m* com *m*=1, as expressões para o número médio de pacotes na fila E{Nq} e tempo médio de espera na fila E{Tq} são dadas por:

$$E(Nq) = \frac{\rho^2}{1-\rho} \qquad \text{e} \qquad (4.4)$$

$$E(Tq) = \frac{E(Nq)}{\lambda} \qquad (4.5)$$

O número médio de pacotes E{N} e o tempo médio de espera no sistema E{T} podem ser calculados por:

$$E\{N\} = \frac{\rho}{1-\rho} \qquad \text{e} \tag{4.6}$$

$$E\{T\} = \frac{E\{N\}}{\lambda} = \frac{1}{\mu - \lambda} = \frac{1}{\mu} + E\{Tq\} \tag{4.7}$$

Exemplo Numérico:

O número médio de pacotes em uma fila de entrada em função do número de enlaces internos, para vários valores de carga de tráfego ρ, é mostrado na Tabela 4.1. Podemos observar que para $m \geq 2$ praticamente não existem pacotes esperando nos *buffers* de entrada. Mesmo sob uma carga de tráfego intensa (ρ=0.9) e m=2, o número de pacotes nos *buffers* de entrada é pequeno, em torno de 0.23 pacotes em média. Os resultados mostrados na primeira coluna (m=1) na Tabela 4.1 representam o número médio de pacotes esperando nos *buffers* de saída.

Tabela 4.1 - Número médio de pacotes nos *buffers* de entrada em função da carga de tráfego.

	m=1	m=2	m=3	m=4	m=5	m=6
ρ=.1	1.1111e-02	2.5063e-04	5.3795e-06	9.9149e-08	1.5702e-09	2.1661e-11
ρ=.2	5.0000e-02	2.0202e-03	8.3542e-05	3.0239e-06	9.4761e-08	2.5961e-09
ρ=.3	1.2857e-01	6.9054e-03	4.1152e-04	2.1916e-05	1.0187e-06	4.1556e-08
ρ=.4	2.6667e-01	1.6667e-02	1.2688e-03	8.8271e-05	5.4065e-06	2.9184e-07
ρ=.5	5.0000e-01	3.3333e-02	3.0303e-03	2.5786e-04	1.9500e-05	1.3054e-06
ρ=.6	9.0000e-01	5.9341e-02	6.1644e-03	6.1520e-04	5.5107e-05	4.3905e-06
ρ=.7	1.6333e+0	9.7721e-02	1.1237e-02	1.2770e-03	1.3165e-04	1.2132e-05
ρ=.8	3.2000e+0	1.5238e-01	1.8921e-02	2.3952e-03	2.7821e-04	2.9041e-05
ρ=.9	8.1000e+0	2.2853e-01	3.0012e-02	4.1600e-03	5.3551e-04	6.2303e-05

Sendo a capacidade do enlace C bits por segundo (bps) e o comprimento médio do pacote L bits, tem-se que a taxa média de atendimento será $\mu = \frac{C}{L}(\text{pcts/seg})$. A fim de calcular o tempo médio de espera por pacote, foi assumido que a capacidade do enlace é de 100Mbits por segundo e o comprimento médio do pacote é de 8000bits. A Fig. 4.6 mostra os resultados obtidos a partir das equações 4.1 e 4.2.

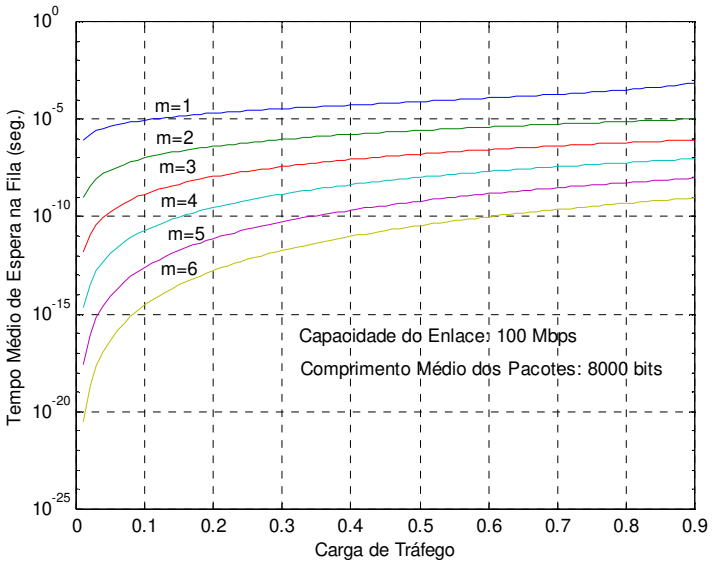

Figura 4.6 – Tempo médio de espera na fila em função da carga de tráfego.

Podemos observar que o tempo médio de espera na fila é muito pequeno. Mesmo para o pior caso, m=2 e sob tráfego intenso (ρ=0.9), o tempo médio de espera é em torno de 10 microsegundos. Para m=1 e ρ=0.9, o tempo médio de espera fica em torno de 720 microsegundos.

A Fig. 4.7 mostra o tempo médio de espera no *buffer* de saída, ou seja, para m=1.

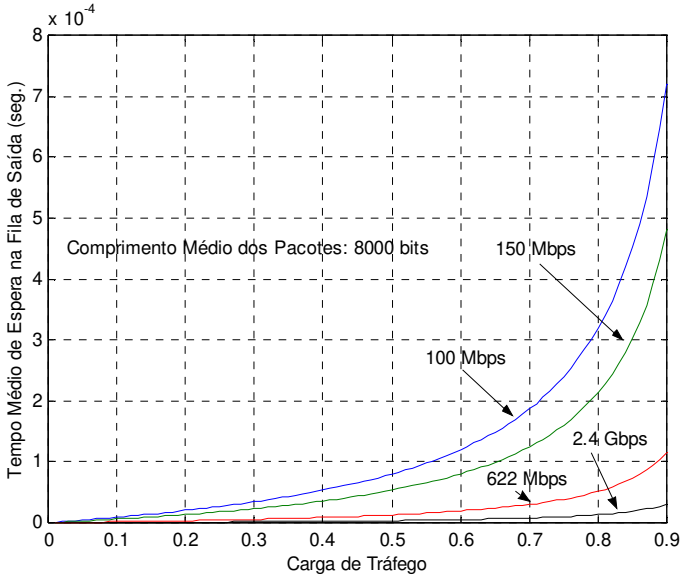

Figura 4.7 – Tempo médio de espera no *buffer* de saída.

A Fig. 4.8 ilustra o tempo médio de espera no *buffer* de entrada considerando o número de enlaces internos *m*=2.

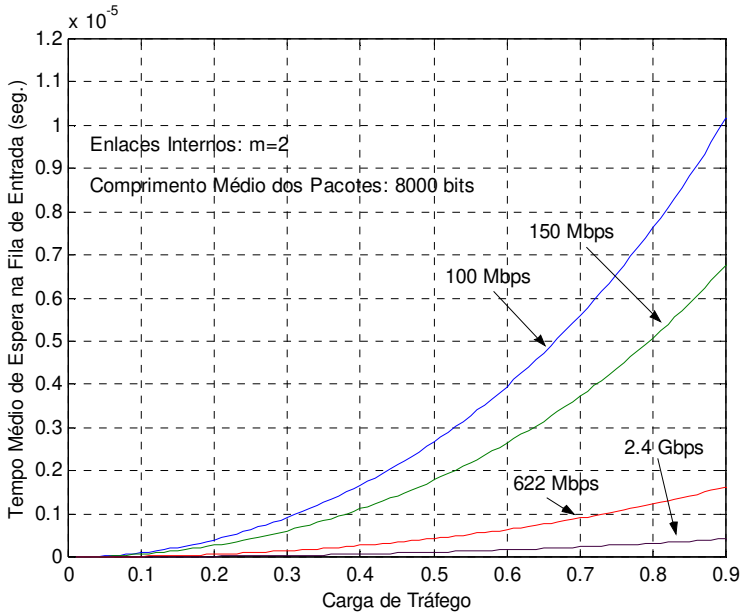

Figura 4.8 – Tempo médio de espera no *buffer* de entrada para *m*=2.

No caso dos *buffers* de entrada, para uma carga de tráfego igual a 90%, o tempo médio de espera é muito baixo. Ele varia de 10 microsegundos, para uma capacidade de enlace de 100 Mbps, a 0.4 microsegundos para uma capacidade de enlace de 2.4 Gbps. Nos *buffers* de saída, o tempo médio de espera varia de 700 microsegundos a 30 microsegundos para as mesmas condições anteriores.

O tempo médio que os pacotes ficam no sistema, ou seja, o intervalo de tempo médio decorrido entre a chegada e a partida, pode ser calculado como a soma dos tempos médios de espera nos *buffers* de entrada e de saída e os tempos médios de atendimento dos servidores de entrada e de saída. Considerando que na estrutura proposta os servidores de entrada e de saída possuem o mesmo tempo médio de atendimento, quem mais impacta o atraso total é o tempo médio de espera no sistema de saída. A Fig. 4.9 mostra o tempo médio total que um pacote permanece na estrutura proposta com dois enlaces internos (*m*=2), considerando pacotes com comprimento médio de 8000 bits e capacidade do enlace de 622 Mbps. Mesmo submetido a um tráfego intenso, o tempo médio total continua ainda pequeno, em torno de 0.14 milisegundos.

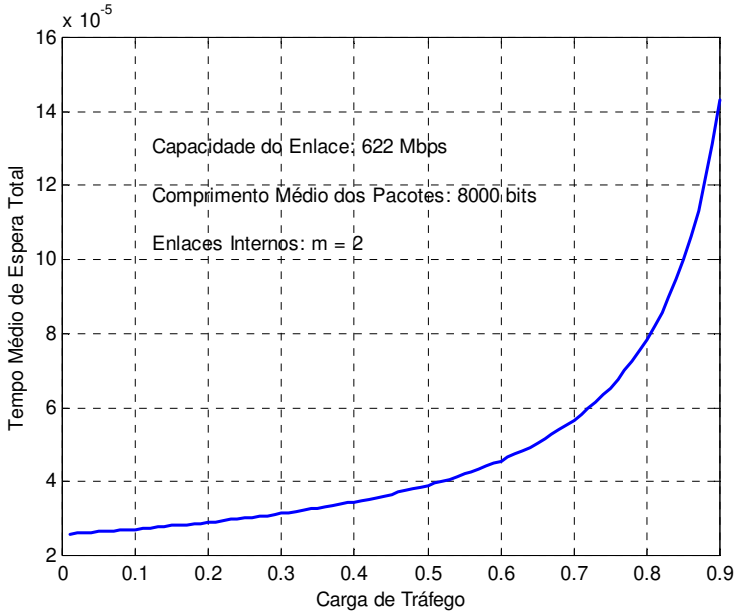

Figura 4.9 – Tempo médio total despendido por um pacote na estrutura proposta.

4.4.1.2 Simulação

A simulação tem como objetivo principal verificar a consistência entre o modelo matemático de teoria de filas proposto a estrutura do comutador proposto. Utilizando o *software* MATLAB, um programa foi desenvolvido e escrito para simular a estrutura do comutador para verificar o desempenho em termos de número médio de pacotes e o tempo médio de espera nas filas de entrada e de saída. Os resultados foram depois confirmados e validados através de simulações com o *software* ARENA [12].

As variáveis utilizadas foram a capacidade do enlace C, o tamanho médio dos pacotes L, a taxa média de chegada de pacotes λ, a quantidade de portas de entrada e de saída N e o número de enlaces internos *m*. Para cada entrada foi gerado um vetor com cem mil pacotes com tamanhos segundo uma distribuição exponencial com média L e com endereços de saída entre 1 e N. Um outro vetor foi gerado com os tempos de chegada dos pacotes anteriores, utilizando uma distribuição exponencial entre chegadas com média λ. O programa verifica em cada fila de entrada o destino e o tempo de chegada de cada pacote para fazer o atendimento, cujo tempo de serviço tem como base a capacidade do enlace e o tamanho do pacote. Os pacotes são encaminhados às respectivas saídas, criando novos vetores de tempo, de tamanho de pacotes e tempos de chegada.

A partir dos instantes de chegada, atendimento e partida, na entrada e na saída, o programa computa o tempo médio de espera e o número médio de pacotes nas filas de entrada e de saída. A Tabela 4.2 mostra alguns valores obtidos com a simulação e com as expressões para o tempo médio de espera, considerando C=100Mbps e L=8000 bits. Como pode ser observado, praticamente não existem diferenças entre eles, mostrando que o modelo de filas proposto para análise, as expressões obtidas e modelo da estrutura do comutador proposto são consistentes.

Tabela 4.2 – Comparação entre os resultados matemáticos e os simulados

	m = 1		*m* = 2		*m* = 3	
carga	matemático	simulado	matemático	simulado	matemático	simulado
ρ=.5	8.0000e-05	8.0502e-05	5.3333e-06	5.3678e-06	4.8485e-07	4.8458e-07
ρ=.6	1.2000e-04	1.2010e-04	7.9121e-06	7.9310e-06	8.2192e-07	8.2049e-07
ρ=.7	1.8666e-04	1.8626e-04	1.1168e-05	1.1166e-05	1.2842e-06	1.2538e-06
ρ=.8	3.2000e-04	3.2010e-04	1.5238e-05	1.5194e-05	1.8921e-06	1.8708e-06
ρ=.9	7.2000e-04	7.1996e-04	2.0314e-05	2.0001e-05	2.6677e-06	2.6642e-06

4.4.2 Modelo da Estrutura do Comutador com Escalonamento por Prioridade para Análise de Desempenho

No modelo são considerados 5 *buffers* em cada entrada, um para cada classe de fluxo. Em cada *buffer* de classe de fluxo os pacotes chegam das N entradas com taxa λ, assumindo que a chegada de pacotes obedecem à distribuição de Poisson. Para fazer o estudo da análise de desempenho do comutador proposto em relação ao atraso dos pacotes nos *buffers* de entrada e de saída, como a estrutura é simétrica, o modelo para análise pode ser simplificado em uma única fila para cada classe de fluxo e as *m*x5 filas podem ser consideradas como uma única fila para cada classe de fluxo, como ilustra a Fig. 4.10.

O servidor j verifica inicialmente se há pacotes a serem transmitidos no *buffer* f_1, que é o de mais alta prioridade e pode servir até *m* pacotes simultaneamente. Se não houver pacotes ou existirem menos do que *m* pacotes no *buffer* f_1, o servidor irá verificar o próximo *buffer* (f_2, f_3, f_4 e f_5, nesta seqüência), até completar *m* pacotes. Os *buffers* de saída em cada enlace interno são divididos em 5 classes de fluxo e os pacotes são transmitidos destes *buffers* pela porta de saída j seguindo a ordem de prioridade.

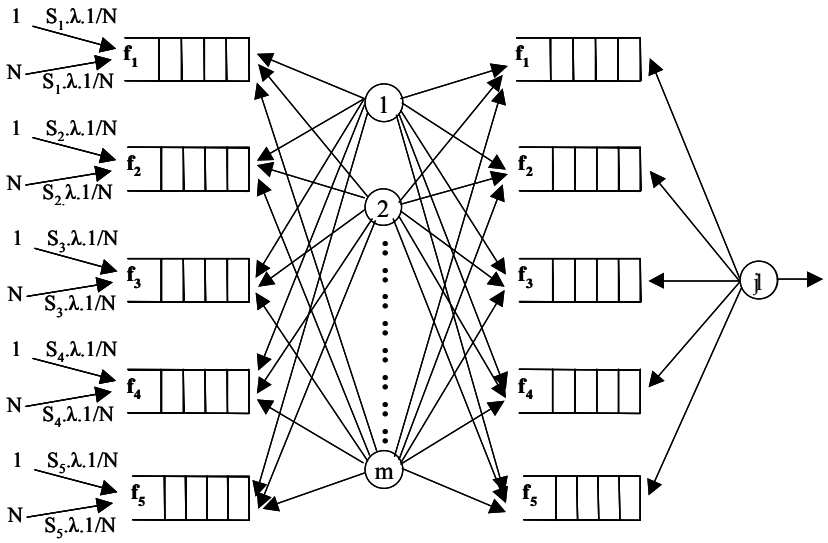

Figura 4.10 – Modelo da estrutura do comutador com prioridade para análise de desempenho.

O modelo considera ainda que o número de portas de entradas e saídas é N e o percentual de tráfego da classe de serviço i é S_i ($\Sigma_i.S_i=1$). A probabilidade de um pacote em uma porta de entrada ser enviado a uma saída em particular é 1/N. Logo, a taxa em cada *buffer* de classe de fluxo será $N.S_i.\lambda.1/N$.

Para análise, foram feitas ainda as seguintes considerações: os pacotes que chegam em cada entrada obedecem à distribuição de Poisson com taxa média de chegada λ e o comprimento dos pacotes obedecem à distribuição exponencial com média $1/\mu$. Os pacotes de uma mesma prioridade são servidos em um esquema FIFO. Desta forma, a rede aberta de filas da Fig. 4.10 pode ser modelada como uma fila M/M/*m* para a entrada e como uma fila M/M/1 para a saída, com prioridade. É também assumido que existem r fluxos e um fluxo genérico de prioridade p pode assumir p=1,2,3.....r, onde r é o valor de menor prioridade.

Para um sistema de fila com prioridade e disciplina de serviço sem preempção, o cálculo do tempo médio de espera E{Tp} é dado por:

$$E(Tp) = \frac{\left(m/\mu\right).E_{2,m}}{(m - \sigma_{p-1}).(m - \sigma_p)} \qquad (4.8)$$

onde $\sigma_p = \sum_{k=1}^{p} \rho_k$, $\sigma_0 = 0$, $\sigma_r < 1$.

Ainda, $\rho_k = \frac{\lambda.S_k}{\mu}$, onde $\lambda.S_k$ e $\frac{1}{\mu}$ são a taxa média de chegada e o tamanho médio do pacote do fluxo k, respectivamente.

Logo, $\rho = \frac{\lambda}{\mu}\sum_{k=1}^{r} S_k = \frac{\lambda}{\mu}$ é a carga total por enlace.

A expressão $E_{2,m} = \frac{\frac{\rho^m.m}{m!(m-\rho)}}{\sum_{n=0}^{m-1}\frac{\rho^n}{n!} + \frac{\rho^m.m}{m!(m-\rho)}}$ é conhecida como fórmula Erlang C.

Exemplo Numérico:

Para este exemplo são assumidas 5 classes de fluxos, f_1, f_2, f_3, f_4 e f_5, onde f_1 é o fluxo de mais alta prioridade. Os percentuais de cada fluxo são S_1=30%, S_2=30%, S_3=20%, S_4=10% e S_5=10% para f_1, f_2, f_3, f_4 e f_5, respectivamente. O comprimento médio dos pacotes para todos os fluxos é de 8000 bits e a capacidade do enlace é de 150 Mbps.

A Fig. 4.11 mostra os resultados numéricos obtidos a partir da Eq. 4.8 para *m*=1, ou seja, o tempo médio de espera no *buffer* de saída.

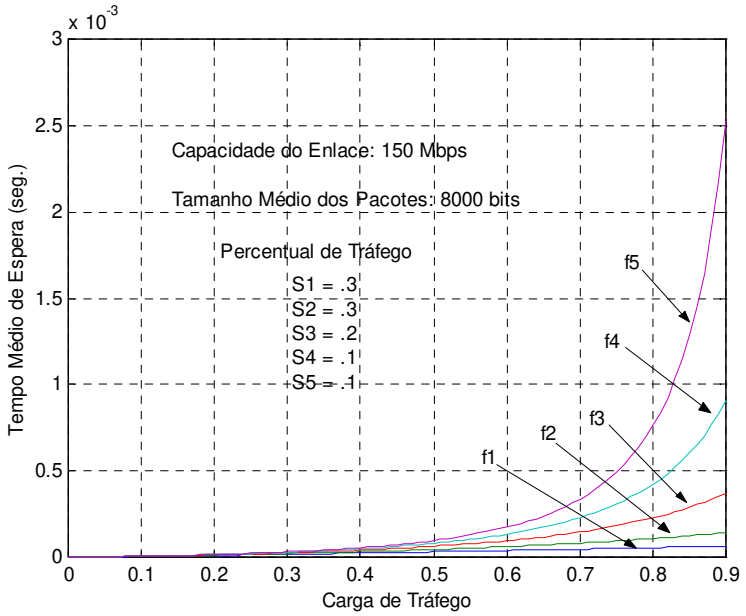

Figura 4.11 – Tempo médio de espera no *buffer* de saída.

Observa-se que, para uma carga de tráfego de até 80%, todos os fluxos possuem um tempo médio de espera insignificante. Para uma carga de tráfego de 90%, apenas o fluxo f_5 possui um tempo médio de espera significativo, em torno de 2.5 milisegundos.

A Fig. 4.12 mostra o tempo médio de espera no *buffer* de entrada, considerando apenas dois servidores (*m*=2). Podemos observar que, mesmo sob um tráfego intenso (ρ=0.9), o tempo médio de espera para todas as classes é insignificante. Mesmo usando apenas dois enlaces internos, praticamente não existem pacotes esperando no *buffer* de entrada.

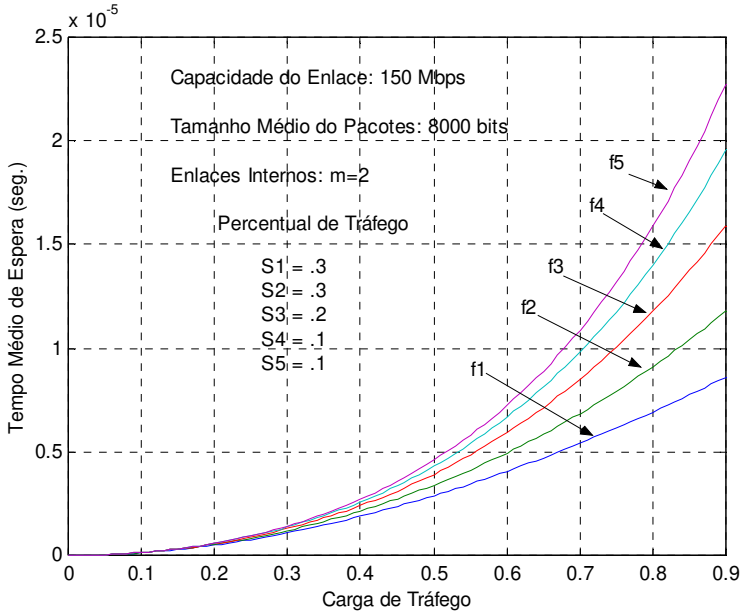

Figura 4.12 – Tempo médio de espera no *buffer* de entrada.

A Tabela 4.3 resume o desempenho da estrutura do comutador proposto em termos de tempo médio de espera em função do número de enlaces internos (*m*), considerando uma carga de tráfego de 90%, capacidade do enlace de 150 Mbps e pacotes com tamanho médio de 8000 bits.

Tabela 4.3 – Tempo médio de espera em função do número de enlaces internos

fluxo	*m*=1	*m*=2	*m*=3	*m*=4
f1	6.5753e-005	8.6107e-006	1.3681e-006	2.0488e-007
f2	1.4294e-004	1.1796e-005	1.6684e-006	2.3686e-007
f3	3.7267e-004	1.5942e-005	1.9977e-006	2.6935e-007
f4	9.0226e-004	1.9560e-005	2.2440e-006	2.9215e-007
f5	2.5263e-003	2.2760e-005	2.4363e-006	3.0911e-007

4.5 Análise de Desempenho para Fonte de Tráfego HTTP

O tráfego *web* continua crescendo e é estimado hoje como sendo mais de 70% do tráfego da Internet. Vários estudos têm mostrado que o tráfego na Internet é autosimilar, ou seja, apresenta estatísticas similares em diferentes escalas de tempo. Com base em um extenso trabalho de campo, com diversas medidas em redes com tráfego real, modelos de simulações de fonte de tráfego *web* têm sido propostos.

Nesta seção é analisado o comportamento da estrutura do comutador proposto considerando fontes de tráfego real. O modelo de tráfego real utilizado é referente ao serviço HTTP (*Hypertext Transfer Protocol*), cujo modelo de fonte é proposto em [42]. Como o protocolo TCP (*Transmission Control Protocol*) é utilizado como protocolo de transporte nas redes IP, o modelo de fonte TCP é introduzido no modelo de fonte HTTP para aumentar a precisão.

4.5.1 O Modelo TCP

A Fig. 4.13 ilustra o mecanismo de estabelecimento e liberação de conexão do protocolo TCP. O modelo TCP utilizado é uma versão simplificada do modelo proposto em [42], onde o mecanismo de controle de congestionamento conhecido como partida lenta é descartado. Uma vez estabelecida a conexão entre o cliente e o servidor, inicia-se a transmissão dos dados. Na seqüência, ao término desta transmissão, a conexão é encerrada.

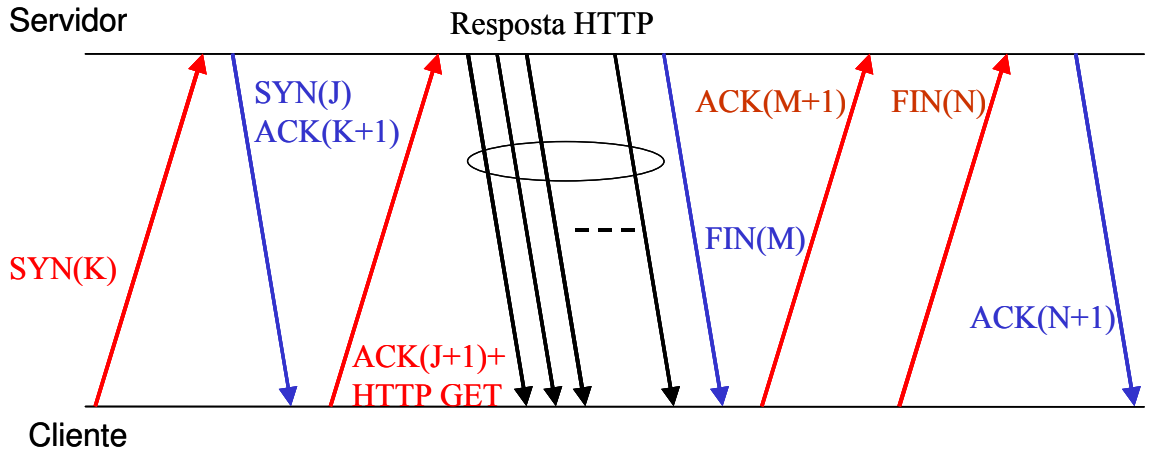

Figura 4.13 – Segmentos de controle de estabelecimento e liberação em uma conexão TCP.

A Fig. 4.14 mostra as interações entre cliente, roteador e o servidor para o estabelecimento de uma conexão TCP entre cliente e servidor, a transmissão dos segmentos TCP do servidor para o cliente e o encerramento da conexão TCP, onde:

τ_c é a soma do tempo que o pacote ACK leva para propagar-se do cliente para o servidor, mais o tempo que o segmento de dados TCP leva para propagar-se do servidor para o roteador.

τ_l é o tempo de transmissão do segmento de dados TCP do roteador para o cliente.

$\tau_{total} = \tau_c + \tau_l$

No modelo TCP proposto, τ_c é modelado por uma variável aleatória com distribuição exponencial de média 50 ms, enquanto τ_l é modelado por uma variável determinística e determinada pela velocidade do enlace de acesso.

Com base na observação da Fig. 4.14, o processo de chegada dos pacotes no roteador para o *download* de um objeto pode ser representado pela Fig. 4.15 e pelo seguinte algoritmo de passo:

1. A variável S armazena o tamanho do objeto em bytes. O número de pacotes do objeto é dado por N = [S/(MTU – 40)].
2. Os pacotes a serem transmitidos são enfileirados de acordo com a ordem de chegada. Sendo P a variável que armazena o número de pacotes a serem transmitidos, se P == 0, vá para o passo 6.
3. Aguarda até que um pacote do objeto seja transmitido entre o roteador e o cliente.
4. Escalona a chegada do próximo pacote do objeto após um intervalo de tempo τ_c. Faz P = P – 1.
5. Se P > 0, vá para o passo 3.
6. Fim.

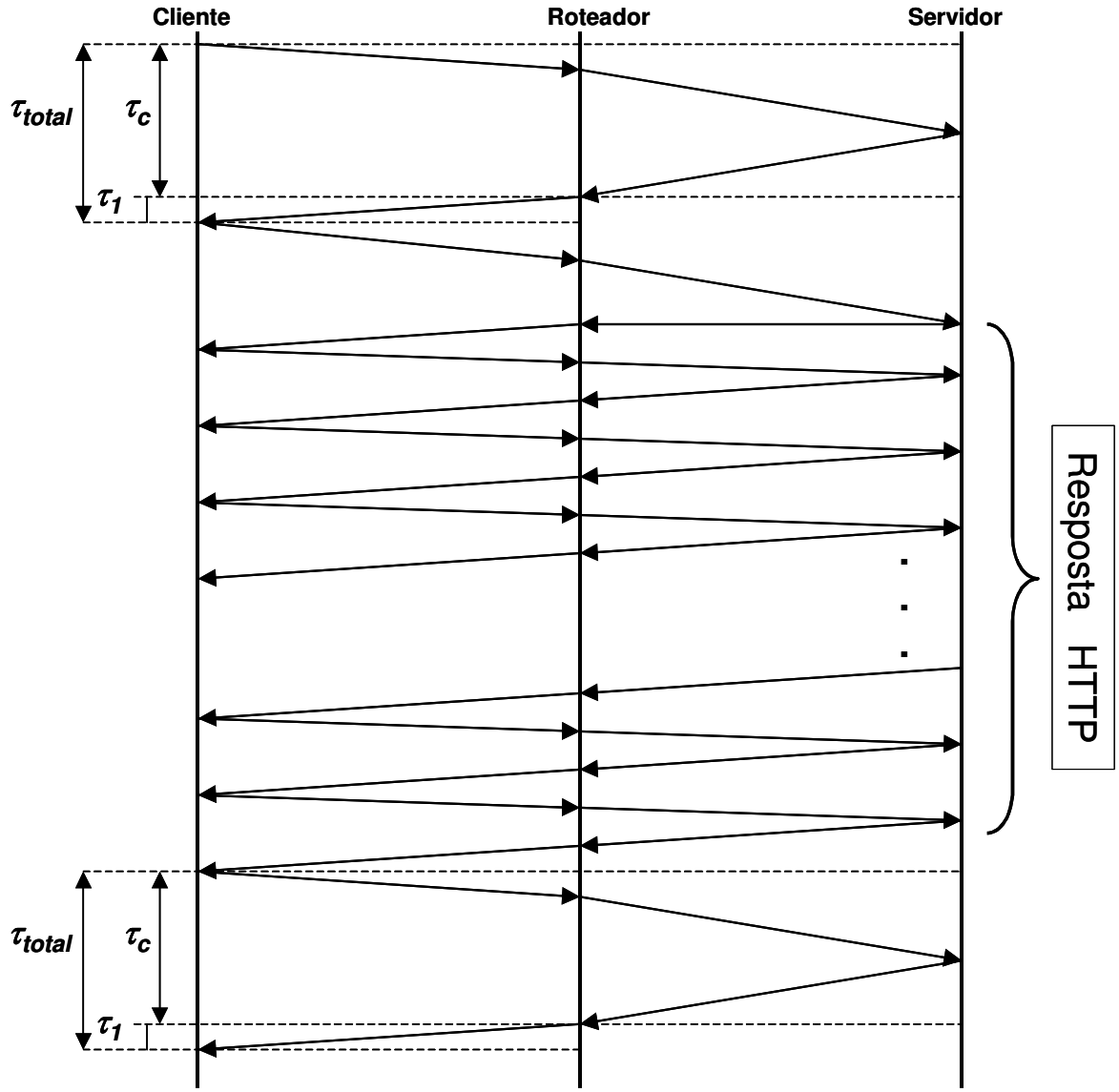

Figura 4.14 – Controle de fluxo TCP.

Os segmentos de dados TCP possuem unidade máxima de transferência MTU (*Maximum Transmission Unit*) de 1500 bytes ou de 576 bytes. A estrutura deste segmento destina 1460 bytes para dados para MTU de 1500 bytes e 536 bytes para dados para MTU de 576 bytes, identificados como MTU-40. Os outros 40 bytes de cada MTU são dedicados a código de redundância e cabeçalhos TCP e PPP. Assim, os segmentos de dados TCP possuem um *overhead* percentual de 2,7% para segmentos com MTU de 1500 bytes e 7% para segmentos com MTU de 576 bytes.

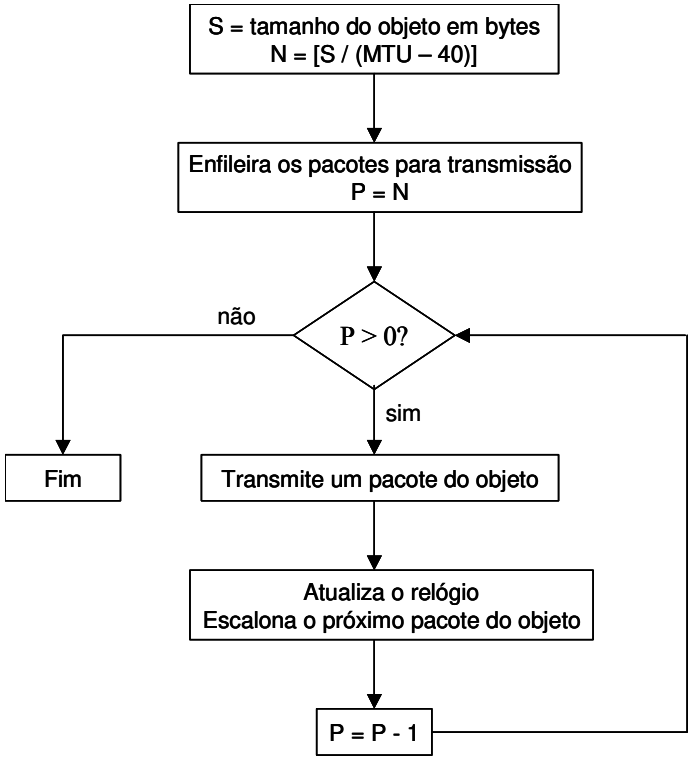

Figura 4.15 – Processo de chegada de pacotes no roteador.

4.5.2 O Modelo HTTP

A Fig. 4.16 ilustra uma típica seção de tráfego *web*. Esta seção apresenta períodos de *downloads* de páginas *web* e períodos de tempo de leitura das mesmas, sendo que em uma seção pode haver n períodos de *downloads* e leituras de páginas *web*. Esta divisão em períodos do tráfego HTTP deve-se à interação homem máquina. Além disso, outra característica importante do tráfego *web* é a autosimilaridade, isto é, estatísticas similares em diferentes escalas de tempo.

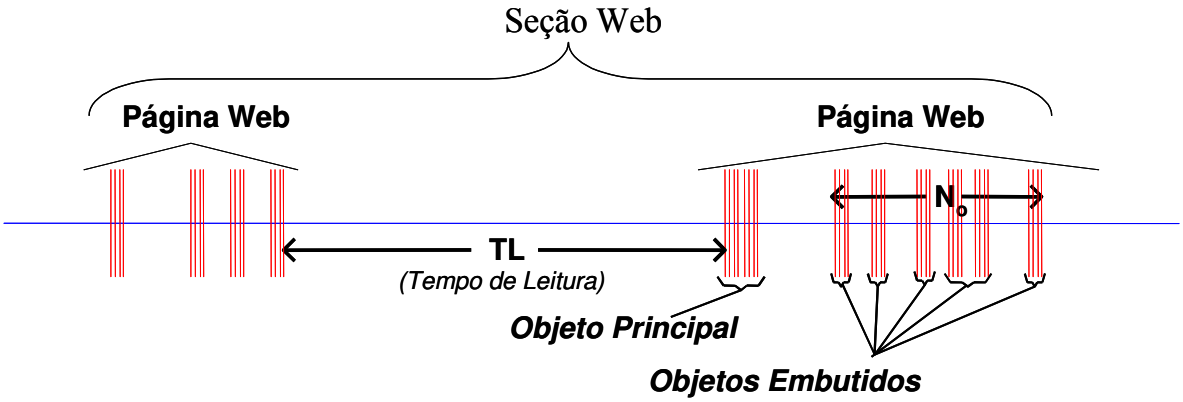

Figura 4.16 – Seção de navegação *web*.

Uma página *web* é composta por um único objeto principal e vários objetos embutidos. Inicialmente, o navegador busca o objeto principal da página HTML (*Hyper Text Markup Language*) usando uma requisição HTTP GET. Após receber o objeto principal, o navegador determina o *layout* da página para introduzir objetos embutidos, tais como figuras, botões estilizados, entre outros objetos. O tempo gasto para a determinação do *layout* da página *web* é chamado tempo *parsing*. Em seguida, o navegador baixa os objetos embutidos e o usuário inicia a leitura desta página *web*. A busca e o *download* dos objetos principal e embutidos são representados pelos períodos On, enquanto o tempo de leitura destas páginas representam o período Off. A Figura 4.17 mostra o objeto principal e seus respectivos objetos embutidos da página *web* do jornal *Wall Street*.

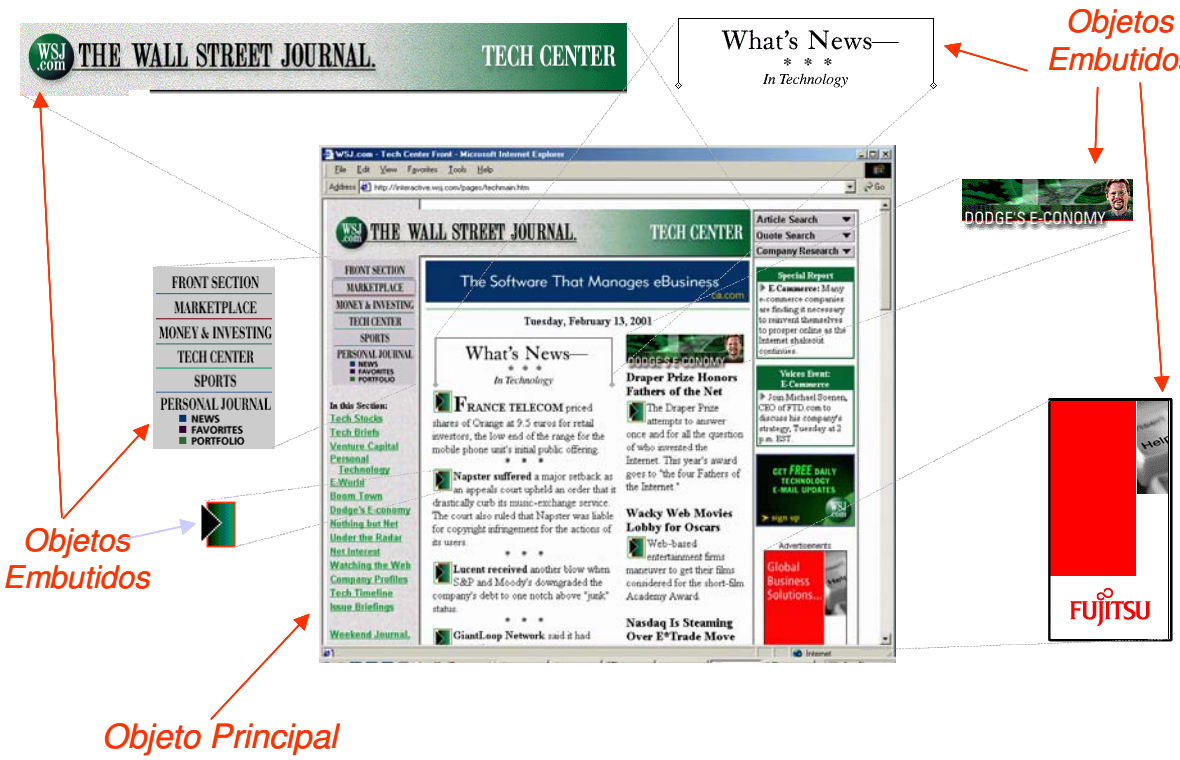

Figura 4.17 – Uma típica página *web* e seu conteúdo.

As características do tráfego HTTP dependerão do modo de download dos segmentos TCP usados pelos servidores *web* e navegadores. Atualmente duas versões do protocolo HTTP são usadas, HTTP/1.0 e HTTP/1.1, na maioria dos servidores e navegadores. Estas duas versões diferem na maneira como são usadas as conexões TCP para transferir os objetos principais e embutidos.

No HTTP/1.0, uma conexão TCP distinta é usada para o *download* de cada objeto principal ou embutido. Esta versão, também conhecida como HTTP/1.0-*burst mode transfer,* é a mais popular entre os navegadores *web*. São permitidas até 4 conexões TCP paralelas na maioria dos navegadores.

Já no HTTP/1.1, também conhecido como HTTP/1.1-*persistent mode transfer,* são utilizadas conexões TCP persistentes em que os objetos são baixados serialmente por meio de uma conexão TCP.

4.5.2.1 Parâmetros do Modelo de Tráfego HTTP

As distribuições dos parâmetros do modelo de tráfego HTTP foram baseadas em uma extensa pesquisa na literatura e propostas em [42].

Uma página *web* é representada por um único objeto principal e vários objetos embutidos. O tamanho do objeto principal segue uma distribuição lognormal truncada, com o tamanho mínimo de 100 bytes, máximo de 2 megabytes e média de 10710 bytes, com os parâmetros σ valendo 1.37 e μ sendo 8.35. Já os objetos embutidos possuem distribuição lognormal truncada, com tamanho médio de 7758 bytes, mínimo de 50 bytes e máximo de 2 megabytes sendo os parâmetros σ com valor de 2.36 e μ valendo 6.17.

O número de objetos embutidos (No) por página *web* obedece à distribuição de pareto truncada com média de 5.64 objetos e máximo de 53 objetos. Os parâmetros k, α e *m* valem 2, 1.1 e 55, respectivamente.

O tempo de leitura (TL) é o tempo entre duas requisições de página *web*, distribuído exponencialmente com média de 30 segundos.

Por fim, o tempo parsing, que é o tempo gasto para determinar o *layout* da página *web* após a busca do objeto principal, obedece também a uma distribuição exponencial com média 0.13 segundos [43]. A Tabela 4.4 resume os parâmetros do modelo de tráfego HTTP.

Na Tabela 4.4 são descritas todas as funções densidades de probabilidade (PDF) usadas para obter o modelo de tráfego HTTP. No entanto, para a geração de números aleatórios com as distribuições lognormal, pareto e exponencial, foi adotado neste estudo o método da transformação inversa, em que as distribuições podem ser invertidas analiticamente. Este método

foi aplicado em todos os modelos de tráfego estudados. Entretanto, para a geração dos números aleatórios com as distribuições citadas acima, grande parte já estava implementada no *toolbox stat* da ferramenta de *software* Matlab, com exceção da distribuição Pareto Truncada obtida em [44].

Tabela 4.4 – Parâmetros do modelo de tráfego HTTP.

Componente	**Distribuição**	**Parâmetros**	**PDF**
Tamanho do objeto principal	Lognormal Truncada	Média = 10710 bytes Desvio = 25032 bytes Mínimo = 100 bytes Máximo = 2 Mbytes	$f_x = \frac{1}{\sqrt{2\pi}\sigma x}\exp\left[\frac{-(\ln x-\mu)^2}{2\sigma^2}\right], x \geq$ $\sigma = 1.37, \mu = 8.35$
Tamanho do objeto embutido	Lognormal Truncada	Média = 7758 bytes Desvio = 126168 bytes Mínimo = 50 bytes Máximo = 2 Mbytes	$f_x = \frac{1}{\sqrt{2\pi}\sigma x}\exp\left[\frac{-(\ln x-\mu)^2}{2\sigma^2}\right], x \geq$ $\sigma = 2.36, \mu = 6.17$
Número de objetos embutidos por página	Pareto Truncada	Média = 5.64 Max. = 53	$f_x = \frac{\alpha k^{\alpha}}{x^{\alpha+1}}, k \leq x < m$ $f_x = \left(\frac{k}{m}\right)^{\alpha}, x = m$ $\alpha = 1.1, k = 2, m = 55$
Tempo de Leitura	Exponencial	Média = 30 seg	$f_x = \lambda e^{-\lambda x}, x \geq 0$ $\lambda = 0.033$
Tempo Parsing	Exponencial	Média = 0.13 seg	$f_x = \lambda e^{-\lambda x}, x \geq 0$ $\lambda = 7.69$

Como mostrado anteriormente, o modelo TCP adotado neste estudo estabelece o processo de chegada dos pacotes ao roteador. Cada objeto ou arquivo dos modelos HTTP é segmentado em pacotes de 576 bytes e 1500 bytes. De acordo com o modelo apresentado em [42], as estatísticas encontradas na literatura mostram uma distribuição de 24% para 576 bytes e 76% para 1500 bytes, bem como uma distribuição de 50% entre os dois modos de *download* disponíveis. Assim,

o processo de geração de tráfego *web* pode ser descrito pelo fluxograma da Fig. 4.18. Para o pacote de confirmação ACK do terminal para o roteador, o tempo total de propagação é considerado distribuído exponencialmente com média 50 ms.

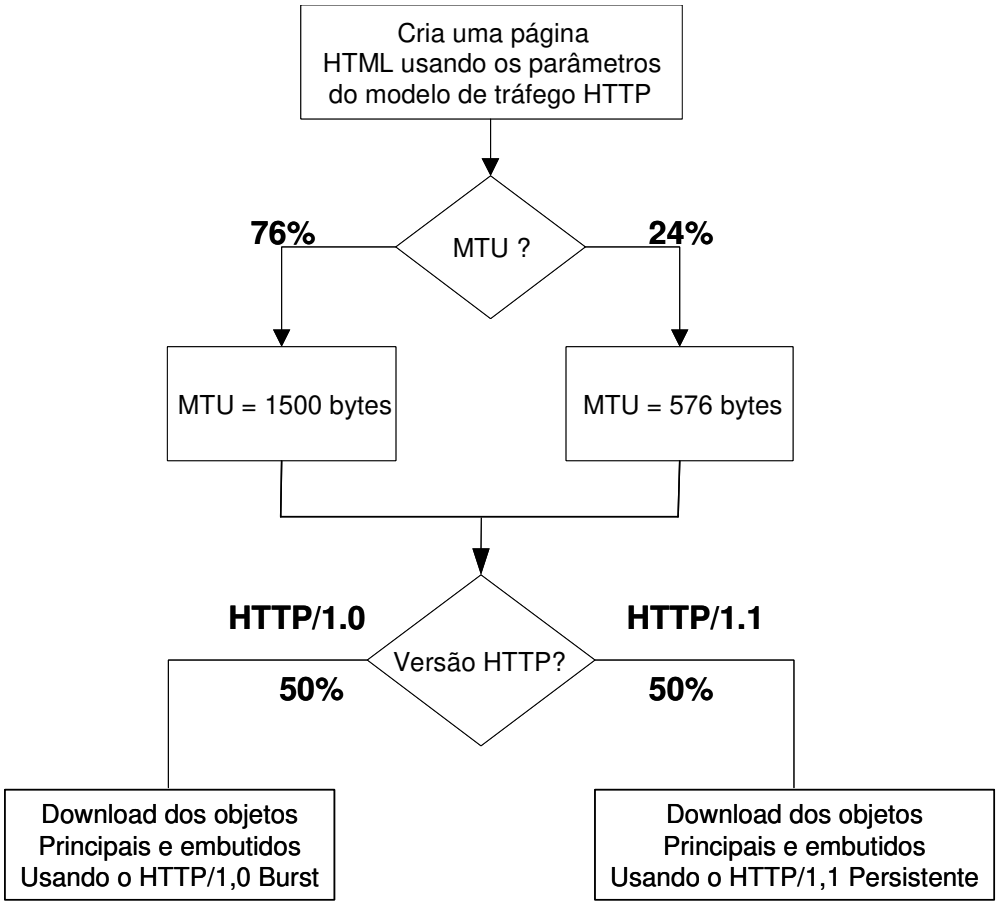

Figura 4.18 – Modelagem de *download* de uma página *web*.

Seguindo o processo e o fluxograma descritos no modelo apresentado em [42], foi desenvolvido e escrito um programa, utilizando o *software* Matlab, para gerar o tráfego HTTP. Devido à simetria da estrutura *crossbar*, o modelo utilizado foi o mesmo descrito no item 4.3.2.1 e ilustrado na Fig. 4.5.

Uma vez que o modelo de tráfego HTTP é composto por várias distribuições, existe uma dificuldade natural de variar a taxa de chegada dos pacotes através da variação do parâmetro λ. Para verificar o desempenho da estrutura submetida a uma carga de tráfego mais intensa, foi adotado na simulação um enlace com capacidade C=22 kbytes por segundo (176kbps). Foram simuladas situações com 1 e com 4 usuários simultâneos, o que corresponde a uma carga de tráfego de ρ=0.62 e ρ=0.72, respectivamente. Os resultados obtidos com relação ao tempo de

espera na fila de entrada em função do número de enlaces internos *m* são mostrados na Tabela 4.5, onde para *m*=1 tem-se o tempo de espera na fila de saída.

Mais do que os valores absolutos, por causa da baixa capacidade do enlace utilizado, deve-se observar o ganho relativo, ou seja, a variação dos tempos de espera na fila quando se varia a quantidade de enlaces internos. Para o caso mais crítico, ou seja, 4 usuários, o tempo de espera na fila de entrada passou de 1.89s (*m*=1), para 4.6ms (*m*=3). O número médio de pacotes esperando na fila foi 32.07, 0.98 e 0.077 para *m*=1, *m*=2 e *m*=3, respectivamente. Ou seja, com apenas 2 enlaces internos existe, em média, menos de 1 pacote em espera.

Tabela 4.5 – Tempo médio de espera dos pacotes na fila para fonte HTTP.

Usuários	*m*=1	*m*=2	*m*=3
1 (ρ=0.62)	8.7322e-1	1.1251e-2	1.9033e-3
4 (ρ=0.72)	1.8944e+0	5.9437e-2	4.6562e-3

Para que se possa ter uma idéia melhor do desempenho da estrutura, em termos do tempo médio de espera na fila, quando submetida à fonte de tráfego HTTP, a mesma foi também submetida à fonte do tipo Poisson com os mesmos parâmetros da fonte HTTP, ou seja, pacotes com tamanho médio L=7672 bits, capacidade do enlace C=176kbps, λ=16.5 (4 usuários) e λ=14.2 (1 usuário). Os valores do tamanho médio de pacotes L e da taxa média de chegada de pacotes λ foram obtidos diretamente da simulação da fonte HTTP. A Tabela 4.6 resume os resultados obtidos.

Tabela 4.6 – Tempo médio de espera dos pacotes na fila para fonte do tipo Poisson.

Usuários	*m*=1	*m*=2	*m*=3
1 (ρ=0.62)	7.7873e-2	4.9455e-3	5.3803e-4
4 (ρ=0.72)	1.1089e-1	6.4422e-3	7.4287e-4

No caso de 4 usuários, o número médio de pacotes foi 1.82, 0.11 e 0.012 para *m*=1, *m*=2 e *m*=3, respectivamente.

4.6 Conclusão

Neste capítulo foi estudado o desempenho da estrutura proposta quando submetida a fontes de tráfego com pacotes de tamanho variável. O princípio básico de operação foi descrito, considerando escalonamento *Round-robin* e por prioridade. Para análise de desempenho, foram propostos modelos para a estrutura com base nos dois tipos de escalonamento.

Para o tráfego do tipo Poisson, ambas as estruturas foram analisadas analiticamente, em termos de tempo médio de espera e número médio de pacotes, utilizando teoria de filas. Os resultados mostraram que apenas 2 enlaces internos são suficientes para se ter uma transferência rápida de pacotes das entradas para as saídas. A estrutura com escalonamento *Round-robin* foi simulada através de um programa desenvolvido e escrito em Matlab, e os resultados mostraram a consistência do modelo matemático proposto.

Com a finalidade de se conhecer o desempenho da estrutura proposta com escalonamento *Round-robin* sob um modelo de tráfego real, a mesma foi submetida, através de simulações, a uma fonte de tráfego HTTP. Inicialmente foi feita uma breve descrição das características de uma página *web* e do modelo de fonte com os seus principais parâmetros. Com base no algoritmo e fluxograma indicados na referência [42], foi desenvolvido um programa em Matlab para gerar a fonte HTTP que foi então aplicada à estrutura proposta.

Os resultados mostraram que, mesmo sob uma fonte de tráfego que se aproxima do comportamento dos usuários de Internet, a estrutura apresentou um desempenho muito bom. Em termos de tamanho médio da fila, com apenas 2 enlaces internos, tem-se em média menos do que 1 pacote em espera. O tempo médio de espera para m=3 é em torno de 400 vezes menor quando comparado com m=1. Para uma fonte de tráfego do tipo Poisson, com parâmetros similares aos do tipo HTTP, esta relação é de aproximadamente 100 vezes, ou seja, o tempo médio de espera para m=3 é em torno de 100 vezes menor quando comparado com m=1. Com base nesses resultados conclui-se que, para as condições de simulações apresentadas, a estrutura apresenta um melhor ganho quando submetida a um modelo de fonte de tráfego real.

Capítulo 5

Conclusões

A crescente demanda por largura de faixa bem como os requisitos requeridos para as aplicações multimídias emergentes, têm exigido um desempenho cada vez maior dos elementos de redes. Se por um lado a transmissão tem acompanhado bem este crescimento, os elementos de encaminhamento de pacotes poderão tornar-se o gargalo destas redes, fazendo com que o desempenho global do sistema seja prejudicado.

Neste trabalho foi proposta uma estrutura relativamente simples de comutador de pacotes de alta velocidade que, utiliza *buffers* nas entradas e nas saídas, possui *m* enlaces paralelos internos conectando cada *buffer* de entrada aos *m buffers* de cada saída, é baseada na estrutura *crossbar* e provê facilidades para aplicações que requerem qualidade de serviço. Desta forma, até *m* pacotes proveniente das entradas podem ser transferidos simultaneamente para uma mesma saída, permitindo uma alta taxa de transferência de pacotes e diminuindo a probabilidade de ocorrência de bloqueio na cabeça da fila. Adicionalmente, como os enlaces internos operam na mesma velocidade dos enlaces externos, não há necessidade de aumentar a velocidade do relógio interno.

Foi feita uma análise de desempenho do comutador operando com pacotes de tamanho fixo e com pacotes de tamanho variável, utilizando simulações e modelos analíticos, considerando escalonamentos *Round-robin* e por prioridade. O objetivo foi avaliar o tempo médio de espera dos pacotes e o número médio de pacotes nas filas de entrada e de saída, admitindo que os pacotes em cada *buffer* são servidos em um esquema FIFO.

Operando com pacotes de tamanho fixo (células ATM), considerando fontes independentes e identicamente distribuídas, com apenas dois enlaces internos ($m=2$) já se obtém um número reduzido de células esperando na fila de entrada, tanto no escalonamento *Round-robin*, em média em torno de cinco células, quanto no escalonamento por prioridade, em média duas células para a classe de mais baixa prioridade. Para três enlaces internos, praticamente não há mais células esperando no *buffer* de entrada. Adicionalmente, os resultados da análise mostraram que, quando o número de enlaces é maior ou igual a três, não há nenhum benefício em se utilizar esquema

aleatório de atendimento dos pacotes no *buffer*. Por outro lado, em termos de número médio de células esperando na fila, a eficiência com a utilização de dois enlaces com esquema FIFO é melhor do que a de um enlace com esquema aleatório, sendo que o esquema FIFO é mais simples de implementar.

Com relação ao desempenho do comutador operando com pacotes de tamanho variável, a análise foi feita considerando pacotes com comprimento exponencial (tráfego poissoniano) e também com modelo de tráfego real (HTTP). Com relação ao tráfego poissoniano, tanto para o escalonamento *Round-robin* quanto para o escalonamento por prioridade, não há necessidade de se ter mais do que dois enlaces internos. Em ambos os casos praticamente não se têm pacotes esperando nas filas de entrada e, conseqüentemente, com tempo médio de espera muito baixo.

Quando submetida a um modelo de fonte de tráfego real, a estrutura também apresentou um desempenho muito bom. Com apenas dois enlaces internos, além de apresentar, nas condições simuladas, valores bem baixos para o número médio de pacotes em espera e o tempo médio de espera nas filas, a estrutura proposta apresentou um alto ganho relativo à estrutura com apenas um enlace interno. Quando comparado com tráfego poissoniano, os valores obtidos para o tempo médio de espera com o tráfego HTTP são relativamente maiores. Isto ocorre porque o tráfego de dados HTTP apresenta correlação entre os pacotes, que não são independentes, e que chegam em rajadas. Assim, a avaliação da estrutura proposta com relação ao tempo médio de espera dos pacotes utilizando fontes poissonianas, resultou sempre em uma estimativa subdimensionada quando comparada com a fonte HTTP.

Com base nos resultados obtidos para as condições analisadas, pode-se concluir que o comutador proposto, implementado com apenas três enlaces internos, é perfeitamente adequado para operar tanto em redes ATM quanto em redes de comutação IP. O comutador proposto, com uma quantidade muito pequena de *buffer* na entrada, apresenta um desempenho similar ao de uma estrutura com *buffer* de saída.

5.1 Trabalhos Futuros

A análise de desempenho da estrutura de comutação proposta foi feita considerando que a mesma possuía o *buffer* de saída ilimitado, ou seja, *buffer* de tamanho infinito. Também, apenas a

estrutura operando com pacotes de tamanho fixo e atendimento *Round-robin* teve o seu desempenho avaliado considerando o *buffer* de entrada com tamanho limitado. Desta forma, uma continuidade deste trabalho seria analisar a estrutura considerando *buffers* de entrada e/ou saída com tamanho finito.

Outros modelos de fonte real, como FTP e WAP, têm aparecido na literatura. A análise de desempenho da estrutura quando submetida a estas fontes pode ser feita, considerando-as individualmente ou simultaneamente. Neste último caso, pode-se também incluir a fonte HTTP e avaliar o comportamento admitindo atendimento por prioridade, onde cada fonte teria um determinado percentual de tráfego e níveis de prioridade de atendimento.

Referências Bibliográficas

[1] Kulzer J. J., Montgomery, W. A., *Statistical Switching Architecture for Future Services*, ISS' 84, Florence, May 1984.

[2] N. Giroux and S. Ganti, *Quality of Service in ATM Networks*, Prentice Hall, 1999.

[3] Newman, P., Lyon, T., and Minsall, G., *Flow Labelled IP: A Connectionless Approach to ATM*, Proc. IEEE Infocom 96, vol. 3, pp. 1251-1260.

[4] Rekhter, Y., Davie, B., Katz, D., Rosen, E., and Swallow, G., *Cisco System Tag Switching Architecture Overview*, IETF RFC 2105, February 1997.

[5] Viswanathan, A., Feldman, N., Boivie, R., and Woundy, R., *ARIS: Aggregate Route-Based IP Swintching*, IETF Internet Draft, March 1997.

[6] K. Nagami, Y. Katsube, Y. Shobatake, A. Mogi, S. Matsuzawa, T. Jinmei, H. Esaki, *Toshiba Flow Attribute Notification Protocol (FANP) Specification*, IEFT RFC 2129, April 1997.

[7] http://www.ietf.org/rfc/rfc3031.txt?number=3031D.

[8] Ginsburg, ATM – *Solutions for Enterprise Internetworking*, second edition, Addison-Wesley, 1999.

[9] Hao Che, San-qi Li, and Arthur Lin, *Adaptive Resource Management for Flow-Based IP/ATM Hybrid Switching Systems*, IEEE/ACM Transaction on Networking, vol.6, N0. 5, Oct. 1998, pp. 544-557.

[10] Newman, P., Minshall, G., Lyon, T., and Huston L.: *IP Switching and Gigabit Routers*, IEEE Comm. Magazine, January 1977, 64-69.

[11] Motoyama, S., and Arantes, M. P.: *An IP Switch with Distributed Scheduling*, IEE Electronics Letters, Vol. 38, No. 8, pp.392-393, April 2002.

[12] Aguiar Filho, S., R., "Modelamento, Simulação e Análise de Desempenho de um Comutador ATM CIOQ", Dissertação de mestrado, 65p, Inatel 2005.

[13] Pattavina, A., *Nonblocking Architectures for ATM Switching*, IEEE Communication Magazine, February 1993, pp. 38-48.

[14] Tobagi, F., *Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks*, in Proc. IEEE, Vol. 78, No. 1, Jan. 1990, pp. 133-167.

[15] Ahmadi, H. and W.E. Denzel, *A Survey of Modern High-Performance Switching Techniques*, IEEE J. Sel. Areas Commun., Vol. 7, No. 7, September 1989, pp. 1091-1103.

[16] Rathgeb, E.P., T.H. Theimer, M.N. Huber, *Buffering Concepts for ATM Switching Networks*, in Proc. GLOBECOM '88, Hollywood, FL, November 1988, pp. 1277-1281.

[17] Hluchyj, M.G. and M.J. Karol, *Queueing in Space-Division Packet Switching*, in Proc. IEEE INFOCOM '88, March 1988, pp. 334-343.

[18] Awdeh, R.Y. and H.T. Mouftah, *Survey of ATM Switch Architectures*, Computer Networks and ISDN Systems, vol. 27 (1995), pp. 1567-1613.

[19] Pattavina, A., *Multichannel Bandwidth Allocation in a Broadband Packet Switch*, IEEE J. Sel. Areas Commun., Vol. 6, No. 9, December 1988, pp. 1489-1499.

[20] Karol, M.J., M.G. Hluchyj and S.P. Morgan, *Input vs Output Queueing on a Space-division Packet Switch*, IEEE Trans. Commun., vol. 35, no. 12, 1987, pp. 1347-1356.

[21] Obara, H. and T. Yasushi, *An efficient contention resolution algorithm for input queueing ATM cross-connect switches*, Int. J. Digital and Analog Cabled Syst., vol. 2, Dec. 1989, pp. 261-267.

[22] Matsunaga, M. and H. Uematsa, *A 1.5 Gb/s 8 X 8 Cross-Connect Switch Using a Time Reservation Algorithm*, IEEE J. Sel. Areas Commun., vol. 9, no. 8, October 1991, pp. 1308-1317.

[23] Motoyama, S., Mavigno, M., C., "Um Novo Algoritmo de Encaminhamento de Células em um Comutador ATM com Buffer na Entrada e com Atendimento Prioritário de Classes de Serviços", XV Brazilian Telecommunication Symposium, September 1997, pp. 141-144.

[24] Motoyama, S., *Cell Delay Modelling and Comparison of Iterative Scheduling Algorithms for ATM Input-queued Switches*, IEE Proceedings on Communication, Vol. 150, N_o. 1, February 2003, pp. 11-16.

[25] Minkenberg, C. J. A., *On Packet Switch Design*, Master Thesis, Eindhoven University of Technology, September 2001.

[26] Nojima, S., E. Tsutsui, H. Fukuda and M. Hashimoto, *Integrated Services Packet Network Using Bus Matrix Switch*, IEEE J. Sel. Areas Commun.,vol. SAC-5, pp. 1284-1292, October 1987.

[27] Del Re, E. and R. Fantacci, *Performance Evaluation of Input and Output Queueing Techniques in ATM Switching Systems*, IEEE Trans. Commun.,vol. 41, no. 10, Oct. 1993.

[28] Hluchyj, M.G. and M.J. Karol, "Queueing in High-Performance Packet Switching," IEEE J. Sel. Areas Commun., vol. 6, no. 9, December 1988, pp. 1587-1597.

[29] Liew, S.C., *Performance of Various Input-buffered and Output-buffered ATM Switch Design Principles under Bursty Traffic: Simulation Study*, IEEE Trans. Commun., vol. 42 no. 2/3/4, Feb/Mar/Apr 1994, pp. 1371-1379.

[30] Tao, Z. and S. Cheng, *A NewWay To Share Buffer - Grouped Input Queueing In ATM Switching*, in Proc. IEEE GLOBECOM '94, vol. 1, pp. 475- 479.

[31] Eckberg, A.E. and T.-C. Hou, "*Effect of Output Buffer Sharing on Buffer Requirements in an ATM Packet Switch,"* in Proc. INFOCOM '88, New Orleans, LA, March 1988, pp. 459-466.

[32] Iliadis, I., *Performance of a Packet Switch with Shared Buffer and Input Queueing*, in Proc. Teletraffic and Datatraffic in a Period of Change, ITC - 13, 1991, pp. 911-916.

[33] Iliadis, I. and W.E. Denzel, *Performance of Packet Switches with Input and Output Queueing*, in Proc. ICC '90, April 1990, pp. 747-753.

[34] McKeown, N., B. Prabhakar and M. Zhu, *Matching Output Queueing with Combined Input and Output Queueing*, in Proc. 35th Annual Allerton Conf. Communication, Control and Computing, Monticello, IL, Oct. 1997.

[35] Prabhakar, B., and N. McKeown, *On the Speedup Required for Combined Input and Output Queued Switching,* Automatica, vol. 35, 1999.

[36] Chuang, S-T., A. Goel, N. McKeown and B. Prabhakar, *Matching Output Queueing with a Combined Input Output Queued Switch*, Stanford CSLTR- 98-758.

[37] Stoica, I., and H. Zhang, *Exact Emulation of an Output Queueing Switch by a Combined Input Output Queueing Switch*, Proc. 6th IEEE/IFIP IWQoS '98, Napa Valley, CA, May 1998, pp. 218-224.

[38] Krishna, P., N. S. Patel, A. Charny and R. J. Simcoe, *On the Speedup Required for Work-Conserving Crossbar Switches,* IEEE J. Sel. Areas Commun.,vol. 17, no. 6, 1999, pp. 1057-1066.

[39] Minkenberg, C., Engbersen, T., *A Combined Input and Output Queued Packet-Switched System Based on PRIZMA Switch-on-a-Chip Technology*, IEEE Commun. Magazine, 38(12), December 2000, 70-77.

[40] Chuang, S. T., Goel, A., Nckeown, N., Prabhakar, B., *Matching Output Queuing with a Combined Input Output Queued Switch*, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 6, pp. 1030-1039, June 1999.

[41] Yang, M., and Zheng S. Q.: *An Efficient Scheduling Algorithm for CIOQ Switches with Space-Division Multiplexing Expansion*, Proc. IEEE Infocom'2003.

[42] 3GPP2 Third Generation Partnership Project Two, *1xEV-DV Evaluation Methodology – Addendum (V6)*, Technical Report, WG5 Evaluation AHG, July 2001

[43] Choi, H., K., and Limb., J. O., *A Behavioral Model of web Traffic*, Seventh International Conference on Network Protocols, pages 327–334, November 1999.

[44] Evans, M., Hastings, N., and Peacock, B., *Statistical Distributions*, JohnWiley and Sons, second edition, 1993.

[45] Pattavina, A., *Switching Theory Architecture and Performance in Broadband ATM Networks,* JohnWiley and Sons, 1998.

Apêndice A

Programas Desenvolvidos em Matlab

Neste apêndice são apresentados alguns dos programas desenvolvidos em Matlab para análise de desempenho da estrutura proposta para o comutador CEP.

A.1 – Programa do Comutador para Pacote Fixo, Round-robin e FIFO

```
%Programa simulação Desempenho de uma Matriz ATM
%Carlos Roberto dos Santos
%=======================================================
%PREENCHIMENTO INICIAL DA MATRIZ
clc
N=input('Número de Entradas N=');
Rho=input('Carga do Tráfego =');
n=input('Número de Canais internos n=');
T=input ('NÚMERO DE CICLOS T=');
I=T/2;
M=zeros(I,N,N);
for k=1:N;
   for j=1:N;
      for i=1:I;
         M(i,j,k)=100;
      end
   end
end
disp('********* GERANDO A MATRIZ *********')
%GERAÇÃO DE DADOS PARA AS ENTRADAS
for t=1:T;
   for k=1:N;
      x=rand;                          %Gera um número aleatório
      x=N*x;                           %entre 0 e N, correspondente
      x=ceil(x);                       %a uma das saídas
      x1=rand;                         %Gera uma probabilidade de carga
      if x1<=Rho;
         for i=1:I;
            if M(i,x,k)==100;
               M(i,x,k)=x;
               break
            else M(i,x,k)=M(i,x,k);
            end
         end
         M(i,x,k)=x;
      else
         for i=1:I;
```

```
            if M(i,x,k)==100;
               M(i,x,k)=0;
               break
            else M(i,x,k)=M(i,x,k);
            end
         end
      end
   end
end
%for k=1:N;
   %disp(M(:,:,k));
%end
%v=input('wait;;;;;');
%============================================================
disp('************ TRANSFERINDO DADOS **********')
%ENCAMINHANDO OS DADOS
for j=1:N;
   a(j)=1;
   S(j)=0;
end
for t=1:T;
   for j=1:N;
      S(j)=0;
      for k=a(j):N;
         if (M(1,j,k)==0)
            for i=1:I-1;
               M(i,j,k)=M(i+1,j,k);
            end
            M(i+1,j,k)=100;
            if k==N;
               a(j)=1;
            %else
               %a(j)=a(j);
            end
         elseif(M(1,j,k))==100;
            if k==N;
               a(j)=1;
            end
         else
            for i=1:I-1;
               M(i,j,k)=M(i+1,j,k);
            end
            M(i+1,j,k)=100;
            S(j)=S(j)+1;
            if k==N;
               a(j)=1;
            else
               a(j)=a(j);
            end
            if S(j)==n;
               S(j)=0;
               a(j)=k+1;
               if a(j)>N;
                  a(j)=1;
               else
                  a(j)=a(j);
               end
```

```
                break;
            else
                a(j)=a(j);
            end
        end
    end
end
disp ('=======================================================');
%for k=1:N;
   %disp (M(:,:,k));
%end
count=0;
for k=1:N;
   for j=1:N;
      for i=1:I;
         if (M(i,j,k)==0)|(M(i,j,k)==100);
            count=count;
         else
            count=count+1;
         end
      end
   end
end
disp(count);
```

A.2 – Programa do Comutador para Pacote Fixo, Prioridade e FIFO

```
%Programa simulação Desempenho de uma Matriz ATM com Classe de Serviço;
%Carlos Roberto dos Santos
%=============================================================================
clc;
disp(clock);
n=2
COMP=zeros(5,1);media2=zeros(5,1);
nu_portas=16;nu_classes=5;ta_buffer=20;perda=zeros(5,1);%Define quantidade de
portas, de buffers e tamanho do buffer;
saida=zeros(nu_classes,nu_portas);
vazao1=zeros(nu_classes,nu_portas);vazao2=zeros(nu_classes,nu_portas);
mat=zeros(ta_buffer,nu_portas,nu_classes);%Gera a matriz de 3 dimensões;
for t=1:100000;%para controle;
   p=rand;
   if p<=.7;
      geracell=ceil(nu_portas*rand(nu_portas,1));
      geraprob=rand(nu_portas,1);
      %disp(geraprob);
      %disp(geracell);
      for porta=1:nu_portas;
         if geraprob(porta)<0.4;%Percentual de células da classe 1 (40%);
            classe=1;
         elseif geraprob(porta)>=0.4 & geraprob(porta)<.6;%Percentual da
classe 2(20%);
            classe=2;
         elseif geraprob(porta)>=0.6 & geraprob(porta)<.8;%Percentual da
classe 3 (20%);
            classe=3;
```

```
         elseif geraprob(porta)>=0.8 & geraprob(porta)<.9;%Percentual da
classe 4 (20%);
            classe=4;
         else;
            classe=5;
         end;
         for buffer=1:ta_buffer;
            if mat(buffer,porta,classe)==0;
               mat(buffer,porta,classe)=geracell(porta);
               break;
            elseif buffer==ta_buffer;
               perda(classe)=perda(classe)+1;
            end;
         end;
      end;
   end;
   %disp(p);
   %disp(mat);
   %disp (perda);
   %Fim da escrita de células na entrada;
   %=====================================================================
   %Transferência de células da entrada para a saída
   chave=zeros(nu_portas,1);c=zeros(nu_portas,1);
   for classe=1:nu_classes;
      for porta=1:nu_portas;
         if chave(porta)==0;
            a=mat(1,porta,classe);
            if a~=0;
               if c(a)<n;
                  c(a)=c(a)+1;
                  vazao1(classe,a)=vazao1(classe,a)+1;
                  saida(classe,a)=saida(classe,a)+1;
                  buffer=1;
                  while mat(buffer,porta,classe)~=0 & buffer<ta_buffer;
                     mat(buffer,porta,classe)=mat(buffer+1,porta,classe);
                     buffer=buffer+1;
                  end;
                  mat(buffer,porta,classe)=0;
                  chave(porta)=1;
               end;
            end;
         end;
      end;
   end;
   %disp(mat);
   %disp('=====================================================');
   %disp(saida);
   chavesai=zeros(nu_portas,1);
   for porta=1:nu_portas;
      for classe=1:nu_classes;
         if chavesai(porta)==0 & saida(classe,porta)>0;
            vazao2(classe,porta)=vazao2(classe,porta)+1;
            saida(classe,porta)=saida(classe,porta)-1;
            chavesai(porta)=1;
         end;
      end;
   end;
```

```
   %disp(saida);
   for k=1:nu_classes;
      B=find(mat(:,:,k));
      COMP(k)=COMP(k)+length(B);
   end;
media2=media2+sum(saida,2);
end;%fim do para controle;
Media=COMP/(nu_portas*t)
%disp(COMP);
Mediasai=media2/(nu_portas*t)
Mediatotal=Media+Mediasai
disp(clock);
%save Classe_n_6_N_32;
```

A.3 – Programa Simulação Sistema M/M/m

```
%Programa Simulação Sistema M/M/m
%Carlos Roberto dos Santos
%=========================================================================
clc;
clear all;
disp(clock);
capacidade=15000;%Define a Capacidade do Enlace
Max=1000000;
lambda=10;
B=1000;% Tamanho médio dos pacotes
m=3;
tempo_atendimento=zeros(1,Max);
tempo_chegada=zeros(1,Max);
tempo_partida=zeros(1,Max);
tempo_fila=zeros(1,Max);
tempo_sistema=zeros(1,Max);
for i=1:Max;
   tempo_atendimento(i)=(round(-log(rand)/(1/B)))/capacidade;%Calcula o tempo
de atendimento (exponencial)
end;
tempo_chegada(1)=0;
tempo_chegada(2)=-log(rand)/lambda;
tempo_chegada(3)=tempo_chegada(2)+-log(rand)/lambda;
tempo_partida(1)=tempo_atendimento(1);
tempo_partida(2)=tempo_chegada(2)+tempo_atendimento(2);
tempo_partida(3)=tempo_chegada(3)+tempo_atendimento(3);
tempomax1=tempo_partida(1);
tempomax2=tempo_partida(2);
tempomax3=tempo_partida(3);
tempo_fila(1)=0;
tempo_fila(2)=0;
tempo_fila(3)=0;
for i=m:(Max-1);
   tempo_chegada(i+1)=tempo_chegada(i)+-log(rand)/lambda;%Calcula os tempos de
Chegada
   if (tempo_chegada(i+1)<tempomax1 & tempo_chegada(i+1)<tempomax2&
tempo_chegada(i+1)<tempomax3);
      aux=min(tempomax1,tempomax2);
      tempo_partida(i+1)=min(aux,tempomax3)+tempo_atendimento(i+1);
```

```
   else
      tempo_partida(i+1)=tempo_chegada(i+1)+tempo_atendimento(i+1);%Calcula
tempo de partida
   end;
   aux=min(tempomax1,tempomax2);
   tempo_fila(i+1)=max(0,(min(aux,tempomax3)-tempo_chegada(i+1)));
   tempo_sistema(i)=tempo_partida(i)-tempo_chegada(i);
   if (tempomax1<tempomax2 & tempomax1<tempomax3);
      tempomax1=tempo_partida(i+1);
   elseif (tempomax2<tempomax1 & tempomax2<tempomax3);
      tempomax2=tempo_partida(i+1);
   else tempomax3=tempo_partida(i+1);
   end;
end;
tempo_sistema(i+1)=tempo_partida(i+1)-tempo_chegada(i+1);
mean(tempo_fila)
mean(tempo_sistema)
%tempo_chegada
%tempo_atendimento
%tempo_fila
%tempo_partida
%tempo_sistema
disp(clock);
```

A.4 – Fonte de Tráfego HTTP

```
% Geração_de_Fonte_HTTP.m
% Carlos Roberto dos Santos
%=========================================================================
clc
clear all
format short e
disp(clock);
Max = 5000;
tempo_de_chegada = zeros(4,Max);
tamanho = zeros(4,Max); % Tamanho do pacote
for pg=1:4;
   On = 1; % estado ativo
   lambda1 = 0.033; % Taxa do tempo de leitura
   MTU = 0; % 76% das vezes o tamanho maximo do pacote sera 1500 bytes e em
24% das vezes 576 bytes
   i = 1;
   Max_Local = 0;
   contador=0;
   contburst=0;
   contpersistent=0;
   b1 = waitbar(0,'Simulando fonte HTTP');
   while contador < Max
      if On == 1
         contador=contador+1;
         sorteio = rand;
         if sorteio <= 0.76
            MTU = 1500;
            MTU_40 = 1460;
         else
```

```
            MTU = 576;
            MTU_40 = 536;
         end

         sorteio = rand;
         if sorteio <= 0.5
            Max_Local = Max_Local + 1000;
            contburst=contburst+1;
            HTTP_Burst_4;
         else
            Max_Local = Max_Local + 1000;
            contpersistent=contpersistent+1;
            HTTP_Persistent_4;
         end
         On = 0;
      else
         tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda1;
         On = 1;
         i = i + 1;
      end
      waitbar(contador/Max,b1)
   end
   close(b1)
end
disp(clock);
```

A.4.1 – HTTP Persistent

```
% Simula uma fonte HTTP 1.1 Persistent Mode
% Carlos Roberto dos Santos
%=================================================================
On = 1; % estado ativo
lambda2 = 0.033;% Tempo de leitura
lambda3 = 7.69; % Taxa parsing Time
lambda4 = 20;   % Tempo de propagacao

% Parametros da distribuicao de Pareto
k = 1.75;
alfa = 1.1;
while i < Max_Local
    if On == 1
        % Envio de segmentos SYN+ACK e segmentos de controle
        tamanho(pg,i) = 80;
% Tempo de propagacao
        tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
        i = i + 1;
        % Tamanho do objeto principal segundo a distribuicao lognormal com
media 10710 bytes
        tamanho_objeto_principal = round(lognrnd(8.35,1.37));
        % Enquanto o tamanho do arquivo for maior que 2 MB e menor que 100
bytes gere outro tamanho aleatorio
        while (tamanho_objeto_principal > 2097152) | (tamanho_objeto_principal
< 100)
```

```
            tamanho_objeto_principal = round(lognrnd(8.35,1.37));
        end

        % Calcula-se o numero de pacotes necessarios para transmitir o arquivo
        n_pacotes = tamanho_objeto_principal/MTU_40;

        P = fix(n_pacotes);

        % Transferencia do objeto
        % Se o numero de pacotes for maior do que 1
        for j = 1:(P + 1)
            % Tempo de propagacao
            tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
            tamanho(pg,i) = MTU;
            i = i + 1;
        end
        % Parsing Time
        tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda3;
        i = i + 1;

        % Numero de objetos embutidos segundo a distribuicao de pareto com
media = 5.64
        n_objetos = round(k*(1 - rand)^(-1/alfa)); % Numero de objetos
embutidos
        % Enquanto o numero de objetos embutidos for maior do que 53 gera-se
outro numero aleatorio
        while n_objetos > 53
            n_objetos = round(k*(1 - rand)^(-1/alfa));
        end

        % Transmite 4 pacotes de 40 bytes
        tamanho(pg,i) = 160;

        Nd = 1;

        while Nd <= n_objetos
            % Tempo de propagacao
            tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
            i = i + 1;

            % Tamanho do objeto embutido segundo a distribuicao lognormal com
media 7758 bytes
            tamanho_objeto_embutido = round(lognrnd(6.17,2.36));
            % Enquanto o tamanho do objeto embutido for maior que 2 MB e menor
que 50 bytes gere outro tamanho aleatorio
            while (tamanho_objeto_embutido > 2097152) |
(tamanho_objeto_embutido < 50)
                tamanho_objeto_embutido = round(lognrnd(6.17,2.36));
            end

            % Calcula-se o numero de pacotes necessarios para transmitir o
arquivo
            n_pacotes = tamanho_objeto_embutido/MTU_40;
```

```
            P = fix(n_pacotes);

            for j = 1:(P + 1)
                tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
                tamanho(pg,i) = MTU;
            end

            Nd = Nd + 1;
        end

        tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
        % Envio do segmento FIN e do segmento ACK, fecha a conexao
        tamanho(pg,i) = 80;
        i = i + 1;

        On = 0;
    else % fim if On == 1
        break;
    end
end % fim while
```

A.4.2 – HTTP Burst

```
% Simula uma fonte HTTP 1.0 Burst Mode
% Carlos Roberto dos Santos
%=========================================================================
======
On = 1; % estado ativo

lambda2 = 0.033;% Tempo de Leitura
lambda3 = 7.69; % Taxa parsing Time
lambda4 = 20;   % Tempo de propagacao

% Parametros da distribuicao de Pareto
k = 1.75;
alfa = 1.1;

while i < Max_Local
    if On == 1

        % Tamanho do objeto principal segundo a distribuicao lognormal com
media 10710 bytes
        tamanho_objeto_principal = round(lognrnd(8.35,1.37));
        % Enquanto o tamanho do arquivo for maior que 2 MB e menor que 100
bytes gere outro tamanho aleatorio
        while (tamanho_objeto_principal > 2097152) | (tamanho_objeto_principal
< 100)
            tamanho_objeto_principal = round(lognrnd(8.35,1.37));
        end

        % Envio de segmentos SYN+ACK e segmentos de controle
        tamanho(pg,i) = 80;

        % Tempo de propagacao
```

```
        tempo_de_chegada(pg,i + 1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
        i = i + 1;

        % Calcula-se o numero de pacotes necessarios para transmitir o arquivo
        n_pacotes = tamanho_objeto_principal/MTU_40;

        P = fix(n_pacotes);

        for j = 1:(P + 1)
            % Tempo de propagacao
            tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
            tamanho(pg,i) = MTU;
            i = i + 1;
        end

        % Envio do segmento FIN e do segmento ACK, fecha a conexao
        tempo_de_chegada(pg,i + 1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda1;
        tamanho(pg,i) = 80;
        i = i + 1;

        % Parsing Time
        tempo_de_chegada(pg,i+1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda3;
        i = i + 1;

        % Numero de objetos embutidos segundo a distribuicao de pareto com
media = 5.64
        n_objetos = round(k*(1 - rand)^(-1/alfa)); % Numero de objetos
embutidos
        % Enquanto o numero de objetos embutidos for maior do que 53 gera-se
outro numero aleatorio
        while n_objetos > 53
            n_objetos = round(k*(1 - rand)^(-1/alfa));
        end

        resto_composto = 0;

        if (n_objetos/4) <= 1
            b = n_objetos;
            n_compostos = 1;
        else
            b = 4;
            n_compostos = fix(n_objetos/4);
            if (((n_objetos/4) - fix(n_objetos/4))*4) > 0
                resto_composto = ((n_objetos/4) - fix(n_objetos/4))*4;
            end
        end

        for i1 = 1:n_compostos
            objeto_composto = 0;
            for w = 1:b
                % Tamanho do objeto embutido segundo a distribuicao lognormal
com media 7758 bytes
                tamanho_objeto_embutido = round(lognrnd(6.17,2.36));
```

```
                % Enquanto o tamanho do objeto embutido for maior que 2 MB e
menor que 50 bytes gere outro tamanho aleatorio
                while (tamanho_objeto_embutido > 2097152) |
(tamanho_objeto_embutido < 50)
                    tamanho_objeto_embutido = round(lognrnd(6.17,2.36));
                end

                objeto_composto = objeto_composto + tamanho_objeto_embutido;
            end

            % Envio de segmentos SYN+ACK e segmentos de controle
            tamanho(pg,i) = 80*b;
            % Tempo de propagacao
            tempo_de_chegada(pg,i + 1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
            i = i + 1;

            % Calcula-se o numero de pacotes necessarios para transmitir o
arquivo
            n_pacotes = objeto_composto/MTU_40;

            P = fix(n_pacotes);

            for j = 1:(P + 1)
                tamanho(pg,i) = MTU;
                tempo_de_chegada(pg,i + 1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
                i = i + 1;
            end

            tamanho(pg,i) = 80*b;
            % Tempo de propagacao
            tempo_de_chegada(pg,i + 1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
            i = i + 1;
        end % fim for

        if resto_composto > 0
            for w = 1:resto_composto
                % Tamanho do objeto embutido segundo a distribuicao lognormal
com media 7758 bytes
                tamanho_objeto_embutido = round(lognrnd(6.17,2.36));
                % Enquanto o tamanho do objeto embutido for maior que 2 MB e
menor que 50 bytes gere outro tamanho aleatorio
                while (tamanho_objeto_embutido > 2097152) |
(tamanho_objeto_embutido < 50)
                    tamanho_objeto_embutido = round(lognrnd(6.17,2.36));
                end

                objeto_composto = objeto_composto + tamanho_objeto_embutido;
            end

            % Envio de segmentos SYN+ACK e segmentos de controle
            tamanho(pg,i) = 80*resto_composto;
            % Tempo de propagacao
            tempo_de_chegada(pg,i + 1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
```

```
            i = i + 1;

            % Calcula-se o numero de pacotes necessarios para transmitir o
arquivo
            n_pacotes = objeto_composto/MTU_40;

            P = fix(n_pacotes);

            for j = 1:(P + 1)
                tamanho(pg,i) = MTU;
                tempo_de_chegada(pg,i + 1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
                i = i + 1;
            end

            tamanho(pg,i) = 80 * resto_composto;
            % Tempo de propagacao
            tempo_de_chegada(pg,i + 1) = tempo_de_chegada(pg,i) + -
log(rand)/lambda4;
            i = i + 1;

        end % fim if resto_composto
        On = 0;
    else % fim if On == 1
        break;
    end
end % fim while
```