Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

MODELAMENTO, SIMULAÇÃO E ANÁLISE DE DESEMPENHO DE UM COMUTADOR ATM CIOQ

SEBASTIÃO RODRIGUES DE A. FILHO

JULHO / 2005

Inatel
Instituto Nacional de Telecomunicações

MODELAMENTO, SIMULAÇÃO E ANÁLISE DE DESEMPENHO DE UM COMUTADOR ATM CIOQ

DISSERTAÇÃO SUBMETIDA AO INSTITUTO NACIONAL DE TELECOMUNICAÇÕES - INATEL - COMO PARTE DOS PRÉ-REQUISITOS NESCESSÁRIOS PARA A OBTENÇÃO DO TÍTULO DE

MESTRE EM REDES DE TELECOMUNICAÇÕES

POR

SEBASTIÃO RODRIGUES DE AGUIAR FILHO
Engenheiro Eletricista pelo Inatel em Dez/1994.

ORIENTADOR

PROF. DR. ANTÔNIO MARCOS ALBERTI – UNICAMP

CO-ORIENTADOR

PROF. DR. ANILTON SALLES GARCIA – PPGEE/UFES

BANCA EXAMINADORA

PROF. DR. JOSÉ MARCOS CÂMARA BRITO – INATEL
PROF. DR. SHUSABURO MOTOYAMA – UNICAMP

Santa Rita do Sapucaí, 11 de Julho de 2005.

Agradecimentos

Primeiramente à DEUS por ter me concedido tamanha oportunidade.
Aos meus pais, Sebastião e Ione, pelo apoio, amor, carinho e incentivo.
Aos meus irmãos, pela amizade e compreensão.
Ao meu orientador, Antônio Marcos Alberti, pela amizade e dedicação.
Ao meu co-orientador, Anilton Salles Garcia, pela amizade, incentivo, sabedoria e ajuda nos momentos difíceis.
A todos os meus professores que foram a base de conhecimento para este trabalho.
A todos os amigos de Montes Claros pela força e valorização desta realização pessoal.
Aos amigos e colegas de mestrado que ajudaram na luta do dia a dia com descontração, amizade e boa vontade.
Ao amigo Rodrigo Barbosa pela incomensurável ajuda nas simulações à distância.
A Rosanna Mara e o William Hisatugu pelo suporte no software Arena.
A todos que de alguma forma contribuíram para a realização deste trabalho.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

RESUMO

Neste trabalho foi realizado o modelamento, a simulação e a análise de desempenho de um comutador ATM (*Asynchronous Transfer Mode*) CIOQ (*Combined Input-Output Queuing*). A estrutura CIOQ visa diminuir o HOLB (*Head of Line Blocking*), visto que até *m* células ATM podem ser transferidas simultaneamente a cada saída em cada *slot* de tempo. Desta forma, foram utilizados *buffers* tipo FIFO (*First In, First Out*) simples em cada entrada e *m buffers* em cada saída. Como esta estrutura utiliza *m links* físicos internos, evita-se assim a necessidade de altas taxas de transferências internamente ao comutador. Os modelos de comutadores implementados possuem *m links* internos destinados à transferência de células. Dois tipos de modelos foram desenvolvidos: um em que todas as entradas do comutador possuem a mesma prioridade de atendimento, e outro em que existem cinco classes de prioridade diferentes em cada entrada, visando assim, prover QoS (*Quality of Service*). No primeiro caso, para filas de entrada do tipo FIFO, foram medidos a probabilidade de bloqueio e o comprimento médio da fila na entrada e, no segundo caso, com filas com prioridades, foi medido o comprimento médio da fila, tanto na entrada quanto na saída, para cada classe de prioridade.

Para modelar os comutadores e realizar as simulações foi utilizado o *software* de simulação de eventos discretos ARENA 5.0 (*Rockell Software*). Um dos objetivos deste trabalho é o de comparar/comprovar os resultados analíticos e também resultados simulados obtidos utilizando-se o *software* Matlab em um trabalho anterior, com os resultados obtidos aqui.

O resultado final foi muito satisfatório, ou seja, os resultados obtidos com este *software, ARENA 5.0,* e na referência utilizada foram bastante próximos. Comprovou-se também que esta estrutura de comutador mostrou-se bastante eficiente, apresentando baixas probabilidades de bloqueio e também baixos comprimentos médio de filas para os parâmetros simulados.

Palavras chave: ATM, Comutação ATM, Comutação de pacotes, CIOQ.

CAPÍTULO 1

1 - INTRODUÇÃO

1.1. MOTIVAÇÃO

De um modo geral, as redes de telecomunicações são caracterizadas pela especialização, ou seja, cada tipo de serviço possui sua rede de transporte própria. Via de regra, estas redes são bastante específicas: tem-se, por exemplo, a rede pública ou privada de telefonia para transporte de voz; as redes baseadas nos protocolos Ethernet para transporte de dados; e também as redes de TV que podem utilizar *broadcast* via ondas de rádio, transmissão via redes de cabos coaxiais ou, até mesmo, via satélite. Cada uma destas redes foi projetada para o transporte específico de apenas um serviço e normalmente o suporte a outros serviços é inflexível e deixa a desejar. Entretanto, existem casos em que uma rede projetada para um serviço é usada para transportar outro serviço. Como exemplo, pode-se citar a rede de telefonia fixa, que é frequentemente utilizada para o transporte de dados, desde que sejam utilizados recursos externos, como *modems*, nas pontas da rede. Todavia, devido às limitações da rede física, as taxas disponíveis não são muito altas. Além disto, a comutação de circuitos, utilizada na telefonia, não permite a otimização do uso de largura de banda na rede. Assim, tem-se uma variedade de redes diferentes para suportar cada uma destas diversas mídias, pois cada uma delas possui diferentes características.

Apenas como exemplo, seguem alguns tipos de mídias e suas respectivas características:

- **Texto:** Tráfego em rajadas. Exceto para aplicações em tempo real, o retardo máximo e a variação estatística não representam grandes preocupações. pequena tolerância a erros;
- **Imagem Gráfica:** Tráfego em rajadas. Atraso máximo e variação estatística do atraso não constituem um problema. Pouca tolerância a erros. No caso de imagens compactadas ou comprimidas, a tolerância é menor ainda.
- **Áudio amostrado à taxa constante:** Tráfego contínuo com taxas constantes. Retardo e a variação estatística do retardo são uma preocupação. No caso de conversação interativa e em tempo real as exigências tornam-se ainda mais severas. Taxa de erro não é uma preocupação crítica;

- **Vídeo sem compactação/compressão:** Tráfego contínuo com taxas constantes. Como no áudio, o retardo e a variação estatística deste são uma preocupação, sendo que os requisitos têm que ser os mesmos para ambos, áudio e vídeo. Taxa de erro não é uma preocupação crítica.

Pelo fato das redes suportarem serviços específicos, as mesmas apresentam diversas desvantagens, tais como:

- **Dependência de serviço:** Cada rede é capaz de transportar de forma ótima apenas o serviço para a qual foi originalmente projetada. Somente com o uso de mecanismos de adaptação a rede pode transportar outros serviços;
- **Inflexibilidade:** Uma rede especializada tem grande dificuldade de se adaptar aos novos requisitos, surgidos com os rápidos avanços tecnológicos;
- **Ineficiência:** Os recursos disponíveis são usados de forma ineficiente. Recursos disponíveis em uma rede podem não ser úteis para outra rede. Na maioria das vezes, é necessária uma adaptação.

Todas estas considerações fazem crer que se torna necessário a existência de uma rede única, flexível, que seja capaz de transportar todos os tipos de serviços, compartilhando todos os seus recursos entre os serviços transportados. Esta rede, além de não sofrer das desvantagens citadas anteriormente, ainda possuirá as seguintes vantagens:

- **Flexibilidade:** A rede será capaz de transportar todos os tipos de serviço e de se adaptar às suas mudanças e novas necessidades.
- **Eficiência:** Todos os recursos da rede poderão ser compartilhados, utilizando-se multiplexação estatística.
- **Custo:** Haverá a necessidade de se projetar, montar e realizar manutenção em apenas uma rede. Com isto, o custo médio, tende a ser menor.

Além dos serviços citados, existe uma crescente demanda dos chamados tele-serviços, que demandarão diferentes, e algumas vezes desconhecidos, requisitos de qualidade. Como exemplo, pode-se citar: vídeo sob demanda, vídeo conferência, transmissão de dados de alta velocidade, videofone e até mesmo HDTV (*High Definition Television*). Todos estes serviços são tipicamente chamados de aplicações em banda larga. Embora não haja um consenso para o tema, banda larga seriam os sistemas com taxas de

transmissão superiores às do acesso primário da RDSI-FE (Rede Digital de Serviços Integrados – Faixa Estreita). Seguem alguns exemplos de aplicações em banda larga:

- **Serviços Conversacionais:** Transferência de mídia em tempo real (videoconferências);
- **Serviços de Recuperação:** Recuperação de dados remotos (livrarias eletrônicas);
- **Serviços de Mensagens:** Transferência de dados sem ser em tempo real (*mail box* em alguns casos);
- **Serviços de Distribuição:** Distribuição de dados com ou sem controle do usuário (jornais, revistas, filmes).

Todos estes serviços possuem diferentes características e requisitos de qualidade. As aplicações em banda larga se caracterizam, principalmente, pela grande quantidade de dados, pela necessidade de altas taxas de transmissão e pela necessidade de acesso sincronizado aos dados.

O surgimento de novos serviços de telecomunicações demandou por uma evolução das redes de telecomunicações. Novas técnicas, baseadas em "modos de transferência" foram desenvolvidas, trazendo vantagens em relação às técnicas anteriormente utilizadas. Entende-se por modo de transferência, a técnica empregada em uma rede de telecomunicações para cobrir os aspectos relacionados à transmissão, multiplexação e chaveamento.

Esta evolução levou ao desenvolvimento da tecnologia ATM (*Asynchronous Transfer Mode*). A rede ATM se propõe a atender a diversidade de serviços requerida pelos usuários, ou seja, se propõe a integrar todas as redes existentes em apenas uma. Portanto, pode-se considerá-la, pelo menos na teoria, como um ponto de convergência para as redes atuais. Claro está, que isto não é tarefa fácil, pois esta rede deve ser capaz de transportar e atender requisitos conflitantes de perda, atraso e variação de atraso.

Alguns motivos propiciaram o desenvolvimento da rede ATM, tais como a evolução tecnológica em semicondutores, transmissão digital e transmissão por fibras ópticas que permitem a transmissão de dados em altíssimas taxas, de modo bastante confiável, bem como, uma evolução na arquitetura das redes. A utilização das fibras ópticas e a mudança sistêmica nos projetos de redes foram fatores essenciais para o desenvolvimento das redes de alta velocidade, pois, desta forma, pôde-se reduzir o processamento em cada nó da rede. O controle de erros, por exemplo, que em algumas redes é feito ponto a ponto, na rede

ATM é feito fim a fim, exceto para a verificação e controle de erros nos dados do cabeçalho da célula, que são realizados no interior da rede. Desta forma, as funções do cabeçalho da célula ATM foram simplificadas, reduzindo-se assim o tamanho do mesmo. A principal função do cabeçalho é a de identificação da conexão virtual, para garantir o encaminhamento correto da célula nos diversos nós da rede. Esta redução no processamento, bem como o tamanho reduzido da célula, são os principais fatores que propiciam uma comutação mais rápida das células no interior da rede.

Pode-se considerar também, como elementos motivadores do desenvolvimento do ATM, a emergente demanda por serviços faixa larga, a disponibilidade de tecnologias de alta velocidade de transmissão, chaveamento e processamento de sinais, a capacidade de melhor processamento de imagens e dados, a necessidade de integrar serviços interativos e distributivos e a necessidade de integrar o modo de transferência de redes de circuitos e de pacotes em uma rede faixa larga universal.

O ATM foi a tecnologia escolhida pelo ITU-T *(International Telecommunications Union – Telecommunication Standardization Sector*) para ser um dos modos de transferência da RDSI-FL (Rede Digital de Serviços Integrada – Faixa Larga).

Como as redes estão se tornando cada vez mais velozes, devido principalmente aos avanços nas tecnologias de transmissão, tornou-se imprescindível que os comutadores da rede também evoluíssem para que os mesmos não se tornassem o gargalo do sistema, ou seja, os comutadores não devem ser os limitadores da velocidade do sistema.

Em se tratando de redes ATM, existem hoje diversas propostas de estruturas de comutadores no cenário mundial, visando exatamente uma melhora na qualidade e eficiência destes comutadores. Dentre estes estudos, destaca-se uma estrutura que é denominada CIOQ (*Combined Input-Output Queuing*). Como o próprio nome já diz, esta estrutura é um comutador com uma combinação de *buffers* na entrada e na saída.

Uma estrutura do tipo CIOQ é apresentada em [4]. Mas a estrutura, conforme foi proposta, possui a indesejável característica de trabalhar com altas taxas internas ao comutador. Existe também uma outra estrutura, proposta em [5] que é uma combinação do CIOQ com o VOQ (*Virtual Output Queuing).* Ambos serão detalhados no decorrer desta dissertação.

A estrutura CIOQ também pode ser utilizada em redes de pacotes de tamanho variável (como TCP/IP), bastando segmentar os pacotes em pequenos fragmentos de tamanho fixo [22]. De acordo com a referência [23], muitos dos atuais roteadores utilizam a estrutura CIOQ. Portanto, as contribuições deste trabalho não se aplicam somente a redes ATM.

1.2. OBJETIVOS DO TRABALHO

Em [1] é apresentada uma estrutura do tipo CIOQ, que utiliza *m links* internos entre todas as entradas e cada uma das saídas. Desta forma diminui-se o HOLB *(Head of Line Blocking*). Como existem *m links* para cada saída, esta estrutura utiliza uma combinação de *buffers* simples em cada entrada e *m buffers* em cada saída. Em [1] foram apresentadas equações analíticas que descrevem o comportamento deste comutador em relação ao comprimento médio das filas e probabilidade de bloqueio para os parâmetros carga, número de *links* internos e número de entradas-saídas. Também foram apresentados resultados de simulações obtidos usando o *software* MATLAB. Duas configurações diferentes foram propostas: na primeira, as filas de entrada obedeciam a uma disciplina do tipo FIFO (*First In, First Out*) e, na segunda, as filas tinham uma disciplina de prioridade com 5 classes de serviço diferentes.

Este trabalho tem por objetivo modelar e simular uma arquitetura de comutação do tipo CIOQ, bem como analisar o desempenho e validar o modelo de simulação desenvolvido. O trabalho é baseado na mesma arquitetura de comutação desenvolvida em [1] e [2]. Assim, este trabalho reproduz as simulações realizadas em [1] e [2], utilizando-se, desta vez, uma ferramenta completamente diferente, ou seja, neste caso foi utilizado um *software* de simulação de eventos discretos, o ARENA. Um dos motivos de se utilizar o software ARENA deve-se ao fato de o mesmo possuir uma interface amigável com o usuário, bem como o fato de, por se tratar de um ambiente específico de simulação, à princípio, produz resultados mais confiáveis.

Foram efetuadas diversas simulações variando-se os seguintes parâmetros do comutador: carga, número de *links* internos e número de entradas-saídas. Estes resultados foram apresentados em gráficos similares aos encontrados em [1] e [2]. O objetivo é

comparar e validar os resultados obtidos aqui com os obtidos em [1] e [2], visto que foram utilizadas ferramentas completamente diferentes.

Conforme esperado, será mostrado que os resultados obtidos estão bem próximos dos encontrados em [1] e [2]. Desta forma, acredita-se que este trabalho terá sido mais uma contribuição no estudo desta estrutura de comutação.

1.3. ORGANIZAÇÃO DA DISSERTAÇÃO

O capítulo 2 apresenta uma introdução sobre comutadores ATM, alguns requisitos dos mesmos, uma visão geral sobre as arquiteturas de comutação encontradas na literatura em relação ao posicionamento/utilização dos *buffers*. São citadas algumas vantagens e desvantagens destas arquiteturas e é explicado o problema do HOLB. Também é apresentado um breve histórico da evolução do modo de transferência assíncrono. O capítulo apresenta ainda a estrutura do comutador CIOQ (*Combined Input-Output Queuing*), assim como o seu funcionamento básico. Por fim, é apresentado e explicado o funcionamento de uma estrutura CIOQ diferenciada, que objetiva atender qualidade de serviço. Ou seja, as filas de entrada deixaram de ser do tipo FIFO e passaram a obedecer a uma disciplina de prioridade.

O capítulo 3 apresenta a modelagem de simulação proposta. Nele são mostrados e detalhados o funcionamento de dois modelos de simulação, utilizando o *software* ARENA: um para a disciplina das filas de entrada tipo FIFO, e o outro para a disciplina com prioridades. Estes modelos foram a base para gerar todos os outros modelos de simulação utilizados e constituem a parte principal deste trabalho.

No capítulo 4 são comentados e mostrados, através de gráficos, os resultados das diversas simulações realizadas. É feita também a comparação dos resultados obtidos em [1] e [2] com os resultados obtidos neste trabalho, comprovando a consistência dos modelos desenvolvidos.

Finalmente, no capítulo 5, são apresentadas as conclusões finais e as propostas de estudos futuros para a sequência deste trabalho.

CAPÍTULO 2

2 - COMUTADORES

2.1. PRINCIPAIS FORMAS DE COMUTAÇÃO

2.1.1. COMUTAÇÃO DE CIRCUITOS

A comutação por circuitos é indicada para os casos em que se tem tráfego constante. Na comutação de circuitos, uma mensagem da estação de origem é enviada até a estação de destino, passando ou não por outras estações. Caso haja recursos disponíveis, há uma alocação sequencial de uma rota entre origem e destino. A estação de destino retorna uma confirmação quando todas as ligações entre nós intermediários estiverem alocadas. Os recursos alocados em todo circuito são exclusivos. As mensagens são então enviadas sem nenhum tipo de processamento intermediário. Quando uma das estações desfaz a conexão, ocorre a liberação do meio de transmissão (circuitos). Quando a rede estabelece o circuito, ela também reserva uma taxa de transmissão constante nos canais de comunicação da rede durante o período da conexão. Esta reserva permite que o remetente transfira os dados ao destinatário a uma taxa constante garantida.

A comutação de circuitos pode se dar por:

- **Chaveamento físico**: caminho físico formado por uma sucessão de enlaces físicos;
- **Chaveamento de frequência**: sucessão de canais de frequência, alocados em cada enlace;
- **Chaveamento de tempo**: sucessão de canais de tempo, alocados em cada enlace;
- **Combinação dos chaveamentos acima**.

As onipresentes redes de telefonia são exemplos clássicos de comutação de circuitos.

2.1.2. COMUTAÇÃO DE MENSAGENS

Nesta forma de comutação, nenhum caminho físico é estabelecido com antecedência entre o transmissor e o receptor. Em vez disto, quando o transmissor tem um bloco de dados a ser enviado, esse bloco é armazenado na primeira estação de comutação e depois é encaminhado, um salto (*hop*) de cada vez. Cada bloco é recebido integralmente, inspecionado em busca de erros, e depois retransmitido. Uma rede que utiliza esta técnica é

chamada *store-and-forward*. Na comutação de mensagens é acrescentado, a cada mensagem original, um cabeçalho, o qual indica o destino da mesma. Como não existe uma definição prévia do circuito, em cada nó a rede decide qual o próximo encaminhamento da mensagem, ou seja, em cada nó da rede é alocada, dinamicamente, uma rota para a mensagem. Como não existe reserva prévia de recursos pode haver a formação de fila nos nós internos da rede pois pode haver acúmulo de mensagens disputando um mesmo enlace. Desta forma, os nós intermediários devem ser providos de uma forma de armazenamento destas mensagens, os *buffers*.

2.1.3. COMUTAÇÃO DE PACOTES

É uma variação da comutação por mensagem pois na comutação de mensagens não há nenhum limite sobre o tamanho do bloco, o que significa que os roteadores (em um sistema moderno) devem ter discos para armazenar temporariamente no *buffer* blocos longos. Isto também significa que um único bloco pode obstruir um linha entre roteadores por alguns minutos, tornando a comutação de mensagens inútil para o tráfego interativo. As redes de comutação de pacotes impõem um limite restrito sobre o tamanho do bloco, permitindo que os pacotes sejam armazenados na memória do roteador e não em um disco. Assegurando que nenhum usuário poderá monopolizar uma linha de transmissão durante muito tempo (milisegundos), as redes de comutação de pacotes se adequam bem à manipulação de tráfego interativo. Na comutação por pacotes, se a mensagem tiver tamanho superior ao tamanho definido para o bloco, a mesma é quebrada em pacotes que podem ou não serem de mesmo tamanho. Na rede ATM, por exemplo, os pacotes têm tamanho fixo de 53 *bytes*, sendo 48 *bytes* de informação e 5 *bytes* de cabeçalho. Estes pacotes são chamados de células. Como os pacotes são de tamanho reduzido, existe a necessidade relativa de pouca capacidade de armazenamento nos nós intermediários. Essa característica também propicia uma comutação mais rápida dos pacotes no interior da rede.

A Internet talvez seja hoje, o melhor exemplo de redes de comutação de pacotes.

A comutação de circuitos e a comutação por pacotes diferem em muitos aspectos. Para começar, a comutação de circuitos exige que um circuito seja configurado de ponta a ponta antes de iniciar a comunicação. A comutação de pacotes baseada em datagramas [19][20] não exige qualquer configuração antecipada. O primeiro pacote pode ser enviado assim que esteja disponível. Entretanto, a comutação de pacotes baseada em circuitos

virtuais [19][20] exige o estabelecimento de um circuito lógico antes que qualquer informação seja transmitida.

O resultado do estabelecimento de um circuito virtual é a reserva de largura de banda em todo o percurso. Todos os pacotes seguem este caminho. Fazer todos os pacotes seguirem o mesmo caminho significa que os mesmos não podem chegar fora de ordem. Na rede de datagramas, não há nenhum caminho fixo, e assim, dependendo das condições da rede, diferentes pacotes podem seguir caminhos diferentes. Portanto, eles podem chegar fora de ordem.

A comutação de pacotes baseada em datagramas é mais tolerante a falhas que a comutação por circuitos, visto que, em caso de falha em um nó, o tráfego pode ser roteado de forma a contornar o nó inativo. Este mesmo problema ocorre quando são utilizados circuitos virtuais.

Nos serviços não orientados a conexão (datagramas), uma mensagem é tratada de forma individual e entregue ao destino, através do caminho mais conveniente, definido pelos algoritmos de roteamento. A inter-rede não dá garantia de entrega em seqüência dos pacotes e, muitas vezes, nem garante a entrega de um pacote no destino final (datagrama não confiável). Neste caso, os níveis superiores se encarregam do controle de erro, de sequência e de fluxo, quando necessário. Este tipo de serviço é útil onde não é essencial que se garanta a entrega de todos os pacotes como, por exemplo, algumas aplicações de transmissão de voz.

Nos serviços orientados a conexão (circuito virtual), um caminho lógico é estabelecido entre a origem e o destino, permanecendo inalterado até o fim da comunicação, quando deve ser desfeito. Por poder conter mecanismos implícitos de controle de erro e de fluxo que garantem, entre outras coisas, a sequência de entrega dos pacotes ao destino. Além disto, nos serviços orientados a conexão, é possível fazer a pré-alocação de recursos (por exemplo, recursos de armazenamento nos *gateways* e nós).

Um resumo das diferenças entre as redes por comutação de circuitos e comutação por pacotes é mostrada na Tabela 2.1.

Item	Comutação de circuitos	Comutação de pacotes	
		Datagramas	**Circuitos Virtuais**
Configuração de chamadas	Obrigatória	Não necessária	Necessária
Caminho físico dedicado	Sim	Não	Não
Cada pacote segue a mesma rota	Sim	Não	Sim
A informação chega em ordem	Sim	Não	Sim
A falha de um nó é fatal	Sim	Não	Sim
Largura de banda disponível	Fixa	Dinâmica	Dinâmica
Momento de possível congestionamento	Durante a configuração	Em todos os pacotes	Em ambas as situações
Largura de banda potencialmente desperdiçada	Sim	Não	Sim
Transmissão *store-and-forward*	Não	Sim	Sim
Transparência	Sim	Não	Não

Tabela 2. 1 - Uma comparação entre redes de comutação de circuitos e redes de comutação de pacotes

2.2. COMUTADORES ATM

Os comutadores ATM utilizam comutação de pacotes baseada em circuitos virtuais. Um comutador ATM é o elemento de rede que tem por objetivo transferir um pacote de um enlace de entrada para seu respectivo enlace de saída. No caso da rede ATM, todos os pacotes têm tamanho fixo de 53 *bytes* e são chamados de células. Na comutação rápida de pacotes, procura-se diminuir ao máximo o processamento em cada nó da rede. Este processamento fica restrito às camadas de enlace e de rede do modelo de referência OSI (*Open Systems Interconnection*), [19] e [20]. Para que a rede ATM seja eficiente, ou seja, para que a mesma consiga atingir seu objetivo de alta transferência de células, os comutadores ATM devem possuir algumas características especiais, conforme veremos na seção a seguir.

De acordo com Pattavina [21], a maioria dos comutadores ATM são construídos a partir de arranjos de múltiplos estágios de elementos de comutação simples (SEs – *Switching Elements*), que formam uma rede de interconexão (IN – *Interconnection Network*). Entretanto, existem outros tipos de arquiteturas de comutadores que não são

baseadas em estruturas multiestágio, como por exemplo comutadores de memória compartilhada.

Como uma rede ATM transporta vários tipos de serviços, as taxas de tráfego sob a mesma podem ser bastante variáveis. A seguir, são comentados alguns requisitos quanto ao serviço que um comutador deve apresentar, segundo De Pricker [6].

2.2.1. REQUISITOS DE UM COMUTADOR ATM

São requisitos de um comutador ATM:

- **Taxa de transferência de dados**: Quantidade máxima de células comutadas por unidade de tempo em uma mesma conexão. Como existem conexões com taxas diferentes, o comutador deverá fazer o desacoplamento de taxa, inserindo células nulas nas conexões com taxas menores. A taxa básica de transferência é de 155.52 Mbps. Em algumas estruturas, o comutador pode trabalhar internamente com taxas maiores, dependendo da topologia adotada.
- ***Broadcast* e *Multicast***: O comutador deverá ser capaz de identificar células destinadas aos tráfegos *broadcast*, ou seja, células que serão encaminhadas a todos os usuários, e as células do tráfego *multicast*, que deverão ser encaminhadas a um grupo específico de usuários, fazer as duplicações e encaminhá-las corretamente.
- **Desempenho**: Os parâmetros de desempenho indicam a qualidade do comutador. Como principais parâmetros de desempenho podem ser citados:
  - Comprimento médio da fila;
  - Probabilidade de bloqueio;
  - Perda de células;
  - Atraso no envio de células;
  - Vazão de dados.

Este trabalho tem como foco o requisito desempenho, citado acima. Especificamente, os dois primeiros parâmetros: comprimento médio da fila e probabilidade de bloqueio.

2.3. ARQUITETURA DOS COMUTADORES

Como citado anteriormente, os comutadores se diferem de acordo com a posição/utilização de seus elementos de memória, ou seja, seus *buffers*.

De acordo com Pattavina [21], quatro critérios podem ser utilizados para classificar as arquiteturas de comutadores ATM:

- **Bloqueio de Células**: Refere-se a possibilidade de perda ou não de células durante o processo de comutação.
- **Número de Planos de Comutação**: Diz respeito ao número de planos da arquitetura de comutação. As opções são: um plano único de comutação (*single plane*) ou vários planos equipados trabalhando em paralelo (*parallel planes*).
- **Posição dos Elementos de Memória**: Diz respeito a posição dos elementos de memória (*buffers*) na rede de interconexão. Três configurações são utilizadas: *buffers* na entrada, *buffers* na saída e *buffer* compartilhado.
- **Profundidade da Rede**: Refere-se as soluções utilizadas para manter a comutação do tráfego através da rede de interconexão, mediante a ocorrência de um evento de perda de célula. As opções são: profundidade mínima (*minimum depth*) e profundidade arbitrária (*arbitrary depth*). A solução de profundidade mínima é baseada no armazenamento das células dentro dos elementos de comutação (SEs). A solução de profundidade arbitrária é baseada no roteamento através de múltiplos caminhos na rede de interconexão, sem armazenamento dentro dos SEs.

A seguir, apresentaremos algumas das arquiteturas elementares de comutadores ATM presentes na literatura. Focaremos nas arquiteturas sem bloqueio e com plano de comutação único.

2.3.1. *BUFFER* NA ENTRADA

Estes comutadores [8] normalmente apresentam *buffers* simples do tipo FIFO (*First In, First Out*) em cada entrada, conforme mostrado na Figura 2.1. Como existem *buffers* independentes para cada entrada, os mesmos crescem linearmente com o aumento do número de entradas. A lógica que decide qual dos terminais de entrada deve ser servido, pode ser muito simples, tipo cíclico, até muito complexa, levando em conta o nível de enchimento dos *buffers*.

Esta estrutura possui a característica de apresentar um problema denominado HOLB (*Head Of Line Blocking*), que limita a vazão máxima em aproximadamente 58% [7] para fontes de tráfego de entrada do tipo Bernoulli, independentes e identicamente distribuídas. Muitas estruturas do tipo *buffer* na entrada têm sido propostas e analisadas na literatura [10], [11], [12], [13],[15], [16].

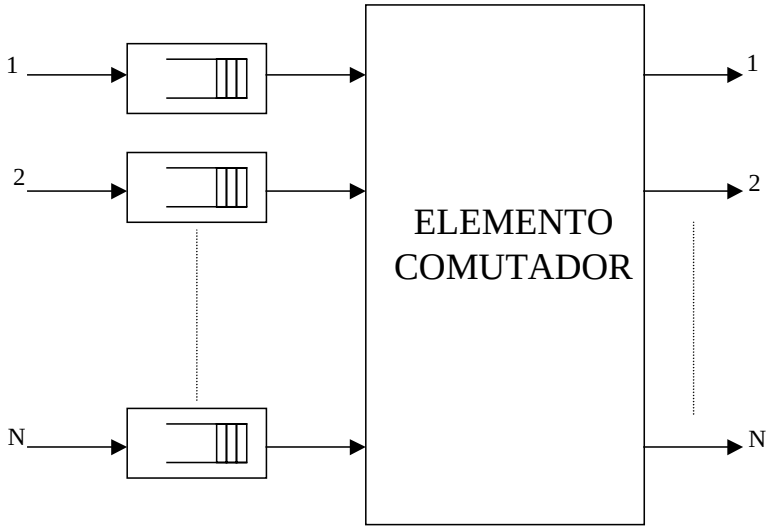

Figura 2.1 – Comutador com *buffers* na entrada.

O problema do HOLB pode ser facilmente explicado conforme mostra a Figura 2.2: suponha que as entradas 1 e 4 do comutador 4X4 tenham células a serem transferidas no mesmo *slot* de tempo para a saída 3. Suponha também que a célula escolhida para ser transferida neste *slot* foi a célula proveniente da entrada 1. Neste caso, a célula que está na entrada 4 será bloqueada, pois existe um conflito. Pode-se reparar que a próxima célula na fila de entrada 4 tem como destino a saída 2 que não possui, neste *slot*, nenhuma outra célula destinada a ela, ou seja, a mesma está livre. Mesmo assim, esta célula não pode ser transferida neste *slot* de tempo, caracterizando um bloqueio da mesma.

Existem muitos estudos que visam melhorar o problema do HOLB. Na sequência deste trabalho, será abordado um pouco mais a respeito.

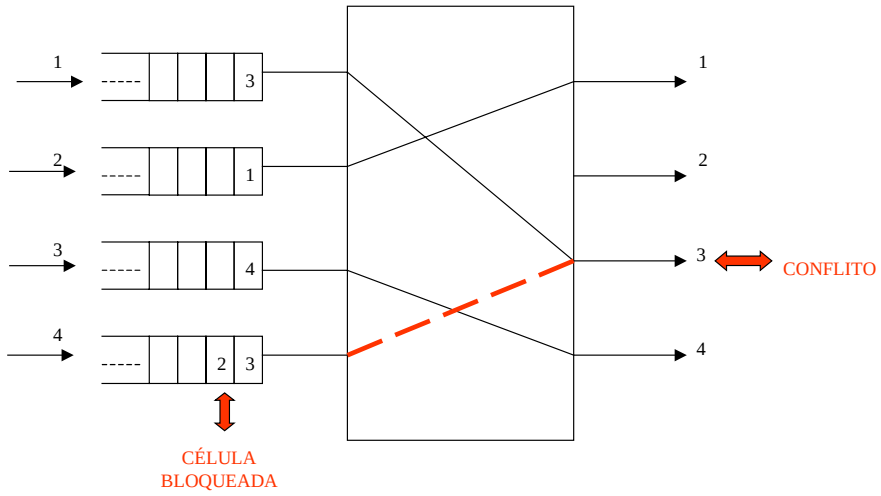

Figura 2.2 – Exemplo de *Head Of Line Blocking* (HOLB)[8].

De acordo com Karol [7], se supormos que todas as filas de entrada de um comutador NxN estejam saturadas de tal forma que sempre hajam células aguardando em cada fila de entrada, o número de células nas cabeceiras das filas que são bloqueadas para uma determinada saída *i* ao final do m-ésimo *time slot* é dado por:

$$B_m^i = \max\left(0, B_{m-1}^i + A_m^i - 1\right), \tag{1}$$

onde B_{m-1}^i é o número de células bloqueadas durante o *slot* anterior e A_m^i é o número de células que chegaram no *slot m*. Fazendo-se N $\rightarrow \infty$ e considerando-se o comutador em regime permanente, o número médio de células bloqueadas é dado por:

$$\overline{B}^i = \frac{{\rho_0}^2}{2\left(1-\rho_0\right)}, \tag{2}$$

onde ρ_0 é a média do número de células que são transferidas das cabeceiras das filas para a saída *i*, ou seja a vazão de cada saída.

Continuando com os resultados de [7], temos, que a cada *time slot* o comutador pode transferir no máximo N células da entrada para a saída. Assim, o número médio de células que iniciam serviço em regime permanente ($\overline{L}$) é igual N menos o número médio de células bloqueadas devido ao HOLB ($\overline{B}^i$) em todas as entradas. Ou seja:

$$\overline{L} = N - \sum_{i=1}^{N} \overline{B}^i , \tag{3}$$

$\overline{L}$ pode ser reescrito como o número de saídas (N) vezes a vazão de cada saída (ρ_0):

$$\overline{L} = N\rho_0 . \tag{4}$$

Então (3) pode ser reescrita como:

$$N\rho_0 = N - \sum_{i=1}^{N} \overline{B}^i . \tag{5}$$

Substituindo-se $\overline{B}^i$ em (5) e isolando-se ρ_0 temos:

$$\rho_0 = 2 - \sqrt{2} = 0{,}586 = 58{,}6\% . \tag{6}$$

Esta é a vazão máxima de saída para um comutador com *buffers* FIFO na entrada.

O atraso médio dos pacotes e a probabilidade de bloqueio para um comutador com *buffers* na entrada podem ser encontrados em [21].

2.3.2. *BUFFER* NA SAÍDA

A idéia consiste em transferir todas as células de entrada para as respectivas saídas. Como só uma célula pode ser atendida por cada saída em um intervalo de um *slot*, existe a necessidade de um *buffer* em cada saída.

Os comutadores com *buffer* na saída, do tipo ilustrado na Figura 2.3, normalmente possuem *buffers* do tipo FIFO em cada saída. Este comutador tem como característica principal o fato de possuir uma vazão máxima teórica igual a 100% [1]. Esta vazão é teórica, pois, para ser alcançada, o comutador deve ser capaz de armazenar todas as células de saída durante o intervalo de um *slot*.

Um dos problemas principais deste tipo de comutador é o fato de descartar células em um momento de congestionamento, após as mesmas terem passado pela matriz de comutação, o que pode contribuir para a degradação do sistema.

Pode-se encontrar, para redução da perda de células, sistemas dimensionados para o pior caso, ou seja, considera-se que as células de todas as entradas vão para uma mesma saída. Neste caso, normalmente há uma superestimação do tamanho do *buffer*. Este comutador se caracteriza, também, pela necessidade de um meio de transferência, ou seja, um barramento interno, que possua uma velocidade N vezes a velocidade de cada entrada, para que o mesmo consiga atender todas as entradas.

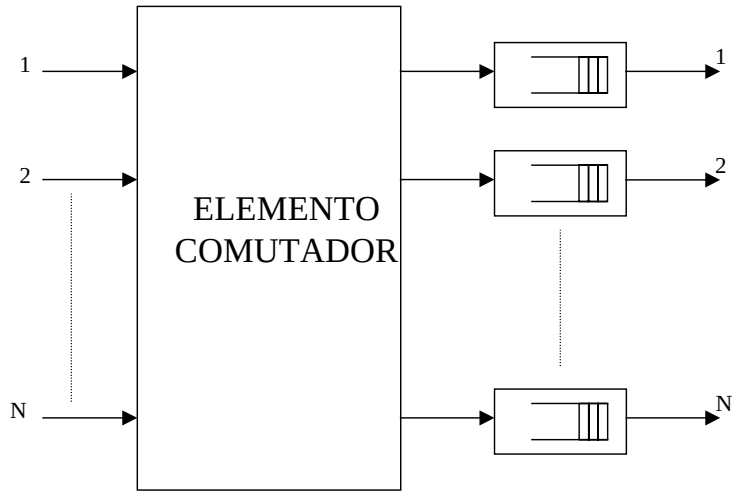

Figura 2.3 – Comutador com *buffers* na saída.

O tempo médio de permanência da células para um comutador NxN é dado por [6]:

$$\overline{W} = \frac{(N-1)}{N}.\overline{W}_{MD/1}, \tag{7}$$

onde $\overline{W}_{M/D/1}$ é o tempo médio em um sistema M/D/1. Em Karol et. al. [7] é mostrada a curva da expressão (7). Esta curva mostra que o atraso médio apresenta valores abaixo da duração de um *time slot* para cargas de até 80%. Acima desta carga, o atraso aumenta rapidamente.

A probabilidade de bloqueio para um comutador com *buffers* na saída pode ser encontrada em [21].

2.3.3. *BUFFER* COMPARTILHADO

Nesta estrutura, não existe mais um *buffer* para cada entrada ou para cada saída, e sim, apenas um único *buffer* que é compartilhado por todos. As células são armazenadas neste *buffer* em uma fila física aleatória, mas são lidas segundo uma fila lógica do tipo FIFO. Como o *buffer* é compartilhado, tem-se uma otimização da utilização do *buffer*, mas, em troca, aumenta-se significativamente a complexidade do gerenciamento do mesmo.

Para que esta estrutura funcione é necessário um dispositivo com alta velocidade de escrita/leitura na memória, o que pode ser inviável em alguns casos. Um comutador com *buffer* compartilhado é mostrado na Figura 2.4.

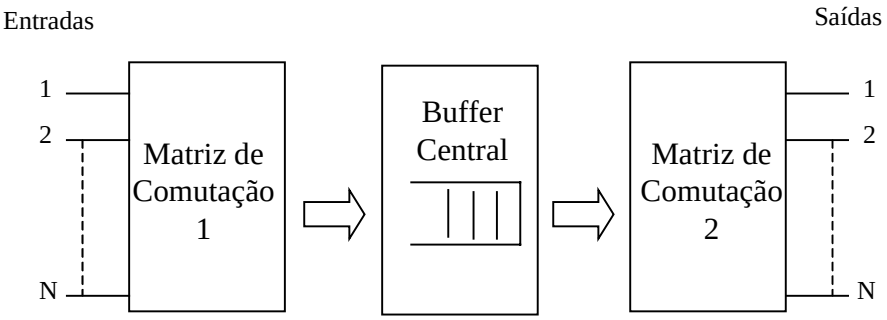

Figura 2.4 – Comutador com *buffer* compartilhado [7][8].

Estes três tipos de estruturas apresentadas possuem desempenhos diferentes em relação ao comprimento das filas, aos atrasos e às perdas de células, levando-se em conta o padrão de tráfego e a carga nas entradas do comutador.

Intuitivamente, pode-se notar que, para uma mesma carga, o comprimento médio das filas e o tempo médio de espera na fila são maiores para estruturas com *buffer* na entrada do que para estruturas com *buffers* na saída ou *buffer* compartilhado.

Isto ocorre devido principalmente, ao HOLB, pois é possível, no caso dos *buffers* de entrada, que células de entrada sejam bloqueadas, enquanto esperam por uma célula da mesma fila à sua frente ser servida, mesmo que ambas possuam destinos diferentes e que o destino da célula posterior esteja livre.

O atraso médio dos pacotes e a probabilidade de bloqueio para um comutador com *buffers* compartilhados podem ser encontrados em [21].

Pode-se explicar esta conclusão intuitiva a partir do seguinte exemplo: suponha que um gerente de uma loja decida colocar dois caixas diferentes para recebimento de suas vendas: um caixa para receber os pagamentos em dinheiro (*cash* – C), e outro caixa para receber em outras formas de pagamento, como cheques e cartões de crédito (outros – O).

Supondo que exista apenas uma fila com disciplina tipo FIFO, conforme mostrado na Figura 2.5, a pessoa, ou pessoas que farão pagamento com cheques ou cartões terão que esperar a pessoa que está à frente para fazer seu pagamento em dinheiro, ser atendida, mesmo que o caixa destinado a outras formas de pagamento esteja vazio. Este exemplo representa uma configuração do tipo *buffers* de entrada.

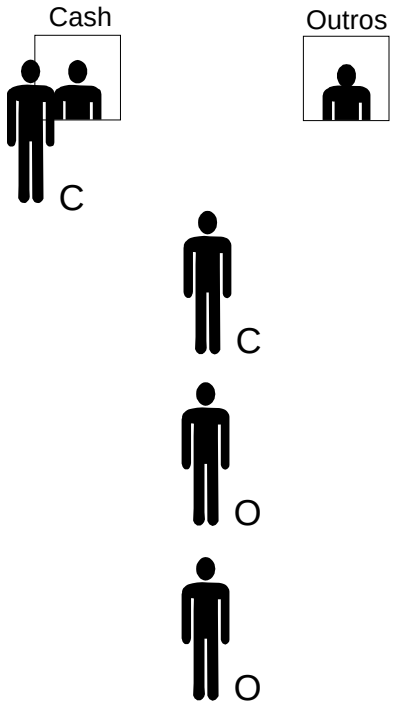

Figura 2.5 – *Buffer* de entrada em um atendimento.

Suponha agora, conforme mostrado na Figura 2.6, que existam duas filas, uma para cada forma de pagamento. As pessoas decidirão em qual fila entrar, de acordo com a forma de pagamento escolhida. Neste caso, os atendentes dos caixas ficarão menos tempo ociosos, pois, sempre que houver alguma pessoa destinada ao seu respectivo caixa ela será atendida. Isto reduzirá o número de pessoas na fila e também o tempo de espera em relação ao exemplo anterior. Este exemplo representa a configuração com filas na saída.

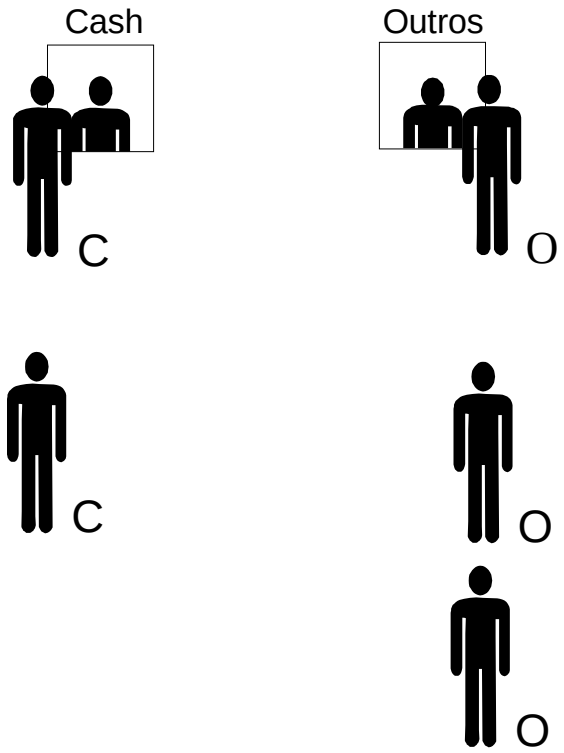

Figura 2.6 – *Buffer* de saída em um atendimento.

Se o número de posições na fila a serem disponibilizadas for um parâmetro importante no dimensionamento do sistema, então, pode-se notar que o modelo com fila na saída requer um maior número de posições disponibilizadas, visto que todas as pessoas que comprarem podem resolver efetuar o pagamento apenas em *cash*, ou apenas com cartões e cheques. Desta forma, as posições devem ser dimensionadas para o pior caso.

Pode-se reduzir o número de posições a serem disponibilizadas, usando uma fila única, mas que seja gerenciada por uma pessoa chamada de servidor (S), conforme ilustrado na Figura 2.7. Este servidor perguntará às pessoas da fila, de qual forma elas gostariam de efetuar o pagamento. Desta forma, pessoas que por ventura estiverem no meio, ou mesmo no final da fila, poderão ser servidas antes das que estiverem à frente, desde que seu caixa esteja livre. Então, esta solução irá diminuir o número de posições a serem disponibilizadas, mas o preço a ser pago é a necessidade do uso de um servidor inteligente. Este exemplo ilustra a configuração com *buffer* compartilhado.

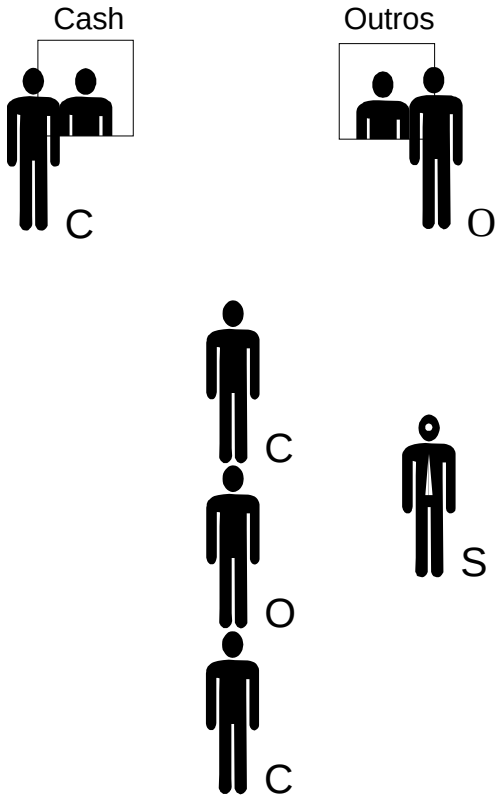

Figura 2.7 – *Buffer* compartilhado em um atendimento.

Além destas estruturas básicas citadas, existem propostas em que se mesclam os posicionamentos dos *buffers* visando obter as características que cada uma delas tem de melhor. Uma destas combinações é justamente o objeto deste estudo, em que foram combinados *buffers* individuais em cada entrada, e um conjunto de até *m buffers* em cada saída. Esta estrutura é denominada CIOQ (*Combined Input-Output Queuing*) e será objeto de estudo a partir do próximo capítulo.

Utilizando-se da mesma analogia usada nas Figuras 2.5, 2.6 e 2.7 para representar, respectivamente, os modelos de filas na entrada, filas na saída e filas compartilhadas, o modelo CIOQ poderia ser representado conforme a Figura 2.8. Neste caso, seriam organizadas várias filas na entrada. Haveria agora um servidor para cada tipo de serviço e cada servidor retiraria a cada ciclo até *m* clientes da cabeceira das filas de entrada para a fila de saída apropriada. Isto reduziria o problema do HOLB, conforme será visto posteriormente. Quanto maior o valor de *m*, menor o comprimento médio das filas de entrada e a probabilidade de bloqueio. Neste caso, para que uma pessoa fique na fila de

entrada devido apenas à influência de pessoas interessadas em outra forma de pagamento, ou seja, outra fila de saída, deverá haver pelo menos *m* mais uma (*m* + 1) pessoas a sua frente que preferem outra forma de pagamento. Na verdade, existem *m* filas físicas em cada saída as quais podem ser representadas como uma única fila lógica conforme mostrado na Figura 2.8.

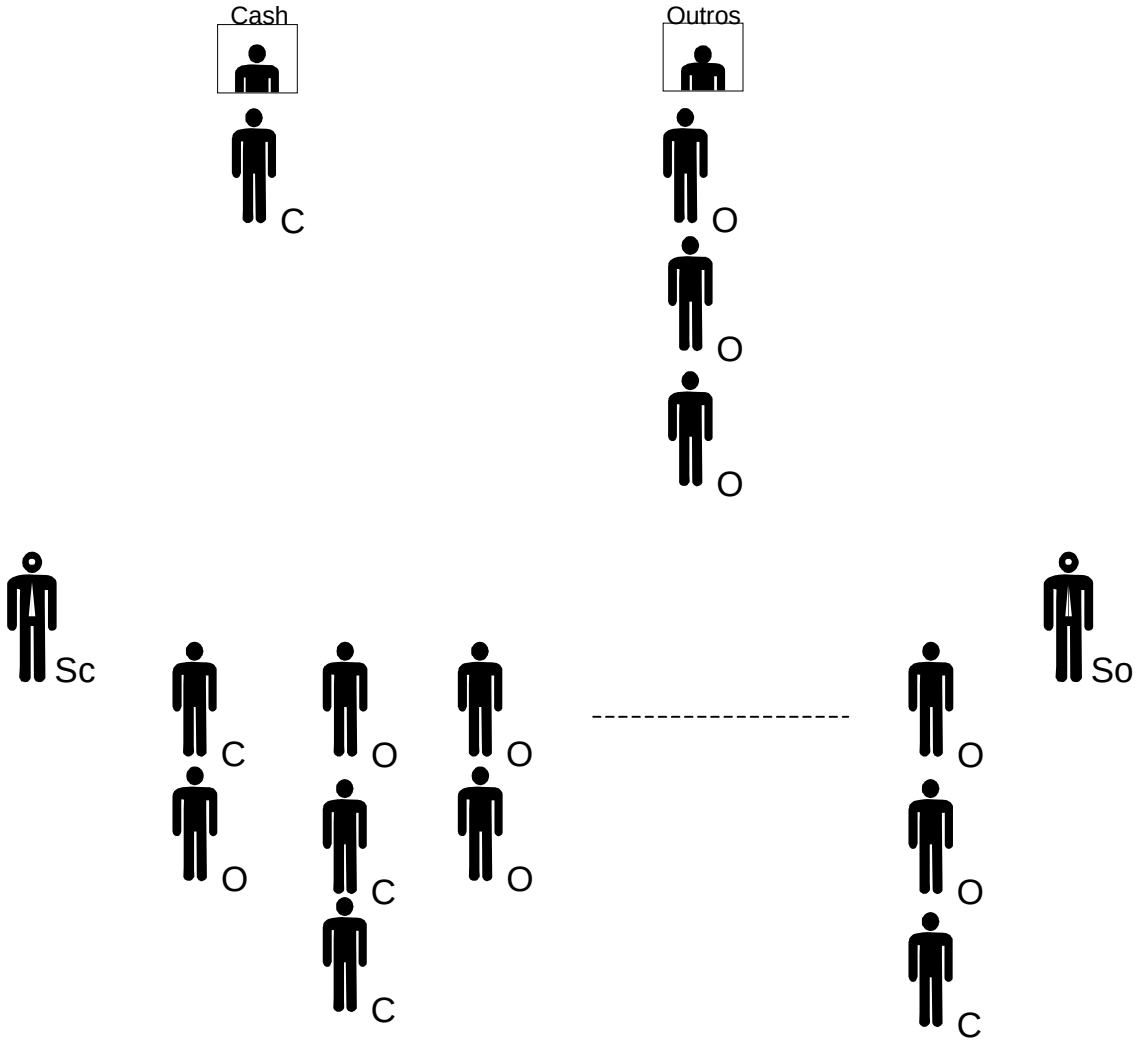

Figura 2.8 – Estrutura tipo CIOQ.

2.3.4. COMUTADORES DO TIPO CIOQ - COMBINED INPUT-OUTPUT QUEUING

Como citado anteriormente, diversos estudos foram e ainda estão sendo realizados baseados na tecnologia ATM. Em se tratando de comutadores ATM, pode-se destacar duas estruturas: *Virtual Output Queuing* (VOQ) e *Combined Input-Output Queuing* (CIOQ). O VOQ é uma estrutura que visa solucionar o problema de HOLB, encontrado nos esquemas com *buffers* na entrada e filas tipo FIFO, e também prover qualidade de serviço. Nesta estrutura, cada porta de entrada possui um *buffer* virtual para cada porta de saída. Já o CIOQ, como o nome sugere, é uma estrutura que combina *buffers* simples, em cada entrada e até *m buffers*, em cada saída. Esta estrutura, ao contrário do esquema com apenas *buffers* simples na entrada, possibilita a transferência de até *m* células para cada saída em um único *slot* de tempo. Esta transferência de células é feita em modo paralelo.

Uma estrutura do tipo CIOQ denominada *High-speed Statistical Retry Switch* (HSR *Switch*) é apresentada em [4]. Trata-se de um comutador de alta velocidade que utiliza um algoritmo simples de repetição e um pequeno *buffer* de entrada. Neste comutador, cada célula de cada entrada é repetidamente transmitida à respectiva saída a uma taxa de *m* vezes a velocidade da porta de entrada até que a mesma receba um sinal de reconhecimento. A taxa interna *m* é escolhida de forma a garantir que a probabilidade máxima de conflito suportada não seja atingida. Apenas uma célula de cada entrada pode ser transmitida em cada ciclo de célula. O algoritmo interno é muito simples: Em cada "*crosspoint*" as células de entrada das linhas horizontais têm prioridade sobre as entradas das linhas verticais. Desta forma, o circuito de decisão em cada *crosspoint* se torna bastante simples e reduzido, o que é uma das principais vantagens deste comutador.

Devido ao algoritmo de decisão usado, não existe um atendimento igualmente distribuído entre as entradas, visto que as *m* primeiras entradas possuem prioridade sobre as restantes. Esta estrutura possui também a indesejável característica de se ter que trabalhar com altas taxas internas, ou seja, *m* vezes maior do que a maior taxa de entrada. Desta forma, quanto maior o número de entradas, mais veloz deve ser a comutação interna. Como as taxas de entrada estão cada vez mais rápidas, a dificuldade de implementação deste comutador se torna um ponto crítico.

Existe também uma outra estrutura, proposta em [5], que é uma combinação do VOQ com o CIOQ. A intenção é a de se combinar as vantagens das estruturas com filas na saída com a diminuição do HOLB proporcionada pela utilização do VOQ. O termo CIOQ tem sido normalmente utilizado para estruturas com filas nas entradas e nas saídas, com arbitração centralizada e velocidade interna (*speedup*) limitada. No entanto, esta combinação utiliza arbitração descentralizada e as filas de saída têm velocidades de acesso à memória N vezes a velocidade das filas de entrada, onde N é o número de entradas/saídas. Os *buffers* de saída, quando aptos a receberem células, enviam um sinal de confirmação aos "árbitros" das entradas que selecionam quais VOQ's enviarão células obedecendo alguma regra pré-determinada, como, por exemplo, o algoritmo *Round Robin*.

Do ponto de vista estrutural, esta arquitetura apresenta duas dificuldades principais, que são: a complexidade de interconexão da memória compartilhada, pois cada entrada e cada saída devem ser capazes de localizar cada posição de memória, e a largura de banda da fila de saída - *the output queue bandwidth* (até N+1 acessos por fila, para cada ciclo).

Neste trabalho, foi abordada a estrutura do tipo CIOQ, especificamente, uma variação do CIOQ, onde, ao invés de se trabalhar com um barramento de alta velocidade internamente ao comutador, são utilizadas *m* linhas, interligando todas as entradas, a cada saída. Desta forma, consegue-se um meio mais eficaz de transferência de células [1] e [2].

Esta estrutura utiliza uma combinação entre *buffer* na entrada e *buffer* na saída, visando obter aquelas características que cada uma destas estruturas tem de melhor. Por exemplo, dimensionamento dos *buffers* de entrada independentes para cada enlace (característica do comutador com *buffer* de entrada) e máxima vazão teórica de 100% (característica do comutador com *buffer* na saída).

Em [1] e [2], foram realizados estudos teóricos visando a obtenção de equações matemáticas que descrevam o comportamento do sistema e também simulações, usando o *software* MATLAB para determinar alguns parâmetros desta estrutura de comutador. Como as simulações usando o MATLAB são do tipo *Monte Carlo*, um dos propósitos deste trabalho é o de comparar os resultados analíticos e simulados em [1] e [2] com resultados obtidos, através de simulação, utilizando um *software* de simulação de eventos discretos. Neste caso, o ARENA 5.0 (*Rockell Software*).

2.3.4.1. ESTRUTURA UTILIZADA

A Figura 2.9 ilustra a estrutura utilizada neste estudo. Como já citado, cada entrada possui um *buffer* do tipo FIFO enquanto se tem *m buffers*, também do tipo FIFO, para cada saída. Cada enlace de entrada é conectado a cada saída através de *m* enlaces físicos.

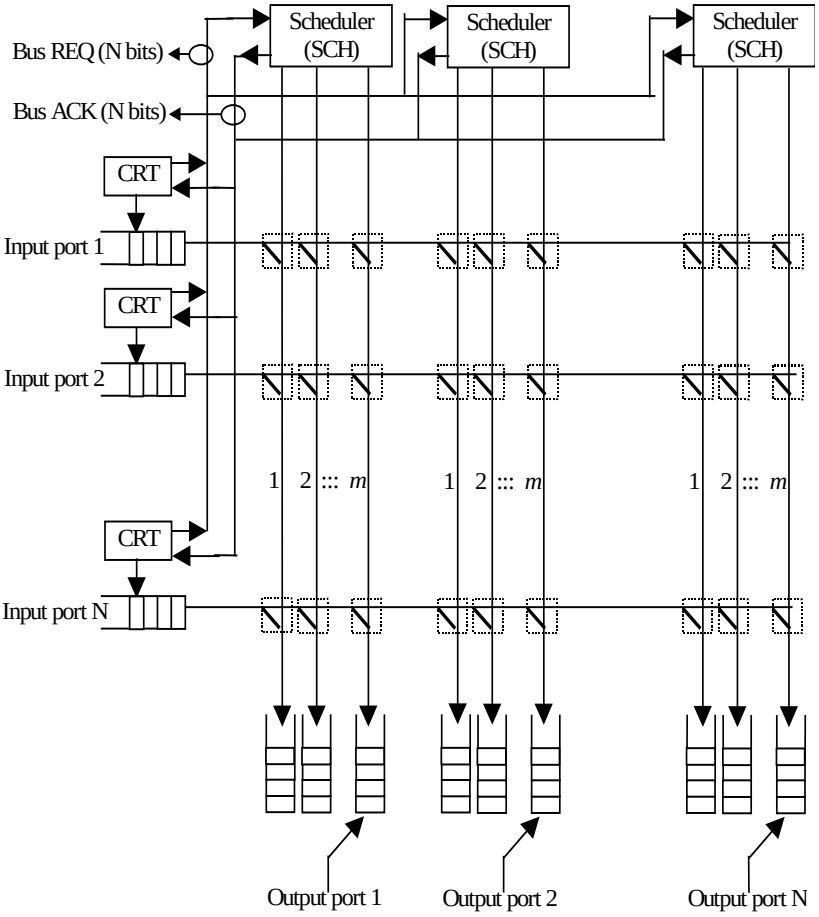

Figura 2.9 – Matriz de comutação com filas de entrada e saída combinadas [2].

Cada porta de entrada possui uma unidade de controle (CTR) que é responsável por indicar se existem ou não células em seus respectivos *buffers* de entrada para serem transferidas. Em caso afirmativo, a unidade de controle envia uma requisição (REQ) para o módulo *scheduler* (SCH) para que a célula daquela entrada possa ser transferida. Apenas uma solicitação pode ser enviada em cada *slot* de tempo por cada módulo CTR. Se houver até *m* solicitações para uma mesma saída em um determinado *slot* de tempo, todas as células serão transferidas. Caso contrário, o módulo SCH selecionará, obedecendo a um

algoritmo do tipo *Round Robin,* as *m* entradas que irão transferir suas células naquele *slot* de tempo. As demais células serão bloqueadas naquele *slot*. Existe um módulo SCH para cada saída. Ele seleciona as entradas a serem atendidas enviando um sinal de aceitação (ACK). Este sinal, bem como o sinal de requisição é composto apenas por um *bit*, visto que cada unidade de controle possui uma linha reservada para REQ e outra para ACK, conectada a cada módulo *scheduler*.

Como em cada saída existem *m buffers*, faz-se necessário um gerenciamento para manter a sequência correta de transmissão, ou seja, para evitar que células provenientes de uma mesma entrada, armazenadas em *buffers* diferentes, entre os *m buffers* existentes em cada saída, sejam transmitidas fora da ordem correta. Este gerenciamento também é feito pelo módulo *scheduler*. O controle também é realizado com a implementação de um algoritmo do tipo *Round Robin*. O algoritmo *Round Robin* funciona da seguinte forma: em cada ciclo, são lidas todas as entradas. A primeira entrada que tiver uma célula bloqueada em um ciclo será a primeira entrada a ser atendida no próximo ciclo, seguindo a sequência normalmente, a partir de então. Esta sequência será mantida até que haja uma célula bloqueada em uma outra entrada, o que alterará a ordem de atendimento, novamente.

2.3.4.2. FUNCIONAMENTO DO COMUTADOR

Seja o comutador da Figura 2.10, o qual possui três *links* internos para cada saída, ou seja, m=3. Suponha que em determinado *slot* de tempo as entradas i_1, i_2, i_3, i_4 e i_5 tenham células a serem transmitidas para mesma saída j. Então, cada entrada enviará uma requisição ao módulo *scheduler* (SCH). Após receber as requisições, o módulo *scheduler*, seguindo um algoritmo *Round Robin,* envia um sinal de liberação (ACK) para, por exemplo, as entradas i_1, i_2, e i_3. O módulo *scheduler* envia também um sinal habilitando os *links* m_1, m_2 e m_3, de forma a transmitir as células das 3 primeiras entradas para a saída j.

Apenas no próximo ciclo é que as entradas i_4 e i_5 serão atendidas, utilizando os *links* internos m_1 e m_2, respectivamente.

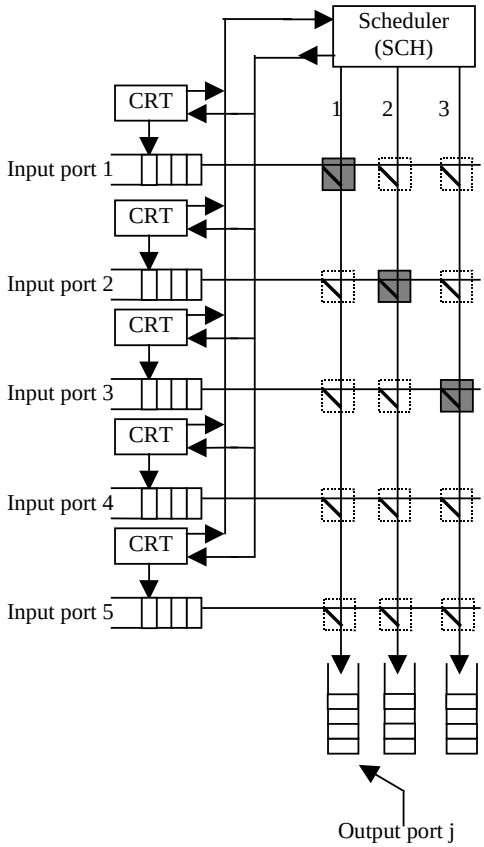

Figura 2.10 – Funcionamento básico do comutador [2].

De acordo com [1], assumindo-se que o número de entradas e saídas é N (matriz N x N), a chegada das células obedecem uma distribuição de Bernoulli independentes e identicamente distribuídas, a probabilidade de uma célula chegar em um slot é ρ, e *m* é o número de *links* internos, então, a probabilidade de uma célula ser enviada para uma porta específica é ρ/N e a probabilidade de j células serem enviadas a uma porta específica é

$$\binom{N}{j}\left(\frac{\rho}{N}\right)^{j}\left(1-\frac{\rho}{N}\right)^{N-j} \tag{8}$$

A probabilidade de até *m* células serem transmitidas a uma porta específica é

$$\sum_{j=1}^{m}\binom{N}{j}\left(\frac{\rho}{N}\right)^{j}\left(1-\frac{\rho}{N}\right)^{N-j} \tag{9}$$

O bloqueio irá ocorrer quando houver mais de *m* células destinadas à mesma saída. Então, a probabilidade de bloqueio é dada por

$$B = \sum_{j=m+1}^{N} \binom{N}{j} \left(\frac{\rho}{N}\right)^{j} \left(1-\frac{\rho}{N}\right)^{N-j} \quad (10)$$

Finalmente, a probabilidade média de bloqueio será

$$B_{av} = \sum_{j=m+1}^{N} (j-m) \binom{N}{j} \left(\frac{\rho}{N}\right)^{j} \left(1-\frac{\rho}{N}\right)^{N-j} \quad (11)$$

2.3.5. COMUTADOR COM CLASSE DE SERVIÇO – ESTRUTURA MODIFICADA

Como pode ser verificado, a estrutura vista anteriormente não é capaz de prover qualidade de serviço, pois todas as filas de entrada são únicas e do tipo FIFO. Então, esta estrutura inicial foi modificada para que a mesma fosse capaz de prover até 5 diferentes classes de serviço. É importante verificar que, para *h* classes de serviço, as linhas de REQ e ACK devem ser capazes de transportar *n* bits, onde $h=log_2 n$ [2]. Neste caso, cada entrada passa a ter *h buffers* do tipo FIFO e cada saída passa a ter *h*x*m buffers*, onde *m* é o número de *links* internos.

A Figura 2.11 mostra a nova estrutura, onde cada entrada possui 5 (*h*=5) diferentes classes de serviço que são denominadas: CBR, rtVBR, nrtVBR, ABR e UBR. Pode-se verificar que neste caso, os *buffers* de classes de serviço de entrada podem ser lógicos. Ou seja, com memória compartilhada, mas os *buffers* de saída são do tipo físico.

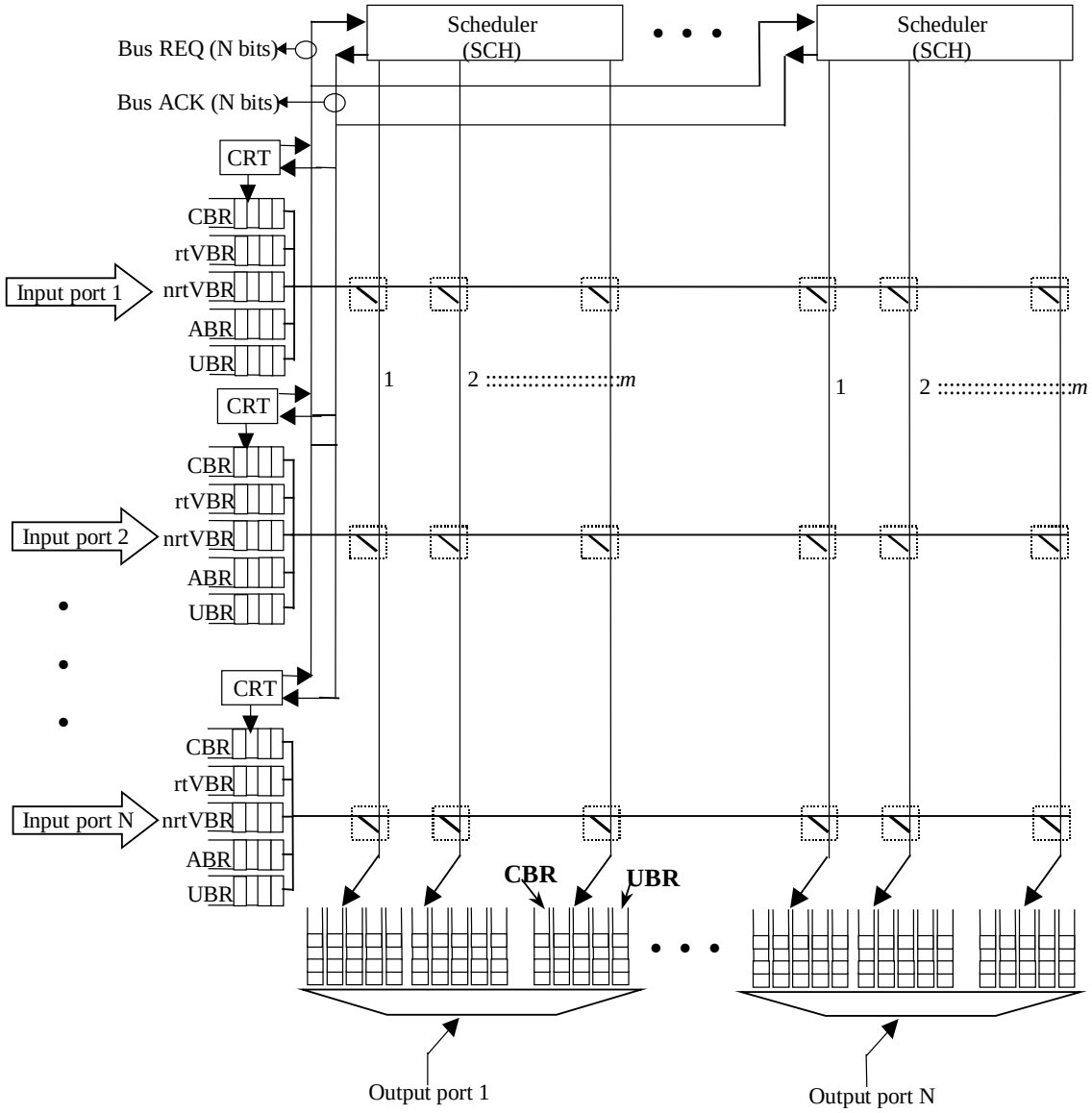

Figura 2.11 – Estrutura modificada para atender qualidade de serviço [2].

As células de chegada são analisadas e armazenadas em suas devidas posições de memória, de acordo com a classe de serviço. As classes de serviço têm prioridade decrescente, obedecendo a seguinte ordem: CBR, rtVBR, nrtVBR, ABR e UBR. Ou seja, CBR possui a maior prioridade e UBR possui a prioridade mais baixa. As unidades de controle (CRT) de cada entrada verificam se existem células, seguindo a prioridade, para transmitir. Se existir, ele envia um código de *n* bits através da linha de requisições (REQ) ao respectivo *scheduler* (SCH), informando o tipo (prioridade) da célula. Cada SCH examina se existem primeiramente células do tipo CBR para transferir em todas as

entradas. Se existir até (m –1) células do tipo CBR, ele as seleciona e passa a verificar as células do tipo rtVBR, e assim por diante, até chegar nas células de menor prioridade UBR. Se existirem mais de m células CBR para transmitir num determinado *slot* de tempo, o SCH seleciona apenas m entradas através da linha ACK. As demais são bloqueadas e atendidas no próximo *slot* seguindo o algoritmo *Round Robin*. Em resumo, se um SCH receber mais que m requisições de diferentes classes de serviço em um mesmo time *slot*, ele selecionará apenas m entradas, obedecendo a hierarquia definida pela classe de prioridade. No próximo ciclo, ele inicia a leitura pela entrada que teve a célula bloqueada de maior prioridade, seguindo sempre, a ordem decrescente de prioridade.

2.3.5.1. FUNCIONAMENTO DO COMUTADOR MODIFICADO

O funcionamento desta estrutura pode ser melhor explicado com a ajuda da Figura 2.12 a seguir. Suponha que as entradas I_1 a I_5 tenham, respectivamente, as células do tipo ABR, CBR, rtVBR, nrtVBR e CBR para serem transferidas para a mesma saída j no mesmo *slot* de tempo. Suponha também que o número de *links* internos m seja igual a 3. Os CRTs de cada entrada enviam os respectivos códigos de prioridade ao módulo SCH, através da linha de requisições (REQ). Após analisá-las, o módulo SCH envia o ACK apenas às portas I_2, I_3 e I_5. Desta forma, apenas estas três entradas terão suas células transferidas para a saída j neste *slot* de tempo. No próximo *slot*, o procedimento é repetido, iniciando-se pela entrada 4.

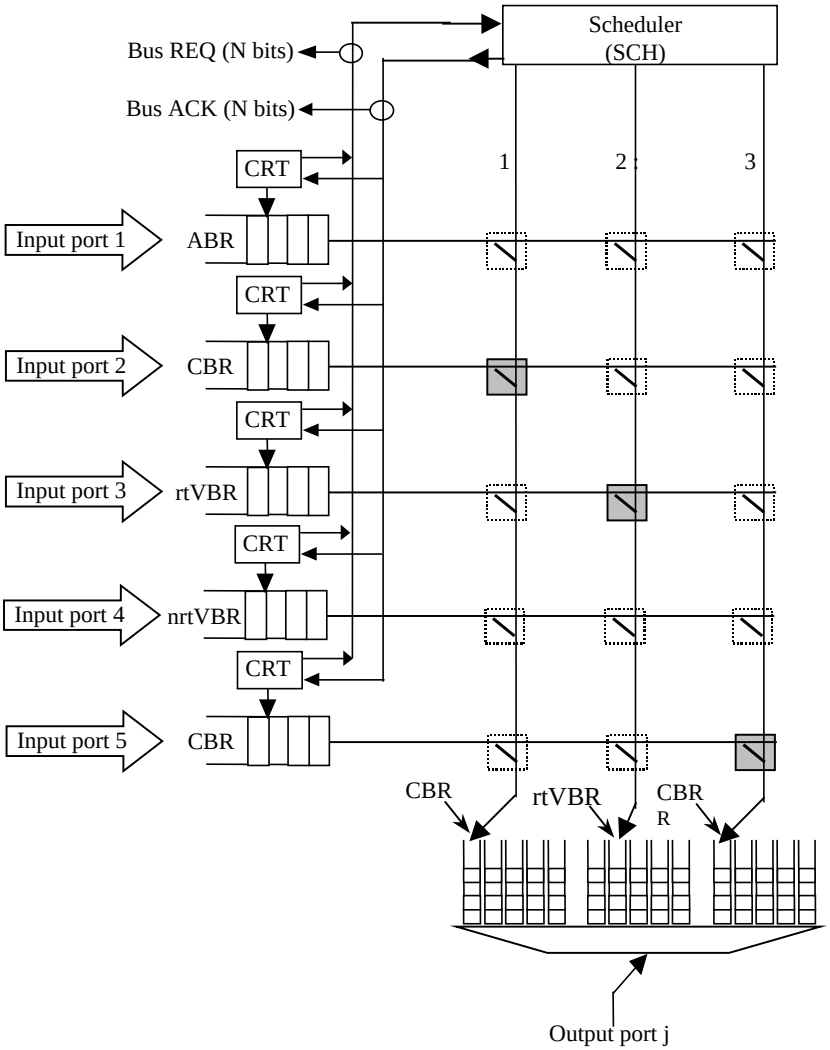

Figura 2.12 – Funcionamento básico da estrutura modificada [2].

CAPÍTULO 3

3 - MODELAGEM DE SIMULAÇÃO PROPOSTA

3.1. INTRODUÇÃO

"Simulação é uma das mais poderosas ferramentas de análise disponíveis para os responsáveis por projeto e operação de processos complexos ou sistemas. Em um mundo de crescente competitividade, a simulação se tornou uma ferramenta muito poderosa para planejamento, projeto e controle de sistemas. Não mais renegado ao posto de "último recurso", hoje ela é vista como uma metodologia indispensável de solução de problemas para engenheiros, projetistas e gerentes"[18].

Simulação é a técnica de estudar o comportamento de um determinado sistema através de modelos que representam, na totalidade ou em parte, as propriedades e comportamentos deste sistema em uma escala menor, permitindo sua manipulação e estudo detalhado, propiciando uma redução de custos e de tempo consideráveis.

Pode-se citar diversos exemplos de aplicações de simulações. Um bom exemplo de simulação é o utilizado na indústria aeronáutica ou de grandes embarcações, como navios. Nestes dois casos, os testes são feitos com pequenas maquetes dos modelos reais a serem testados. A técnica de simulação é aplicada, pois seria inviável a construção de todo o avião, por exemplo, e depois tentar fazê-lo voar com pilotos de prova. A perda de vidas, tempo e investimentos seriam enormes.

As simulações realizadas neste trabalho têm como objetivo principal a determinação de dois parâmetros do comutador ATM CIOQ: probabilidade média de bloqueio e comprimento médio de filas. Para realizar estas simulações, vários modelos de comutadores foram implementados, combinando diversas variações do número de entradas-saídas do comutador (N), número de *links* internos do comutador (*m*) e carga oferecida nas entradas (ρ). Estas simulações foram feitas utilizando-se o *software* de simulação de eventos discretos, ARENA 5.0.

O *software* ARENA é um *software* de simulação de processos por computador [18]. Entende-se por processos uma situação onde elementos estáticos, formando um ambiente bem definido com suas regras e propriedades, interage com elementos dinâmicos, que

fluem dentro deste ambiente. Neste caso específico, o elemento estático é o comutador ATM e os elementos dinâmicos são as células ATM.

Para realizar as simulações, foram implementados, basicamente, dois modelos de simulação: um modelo onde as células de entrada obedecem a uma disciplina de atendimento do tipo FIFO e outro modelo onde as células possuem prioridades, com o objetivo de propiciar qualidade de serviço. No caso em que as células possuem prioridades, foram determinadas 5 classes de prioridades diferentes. Além disto, para cada classe de serviço foi definida uma porcentagem de carga oferecida. A partir destes modelos básicos, foram derivados os demais modelos para a análise das diversas combinações de número de entradas-saídas, carga e número de *links* internos. Estas derivações foram feitas através da alteração dos valores das variáveis envolvidas (para variar a carga e número de *links* internos) e/ou duplicações do modelo básico (para variar o número de entradas–saídas).

Foram implementados, analisados e validados mais de 121 modelos de simulação, considerando-se que a cada alteração dos valores de qualquer um dos parâmetros como número de entradas/saídas, número de *links* internos e carga oferecida, correspondia a um novo modelo. Estes modelos estão listados na Tabela 3.1 a seguir.

Nn	Mm	Rr
16	1	0,9
16	2	0,9
16	3	0,9
16	4	0,9
16	5	0,9
16	6	0,9
16	7	0,9
16	8	0,9
16	9	0,9
16	10	0,9

Nn	Mm	Rr
32	1	0,9
32	2	0,9
32	3	0,9
32	4	0,9
32	5	0,9
32	6	0,9
32	7	0,9
32	8	0,9
32	9	0,9
32	10	0,9

Nn	Mm	Rr
64	1	0,9
64	2	0,9
64	3	0,9
64	4	0,9
64	5	0,9
64	6	0,9
64	7	0,9
64	8	0,9
64	9	0,9
64	10	0,9

Nn	Mm	Rr
64	1	0,7
64	2	0,7
64	3	0,7
64	4	0,7
64	5	0,7
64	6	0,7
64	7	0,7
64	8	0,7
64	9	0,7
64	10	0,7

Nn	Mm	Rr
64	1	0,8
64	2	0,8
64	3	0,8
64	4	0,8
64	5	0,8
64	6	0,8
64	7	0,8
64	8	0,8
64	9	0,8
64	10	0,8

Nn	Mm	Rr
64	1	1,0
64	2	1,0
64	3	1,0
64	4	1,0
64	5	1,0
64	6	1,0
64	7	1,0
64	8	1,0
64	9	1,0
64	10	1,0

Nn	Mm	Rr
8	2	0,7
8	3	0,7
8	4	0,7
8	5	0,7
8	6	0,7

Nn	Mm	Rr
8	2	0,8
8	3	0,8
8	4	0,8
8	5	0,8
8	6	0,8

Nn	Mm	Rr
8	2	0,9
8	3	0,9
8	4	0,9
8	5	0,9
8	6	0,9

Nn	Mm	Rr
16	2	0,7
16	3	0,7
16	4	0,7
16	5	0,7
16	6	0,7

Nn	Mm	Rr
16	2	0,8
16	3	0,8
16	4	0,8
16	5	0,8
16	6	0,8

Nn	Mm	Rr
16	2	1,0
16	3	1,0
16	4	1,0
16	5	1,0
16	6	1,0

Nn	Mm	Rr
32	2	0,7
32	3	0,7
32	4	0,7
32	5	0,7
32	6	0,7

Nn	Mm	Rr
32	2	0,8
32	3	0,8
32	4	0,8
32	5	0,8
32	6	0,8

Nn	Mm	Rr
32	2	1,0
32	3	1,0
32	4	1,0
32	5	1,0
32	6	1,0

Nn	Mm	Rr
8	1	1,0
8	2	1,0
8	3	1,0
8	4	1,0
8	5	1,0
8	6	1,0

Tabela 3. 1 - Relação de simulações com filas sem prioridade

FILAS NA ENTRADA			FILAS NA SAÍDA		
Nn	Mm	Rr	Nn	Mm	Rr
16	2	0,7	16	2	0,8
16	3	0,7	16	3	0,8
16	4	0,7	16	4	0,8
16	5	0,7	16	5	0,8
16	6	0,7	16	6	0,8

Tabela 3. 2 - Relação de simulações sem filas com prioridade.

Para garantir a confiabilidade dos dados gerados, o tempo de simulação de cada modelo foi variado de dezenas de minutos, até vários dias, dependendo da ordem de grandeza dos dados a serem coletados. Os dados só eram considerados válidos quando variavam de, no máximo ± 5% do valor médio em mais de 95% do tempo de observação. Este grande volume de dados coletados nas simulações geraram os gráficos mostrados no capítulo 4. Os diversos modelos analisados seguem a seguinte nomenclatura: Modelo **NnMmRr** , onde:

Nn = Número de entradas/saídas;

Mm = Número de *links* internos para cada porta de saída;

Rr = Carga oferecida.

3.2. FUNCIONAMENTO BÁSICO DE UM MODELO NnMmRr

3.2.1. MODELO DO COMUTADOR TIPO FIFO

As Figuras 3.1, 3.2 e 3.3, a seguir, mostram uma representação, em diagrama de blocos do modelo N8M2R9, o qual servirá de referência para se explicar o funcionamento básico para todos os modelos com células sem prioridade (FIFO).

Os blocos *Create* (1) a (8) são os responsáveis por gerar as células de entrada do comutador. Os mesmos foram programados para gerarem uma célula de cada vez a um determinado espaço de tempo, neste caso, uma célula por segundo. O tempo entre as gerações é irrelevante, pois se trabalhou, nestes modelos, com o conceito de ciclos. Cada ciclo inicia-se com a geração da primeira célula e termina quando a última célula útil gerada é atendida. Todos os *Create* geram, em seqüência, apenas uma célula a cada ciclo.

Após os módulos *Create*, as células entram nos módulos *Decide* (28) a (35). Estes módulos são responsáveis por estabelecer a carga de cada entrada. Este módulo é uma

estrutura de decisão baseada em probabilidade. Para estabelecer uma carga de 90%, por exemplo, define-se que, estatisticamente, 90% das células geradas em cada entrada seguirão efetivamente para o comutador, e as outras 10% serão descartadas, sendo que as probabilidades são uniformemente distribuídas.

Em seguida, as células passam pelos módulos *Assign* (1) a (8), os quais atribuem dois valores de atributos a cada célula. Atributos serão valores e ou nomes que acompanham cada célula durante todo o processo de simulação. Estes atributos serão usados mais adiante em outras estruturas de decisão. Estes atributos são:

- **Atributo IN**: Atribuído de acordo com a respectiva entrada: IN=1 para a entrada 1, até IN=N para a entrada N;
- **Atributo OUT**: Atribuído segundo uma distribuição discreta de probabilidades uniformemente distribuídas: OUT=1 para a saída 1, até OUT= N para a saída N.

O atributo OUT indica para qual saída do comutador a célula será encaminhada. Neste caso, os valores tanto de IN quanto de OUT variam entre 1 e 8, pois existem 8 entradas-saídas.

Ao sair dos módulos *Assign* as células entram nos módulos *Hold* (1) a (8). Estes módulos são responsáveis por gerar as estatísticas do programa. Neste caso específico, apenas o comprimento médio da fila. Este módulo funciona da seguinte maneira: ele segura (*hold*) as células até que um sinal lhe seja enviado. Somente após receber este sinal é que a primeira célula da fila é liberada. Os sinais para os respectivos *Hold* são gerados pelos módulos *Signal*, obedecendo a um algoritmo *Round Robin* implementado adiante.

Após sair dos módulos *Hold*, as células são encaminhadas pelo *Decide* (1) para as respectivas saídas. O encaminhamento é feito pela análise do atributo OUT de cada célula. As células que possuem atributo OUT=N são encaminhadas para saída N. Em seguida, as células passam pelo conjunto formado por *Assign-Decide-Assign* que é o conjunto responsável por decidir se a célula segue ou não para o módulo *Process*, o que significa que a célula vai ser transferida para a respectiva saída. Este conjunto utiliza duas variáveis (a_N e m_N) que são incrementadas e decrementadas à medida que as células são encaminhadas para o módulo *Process*. Os módulos *Assign* (9) a (16) incrementam as variáveis a_N e os módulos *Assign* (17) a (24) as decrementam. Da mesma forma, as variáveis m_N são decrementadas pelos *Assign* (25) a (32) e incrementadas pelos *Assign* (73) a (80). O

controle é feito comparando os valores destas duas variáveis. Estes contadores só deixam passar, no máximo, *m* células para cada saída em cada ciclo, pois *m* é o número de *links* internos de cada saída e indica o número máximo de células que podem ser transferidas para cada saída em cada ciclo. As outras células que por acaso ultrapassarem o número *m* são encaminhadas de volta para as suas respectivas entradas, através do bloco *Decide* (10) que analisa o atributo IN de cada célula.

Portanto, antes das células retornarem às respectivas entradas, elas passam pelos módulos *Process* (81) a (88), mostrados na Figura 3.2, os quais são os responsáveis por indicar, através de variáveis, de qual entrada pertence a primeira célula bloqueada, para a implementação do algoritmo *Round Robin*, que garantirá que a primeira célula bloqueada em um ciclo será a primeira célula a ser atendida no próximo ciclo.

Ao serem encaminhadas para os blocos *Process* (81) a (88), as células passam pelo bloco *Decide* (19) interno que verificará a variável chamada *block*. Caso esta variável seja diferente de zero, significa que alguma célula já retornou, ou seja, não foi atendida e o próximo ciclo começará por aquela respectiva entrada. Neste caso, esta célula vai diretamente para o bloco *Assign* (60) que incrementará uma variável denominada *ret*, indicando o número de células que retornaram, para posterior cálculo de probabilidade de bloqueio. Caso a variável *block* seja zero, significa que nenhuma célula retornou até aquele momento. Então, a célula passa ao *Assign* (43) que fará a variável *block* igual a 1 e em seguida altera uma variável denominada *v1* no bloco *Assign* (44). A variável *v*1 indica qual a entrada a ser atendida primeiramente no próximo ciclo. Como a variável *block* foi alterada para 1, todas a células a partir deste momento são desviadas, impedindo de alterar a variável *v*1. Em seguida, esta célula, também passa pelo bloco *Assign* (60) para incrementar a variável *ret*.

A implementação efetiva do algoritmo *Round Robin* é feita no módulo *Process* (100), mostrado na Figura 3.3. Neste módulo é feita a ordenação dos módulos *Signal*, os quais são responsáveis por enviar os sinais, na seqüência correta, aos módulos *Hold* para a liberação das células. A implementação é feita da seguinte maneira: a primeira célula que retorna altera o valor da variável de controle *v*1, que é a responsável por indicar por onde começa a próxima seqüência de “disparo” dos módulos *Hold*, através dos módulos *Signal*. A alteração da variável de controle é feita de modo que, mesmo se houver mais de uma

célula retornando por ciclo, apenas a primeira célula é capaz de alterar o valor da mesma, garantindo assim que o atendimento do próximo ciclo realmente se inicie pela célula correta. Isto é feito nos módulos *Process* (81) a (88) já mostrados. O módulo *Signal* (9), mostrado na Figura 3.3, envia um sinal de valor igual ao da variável *v*1, que pode variar, de modo discreto, entre 1 e 8 (neste caso específico de 8 entradas-saídas), para o módulo *Hold* (*v*1). Em seguida, dependendo do valor desta variável, haverá uma sequência de disparo diferente para os demais *Hold*, determinada pelo *Decide* (27). Como neste exemplo tem-se 8 entradas distintas, tem-se 8 linhas diferentes de sequência de disparo.

As células que retornam recebem um atributo (K_n), o qual garante que as mesmas sejam realmente encaminhadas para o início das respectivas filas de entrada. Este controle é feito no módulo *Hold*, onde se pode configurar a fila por ordem de atributo, neste caso o atributo configurado foi K_n.

Finalmente, após saírem dos módulos *Process* (atendimento), as células passam por módulos *Assign* para que as variáveis retornem aos seus valores iniciais. Em seguida, passam pelo módulo *Assign* (81) o qual, através de uma equação matemática que divide o número de células que retornaram à entrada, indicados pela variável *ret*, pelo número de células válidas geradas, determinam a porcentagem de células "bloqueadas". Como este cálculo é realizado com um número muito grande de células, pode-se aproximar este número para a probabilidade de bloqueio do comutador.

Todos os blocos *Dispose*, mostrados em todas as Figuras, servem como destino final das células, sejam elas atendidas ou descartadas, assim como todos os blocos *Delay*, mostrados em todas as Figuras, servem para provocar um pequeno atraso nas células, visando o perfeito funcionamento dos modelos.

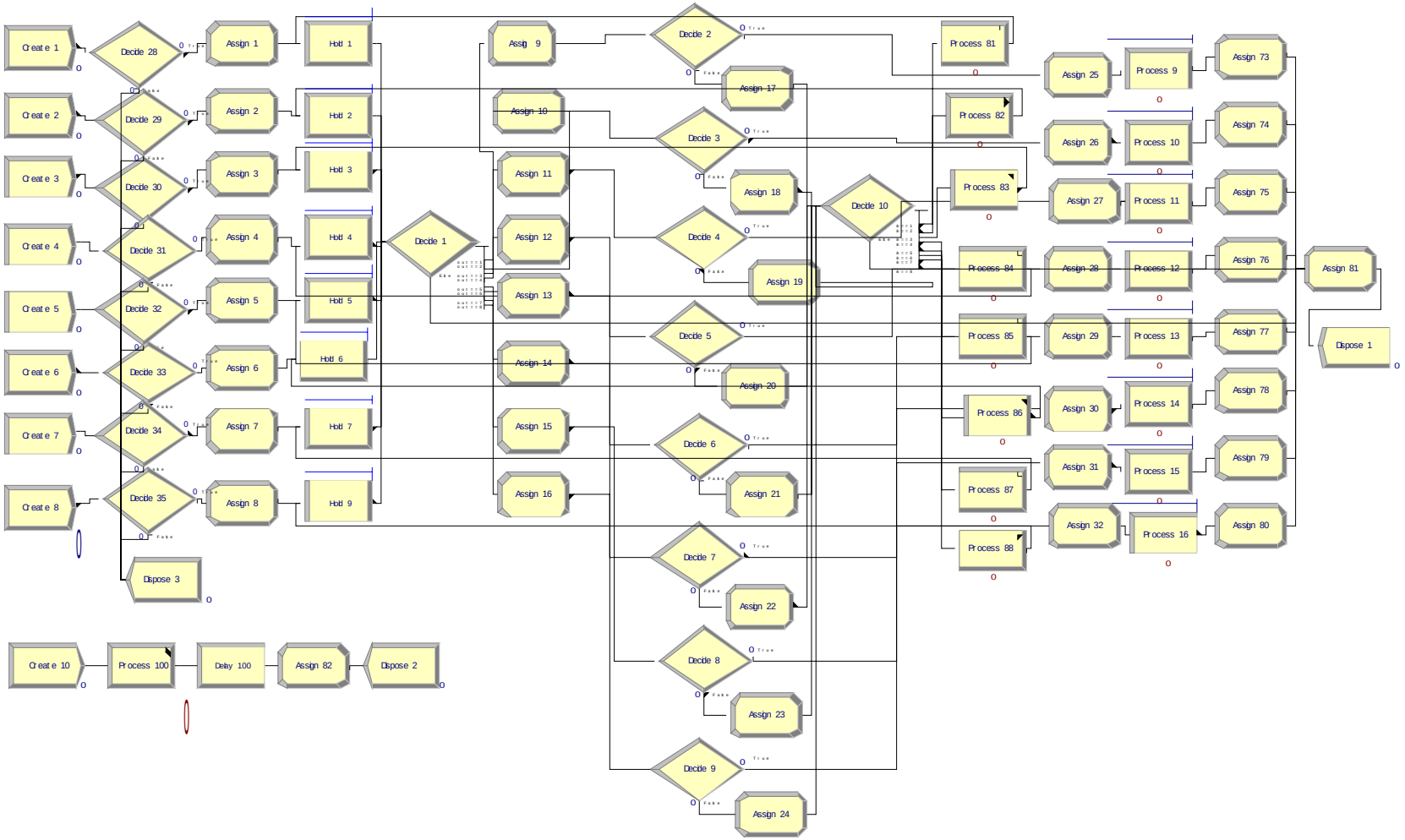

Figura 3.1 – Diagrama em blocos do modelo N8M2R09.

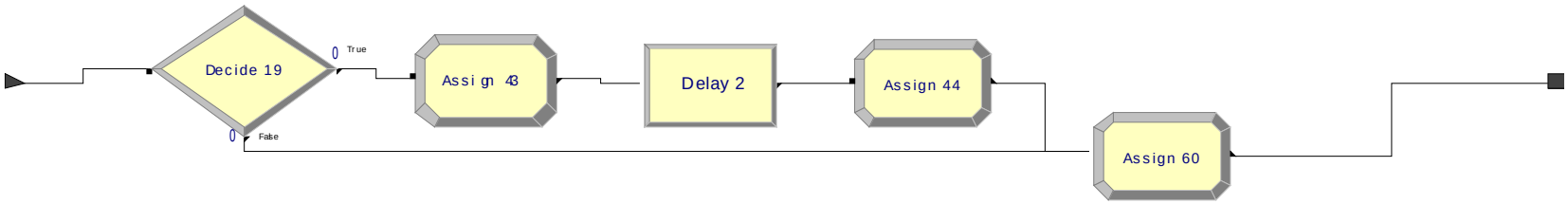

Figura 3.2 – Estrutura Interna dos Blocos *Process* (81)-(88).

Figura 3.3 – Estrutura Interna do Bloco *Process* (100).

3.2.2. MODELO DO COMUTADOR COM DISCIPLINA DE PRIORIDADES

A Figura 3.4 mostra apenas o fluxograma de entrada da estrutura com células com classe de serviço.

Neste modelo, conforme Figura 3.4, vê-se que, após as células serem geradas pelos módulos *Create*, elas passam pelo módulo *Decide* que determinará, assim como no modelo FIFO já visto, a carga oferecida em cada entrada. Em seguida, as células passam pelo módulo *Assign* que definirá 3 atributos para cada célula. Os atributos IN e OUT vistos anteriormente e, agora, também o atributo PR, que definirá a prioridade, ou classe de serviço de cada célula. Neste caso, as classes foram configuradas na seguinte proporção: 40%, 20%, 20%, 10% e 10%, representando, respectivamente, as classes de serviço CBR, rtVBR, nrtVBR, ABR e UBR. Após o *Assign*, tem-se os módulos *Decide*, os quais decidirão, através do atributo PR, o encaminhamento de cada célula. Cada saída de cada *Decide* está ligada a um *Hold* diferente. Então, para N entradas tem-se Nxh *Hold's*, onde N é o número de entradas e *h* é o número de classes de serviço. Neste exemplo, N=16 e *h*=5, implicando em 80 *Hold's* de entrada diferentes. Cada entrada possui 5 módulos *Hold*, um para cada classe diferente.

A ordem de "disparo" destes *Hold's* segue o mesmo funcionamento descrito anteriormente, mas, começando sempre pelo *Hold* que representa a classe de maior prioridade da entrada designada pelo algoritmo *Round Robin*. Na Figura 3.4, o *Hold* (1) da entrada 1 tem prioridade 1 (classe CBR), que é a mesma prioridade do *Hold* (16) da entrada 16, e assim sucessivamente até o *Hold* (65) da entrada 1, que tem prioridade 5 (classe UBR), a qual é igual ao *Hold* (80) da entrada 16.

O *Process*o de atendimento a partir dos *Hold's* é similar ao descrito anteriormente para a arquitetura sem prioridade.

Para medir a fila de saída foi montada, na saída, uma estrutura quase idêntica à de entrada, conforme mostra a Figura 3.5. Após serem atendidas, as células são encaminhadas através do módulo *Decide* (80) para os respectivos *Hold's* de acordo com a classe de prioridade. Os *Hold's* são então "disparados" na sequência determinada, começando sempre pela porta de saída que teve a primeira célula bloqueada no ciclo anterior. Como apenas uma célula de cada saída pode ser transferida em cada ciclo, o módulo Assign (84)

incrementa uma variável toda vez que uma célula passa por ele. No módulo *Decide* (86) é feita uma comparação: se a variável for menor ou igual a um, a célula passa; caso contrário, a célula volta para o respectivo *Hold* através do *Decide* (80), passando antes pelo módulo *Process* (821).

O módulo *Process* (821) tem a mesma função descrita anteriormente para os módulos *Process* (81)-(88). Já o *Assign* (85) é responsável por retornar as variáveis aos valores iniciais.

Após toda esta sequência, o ciclo se inicia novamente.

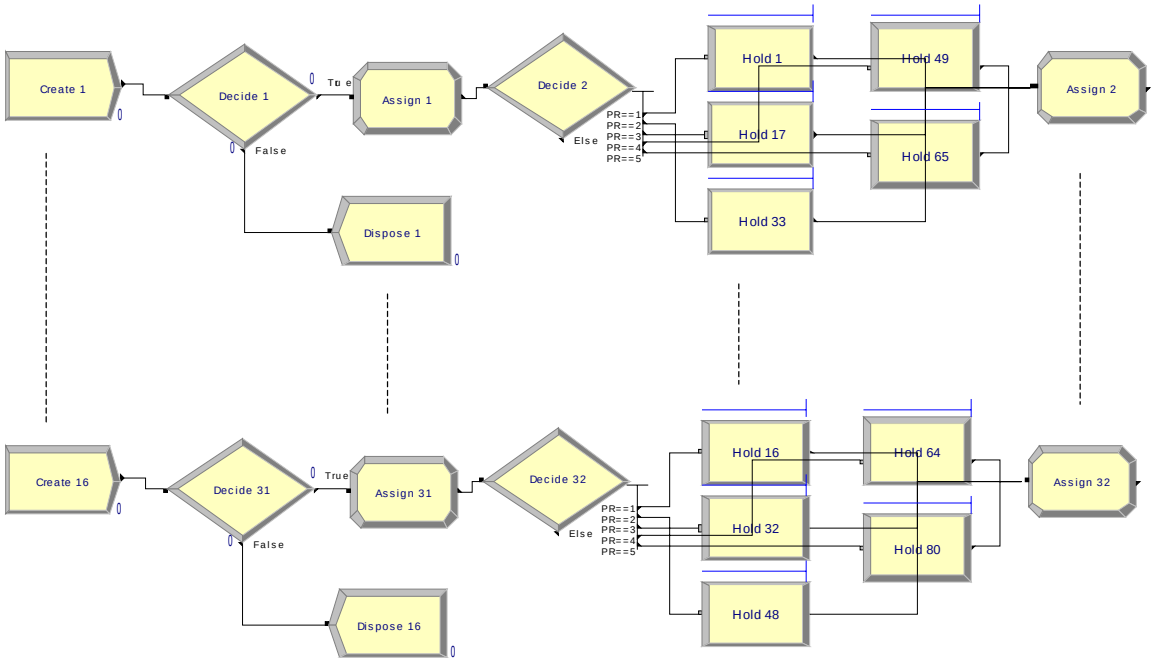

Figura 3.4 – Blocos de Entrada do Comutador com Disciplina de Prioridade.

A estrutura mostrada na Figura 3.5 é repetida para cada saída. Neste caso, 16 vezes.

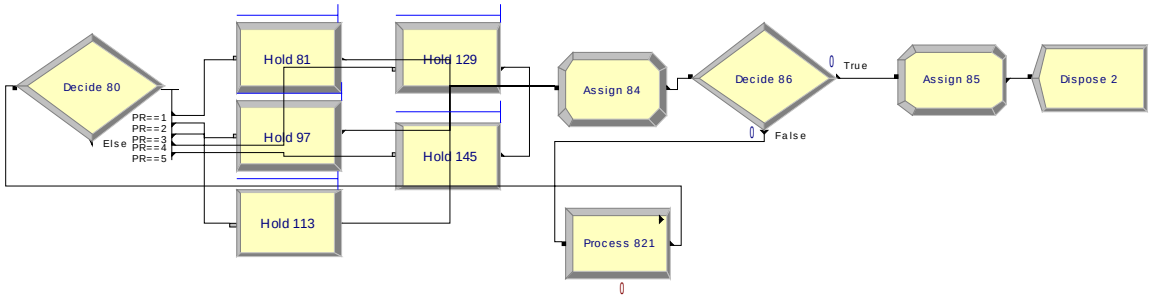

Figura 3.5 – Blocos de Saída do Comutador com Disciplina de Prioridade.

CAPÍTULO 4

4 - RESULTADOS E DISCUSSÕES

4.1. INTRODUÇÃO

Neste capítulo são mostrados, através de gráficos, os resultados obtidos nas diversas simulações realizadas. É realizada também uma comparação dos resultados obtidos nesta dissertação com os resultados obtidos em [1] e [2] a fim de se analisar o modelo de simulação do comutador ATM CIOQ implementado.

4.2. FILAS SEM PRIORIDADE

A Figura 4.1.a mostra a probabilidade de bloqueio, determinada através de simulações no ARENA para uma carga de entrada $\rho = 0.9$ em função do número de *links* internos (*m*) para os diversos tamanhos de matriz de comutação (N).

Pode-se observar que no pior caso, ou seja, quando o número de *links* internos é igual a um, o sistema se comporta como um comutador simples com *buffer* FIFO na entrada. Sendo assim, não é necessária a utilização dos *m buffers* na saída. Neste caso, a probabilidade de bloqueio é relativamente alta, aproximadamente 40% para todos os valores de N, ou seja, existe uma convergência da probabilidade de bloqueio independente do número de entradas-saídas. Verifica-se também, como era de se esperar, que a probabilidade de bloqueio cai rapidamente, à medida que se aumenta o número de *links* internos. Por exemplo: ao se comparar a probabilidade de bloqueio de uma matriz 16x16 para *m*=1 e *m*=4, tem-se, respectivamente, os valores 3,95E-01 e 1,81E-03. Ou seja, existe um ganho de 218 vezes do comutador com 4 *links* internos em relação a um comutador com apenas 1 *link* interno (comutador tradicional). Para um comutador 32x32 este ganho é de 167 vezes e para um comutador 64x64 este ganho relativo passa a ser de 149 vezes.

Na Figura 4.1b, retirada de [1], tem-se o mesmo conjunto de curvas da Figura 4.1.a, ou seja, a probabilidade de bloqueio para uma carga de entrada $\rho = 0.9$ em função do número de *links* internos (*m*) para os diversos tamanhos de matriz de comutação (N). Esta Figura mostra também a curva para um matriz 128x128, a qual não foi simulada no ARENA, devido a dificuldade de modelamento de um comutador com estas proporções

utilizando-se os modelos básicos desenvolvidos. As curvas da Figura 4.1.b foram obtidas através de resultados analíticos em [1] e comprovados através de simulações no ARENA neste trabalho, visto que pode-se observar a grande semelhança entre as duas Figuras.

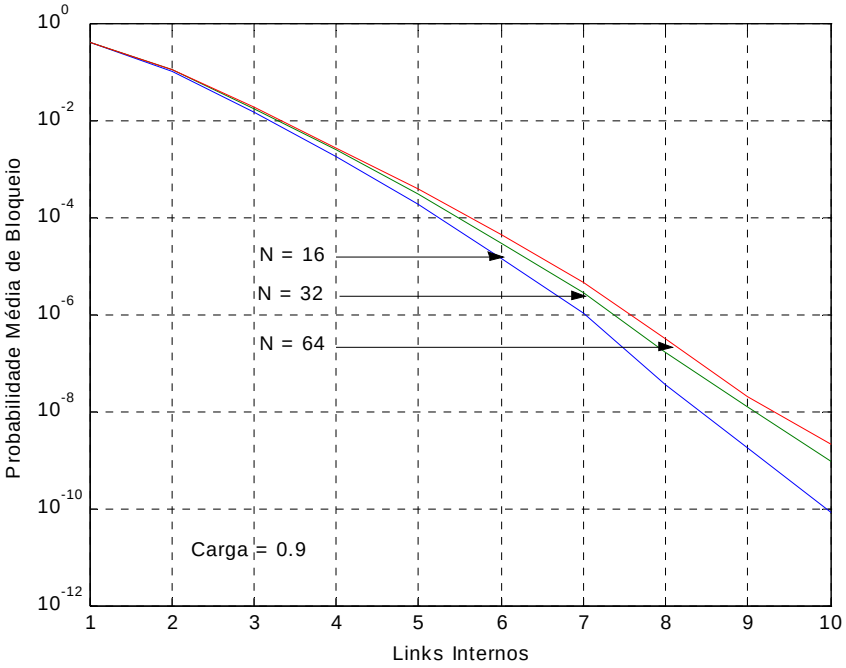

Figura 4.1.a – Probabilidade de bloqueio x *links* internos.

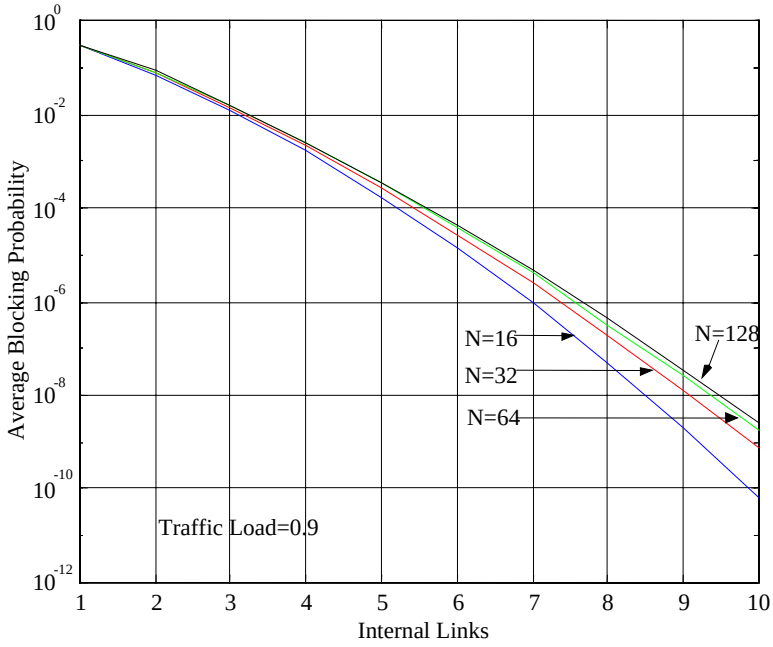

Figura 4.1.b – Probabilidade de bloqueio x *links* internos [1].

Na Figura 4.2.a, procurou-se analisar o comportamento do comutador em termos de probabilidade média de bloqueio em função da variação da carga oferecida (ρ) e do número de *links* internos (*m*). Neste caso específico, foi utilizado um comutador de tamanho 64x64 e variou-se a carga entre 70 e 100% para *links* internos entre 1 e 8. Este gráfico mostra que os valores de probabilidade não tiveram um acréscimo considerável mesmo quando se utilizou uma carga de 100%. Na realidade, o que se observa é que a probabilidade de bloqueio, em média, tende a ter o seu valor dobrado a cada aumento de 0,1 na carga para *m* entre 2 e 10, quando se varia a carga oferecida e se mantêm os demais parâmetros. Estes resultados tiveram o mesmo comportamento para diferentes valores de N. Os resultados deste gráfico foram obtidos através de simulações no ARENA.

A Figura 4.2.b mostra os resultados para os mesmos parâmetros obtidos analiticamente em [1]. Os resultados obtidos em ambos os casos foram muito próximos, como pode ser observado a seguir.

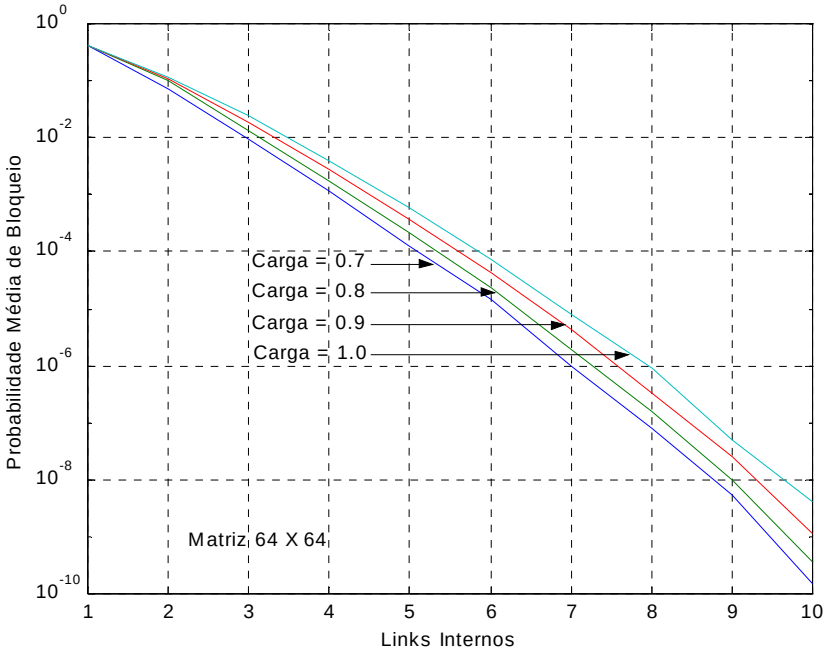

Figura 4.2.a – Probabilidade de bloqueio para diversas cargas.

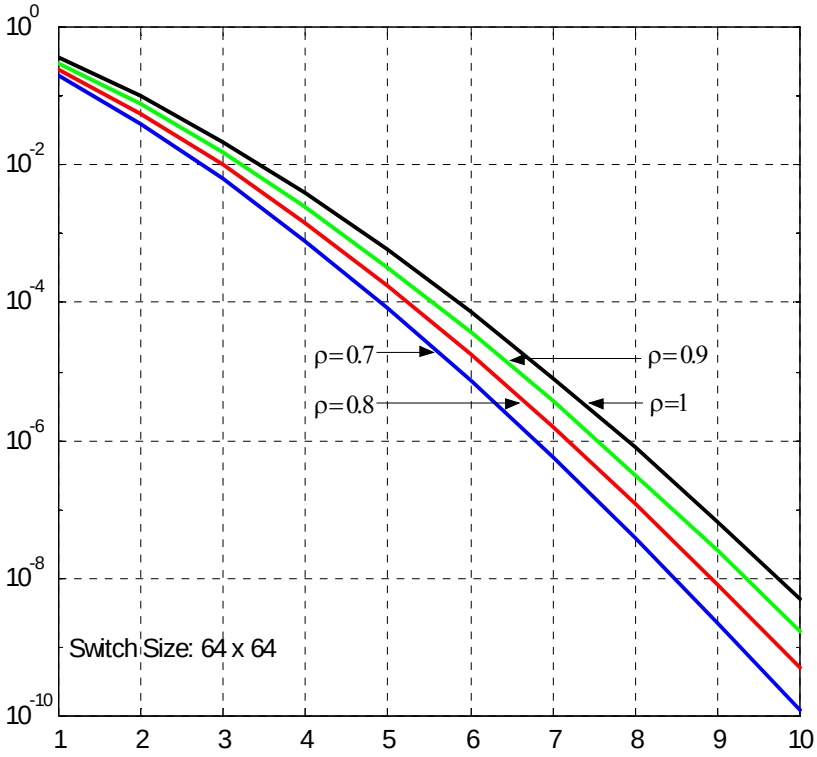

Figura 4.2.b – Probabilidade de bloqueio para diversas cargas [1].

Na Figura 4.3.a é mostrada a probabilidade de bloqueio em um comutador de matriz 8x8 com carga igual a 100%. Verificou-se neste caso uma probabilidade de bloqueio de pouco mais de 38% para m=1, que é o pior caso, visto que o comutador se comporta como um comutador com fila simples de entrada apresentando problemas de HOLB. Observa-se aqui, novamente, que a probabilidade de bloqueio cai rapidamente com o aumento dos *links* internos.

A Figura 4.3.b mostra uma comparação entre os resultados analíticos e os resultados simulados usando o *software* Matlab, realizados em [1]. Neste caso, os resultados obtidos de forma analítica, ou com o Matlab, ou mesmo com o ARENA, foram praticamente os mesmos.

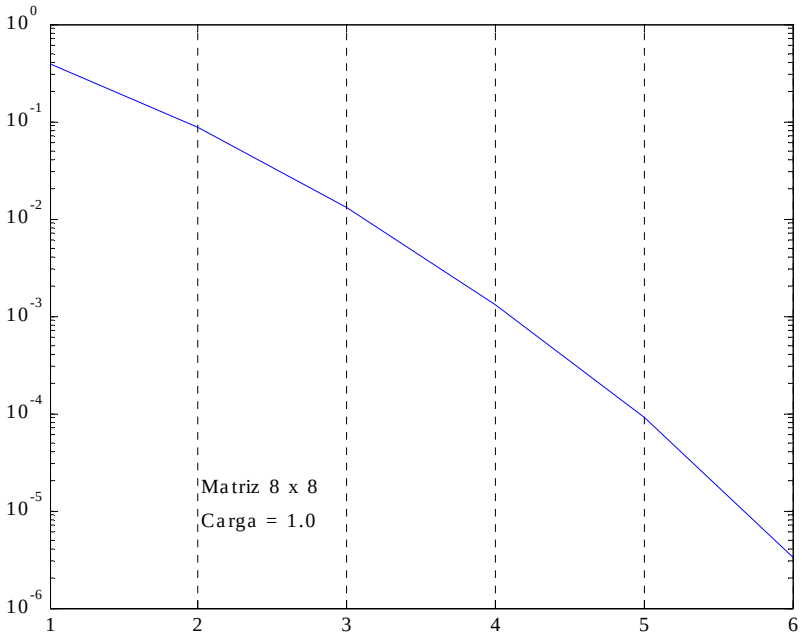

Figura 4.3.a – Probabilidade de bloqueio para em um switch 8X8 com carga =1.

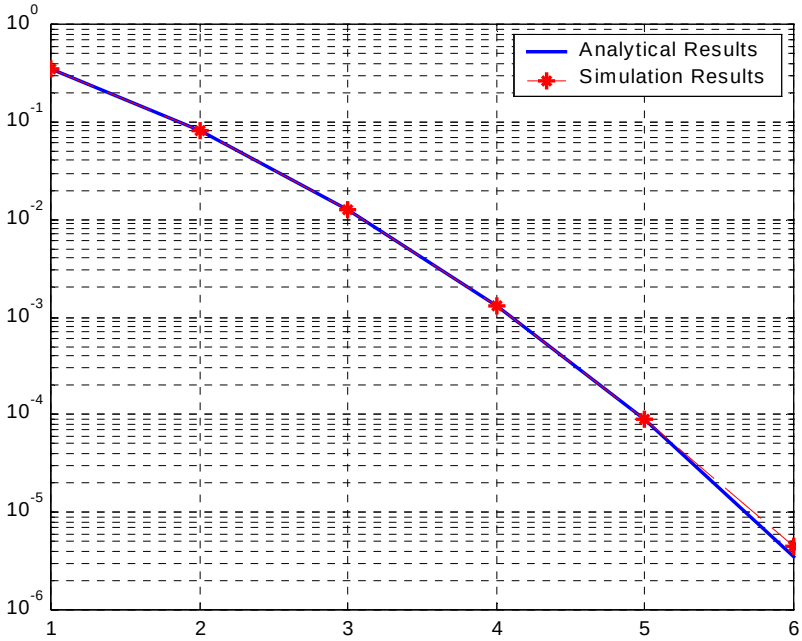

Figura 4.3.b – Resultado analítico x resultado simulado [1].

O comprimento médio da fila de entrada para um sistema no qual se tem uma carga de 90% com diversos tamanhos de matriz e também diversos valores de *links* internos pode ser visualizado na Figura 4.4.a. No pior caso, ou seja, para um comutador 64x64 e com número de *links* internos igual a dois (*m*=2), o comprimento médio da fila é menor que 5 células e este número reduz-se drasticamente quando se aumenta o número de *links* internos. Por exemplo, o comprimento médio da fila para a maior matriz, ou seja, N=64, é de menos de uma célula (0,192) quando se utiliza *m*=3 e aproximadamente 0,025 para *m*=4.

Mais uma vez, os resultados obtidos através de simulações com o ARENA foram bastante próximos dos resultados obtidos nas simulações com o Matlab [1], os quais são mostrados na Figura 4.4.b.

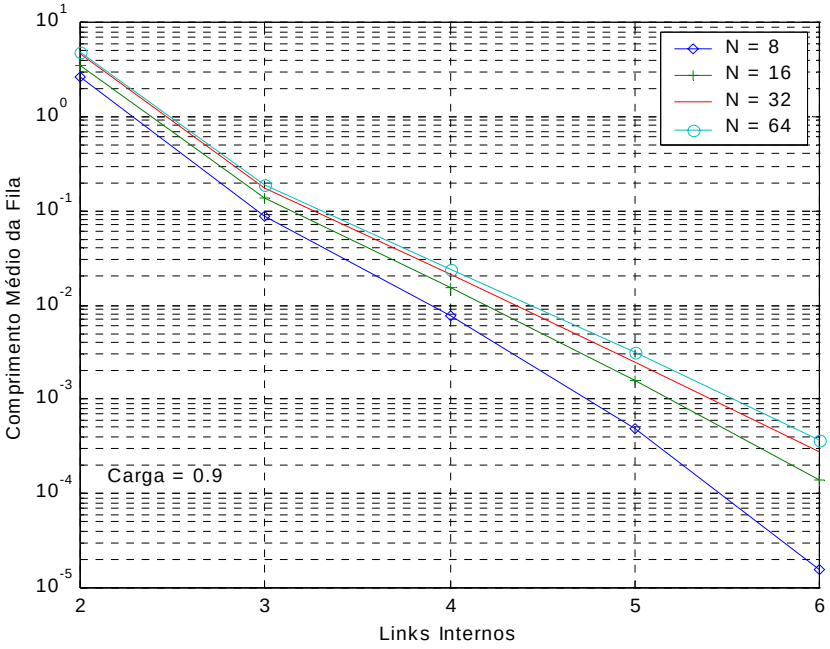

Figura 4.4.a - Comprimento médio da fila de entrada em função de *links* internos.

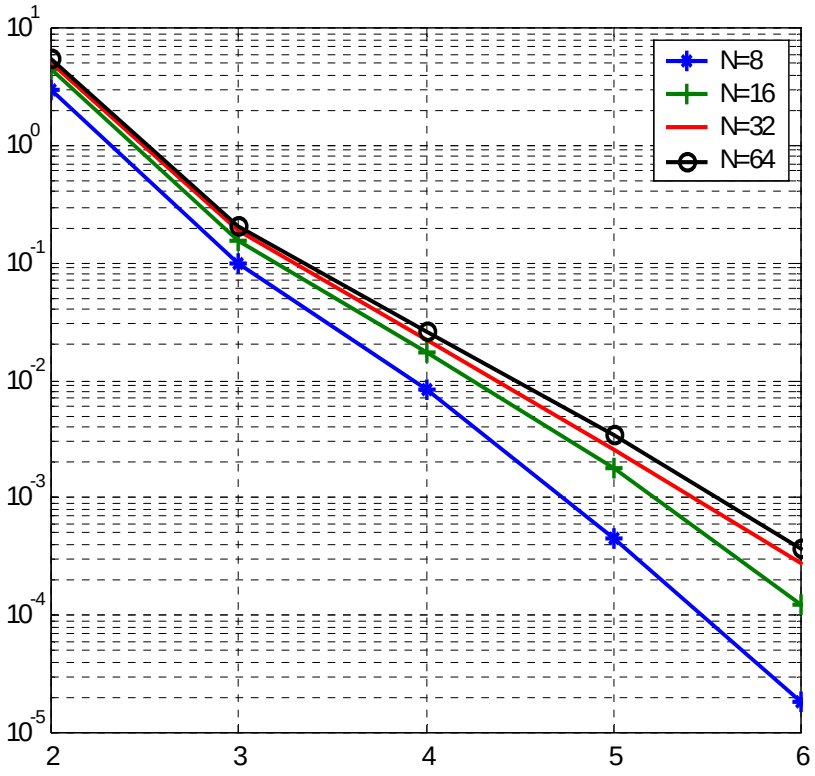

Figura 4.4.b - Comprimento médio da fila de entrada em função de *links* internos[1].

Os gráficos e as simulações mostradas até aqui visaram a análise dos comutadores CIOQ, a comparação dos resultados obtidos em [1], [2] e [3] e a validação dos modelos utilizados. A seguir serão mostrados alguns resultados complementares que poderão ser bastante úteis na análise de desempenho dos comutadores ATM do tipo CIOQ.

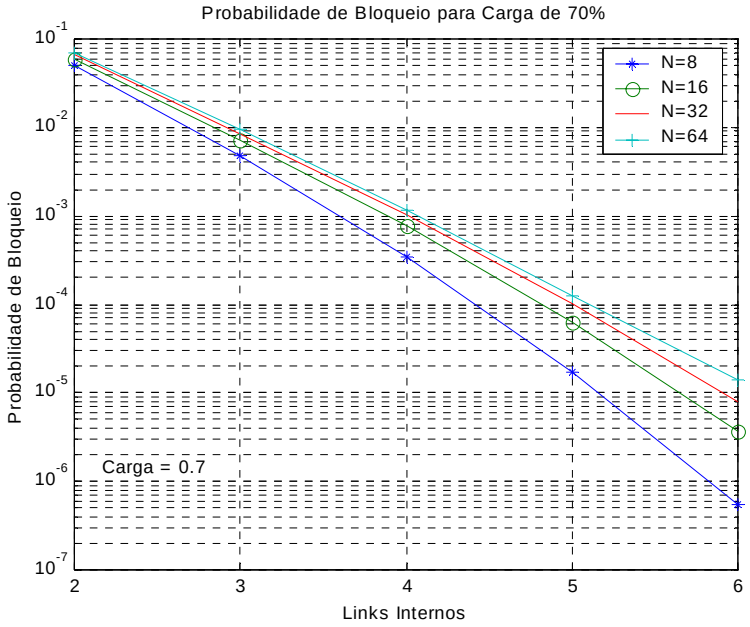

Figura 4.5 – Probabilidade de Bloqueio para Carga de 70%.

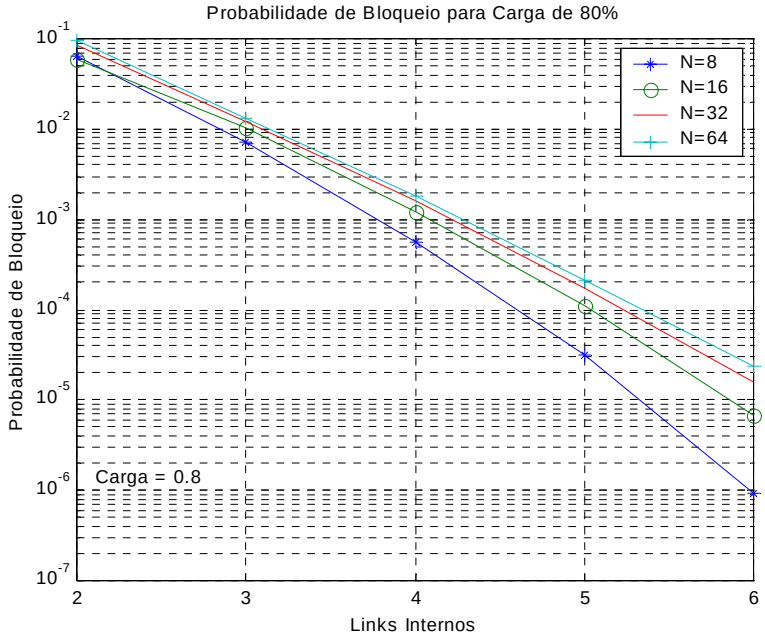

Figura 4.6 – Probabilidade de Bloqueio para Carga de 80%.

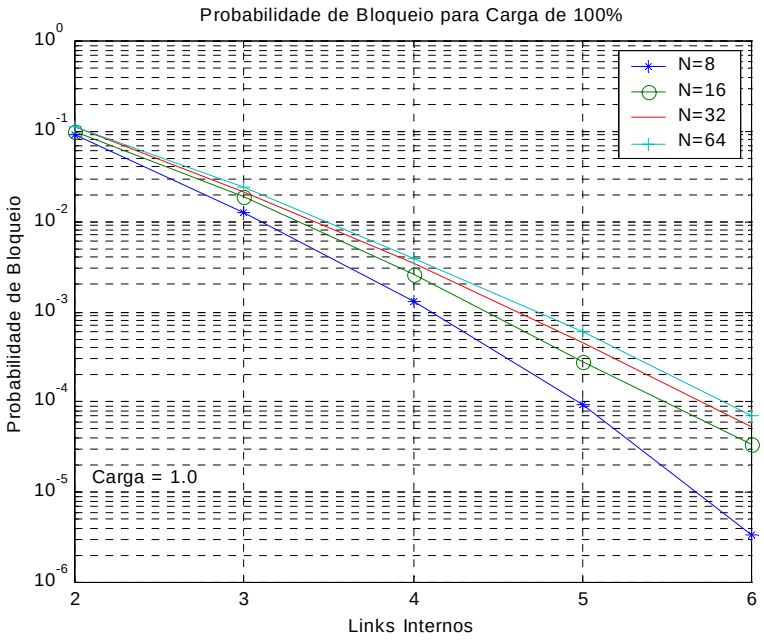

Figura 4.7 – Probabilidade de Bloqueio para Carga de 100%.

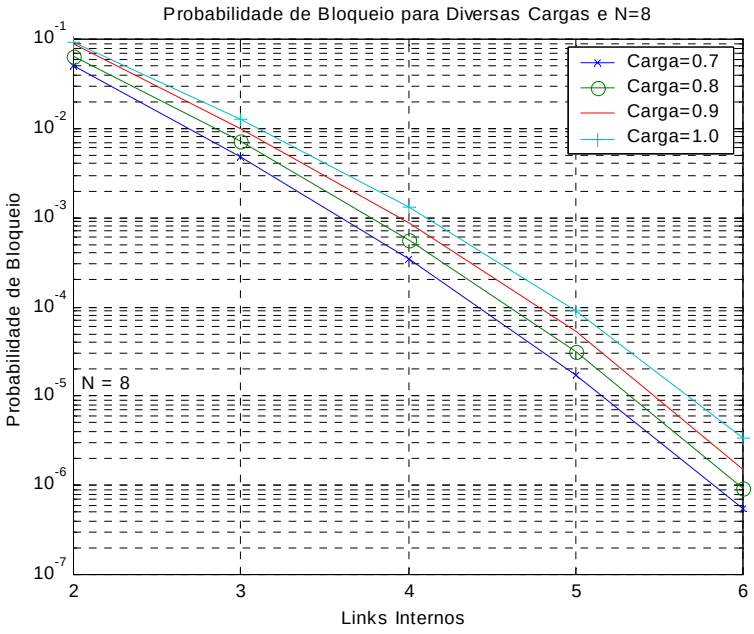

Figura 4.8 – Probabilidade de Bloqueio para Diversas Cargas e N=8.

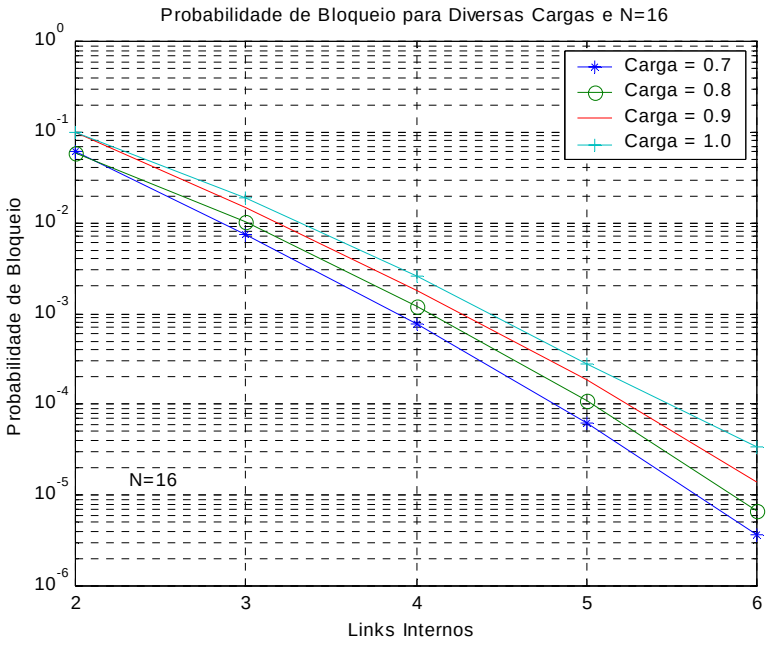

Figura 4.9 – Probabilidade de Bloqueio para Diversas Cargas e N=16.

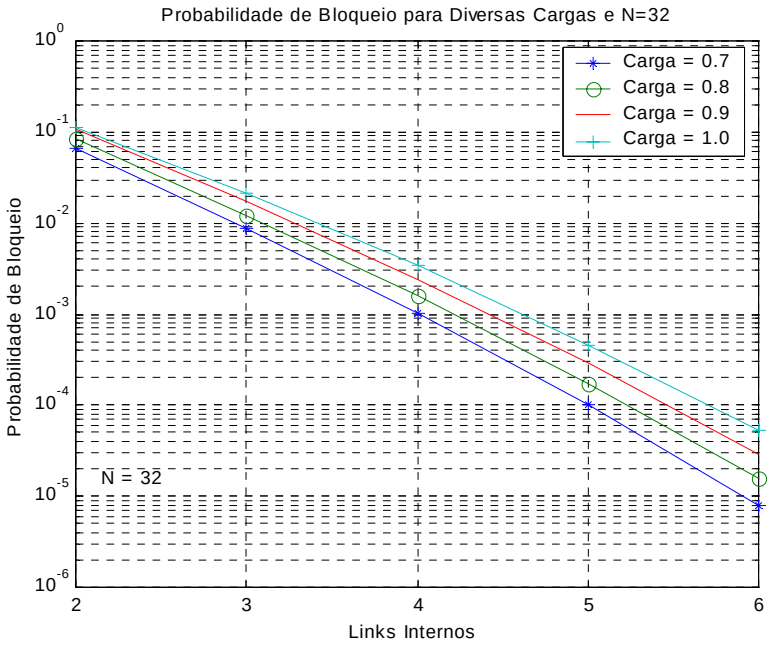

Figura 4.10 – Probabilidade de Bloqueio para Diversas Cargas e N=32.

Na Tabela 4.1 são mostrados os comprimentos máximos das filas para os diversos tamanhos de comutadores (N) com carga de 90% e número de links internos (*m*) variando entre 3 e 6. Estes resultados são úteis para dimensionamento dos *buffers* de entrada do comutador.

Pode-se verificar que no pior caso avaliado, ou seja, número de *links* internos igual a 3 (*m*=3) e matriz de comutação 64x64 (N=64), o comprimento do *buffer* é menor do que 10.

m \ N	8	16	32	64
3	5	7	7	9
4	3	3	4	4
5	2	2	2	3
6	2	2	2	2

Tabela 4. 1 - Comprimento máximo das filas de entrada para carga de 90%.

4.3. FILAS COM PRIORIDADE

Para análise do comportamento das filas, tanto na entrada quanto na saída, foi utilizado um comutador 16x16 com carga de 90% e 5 classes de serviço diferentes. As classes de serviço foram divididas nas proporções de 40%, 20%, 20%, 10% e 10%, respectivamente às classes 1, 2, 3, 4 e 5.

4.3.1. FILAS NA ENTRADA

Na Figura 4.11.a, mostrada a seguir, verifica-se que o comprimento médio das filas de entrada para todas as classes de serviço é pequeno. Tem-se, por exemplo, para o pior caso, que é a classe de serviço 5 (classe menos prioritária) com número de *links* internos igual a 2 (m = 2), menos de uma célula, em média, na fila; enquanto que para a fila de maior prioridade este número cai para pouco mais de 0,02. Mais uma vez, o tamanho médio da fila cai rapidamente com o aumento do número de *links* internos.

Na Figura 41.b tem-se o mesmo conjunto de curvas, desta vez retiradas de [2].

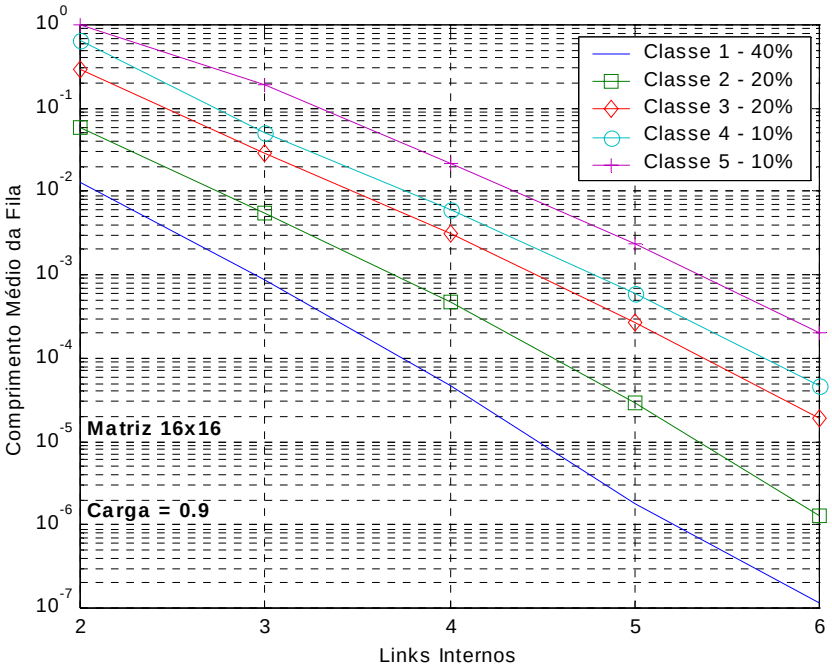

Figura 4.11.a - Comprimento médio da fila de entrada para cada classe de serviço.

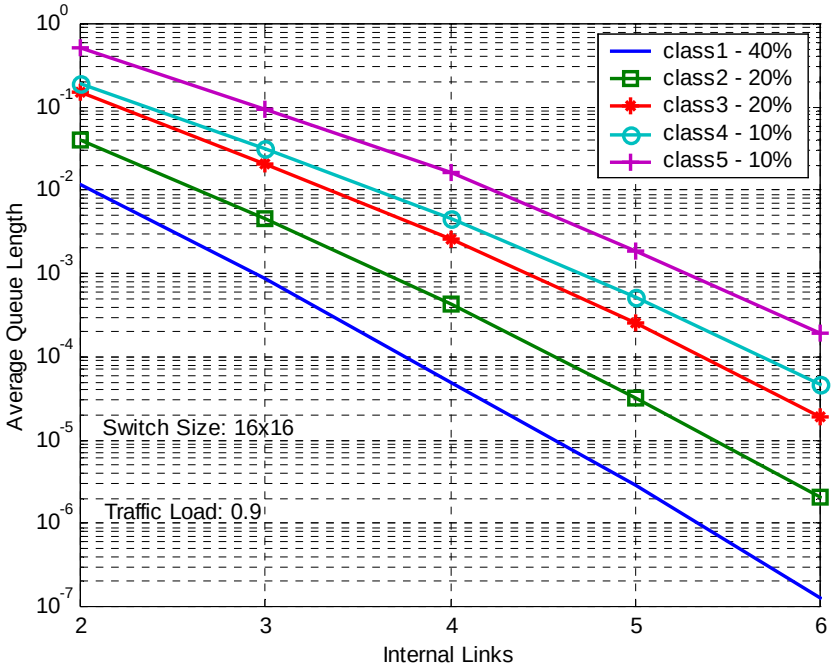

Figura 4.11.b - Comprimento médio da fila de entrada para cada classe de serviço[2].

4.3.2. FILAS NA SAÍDA

Observa-se na Figura 4.12.a que o número médio de células nas diversas filas de saída é pequeno. No pior caso, *m*=6 e classe de prioridade 5, tem-se, em média, menos de três células. Para as outras classes, mesmo para *m*=6, existe menos de uma célula, em média, em cada fila. Para a fila de maior prioridade o número médio de células na fila é de aproximadamente 0,2.

Nota-se, também, como era de se esperar, que o tamanho das filas de saída tendem a crescer com o aumento do número de *links* internos, pois, com o aumento do número de *links*, aumenta-se a probabilidade de transferência de células em um mesmo *slot* de tempo. Este crescimento, porém, é bastante pequeno. Basta verificar no gráfico a variação entre *m*=2 e *m*=6 para qualquer classe de prioridade.

Na Figura 4.12.b são mostrados os resultados obtidos em [2] com simulações realizadas utilizando-se o Matlab. Novamente, é clara a semelhança entre os resultados.

Na Figura 4.13 tem-se o comprimento médio acumulado para cada porta de saída, ou seja, o comprimento médio das filas de saída somando-se o comprimento médio da fila de todas as classes.

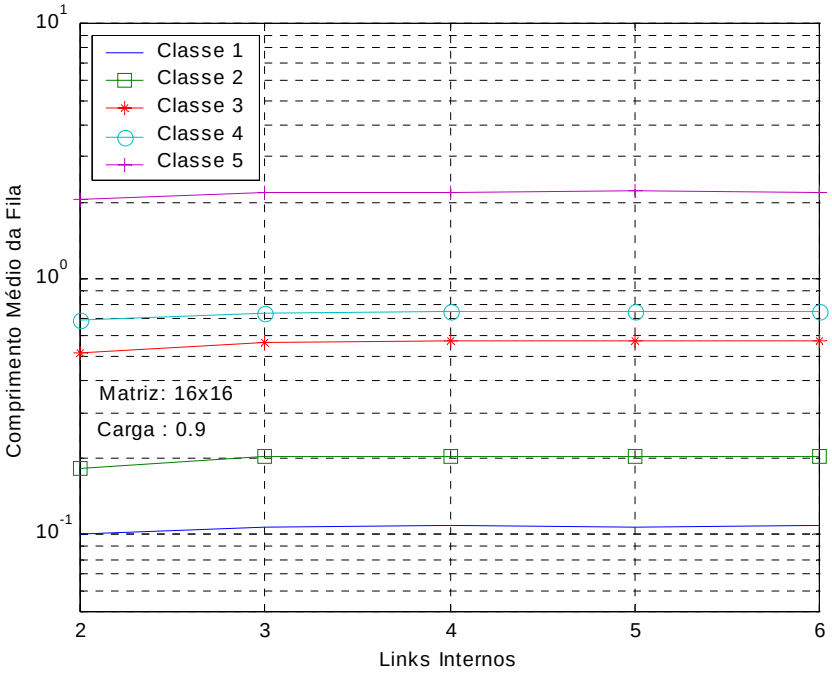

Figura 4.12.a - Comprimento médio da fila na saída para cada classe de serviço.

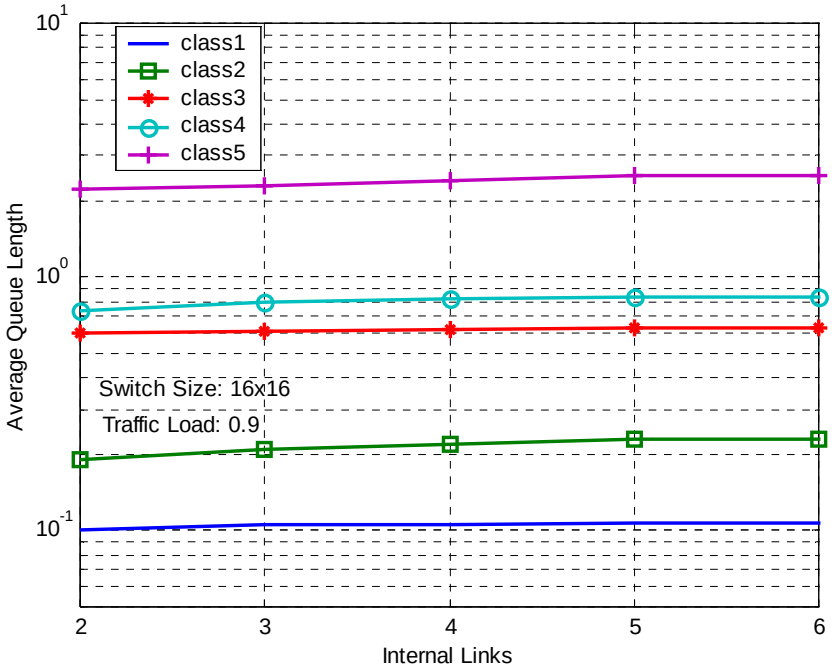

Figura 4.12.b - Comprimento médio da fila na saída para cada classe de serviço [2].

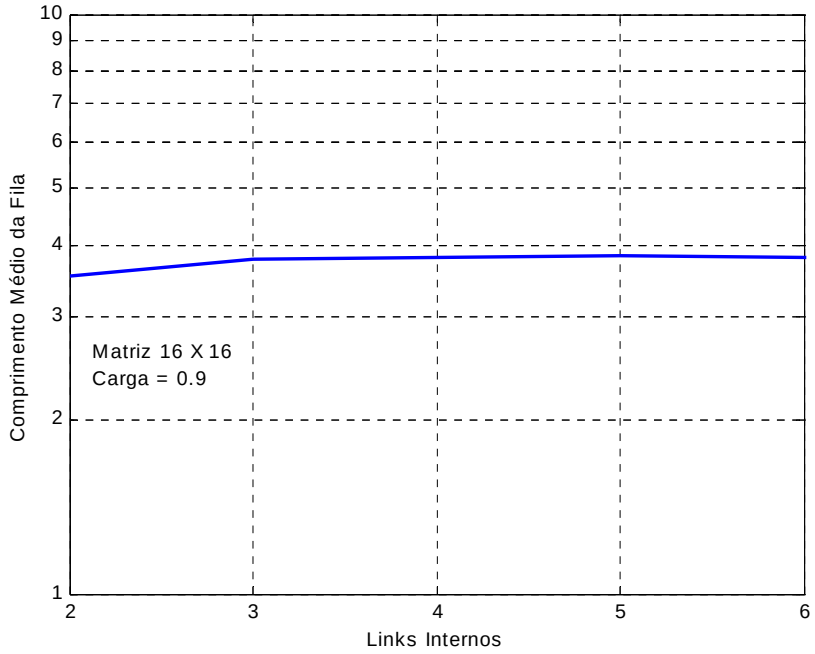

Figura 4.13 - Comprimento médio acumulado da fila de cada porta de saída .

CAPÍTULO 5

5 - CONCLUSÕES GERAIS

Este trabalho teve como objetivos modelar e simular as estruturas de comutadores ATM CIOQ propostas, utilizando o *software* ARENA, bem como analisar o desempenho e validar os modelos de simulação desenvolvidos. Objetivou também, realizar uma comparação destes resultados com os resultados obtidos em [1], [2] e [3].

Baseado nos resultados obtidos no processo de simulação pode-se concluir que as estruturas de comutadores estudadas, ou seja CIOQ, apresentam uma ótima solução ao problema de HOLB, pelo menos para o tipo de tráfego utilizado. Nos diversos modelos estudados, observou-se uma grande melhoria tanto no número médio de células nas filas, como também a probabilidade de bloqueio de células em relação ao modelo de filas simples tipo FIFO.

Foi verificado também que os resultados obtidos neste trabalho tiveram seus valores muito próximos aos valores encontrados em [1], [2] e [3] em todos os gráficos comparados, o que contribuiu para validar os modelos de simulação propostos.

Como se tratou de um primeiro contato com o *software* utilizado, muitas foram as dificuldades encontradas no desenvolvimento dos modelos. Dificuldades agravadas pelo reduzido número de pessoas encontradas que utilizam este *software* para a simulação em redes e/ou comutadores. No final, o *software* se mostrou uma ferramenta bastante útil para o modelamento e a simulação de comutadores de pacotes.

A partir dos resultados obtidos, e das comparações feitas com outros trabalhos, concluí-se que todos os objetivos propostos para este trabalho foram atingidos com sucesso.

Como sugestão para trabalhos futuros, sugere-se avaliar o comportamento destas mesmas estruturas de comutadores trabalhando com fontes de tráfego diferentes, como por exemplo, fontes de tráfego em rajadas (*burst*) e também combinações diferentes de tamanho da matriz, carga, número de *links* internos, bem como diferentes distribuições de probabilidades para as classes de serviço.

LISTA DE ABREVIATURAS

ABR – Avaliable Bit Rate
ACK – Acknowledgment
ATM – Asynchronous Transfer Mode
CBR – Constant Bit Rate
CIOQ – Combined Input-Output Queueing
CTR – Control
FIFO – First In-First Out
HOLB – Head of Line Blocking
HSRS – High-speed Statistical Retry Switch
IN – Interconnection Network
ISO – International Standards Organization
ITU-T – International Telecommunications Union
MATLAB – Matriz Laboratory
nrtVBR – non-real time Variable Bit Rate
OSI – Open Systems Interconnection
QoS – Quality of Service
REQ – Requisition
RDSI-FE – Rede Digital de Serviços Integrados – Faixa Estreita
RDSI-FL – Rede Digital de Serviços Integrados – Faixa Larga
rtVBR – real-time Variable Bit Rate
SCH – Schedule
SEs – Switching Elements
UBR – Unspecified Bit Rate
VBR – Variable Bit Rate
VOQ – Virtual Output Queueing

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Motoyama, S. Santos, C. R.: “Performance Analysis of a CIOQ ATM Switch With m Internal *Links* for Cell Transfer”, 10th International Conference on *Software*, Telecommunications & Computer Networks – SoftCom 2002, November 2002, 636 – 640.

[2] Motoyama, S. Santos, C. R.: "A QoS Provisioned CIOQ ATM Switch with m internal Links", 11th International Conference on Telecommunications - ICT 2004, August 2004, 698 - 703.

[3] Motoyama, S. Santos, C. R.: “A Combined Input-Output Queuing ATM Switch With m Internal *Links* for Cell Transfer”, International Telecommunications Symposium, September 2002, 142-146.

[4] Genda, K., Yamanaka N., Doi, Y.: “A High-Speed ATM Switch that uses a Simple Retray Algorithm and Small Input Buffers”, IEICE Trans. Commun. Vol. E76-B. No. 7, July 1993.

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

[6] Pricker, Martin de, “Asynchronous Transfer Mode: Solution for Broadband ISDN”, Third Edition, Prentice Hall, 1995.

[7] Karol, M., Hluchyj, M., and Morgan, S.:“Input Versus Output Queueing on a Space-Division Packet Switches”, IEEE Transacions on Communications, 35(12):1347-1356, December 1987.

[8] Motoyama, S.; Ono. L. M.: “Estudo de Desempenho de Algoritmos de Encaminhamento de Células em Comutadores ATM com Buffer de Entrada”. Campinas, SP, 1997. Dissertação de Mestrado – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

[9] Motoyama, S.; Mavigno, M.C.: “Algoritmos de Encaminhamento de Células em Comutadores ATM com buffer na Entrada”. Campinas, SP, 1997. Dissertação de Mestrado – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

[10] Motoyama, S., Petr, D. W. and Frost, V. S.: “Input – queued switch based on a scheduling algorithm”, Electronics Letters, 31(14), july 1995, 1127-1128.

[11] McKeown, N., Varaiya, P., Walrand, J.: “Scheduling cells in an Input-Queued Switch”, Electronics Letter, 3(11), December 1993, 323-325.

[12] Motoyama, S.; Ono, L. M., Mavigno, M.C.: “An iterative cell scheduling algorithm for ATM input-queued switch with service class priority”, IEEE communications Letters, 3(11), 1999, 323-325.

[13] Anderson, T. E., Owicki, S.S., Saxe, J.B and Tacker, C. P.: “High Speed Switch Scheduling for Local Area Networks”, Proc. Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 1992, 98-110.

[14] Karol, M., Eng, K., Obara: “Improving the performance of Input-Queued ATM Packet Switches”, Proc. INFOCOM 92, 1992, 110-115.

[15] Obara, H.: “Optimum Architecture for Input Queuing ATM Switches”, Electronics Letters, March 1991, 555-557.

[16] Motoyama, S.: “Simple high speed ATM switch with service class priority”, Electronics Letters, 36(6), March 2000, 590-591.

[17] Motoyama, S.: “Cell Delay Modelling and Comparison of Iterative Scheduling Algorithms for ATM Input-Queued Switches”, IEE Proc.-Commun., Vol.150, Nº 1, February 2003, 11-16.

[18] “Introdução à Simulação com o Arena 5.0”; Apostila da Paragon Cursos.

[19] Stallings, W. “Data and Computer Communications”, 6nd Edition. Prentice Hall, 2000.

[20] Tanenbaum, A. S.; “Computer Networks”, 4rd Edition. Prentice Hall, 2003.

[21] Achille Pattavina, “Switching Theory: Architecture and Performance in Broadband ATM Networks”, John Wiley and Sons, 1998.

[22] Junzhou Luo, Yong Lee e Jun Wu, “DRR: A Fast High-Throughput Scheduling Algorithm for Combined Input-Crosspoint-Queued(CICQ)Switches”, IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2005.

[23] Sundar Iyer, Rui Zhang e Nick McKeown, “Routers with a single stage of buffering”, Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2002.