1998

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

DEPARTAMENTO DE TELEMÁTICA

Estudo de Desempenho de Algoritmos de Encaminhamento de Células em Comutadores ATM com Buffer na Entrada

Luiz Motomu Ono

Dissertação submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para obtenção do título de Mestre em Engenharia Elétrica.

Campinas

São Paulo - Brasil

Setembro , 1998

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE TELEMÁTICA

Estudo de Desempenho de Algoritmos de Encaminhamento de Células em Comutadores ATM com Buffer na Entrada

Luiz Motomu Ono

Orientador: Prof. Dr. Shusaburo Motoyama

Dissertação de Mestrado

Campinas
São Paulo - Brasil
Setembro , 1998

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

On6e

Ono, Luiz Motomu
Estudo de desempenho de algoritmos de encaminhamento de células em comutadores ATM com buffer na entrada. . / Luiz Motomu Ono.--Campinas, SP: [s.n.], 1998.

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

1. Comutação de pacotes (Transmissão de dados). 2. Algoritmos. 3. Rede digital de serviços integrados. I. Motoyama, Shusaburo. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Agradecimentos

Aos meus pais, Sheizi e Akiko, e aos meus irmãos, pelo apoio e incentivo na realização deste trabalho.

Ao professor Shusaburo Motoyama pela orientação, incentivo e paciência no desenvolvimento dos trabalhos.

Aos amigos que estiveram presentes nos momentos de estudo e nos momentos de lazer.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq - pelo suporte financeiro.

Para meus pais

Sumário

Lista de Figuras

Resumo

ONO, L. M.. **Estudo de Desempenho de Algoritmos de Encaminhamento de Células em Comutadores ATM com Buffer na Entrada.** Campinas: DT/FEEC/UNICAMP, 1998. (Dissertação de Mestrado)

Este trabalho objetiva o estudo de desempenho de algoritmos de encaminhamento de células em comutadores ATM com buffer na entrada. Comutadores com buffer na entrada apresentam vazão limitada e portanto o emprego de algoritmos de encaminhamento eficientes torna possível o aumento da vazão e a diminuição do tempo de espera de atendimento das células nos buffers. Algumas estruturas de comutadores com buffer na entrada apresentadas na literatura são estudadas e simulações são feitas para analisar o desempenho dos algoritmos de encaminhamento sob condições de tráfego binomial e em surtos. Duas estruturas de comutadores ATM são também propostas e feitos os seus estudos de desempenho. A primeira estrutura é baseada em comutadores com buffers na entrada e a segunda estrutura é uma proposta de implementação de um comutador matricial com um esquema de encaminhamento de células bastante simples.

Palavras-Chave: ATM, Comutação ATM, Encaminhamento de Células, Desempenho

Capítulo 1

1. Introdução

1.1. Introdução

Atualmente os sistemas de comunicação são caracterizados pela especialização no fornecimento dos seus serviços, ou seja, o sistema telefônico para tráfego de voz, redes de radiodifusão ou TV a cabo para vídeo ou televisão, redes de comutação de pacotes para tráfego de dados. Porém não há atualmente uma rede única capaz de suportar serviços com requisitos tão diferenciados como voz e teleconferência, com as qualidades específicas e necessárias para cada tipo de serviço. Portanto estudos vêm sendo feitos internacionalmente para a proposta de implementação de uma Rede Digital de Serviços Integrados (RDSI) que seja flexível e capaz de fornecer em uma plataforma comum os diversos serviços, bem como os novos serviços que venham a ser propostos. Os principais fatores que vêm impulsionando o desenvolvimento das RDSI são o desenvolvimento da tecnologia digital, por ser mais eficiente, confiável e econômico que os sistemas analógicos e a utilização da tecnologia de fibras óticas, que garante altas taxas de transmissão com baixas taxas de erro.

Para que haja um controle mais efetivo no desenvolvimento dessa rede, um esforço maciço de padronização vem sendo feito internacionalmente e em fins da década de 80, teve início a proposta de implementação de uma RDSI baseada na tecnologia de comutação rápida de pacotes, levando à definição pela ITU-T e ATM Forum das redes ATM (*Asynchronous Transfer Mode*), que surgem como uma das

formas mais promissoras para a implementação de redes digitais de serviços integrados.

A rede ATM tem como objetivo oferecer uma rede única e universal para a transmissão dos mais variados serviços, com os mais variados requisitos de taxas de transmissão, qualidade e eficiência na utilização de recursos.

Sua tecnologia é baseada na comutação rápida de pacotes. Cada pacote tem comprimento fixo de 53 *bytes* e é denominado célula. Para garantir a eficiência e rápido processamento na rede, a comutação rápida de pacotes diminui ao máximo o processamento nos nós de comutação da rede, deixando o processamento restrito às camadas mais baixas do modelo de referência OSI (*Open Systems Interconnection*). As principais simplificações no processamento efetuado pelos nós refere-se à eliminação da confirmação e retransmissão de quadros a nível de cada enlace, encaminhamento de pacotes baseado em conexões virtuais e a eliminação do controle de erros nos nós internos da rede. O controle de erros é feito somente fim a fim. Desse modo as funções do cabeçalho da célula ATM são simplificadas. Sua principal função é a identificação da conexão virtual por um identificador que é selecionado durante a fase de sinalização e garante o encaminhamento apropriado da célula na rede.

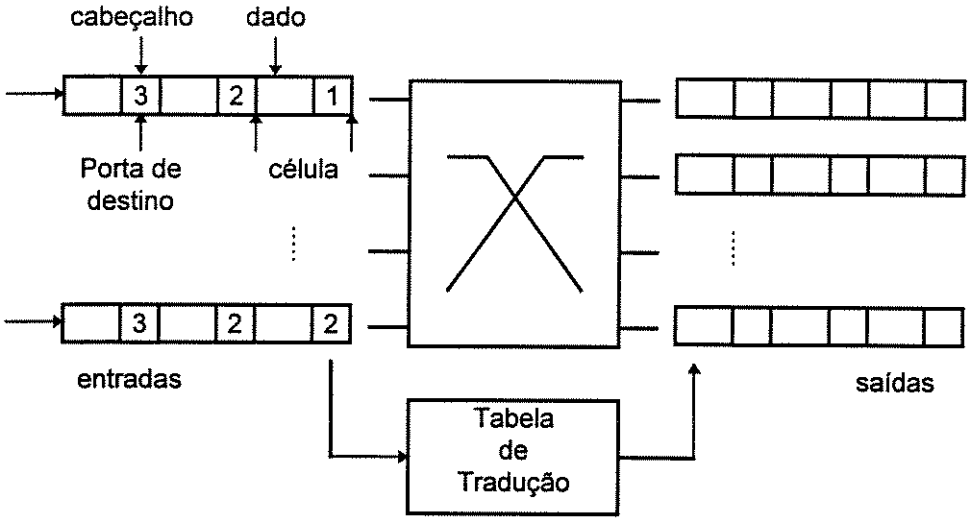


Figura 1.1 Princípio de comutação.

O princípio básico de comutação é mostrado na Fig. 1.1, onde N entradas transportam células ATM para o comutador, e dependendo do endereço do canal virtual apontado pelo cabeçalho da célula, esta célula é transferida para a porta de destino específica. A tabela de tradução é utilizada para efetuar a conversão dos endereços dos cabeçalhos de entrada para os novos endereços dos cabeçalhos das células na saída. Esse processo de comutação envolve o armazenamento de células em elementos chamados *buffers*, pois várias células podem chegar simultaneamente em diversas entradas e serem destinadas a uma única saída. Para reduzir o tamanho dos *buffers* nos nós de comutação e limitar os atrasos sofridos pelas células, o campo de informação das células é mantida pequena, proporcionando dessa maneira baixos valores de atraso necessários para serviços em tempo real. Considerando a posição do elemento de armazenamento de células na estrutura do comutador, tem-se descrito na literatura basicamente três arquiteturas: comutador com *buffer* na entrada, comutador com *buffer* na saída e comutador com memória compartilhada. Cada uma dessas arquiteturas oferece vantagens e desvantagens, em termos de complexidade de implementação em *hardware* e complexidade no algoritmo de encaminhamento de células utilizado. O estudo dessas arquiteturas é imprescindível, pois delas depende a eficiência da rede no transporte das células.

1.2. Objetivos do Estudo

Em [11] é apresentado o estudo de algoritmos iterativos para comutadores com *buffer* na entrada. Os algoritmos estudados são o PIM [1], o SPIM [14], o SLIP-IRRM [12] e o IRRM-MC [15]. No estudo foi utilizado fonte de tráfego binomial, porém este modelo de fonte é bastante simples e não reflete a realidade do tipo de tráfego esperado nas redes RDSI de faixa larga. Portanto é necessário verificar o comportamento dos algoritmos em condições mais próximas do tráfego esperado empregando, por exemplo, fontes de tráfego *on-off*.

Esta dissertação tem por objetivo dar continuidade ao trabalho realizado em [11], avaliando o desempenho dos algoritmos iterativos em presença de tráfego em surtos. Além dos estudos de desempenho dos algoritmos de encaminhamento, foram propostas duas outras arquiteturas de comutadores. A primeira arquitetura é baseada

no IRRM-MC, porém o encaminhamento de células é ponderado de acordo com a faixa de serviço. Espera-se com esse algoritmo tratar os diversos tipos de serviços de maneira mais eqüitativa do ponto de vista de atraso de células. A segunda arquitetura proposta é baseada em um comutador tipo matricial, com fator de crescimento do número de *buffers* igual a N^2.

O capítulo 2 apresenta uma visão geral sobre as principais arquiteturas de comutadores ATM encontradas na literatura. Algumas vantagens e desvantagens são apontadas para cada uma dessas arquiteturas. No final do capítulo são apresentados diversos comutadores com *buffer* na entrada e algoritmos de encaminhamento de células, tendo em vista o principal objetivo do trabalho: o estudo de comutadores ATM com *buffers* na entrada.

O capítulo 3 apresenta os resultados obtidos em simulação para os algoritmos iterativos de encaminhamento de células para comutadores com *buffer* na entrada. O capítulo inicia com a introdução dos tipos de fonte empregados para geração de tráfego de células nas simulações. Os resultados obtidos na simulação para os algoritmos PIM, SPIM, SLIP-IRRM e IRRM-MC quando submetidos a tráfego em surtos é comparado aos resultados obtidos em [11] com fonte de tráfego binomial.

O capítulo 4 apresenta uma proposta de arquitetura de comutador baseado no algoritmo IRRM-MC. O algoritmo IRRM-MC utiliza um esquema de ponteiros do tipo *Round-Robin* que realiza uma operação seqüencial de visitas em cada porta de entrada. O encaminhamento das células é feito por meio de prioridades de classes de serviço. No entanto, um serviço de faixa larga que porventura tenha sido colocado com menor prioridade poderá ser penalizado com tempos de atraso de células muito grande. No algoritmo proposto, um esquema de ponteiros do tipo *Round-Robin* também é utilizado, mas a freqüência de visitas em cada entrada é feita de modo ponderado, ou seja, com freqüência de visitas diferenciado em cada entrada. O principal objetivo é oferecer um desempenho de atraso de células mais eqüitativo entre as diversas faixas de serviço.

O capítulo seguinte é uma proposta de um esquema de encaminhamento de células em um comutador com estrutura matricial. A arquitetura proposta emprega um esquema extremamente simples de resolução de contenção em cada porta de saída. O desempenho da arquitetura é avaliado para fontes de tráfego em surto e fontes de

tráfego binomial. No apêndice é mostrado o modelo teórico para o atraso de células para este modelo de comutador, tanto para atendimento na mesma janela de tempo (*time slot*) quanto para a janela de tempo seguinte. Finalmente, o capítulo 6 apresenta os principais resultados obtidos nesta dissertação.

Capítulo 2

2. Arquitetura de Comutadores ATM

2.1. Introdução

Sistemas de comutação ATM têm como função básica efetuar o transporte de células de uma porta de entrada para uma porta de saída em um nó da rede. Este processo exige o armazenamento das células que chegam nas portas de entrada em memórias chamadas *buffers*, pois é possível que várias células estejam sendo destinadas para uma mesma porta de saída e, nesse caso, a formação de uma fila de espera é inevitável. A posição do *buffer* na estrutura do comutador determina basicamente três tipos de arquitetura de comutadores ATM: comutador com *buffer* na entrada, comutador com memória compartilhada e comutador com *buffer* na saída. As demais arquiteturas se originam dessas estruturas básicas. Neste capítulo são descritas as arquiteturas básicas de comutadores ATM, mostrando as principais vantagens e desvantagens, bem como algumas arquiteturas mais elaboradas encontradas na literatura [5],[6],[13],[18].

2.2. Comutadores ATM com Memória Compartilhada

Na arquitetura de comutador com memória compartilhada, apresentada na Fig. 2.1, as células que chegam são armazenadas em um espaço de memória comum, ou seja, não há um *buffer* dedicado ao armazenamento de células de uma única porta de entrada ou para uma única porta de saída. Portanto esta arquitetura apresenta a vantagem de otimizar o espaço de memória disponível, porém o fato de células de

diferentes portas de entrada estarem sendo armazenadas seqüencialmente torna necessário a implementação de um esquema de gerenciamento de memória mais complexo, como no caso de sistemas de computadores. Além disso, um comutador ATM de alta velocidade é impraticável de se realizar com uma estrutura clássica de memória compartilhada devido à limitação do tempo de acesso às memórias necessário para a escrita e a leitura das células a serem encaminhadas. Dois exemplos utilizando essa arquitetura são encontrados em [4]. O primeiro é conhecido como comutador Coprin e foi desenvolvido na França pelo CNET, em 1987. O segundo é conhecido como comutador Roxanne e foi descrito em 1990 pelos pesquisadores da Alcatel.

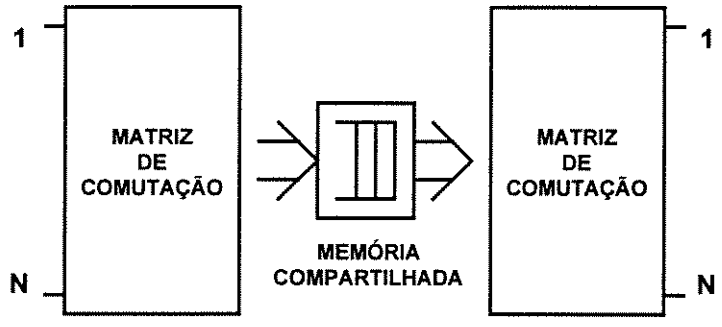


Figura 2.1 Arquitetura de comutador ATM com memória compartilhada.

2.3. Comutadores ATM com *Buffer* na Saída

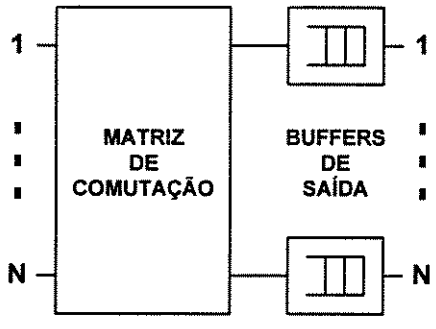


Figura 2.2 Arquitetura de comutador com *buffer* na saída.

A Fig. 2.2 mostra a arquitetura de um comutador com *buffer* na saída. Neste caso, as células que chegam ao comutador são armazenadas diretamente nos *buffers* das respectivas portas de destino. Esta configuração permite uma vazão teórica máxima de 100%, uma característica bastante desejável. Entretanto, o comutador deve operar a uma velocidade diferente das entradas e saídas e os elementos de memória devem ser suficientemente rápidos para armazenar todas as células que chegam ao *buffer* de saída em uma janela de tempo (*time slot*). O comutador *Knockout* é baseado na arquitetura de comutadores com *buffers* na saída e foi desenvolvido em 1987 pela *ATT Bell Laboratories*.

2.4. Comutadores ATM com *Buffer* na Entrada

A Fig. 2.3 mostra a arquitetura de um comutador com *buffer* na entrada. Neste caso as células que chegam são armazenadas nos *buffer*s de cada porta de entrada, ou seja, cada *buffer* possui células a serem encaminhadas para diferentes portas de saída. Neste tipo de arquitetura pode ocorrer o fenômeno conhecido como HOLB (*head-of-line blocking*), que limita a vazão do comutador.

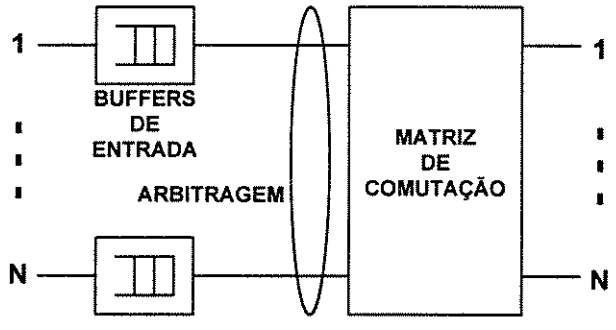


Figura 2.3 Arquitetura de comutador com *buffer* na entrada.

O problema de HOLB ocorre quando células chegam simultaneamente em diversas entradas e são destinadas a uma mesma saída. Suponha que uma célula de uma entrada i tenha sido selecionada para ser transmitida para uma saída j. Além disso, uma outra entrada k também possui uma célula destinada à mesma saída j. Neste caso, a célula da entrada k ficará impedida de ser transmitida, bem como as

demais células que estiverem esperando no *buffer* da entrada *k*. Suponha que a segunda célula deste *buffer k* esteja destinada para uma outra porta de saída *l*, que não está ocupada. Esta célula não poderá ser transmitida pois a primeira célula deste *buffer* está bloqueando a transferência das células. A Fig. 2.4 ilustra o problema de HOLB para um comutador 4 x 4.

Em [7] é analisado o desempenho deste tipo de arquitetura nas seguintes condições:

1. As chegadas das células em cada entrada são independentes e identicamente distribuídas.
2. Os processos de chegada de células em cada entrada são independentes das chegadas nas outras entradas.
3. Todos os processos de chegada têm a mesma taxa de chegada e o destino das células é uniformemente distribuído para todas as saídas.
4. Os *buffers* nas entradas são do tipo FIFO (*first-in-first-out*).

O estudo demonstrou que a vazão teórica máxima é de 58,6%. Para melhorar a vazão deste tipo de comutador, é possível utilizar *buffers* do tipo FIRO (*first-in-random-out*) e esquemas de pré-encaminhamento de células, como os algoritmos iterativos que serão apresentados.

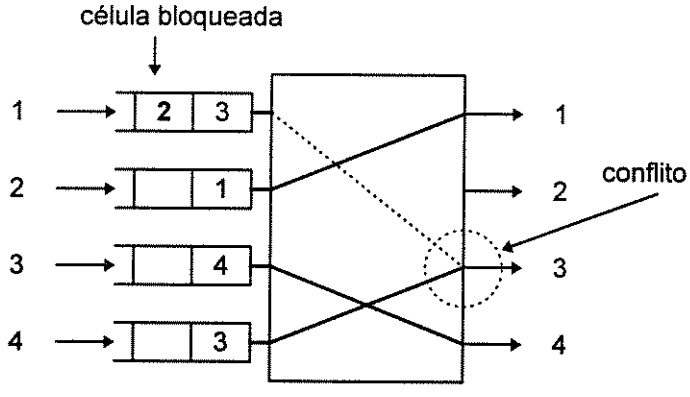


Figura 2.4 Bloqueio de células - *HOLB*.

Como visto anteriormente, comutadores com *buffer* compartilhado ou *buffer* na saída não apresentam o problema de contenção de células e portanto eram preferidos nos projetos de comutadores. No entanto, o aumento nas taxas de

transmissão de dados em fibras óticas e o crescente aumento do número de portas dos comutadores ATM tornou o projeto desses comutadores uma tarefa bastante difícil, uma vez que em ambas as arquiteturas os elementos de armazenamento de células necessitam operar a uma velocidade N vezes maior que os enlaces de entrada. Por este motivo os comutadores com *buffers* na entrada voltaram a ser considerados nos projetos de implementação de comutadores ATM. A seguir serão descritas algumas das estruturas mais representativas e de maior importância encontradas na literatura.

2.4.1. Comutadores com *Buffers* na Entrada - *Buffers* Divididos

Sabe-se que os comutadores com *buffers* na entrada têm a vazão limitada em 58,6% devido ao problema de bloqueio de células. Mas o resultado obtido por Karol aplica-se a uma arquitetura com um único *buffer* em cada porta de entrada do comutador. Esse problema pode ser eliminado com a utilização de uma simples estratégia de distribuição dos *buffers* nas entradas. Em cada porta de entrada são colocados N *buffers* do tipo FIFO, um para cada porta de saída, conforme indicado pela Fig. 2.5 (a). O problema de bloqueio de células é inteiramente eliminado uma vez que as células são previamente selecionadas e armazenadas nos *buffers* de cada porta de destino.

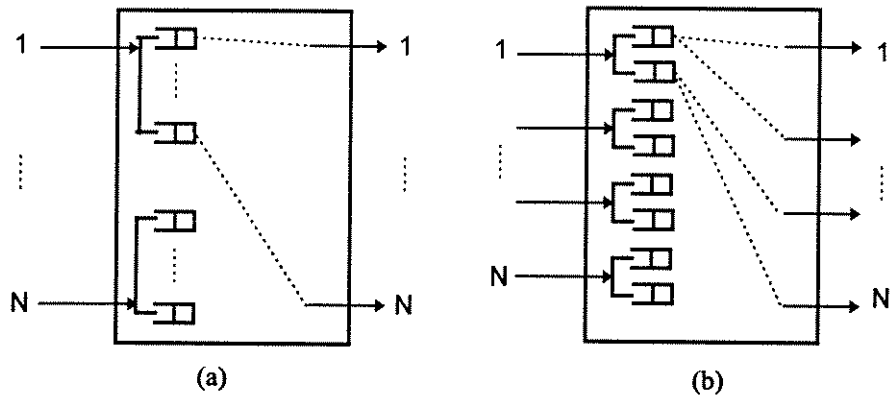


Figura 2.5 Arquitetura de comutadores com *buffers* divididos.

Em [9] e [13] são estudadas arquiteturas de comutadores com *buffers* na entrada, considerando N *buffers* em cada porta de entrada, um para cada porta de saída. Em [13] é mostrado que a vazão máxima para um comutador com esta

arquitetura pode chegar a 100%. A desvantagem desta configuração é que o número total de *buffers* tende a crescer na proporção N^2.

Um outro esquema possível é adotar uma solução intermediária entre um comutador com um único *buffer* em cada entrada e um comutador com N *buffers* em cada entrada. Em [18], é mostrado que a divisão da porta de entrada em k *buffers* ($k \leq N$) em um comutador N x N pode levar à redução do problema de bloqueio de células na entrada do comutador. A proposta tem origem na seguinte idéia intuitiva. Suponha que cada porta de entrada seja dividida em dois *buffers* (k=2) em vez de N *buffers,* sendo um *buffer* destinado a $N/2$ portas de saída e o outro *buffer* para as $N/2$ portas de saída restantes, como mostrado na Fig. 2.5 (b). O problema de bloqueio de células deve ser menor do que no caso para um comutador com *buffer* na entrada convencional, e portanto espera-se uma melhora na vazão. Para o caso de k >2, espera-se um aumento da vazão com o aumento do valor de k.

Para a arquitetura mostrada na Fig. 2.5 (b), pode-se considerar genericamente um comutador com N portas de entrada e N portas de saída. O tráfego é uniformemente distribuído entre as saídas e é dividido entre os k *buffers* em cada porta de entrada. Considere que $M=N/k$ é um número inteiro. Então em cada porta de entrada, se a célula que chega tem destino para a porta de saída m, onde m está na faixa de variação $0 \leq m \leq M-1$, a célula entra no *buffer* 1, se o destino da célula estiver na faixa $M \leq m \leq 2M-1$, então a célula entra no *buffer* 2, e assim por diante. Mostrou-se que para valores de k=2 a k=4, é possível obter uma melhoria na vazão bastante significativa, no caso, $\rho_2 = 0.76$ e $\rho_4 = 0.88$ para $k = 2$ e 4 respectivamente.

Uma proposta ligeiramente diferente foi considerada em [8], com um comutador ATM com 2 *buffers* em cada enlace de entrada. O primeiro *buffer* armazena células destinadas às saídas pares, e o segundo *buffer* armazena as células destinadas às saídas ímpares. Para esta configuração chegou-se a resultados próximos aos obtidos em [18] para k=2.

2.4.2. Algoritmo PIM (*Parallel Iterative Matching*)

Este algoritmo foi apresentado em [1] e é executado em três fases, para cada iteração. Em cada iteração, o algoritmo procura combinar cada entrada à sua saída correspondente. As três fases são:

1. Fase de Requisição (*Request Phase*) : cada entrada não combinada envia uma requisição para cada saída para a qual ela tenha um célula armazenada.
2. Fase de Concessão (*Grant Phase*) : se uma saída não combinada receber mais de uma requisição , então ela escolhe uma requisição aleatoriamente e envia um sinal de concessão para a entrada escolhida.
3. Fase de Aceitação (*Accept Phase*) : se uma entrada não combinada receber mais de uma concessão, então ela escolhe uma concessão aleatoriamente e envia um sinal de aceitação para a saída escolhida.

A Fig. 2.6 mostra um exemplo para uma iteração do algoritmo. Há duas células destinadas à saída 2 e uma célula destinada para a saída 4. Na fase de concessão, o algoritmo escolhe aleatoriamente uma entrada, no caso, a entrada 3. Na fase de aceitação, o algoritmo escolhe uma saída aleatoriamente, no caso, a saída 4. Ao final das três fases, a entrada 3 fica combinada com a saída 4. Como a saída 2 continua disponível, uma segunda iteração pode ser usada para combinar a entrada 1 com a saída 2.

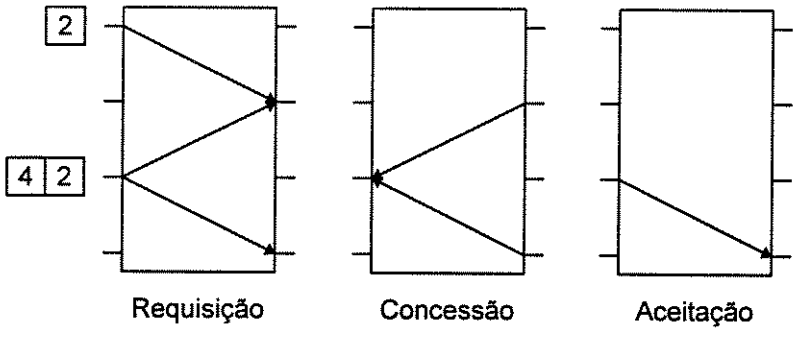


Figura 2.6 Exemplo para uma iteração do algoritmo PIM.

A questão sobre o número de iterações possíveis em cada janela de tempo para o algoritmo PIM também é abordada em [1], chegando-se à conclusão de que para quatro iterações a vazão chega a 99,9% para 100% de carga. Para um número de

iterações maior que quatro, praticamente não há um ganho significativo no desempenho.

2.4.3. Algoritmo SPIM (*Simplified Parallel Iterative Matching*)

O algoritmo SPIM [14] é semelhante ao PIM, com uma única diferença na fase de requisição: cada porta de entrada envia uma, e somente uma, requisição para a porta de saída. Como conseqüência, na fase de aceitação, o algoritmo apenas envia um sinal para as portas de saída para informar o estado de ocupação da porta de entrada.

2.4.4. Algoritmo SLIP-IRRM (*Iterative Round-Robin Matching with* SLIP)

Este algoritmo foi apresentado em [12] e é, similarmente ao algoritmo PIM, executado em três fases:

1. Fase de Requisição (*Request Phase*) : cada entrada não combinada envia uma requisição para cada saída para a qual ela tenha um célula armazenada.
2. Fase de Concessão (*Grant Phase*) : se uma saída não combinada receber mais de uma requisição , então ela escolhe a requisição que estiver mais próxima da posição apontada pelo ponteiro de *Round-Robin*. O ponteiro C_i é incrementado para a próxima posição depois da posição para onde o sinal de concessão foi enviado se, e somente se, o sinal de concessão foi aceito na terceira fase.
3. Fase de Aceitação (*Accept Phase*) : se uma entrada não combinada receber mais de uma concessão, então ela escolhe a concessão que estiver mais próxima da posição apontada pelo ponteiro de *Round-Robin*. O ponteiro A_i é incrementado para a próxima posição depois do elemento que recebeu o sinal de aceitação.

Similarmente ao algoritmo PIM, o algoritmo SLIP-IRRM pode executar mais de uma iteração por janela de tempo. Como a cada nova iteração são efetuadas mais combinações entre as entradas e as saídas, é possível melhorar a vazão do comutador.

2.4.5. Algoritmo IRRM-MC (*Iterative Round-Robin Matching with Mulitple Classes*)

Este algoritmo iterativo representa uma extensão do algoritmo apresentado na seção anterior para múltiplas classes de serviço [15]. A arquitetura do comutador é apresentada na Fig. 2.7.

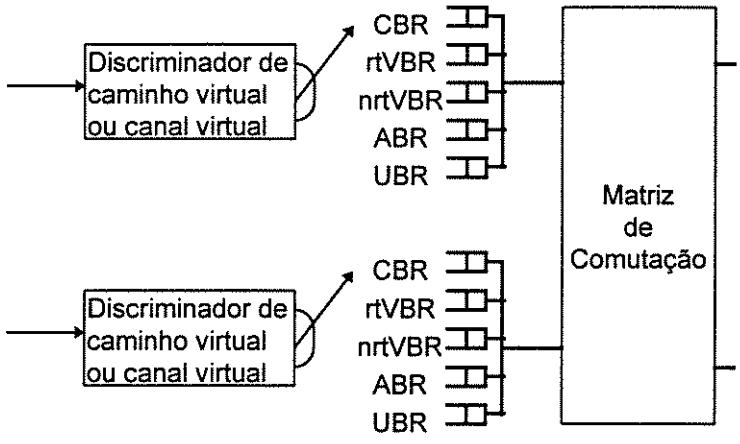


Figura 2.7 Modelo da arquitetura do IRRM-MC.

Cada porta de entrada é dividida em cinco *buffers*, um para cada tipo de serviço: CBR, rtVBR, nrtVBR, ABR e UBR. A vantagem apresentada é a facilidade de manter a qualidade de serviço para cada tipo de serviço. As células que chegam são discriminadas por um discriminador na entrada de cada enlace e são enviadas para os respectivos *buffers* de serviço. Em uma primeira iteração, o algoritmo utiliza os *buffers* da classe de serviço CBR para combinar as entradas e as saídas. Caso haja entradas não combinadas após a primeira iteração, o algoritmo executa uma nova iteração, mas desta vez utilizando os *buffers* da classe de serviço rtVBR. O processo é repetido para todas as classes de serviço ou até que todas as entradas estejam combinadas.

2.5. Estruturas Mistas

2.5.1. Comutador Tipo Matricial (*Crossbar*)

Comutadores matriciais operam com a mesma velocidade das portas de entrada. Como mostra a Fig. 2.8, em cada nó da matriz existe um *buffer* para armazenar as células que chegam. Células que chegam ao comutador são diretamente enviadas aos *buffers* do nó através da seleção por um filtro de endereços. A vazão é equivalente ao obtido pelo comutador com *buffer* na saída. A diferença com o comutador com *buffer* na saída é que o *buffer* é distribuído ao longo das saídas, mas apresenta como desvantagem o tamanho do *hardware* (proporcional a N^2). O capítulo 5 apresenta um estudo mais detalhado deste tipo de arquitetura.

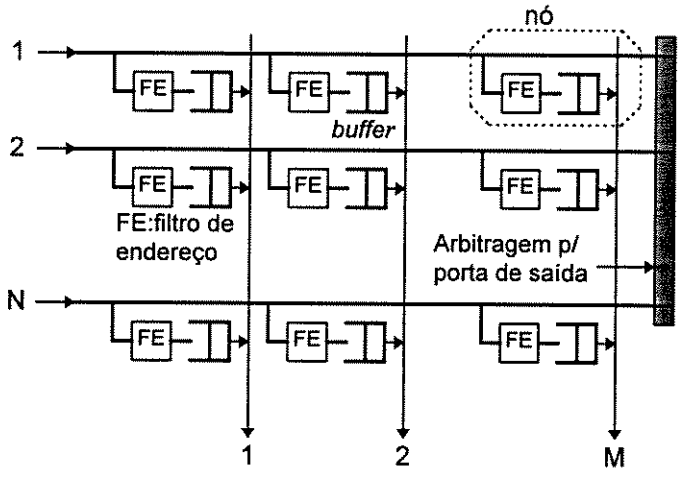


Figura 2.8 Comutador matricial convencional.

2.5.2. Comutador Tipo *Knockout* com *Buffers* na Entrada (KSI)

Em [17] é proposto um comutador com *buffer* na entrada baseado no comutador *knockout*. A vantagem, segundo o autor, é que a arquitetura apresenta desempenho semelhante ao comutador *knockout* convencional, porém apresenta uma redução na complexidade de *hardware* envolvida. A estrutura do comutador KSI (*knockout switch with input buffers*) é semelhante ao comutador *knockout* tradicional. A única diferença é a presença de um *buffer* na entrada, como mostra a Fig. 2.9. As células chegam e partem dos *buffers* nas janelas de tempo. Todas as células que chegam esperam nos *buffers* de entrada até que cheguem na primeira posição do

buffer, ou seja, no início do *buffer*. Quando uma célula estiver na primeira posição, poderá competir com outras células destinadas à mesma porta de saída. A competição ocorre em um concentrador que possui um número limitado de saídas. Somente as células vencedoras são transferidas para os respectivos *buffers* de saída. As outras células permanecem nos *buffers* de entrada até que eventualmente sejam transmitidas. Portanto não ocorre perda de células na entrada do concentrador.

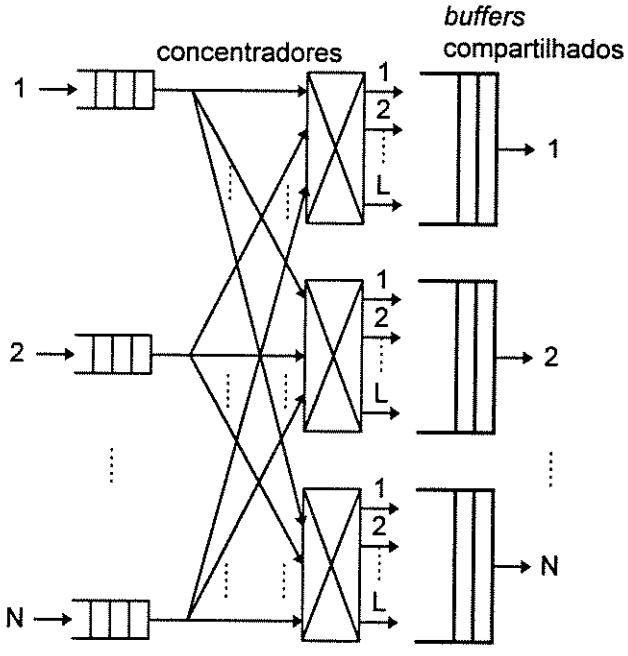


Figura 2.9 Modelo do comutador KSI.

Este processo é realizado através dos seguintes passos:

1. Copia-se o cabeçalho de cada célula a ser transferida, que contém o endereço de destino.
2. Enviar a cópia do cabeçalho da célula para o concentrador correspondente para a competição
3. Se vencer a competição, retorna uma mensagem de confirmação pelo caminho aberto pela cópia do cabeçalho da célula e então transfere a célula que recebeu a confirmação, caso contrário a célula não confirmada se mantém no *buffer* de entrada.

2.5.3. Comutador HSR (*High-speed Statistical Retry switch*)

A arquitetura apresentada em [6] é baseada em uma estrutura de matriz de comutação com *buffers* na entrada e na saída. O esquema de arbitragem empregado é simples e possibilita o seu emprego em comutadores ATM de alta velocidade.

A estrutura do comutador HSR é mostrado na Fig. 2.10. Os *buffers* são colocados na entrada e na saída do comutador e são interligados pelos nós da matriz de comutação. Cópia de uma célula é repetidamente transmitida de cada *buffer* da entrada com uma freqüência m vezes o período de entrada T do comutador ($1<m<N$, onde N é o número de portas do comutador). Define-se o período interno do comutador t como a relação entre m e T, isto é, dada por $t=T/m$.

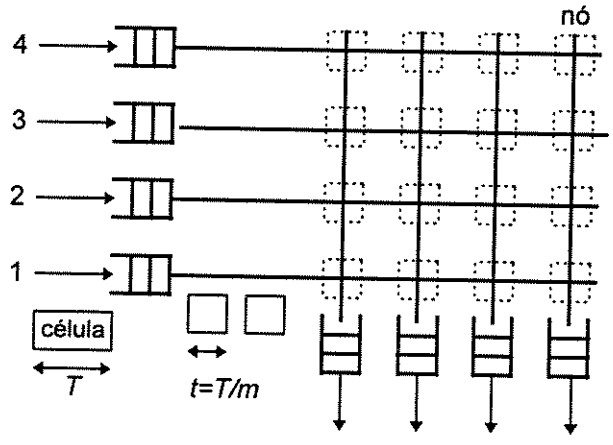


Figura 2.10 Arquitetura do comutador HSR.

Cada *buffer* da entrada transmite repetidamente a cópia de uma célula até receber um sinal de confirmação ACK (*acknowledge*) do *buffer* da saída. O sinal ACK indica o sucesso de transmissão da célula para a saída.

Durante cada janela de tempo T, uma célula, e somente uma, pode ser lida de cada *buffer* da entrada. Após receber o sinal de confirmação ACK, a transmissão da cópia da célula é cessada e a célula é extraída do *buffer* da entrada e enviada para o *buffer* da saída. Caso não ocorra um sinal de confirmação, o *buffer* da entrada continua a retransmitir uma cópia da célula na próxima janela de tempo T.

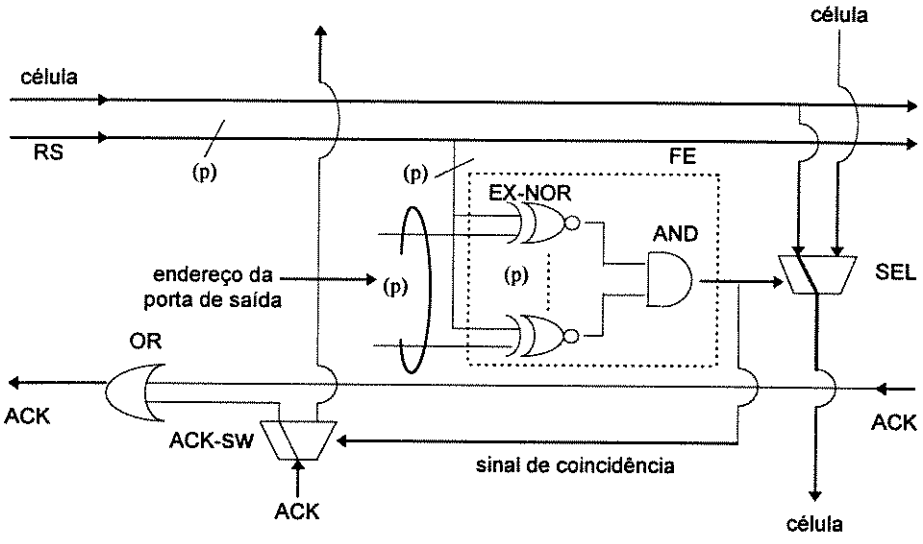


Figura 2.11 Diagrama do nó da matriz de comutação.

Na Fig. 2.11 cada nó consiste de um filtro de endereços (FE) , um seletor (SEL) para as células, uma chave para o sinal de confirmação ACK enviado para os *buffer*s da entrada e um sinal de encaminhamento (RS) enviado pelos *buffer*s da entrada. O sinal de encaminhamento, indicando a porta de destino, é enviado ao mesmo tempo que as células e permanece em nível constante durante o tempo de transmissão da respectiva célula. Os filtros de endereços comparam o sinal de encaminhamento com o endereço de cada *buffer* da saída e, havendo coincidência, envia um sinal de coincidência para o seletor SEL e a chave ACK-SW. O seletor SEL conecta cada linha vertical da matriz com a linha horizontal que recebeu o reconhecimento. A seleção, no entanto, é feita por meio de prioridades, sendo a entrada 1 a que possui a maior prioridade. O sinal de confirmação ACK é enviado pelo *buffer* da saída para o *buffer* da entrada para indicar aceitação da célula.

A Fig. 2.12 mostra a operação realizada pelo comutador. No exemplo, 3 células nas entradas 1,2 e 4 devem ser enviadas para uma mesma saída j. No primeiro período interno t_1 , a célula D da entrada 1, que possui a maior prioridade, é enviada para a saída j, enquanto que as células A e C são rejeitadas. Após a entrada 1 receber o sinal ACK, a retransmissão da célula D é interrompida. No próximo período interno t_2, as células A e C são retransmitidas e a célula C recebe o sinal de ACK. Finalmente, no período interno t_3, a célula A obtém a confirmação da saída.

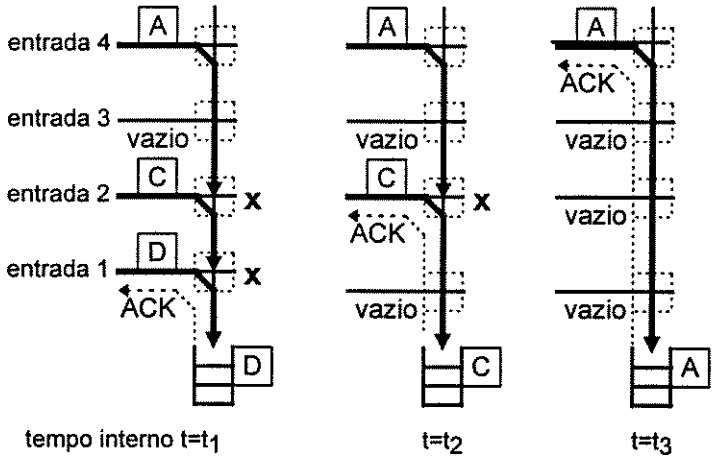


Figura 2.12 Exemplo de operação do HSR.

2.5.4. Comutador ICB (*Input and Crosspoint Buffering Switch*)

Esta arquitetura, apresentada em [5], possui a estrutura de um comutador matricial com *buffers* na entrada e nos nós da matriz. A Fig. 2.13 mostra o arranjo do esquema proposto, consistindo de *buffers* nas entradas e *buffers* nos nós da matriz. A velocidade de operação é a mesma das portas de entrada. Um algoritmo simples é empregado para dividir a carga entre os *buffers* da entrada e os *buffers* dos nós da matriz.

A Fig. 2.14 mostra o esquema de funcionamento do algoritmo utilizado para dividir a carga entre os *buffers*. O esquema de divisão da carga entre os *buffers* é conseguido com o uso de uma linha de controle que conecta os *buffers* da entrada com os *buffers* nos nós da mesma linha da matriz. A Fig. 2.14 (a) mostra uma célula sendo transmitida do *buffer* da entrada para o *buffer* do nó da respectiva porta de saída. Todas as células que chegam ao comutador são primeiramente armazenadas nos *buffers* da entrada do comutador. Uma cópia do cabeçalho da célula no *buffer* da entrada é enviada para os nós da matriz. Um filtro de endereços é utilizado para selecionar a célula cujo endereço coincide com o endereço da porta de saída. É considerado sucesso de transmissão se após um tempo de espera determinado o *buffer*

da entrada não receber um sinal de NACK (*not-acknowlegde*) da linha de controle. Em caso de sucesso de transmissão da célula ao respectivo *buffer* da matriz, a célula no *buffer* da entrada é removida. Há um esquema de arbitragem em cada coluna para efetuar a decisão do envio de células para os enlaces da saída em cada coluna da matriz.

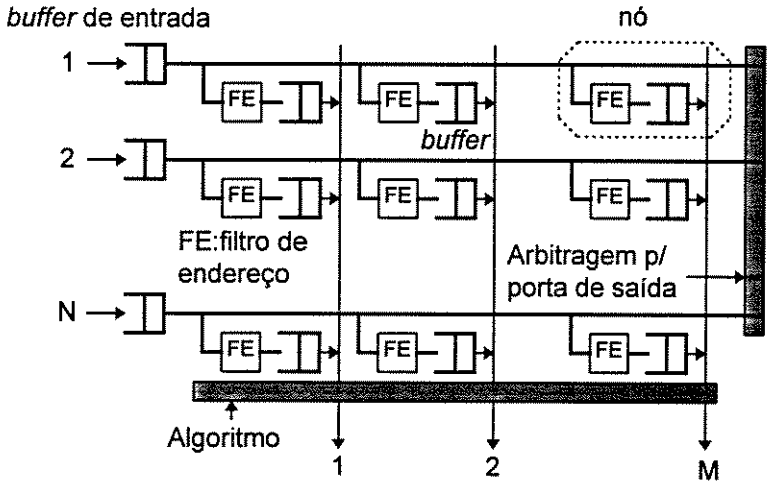
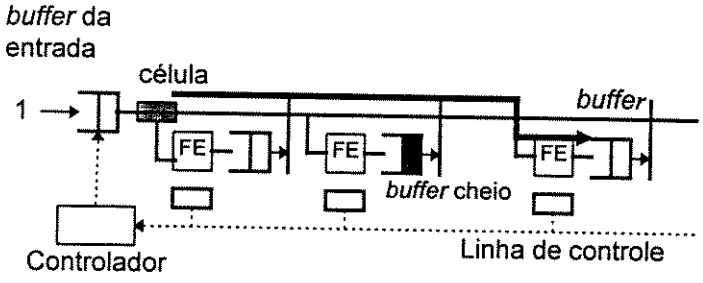
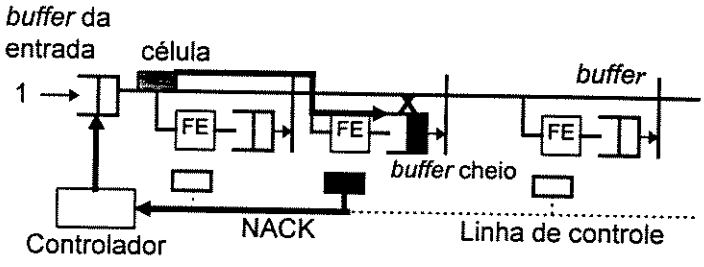


Figura 2.13 Esquema do ICB para comutadores ATM.

Pode ocorrer o caso de uma célula não poder ser enviada para o respectivo *buffer* da matriz pelo fato de o *buffer* estar cheio, como no caso da Fig. 2.14 (b). Nesse caso, o *buffer* do nó envia um sinal de NACK para indicar *buffer* cheio pela linha de controle e o *buffer* da entrada cancela a transmissão de cópias de células até que o nó em questão cancele o envio deste sinal de NACK. O nó determina o bloqueio (*buffer* cheio) com uma simples porta AND que compara o sinal de *buffer* cheio com o endereço da célula. O algoritmo requer um *hardware* muito simples e não necessita negociar com outras entradas ou programar os tempos de transmissão das células antes da fase de transmissão. O ICB pode ser implementado com simples memórias FIFO, uma função para cópia de células e um sistema de controle bem simples. No ICB, o bloqueio de células só ocorre nos *buffers* de entrada, quando ocorre um problema de contenção extremo nas portas de saída, enquanto que no comutador matricial convencional, a contenção pode ocorrer nos nós.



(a) Transmissão de uma célula.



(b) Bloqueio no *buffer* da matriz.

Figura 2.14 Transmissão de células entre os *buffers*.

A vantagem apresentada pelo autor é a diminuição do número de elementos de memória necessários para armazenar as células. Para um comutador 32 x 32 e uma taxa de perdas de células de 10^{-8}, o comutador ICB requer um *buffer* nos nós da matriz de somente 4 células e 32 células na entrada, enquanto que o comutador convencional requer 8 células nos nós da matriz para obter o mesmo desempenho. Ou seja, o tamanho do *hardware* pode ser reduzido a quase a metade em relação ao tamanho do comutador convencional. Quanto às características de atraso de células, o comutador ICB apresenta desempenho semelhante ao do convencional exceto para altas cargas (em torno de 80%).

2.6. Conclusão

Neste capítulo foram apresentadas as diversas arquiteturas de comutadores ATM para fundamentar o desenvolvimento desta dissertação. Estas arquiteturas são basicamente divididas em três arquiteturas: comutadores com *buffer* na entrada, comutadores com *buffer* na saída e comutadores com memória compartilhada. As

arquiteturas mais elaboradas encontradas na literatura combinam as técnicas utilizadas nessas estruturas básicas.

A principal desvantagem dos comutadores com *buffer* na saída ou com memória compartilhada é que os seus elementos de memória devem operar a uma velocidade N vezes maior que os enlaces de entrada. No entanto, o crescente aumento das taxas de transmissão em fibras óticas e o aumento do número de portas nos comutadores ATM tem forçado o redirecionamento das pesquisas para os comutadores ATM com *buffer* na entrada. Em particular, é possível eliminar o problema de bloqueio de células nos *buffers* com a adoção de uma simples estratégia: em cada porta de entrada do comutador são colocados N *buffers*, um para cada porta de saída. Desse modo, cada *buffer* irá armazenar células destinadas à mesma porta de saída. O desempenho dessa arquitetura irá depender essencialmente da estratégia de encaminhamento das células.

Capítulo 3

3. Análise de Algoritmos Iterativos

3.1. Introdução

Muitos estudos teóricos e simulações de comutadores ATM encontrados na literatura foram baseados em fontes de tráfego binomial, ou seja, as células que chegam em cada porta de entrada do comutador são independentes e identicamente distribuídas. O processo de chegada das células é independente para cada porta de entrada, com a mesma taxa de chegada e uniformemente distribuídas para todas as saídas. No entanto, o tráfego estimado em sistemas de comutação ATM provavelmente não obedece a este tipo de distribuição. O tráfego pode consistir de um surto de células destinados a uma mesma saída, como por exemplo, em serviços de vídeo. Portanto é necessário um modelo de fonte mais complexo para realizar estudos mais realísticos.

Os objetivos deste capítulo são estudar o comportamento dos algoritmos PIM, SPIM, IRRM-MC e SLIP-IRRM quando submetidos a tráfego *on-off* e comparar os resultados com aqueles obtidos através de fontes de tráfego binomial em [11].

3.2. Modelos de Fonte

3.2.1. Fontes de Tráfego Binomial

Em um modelo de fonte de tráfego binomial, as células são geradas de acordo com um processo aleatório discreto de Bernoulli. A cada janela de tempo, uma célula

é gerada na entrada com uma probabilidade p. Conseqüentemente, a probabilidade de não chegar nenhuma célula é 1 - p. A carga oferecida por porta de entrada é igual a p. Esse processo caracteriza-se pela geração de células de maneira independente a cada janela de tempo, não dependendo de chegadas de células em janelas de tempo anteriores e não influindo nas condições de geração de células nas próximas janelas de tempo, propriedade conhecida como processo "sem memória".

3.2.2. Fontes de Tráfego *On-Off*

No modelo de fonte *on-off*, a fonte alterna entre um período de atividade (*on*) e um período de silêncio (*off*). A distribuição de probabilidades de cada período obedece a uma distribuição geométrica. No período de atividade, as células são continuamente geradas para uma mesma saída. Este modelo de fonte considera dois parâmetros: o comprimento médio do surto L, em número de células, e a carga média ρ, definida como a probabilidade da fonte estar no período ativo. Define-se α como sendo a probabilidade de transição do período *on* para o período *off* (P_{on_off}) e define-se β como sendo a probabilidade de transição do período *off* para o período *on* (P_{off_on}). A partir do diagrama de estados da Fig. 3.1, obtêm-se as seguintes equações no estado estacionário:

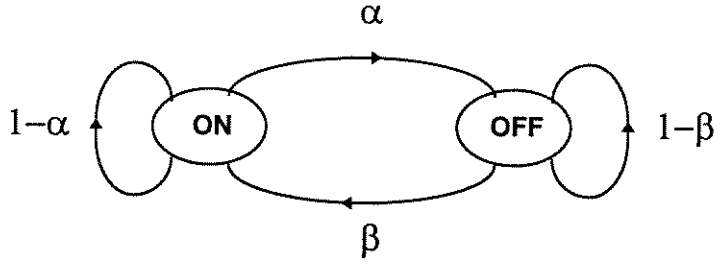


Figura 3.1 Diagrama de estados para fonte de tráfego *on-off*.

$$P_{on} = (1-\alpha)P_{on} + \beta \cdot P_{off} \qquad \textbf{Eq. 3.1}$$

$$P_{off} = (1-\beta)P_{off} + \alpha \cdot P_{on} \qquad \textbf{Eq. 3.2}$$

$$P_{on} + P_{off} = 1 \qquad \textbf{Eq. 3.3}$$

A carga média é definida como a probabilidade da fonte estar no período ativo e é dada por:

$$\rho = P_{on} \qquad \textbf{Eq. 3.4}$$

O comprimento médio do surto L é a média da distribuição geométrica que descreve a duração do surto em i janelas de tempo:

$$P(i) = (1-\alpha)^{i-1}\alpha \qquad \textbf{Eq. 3.5}$$

$$L = \sum_{i=1}^{\infty} iP(i)$$

$$L = \frac{1}{\alpha} \qquad \textbf{Eq. 3.6}$$

A variância correspondente é dada por:

$$\sigma^2 = \frac{1-\alpha}{\alpha^2} \qquad \textbf{Eq. 3.7}$$

Das Eqs. 3.1 e 3.2 obtêm-se:

$$P_{on} = \rho = \frac{\beta}{\alpha+\beta} \qquad \textbf{Eq. 3.8}$$

$$P_{off} = \frac{\alpha}{\alpha+\beta} \qquad \textbf{Eq. 3.9}$$

Dado que a fonte está no período de atividade (*on*) , a probabilidade da fonte continuar no período ativo é dado por:

$$P_{on\_on} = 1-\alpha$$

$$\boxed{P_{on\_on} = 1 - \frac{1}{L}} \qquad \textbf{Eq. 3.10}$$

A probabilidade β da fonte passar do período *off* para o período *on* é dada por:

$$\beta = P_{off\_on} = \frac{\rho}{L(1-\rho)} \qquad \textbf{Eq. 3.11}$$

Conseqüentemente, a probabilidade da fonte permanecer no período *off* é dada por

$$\boxed{P_{off\_off} = 1 - \beta} \qquad \textbf{Eq. 3.12}$$

As Eqs. 3.10 e 3.12 são utilizadas para descrever o comportamento da fonte de tráfego na simulação dos algoritmos. O modelo de fonte considerou que há pelo menos uma célula no período de atividade e pelo menos uma janela de tempo no período de silêncio. Portanto pode ocorrer que L tenha um surto de comprimento unitário e, nesse caso, a carga máxima ρ restringe-se a 50%. Deve-se levar em consideração os valores de L e ρ mais apropriados para cada simulação, uma vez que os valores escolhidos poderão afetar diretamente a variação da faixa de carga submetida ao sistema em simulação.

3.3. Estudo de Desempenho

3.3.1. Desempenho de Vazão para Tráfego Binomial

Em tráfego binomial, as células são geradas de acordo com o processo discreto de Bernoulli e o destino das células é distribuído uniformemente sobre todas as saídas. O parâmetro p da fonte de tráfego indica a probabilidade de chegada de uma célula na janela de tempo, ou ainda, a carga do sistema. A Fig. 3.2 [11] mostra a comparação para a vazão entre os algoritmos PIM e SPIM, para 100% de carga. É claramente visível o melhor desempenho obtido pelo algoritmo PIM, requerendo menos iterações

para chegar a 100% de vazão. Também é possível notar a melhor aproximação do algoritmo PIM em relação aos resultados teóricos [14]. Isto pode ser explicado pelo fato de que na fase de requisição do algoritmo PIM, maior número de requisições são enviadas para as saídas, o que produz um maior número de conexões entre a entrada e a saída por iteração. Isso permite uma melhoria na vazão. Por outro lado, verifica-se que quanto maior o número de iterações, mais próximo é o desempenho entre os dois algoritmos. Portanto, o algoritmo SPIM atinge praticamente os mesmos níveis de desempenho do algoritmo PIM com a vantagem de ter uma implementação mais simples. Com relação ao número de iterações, verifica-se que para um número de iterações maior que quatro, o ganho de desempenho tanto para o algoritmo PIM quanto para o algoritmo SPIM passa a ser desprezível. Ambos atingem quase 100% de vazão para quatro iterações e um número maior de iterações torna-se irrelevante. Um estudo similar foi feito em [2] considerando os algoritmos PIM e SLIP-IRRM, cujos resultados são semelhantes aos apresentados neste trabalho.

A Fig. 3.3 mostra a vazão em função da carga para os algoritmos SLIP-IRRM e IRRM-MC. Para o SLIP-IRRM, observa-se que a vazão se mantém em praticamente 100% para todas as cargas. Para o caso do algoritmo IRRM-MC, a curva de vazão foi obtida considerando a seguinte distribuição dos serviços: 40% da carga para a classe de serviço CBR, 20% para a classe de serviço rtVBR, 20% para a classe de serviço nrtVBR, 10% para a classe de serviço ABR e 10% para a classe de serviço UBR. As curvas mostram que para até 90% de carga total oferecida, as vazões para todas as classes são máximas, ou seja, todas as células que chegam são enviadas ao seus destinos. Com isso, a vazão chega a 100% para todas as classes, independente da carga oferecida. Para uma carga oferecida de 94%, a classe de serviço UBR exibe uma queda de desempenho substancial, o que pode ser explicado pelo fato de que em altas cargas, têm mais chance de serem transmitidas as células das classes de maior prioridade, diminuindo o uso das últimas classes de serviço.

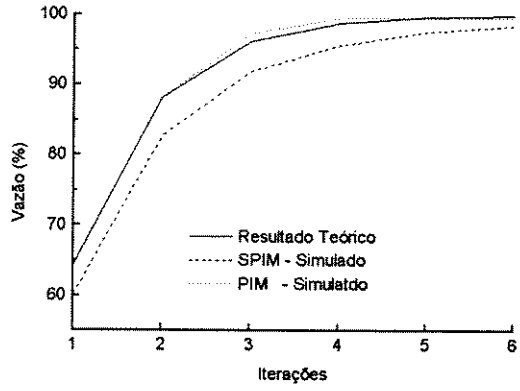


Figura 3.2 Vazão em função do número de iterações para PIM e SPIM.

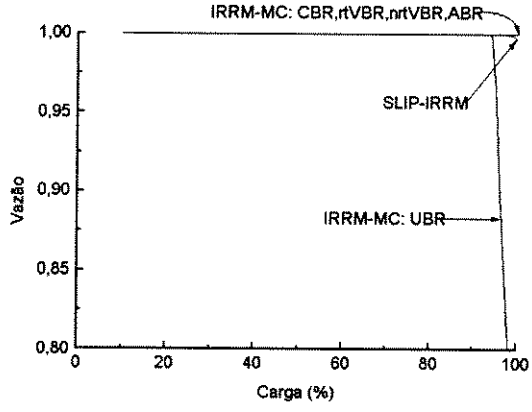


Figura 3.3 Vazão em função da carga para os algoritmos SLIP-IRRM e IRRM-MC.

3.3.2. Desempenho de Vazão para Tráfego *On-Off*

A simulação foi feita para um comutador com 16 portas, utilizando comprimentos médios de surto L = 10 e 15. Para L = 10, a carga média variou de 10 a 90%, uma vez que a máxima carga média possível utilizando uma fonte de tráfego *on-off* e L =10 é 90,90%. Para a segunda configuração, com L =15, a carga média variou de 10 a 93%. A Fig. 3.4 mostra os resultados obtidos para os algoritmos PIM e SPIM.

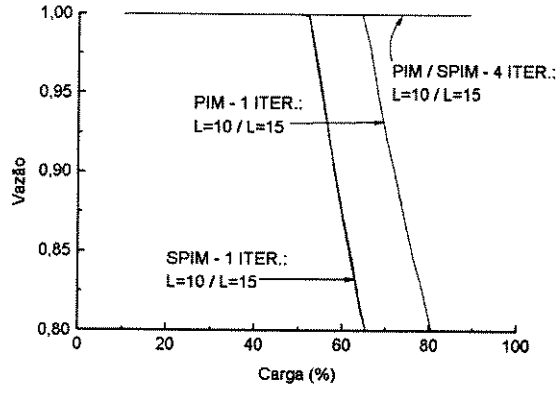


Figura 3.4 Vazão para comutador 16 x 16, utilizando algoritmos PIM e SPIM.

Para ambos os casos, L = 10 e 15, a vazão média exibe praticamente o mesmo comportamento. Para uma iteração a vazão média satura em torno de 50-55% para o algoritmo SPIM e em torno de 65% para o algoritmo PIM. Para quatro iterações, a simulação não mostrou nenhuma queda na vazão para as cargas máximas de ambos os algoritmos. Comparando com os resultados obtidos para fonte de tráfego binomial, nota-se que um surto de células pode afetar a vazão do comutador. Para o algoritmo SPIM, a vazão média caiu em torno de 10% para uma iteração. Considerando os algoritmos SLIP-IRRM e IRRM-MC, a Fig. 3.5 mostra que o desempenho do comutador não é afetado em presença de tráfego em surtos. Para cargas de até 90%, os algoritmos apresentam uma vazão de praticamente 100%, ambos os algoritmos utilizando L =10.

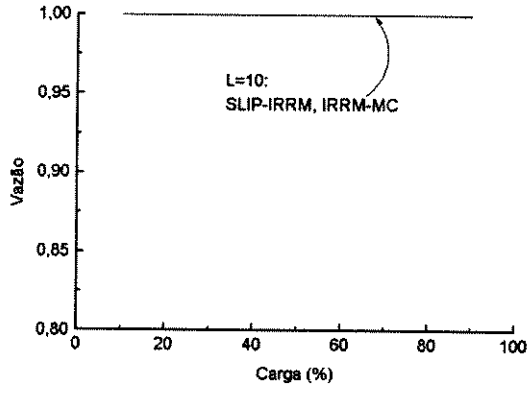


Figura 3.5 Vazão em função da carga para um comutador 16 x 16 utilizando os algoritmos SLIP-IRRM e IRRM-MC.

3.3.3. Desempenho de Atraso de Células para Tráfego Binomial

Como o atendimento de células não obedece ao esquema FIFO e não há um esquema de prioridades determinado , torna-se difícil definir um modelo teórico para a análise do atraso de células. Mas para um número razoável de portas, é possível modelar o processo de chegada como uma distribuição Poissoniana. Apesar de as células terem comprimento fixo, deve-se notar que para um comutador com *buffer* na entrada, o tempo de transmissão da célula não é determinística, pois inclui não somente o tempo gasto na transmissão da célula como também o tempo gasto pelas células que estão bloqueadas e o tempo de execução do algoritmo para efetuar o casamento entre as entradas e as saídas, tempo aleatório por natureza. Assumindo as condições acima, é possível aproximar os algoritmos PIM e SPIM como um modelo de fila tipo M/M/1, onde o comprimento médio das células é dado por:

$$L = \frac{\rho}{1-\rho}$$

onde ρ é a carga considerada.

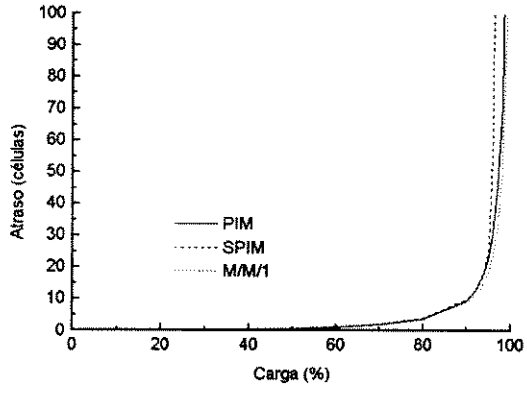


Figura 3.6 PIM e SPIM em comparação ao modelo teórico.

O gráfico da Fig. 3.6 mostra que o modelo de fila M/M/1 adotado é uma boa aproximação para os algoritmos PIM e SPIM. A Fig. 3.7 compara o atraso de células para os algoritmos PIM, SPIM, SLIP-IRRM e IRRM-MC. Para uma iteração, a curva para o algoritmo SPIM é o mesmo para o esquema FIFO. Para quatro iterações, o comutador atinge uma vazão em torno de 95%, sem degradação no desempenho de atraso de células, independente do algoritmo utilizado. Acima de 95%, o algoritmo

PIM exibe melhor desempenho, embora esta seja um região instável de operação. Para o algoritmo SLIP-IRRM, é possível atingir praticamente 100% de vazão assintoticamente [12].

A Fig. 3.8 mostra o atraso de células para as classes de serviço no algoritmo IRRM-MC. A curva mostra que a classe de serviço UBR é a única classe com região de instabilidade, com atraso de células tendendo ao infinito, pois como esta classe está alocada para menor prioridade, o tempo de atendimento tende a ser maior. Em geral, os tempos de atraso de células para todas as classes de serviço atingem um valor baixo para carga total oferecida de até 90%. É o caso da classe de serviço CBR, que exige tratamento em tempo real , possuindo tempo de atraso de menos de uma célula.

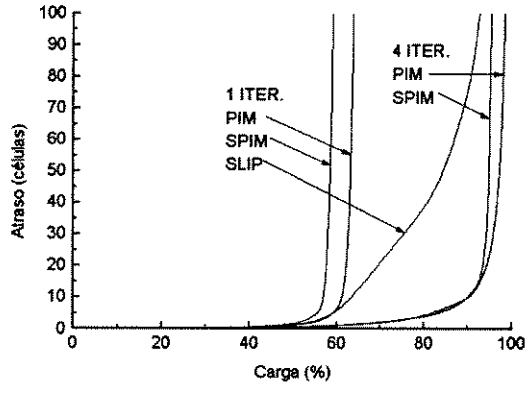


Figura 3.7 Comparação de atraso de células para PIM, SPIM e SLIP-IRRM.

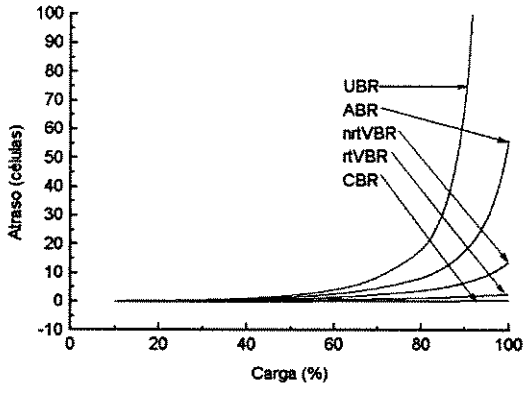


Figura 3.8 Atraso de células em função da carga para IRRM-MC.

3.3.4. Desempenho de Atraso de Células para Tráfego *On-Off.*

Considerando o desempenho dos algoritmos em relação ao atraso de células com o emprego de fontes *on-off*, verifica-se na Fig. 3.9 que há uma degradação significativa comparado com os resultados obtidos em [11]. O atraso de células tende a piorar quando é empregado um surto maior. Na Fig. 3.9, considerando o algoritmo SPIM e uma carga de 70% e quatro iterações, o atraso de células para L = 10 e 15 são respectivamente de 30 e 48 células, enquanto que para tráfego binomial é de menos de duas células.

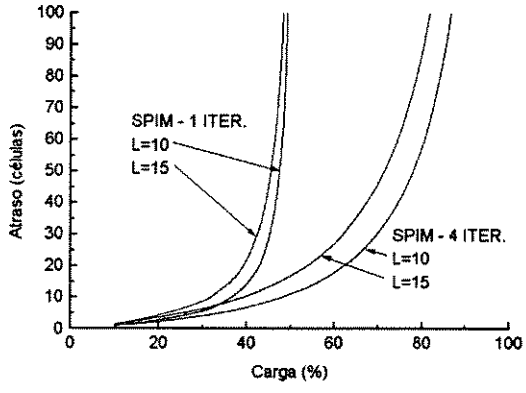


Figura 3.9 Desempenho para o algoritmo SPIM com fonte *on-off*.

O algoritmo PIM também exibe comportamento semelhante quando é submetido a tráfego em surtos. Na Fig. 3.10, verifica-se um desempenho um pouco melhor que o obtido pelo algoritmo SPIM. Observe que independente do tipo de fonte utilizado, é possível manter a vazão do comutador em até 100% utilizando quatro iterações. Porém no limite de carga oferecido, o atraso de células torna-se inaceitável.

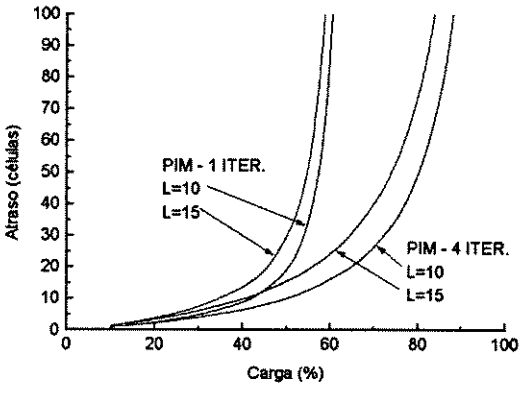


Figura 3.10 Desempenho para algoritmo PIM com fonte *on-off*.

Para os algoritmos SLIP-IRRM e IRRM-MC, os resultados obtidos seguem a mesma tendência, ou seja, há uma degradação significativa no atraso sofrido pelas células. Na Fig. 3.11, os resultados foram obtidos com uma iteração.

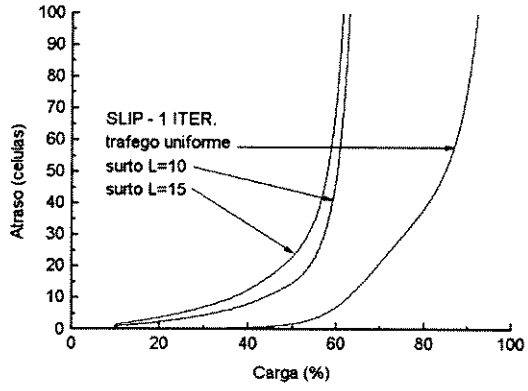


Figura 3.11 Desempenho para SLIP-IRRM com tráfego binomial e tráfego em surtos.

No algoritmo IRRM-MC para tráfego binomial, foi mostrado que somente o atraso de células da classe de serviço UBR tende para o infinito. Com o emprego de fontes de tráfego *on-off*, mais classes de serviço poderão ser afetadas , como é mostrado na Fig. 3.12, onde os atrasos de células das classes UBR e ABR tendem para o infinito. Para as classes de serviço de maior prioridade, ainda é possível manter um atraso de células bastante baixo.

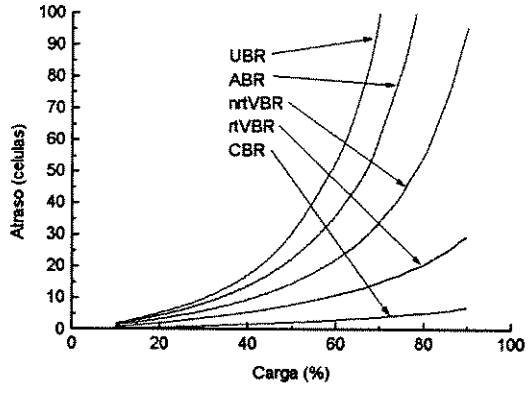


Figura 3.12 Desempenho para algoritmo IRRM-MC com fonte *on-off* e L =10.

3.4. Conclusão

Foi mostrado que, em presença de tráfego em surtos, os algoritmos de encaminhamento de células apresentam uma queda significativa de desempenho, notadamente em relação ao atraso sofrido pelas células, enquanto que o desempenho de vazão mostrou ser menos suscetível ao tipo de tráfego utilizado. Entre os algoritmos estudados neste capítulo, o algoritmo IRRM-MC apresenta a vantagem de permitir valores de atraso de células bastante baixos para os serviços de maior prioridade, mesmo com a utilização de fontes de tráfego em surtos.

Comparando o desempenho entre os algoritmos SPIM, PIM e SLIP-IRRM, verifica-se que o algoritmo SLIP-IRRM mostrou ser superior tanto para atraso de células como para vazão, quando submetido a tráfego binomial e tráfego em surtos. Considerando apenas uma iteração para cada algoritmo e tráfego binomial, o SLIP-IRRM apresenta o mesmo desempenho que os algoritmos SPIM e PIM para até aproximadamente 55% de carga. Acima de 60% de carga, o algoritmo SLIP-IRRM apresenta melhor desempenho, podendo chegar assintoticamente a 100% de vazão. Para tráfego em surtos, o SLIP-IRRM continua apresentando melhor desempenho. Analisando o aspecto de implementação em *hardware*, o algoritmo SLIP-IRRM possui a vantagem de não necessitar de um circuito para geração de números aleatórios, o que é difícil de ser implementado em *hardaware*. Seu funcionamento baseia-se em ponteiros de *Round-Robin*. Já os algoritmos SPIM e PIM requerem um circuito de geração de números aleatórios para as fases de requisição e concessão. No entanto, o algoritmo SPIM possui a vantagem de ter implementação mais fácil em relação ao PIM devido à simplificação na fase de requisição e a eliminação de fase de aceitação. Além disso, o desempenho do algoritmo SPIM é bastante próximo ao algoritmo PIM considerando quatro iterações tanto em presença de tráfego binomial quanto em presença de tráfego em surtos. O algoritmo IRRM-MC mostrou ter bom desempenho tanto para tráfego binomial quanto para tráfego em surtos. Para o atraso de células, o algoritmo IRRM-MC possui desempenho equiparável ou superior aos algoritmos SPIM, PIM e SLIP-IRRM (com 4 iterações) considerando as classes CBR, rtVBR, nrtVBR e ABR, e desempenho ligeiramente inferior para a classe UBR. As mesmas observações se aplicam quando o algoritmo IRRM-MC é submetido ao tráfego em surtos.

Capítulo 4

4. Comutador com Faixa Ponderada

4.1. Introdução

Na maioria das estruturas de comutadores ATM com *buffer* na entrada estudadas no capítulo 3, o tratamento de células é feito de maneira eqüitativa, isto é, cada enlace é visitado uma vez em cada janela temporal (*time slot*). Este tratamento pode acarretar atrasos maiores para serviços de faixa larga. No algoritmo IRRM-MC, o tratamento de células é feito de maneira prioritária. As conexões virtuais são separadas em classes com várias prioridades. Dessa maneira, as células que estão em classe de prioridade maior são atendidas com maior freqüência. Este esquema pode, entretanto, penalizar em demasia alguns serviços de faixa larga que são separados como menos prioritários. Pode-se conceber um outro critério de tratamento de células. Neste caso, o critério é baseado em manter os atrasos aproximadamente iguais e aceitáveis para todas as células.

Neste capítulo é proposto um comutador em que o tratamento das células é feito ponderando as faixas de serviço para obter atrasos aproximadamente iguais para todas as células.

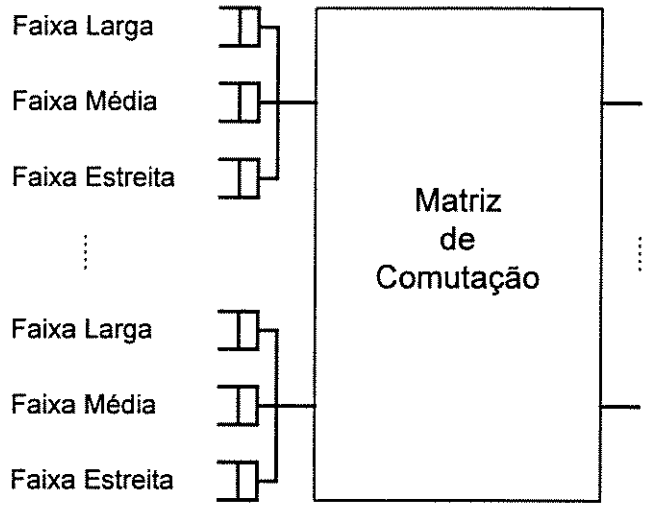


Figura 4.1 Esquema do comutador proposto.

4.2. O Algoritmo

A Fig. 4.1 mostra o esquema da um comutador com faixas ponderadas. O *buffer* de cada enlace de entrada é dividido em três *buffers*. Cada *buffer* recebe uma faixa de serviço. As faixas de serviço são: faixa larga, faixa média e faixa estreita. O algoritmo de encaminhamento de células proposto baseia-se em um conjunto de ponteiros tipo *Round-Robin*. Tradicionalmente um ponteiro é incrementado de modo seqüencial para uma próxima entrada depois que a entrada atual enviou a sua célula. Porém pode acontecer que alguma entrada receba mais células que as outras entradas levando a uma distribuição não muito uniforme da banda disponível para as entradas, como ocorre por exemplo, em diferentes tipos de serviço. Uma alternativa é aumentar a freqüência de atendimento nas entradas que demandem maior banda através do uso de pesos proporcionais às faixas de serviço.

Na Fig. 4.2 é ilustrada a diferença do esquema de *Round-Robin* tradicional com o esquema empregando pesos diferenciados para cada faixa de serviço nas entradas. Na Fig. 4.2 (a), o algoritmo seleciona uma célula de cada entrada de forma seqüencial e atende uma entrada por vez, quando esta entrada tiver alguma célula a ser enviada. Já no esquema da Fig. 4.2 (b), o algoritmo atende de forma igualmente seqüencial, mas a entrada que tiver maior peso terá uma freqüência de visitas maior. O peso indica a freqüência de visitas em cada entrada. No caso ilustrado, supondo que o

ponteiro do círculo de *Round-Robin* inicialmente esteja apontando para o serviço de Faixa Larga, o incremento do ponteiro somente será feito para uma próxima entrada após 3 visitas consecutivas ou quando não houver mais células a serem enviadas.

O algoritmo é executado em 2 etapas: na primeira etapa, o algoritmo consulta cada entrada para verificar qual faixa de serviço está com a prioridade de envio de células. Cada entrada possui um ponteiro de *Round-Robin* para controlar qual faixa de serviço tem a prioridade de envio de células. Com a adoção de pesos na entrada de cada *buffer*, este ponteiro de *Round-Robin* só será incrementado para uma próxima faixa de serviço caso o número de visitas tenha se esgotado ou não existam mais células a serem encaminhadas. Feita a escolha, na etapa seguinte o algoritmo deverá combinar as entradas que tiverem células para enviar para as respectivas saídas. Esta segunda etapa é executada iterativamente 3 vezes e o esquema empregado é o mesmo aplicado no SLIP-IRRM, ou seja, há dois conjuntos de ponteiros de *Round-Robin*, um para cada entrada e outro para cada saída, controlando os processos de requisição e concessão.

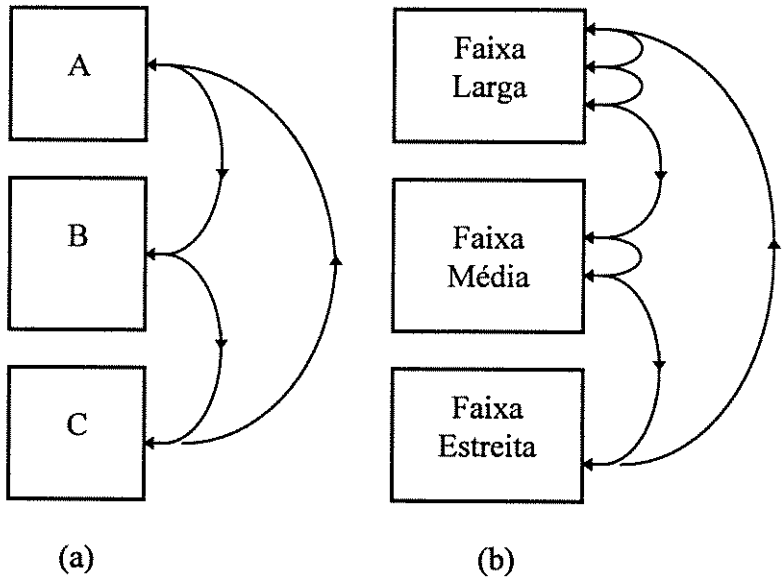


Figura 4.2 *Round-Robin* (a) e *Round-Robin* com ponderação (b).

4.3. Simulações

Neste trabalho foram feitas várias simulações considerando diferentes distribuições de carga para cada tipo de serviço, por exemplo, 30% para serviço de faixa larga, 40% para faixa média e 30% para faixa estreita. Para simular as faixas de serviço foram consideradas fontes do tipo *on-off* gerando surtos de comprimento médio L=25 para faixa larga, L=15 para faixa média e L=10 para faixa estreita. O tempo de simulação considerado foi de 1000000 de janelas de tempo em um comutador com 16 portas de entrada e variando a carga na entrada até 90%. As simulações mostraram que a distribuição ideal dos pesos é proporcional tanto ao surto quanto à distribuição de carga para cada tipo de serviço. Para exemplificar, considere que as faixas de serviço estão chegando com a distribuição citada acima:30% para faixa larga, 40% para a faixa média e 30% para a faixa estreita. Pode-se obter os pesos (número inteiro mais próximo dos valores encontrados) para cada faixa de serviços da seguinte forma:

$$peso = P_{f1} * \frac{S_{f1}}{S_{f1} + S_{f2} + S_{f3}}$$

onde:

Pf: Porcentagem de carga da faixa de serviço,

Sf: comprimento médio do surto.

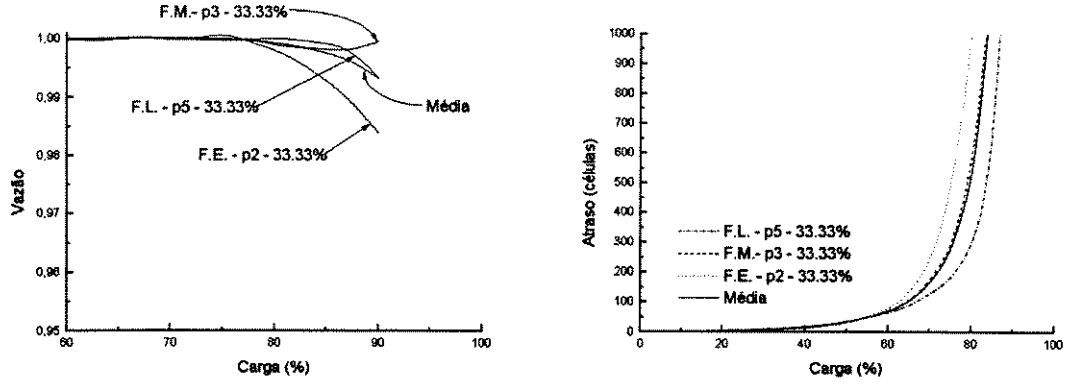
Para o exemplo citado acima encontram-se os seguintes pesos: p=5 para faixa larga, p=4 para faixa média e p=2 para faixa estreita. Portanto ao executar o algoritmo, cada entrada que estiver apontando para uma faixa larga de serviço deverá enviar 5 células para o seu destino ou até que não existam mais células a serem enviadas e então passa a prioridade para uma faixa de serviço menor. Da mesma forma, cada entrada que estiver apontando para uma faixa média de serviço deverá enviar 4 células para o seu destino ou até que não existam mais células a serem enviadas para enfim passar a prioridade para a faixa de serviço menor. O mesmo ocorre com a terceira

faixa de serviços. Terminada a seqüência, a prioridade volta para a primeira faixa de serviço.

Na Fig. 4.3, foi considerada distribuição uniforme para as faixas de serviço, ou seja, aproximadamente 33% de carga para cada faixa de serviço. Os pesos obtidos foram p=5 (p5) para serviço de faixa larga, p=3 (p3) para serviço de faixa média e p=2 (p2) para serviço de faixa estreita. A vazão se manteve em 100% até aproximadamente 80% da carga total na entrada. Ainda assim o comutador consegue manter uma vazão acima de 97% até 90% de carga. Quanto ao desempenho no atraso de células, as 3 faixas de serviço obtêm características de atraso semelhantes.

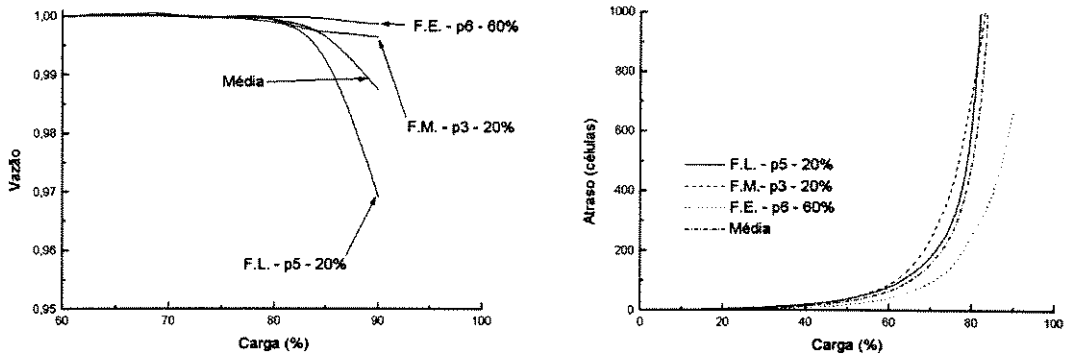
Na prática porém, a distribuição de carga para os diferentes tipos de serviço varia muito e para simular tal condição foram consideradas várias distribuições de carga mostradas a seguir nos gráficos das Figs. 4.4 a 4.7. A Fig. 4.4 mostra o desempenho do comutador considerando pesos para a proporção exata em cada faixa de serviço, cujos resultados de atraso de células se aproximam da Fig. 4.3.

Uma questão a ser levantada é se há alguma interferência no número de ciclos (soma dos pesos), pois é possível encontrar várias proporções de pesos para uma mesma distribuição de carga. Nesse caso, o aumento do número de ciclos pode afetar o desempenho de atraso do comutador? As Figs. 4.6 e 4.7 mostram que o tamanho do ciclo não afeta o desempenho do comutador, tendo a Fig. 4.6 um ciclo de 15 visitas e a Fig. 4.7 um ciclo de 30 visitas. Verificou-se que o atraso médio de células para ambos os casos é praticamente o mesmo.



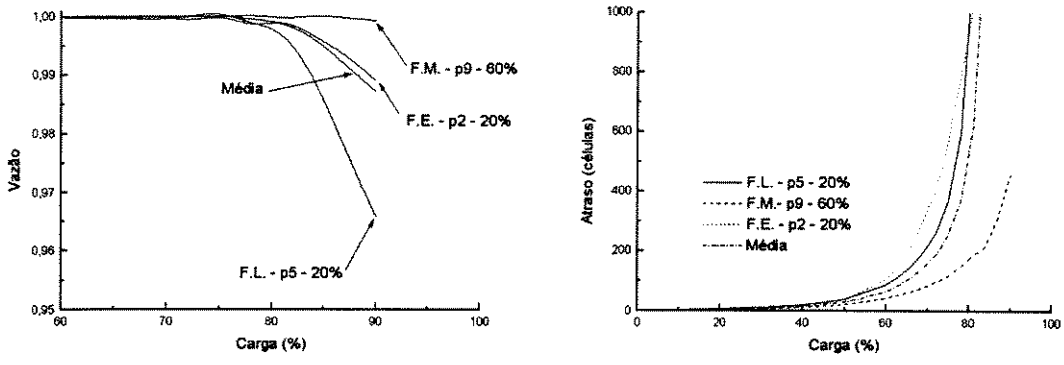
(a) (b)

Figura 4.3 Resultados para distribuição uniforme de carga para as faixas de serviço, (a) Vazão e (b) Atraso de células.



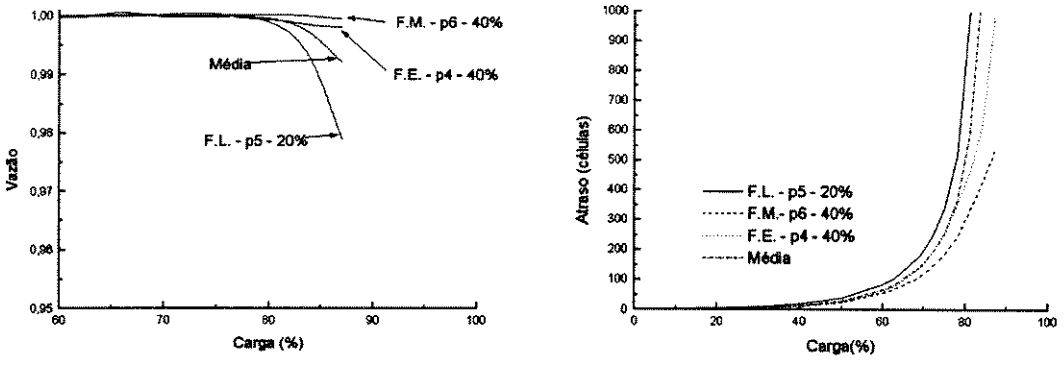
(a) (b)

Figura 4.4 Distribuição de pesos proporcionais. (a) Vazão, (b) Atraso de células.



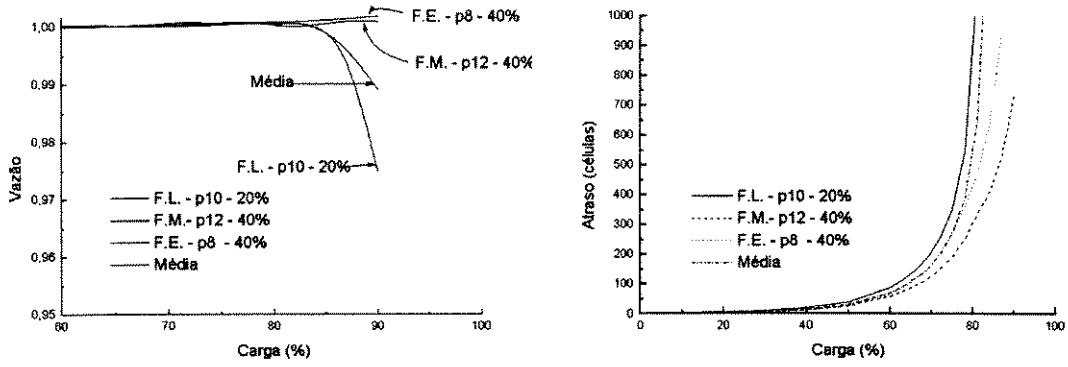
(a) (b)

Figura 4.5 Distribuição de pesos proporcionais. (a) Vazão, (b) Atraso de células.



(a) (b)

Figura 4.6 Desempenho para comutador com ciclo de 15 visitas. (a) Vazão, (b) Atraso de células.



(a) (b)

Figura 4.7 Desempenho para comutador com ciclo de 30 visitas. (a) Vazão, (b) Atraso de células.

4.4. Conclusões

As simulações foram feitas para casos estáticos, ou seja, foram simuladas diversas distribuições de carga, sendo as condições de carga fixas ao longo de cada simulação. Os resultados mostraram que o algoritmo obtém melhor desempenho quando a relação de proporcionalidade de carga de serviço e comprimento médio do surto é mantida. Mantendo essas proporções, é possível atingir no mínimo 95% de vazão para as condições estudas e mantendo condições de atraso semelhantes independentemente da faixa de serviço considerada. No entanto, o afastamento dessa relação de proporcionalidade conduz a uma drástica queda de desempenho para uma ou mais faixas de serviço. Para a maioria dos casos simulados com o emprego dos pesos apropriados, os resultados foram satisfatórios. Em alguns casos uma das classes ficou prejudicada.

Como é possível escolher vários conjuntos de pesos para cada configuração de carga, surge a dúvida sobre a possibilidade de que ciclos de tamanhos diferentes para uma dada configuração de carga interfiram no desempenho de atraso de células do comutador. A simulação mostrou que o tamanho do ciclo, ou seja, a soma dos pesos, não influi no desempenho do comutador, como mostram as Figs. 4.6 e 4.7 .

Tendo em vista os resultados obtidos, é possível pensar em um algoritmo que seja executado dinamicamente de forma que seja possível calcular os pesos para cada

tipo de serviço sempre que houver variação na distribuição de carga das faixas de serviço. O cálculo pode ser feito durante a fase de sinalização tendo como parâmetros para cálculo dos pesos a taxa de pico do tráfego e a taxa média, quando da fase de sinalização. As faixas de serviço são separadas nas entradas através de um discriminador de caminho virtual ou canal virtual.

Capítulo 5

5. Comutador Matricial (*Crossbar Switch*)

5.1. Introdução

Comutadores ATM com *buffers* na saída têm sido muito estudados nos últimos anos pelo fato de não apresentarem o problema de bloqueio de células que afeta os comutadores com *buffer* na entrada. Porém os comutadores com *buffer* na saída devem operar com velocidade interna N vezes maior que a velocidade das portas de entrada para garantir a transferência de células, o que torna esta arquitetura problemática devido à demanda crescente por maiores taxas de transmissão e a tendência do aumento do número de portas dos comutadores. Portanto o estudo de comutadores com *buffers* na entrada voltou a ter importância e muitas propostas têm sido feitas considerando o *buffer* na entrada dividido em vários *buffers*. Para o caso de divisão máxima, ou seja, para cada entrada com N *buffers*, um para cada porta de saída, o número destes elementos tende a crescer na proporção N^2. Como a arquitetura matricial apresenta esse mesmo crescimento e essa característica representa a sua principal desvantagem, então este capítulo reanalisa em detalhes o comutador matricial. Em um comutador matricial, os *buffers* são colocados em cada nó da matriz de comutação. A vantagem é que não ocorre o problema de bloqueio de células. É necessário apenas um elemento de controle em cada coluna dessa matriz de comutação para escolher a célula a ser transmitida para a saída. Neste capítulo é proposto um esquema bastante simples para o controle. Os resultados de desempenho

também são apresentados, considerando fontes de tráfego binomial e em surto e com atendimento na mesma janela de tempo e na próxima janela de tempo.

5.2. Sistema de Controle para o Comutador Matricial

O esquema proposto baseia-se em uma matriz de comutação utilizando um círculo de ponteiros tipo *Round-Robin* para seleção de células. Em cada coluna há um bloco de controle para realizar a função de arbitragem, selecionando a porta que tiver a preferência de envio de células.

Nesta matriz, as linhas representam as portas de entrada do comutador e as colunas representam as portas de saída. Em cada linha há um *buffer* conectado diretamente a cada saída, possibilitando uma pré-seleção das células que chegam, ou seja, cada célula que chega será armazenada no *buffer* da respectiva porta de destino, sendo esta operação feita através de um filtro de endereços (FE) situado em cada entrada dos *buffers*. Em cada coluna a função de arbitragem seleciona a porta que tiver a prioridade de envio e faz as operações necessárias para atualizar o sistema. O sistema é bastante simples e o seu esquema é mostrado na Fig. 5.1.

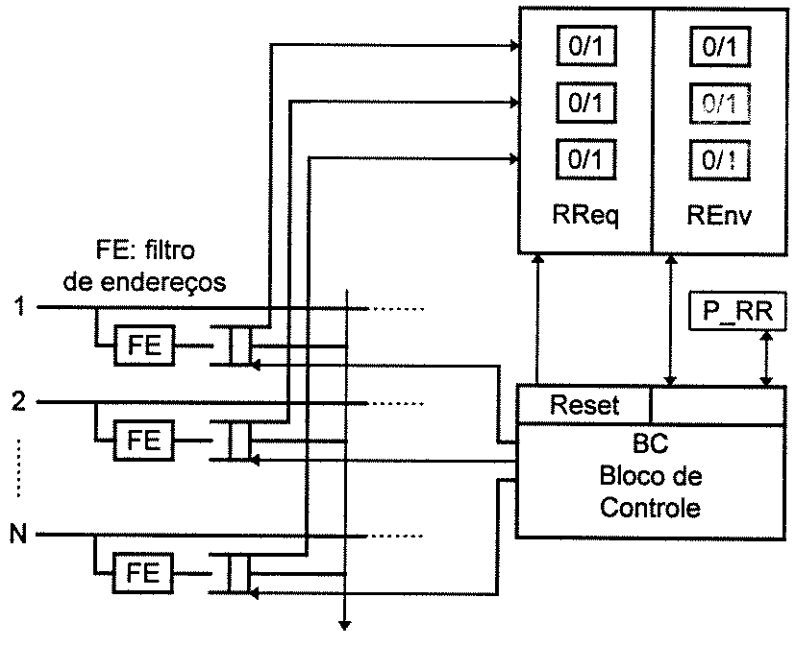


Figura 5.1 Proposta para Controle usando esquema *Round-Robin* para seleção de células.

O Filtro de Endereços (FE) realiza a operação de filtragem e mudança de endereço de cada célula, ou seja, cada célula que chega é encaminhada ao respectivo *buffer* da porta de destino. Cada *buffer* envia um pedido de transferência de células para o Registro de Requisições (RReq), que assume dois estados: 1 para indicar presença de célula no cabeçalho do *buffer* e 0 para indicar ausência. O Bloco de Controle (BC) acessa o RReq para selecionar a entrada que tiver a prioridade de envio de células. Feita a seleção, o Bloco de Controle envia um sinal em nível alto para o Registro de Envio (REnv) da porta de entrada que foi selecionada para enviar a célula para indicar a porta que foi selecionada. Finalmente é feita a operação de transferência da célula da porta de entrada para a saída correspondente. O ponteiro do círculo de *Round-Robin* é incrementado e os registros REnv e RReq são zerados para a próxima janela de tempo.

O Bloco de Controle utiliza o círculo de *Round-Robin* no esquema de seleção , cujo valor indica a posição imediatamente após a última porta de entrada que obteve a permissão para o envio de célula, devendo o esquema de seleção obedecer as seguintes etapas:

1. Cada *buffer* da coluna *n* envia um pedido de transferência de célula para a saída, procedimento este que é realizado pelo envio de um sinal em nível alto para o Registro de Requisição.

2. A saída irá escolher a porta de entrada que estiver na posição imediatamente após a última porta que obteve a permissão pelo ponteiro do círculo de *Round-Robin*. Este procedimento é realizado pela leitura da posição atual do ponteiro do circulo de *Round-Robin* (P_RR) e pela leitura do Registro de Requisição. O Bloco de Controle seleciona a porta de entrada que tiver prioridade e envia um sinal em nível alto para o Registro de Envio para indicar qual porta obteve a requisição.

3. Nesta etapa, o Bloco de Controle faz a leitura do Registro de Envio para identificar a porta que deverá enviar a célula e informa ao *buffer* de entrada para que este realize a operação necessária para a transferência da célula para a saída. A célula escolhida é encaminhada e o ponteiro do círculo de *Round-Robin* é incrementado para a posição imediatamente após a porta que obteve a permissão.

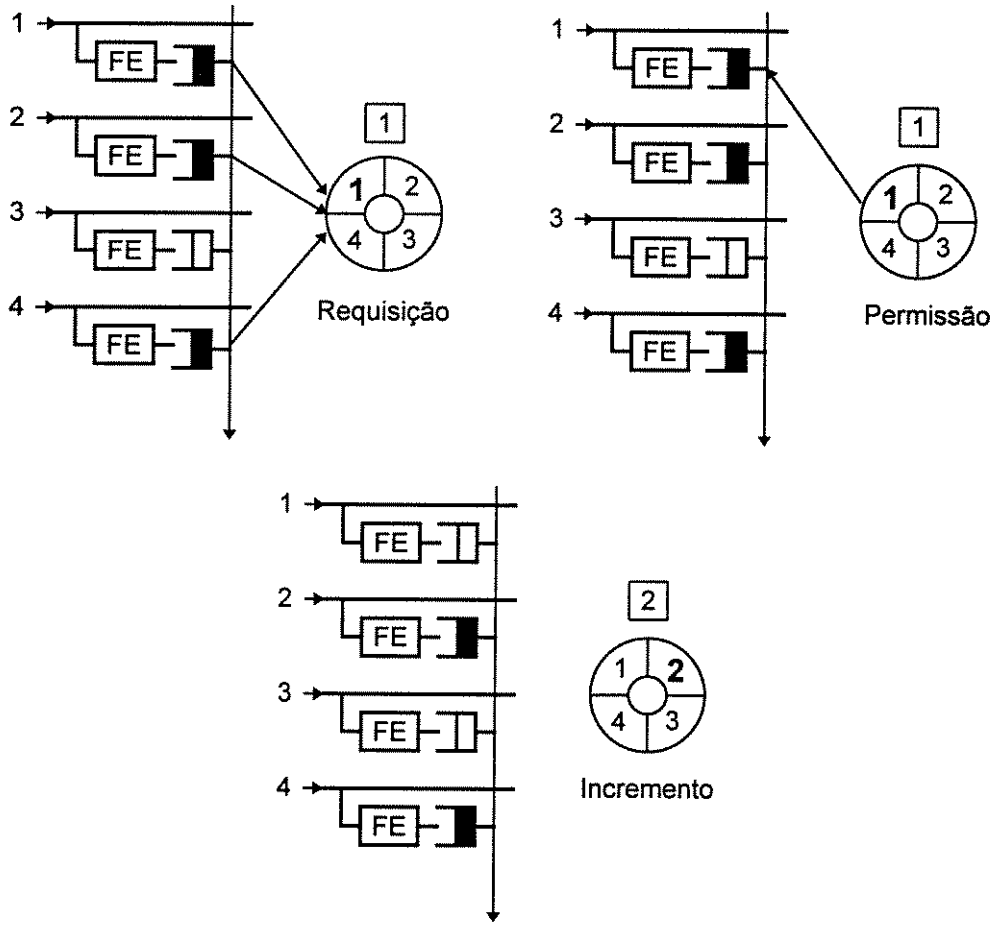


Figura 5.2 Seqüência de operações realizadas pelo controle .

Verifica-se que o algoritmo proposto não necessita de uma fase de aceitação como ocorre com o SLIP - IRRM e o PIM, pois cada célula que chega é encaminhada diretamente ao seu respectivo *buffer* de destino, eliminando a possibilidade de cada fila enviar mais de uma requisição para as portas de saída. Portanto cada *buffer* de entrada que possuir células a serem encaminhadas envia uma, e somente uma requisição para a saída e na fase seguinte a porta de saída da coluna *n* escolhe uma, e somente uma porta de entrada para o envio de células.

A Fig. 5.2 ilustra um exemplo para uma seqüência do algoritmo, assumindo que o ponteiro P_RR da coluna *n* aponta para a porta de entrada 1. As portas de entrada 1,2 e 4 possuem células para enviar para a porta de saída *n*. Cada porta de entrada envia um pedido de requisição para o bloco de controle. Como o ponteiro do círculo de *Round-Robin* está na posição 1, a porta de entrada 1 é escolhida para enviar

a célula. O ponteiro é incrementado e na próxima janela de tempo a porta de entrada 2 da coluna *n* terá a prioridade de envio de células.

5.3. Simulações

5.3.1. Fonte Binomial

As Figs. 5.3 e 5.4 mostram os resultados da simulação levando em consideração a variação do número de portas no comutador, para N=4, 8 e 16 portas. A diferença de desempenho ao se variar o número de portas não é apreciável e o comutador consegue operar satisfatoriamente com uma carga de aproximadamente 90% , considerando uma fonte binomial. A Fig. 5.5 compara os resultados da simulação com os valores teóricos, considerando um comutador com 8 portas e com atendimento de células na mesma janela de tempo e na janela de tempo seguinte. Verifica-se que o modelo de fila discreto *Geom/D/1* adotado na análise ajusta-se perfeitamente aos valores obtidos na simulação. No apêndice encontra-se o modelo analítico adotado para a análise comparativa.

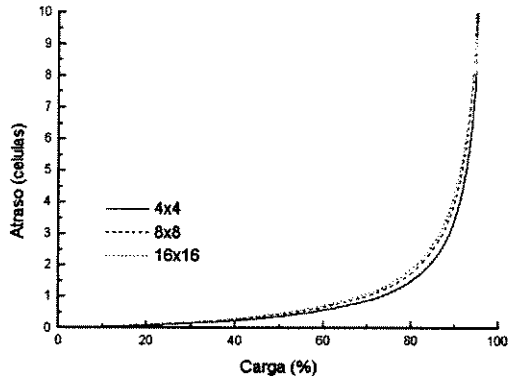


Figura 5.3 Atraso de células em função da carga para atendimento na mesma janela de tempo.

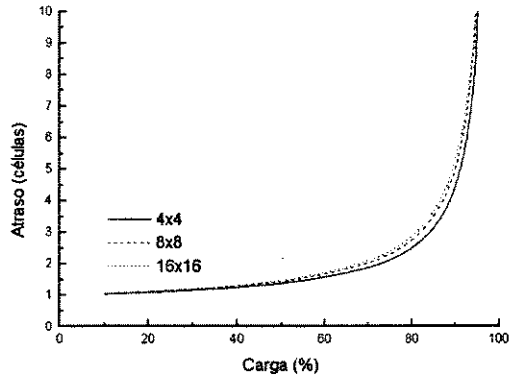


Figura 5.4 Atraso de células em função da carga para atendimento na janela de tempo seguinte.

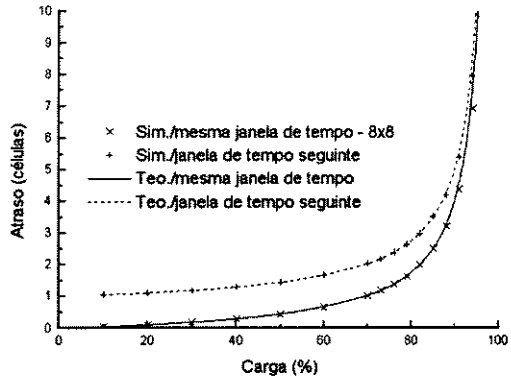


Figura 5.5 Comparação dos resultados teóricos e simulados para atraso de células com atendimento na mesma janela de tempo e janela de tempo seguinte.

5.3.2. Fonte *On-Off*

Os gráficos a seguir mostram o comportamento do comutador considerando-se o efeito do número de portas e o comprimento do surto. Os gráficos das Figs. 5.6 e 5.7 mostram o comportamento do comutador para N=4, 8 e 16 portas e surto L = 3. No caso de utilização de uma fonte *on-off*, verifica-se uma influência maior no número de portas em relação aos resultados obtidos com fonte binomial. Os gráficos das Figs. 5.8 e 5.9 mostram a influência na variação do comprimento do surto. Verifica-se uma queda de desempenho bastante acentuada quando o tamanho do surto cresce. Para uma carga de 60%, o atraso de células para uma fonte binomial é menor que uma janela de tempo e para um surto L=7, o atraso é maior que oito janelas de tempo.

Ainda, comparando a utilização de fontes binomiais e fontes *on-off*, para um surto L=1, os resultados obtidos para cargas até 50% são praticamente os mesmos, como se esperava, pois nesse caso a fonte *on-off* se aproxima da fonte binomial.

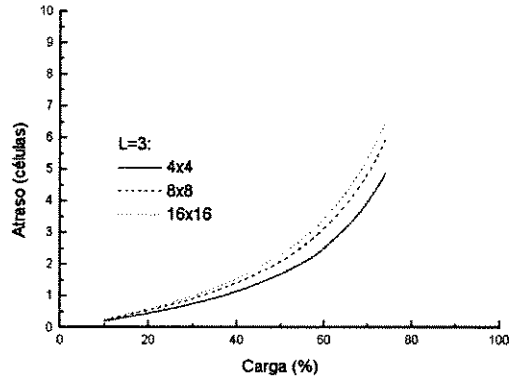


Figura 5.6 Atraso de células em função da carga para tráfego *on-off* e com atendimento na mesma janela de tempo.

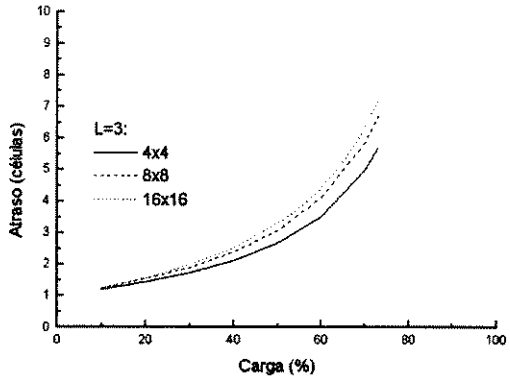


Figura 5.7 Atraso de células em função da carga para tráfego *on-off* e com atendimento na janela de tempo seguinte.

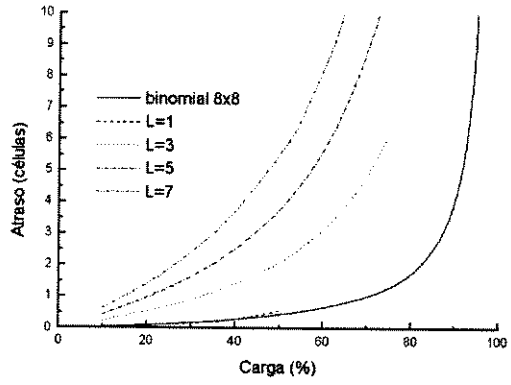


Figura 5.8 Comparação de atraso de células entre o tráfego *on-off* e binomial (atendimento na mesma janela de tempo).

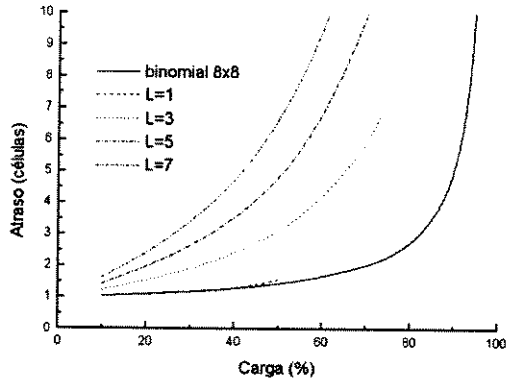


Figura 5.9 Comparação de atraso de células entre o tráfego *on-off* e binomial (atendimento na janela de tempo seguinte).

5.4. Conclusões

Foram analisadas duas situações de atendimento para um comutador do tipo matricial com algoritmo de seleção baseado em *Round-Robin*. Para o caso de fontes binomiais, a utilização de um modelo de fila discreto do tipo *Geom/D/1* ajusta-se perfeitamente aos resultados obtidos por simulação. O emprego de fontes *on-off* para a geração de células mostrou claramente o efeito do comprimento do surto sobre o atraso de propagação das células.

Embora o crescimento de *buffer* seja N^2, o comutador matricial apresenta características bastante desejáveis como simplicidade na implementação do algoritmo de encaminhamento de células e uma vazão de 100%.

Capítulo 6

6. Conclusões Gerais

Os objetivos principais desta dissertação foram estudar e propor estruturas de comutação para redes ATM.

As estruturas de comutação ATM possuem características bastante diferentes conforme a localização dos elementos de armazenamento das células. Desta maneira foram feitas as descrições das principais estruturas existentes na literatura, enfatizando as estruturas com *buffer* na entrada devido à sua relevância atual.

Foram feitos estudos através de simulações para comparar os desempenhos de algoritmos iterativos PIM, SPIM, SLIP-IRRM e IRRM-MC, quando submetidos a tráfego em surtos e também comparar com os resultados obtidos em [11] para tráfego binomial. Os estudos mostraram que com o tráfego em surtos os algoritmos apresentaram uma sensível queda de desempenho, notadamente em relação ao atraso de células. As vazões de todos os algoritmos mostraram ser menos suscetíveis.

Foi apresentada também uma proposta de comutador com *buffers* na entrada em que o atendimento das células é feito ponderando as faixas de serviços. Este algoritmo permite um melhor controle dos atrasos, fazendo com que os atrasos de células de serviços de faixa larga e faixa estreita fossem aproximadamente iguais. Os estudos indicam que controlando as ponderações pode-se obter atrasos diferenciados e controlados para cada faixa de serviço.

Foi proposta também uma estrutura de comutador matricial com um esquema de encaminhamento de células bastante simples. Embora esta estrutura tenha um fator N^2 de crescimento dos *buffers*, mostrou-se extremamente simples na sua implementação.

Como proposta para trabalhos futuros, pode-se efetuar a análise do algoritmo proposto no capítulo 4 para uma variação dinâmica da carga, com os pesos sendo

calculados em função das faixas de serviço ao longo de um período de simulação. Um outra proposta é estudar o modelo matemático para o algoritmo IRRM-MC.

Apêndice

A. Modelo Analítico para Comutador Matricial

A.1. Análise do Atraso de Células com Atendimento na Janela de Tempo Seguinte

A análise foi feita considerando atendimento de células na janela de tempo seguinte. A Fig. A.1 mostra o esquema de um comutador matricial, onde cada célula que chega é diretamente armazenada no *buffer* da respectiva porta de destino. Este procedimento é feito por meio de um filtro de endereços que compara o endereço do cabeçalho da célula com o endereço de cada *buffer*. Em cada *buffer* do comutador, as células são servidas de acordo com o esquema FIFO (*first-in-first-out*). O tipo de armazenamento das células nos *buffers* permite aproximar o modelo de análise basicamente como uma arquitetura de comutador com *buffer* na saída.

No esquema da Fig. A.1, considera-se que as células chegam a cada entrada com uma probabilidade igual a p em cada janela de tempo, obedecendo a um processo independente e idêntico de Bernoulli, e que cada célula tem uma probabilidade $1/N$ de ser enviada para qualquer saída. Além disso cada nova célula que chega é endereçada de maneira independente para qualquer saída. Considerando uma porta de saída específica, pode-se notar que o número de células que chegam em um dado instante de tempo possui uma distribuição uniforme.

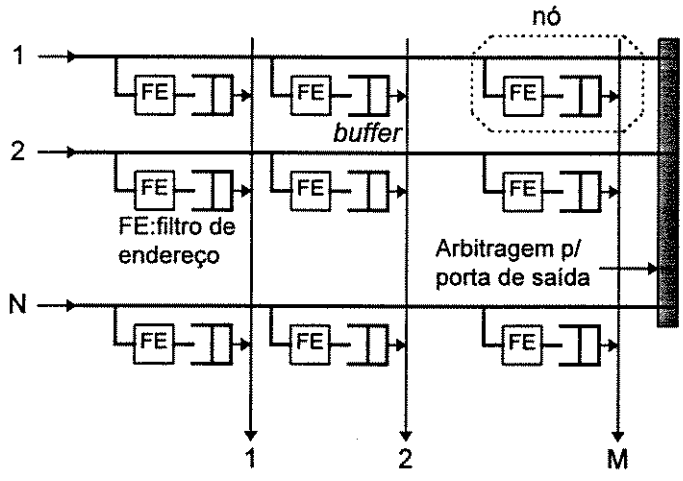


Figura A.1 Comutador matricial.

Considere inicialmente a probabilidade p/N de chegada de uma célula em um *buffer* na saída em uma janela de tempo. Então

$$\Pr(1 \text{ chegada}) = p/N \qquad \textbf{Eq. A.1}$$

$$\Pr(0 \text{ chegadas}) = 1 - p/N \qquad \textbf{Eq. A.2}$$

Define-se a transformada Z como:

$$G_X(z) = \sum_{k=0}^{\infty} z^k p_k = E\{z^X\} \qquad \textbf{Eq. A.3}$$

Aplicando a definição da transformada Z, obtemos

$$G_X(z) = \sum_{k=0}^{\infty} z^k p_k = (1 - p/N) + z\,p/N \qquad \textbf{Eq. A.4}$$

A transformada Z da soma de variáveis aleatórias independentes é o produto das transformadas Z. Considerando X_1, X_2 ,..., X_n variáveis aleatórias de Bernoulli independentes e com a mesma distribuição de probabilidade

$$\Pr(X_i = 1) = p\,,\ \Pr(X_i = 0) = 1 - p = q \qquad i = 1,2,...,n \qquad \textbf{Eq. A.5}$$

Definindo $Y = X_1 + X_2 + \cdots + X_n$, a probabilidade de k chegadas em n intervalos de tempo é dado por

$$G_Y(z) = G_{X_1}(z)G_{X_2}(z)\cdots G_{X_n}(z) = \left(1 - p/N + z\,p/N\right)^n \qquad \textbf{Eq. A.6}$$

Expandindo o binômio de Newton, tem-se

$$\left(1 - p/N + z\,p/N\right)^n = \sum_{k=0}^{n}\binom{n}{k}\left(z\,p/N\right)^k\left(1 - p/N\right)^{n-k} \qquad \textbf{Eq. A.7}$$

$$G_Y(z) = \sum_{k=0}^{n} z^k\left[\frac{n!}{k!(n-k)!}\left(p/N\right)^k\left(1 - p/N\right)^{(n-k)}\right] \qquad \textbf{Eq. A.8}$$

Portanto para uma Variável Aleatória Binomial, tem-se

$$\Pr(Y = k) = \frac{n!}{k!(n-k)!}\,p/N^k\,(1 - p/N)^{(n-k)} \qquad \textbf{Eq. A.9}$$

A função geradora de probabilidades correspondente é dada por

$$Y(z) \equiv \sum_{k=0}^{N} z^k\,\Pr[Y = k] = \left(1 - \frac{p}{N} + z\frac{p}{N}\right)^N \qquad \textbf{Eq. A.10}$$

Quando $N \to \infty$, a distribuição de Y se aproxima de uma distribuição de Poisson, e então

$$a_k \equiv \Pr[A = k] = \frac{e^{-p} p^k}{k!}, \qquad k = 0,1,2,... \qquad \textbf{Eq. A.11}$$

A função geradora de probabilidades correspondente é dada por

$$A(z) \equiv \sum_{k=0}^{N} z^k \Pr[A = k] = e^{-p(1-z)} \qquad \textbf{Eq. A.12}$$

Seja n_i o número de células na fila j no final da i-ésima janela de tempo, K_i o número de chegadas de células durante a i-ésima janela de tempo e S_i o número de células que partiram durante a i-ésima janela de tempo. Então

$$n_{i+1} = n_i - S_i + K_i \qquad \textbf{Eq. A.13}$$

Pode ocorrer somente uma partida por janela de tempo e portanto o termo S_i pode assumir dois valores: $S_i = 1$, se $n_i > 0$ e $S_i = 0$, se $n_i = 0$.

Tirando a média de ambos os termos na Eq. A.13, obtemos

$$E\{n_{i+1}\} = E\{n_i\} - E\{S_i\} + E\{K_i\} \qquad \textbf{Eq. A.14}$$

No equilíbrio estatístico,

$$E\{n\} = E\{n\} - E\{S\} + E\{K\} \qquad \textbf{Eq. A.15}$$

$$E\{S\} = E\{K\} \qquad \textbf{Eq. A.16}$$

Aplicando a transformada Z em n_{i+1}, obtemos

$$G_{n_{i+1}}(z) = G_{k_i}(z) \cdot G_{n_i - S_i}(z) \qquad \textbf{Eq. A.17}$$

No equilíbrio estatístico

$$G_n(z) = G_k(z) \cdot G_{n-S}(z) \qquad \textbf{Eq. A.18}$$

$$G_{n-S}(z) = \sum_{n=0}^{\infty} z^{n-S} P_n \qquad \textbf{Eq. A.19}$$

Se $n = 0$, então $S = 0$ e

$$G_{n-S}(z) = \sum_{n=0}^{\infty} z^{n-S} P_n$$

$$G_{n-S}(z) = p_0 + \sum_{n=1}^{\infty} z^{n-1} P_n$$

$$G_{n-S}(z) = p_0 + \frac{1}{z}\left(G_n(z) - p_0\right) \qquad \textbf{Eq. A.20}$$

Substituindo a Eq. A.20 na Eq. A.18 obtemos

$$G_n(z) = p_0 \frac{(z-1) \cdot G_K(z)}{z - G_K(z)} \qquad \textbf{Eq. A.21}$$

Pela propriedade da transformada Z, sabe-se que:

$$\frac{d}{dz} G_X(z) = \sum_{k=1}^{\infty} k z^{(k-1)} p_k = \sum_{k=0}^{\infty} k z^{(k-1)} p_k$$

$$\frac{d}{dz}G_X(z)\Big|_{z=1} = \sum_{k=0}^{\infty} kp_k = E\{X\} \qquad \textbf{Eq. A.22}$$

Ou seja, diferenciando a Eq. A.21 em relação a z e tomando o limite de z→1, obtém-se o número médio da fila:

$$G_n'(z) = p_0 \frac{\left(z - G_K(z)\right)\cdot\left[(z-1)\cdot G_K'(z) + G_K(z)\right] - (z-1)\cdot G_K(z)\cdot\left(1 - G_K'(z)\right)}{\left(z - G_K(z)\right)^2}$$

$$G_n'(z) = p_0 \frac{z(z-1)G_K'(z) + G_K(z) - G_K(z)^2}{\left(z - G_K(z)\right)^2} \qquad \textbf{Eq. A.23}$$

Para $z = 1$, obtemos:

$$G_n'(z) = E\{n\} = \frac{G_K(1) - G_k(1)^2}{\left(1 - G_K(1)\right)^2} \qquad \textbf{Eq. A.24}$$

Mas G_K (1) = 1 e portanto a Eq. A.24 fica indeterminado. Aplicando a Regra de L'Hôpital na Eq. A.23, obtemos

$$E\{n\} = p_0 \frac{\left(z^2 - z\right)\cdot G_K''(z) + G_K'(z)\cdot(2z-1) + G_K'(z) - 2G_K(z)G_K'(z)}{2\cdot\left(z - G_K(z)\right)\cdot\left(1 - G_K'(z)\right)}$$

Eq. A.25

Para z =1, obtemos:

$$E\{n\} = p_0 \frac{G_K'(1) + G_K'(1) - 2G_K(1)G_K'(1)}{2\cdot\left(1 - G_K(1)\right)\cdot\left(1 - G_K'(1)\right)} \qquad \textbf{Eq. A.26}$$

Novamente obtemos uma indeterminação. Aplicando L'Hôpital na Eq. A.25, obtemos

$$E\{n\} = p_0 \frac{(z^2 - z)G_K'''(z) + G_K''(z)(2z-1) + 2G_K'(z) + (2z-1)G_K''(z) + G_K''(z)}{2 \cdot \left[\left(1 - G_K'(z)\right)^2 - G_K''(z) \cdot \left(z - G_K(z)\right)\right]} - \frac{2\left(G_K(z)G_K''(z) + G_K'(z)^2\right)}{2 \cdot \left[\left(1 - G_K'(z)\right)^2 - G_K''(z) \cdot \left(z - G_K(z)\right)\right]}$$

Eq. A.27

Para $z = 1$, obtemos:

$$E\{n\} = p_0 \frac{G_K'(1)}{1 - G_K'(1)} + p_0 \frac{G_K''(1)}{2 \cdot \left[1 - G_K'(1)\right]^2}$$

Eq. A.28

Além disso, temos:

$$G_K'(1) = E\{K\} = p$$

Eq. A.29

$$G_K''(1) = E\{K^2\} - E\{K\}$$

Eq. A.30

$$E\{K^2\} = \sigma_K^2 + E^2\{K\} = p\left(1 - \frac{p}{N}\right) + p^2$$

Eq. A.31

$$G_K''(1) = p^2\left(1 - \frac{1}{N}\right)$$

Eq. A.32

Além disso,

$$E\{S\} = 0 \cdot p_0 + \sum_{i=1}^{\infty} 1 \cdot p(n_i) = 1 - p_0 = E\{K\} = p$$

Eq. A.33

Portanto:

$$p_0 = 1 - p \qquad \textbf{Eq. A.34}$$

$$\boxed{E\{n\} = p + \frac{p^2\left(1 - \frac{1}{N}\right)}{2(1-p)}} \qquad \textbf{Eq. A.35}$$

Para $N = 1$, temos $E\{n\} = p$.

Há no máximo uma chegada de célula por janela tempo, e será sempre atendida na janela de tempo seguinte.

Utilizando a Lei de Little,

$$E\{T\} = \frac{E\{n\}}{\lambda} \qquad \textbf{Eq. A.36}$$

onde λ é a taxa de chegada de células.

$$\lambda = \frac{p}{T_{slot}} \qquad \textbf{Eq. A.37}$$

onde T_{slot} é igual a uma janela de tempo. Portanto

$$\boxed{E\{T\} = T_{slot}\left[1 + \frac{p(1 - 1/N)}{2(1-p)}\right]} \qquad \textbf{Eq. A.38}$$

A.2. Análise do Atraso de Células com Atendimento na Mesma Janela de Tempo

A análise para atendimento de células na janela de tempo seguinte considera que o atraso total experimentado por uma célula é a soma de uma janela devido ao

atendimento na janela de tempo seguinte e o tempo de espera da célula na fila. Portanto para o caso de atendimento na mesma janela de tempo basta subtrair da Eq. A.38 o tempo equivalente a uma janela de tempo.

$$E\{T\} = T_{slot}\left[\frac{p(1-1/N)}{2(1-p)}\right] \qquad \textbf{Eq. A.39}$$

Referências Bibliográficas

[1] ANDERSON, T. E.; OWICKI, S. S.; SAXE, J. B. and THACKER, C. P. High Speed Switch Scheduling for Local Networks. **ACM Transactions on Computer Systems,** v.11, n.4, p. 319-352, November 1993.

[2] ANDERSON, T. E.; MCKEOWN, N. A quantitative comparison of iterative scheduling algorithms for input-queued switches. http://pocari.stanford.edu/telecom/faculty/mckeown.html

[3] DEL RE, E. and FANTACCI, R. Performance Evaluation of Input and Output Queueing Techniques in ATM Switching Systems. **IEEE Transactions on Communications**, v. 41, n. 10, p. 1565-1575, October 1993.

[4] DEPRICKER, M. ASYNCHRONOUS TRANSFER MODE: Solution for Broadband ISDN, ed. Ellis Horwood, p. 181-188, 1995.

[5] DOI, Y. and YAMANAKA, N. A High-Speed ATM Switch with Input and Cross-Point Buffers. **IEICE Trans. Commun.**, v. E76-B, n. 3, p. 310-314, March 1993.

[6] GENDA, K.; YAMANAKA, N. and DOI, Y. A High-Speed ATM Switch that Uses a Simple Retry Algorithm and Small Input Buffers. **IEICE Trans. Commun.**, v. E76-B, n. 7, p. 726-730, July 1993.

[7] KAROL, M.; HLUCHYJ, M. and Morgan, S. Input Versus Output Queueing on a Space-Division Packet Switch. **IEEE Transactions on Communications**, v. 35, n. 12, p. 1347-1356, December 1987.

[8] KOLIAS, C. and KLEINROCK, L. Throughput analysis of multiple input-queueing ATM switches. **IFIP-IEEE, Broadband Communications,** Eds. L Mason and A Casaca, pp 382-393, Chapman and Hall, 1996.

[9] LAMAIRE, R. O. and SERPANOS, D. N. Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues. **IEEE/ACM Transactions on Networking,** v.2, n. 5, p. 471-482, October 1994.

[10] LIEW, S. C. Performance of Various Input-buffered and Output-buffered ATM Switch Design Principles under Bursty Traffic: Simulation Study. **IEEE Transactions on Communications**, v. 42, n. 2/3/4, p. 1371-1379, Feb./Mar./April 1994.

[11] MAVIGNO, M. C. **Algoritmos de encaminhamento de células em comutadores ATM com buffer na entrada**. Campinas, SP, 1997. Dissertação (Mestrado) - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

[12] MCKEOWN, N.; VARAIYA, P. and WALRAND, J. Scheduling Cells in an Input-Queued Switch. **Electronics Letters,** v. 29, n. 25, p. 2174-2175, 9th December 1993.

[13] MCKEOWN, N.; ANANTHARAM, V. and WALRAND, J. Achieving 100% throughput in an input-queued switch. **Proc. IEEE Infocom**, San Francisco, p. 3A.4.1-3A.4.6, 1996.

[14] MOTOYAMA, S.; PETR, D. W. and FROST, V. S. Input-queued switch based on a scheduling algorithm. **Electronics Letters,** v. 31, n. 14, p. 1127-1128, 6th July 1995.

[15] MOTOYAMA, S. e MAVIGNO, M. C. Um novo algoritmo de encaminhamento de células em um comutador ATM com buffer na entrada (Input Buffering) e com atendimento prioritário de classes de serviço. **XV Simpósio Brasileiro de Telecomunicações**, p. 141-144, setembro 1997.

[16] ROBERTAZZI, T. G. Computer Networks and Systems:queueing theory and performance evaluation. Second Edition 1994, Springer-Verlag.

[17] SUH, J-J. and JUN, C-H. Performance analysis of the knockout switch with input buffers. **IEE Proc. Commun.**, v. 141, n. 3, p. 183-189, June 1994.

[18] THOMAS, G. Bifurcated Queueing for Throughput Enhancement in Input-Queued Switches. **IEEE Communications Letters**, v. 1, n. 2, March 1997.

Publicação

[19] ONO, L. M., MAVIGNO, M. C. , MOTOYAMA, S. Performance Analysis of Iterative Scheduling Algorithms for ATM Input-Queued Switches. **SBT/IEEE/International Telecommunications Symposium - ITS'98**, São Paulo, v. 1, p. 195-200, August 1998

Artigos Submetidos

[20] ONO, L. M., MAVIGNO, M. C., MOTOYAMA, S. An Iterative Cell Scheduling Algorithm for ATM Input-Queued Switch with Service Class Priority. **IEEE Communications Letters.**

[21] ONO, L. M., MAVIGNO, M. C., MOTOYAMA, S. Performance Analysis of Iterative Scheduling Algorithms for ATM Input-queued Switches. **Automatica.**

Lista de Abreviaturas

ABR - Available Bit Rate

ATM - Asynchronous Transfer Mode

CBR - Constant Bit Rate

FIFO - First-in-First-Out

FIRO - First-in-Random-Out

HOLB - Head of Line Blocking

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

ITU - International Telecommunications Union

nrt-VBR - non-real time Variable Bit Rate

OSI - Open Systems Interconnection

PIM - Parallel Iterative Matching

RDSI - Rede Digital de Serviços Integrados

rt-VBR - real time Variable Bit Rate

SLIP-IRRM - Iterative Round-Robin Matching with SLIP

SPIM - Simplified Parallel Iterative Matching

UBR - Unexpected Bit Rate