Universidade Federal da Paraíba
Centro de Ciências e Tecnologia
Departamento de Sistemas e Computação
Coordenação de Pós-Graduação em Informática

Dissertação de Mestrado

Avaliação de Desempenho do Buffer Multiclasse COMATM

por

Karina Rocha Gomes da Silva

Universidade Federal da Paraíba

Centro de Ciências e Tecnologia
Coordenação de Pós-graduação em Informática

Karina Rocha Gomes da Silva

Avaliação de Desempenho do Buffer Multiclasse COMATM

Dissertação apresentada ao curso de mestrado em Informática da Universidade Federal da Paraíba,em cumprimento às exigências parciais para obtenção do Grau de Mestre.

Área de concentração: Ciência da Computação

Linha de pesquisa: Redes de Computadores e Sistemas Distribuídos

Orientador: Elmar Uwe Kurt Melcher

Campina Grande, Maio de 2001

Aos meus pais e minha vó Dora pela dedicação, amor e por sempre me apoiarem de forma incondicional.

Agradecimentos

Quero agradecer a Deus pela vida e tudo que se relaciona a ela. Também por me ajudar a continuar acreditando que tudo de bom que se busque vale a pena.

Aos meus familiares, em especial aos meus pais, irmãos, minha vó Dora e Tia Maria que são a base de tudo.

Ao meu orientador, professor Elmar, pela dedicação, paciência e por ser essa pessoa tão compreensiva.

Aos professores Bibi e Peter pelo muito que me ajudaram.

Aos meus amigos, de farra ou não, todos os colegas de trabalho, funcionários do departamento, principalmente à Aninha, Vera e aqueles que direta ou indiretamente colaboraram de alguma forma.

Quero agradecer de forma especial ao Glauco, Juliana, Renato, Cláudia e Tarig que estiveram ao meu lado em todos os momentos, principalmente nos mais difíceis.

Aos meus amigos PC, Carlos, Hilmer, Giovanni, Renata, Kyller e D. Francisca.

De uma forma muito carinhosa a todos os Tora-Toras que mesmo longe sempre estiveram presentes.

Resumo

Um dos fatores que causam um impacto muito grande no desempenho de uma rede de computadores são os comutadores. Dentro de comutadores são usados buffers para evitar que dados que não podem ser encaminhados imediatamente, fossem descartados. Além do tamanho do buffer, os seus algoritmos de envio e descarte têm o maior efeito sobre o desempenho de um comutador.

Esse trabalho trata da avaliação de desempenho de um buffer Multiclasse ATM, inserido no projeto de um comutador denominado COMATM. Essa avaliação foi realizada através da implementação desse modelo de buffer no ambiente de simulação Arena, selecionando cenários de tráfegos e submetendo-os a simulações no modelo. As análises são feitas comparando os resultados das simulações do modelo do buffer COMATM com os resultados de simulações feitas com os mesmos dados no modelo de um buffer usando o algoritmo FIFO.

Abstract

In a computer network, switches have a major influence on performance. Switches use buffers in order to avoid loss of data that cannot be passed on immediately. Beside the size of buffers, the algorithms they use for inserting and retrieving data have a big influence on switch performance.

This work presents a performance evaluation of an ATM multi-class buffer used in a switch called COMATM. The evaluation was done within a simulation framework called Arena, using a model of the buffer and injecting various selected types of network traffic into it. The performance parameters obtained where compared to those obtained by simulations of a FIFO buffer under the same traffic conditions.

Sumário

Lista de Figuras

Capítulo 1

1. Introdução

Nos primórdios das telecomunicações os comutadores foram introduzidos com a finalidade de evitar que cada telefone tivesse que ter uma conexão física com cada um dos demais. Se tivéssemos um número N de telefones seriam necessários N.(N-1)/2 conexões. Com a introdução das centrais de comutação ou comutadores, é necessário apenas que cada telefone tenha uma única conexão com a central mais próxima [11].

Para integrar as diversas tecnologias foi necessário formalizar o conceito de interligamento entre comutação e transmissão. Isso foi feito em Junho de 1971, numa reunião do CCITT, que definiu o termo Rede Digital de Serviços Integrados (RDSI – *Integrated Services Digital Network* (*ISDN*)) [23].

O surgimento da RDSI – FL (Redes Digitais de Serviços integrados Faixa Larga) tendo como modo de transmissão o ATM (*Asynchronous Transfer Mode*), que possui uma comutação rápida, garantiu um uso mais eficiente da banda passante e menor complexidade no processamento da comutação [02].

A partir de 1990 começaram a surgir os primeiros protótipos dos comutadores ATM. Porém, apesar dos esforços para a padronização destes, a padronização básica de ATM não possui um conjunto de requisitos para a implementação de comutadores. Isso dá margem a novas pesquisas e buscas de novos algoritmos para implementações cada vez mais eficientes ou com características específicas de grandes empresas e provedores de serviço [02]. Por isso, muita pesquisa tem sido feita em relação aos comutadores ATM.

A função básica de um comutador ATM é a transferência de blocos de informação (contendo dados do usuário, sinalização de controle ou de manutenção) das portas de entrada para as portas de saída (comutação propriamente dita), mantendo a ordem das células em cada conexão virtual [02].

Um comutador ATM é formado, basicamente, por controladores de entrada, elemento comutador, processador de controle e controladores de saída [02]. Como indicado na figura 8.

As células trafegam na linha física como uma seqüência de bits, e é função dos controladores de entrada delimitá-las e verificar possíveis erros. Na saída do comutador, essas mesmas células devem ser transformadas novamente em uma seqüência de bits para serem encaminhadas na linha física. Esse é o papel dos controladores de saída.

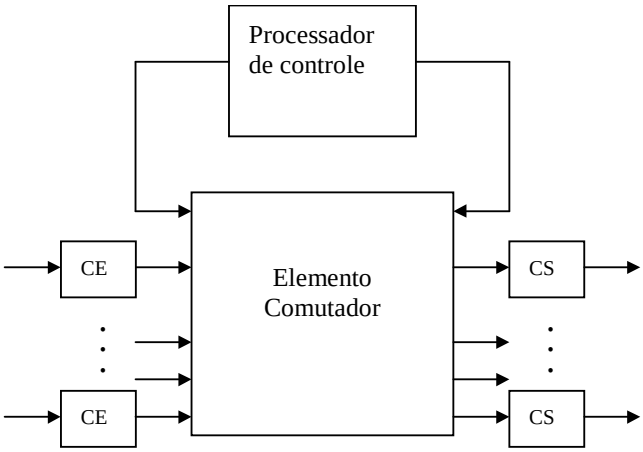

Figura 1 - Estrutura de um nó de Comutação ATM

O Elemento Comutador faz a comutação propriamente dita. Ele é composto da malha de comutação física e de buffers. Seu papel é transferir as informações das Portas de Entrada para as Portas de Saída.

O Processador de Controle é responsável por todas as funções de controle da comutação. Existe uma tabela de rotas, que pode ou não ser interna ao Processador, que possui a função de armazenar informações referentes à conexão ativa, tais como novo VPI, novo VCI e endereço da nova porta de saída. Estão também os parâmetros referentes às

especificações de tráfego da conexão, usados no algoritmo de policiamento das conexões e dados resultante do monitoramento do tráfego, como número de células roteadas, número de células descartadas, etc. O Processador de controle interage com essa tabela, e é responsável pelos procedimentos de controle da comutação. Esses procedimentos são, por exemplo: leitura dos parâmetros da tabela de rotas referentes à célula a ser roteada, manutenção dos dados da tabela, monitoração de utilização do comutador (policiamento da conexão) e geração de estatísticas da comutação (número de células roteadas, número de células descartadas, etc) [02].

O Comutador recebe muitas células para serem roteadas. As células que chegam em uma porta de entrada devem ser enviadas para uma porta de saída. Como ele possui muitas portas de entrada e saída, pode ser que duas ou mais células de portas de entrada diferentes, requeiram ao mesmo tempo, uma mesma porta de saída. Como somente uma delas pode ser transmitida em um instante de tempo, para que não haja perda de nenhuma delas, deve haver alguma forma de armazenamento das mesmas. Essa forma de armazenamento são os buffers.

Uma característica dos comutadores é que eles possuem uma capacidade física de recepção e transmissão de células e uma quantidade limitada de portas de entrada e saída. Quando essa capacidade se excede, ou seja, quando duas ou mais células necessitam, ao mesmo tempo, serem transmitidas para a mesma porta de saída, pode haver perda de células. Para resolver esse problema é necessária alguma forma de armazenamento, em nosso caso, os buffers.

Os buffers estão entre os recursos mais importantes que devem ser gerenciados em uma Rede de Computadores. De acordo com a alocação de tráfegos e canais (e conseqüentemente, buffers), uma política de atendimento e prioridades na ocupação do buffer possibilita garantir uma Qualidade de Serviço desejada para cada uma das classes de aplicações [11].

Nesse trabalho será avaliado o desempenho de um buffer multiclasse, que faz parte do Comutador COMATM [07], um projeto desenvolvido por várias entidades acadêmicas, assim dividido: a comutação física chamada Rcube crossbar foi realizado por um grupo de

pesquisas europeu formado pelo laboratório MASI, Universidade de Paris VI e Bull. Os circuitos de suporte foram desenvolvidos por pesquisadores da USP, UFPE e UFPB. O buffer multiclasse foi implementado usando VHDL (*Very Large Scale Integration Hardware Description Language*).

As seções abaixo mostram os objetivos deste trabalho, motivação, relevância e a forma como este documento está estruturado.

1.1. Objetivos

Sendo o buffer muito importante dentro de um comutador e conseqüentemente dentro de uma rede, é necessário que seus algoritmos funcionem de forma a atender aos requisitos de cada aplicação que está usando a rede.

O objetivo a ser alcançado nesta dissertação é a avaliação de desempenho do Buffer Multiclasse do Comutador COMATM, através de simulações. Essa avaliação analisará a taxa de perda e atraso médio das células, assim como irá compará-lo a outro algoritmo de disciplina de filas, o algoritmo FIFO (*first in first out*).

Para fazer essas simulações o modelo do Buffer foi projetado no ambiente de simulação Arena e submetido a vários cenários de tráfego para saber como ele se comporta.

1.2. Motivação e Relevância

Nem sempre é possível fazer experimentos com um sistema, mesmo que ele já esteja implementado, pois eles podem vir a causar danos no ambiente onde esse sistema está inserido [03].

É importante que um sistema a ser implementado seja analisado antes de sua implementação para saber se ele realmente irá se comportar como se propõe [03].

A importância deste trabalho de dissertação está na avaliação, através de experimentos simulados, dos algoritmos usados no buffer, que influenciam sobremaneira o

desempenho global dele, e conseqüentemente de toda a Rede. Esses algoritmos são avaliados com relação às perdas e atrasos de células dentro do buffer.

1.3. Estrutura da Dissertação

No Capítulo 2 é apresentada uma introdução às Redes ATM, enfocando as características relevantes para este trabalho.

O Capítulo 3 introduz conceitos de Avaliação de Desempenho, bem como a metodologia usada para realizá-la neste trabalho. Esse Capítulo mostra itens da metodologia adotada na Avaliação de Desempenho.

O Capítulo 4 apresenta o ambiente de Simulação Arena, escolhido para fazer a Simulação de Desempenho do Buffer Multiclasse, e apresenta também a implementação do modelo do Buffer Multiclasse, nesse ambiente.

O Capítulo 5 apresenta a validação do Buffer Multiclasse, as simulações e os resultados obtidos, juntamente com a interpretação dos dados.

O Capítulo 6 apresenta as conclusões deste trabalho.

Capítulo 2

2. Redes ATM

A evolução da tecnologia digital, juntamente com a necessidade de uma forma mais eficiente de comunicação, impulsionaram o desenvolvimento de novas formas de compartilhamento de diferentes mídias [11].

Cada uma dessas mídias necessitava de um meio dedicado para ser utilizada pelos usuários. Além de mídias com requisitos já conhecidos, foram surgindo novas mídias que necessitavam dos mais variados requisitos, alguns deles ainda sendo estudados.

Com o aumento da demanda por essas novas tecnologias, surgiu a necessidade de uma forma de integração dessas aplicações. Além disso, o fato da rede telefônica ter iniciado o processo de digitalização e o surgimento da fibra ótica com sua alta vazão e sua baixa taxa de erro levaram a uma maior facilidade para essa integração [02].

Surgiu então, em 1971, a idéia da RDSI (*Rede Digital de Serviços Integrados*). A idéia básica da RDSI é dar suporte a uma vasta gama de serviços através de um conjunto de interfaces de acesso único e padronizado, que assim como a tomada elétrica, seja universal e corriqueira, além de garantir flexibilidade para acomodar novos serviços sem a necessidade de se criar uma rede dedicada para os mesmos [11].

No início, foram usadas redes dedicadas para separar determinados tipos de tráfegos. Essas redes foram denominadas de RDSI – FE (*Rede Digital de Serviços integrados -Faixa Estreita*). A figura 1 ilustra esse tipo de Rede.

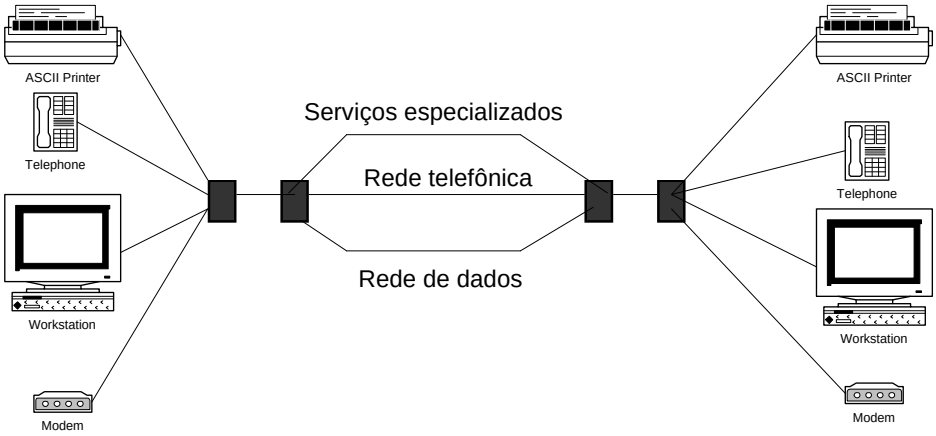

Figura 2 - Rede Digital de serviços Integrados - Faixa estreita [02]

A RDSI-FE evoluiu para uma tecnologia que integrava todos os tipos de tráfegos (tais como áudio, vídeo, dados), em um único meio compartilhado, e possuía acesso integrado. Essa Rede foi denominada de RDSI – FL (*Rede Digital de Serviços Integrados - Faixa Larga*). Ela usou o Modo de Transferência Assíncrono (ATM), que é uma tecnologia independente do meio físico utilizado. A figura 2 mostra o esquema de uma RDSI - FL.

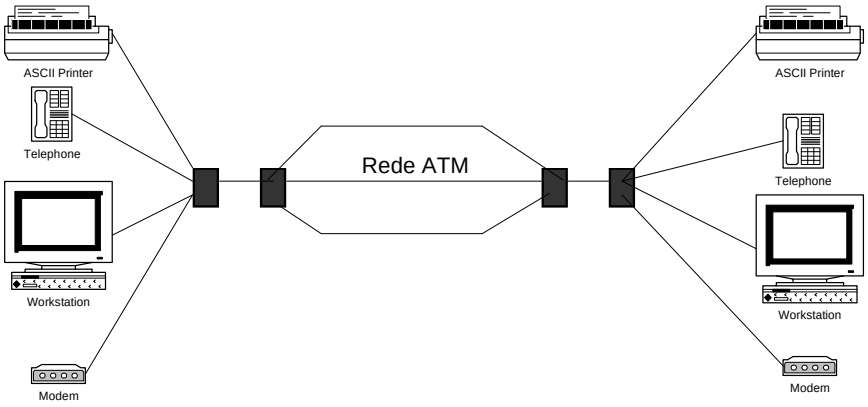

Figura 3 - Rede Digital de Serviços Integrados - Faixa Larga [02]

2.1. Modo de Transferência Assíncrono (ATM)

Existem dois modos de se transmitir dados em uma rede: usando o Modo de Transferência Síncrono (STM) e o Modo de Transferência Assíncrono (ATM – *Asynchronous Transfer Mode*) [01].

No Modo de Transferência Síncrono a transmissão é realizada em um *slot* de tempo. Dessa forma uma determinada conexão, que alocou o canal e não está usando o mesmo, faz com que parte da largura de banda alocada seja desperdiçada.

O Modo de Transferência Assíncrono, ao contrário, usa alocação do canal por Divisão Estatística de tempo. Existe um aproveitamento muito melhor da largura de banda, pois as aplicações se revezam mandando células no canal, e quando uma aplicação não está enviando nada a outra pode utilizar o canal, ganhando assim em desempenho [01].

A tecnologia ATM (*Asynchronous Transfer Mode*) foi adotada como modo de transmissão na RDSI-FL. Ela é capaz de integrar os mais diversos tipos de serviços, como também de suportar várias outras aplicações para as quais não foi desenvolvida, como interconexão de LAN's e WAN's ou interconexão de mainframes e supercomputadores [02].

A tecnologia ATM usa uma tecnologia baseada em unidades de informação de tamanho fixo chamadas células.

2.2. Células ATM

A célula ATM possui um tamanho fixo de 53 bytes. Ela possui 5 bytes para o cabeçalho e 48 bytes para os dados (*payload*). A figura 3 mostra como essa célula é dividida.

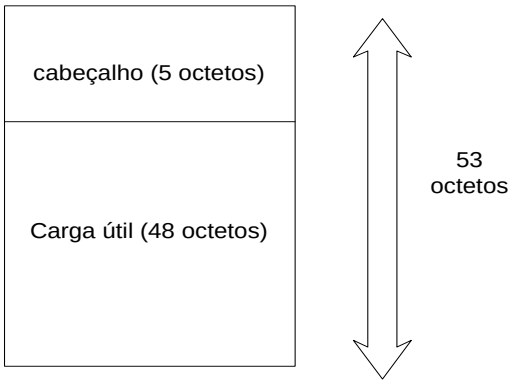

Figura 4 - Esquema de uma célula ATM

Existem vantagens e desvantagens em se usar células pequenas de tamanho fixo. Uma das vantagens é que uma célula pequena, com rígidos requisitos de tempo, não irá se atrasar esperando que uma célula grande seja transportada para que chegue a sua vez. Outra vantagem é a previsibilidade conferida pela célula [01].

Uma desvantagem decorrente desse padrão é que existe um *overhead* muito grande com relação ao tamanho do cabeçalho da mesma. Como em cada 53 bytes, 5 são de cabeçalho, esse desperdício chega a ser de quase 10%.

Existem dois tipos de cabeçalhos para as células ATM. Um deles é usado na interface Usuário-rede e o outro na interface Rede-rede. A figura 4 representa o cabeçalho usado nessas duas interfaces.

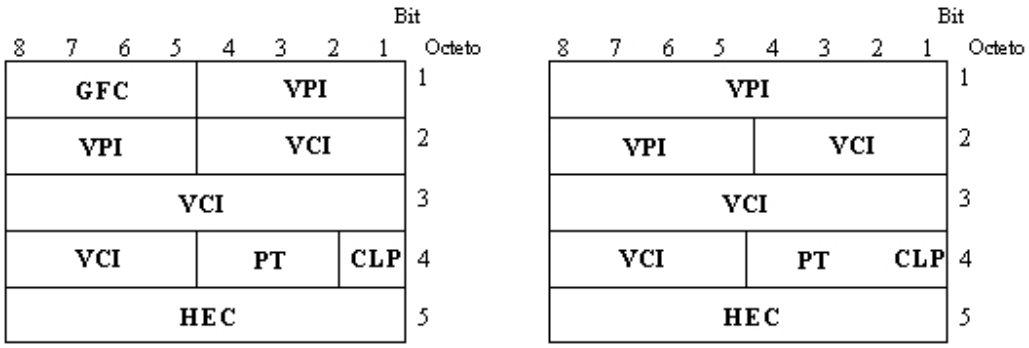

Figura 5 – Cabeçalhos de uma célula ATM para as Interfaces UNI e NNI

Esse cabeçalho é dividido da seguinte forma:

O campo GFC (*Generic Flow Control*) está presente somente em células enviadas entre o usuário e a rede. Ele foi criado para regular o fluxo de tráfego em redes ATM, mas ainda não teve o seu uso padronizado. Ele é sobrescrito pelo primeiro comutador que encontra, por isso não possui um significado fim-a-fim [01].

O campo VPI (*Virtual Path Identifier*) é um pequeno número inteiro, que indica a qual Caminho Virtual a célula pertence. Possui 8 bits na interface UNI, e 12 na interface NNI.

O campo VCI (*Virtual Channel Identifier*) indica a qual Circuito Virtual a célula pertence. Ele possui 16 bits, por isso, teoricamente um *host* pode ter até 65.536 circuitos virtuais [01].

O PT (*Payload Type*) define o tipo de carga útil que a célula contém. Se ela contém dados de usuário ou de gerenciamento.

O campo CLP (*Cell Loss Priority*) pode ser preenchido por um *host* para diferenciar entre células de alta e baixa prioridades. Sendo que 1 indica baixa e 0 alta prioridades. Esse campo é muito importante para esse projeto, pois em caso de congestionamento, o comutador irá descartar primeiro as células de baixa prioridade (CLP = 1).

O campo HEC (Header Error Control) permite a delimitação entre células e faz um controle de erros no cabeçalho. Ele não verifica a carga útil da célula.

2.3. Modelo de Referência dos Protocolos ATM

A RDSI-FL tem o seu próprio modelo de referência diferente do modelo OSI e diferente do modelo TCP/IP. Esse modelo consiste em 3 camadas, como é representado na figura 5, Camada física, camada ATM, camada de adaptação ATM e outras camadas superiores. Cada uma dessas camadas possui uma função específica [01].

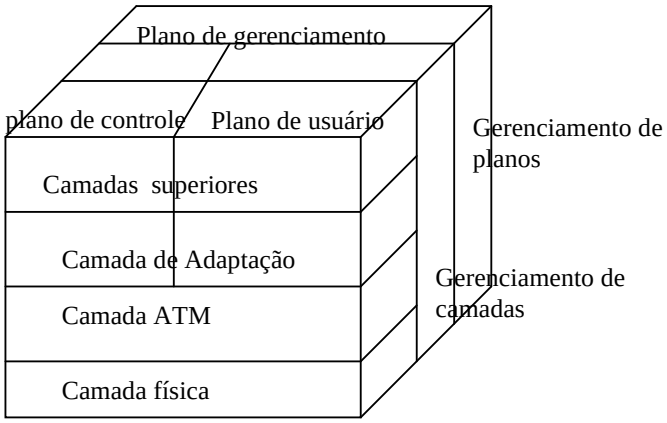

Figura 6 - Modelo de Referência ATM

A Camada Física lida com o meio físico como bits, voltagens e etc. Redes ATM foram projetadas para serem independentes do meio físico, podendo usar fibras óticas ou outro meio qualquer. Porém, a camada física prevê recursos de correção de erros de transmissão somente para o cabeçalho da célula, dificultando assim o uso de ATM para meios físicos ruidosos.

A Camada ATM lida com as células e o cabeçalho das células. Ela é responsável por verificar o conteúdo do cabeçalho da célula, e verificar o que significa cada um dos campos contidos nele. Também lida com controle de congestionamento. O buffer, assim como suas funcionalidades estão inseridas nessa camada. Por isso, essa será a camada principal a ser tratada nesse trabalho.

A Camada de adaptação é responsável por lidar com pacotes de outras aplicações que não trabalham com células ATM. Essa camada recebe os pacotes, segmenta-os, transformando-os em células, transmite-as e do outro lado da rede ele reagrupa essas células em pacotes novamente.

Como o modelo ATM é tridimensional, além das camadas, ele também é dividido em planos. O Plano de usuário lida com o transporte de dados, controle de fluxo, correção de erros e outras funções de usuário. O plano de controle é responsável pelo gerenciamento da conexão, incluindo estabelecimento e liberação da conexão e o Plano de Gerenciamento é responsável pelo gerenciamento de planos e gerenciamento de camadas.

2.4. Conexões ATM

Redes ATM são orientadas a conexão, por isso antes de enviar células, é necessário que haja uma prévia negociação e o estabelecimento de conexão entre a origem e o destino [01].

Os campos VPI e VCI juntos indicam uma Conexão Virtual e cada uma dessas conexões possui uma classe de serviço associada a ela, que se denomina QoS. Como mostra a figura 6, dentro de um mesmo VPI podem existir muitos VCI´s.

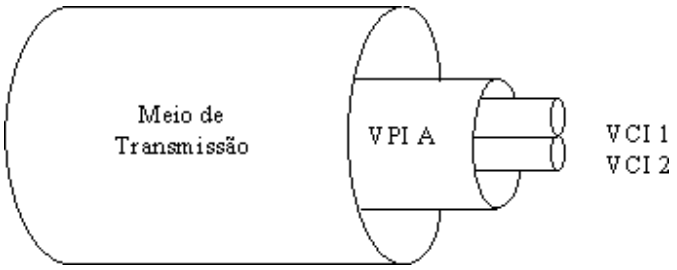

Figura 7 - Conexões Virtuais em Redes ATM

No momento do estabelecimento de uma conexão, a aplicação negocia seus atributos para se certificar de que a rede poderá atendê-la tanto sem prejudicar a aplicação que está fazendo a conexão, quanto as que já estavam usando a Rede. Esses atributos que a rede negocia com a aplicação são chamados de QoS.

2.5. A Qualidade de Serviço (QoS) em redes ATM

QoS (*Qualidade de Serviço*) é a capacidade de se fornecer um serviço de acordo com parâmetros pré-estabelecidos. Estes parâmetros incluem *jitter*, requisitos de latência e de perda controlados [12].

Em redes ATM, com determinadas características do meio físico, as possibilidades de se fornecer a QoS solicitada por cada conexão virtual são determinadas pelas

funcionalidades dos comutadores da rede. Dentro de um comutador ATM, os parâmetros de QoS possíveis resultam principalmente do desempenho do buffer e da carga já aceita. Não somente pelo seu tamanho, mas também pelos mecanismos de controle implementados por ele [02].

Uma das grandes vantagens de uma rede ATM é o suporte garantido para QoS nas conexões. Portanto, quando se deseja estabelecer uma conexão, a rede deve garantir que os requisitos contratados sejam atendidos durante o tempo em que a conexão está ativa.

De forma a simplificar a operação de gerenciamento, um certo número de classes podem ser definidas. Cada classe corresponde a um conjunto de características de requisitos de QoS [01].

Uma conexão pode ser especificada de acordo com os seguintes tipos de tráfego [01]:

CBR (C*onstant Bit Rate*): emula um fio de cobre ou fibra ótica. Os bits são colocados de um lado e são recebidos do outro lado da rede. Não há checagem de erros, nem controle de fluxo ou qualquer outro tipo de processamento;

VBR (*Variable Bit Rate*): como o próprio nome diz, possui uma taxa de bits variável. Essa classe é dividida em duas subclasses, uma de tempo real (*RT-VBR*) e outra não de tempo real (*nrt-VBR*). A RT-VBR é usada para aplicações de tempo-real que exigem que sejam obedecidos determinados argumentos de restrições ao tempo de apresentação.

ABR (*Available Bit Rate*): projetado para tráfegos em rajada, e cuja largura de banda disponível é conhecida antecipadamente. Esse é a única categoria de serviço na qual a rede fornece um *feedback* para a origem, dizendo para ela parar de enviar, em momentos de congestionamento.

UBR (*Unspecified Bit Rate*): nessa categoria o usuário não especifica a taxa que irá transmitir nem requisitos de QoS e a rede faz o melhor possível para entregar os bits que recebe da estação cliente. Se ocorrer congestionamento, os pacotes são descartados sem nenhum aviso prévio.

Os requisitos de QoS, tais como: taxa de perda da célula, atraso da célula, e variação do atraso da célula são estabelecidos durante o SETUP da conexão, de acordo com o tipo de serviço ATM requerido pelo cliente.

Para que tais garantias sejam realizadas, os comutadores ATM implementam uma função conhecida como CAC (*Controle de Admissão de Conexão*).

Essa função é ativada sempre que uma requisição de conexão é recebida por um comutador. Baseado nas conexões existentes e nas características de conexão da nova requisição, o comutador verifica se ao aceitar esta conexão com os parâmetros especificados continuará atendendo os requisitos de QoS das conexões já estabelecidas. Em caso afirmativo ele aceita a conexão, caso contrário, ele rejeita.

O CAC é uma função de comutação local, e é dependente da arquitetura do comutador. A figura 7 ilustra o processo simplificado de um Controle de Admissão de Conexão.

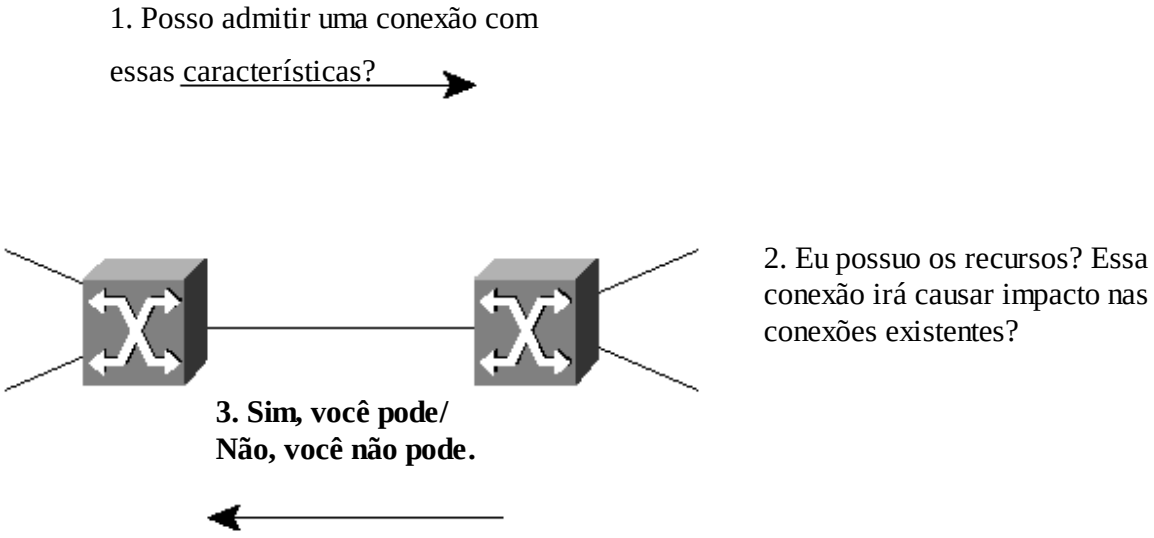

Figura 8 - Esquema usado para fazer o CAC

Fazer o estabelecimento de conexão em um comutador, de uma rede ATM, requer a sinalização e posterior negociação de parâmetros para que as características das aplicações sejam respeitadas e executadas de forma a não sofrer perda de qualidade.

Capítulo 3

3. Avaliação de Desempenho

Este capítulo introduz a técnica da simulação digital escolhida para esse trabalho, bem como os passos necessários para realizar essa atividade. A seção 3.1 faz uma introdução a alguns conceitos de Avaliação de Desempenho. A seção 3.2 introduz a Simulação Digital e a metodologia abordada durante a realização desse trabalho e a seção 3.3 faz a descrição do sistema.

3.1. Introdução

Saber como um sistema se comporta é muito importante para conhecer a capacidade do mesmo. Caso o sistema não esteja implementado, um teste faria com que fosse revelado se ao ser construído ele realizará o que se propõe. No caso de um sistema já pronto, fazer testes diretamente no sistema poderia ser prejudicial para as atividades do ambiente, além de custar caro e levar muito tempo. Por isso muitas vezes é preferível que os experimentos sejam conduzidos em um modelo do sistema [08].

Esse modelo, não necessita considerar o sistema com todos os seus detalhes. Por isso o modelo não é somente um substituto do sistema, ele é também uma simplificação do mesmo [08].

Na Avaliação de Desempenho deve-se estabelecer quais serão os parâmetros a serem medidos. Esses parâmetros devem conter todas as informações relevantes na análise do sistema. Outro fator importante é o método através do qual a avaliação será feita. Pode-se utilizar técnicas de Avaliação de Desempenho para prever o seu comportamento, tais como Simulação Digital e Métodos Analíticos.

Cada método possui suas vantagens e desvantagens. Os Métodos Analíticos possuem a vantagem de dar uma solução mais geral para o problema, mais econômica e mais eficiente, porém muitas vezes a sua aplicação é limitada pela complexidade do sistema que está sendo modelado. Nesses casos escolhe-se a simulação, por ser um método mais simples de ser implementado, e que se fazendo experimentos sob várias condições com o mesmo modelo, vem a fornecer resultados gerais do sistema em estudo [08].

3.2. Simulação Digital

Do ponto de vista prático, simulação é o processo de projetar e criar um modelo computadorizado de um sistema, para conduzir experimentos numéricos, objetivando um melhor entendimento do sistema em um dado conjunto de condições [03].

No final das décadas de 50 e 60, a simulação era extremamente complicada devido à necessidade de modelagem matemática dos sistemas e a implementação de algoritmos em linguagens de programação de propósito geral. O grau de dificuldade diminuiu com o surgimento das linguagens específicas para simulação, tais como, GPSS, SIMSCRIPT, SLAM, e SIMAN [03]. Atualmente estão surgindo simuladores de alto nível, cada vez mais fáceis de serem usados, que oferecem uma interface agradável e intuitiva para o usuário.

No entanto, para que uma simulação seja realizada é necessário primeiro que se estabeleça qual o problema que deve ser resolvido. Ou seja, o que deve ser analisado durante a mesma [03]. Os problemas devem estar bem claros antes do início de qualquer projeto.

Após o problema ter sido formulado, é necessário encontrar uma metodologia que seja útil para resolvê-los. A metodologia seguida neste trabalho está definida em [03] e possui os seguintes passos:

- Formulação do problema;
- Formulação do modelo e construção;
- Verificação e validação;

- Experimentação e análise;
- Apresentação dos resultados.

3.3. Descrição do Sistema

Existem várias formas de se usar buffers, sendo que cada uma causa um impacto diferente no comutador. Pode-se usar buffers nas portas de entrada, centralizados, compartilhados ou não e buffers nas portas de saída.

Usar buffers nas portas de entrada causa um problema de bloqueio de células chamado HOL (*Head of line blocking*). Esse bloqueio é causado quando se usa o algoritmo de escalonamento FIFO e alguma célula do início da fila, na porta de entrada, está esperando por uma porta de saída que está "ocupada" e chega uma célula na mesma fila que deve ser transmitida para uma porta de saída "livre", então essa segunda célula não pode ser transmitida devido ao fato da célula da frente estar bloqueada, impedindo que ela prossiga o seu caminho [15]. Além disso, em [14] foi mostrado que buffers nas portas de saída possuem um desempenho melhor que nas portas de entrada.

Para evitar esse problema, o COMATM adota o uso de buffers nas portas de saída. Dessa forma uma célula que chega, pela respectiva porta de entrada, é armazenada na fila da porta de saída pela qual ela deve ser transmitida. Os dados entram no buffer, vindos da malha de comutação física. A malha de comutação física, dentro da filosofia do projeto do comutador COMATM, sempre é capaz de encaminhar as células das suas entradas para suas saídas [07, 09].

Além da escolha da localização dos buffers, é necessário levar em consideração o fato de que uma Rede ATM possui suporte a QoS, por isso é necessário que haja uma diferenciação entre as aplicações que estão usando o meio, pois elas exigem requisitos diferentes. O buffer faz essa diferenciação usando múltiplas classes, sendo que cada uma delas contêm aplicações que possuem requisitos iguais de QoS.

Muitos algoritmos de gerenciamento têm sido propostos, baseados em buffer com filas separadas atribuídas para cada classe de tráfego [01]. Algumas dessas estratégias são:

Head of Line Priority : células que pertencem a classes mais altas são sempre servidas primeiro, enquanto as células de classes mais baixas devem esperar até que as de classe mais altas sejam servidas;

Weighted round-robin: As filas são servidas de uma forma cíclica. Filas pertencendo a classes mais altas são servidas mais freqüentemente que as pertencentes a classes mais baixas;

Push-out: Uma célula de alta prioridade pode entrar em um buffer que já está cheio, retirando as células de mais baixa prioridade;

Partial sharing: se uma fila é preenchida acima de um certo limiar, somente células de alta prioridade podem entrar na fila.

A escolha de qual algoritmo usar causa um impacto muito grande no desempenho do buffer. Nesse trabalho, a implementação do buffer multiclasse usa o algoritmo *Head of Line Priority* associado ao *Push Out*. As características do modelo estão descritas abaixo.

O controlador do buffer suporta 3 classes. Dentro de cada classe existem células de alta e baixa prioridades. Essas prioridades são indicadas pelo CLP de cada célula, sendo que CLP = 1 é baixa prioridade e CLP = 0 é alta prioridade [04].

O buffer usa listas encadeadas para guardar a posição das células contidas nele. Cada classe de células dentro do buffer possui duas listas, sendo que uma delas aponta para todas as células dessa classe e a outra somente para as células de baixa prioridade dela (CLP = 1). Essas listas são necessárias no momento da inserção de células, para evitar buscas iterativas demoradas [04].

Além dessas listas, o buffer possui uma lista de células livres, para guardar as posições livres do buffer, e um apontador para a posição que vai receber a próxima célula, pois o buffer assume que sempre existe pelo menos um espaço disponível para receber a próxima célula.

Quando uma célula chega no buffer, ela é imediatamente escrita no espaço livre, e em seguida é feita uma verificação para saber se o buffer está cheio. Se estiver, usando o

algoritmo de descarte de células, o buffer descarta uma célula mantendo um endereço livre para receber a próxima célula.

Esse algoritmo de descarte funciona da seguinte forma: faz-se uma busca nas células (incluindo a célula recém-chegada), partindo-se das classes de prioridade mais baixa para as classes de prioridade mais alta. Verifica-se se a classe possui células com CLP = 1, se houver alguma, então se descarta a que entrou no buffer por último. Se não existir células com CLP = 1, verifica-se se possui células com CLP = 0, se houver, descarta-se a que entrou no buffer por último. Se não houver nenhuma célula, parte-se para a próxima classe mais prioritária, até atingir a classe da célula que chegou.

A figura 9 mostra a forma como o algoritmo de recepção de células trabalha.

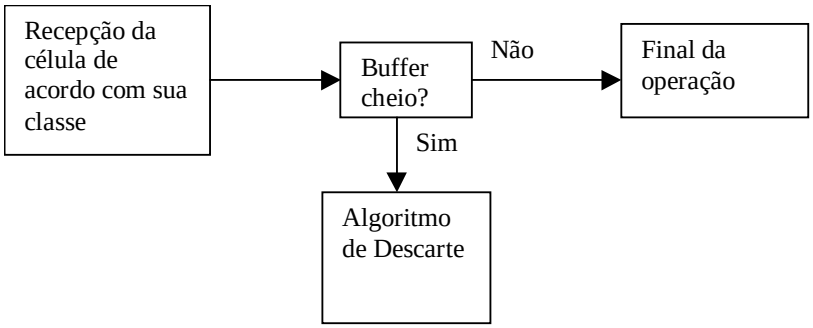

Figura 9 - Algoritmo de recepção das células

A figura 10 mostra como funciona o algoritmo de Descarte de células:

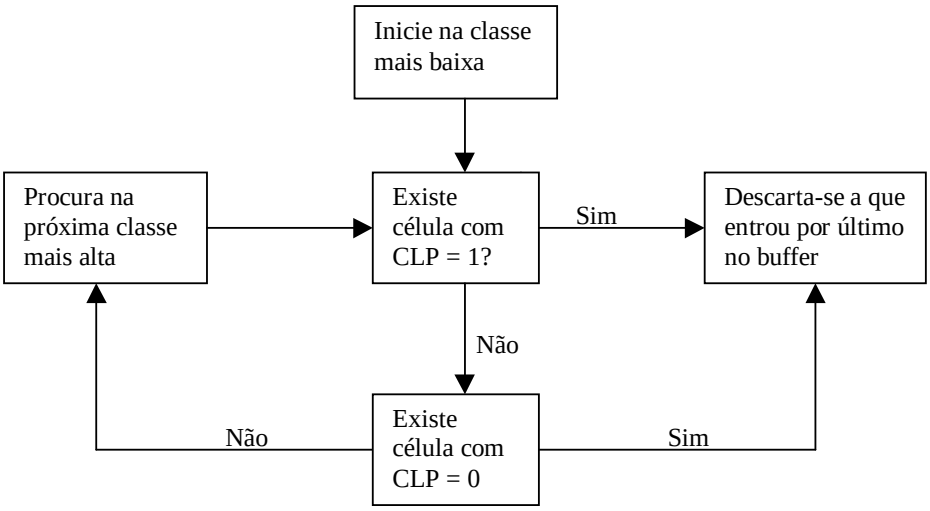

Figura 10 - Algoritmo de descarte de células

O algoritmo de recepção e descarte de uma célula **d** está mostrado abaixo:

Ligue *próxima_endereço_célula* para o fim da fila de *todas_as_prioridades* de classe *d*

Se CLP = 1 Então

Ligue *próxima_endereço_célula* para fim da fila de *baixa_prioridade* da classe *d*

Fim Se

Se *Livre* não estiver vazio Então

Tire endereço a partir do fim de *Livre* e defina-o para *próxima_endereço_célula*

Senão

Para classe = *menor_prioridade_classe* até *maior_prioridade_classe*

Se classe não está vazia Então

Se existe célula com CLP=1 Então

Tire endereço a partir da célula com CLP=1 que entrou no buffer por último e defina-o para *próxima_endereço_célula*;

Senão

Tire endereço a partir do fim da classe e defina-o para *próxima_endereço_célula*;

Fim Se

Exit Para

Fim Se

Fim Para

Fim Se

Quando se deseja enviar uma célula, deve-se enviar a primeira da fila, pois em uma mesma conexão virtual podem ter células com CLP = 0 e CLP = 1, e se isso fosse levado em consideração à ordem dessa célula na conexão seria alterada. O algoritmo de envio de células está mostrado abaixo.

Para classe = *maior_prioridade_*c*lasse* até *menor_prioridade_classe*

Se classe não está vazia Então

Retire a célula no endereço de *início_de_todas_as_prioridades* e o ligue para o fim de *livre*;

Exit Para;

Fim If;

Fim Para

Abaixo se encontram alguns exemplos de operações de inserção e remoção de células no buffer. Para esse exemplo serão usadas duas classes, Classe A e Classe B e a capacidade do buffer será de 4 células. O buffer possui uma lista de células livres, a qual aponta para todos os endereços de células livres. Possui também uma lista com o endereço da próxima célula, essa lista terá sempre um endereço livre para que se possa receber a próxima célula.

Inicialmente o buffer encontra-se vazio, com os endereços 0, 1, 2 e 7 sendo usados como flags para indicar o início da fila.

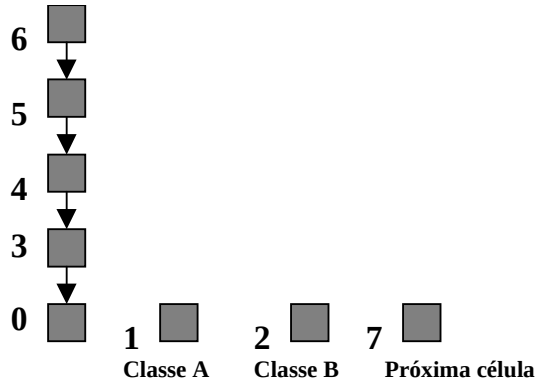

Figura 11 - Estado inicial do buffer

O Buffer recebe 3 células de classe A, 2 de alta e 1 de baixa prioridades. No momento da recepção, o buffer usa endereços da lista de próxima célula para colocar as células que estão chegando e aloca um novo endereço da lista de livres para colocar na fila de próxima célula.

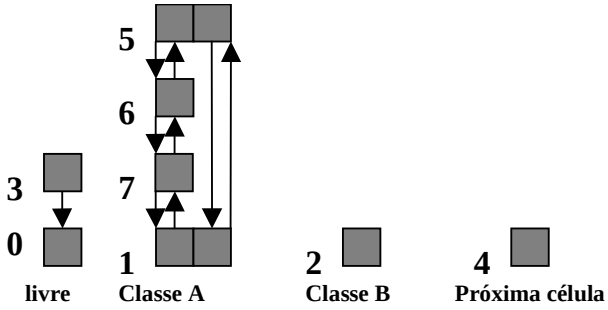

Figura 12 - Recepção de células de Classe A

Recepção de uma célula de classe B de alta prioridade. Nesse momento o buffer fica completamente cheio.

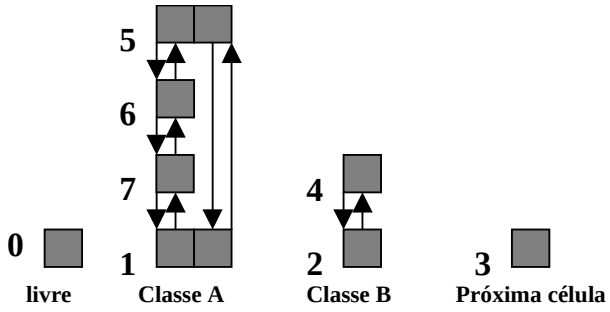

Figura 13 - Recepção de célula de Classe B

Envio de uma célula de Classe A. O envio é sempre feito na classe de mais alta prioridade, enviando a primeira célula, para que não haja um desordenamento das mesmas na sua conexão. Quando essa célula é enviada, o seu endereço é liberado e ela volta para a lista de livres.

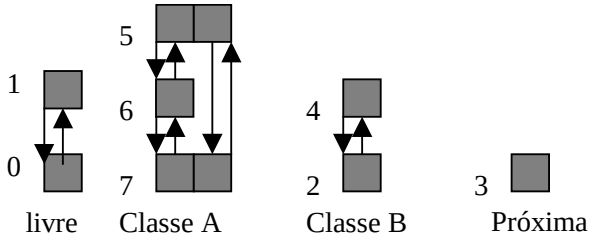

Figura 14 - Envio de uma célula de Classe A

Recepção de uma célula de Classe A, de baixa prioridade.

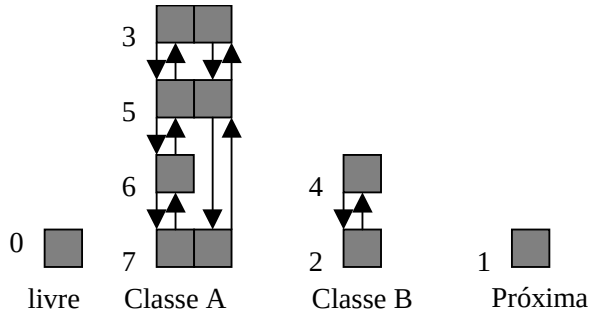

Figura 15 - Recepção de uma célula de classe A

Recepção de uma célula de Classe A de alta prioridade. Como o buffer está cheio, a recepção dessa célula causa descarte da célula 4 da Classe B, por essa pertencer à classe de prioridade mais baixa.

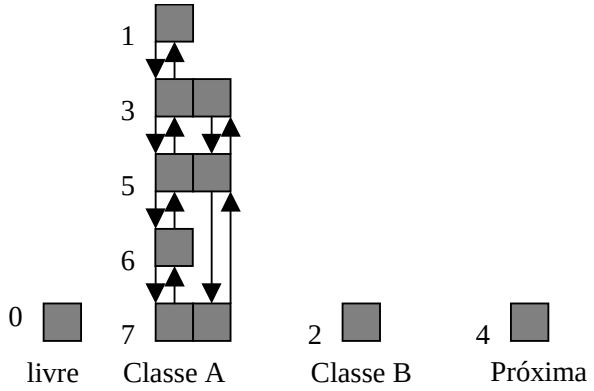

Figura 16 - Recepção de uma célula de Classe A

Recepção de uma célula de Classe A de alta prioridade. A recepção dessa célula causa descarte da célula 3, por não haver células na classe B para serem descartadas, e ela possuir CLP = 1 (baixa prioridade).

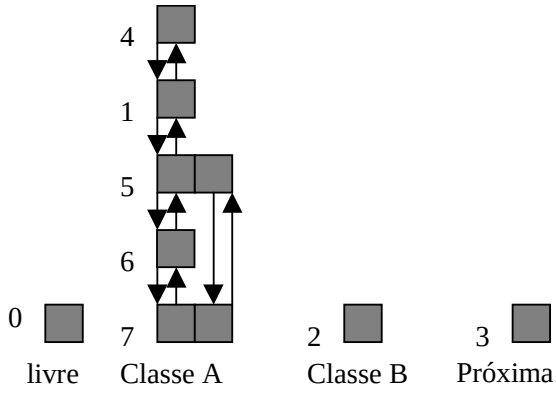

Figura 17 - Recepção de uma célula de Classe A

Capítulo 4

4.Ambiente e Modelo de Simulação

Esse capítulo introduz o ambiente de simulação escolhido para realizar a Avaliação de Desempenho, bem como mostra a forma como a implementação do modelo do Buffer Multiclasse foi realizada. A seção 4.1 introduz o ambiente de simulação Arena e a seção 4.2 mostra a implementação do modelo nesse ambiente.

4.1. O ambiente de simulação

O ambiente de simulação é muito importante para a realização dos testes de avaliação de desempenho. Se a escolha do ambiente não for adequada, pode ser que o mesmo venha a impedir que o modelo possa ser implementado no nível de particularidades requerido [03].

Por esse motivo é necessária à escolha de um ambiente de simulação que possua todas as características para implementar o modelo do sistema que deve ser avaliado.

Durante a fase de escolha de qual simulador deveria ser usado, alguns simuladores foram analisados. Entre os softwares gratuitos, um ambiente estudado foi o Simjava, um ambiente de simulação criado na Universidade de Edinburgh na Inglaterra. Ele é formado por pacotes de simulação para a linguagem Java [24].

Uma simulação no Simjava é realizada por um conjunto de entidades, cada uma executando em sua própria *thread*. Essas entidades estão conectadas por portas e podem se comunicar umas com as outras enviando e recebendo eventos. Uma classe do sistema central controla todas as *threads*, avança o tempo de simulação e dispara os eventos. O

avanço da simulação é registrado através de mensagens de *trace*, produzidas pelas entidades e salvas em arquivos [24].

Foram feitos testes com essa ferramenta. No decorrer dos mesmos ela se mostrou bastante flexível, e com as funcionalidades necessárias para realizar a simulação do buffer do comutador ATM.

Porém, os testes feitos demonstraram que os tempos necessários para completar simulações com razoáveis quantidades de eventos eram muito altos. O SIMJAVA demora um tempo excessivamente grande para gerar entidades. Um teste feito com uma única fonte e um único sorvedouro, com a geração de 100.000 células demorou 17 horas para simular. Para uma atividade simples como essa, o tempo foi considerado excessivamente grande. Outros testes como este também foram feitos e constataram altos tempos de execução. Além disso, essa ferramenta não possui um interpretador para os dados de entrada.

A outra ferramenta estudada foi o Arena, da Corporação *Systems Modeling*. Essa foi a ferramenta escolhida para a realização da simulação, e as suas principais características serão mostradas na próxima seção.

4.1.1. Arena

O Arena foi escolhido para a realização desse trabalho devido à sua disponibilidade, por possuir elementos de alto nível que podem ser integrados com elementos de baixo nível, dando-lhe flexibilidade, e por possuir ferramentas para interpretar dados de entrada e fornecer estatísticas de saída, o que facilita a interpretação da simulação.

Além disso, observa-se no Arena um crescimento aproximadamente linear do tempo de processamento em função do número de entidades geradas e a quantidade de componentes que integram o modelo. Tal fato permite facilmente prever seu comportamento perante a modelos mais complexo e/ou com maiores números de entidades [06].

Outro fator importante é que o Arena combina a facilidade de uso encontrada em simuladores de alto nível (interface gráfica, componentes que podem ser inseridos e configurados sem a necessidade de programação e etc) com a flexibilidade de linguagens

de simulação e possui até mesmo conexão com linguagens procedurais como Visual Basic, Fortran e C [03].

Existem duas edições do Arena, a acadêmica e a profissional. Na edição acadêmica pode-se usar o produto sem a licença, porém somente 150 entidades podem ser geradas. Na edição profissional, disponível na UFPB, não há imposições em relação à quantidade de entidades que podem ser geradas.

4.1.2. Módulos do Arena

O Arena possui muitos módulos que podem ser combinados para construir uma grande variedade de aplicações.

Existem módulos mais avançados, que possuem todas as informações necessárias para definir o componente a ser modelado. E existem módulos mais simples, que contêm elementos de simulação básicos que podem ser combinados para criar lógicas mais complexas.

O Arena é composto de 3 painéis: O painel *COMMON*, contendo módulos representando processos de simulação fundamentais, tais como chegadas, serviços e partidas; o painel de *SUPPORT*, contendo módulos suplementares para ações específicas e lógicas de decisão; e o painel *TRANSFER*, cujos módulos são usados para modelar o fluxo de transferência de entidades através do sistema [03].

A tecnologia interna do Arena é a linguagem de simulação SIMAN. Os módulos, contidos no template ARENA, foram criados usando blocos de modelagem SIMAN, assim como seus componentes. Esse conceito hierárquico prevalece no template ARENA. Blocos SIMAN estão disponíveis para os usuários do Arena, no template SIMAN [05].

SIMAN é uma linguagem de simulação, que foi introduzida no início dos anos 80 e executa em Mainframes, mini e microcomputadores. Ele permite rápida e precisamente implementar certas simulações em um computador [03].

Como mostra a figura 18, os *módulos* são agrupados em *painéis* para compor um *template* [03].

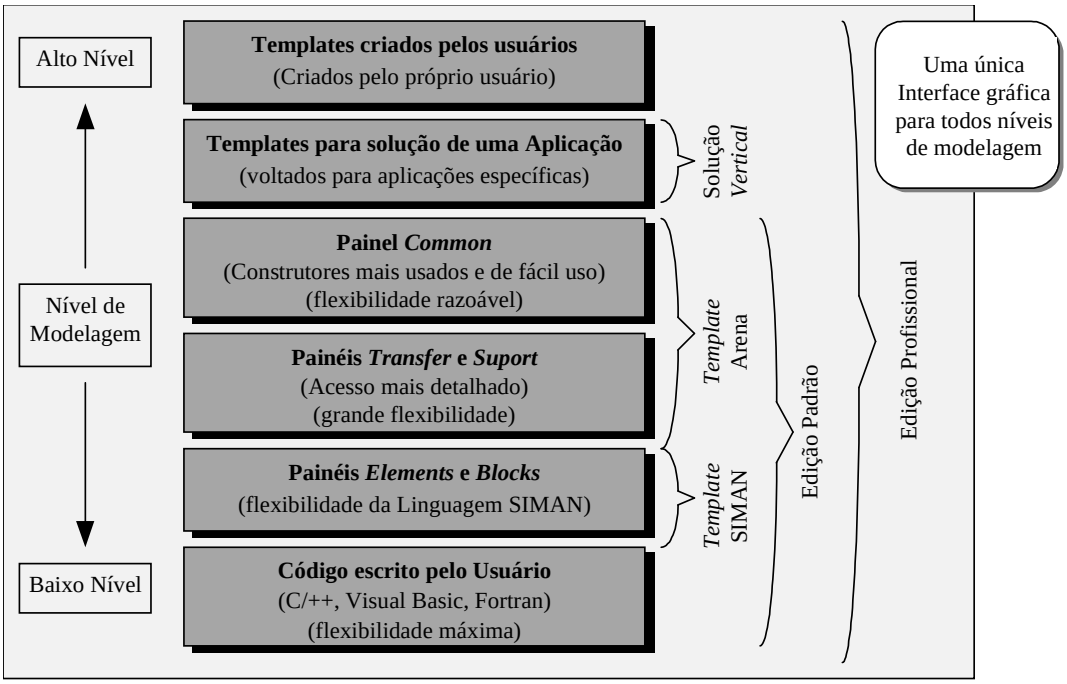

Figura 18 - Estrutura hierárquica do Arena [03]

O Arena mantém sua flexibilidade de modelagem sendo altamente hierárquico. Em qualquer momento pode-se usar *módulos* do *template* SIMAN e ganhar acesso para a flexibilidade da linguagem de simulação, e pode-se misturá-lo também com *módulos* de alto nível de outro *template* [03].

Os *módulos*, nos *templates* do Arena, são formados por componentes SIMAN, e usando a edição profissional do Arena, pode-se criar módulos próprios e colocá-los em *templates* para várias classes de sistemas [03].

O Arena possui algumas ferramentas que ajudam na geração e interpretação dos dados em uma simulação. Algumas dessas ferramentas serão descritas abaixo:

Analisador de Entradas (*Input Analyzer*): é uma ferramenta padrão, que acompanha o Arena, e é projetado especificamente para gerar distribuições para os dados observados, fornecer estimativas para os seus parâmetros e medir quão bem eles se encaixam nos dados.

Ele recebe dados de um arquivo de entrada e a partir deles gera uma função de distribuição de probabilidade que corresponda aos mesmos. Essa função pode ser usada diretamente no modelo. Quando o *Input Analyzer* encaixa uma distribuição para os dados, ele estima os parâmetros de distribuição e calcula um número de medidas que avaliam a correspondência deles com os dados. Essa ferramenta terá o seu uso mostrado mais a frente no momento da definição de cenários para o uso deles como dados de entrada para análise do modelo do Buffer Multiclasse.

Editor de Visual Basic (*Visual Basic Editor*): acompanha o Arena e pode ser usado na integração do Visual Basic com o ambiente de simulação. Para que haja essa integração, deve-se fazer uso do módulo *VBA* (Visual Basic for Applications) do template *BLOCKS*. Esse módulo declara automaticamente uma rotina no Visual Basic para escrever o código referente à parte do problema que deve ser modelada.

Essa ferramenta é muito útil quando é necessária a modelagem de elementos que o Arena não suporta, ou suporta fracamente.

A próxima seção mostra a implementação do modelo do Buffer Multiclasse usando as ferramentas do Arena mostradas.

4.2. Modelagem do Sistema

Essa seção expõe como o modelo foi implementado na ferramenta de simulação Arena, as simplificações efetuadas e os métodos utilizados para que o modelo pudesse ser implementado nessa ferramenta.

Para a construção do modelo, usado na avaliação de desempenho do Buffer Multiclasse, foi levada em consideração a descrição do buffer do Capítulo 3.

Esse modelo foi implementado no Arena e consiste das seguintes partes: possui 3 fontes gerando dados com diversas taxas, dependendo do cenário. Cada fonte dessas foi implementada por duas fontes no modelo, sendo que uma delas gera células de alta prioridade (CLP = 0) e a outra células de baixa prioridade (CLP = 1). Porém, para que a velocidade de geração não exceda o tempo de 0.9μs (tempo de 3 células), que é o tempo de

recebimento de células que a malha RCUBE do comutador suporta, é necessário que essas células fiquem armazenadas em uma fila, denominada doravante fila limitadora, para garantir a liberação de uma célula somente a cada 0.9μs. Essa fila garante que nenhuma célula será perdida na malha de interconexão do comutador, e ela terá um tamanho suficiente para não perder células. Dessa fila as células prosseguem para as filas do buffer que executam os algoritmos do mesmo. Depois dessas células serem colocadas nessas filas elas são enviadas ou descartadas, conforme a situação do buffer. A figura 19 ilustra como é o funcionamento do modelo do Buffer. O restante do capítulo explica detalhadamente como foi realizada essa implementação.

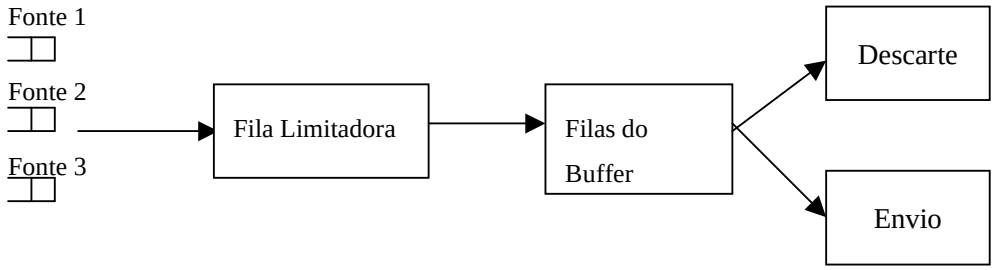

Figura 19 - Modelo do Buffer para implementação

4.2.1. Simplificações do Modelo

O modelo possui uma capacidade de armazenamento de até 600 células. Ele permite 3 classes com prioridades diferentes. Essas classes são distinguidas pela QoS que pode assumir os valores 0, 1 e 2, sendo que valores menores estão associados à prioridades maiores. Em cada classe, uma célula pode possuir alta ou baixa prioridades, respectivamente 0 ou 1, dependendo do valor estabelecido em seu CLP. As células fazem a identificação do Caminho e Circuito Virtuais que utilizam através da sua variável VPIVCI, que pode assumir os valores 0 até 6.

O Buffer Multiclasse possui um relógio de sistema com ciclo de 15 ns e a malha do comutador precisa de um intervalo entre uma célula e outra de 0,9 μs. O tempo para o envio de uma célula para a linha física é de 2,61μs.

No modelo implementado no ambiente Arena, a variável TNOW guarda o tempo simulado corrente, esse tempo é importante para identificar o tempo em que uma célula entrou no buffer, e o momento em que ela saiu, pois a partir desses dados pode-se calcular o atraso dela dentro do buffer. O componente WRITE escreve em um arquivo esse tempo de chegada e saída de cada célula, os seus valores de CLP, VPIVCI, QoS, número da fonte que a gerou e número de seqüência da célula.

No modelo, para ser compatível com a implementação real do buffer, o tempo de chegada de uma célula no buffer é gravado em ciclos de relógio, e o tempo de saída do buffer é gravado em nanossegundos. Como a unidade de tempo adotada nesse projeto foi milissegundos, é necessário que ao gravar os dados de tempo das células para um arquivo de saída, seja feita uma conversão nos mesmos. Essa conversão é realizada no momento de gravação do tempo no componente WRITE. O tempo de chegada da célula é dividido pelo ciclo do relógio (15 ns) e multiplicado por 1.000.000 para obter o tempo de ciclos do relógio em nanossegundos. E no momento de saída da célula, o componente WRITE multiplica o tempo por 1.000.000, para obtê-lo em nanossegundos.

4.2.2. Implementação usando elementos do Arena

Os elementos do Arena que foram usados para fazer a implementação do modelo do Buffer Multiclasse serão mostrados nessa seção.

As fontes de tráfego para a geração dos dados do Buffer Multiclasse foram criadas usando o componente CREATE. Esse componente permite que se especifique o tempo entre as células que serão geradas, o tempo de criação da primeira célula, a quantidade máxima de células que essa fonte irá gerar, e atributos que devem pertencer a cada entidade individualmente. Os atributos que foram definidos para cada célula ATM são QoS, VPIVCI, CLP, número da fonte e o número de seqüência da célula gerada. A figura 20 mostra um exemplo de uma interface do componente CREATE.

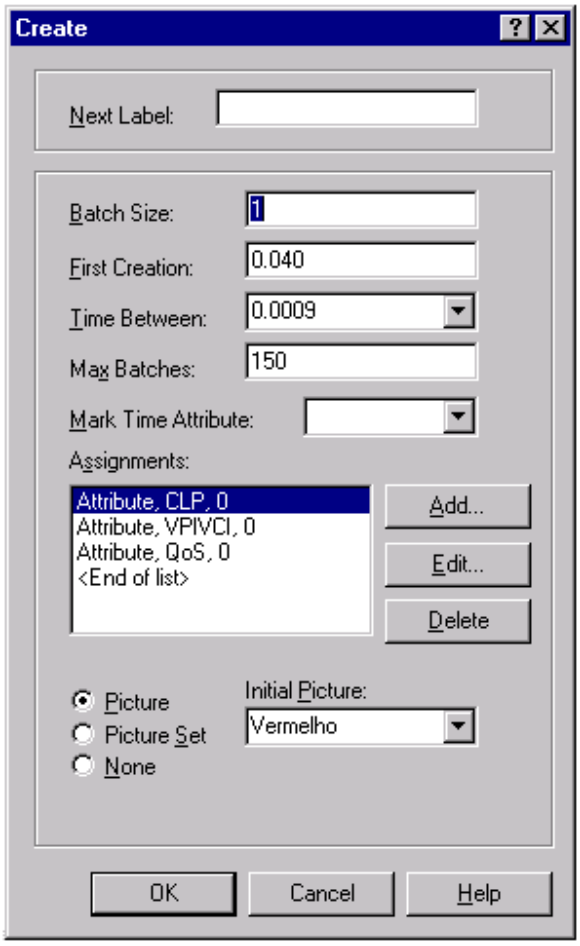

Figura 20 - O Elemento CREATE do Arena

No modelo foi implementada a possibilidade de geração de até 3 classes. Cada classe é alimentada por duas fontes: uma fonte gera células de alta prioridade (CLP = 0), e a outra gera células de baixa prioridade (CLP = 1). Em cada fonte o tempo de interchegada de células varia de acordo com uma função que será escolhida para representar o tráfego a ser usado na avaliação.

A figura 21 apresenta as fontes de tráfego (modeladas com o componente CREATE), que enviam células para a fila limitadora. Nessa figura, o componente VBA é usado para inserir as células na fila limitadora (código mostrado no Apêndice A), sendo que as células são inseridas na ordem em que elas são geradas, e são retiradas utilizando o algoritmo FIFO. Esse componente (VBA) permite a integração do Arena com o Visual

Basic, para que esse realize algumas funções que não podem, ou podem de forma ineficiente ser executadas pelo Arena.

O componente DISPOSE registra a saída de células dessa fila.

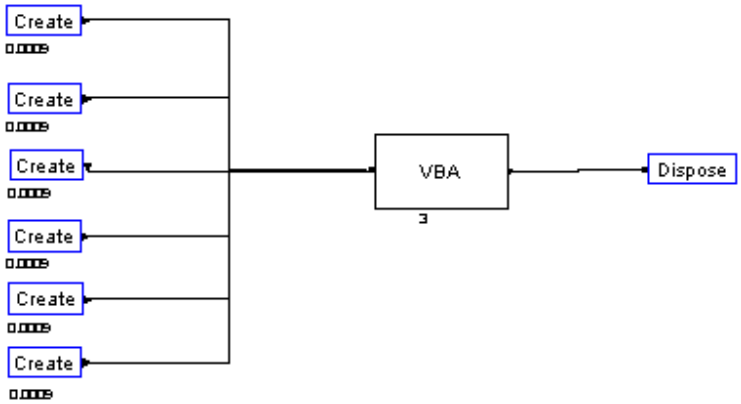

Figura 21 - Geração de Tráfego

A figura 22 mostra a entrada das células no Buffer Multiclasse, retiradas da fila limitadora. Na retirada de uma célula da fila limitadora, o componente CREATE atua como um disparador para acionar o elemento SEARCH que precisa ser disparado por algum evento. O CREATE envia uma célula a cada 0,9 μs. Esse disparador aciona o componente SEARCH, que vai procurar a primeira célula dessa fila, e retorna a sua posição para o Arena através da variável interna J. O componente REMOVE, que serve para retirar células de uma fila na posição da variável J, retira essa célula da fila.

O tempo de chegada da célula, assim como seus atributos (QoS, CLP, VPIVCI, número da fonte e ordem da célula gerada) são gravados através do componente WRITE, em um arquivo para análises posteriores.

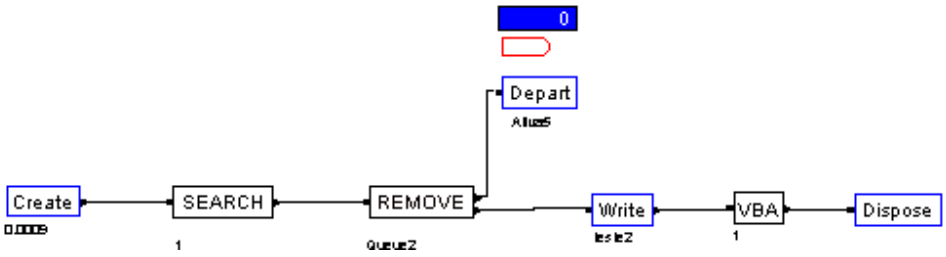

Figura 22 - Chegada de células no Buffer Multiclasse

Após isso, as células devem ser encaminhadas para o Buffer Multiclasse propriamente dito. O componente VBA da figura 22 tem a função de inserir as células no Buffer Multiclasse. Essas células são inseridas de acordo com o algoritmo de recepção de células do modelo, que pode ser observado no Apêndice A.

No momento de inserção das células, observam-se aquelas de classe mais prioritária. Ou seja, todas as que contêm a QoS menor são colocadas na frente da fila, garantindo que no momento da liberação de uma célula, seja enviada aquela de classe mais prioritária. A escolha de qual célula deve ser inserida primeiro na fila foi feita através do componente QUEUE, que possui um campo onde se pode especificar como a célula deve ser inserida na fila. Esse campo chama-se *Ranking Rule*, escolhendo-se a partir dele a opção *LowValueFirst*, ele insere a célula que possui o menor valor de QoS, ou seja a maior prioridade na frente. A figura 23 mostra um exemplo de um componente QUEUE.

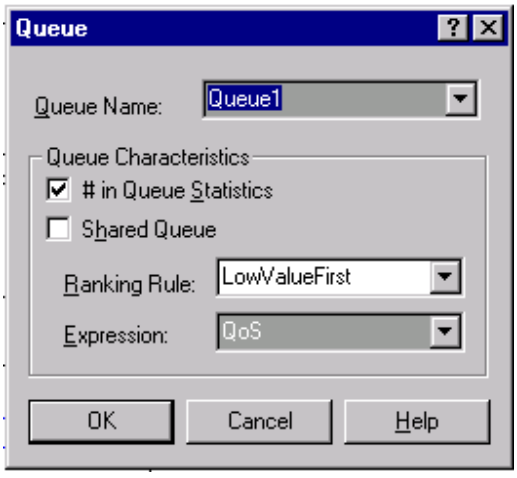

Figura 23 - O componente QUEUE do Arena

A figura 24 mostra o processo de envio de uma célula do Buffer Multiclasse. Esse envio é feito através do disparo de um componente CREATE a cada 2,61ns (tempo de envio de uma célula para o meio físico, na taxa de 155Mbps). O componente SEARCH é disparado e coloca a posição da primeira célula na variável J e então o componente REMOVE retira essa célula. Nesse momento, o componente WRITE registra o tempo em que a célula foi retirada, e os demais atributos já citados, em um arquivo para serem analisados depois.

A partir daí o componente BRANCH, Através do componente VPIVCI de cada célula organiza cada uma delas de acordo com a conexão a qual ela pertença, separando-as. O componente DEPART retira as células do modelo.

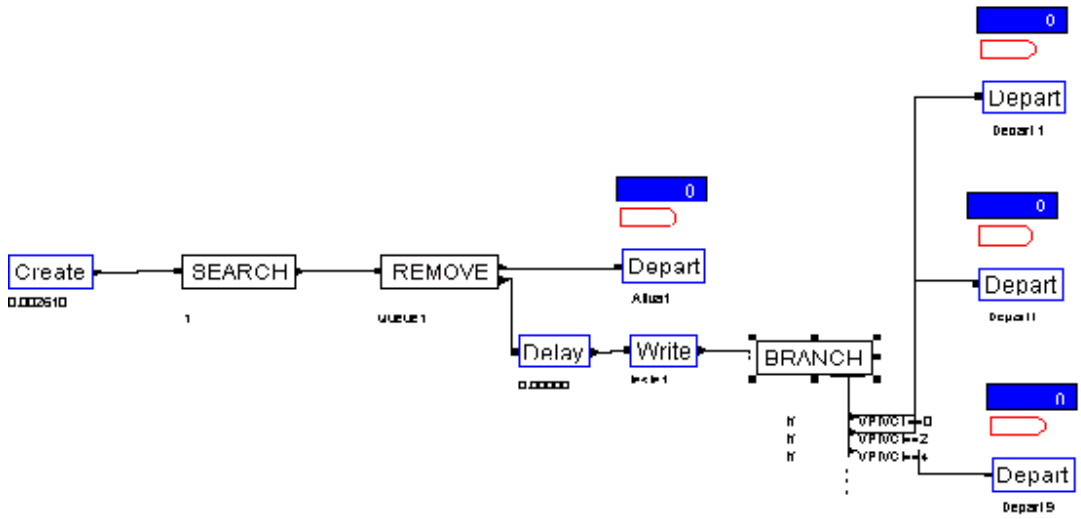

Figura 24 - Envio de células do Buffer Multiclasse

A figura 25 ilustra o processo de descarte de uma célula. Esse processo ocorre quando a capacidade de armazenamento do buffer chega ao seu limite (599 células, deixando o espaço para mais uma célula). O algoritmo de descarte é ativado pelo CREATE, que é disparado a cada 0.9µs, para escolher qual célula será descartada. Esse algoritmo foi implementado no Visual Basic e integrado ao Arena através de um componente VBA, ele pode ser visto no apêndice A. Quando o VBA é disparado o algoritmo de descarte é

acionado e retorna a posição da célula que deve ser retirada, através da variável interna J, e esta célula é retirada pelo componente REMOVE. A partir daí o componente BRANCH divide as células, determinando qual classe a célula pertence observando o VPIVCI atribuído a ela.

Depois disso o componente DEPART retira a célula do sistema.

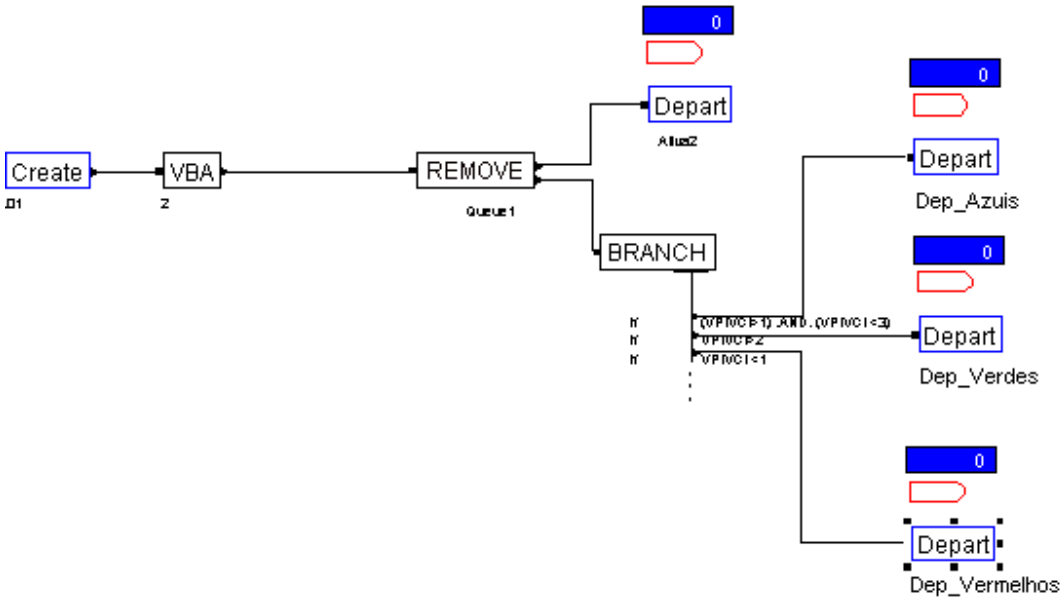

Figura 25 - Descarte de Células

Capítulo 5

5. Resultados das Simulações

Este capítulo trata da validação do modelo implementado no simulador Arena, juntamente com os resultados das simulações e a análise feita dos dados resultantes. A seção 5.1 dá uma introdução sobre validação, a seção 5.2 mostra os métodos utilizados para a realização dessa validação, a seção 5.3 mostra os cenários de tráfego utilizados na realização das simulações e a seção 5.4 mostra os resultados obtidos em cada simulação.

5.1. Introdução

Validação é o processo de determinar se o modelo conceitual reflete os aspectos do espaço do problema que precisa ser mapeado. É também uma forma de determinar se a operação do modelo do software final está consistente com o mundo real, usualmente através de experimentação e comparação com um conjunto conhecido de dados [03].

É necessário validar o modelo do buffer Multiclasse, isto é, verificar se o modelo do buffer implementado no Arena realizará as mesmas operações que o modelo original do Buffer implementado em VHDL realiza. Para isso foi utilizado um método para verificar que os modelos são equivalentes e executam as mesmas atividades. Este método está mostrado na seção 5.2.

5.2. Validação

Para validar o Buffer implementado no Arena foram comparados os resultados obtidos deste modelo com os resultados obtidos no Projeto COMATM implementado em VHDL (*Very Large Scale Integration Hardware Description Language*), pois esse é o modelo original do buffer.

Para fazer essa comparação foi necessário coletar dados dos arquivos de saída do Buffer implementado no Arena e do buffer implementado no VHDL.

Esses dados são coletados no Arena da seguinte forma: quando as células são criadas no modelo do Arena, os dados de entrada das células geradas na simulação (tempo de entrada da célula no buffer em ciclos de relógio, VPIVCI, QoS, CLP, número da fonte e número de seqüência da célula) são gravados em um arquivo pelo componente WRITE.

No momento em que as células são retiradas do buffer, no modelo do Arena, elas passam por um componente WRITE que grava os seus atributos de saída (agora com tempo de saída em nanossegundos), permitindo que as células que estão saindo sejam identificadas unicamente, e que o seu tempo seja registrado.

Depois que esses arquivos são gravados é necessário fazer a simulação com VHDL para comparar. Para fazer essa simulação os arquivos de entrada e saída do Arena são convertidos para o formato aceito em VHDL, através de um programa escrito em C. O arquivo de entrada é então submetidos à simulação no VHDL, que gera um arquivo de saída, e o resultado é comparado com o arquivo de saída do Arena que foi convertido para o formato VHDL.

Esse processo foi realizado de forma iterativa. Algumas vezes foi necessário voltar para a fase de implementação do modelo e fazer modificações e depois voltar novamente para a fase de validação.

A validação teve o seu término quando se obteve resultados semelhantes nos arquivos de saída dos dois modelos, com um limite de tolerância de tempo de 2.6μs.

Seguem exemplos de resultados das simulações realizadas que validam o modelo do Buffer Multiclasse implementado.

1. Foram simuladas 1800 células, de uma única classe com prioridade 0. Essa classe gerou 900 células de alta prioridade (CLP = 0) e 900 células de baixa prioridade (CLP=1). A simulação foi realizada durante 5ms.

Os dois modelos, Arena e VHDL, apresentaram arquivos de saída iguais. Cada um deles resultou em um arquivo com 1200 células.

2. Nessa segunda simulação foram enviadas para o Buffer Multiclasse 900 células, com 3 classes diferentes, com prioridades 0, 1 e 2. Foram geradas 300 células para cada classe, 150 de alta prioridade e 150 de baixa prioridade. A simulação foi realizada durante 3,4ms.

Nessa simulação não houve descarte, já que todas as células que foram chegando encontraram espaço livre no buffer. Houve uma reordenação das células de saída em função das suas prioridades. Os arquivos de saída do Arena e do VHDL apresentaram resultados iguais.

3. Nessa simulação foram simuladas 1800 células, com 3 classes diferentes de prioridades 0, 1 e 2. Foram geradas 600 células para cada classe, 300 de baixa prioridade (CLP = 1) e 300 de alta prioridade (CLP = 0). A simulação foi realizada durante 5ms. Como algumas células que chegaram encontraram o buffer com a capacidade máxima permitida preenchida, o algoritmo de descarte foi ativado. O modelo construído no Arena resultou em um arquivo de saída com 1220 células, enquanto o modelo em VHDL resultou em um arquivo de saída com 1217 células. O modelo VHDL descartou 3 células a mais que o ARENA, essas células pertenciam à classes de prioridade baixa. Outra diferença entre os dois arquivos foi que a terceira célula do VHDL era uma célula de classe de baixa prioridade, enquanto a terceira célula do Arena era uma célula de classe de alta prioridade. Esse resultado mostra diferenças entre os dois arquivos muito reduzidas em relação ao número de células que foram simuladas, e essas diferenças não iriam alterar significativamente os resultados da simulação. Por esse motivo não foi investigado mais a fundo a razão dessas pequenas diferenças.

4. Nessa simulação foram usadas duas fontes com tráfego constante sendo que cada uma delas gerou 400 células a taxa de 2,7μs .

Não foram perdidas células nas simulações do Arena e do VHDL. A diferença entre os dois arquivos de saída foi que a simulação VHDL se antecipou em 0.9μs para retirar a

primeira célula. Esta diferença é bem menor que o tempo de transmissão de uma única célula e por essa razão não iria afetar os resultados das simulações de desempenho.

5. Essa simulação usou duas fontes com o seguinte intervalo de geração de células 2.26E-4*(-0.001 + 280 * BETA(0.29, 8.73)) ms, sendo que cada uma gerou 450 células, das quais 74 foram descartadas pela *Fila Limitadora* antes de poder entrar no Buffer Multiclasse.

Nenhuma célula foi descartada nas duas simulações e os arquivos de saída obtiveram resultados iguais.

6. Nessa simulação foram simuladas 1000 células, sendo que 24 delas foram descartadas antes de entrarem no Buffer. Duas fontes com as seguintes características foram usadas: A fonte1 gerou 500 células com QoS = 0 e CLP = 1, com o seguinte intervalo entre células 2,26E-4*(-0,001 + 280 * BETA(0.29, 8.73)) ms, a fonte2 gerou 500 células, com QoS = 1, CLP = 1 e intervalo de geração de células de 0,0027ms.

Tanto no Arena quanto no VHDL nenhuma célula foi descartada, logo os respectivos arquivo de saída possuíam as mesmas 976 células.

7. Essa simulação usou o seguinte intervalo de geração de células 7.0E-4*LOGN(6.36, 3.23) ms. Foram simuladas duas fontes com essa taxa, as duas possuindo QoS = 0 e VPIVCI = 0, uma delas com CLP = 0 e a outra com CLP = 1.

Cada fonte simulou 2000 células, a simulação teve uma duração de 11ms. Foram descartadas 78 células, todas da Fonte2.

As diferenças entre o buffer simulado no VHDL e o buffer simulado no Arena foram que no VHDL foram descartadas duas células a mais que no Arena.

Como essas diferenças eram muito pequenas, elas não foram levadas em consideração.

5.3. Cenários de tráfegos

Essa seção mostra os cenários de tráfego que foram escolhidos para serem usados nas simulações do modelo do Buffer COMATM, e os motivos que levaram a suas escolhas.

Durante a fase de pesquisa, foram procurados tráfegos de Redes ATM, para serem usados nas simulações do modelo do buffer. Foram pesquisados vários artigos, não obtendo respostas de nenhum deles. Entre eles um que trabalhou com tráfego multimídia em Redes ATM [15].

Sem perspectiva de conseguir cenários de tráfego mais adequados, fomos procurar no âmbito do campus de Campina Grande e encontramos um vídeo muito usado para *benchmarks* de algoritmos de compressão, usado em testes de praticamente todos os padrões de vídeos existentes, denominado Claire e resultados de monitoramento de tráfego do POP-PB.

Os cenários de tráfegos usados estão caracterizados na seção abaixo.

5.3.1. Primeiro cenário

Esse primeiro cenário de tráfegos usou dados que foram capturados no PoP da RNP da Paraíba, e engloba todas as entidades acadêmicas ligadas a ela.

O tráfego foi capturado no dia 8 de Novembro de 2000, quarta-feira dia característico do tráfego típico do Campus II durante a semana.

Os dados coletados nesse arquivo contêm todas as informações de entrada e saída de todas as entidades que usam o backbone da RNP na Paraíba. Porém, para efeito de análise, o tráfego analisado nas simulações foi reduzido para o tráfego de entrada e saída do Campus II.

Durante a coleta foram medidos no POP os dados que entraram e saíram de um em um minuto durante 24 horas e foram gravados em um arquivo. Essa coleta de dados é feita através de um contador, que é incrementado com o número de bytes de dados. O contador é

lido a cada minuto. Esse contador só pode suportar 2^{32}-1 bits, quando esse valor é atingido ele é zerado e recomeça a contagem de bytes de dados.

Para usar o cenário de tráfego nas fontes de geração do modelo do Buffer no Arena, é necessário que exista um modelo para a taxa de geração de células e que ele seja colocado no CREATE para que este possa criar células. Essa taxa tanto pode ser constante quanto pode ser uma função. Como o tráfego do PoP é variável, é necessária uma função para gerar os dados. Para isso os dados foram convertidos e submetidos ao *Input Analyzer* que devolveu a função que mais se aproximava dessa taxa.

Esse processo de conversão dos dados para uma função que caracteriza os tempos entre envios de células consecutivas se deu da seguinte forma: Na captura dos dados de entrada e saída do campus II, os dados armazenados no contador do arquivo foram capturados diminuindo-se os dados gravados no minuto atual dos dados gravados no minuto anterior, tendo assim, a quantidade de bytes que chegaram no intervalo de tempo de um minuto.

Depois de todos os dados que chegaram de minuto em minuto terem sido capturados em um arquivo texto, calcula-se o tempo Y que 53 bytes levam para chegar, sabendo-se que X bytes (quantidade que chega em um minuto) levam 60000 milisegundos para chegar. Faz-se então uma regra de três: Y = (53*60000)/X. Esse processo foi realizado por um programa escrito na Linguagem C.

Esse procedimento gerou um arquivo com os tempos de chegada entre células em milissegundos. Esses dados de interchegada (Y) foram submetidos ao *Input analyzer* e este devolveu uma Função de Distribuição de Probabilidade que mais se encaixava para os dados de tempo de geração de células.

O gráfico gerado pelo Arena, com Função de Distribuição de probabilidades do tempo entre chegadas de células está ilustrado na figura 26.

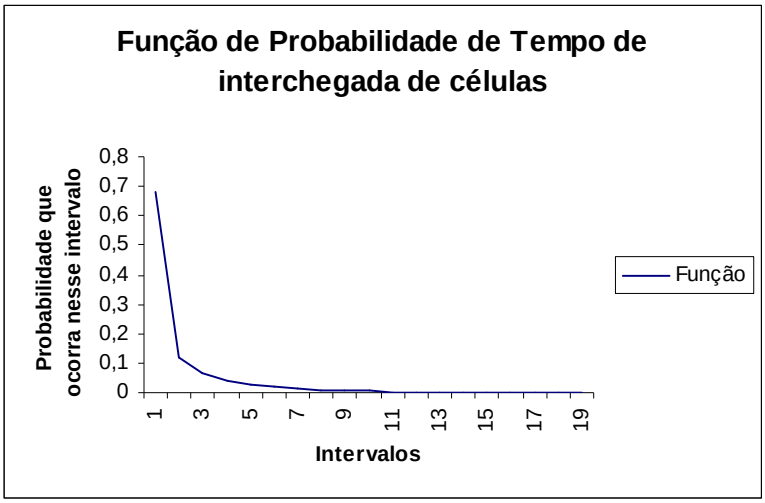

Figura 26 - Função de Distribuição de Probabilidade (Cenário1)

A função gerada pelo *Input analyzer* para esses dados foi -0.001 + 280 * BETA(0.29, 8.73).

5.3.2. Segundo Cenário

Esse cenário usou dados do vídeo Claire. O arquivo desse vídeo forneceu os dados de geração de células, o número de bits para cada quadro do vídeo com 30 quadros/segundo, codificado usando MPEG4 nível 1 com o código fonte disponível no site oficial do MPEG4 (http://www.cselt.it).

Esses dados foram convertidos para saber o tempo médio de interchegada de uma célula. Essa conversão foi realizada da seguinte forma: Se foram passados 30 quadros por segundo, um quadro deveria ser passado em 1/30 segundos. A quantidade de bits em um quadro será denominada Y. Portanto, Y bits são passados em 1/30 segundos. Como se deseja saber em quanto tempo devem ser recebidos 53 bytes, pode-se utilizar uma regra de três com cada valor de Y fornecido. Essa regra de três é da seguinte forma: X= (8*53)/(Y*30), sendo X o tempo para receber 53 bytes. Esses valores, convertidos pelo programa, foram colocados em um arquivo e submetidos ao *Input Analyzer*, para que o mesmo retornasse uma função de geração de células. Como já foi explicado na seção

anterior, o Arena precisa do tempo de geração de células em uma função. Essa função está representada na figura 27.

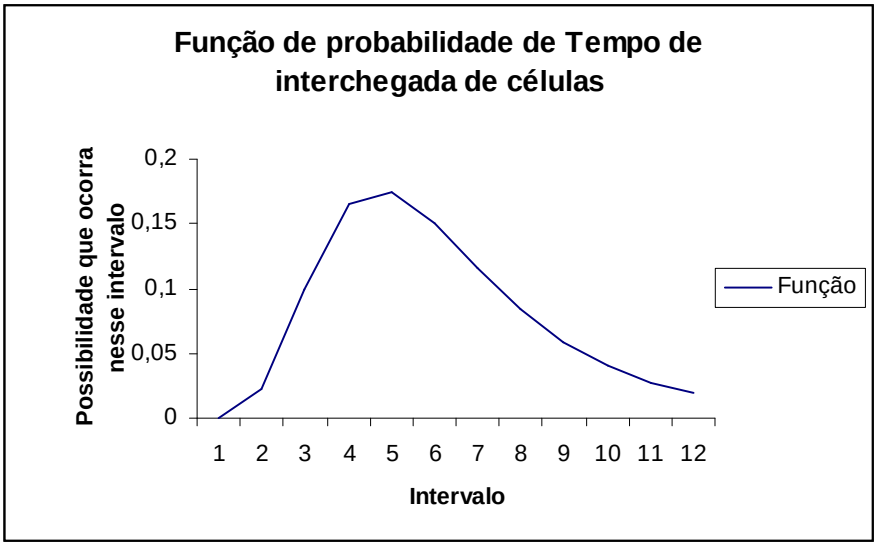

Figura 27 - Função de Distribuição de Probabilidades (Cenário2)

A função usada para indicar o tempo de interchegada de células é a seguinte: LOGN(6.36, 3.23).

5.3.3. Terceiro Cenário

Esse terceiro cenário possui dados que são gerados a uma taxa constante de uma célula a cada 2.7 μs, que é o tempo de geração de uma célula ATM com a taxa de 155Mbps. Esses dados são gerados de forma que não se têm rajadas ou períodos de inatividade, pois eles possuem uma taxa que não varia no tempo.

5.4. Simulações

Essa seção mostra as simulações que foram realizadas no modelo do Buffer Multiclasse, construído no Arena, bem como os resultados que foram obtidos delas.

As simulações foram realizadas em duas fases, uma com o algoritmo do buffer Multiclasse COMATM, e a outra com algoritmo FIFO. Foram realizadas 6 simulações com

o buffer Multiclasse e 6 simulações com o buffer FIFO com os mesmos dados. Cada simulação do buffer COMATM foi realizada com 5 replicações, foi calculado o intervalo de confiança usando um nível de confiança de 95%. Depois de prontas foram feitas comparações entre os dois buffers para saber qual obtinha o melhor desempenho, como em [16].

Para assegurar que foi feita uma comparação justa, o algoritmo foi implementado usando a mesma política de compartilhamento do buffer por múltiplas classes. A diferença que existe é a política dos algoritmos de envio e descarte de células. No algoritmo FIFO de envio as células são colocadas na fila na ordem em que chegam e a primeira célula é sempre enviada primeiro. No envio a última é sempre descartada quando o buffer está cheio.

Todas as simulações foram realizadas em um PC com a seguinte configuração: 192MB de Ram e um processador Pentium 3 com 600 MHz.

5.4.1. Primeira simulação

Essa primeira simulação usou o cenário do vídeo Claire que possui a seguinte taxa de interchegada de células LOGN(6.36, 3.23). Foram simuladas duas classes com essa taxa, as duas possuindo QoS = 0 e VPIVCI = 0, uma delas com CLP = 0 e a outra com CLP = 1.

Como as simulações usando unicamente a taxa do vídeo Claire não causaram descarte, para que houvesse uma perda em torno de até 10% de células, com isso podendo testar o algoritmo de descarte e a capacidade do buffer, foi criado um fator de escala na fonte de geração de células para que sua velocidade ultrapassasse 155Mbps e esse fator de escala foi multiplicado pelo intervalo de geração de células. Esse fator de escala causa descarta quando há rajada suficiente nos dados da simulação, e essa rajada vai depender da semente usada na replicação. O fator de escala utilizado foi 7.0E-4, que foi multiplicado pelo tempo de interchegada de cada célula. Portanto, o intervalo de geração de cada célula foi 7.0E-4*LOGN(6.36, 3.23) ms.

Na simulação do Buffer COMATM cada fonte simulou 2000 células, a simulação teve uma duração de 11ms e sua execução no Arena levou 38 horas. A quantidade mínima de células no buffer foi 0 e a quantidade máxima foram 600 células, com uma quantidade média de 298 células.

O gráfico da figura 28 mostra os atrasos das células do buffer COMATM, que foram agrupados de acordo com o intervalo de tempo de atraso no qual elas se encaixavam. As células foram agrupadas em intervalos de 100 microssegundos.

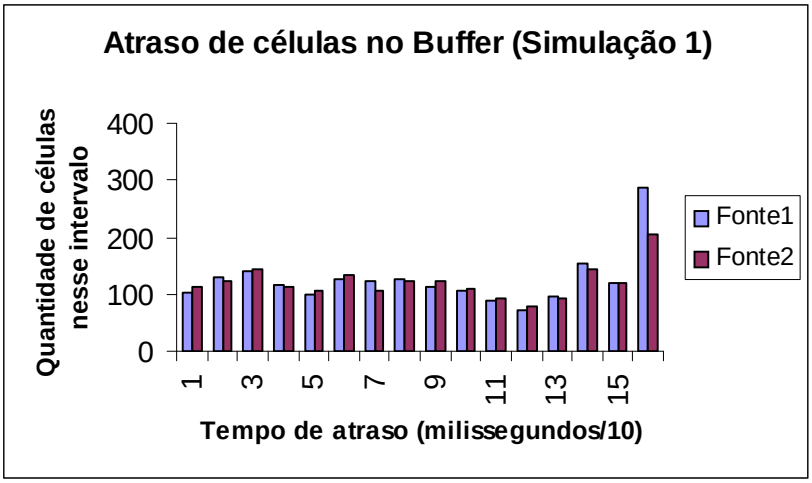

Figura 28 - atraso de células do buffer COMATM (simulação 1)

O gráfico mostra que no início as fontes 1 e 2 se alternam com maior ou menor quantidade de células que estão contidas nos intervalos de atraso. Esse fato se deve ao algoritmo de envio não atribuir prioridade para células de uma mesma classe no momento de recepção das mesmas e por isso são enviadas a mesma quantidade de células das duas fontes e também a não estar descartando células por não estar cheio. No momento de envio o CLP não é levado em consideração para não desordenar as células de uma mesma conexão virtual.

Quando o buffer se enche o algoritmo de descarte começa a executar. De acordo com a sua política, células de CLP = 0 possuem prioridade sobre as células de CLP = 1. Logo, quando as células começam a ser descartadas percebe-se que as células da fonte1 (CLP = 0) ficam mais tempo na fila, pois as da fonte2 (CLP = 1) vão sendo descartadas, e

isso aumenta o atraso global das células da fonte1, superando o atraso das células da fonte2.

A simulação do buffer usando o algoritmo FIFO simulou por 11,5 ms com os mesmos dados do buffer COMATM. A quantidade média de células na fila foi 288 células, com as quantidades máxima e mínima 600 e 0 células, respectivamente.

Foram descartadas 42 células da fonte1 e 40 células da fonte2. O atraso médio das células foi para a fonte1 = 844,5μs e para a fonte2 = 847,7μs.

O gráfico da figura 29 mostra uma comparação entre o atraso sofrido pelas duas fontes do buffer COMATM e as duas fontes do buffer FIFO.

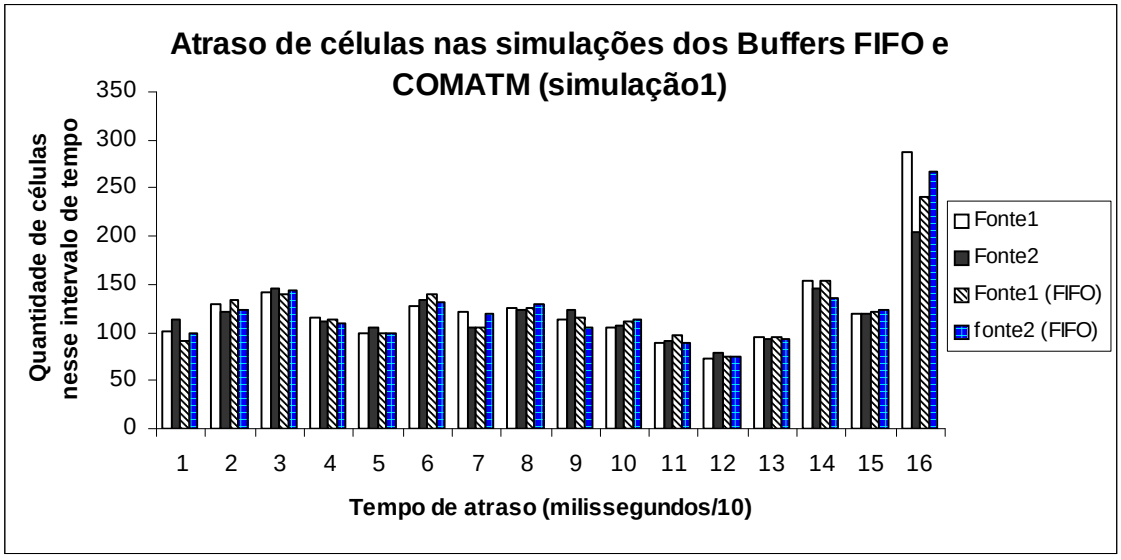

Figura 29 - Atraso de células dos Buffers FIFO e COMATM (simulação1)

Esse quadro mostra que as células do buffer COMATM possuem um atraso similar às células do algoritmo FIFO, quando são utilizadas fontes da mesma classe. A maior diferença se encontra no fato de algumas células da fonte2 do buffer COMATM terem sido descartadas e não ter sido descartada nenhuma célula da fonte1 e isso elevar em uma quantidade pequena o atraso das células da Fonte1 no último intervalo de tempo.

O quadro com a quantidade de perdas e atrasos de células está mostrado na figura 30. Esse quadro mostra que não há uma prioridade no momento de descartar células do buffer FIFO. Por isso, células que possuem maior prioridade (CLP = 1) são tão descartadas

quanto células de prioridade menor. Percebe-se também que foi descartado praticamente a mesma quantidade total de células do buffer FIFO que do buffer COMATM. Isso mostra que o algoritmo de descarte do buffer COMATM prioriza células de CLP = 0 (alta prioridade). O intervalo de confiança apresentado possui um valor de 30,5 células para as células descartadas e 49,6 e 38,6 microssegundos para o atraso. Isso acontece, como foi explicado anteriormente, devido a variabilidade das sementes usadas nas replicações de uma mesma simulação.

	Descart. COMATM	**Int. Confiança**	**Desc. FIFO**	**Atraso médio COMATM**	**Int. Confiança**	**Atraso médio FIFO**
Fonte1	-	0	42	743,9µs	49,6µs	844,5µs
Fonte2	15,6	30,5	40	752,6µs	38,6µs	847,7µs

Figura 30 - Atraso médio e células descartadas (simulação 1)

5.4.2. Segunda Simulação

Essa simulação utilizou o primeiro cenário, que são dados do PoP. Foram usadas duas fontes, com tempo de geração de cada célula de -0.001 + 280 * BETA(0.29, 8.73) ms, as duas sendo da mesma classe com QoS = 0, uma delas possuindo CLP = 0 e a outra CLP = 1. Cada fonte simulou 2000 células.

Como esses dados não possuem rajadas suficientes para causarem descarte, para que houvesse um descarte em torno de 0 a 10% de células foi usado um fator de escala de 3.9E-4, que foi multiplicado pelo tempo de interchegada de cada célula. Portanto, o tempo de interchegada final de cada célula foi 3.9E-4*(-0.001 + 280 * BETA(0.29, 8.73)) ms.

No modelo do buffer COMATM a simulação levou 11ms e executou no Arena durante 43 horas. A quantidade média de células no buffer foi 321 células.

O gráfico da figura 31 mostra o atraso das células das fontes 1 e 2, na forma de um histograma, que agrupa todas as células que contêm o atraso dentro de um certo intervalo de tempo.

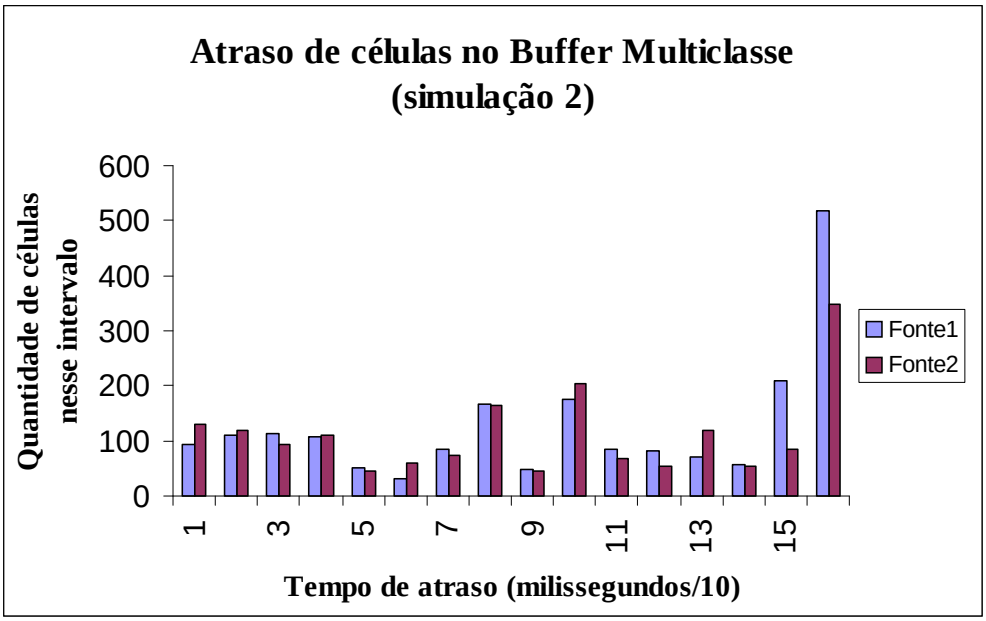

Figura 31 - Atraso de células do buffer COMATM (simulação 2)

Como as células das fontes 1 e 2 pertencem à mesma classe de prioridade, elas são tratadas da mesma forma pelo buffer no momento de recepção das células. Por isso, antes de ocorrer os primeiros descartes as duas fontes possuem uma quantidade de células equivalente nos intervalos de tempo. Porém, quando o buffer enche de células começa a ocorrer descarte de células da fonte2 por possuir CLP = 1. Isso faz com que nos últimos intervalos de tempo existam mais células da fonte1, já que algumas células da fonte2 foram descartadas. Como as células da fonte1 permanecem mais tempo na fila, o seu atraso médio aumenta.

A simulação realizada no buffer FIFO executou com os mesmos dados daquela realizada pelo buffer COMATM.

Nessa simulação (FIFO) foram descartadas 133 células da fonte1 e 136 células da fonte2, a quantidade média de células na fila foi 305 células. O atraso médio da fonte1 foi 934,9μs e da fonte2 foi 948,7μs.

A figura 32 mostra o gráfico das células atrasadas pelo buffer COMATM em comparação com as células atrasadas do buffer FIFO.

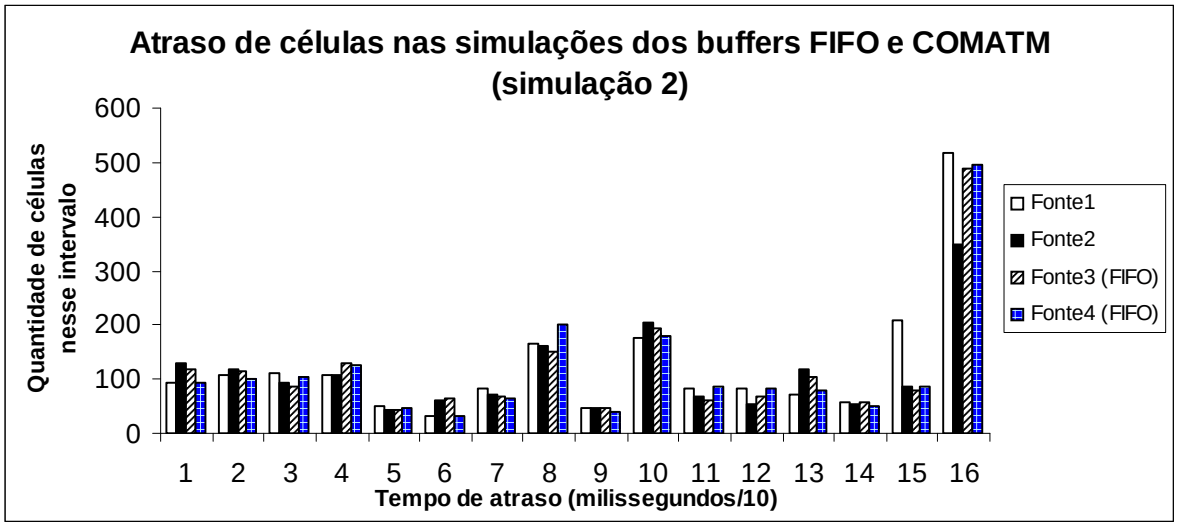

Figura 32 - Atraso de células dos buffers FIFO e COMATM (simulação 2)

Como na simulação anterior, o buffer COMATM possui uma concentração de células da fonte1 nos últimos intervalos de tempo devido ao descarte das células da fonte2. O buffer FIFO não possui esta diferença entre as duas fontes devido a descartar células das fontes de igual, sem atribuir prioridades.

A tabela da figura 33 mostra a comparação entre as células atrasadas e perdidas do buffer FIFO e as do buffer COMATM. Pela tabela percebe-se que a fonte2 do buffer COMATM possui um atraso menor devido aos descartes. Além disso as células descartadas do buffer FIFO se dividem entre as duas fontes.

	Descart. COMATM	**Int. Confiança**	**Desc. FIFO**	**Atraso médio COMATM**	**Int. Confiança**	**Atraso médio FIFO**
Fonte1	-	0	133	703,5µs	174,4µs	934,9µs
Fonte2	46	90,1	136	700,5µs	169,6µs	948,7µs

Figura 33 - Atraso médio e células descartadas (simulação 2)

5.4.3. Terceira Simulação

Essa simulação usou o terceiro cenário de tráfego, que possui um intervalo de geração de células constante. Foram simuladas 4000 células, sendo 2000 pela primeira e 2000 pela Segunda fonte. Ambas as fontes possuem QoS = 0, VPIVCI = 0, uma com CLP = 0 e a outra com CLP = 1.

Para forçar um descarte em torno de 0 a 10% de células foi usado um fator de escala que multiplicado a 2,7 μs resultou em um intervalo de geração de células de 4,3 μs .

A duração da simulação do buffer COMATM foi de 11ms e sua execução no Arena demorou 39 horas. A quantidade média de células no buffer foi 303 células.

O gráfico da figura 34 mostra as células agrupadas em intervalos de tempo, de acordo com o atraso que ela sofreu dentro do buffer. Esses intervalos de tempo são de 100 microssegundos.

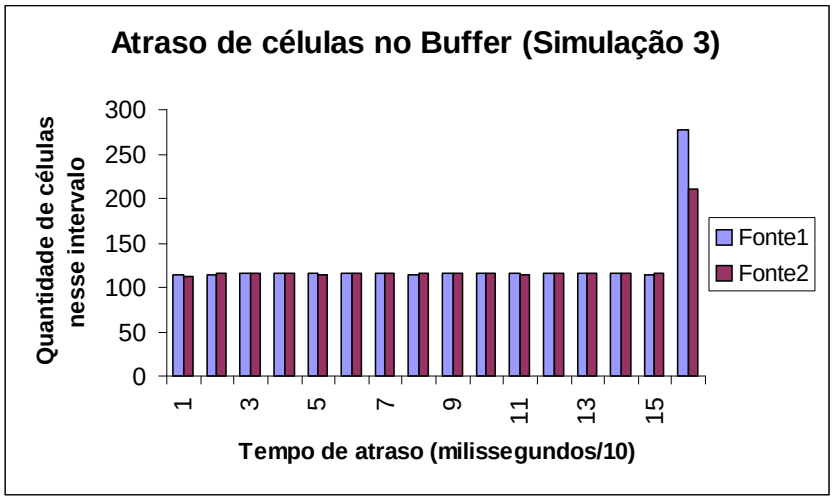

Figura 34 - Atraso de células do buffer COMATM (simulação 3)

O gráfico mostra que até o intervalo de tempo 15, as fontes 1 e 2 possuíam a mesma quantidade de células atrasadas compreendidas nos intervalos de tempo. Isso ocorre porque o tempo de geração das células é constante, pelas duas fontes pertencerem a mesma classe de prioridade e o algoritmo de recepção tratá-las de forma igual e pelo buffer ainda não

estar cheio, dessa forma não começando a descartar células. Quando o buffer se enche ele começa a descartar células e elas devem ser descartadas levando-se em consideração tanto a classe à qual pertence e o CLP que ela possui. Como as duas fontes são de mesma classe, e a fonte2 possui CLP = 1, então as células que serão descartadas primeiro serão aquelas da fonte2. Por isso, após o início do descarte, a partir do intervalo de tempo 15 a fonte1 possui mais células compreendidas nesse intervalo de tempo de atraso. Por esse motivo também o atraso médio das células da fonte1 é maior que o atraso médio das células da fonte2.

A simulação realizada no buffer FIFO executou com os mesmos dados da realizada pelo buffer COMATM. Foram descartadas 73 células da fonte1, a quantidade média de células na fila foi 293,3 células. O atraso médio da fonte1 foi 861,7μs e da fonte2 foi 838,3μs.

A figura 35 mostra o gráfico com o resultado dos tempos de atraso das células por fonte, agrupadas em intervalos de tempo de 100 microssegundos, das simulações do buffer COMATM juntamente com as simulações do buffer FIFO.

Como tanto as células descartadas do buffer FIFO quanto as descartadas pelo buffer COMATM foram todas da fonte2, percebe-se que em todos os intervalos de tempo as fontes equivalentes possuem uma quantidade de células semelhantes nos intervalos de tempo de atraso.

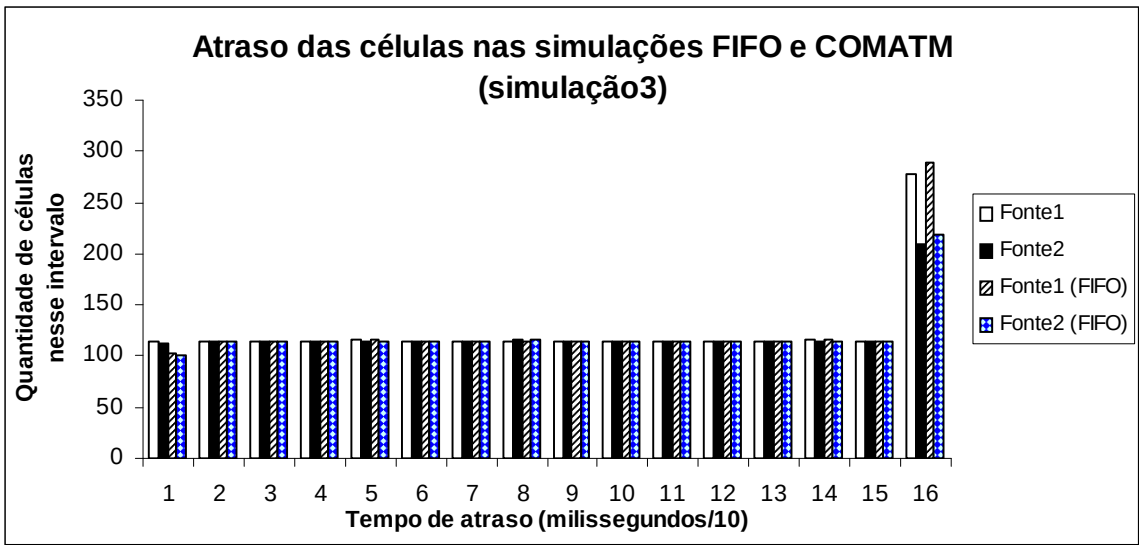

Figura 35 - Atraso de células dos buffers COMATM e FIFO (simulação 3)

O quadro da figura 36 mostra o atraso médio das células das duas fontes dos buffers COMATM e FIFO.

	Descart. COMATM	**Int. Confiança**	**Desc. FIFO**	**Atraso médio COMATM**	**Int. Confiança**	**Atraso médio FIFO**
Fonte1	-	0	-	899,7µs	0µs	871,05µs
Fonte2	107	0	73	863,9µs	0µs	846,5µs

Figura 36 - Atraso médio e células descartadas (simulação 3)

5.4.4. Quarta Simulação

Essa simulação utilizou os cenários 1 e 2, com 4 fontes para a geração de suas células. As fontes 1 e 2 se utilizaram do cenário do vídeo Claire, sendo configuradas com QoS = 0, VPIVCI = 0 e CLP's 0 e 1 respectivamente. Sendo que cada uma delas gerou 1000 células com o intervalo de geração de 13,5E-4*(LOGN(6.36, 3.23)) ms.

As fontes 3 e 4 usaram o cenário dos dados do POP, sendo que cada uma delas possui QoS = 1 e VPIVCI = 2. A fonte3 possui CLP = 0 e a fonte 4 CLP = 1. Cada fonte gerou 1000 células com um intervalo de 6.7E-4*(-0.001 + 280 * BETA(0.29, 8.73)) ms.

Para provocar o descarte de 0 a 10% de células para testar o algoritmo, foi usado um fator de escala em cada fonte.

A duração da simulação do buffer COMATM foi de 11,5ms e sua execução no Arena foi de 47.9horas. A quantidade média de células no buffer foi 329.

O atraso médio das células, agrupados por fonte foi o seguinte: fonte1 = 1,7μs, fonte2 = 1,7μs, fonte3 = 1717,02μs e fonte4 = 2056,02μs. Isso mostra que para células de classes diferentes existe uma diferença muito grande. Isso ocorre devido ao algoritmo de recepção de células tratá-las de forma diferente, colocando as células de classe mais prioritárias na frente das outras no buffer.

Para a representação dos dados em forma de gráfico foi necessário dividir os dados em dois gráficos porque os dados das fontes 1 e 2 não aparecem de forma legível no gráfico quando são agrupados com intervalos de 250 microssegundos. Para que fosse possível visualizar os dados das fontes 1 e 2 foi usado um gráfico com intervalos de microssegundos para essas duas fontes. Os dados das fontes 3 e 4 não aprecem no intervalo dos primeiros 5 microssegundos por não possuírem um atraso tão pequeno, e assim não estão inseridos no gráfico.

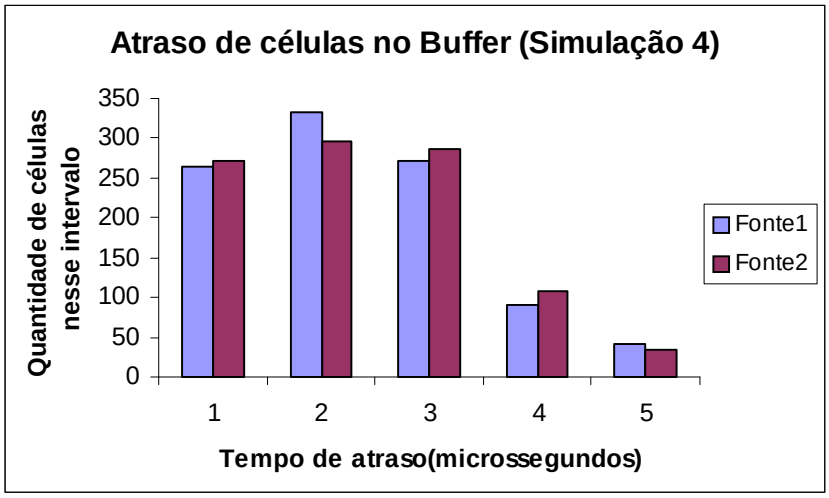

Figura 37 - Atraso de células do buffer COMATM (simulação 4)

O gráfico da figura 37 mostra os dados das fontes 1 e 2 agrupadas em intervalos de tempo de microssegundos. Como essas fontes são da mesma classe mais prioritária elas são tratadas da mesma forma pelo buffer e por isso possuem uma quantidade equivalente de células nos intervalos de atraso.

As fontes 3 e 4 da simulação do buffer COMATM estão agrupadas no gráfico da figura 38. Esse gráfico possui os dados agrupados em intervalos de 250 microssegundos.

O gráfico mostra que até o intervalo 13 as fontes 3 e 4 se alternam com relação à quantidade de células atrasadas nesses intervalos. Isso se deve às duas fontes serem da mesma classe, e no momento de colocar essas células na fila esse é o único atributo observado. A partir do momento que o buffer se enche e chegam mais células, ele começa a descartá-las, e esse descarte é feito levando-se em consideração o CLP das células, descartando primeiro as células da classe mais baixa e com CLP = 1. Logo, percebe-se que nos últimos intervalos de tempo desse gráfico existem muito mais células da fonte3, porque as células da fonte4 vão sendo descartadas por possuir QoS = 1 e CLP = 1.

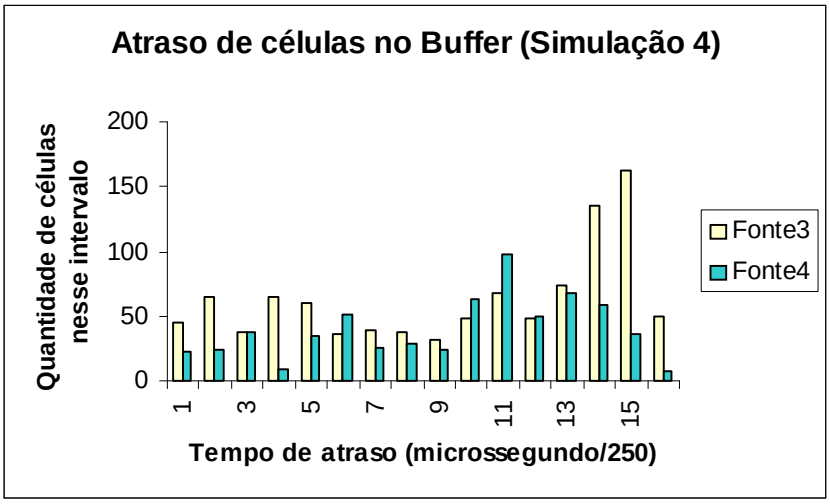

Figura 38 - Atraso de células do buffer COMATM (simulação 4)

A simulação do buffer FIFO utilizou os mesmos dados do buffer COMATM. Ela foi realizada em 11,5ms. Foram descartadas 62 células da fonte1, 61 células da fonte2, 116 células da fonte3 e 128 células da fonte4.

O atraso médio sofrido pelas células foi para a fonte1 = 1069,9µs, para a fonte2 = 1094,1µs, para a fonte3 = 898,9µs e para a fonte4 = 1113,2µs.

A quantidade média de células na fila foi 330,1 células.

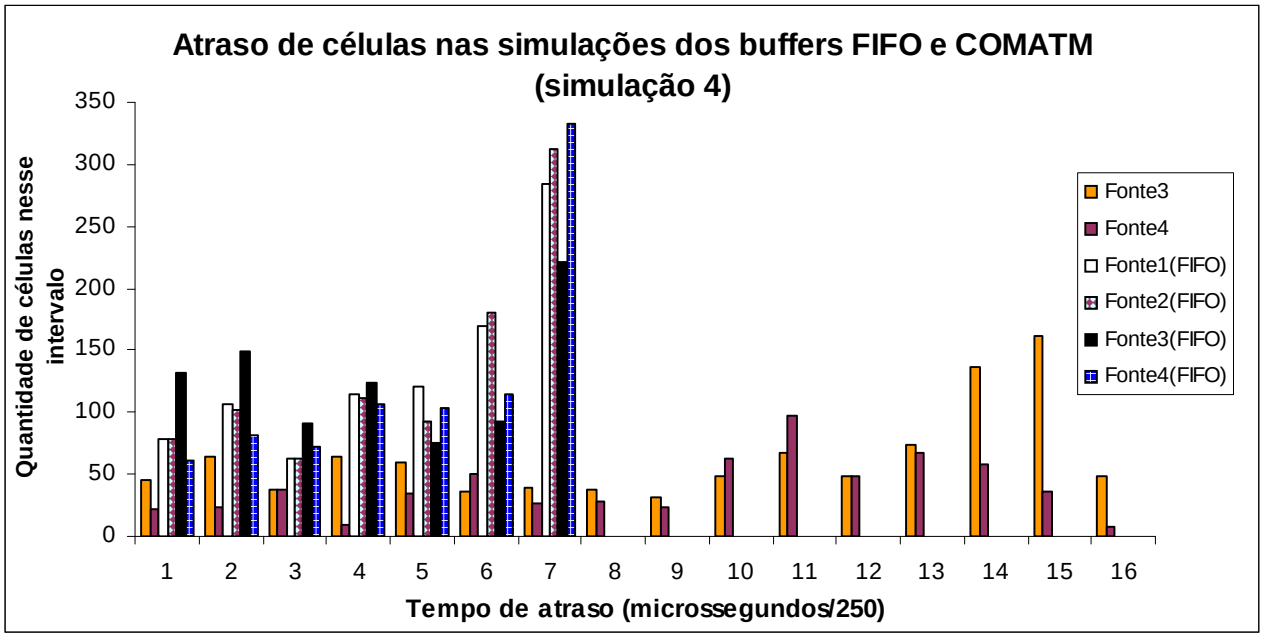

Figura 39 - Atraso de células dos buffers FIFO e COMATM (simulação 4)

Através do gráfico 39 pode-se perceber que as células das fontes 1, 2, 3 e 4 do buffer FIFO estão concentradas nos primeiros 7 intervalos de tempo, e possuem muito mais células concentradas nesses tempos que as células das fontes 3 e 4 do buffer COMATM. Isso ocorre porque o algoritmo de envio do buffer COMATM coloca as células de classe mais prioritárias na frente da fila e como as fontes 3 e 4 possuem o menor QoS elas estão sempre no final da fila, e têm que esperar mais tempo até que sejam enviadas. Por outro lado, no buffer FIFO as células são enviadas sem levar em consideração a classe à qual elas pertençam. Como as fontes 1 e 2 do buffer COMATM, que foram mostradas no gráfico da figura 37 demonstraram, o buffer COMATM possui um atraso muito menor que o buffer FIFO para as classes de prioridades altas e um atraso maior para células de classes de prioridades mais baixa, pois o buffer FIFO trata todas as células de forma igualitária. Isso pode ser visto também na tabela da figura 40 que mostra que o atraso médio das fontes 1 e 2 do buffer COMATM são muito menores que o atraso médio das fontes 1 e 2 do buffer FIFO e o atraso médio das fontes 3 e 4 do buffer COMATM são maiores que o atraso médio das fontes 3 e 4 do buffer FIFO. Isso mostra que não existe muita justiça com

relação às células das classes de prioridades mais baixas, pois elas sempre são descartadas ou deixadas no final da fila.

O quadro da figura 40 mostra aa média das células descartadas pelas replicações do buffer COMATM e os atrasos sofridos, assim como seus intervalos de confiança. Essa tabela mostra também as células descartadas e os atrasos sofridos pelas células do buffer FIFO. As células descartadas do buffer COMATM são todas de uma classe menos prioritária e as células descartadas pelo buffer FIFO são de todas as prioridades.

	Desc. COMATM	**Int. Confiança**	**Desc. FIFO**	**Atraso médio COMATM**	**Int. Confiança**	**Atraso médio FIFO**
Fonte1	-	0	62	1,7µs	0,05µs	1069,9µs
Fonte2	-	0	61	1,7µs	0,05µs	1094,1µs
Fonte3	-	0	116	1717,02µs	152,1µs	898,9µ
Fonte4	192,8	92,5	128	2056,02µs	102,04µs	1113,2µs

Figura 40 - Atraso médio e células descartadas (simulação 4)

5.4.5. Quinta Simulação

Foram simuladas quatro fontes, duas delas utilizando o cenário de dados do Vídeo Claire e duas utilizando tráfego constante. Foram utilizados fatores para aumentarem a taxa de envio de células e fazer o teste do algoritmo de descarte, pois as rajadas geradas pelos dados utilizados não causaram descarte. Os fatores, multiplicados pelos intervalos de geração de células foram para as fontes 1 e 2 iguais a 14E-4*(LOGN(6.36, 3.23)) ms e para as fontes 2 e 3 iguais a 0,0086 ms.

As fontes 1 e 2 geraram 1000 células, todas com QoS = 0, VPIVCI = 0, sendo que a fonte1 com CLP = 0 e a fonte2 com CLP = 1.

A duração da simulação do buffer COMATM foi de 11,5 ms, com um tempo de execução no Arena de 40,6 horas. A quantidade média de células no buffer foi 293,40 células.

O atraso das células separadas por um determinado intervalo de tempo no buffer foi representado em gráficos, e para uma melhor visualização foram feitos 2 gráficos, o primeiro com as fontes 1 e 2 e o segundo com as fontes 3 e 4.

O gráfico da figura 41 mostra o atraso de células no buffer das fontes 1 e 2, agrupadas por quantidades de células atrasadas. O intervalo de tempo utilizado foi microssegundos. Nenhuma célula das fontes 3 e 4 possuem o atraso contido nos primeiros 7 microssegundos e por isso não aparecem no gráfico.

Como o algoritmo de inserção de células no buffer prioriza as células de classe mais prioritária e não leva em consideração o CLP da célula para não desorganizar a conexão virtual de células com o mesmo VPIVCI, as células das fontes 1 e 2, que possuem QoS = 0 são tratadas da mesma forma, sempre colocadas na frente do buffer e enviadas primeiro.

Por isso a quantidade de células atrasadas das duas fontes, por intervalo, são mais ou menos equivalentes.

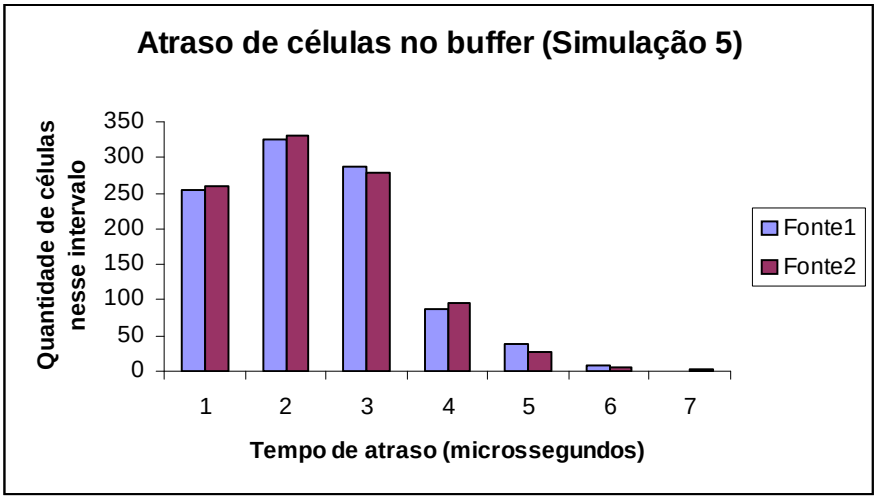

Figura 41 - Atraso de células do buffer COMATM (simulação 5)

O gráfico da figura 42 mostra o atraso das células das fontes 3 e 4 agrupadas em intervalos de 250 microssegundos.

Percebe-se que nos intervalos de tempo 8 e 9 existem mais células da fonte3 que células da fonte4. Isso acontece porque nesses intervalos houve rajadas da fonte3 quando o buffer estava cheio. Isso se deve também ao descarte de células da fonte4. Como a fonte4 é a que possui a menor classe e seu CLP é igual a 1, as células escolhidas para o descarte são dessa fonte.

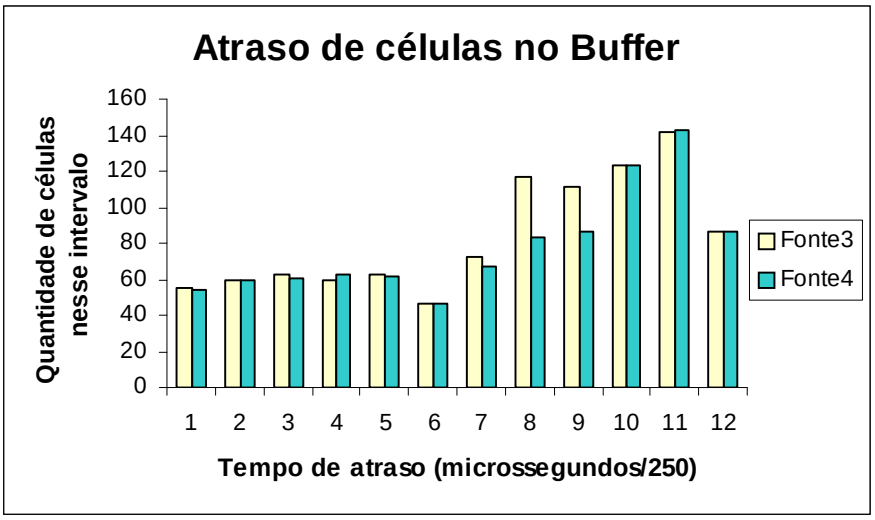

Figura 42 - Atraso de células do buffer COMATM (simulação 5)

A simulação do buffer FIFO foi realizada em 11,5 ms e utilizou os mesmos dados da simulação do buffer COMATM. Foram descartadas das fontes: fonte1 = 17, fonte2 = 17, fonte3 = 8 e fonte4 = 24 células.

O atraso médio sofrido pelas fontes foi de fonte1 = 875,1µs, fonte2 = 861,1µs, fonte3 = 859,8µs e fonte4 = 850,03µs.

O gráfico da figura 43 mostra que as fontes 3 e 4 do buffer COMATM, pelo fato de serem de classe de baixa prioridade, permanecem mais tempo no final da fila e possuem muito mais células com intervalos de atrasos maiores que o buffer FIFO, que trata todas as classes de forma igual.

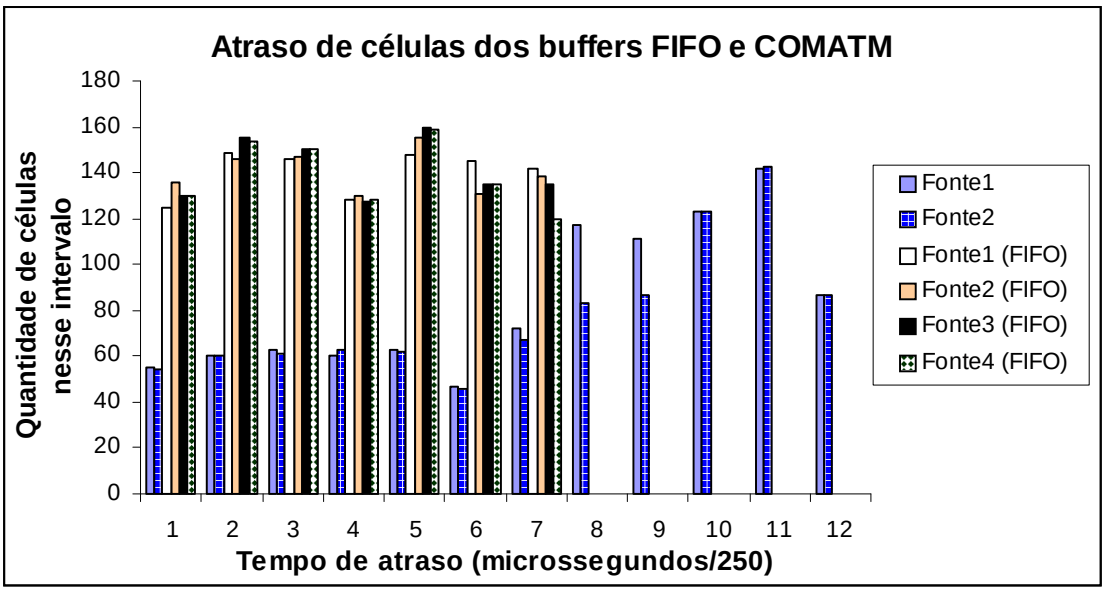

Figura 43 - Atraso de células dos buffers FIFO e COMATM (simulação 5)

A tabela da figura 44 mostra o atraso médio e as células descartadas das simulações do buffer COMATM e do buffer FIFO.

A quantidade de células descartadas pelas duas fontes é similar, no entanto o buffer COMATM descarta células da classe menos prioritária, enquanto o buffer FIFO descarta células de todas as classes.

	Desc. COMATM	**Int. Confiança**	**Desc. FIFO**	**Atraso médio COMATM**	**Int. Confiança**	**Atraso médio FIFO**
Fonte1	-	0	17	1,7µs	0,04µs	875,1µs
Fonte2	-	0	17	1,78µs	0,03µs	861,1µs
Fonte3	-	0	8	1709,5µs	88,27µs	859,8µs
Fonte4	49,8	32,45	24	1704,7µs	83,63µs	850,03µs

Figura 44 - Atraso médio e células descartadas (simulação 5)

5.4.6. Sexta Simulação

Essa simulação usou 6 fontes, sendo que duas delas usaram os dados do vídeo Claire, duas os dados do POP e as outras duas dados de fontes constantes. Todos os intervalos de geração de células foram multiplicados por fatores de escala para aumentar a taxa de envio de células.

Foram utilizadas 6 fontes. As fontes 1 e 2 possuem QoS = 0, VPIVCI = 0, com intervalo de geração de células de 21E-4*(LOGN(6.36, 3.23)) ms, cada uma gerando 670 células. Sendo que a fonte1 possui CLP = 0 e a fonte2 CLP = 1.

As fontes 3 e 4 possuem QoS = 1, VPIVCI = 2 e intervalo de geração de células de 12E-4*(-0.001 + 280 * BETA(0.29, 8.73)), cada uma com 670 células. A fonte3 possui CLP = 0 e a fonte4 CLP = 1.

As fontes 5 e 6 possuem QoS = 2, VPIVCI = 4 e taxa de geração de células de 0,012ms, sendo que cada uma delas gerou 670 células. A fonte5 possui CLP = 0 e a fonte6 CLP = 1.

Através da QoS de cada fonte, que indica a classe de prioridade à qual a célula pertence, pode-se perceber que o Vídeo Claire possui prioridade sobre os dados do POP e esse por sua vez possui uma classe mais prioritária que os dados constantes.

A simulação do buffer COMATM teve a duração de 11,5 ms, e a sua execução no Arena foi de 42 horas. A quantidade média de células no buffer foi 292 células.

As células foram agrupadas em intervalos de tempo, de acordo com o atraso que elas sofreram dentro do buffer, e de acordo com a fonte à qual elas pertençam. Para um melhor efeito de visualização esses dados foram divididos em 3 gráficos, porque em um único não dava para perceber a quantidade de células que estava em cada intervalo.

Esse primeiro gráfico (figura 45) mostra os 6 primeiros microssegundos de simulação. Nenhuma célula das fontes 3, 4, 5 e 6 obtiveram um atraso tão pequeno dentro

do buffer e por isso mesmo não aparecem no gráfico. Esse gráfico mostra as fontes 1 e 2 com as respectivas quantidades de células contidas nesse intervalo de tempo.

O gráfico mostra que as fontes 1 e 2 possuem razoavelmente a mesma quantidade de células nos mesmos intervalos de tempo. Isso porque elas pertencem à mesma classe e são tratadas de forma igual pelo algoritmo de envio.

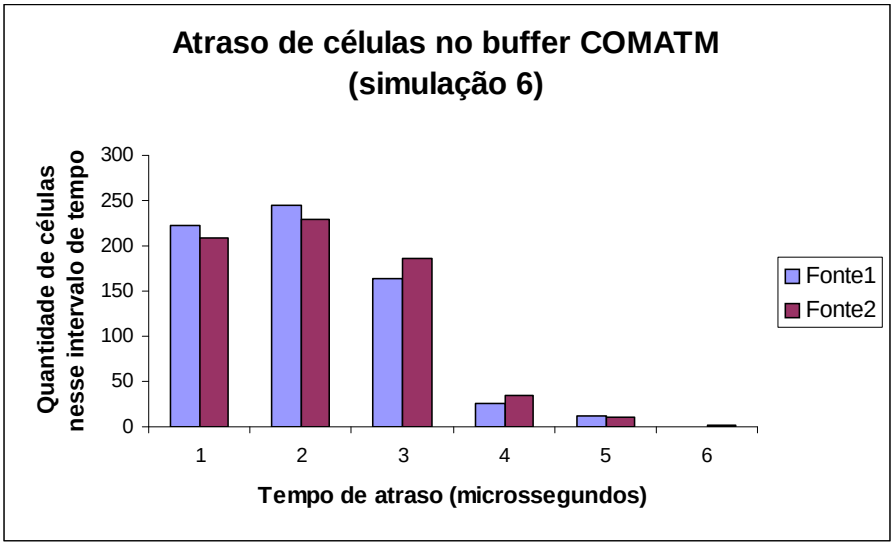

Figura 45 - Atraso de células do buffer COMATM (simulação 6)

O gráfico da figura 46 mostra o atraso das fontes 3 e 4 agrupadas em intervalos de 10 microssegundos. Como nenhuma célula das fontes 5 e 6 possuem células com atraso de até 70 microssegundos, elas não aparecem no gráfico.

As fontes 3 e 4 possuem um atraso semelhante entre si. Isso se deve a elas pertencerem à mesma classe.

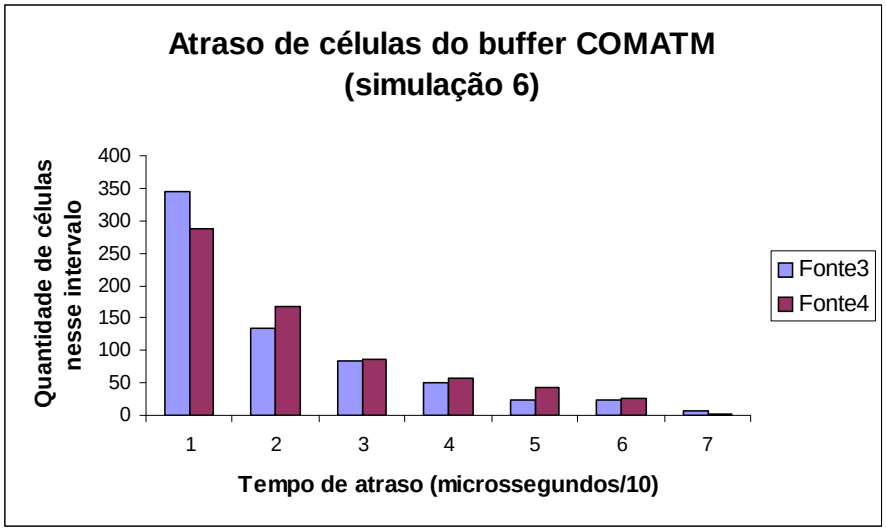

Figura 46 - Atraso de células do buffer COMATM (simulação 6)

O gráfico da figura 47 mostra os atrasos das fontes 5 e 6 agrupadas em intervalos de 250 microssegundos. O gráfico mostra que nos intervalos de tempo 10, 11, 12 e 13 houver muito mais células da fonte 5 em relação à fonte6. Esse fato ocorreu devido ao algoritmo de descarte levar em consideração tanto a classe à qual a célula pertença quanto o seu CLP. Esse algoritmo descarta primeiro as células da classe mais baixa e que possua CLP = 1, e como a fonte 6 é a menos prioritária e seu CLP = 1, todas as células descartadas são dessa fonte. Logo a fonte5 possui mais células que a fonte6, agrupadas em alguns intervalos de atraso.

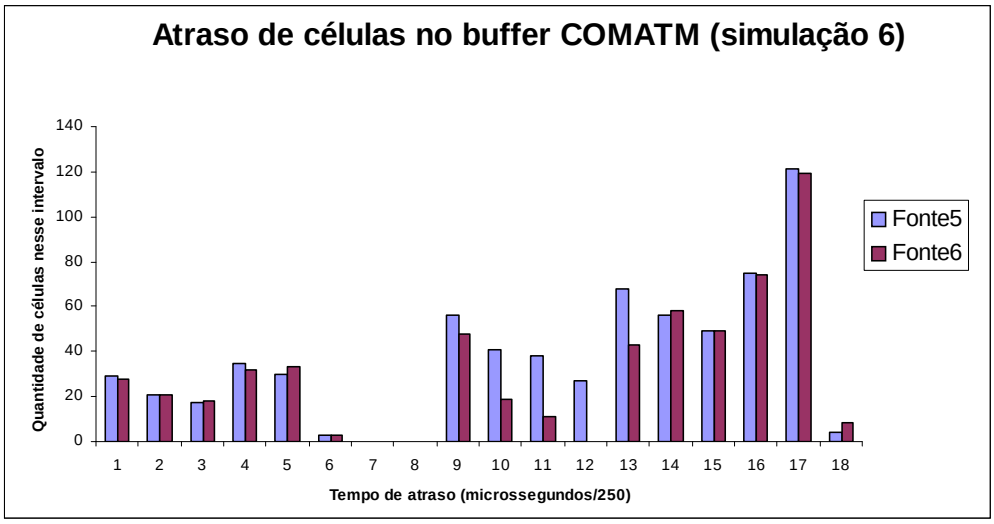

Figura 47 - Atraso de células do buffer COMATM (simulação 6)

A simulação no buffer FIFO foi realizada com os mesmos dados da simulação do buffer COMATM. Foram descartadas 106 células, distribuídas pelas fontes. Essa distribuição, assim como o atraso médio das células por fonte de geração estão mostradas na tabela da figura 51.

A figura 48 mostra a comparação entre os tempos de atraso das simulações realizadas nos buffers FIFO e COMATM. Através desse gráfico percebe-se que o buffer COMATM dá muito mais prioridades para células de classes de prioridades altas que o buffer FIFO. Por isso que as células do buffer COMATM estão agrupadas no primeiro intervalo de tempo e as do buffer FIFO estão compreendidas nos 7 intervalos de tempo de 250 microssegundos.

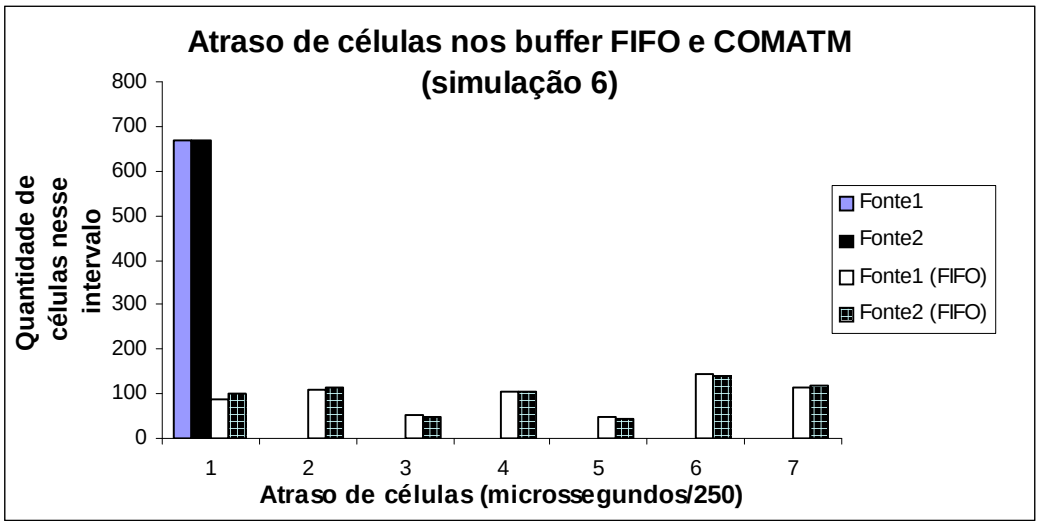

Figura 48 - Atraso de células dos buffers FIFO e COMATM (simulação 6)

A figura 49 mostra a comparação entre as células das fontes 3 e 4 dos buffers FIFO e COMATM. Nesse gráfico, como no anterior percebe-se que, como as fontes 3 e 4 são de classe 1, elas são tratadas de forma prioritária no buffer COMATM e não no buffer FIFO, por isso a enorme diferença entre as fontes do modelo do buffer.

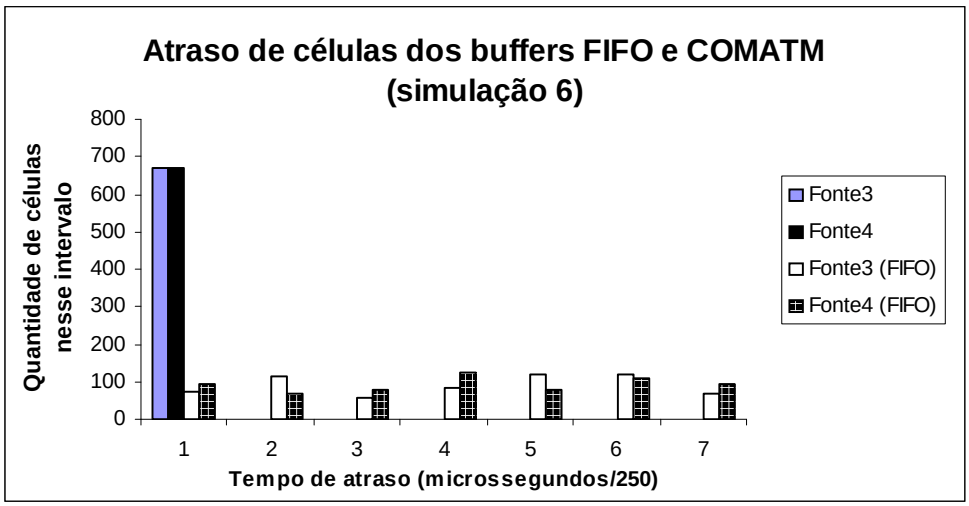

Figura 49 - Atraso de células dos buffers FIFO e COMATM (simulação 6)

O gráfico da figura 50 mostra a comparação entre os atrasos sofridos pelas células das fontes 5 e 6 nos buffers FIFO e COMATM.

Pode-se perceber através desse gráfico que as células das fontes 5 e 6 do buffer FIFO estão agrupadas nos primeiros 7 intervalos de tempo, enquanto as do buffer COMATM estão compreendidas nos 18 intervalos de tempo. Isso acontece porque como o buffer COMATM dá prioridade para as células de classe mais prioritária e as células das fontes 5 e 6 são menos prioritárias elas são sempre deixadas no final da fila e possuem um atraso maior do que as fontes do buffer FIFO que não possuem o conceito de prioridade.

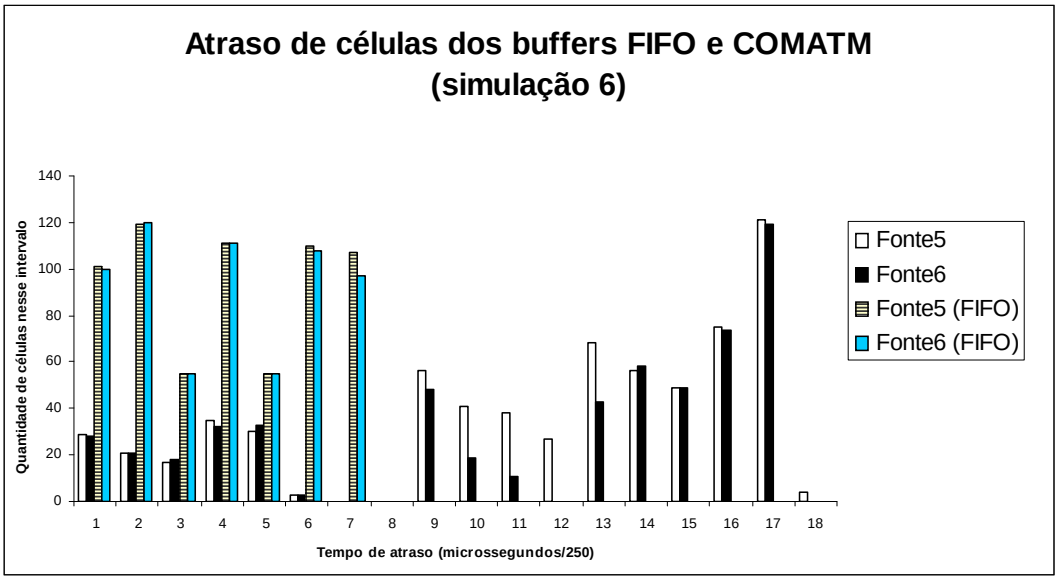

Figura 50 - Atraso de células dos buffers FIFO e COMATM (simulação 6)

A tabela da figura 51 mostra o atraso médio e as células descartadas pelos buffers FIFO e COMATM.

Através dessa tabela é possível notar que o buffer COMATM leva em consideração a classe à qual a célula pertença, enquanto o buffer FIFO descarta célula sem levar esse fato em consideração.

A tabela evidencia novamente o conceito de justiça, pois as classes de baixa prioridade do buffer COMATM possuem um atraso muito maior que as da classe de alta prioridade.

	Desc. COMATM	Int. Confiança	Desc. FIFO	Atraso médio COMATM	Int. Confiança	Atraso médio FIFO
Fonte1	-	0	12	1,48µs	0,07µs	925,7µs
Fonte2	-	0	10	1,5µs	0,06µs	914,7µs
Fonte3	-	0	30	14,76µs	1,7µs	890,3µs
Fonte4	-	0	18	14,56µs	1,3µs	902,2µs
Fonte5	-	0	12	2523,4µs	338,3µs	875,9µs
Fonte6	14,4	17,35	24	2546,3	358,9µs	864,8µs

Figura 51 - Atraso médio e células descartadas (simulação 6)

Capítulo 6

6. Conclusões

Comutadores ATM não têm sua implementação física definida pelos padrões de Redes ATM. Por isso, uma quantidade de pesquisa muito grande tem sido feita para explorar várias alternativas de construções para eles. Cada construção tem seus próprios méritos e inconvenientes em termos de vazão, atraso, escalabilidade, aproveitamento do buffer e tolerância a faltas. Examinado as características dos projetos básicos de comutação, muitas conclusões podem ser realizadas sobre os princípios de projetos de comutadores ATM [25].

No comutador COMATM, o buffer foi implementado usando algoritmos de recepção e envio de células, baseados com múltiplas classes, em compartilhamento de memória e na priorização de células de classe mais alta. Devido a ser um algoritmo ainda não testado, foi necessário fazer uma avaliação de desempenho para saber como ele iria se comportar diante dos tráfegos que lhe fossem impostos.

Foi então construído um modelo do buffer do Comutador COMATM no simulador ARENA que se ajustasse da forma mais fiel possível ao buffer original. A partir daí foram estudados tráfegos e estes foram submetidos ao buffer em simulações que avaliaram o desempenho do buffer COMATM.

Foram analisados os tempos de atraso e a quantidade de células perdidas no buffer. Foram levadas em consideração para essa análise algumas características de aplicações multimídia. De acordo com [17], as aplicações multimídia têm um tempo de transmissão e uma perda mínima que necessitam ser respeitados. No caso de uma conversa de telefone, é necessário um atraso menor que 0,3 segundos, para que seja estabelecida uma conversação normal. Aplicações práticas com sistemas de conferências multimídia e padrões ITU

sugerem um total máximo de atraso fim-a-fim de 150ms para aplicações de vídeo interativo.

Como as pontes, roteadores e comutadores introduzem um atraso, além daquele de compressão, descompressão, atraso de envio e delay da rede, é deixado um atraso para as aplicações multimídia de 10 a 15 ms em cada *hop* [17].

De acordo com as simulações realizadas no buffer COMATM percebe-se que nenhuma das simulações gerou um atraso maior que 3 ms. Como o buffer possui um tamanho de 600 células, de acordo com os dados apresentados em [17] e com o atraso resultante das simulações, o buffer poderia ter seu tamanho aumentado de 3 a 5 vezes que ainda assim poderia atender aos requisitos das aplicações multimídia.

Quanto à quantidade de células descartadas, não é possível fazer uma previsão fiel com relação à quantidade de células que o buffer poderia descartar caso fosse usado um tráfego específico de Redes ATM, pois as células que foram descartadas são provenientes de simulações com tráfegos multiplicados por fatores de escala e ainda as rajadas dos tráfegos são menores que as rajadas que algumas aplicações poderiam gerar em uma rede ATM. No entanto, através das simulações realizadas no modelo do buffer usando os algoritmos de recepção e envio FIFO percebe-se que o modelo do buffer COMATM descarta sempre o mesmo número ou uma quantidade inferior de células em suas simulações. Porém o buffer COMATM sempre descarta células da classe de menor prioridade, o que favorece as aplicações que possuem requisitos mais exigentes que as demais.

Um aspecto observado nas simulações do buffer Multiclasse é que o tempo de envio das células difere de acordo com a classe à qual elas pertencem. As classes de maior prioridade possuem um tempo de envio pequeno, quando comparado com as simulações do buffer FIFO. No entanto, as células de classe e prioridade menores possuem um atraso alto com relação às demais e um atraso maior que no caso FIFO.

Esse aspecto do buffer é muito bom para as classes de maior prioridade, no entanto as de prioridade mais baixa ficam bastante prejudicadas com este fato. Existe uma política

de justiça que deve atender todas as aplicações de acordo com as suas necessidades. Essa justiça que deve ser aplicada a todas as células é denominada *fairness*. No caso do buffer COMATM, quando é simulada mais de uma classe percebe-se que há um aumento muito significativo nos atrasos das células. As simulações 4, 5 e 6 ilustram essa questão.

Uma sugestão para trabalhos futuros seria implementar um algoritmo de envio onde houvesse uma política voltada para o *fairness*, caso as aplicações de classe mais baixa comecem a ter seus requisitos mínimos não atendidos. Essa política poderia ser implementada, usando uma técnica chamada *aging*, colocando um contador em cada célula, e esse contador seria incrementado a cada intervalo de tempo do relógio, quando esse tempo excedesse um determinado limite pré-determinado essa célula deveria ganhar prioridade sobre as demais e ser colocada na frente da fila para ser a próxima a ser enviada.

Outra sugestão para trabalhos futuros seria estudar cenários de tráfegos obtidos de uma rede ATM real em funcionamento, sem a necessidade de utilizar fatores de escala e fazer mais simulações usando esses tráfegos. A partir daí deve-se verificar os resultados e compará-los com os apresentados nesse trabalho.

Referências Bibliográficas

[01] **Tanenbaum, A. S.** *Computer Networks*, terceira Ed., New Jersey: Prentice Hall, 1996.

[02] **Lima, J. A. Gomes de,** *Um Controlador Microprogramável para Comutadores ATM*, Campina Grande, 1999. Tese (doutorado em Engenharia Elétrica) - CCT, UFPB.

[03] **Kelton, W. David**, *Simulation with Arena* WCB/McGraw-Hill, 1998.

[04] **Melcher, Elmar U. K., Naviner, Lírida A. De B., Leite, Lavoisier, Giozza, William F.; Monteiro, José A. S.,** *ATM Switch Buffer Controller based on linked lists suited for hardware implementation*, SBRC 97, São Carlos, Maio de 1997.

[05] **Takus, David A., Profozich, David M.,** *Arena Software Tutorial*, Pensylvania, System Modeling Corporation, 1997.

[06] **Alves, Carlos A. C., Wagner, Marcus V. da S.,** *Avaliação do ambiente Arena para a construção de Simuladores de redes ATM*, VII Seminfo, UFBA, 2000.

[07] **Silva, H.S., Monteiro, J. A. S.,** *Arquitetura Funcional do Comutador ATM*, Protem/Projeto COMATM (UFPb/UFPe/USP) Relatório Técnico RT 01/95, versão 1.0, Campina Grande, 1995.

[08] **Geoffrey, Gordon,** *System Simulation*, Segunda Edição, New Jersey, Prentice-Hall, 1978.

[09] **Silva, H.S., Monteiro, J. A. S.,** *Concepção da Tabela de estados das conexões para o comutador ATM*, Protem/Projeto COMATM (UFPb/UFPe/USP) Relatório Técnico RT 01/95, versão 1.0, Campina Grande, 1995.

[10] **Farias, L. J. Leite**, *Implementação de um controlador de Buffer Multiclasse para um comutador ATM usando VHDL*, Projeto de iniciação Científica, Campina Grande, 1997.

[11] **Monteiro, J. A. S.,** *Redes Digitais de Serviços Integrados de Faixa Larga (RDSI-FL)*, Recife, Maio de 1994.

[12] **Alles, Anthony**. *ATM Internetworking. Engineering InterOp*, Las Vegas, Março, 1995. Online, www.cisco.com.br.

[13] **Moraes, L. F. Magalhães de.** *Arquiteturas de Comutadores ATM*, Rio de Janeiro, Laboratório de Alta Velocidade, COPPE/UFRJ, 1997.

[14] **Karol, Mark J., Hluchyj, Michael G. Morgan, Samuel P.**, *Input Versus Output Queueing on a Space-Division Packet Switch*, IEEE Trans. On Comunication, Vol.COM-35, N.12, p.1347-1356, Dezembro 1987.

[15] **Nong, G., Muppala, Jogesh K., Hamdi, M.**, *Analysis of Nonblocking ATM Switches with Multiple Input Queues*, IEEE/ACM Trans. On Networking, Vol., N.01, Fevereiro 1999.

[16] **Lizambri, T., Duran, F., Wakid, S.**, *Priority Scheduling and Buffer Management for ATM Traffic Shapping*, proceedings do 17 workshop IEEE em Tendências Futuras de Sistemas de Computação Distribuídos, 1998.

[17] **Stuttgen, Heinrich J.**, *Network Evolution and Multimedia Communications*, IEEE Multimedia, vol 2, N. 3. Outono 1995.

[18] **Hwang, W., Chen, W., Deng, Y.**, *A High-Performance ATM Switch with Completely and Fairly Shared Buffers*, Proceedings da conferência internacional em sistemas paralelos e distribuídos, IEEE, 1997.

[19] **Lea, C.**, *Buffered or Unbuffered: A case Study Based on* $Log_d(N, e, p)$, IEEE Transaction on Communication, vol.44, N.1, Janeiro, 1996.

[20] **Chao, H, Jonathan**, *An ATM Queue Manager Handling Multiple Delay and Loss Priority*, IEEE/ACM Transactions on Networking, vol.3, N.6, Dezembro 1995.

[21] **McKeown, N., Mekkittikul, A., Anantharam, V., Walrand, J.**, *Achieving 100% Throughput in a Input-Queued Switch*.IEEE Transactions on Communications, vol. 47, N.8. Agosto de 1999.

[22] **Chatteerjee, S., Xiao, W.**, *Selective Discarding with FEC for ATM-based Internetworks*, Proceeding da 22° Conferência em Redes Locais de computadores, IEEE, 1997.

[23] **Habara, K.**, *ISDN: A Look at the Future Through the Past*, IEEE Communicationa Magazine, v.26, pp. 25-32, Novembro de 1998.

[24] **Simjava** - http://www.dcs.ed.ac.uk/home/hase/simjava

[25] *A survey of ATM Switching Techniques* - http://www.cis.ohio-state.edu/~jain/

[26] **Wagner, Marcus V. da S.,** *Especificação de componentes para a Simulação de Redes TCP/IP*, Campina Grande, 2000. Dissertação (Mestrado em Ciências da Computação) - CCT, UFPB.

Apêndice A

A. Programas usados

Esse apêndice apresenta o programa usado na integração do Visual Basic com o Arena para fazer a simulação.

Esse programa foi feito na Linguagem de programação Visual Basic e faz a integração dessa Linguagem com o Arena para realizar algumas funções que não podem, ou podem fracamente, serem realizadas no Arena.

Essa primeira função introduz as células na Fila Limitadora, que serve para limitar a velocidade de entrada de células no buffer a 0.9μs. Além disso, ela serve para registrar qual o número de sequência da célula entre as que estão sendo geradas. Cada vez que uma célula é gerada, o contador *inc* é aumentado de uma unidade. Esse contador é passado para o Arena, e o componente WRITE grava em um arquivo. Isso faz com que cada célula possua um número único de identificação.

Private Sub VBA_Block_3_Fire( )

```
Dim s As SIMAN
Set s = ThisDocument.Model.SIMAN
s.EntityInsertIntoQueue s.ActiveEntity, s.SymbolNumber("Queue2")
s.VariableArrayValue(s.SymbolNumber("inc")) = inc1
inc1 = inc1 + 1
```

End Sub

Esse Segunda função é usada para colocar as células nas filas do buffer, das portas de saída do comutador.

```
Private Sub VBA_Block_1_Fire( )
    Dim s As SIMAN
    Set s = ThisDocument.Model.SIMAN
    s.EntityInsertIntoQueue s.ActiveEntity, s.SymbolNumber("Queue1")
End Sub
```

Essa terceira função implementa o algoritmo de descarte do buffer. Essa implementação é realizada com uma única fila de células, que faz o papel da várias filas nas portas de saída do comutador. Essa fila possui dois loops para procrar células para descartar. O loop externo procura por células que sejam de classe menos prioritária, e os loops internos, dentro de cada classe, procuram por células que possuam CLP = 1, caso encontrem, retornam a que entrou no buffer por último. Se não encontrarem procuram por células com CLP = 0 e retornam a que entrou no buffer por último, se encontrar. Se não encontrar incrementa o loop externo, partindo para a classe seguinte.

```
Private Sub VBA_Block_2_Fire( )
    'Subprocedimento usado para percorrer a fila e descartar as células que excedem a
    capacidade do buffer. Parte-se do príncipio que sempre existe um lugar vago para
    receber uma nova célula que necessita entrar no buffer.
    Dim s As SIMAN
    Set s = ThisDocument.Model.SIMAN
    Dim CLP1 As Double          'CLP da célula (assume valores 0 e 1)
    Dim QoS1 As Double          'QoS da célula
    Dim RankDentro As Long      'Posição física da célula na fila
    Dim PercorreDentro As Long  'variável para percorrer a fila em busca de um CL
                                  e QoS específicos
    Dim PercorreFora As Long    'Váriável para percorrer a fila em busca de novo CLP
                                e nova QoS
    Dim NumFila As Long         'Número de elementos contidos na fila
    Dim Capacidade As Integer
    Capacidade = 600

    CLP1 = 0
    QoS1 = 0
    s.IndexVariable = 0

    'Loop usado para percorer em busca de uma nova classe de mais baixa QoS
    PercorreFora = s.QueueLastEntity(s.SymbolNumber("Queue1"))
    Do While PercorreFora <> 0

        'Loop para percorrer a fila em busca de uma célula com QoS=2 e CLP=1
        RankDentro = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
```

```
PercorreDentro = s.QueueLastEntity(s.SymbolNumber("Queue1"))
Do While PercorreDentro <> 0
        CLP1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("CLP"))
        QoS1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("QoS"))
        NumFila = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
        If (NumFila >= Capacidade) And (QoS1 = 2) And (CLP1 = 1) Then
                'seta J para receber o registro de localização, caso encontre
                variável para descartar
                s.IndexVariable = RankDentro
        End If
        If s.IndexVariable <> 0 Then
                Exit Do
        End If
        PercorreDentro = s.QueuePreviousEntity(PercorreDentro)
        RankDentro = RankDentro - 1
Loop
If s.IndexVariable <> 0 Then
        Exit Do
End If

'Loop para percorrer a fila em busca de uma célula com QoS=2 e CLP=0
RankDentro = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
PercorreDentro = s.QueueLastEntity(s.SymbolNumber("Queue1"))
 Do While PercorreDentro <> 0
        CLP1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("CLP"))
        QoS1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("QoS"))
        NumFila = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
        If (NumFila >= Capacidade) And (QoS1 = 2) And (CLP1 = 0) Then
                'seta J para receber o registro de localização, caso encontre
                variável para descartar
                s.IndexVariable = RankDentro
        End If
        If s.IndexVariable <> 0 Then
                Exit Do
        End If
        PercorreDentro = s.QueuePreviousEntity(PercorreDentro)
        RankDentro = RankDentro - 1
Loop
If s.IndexVariable <> 0 Then
        Exit Do
End If

'Loop para percorrer a fila em busca de uma célula com QoS=1 e CLP=1
RankDentro = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
PercorreDentro = s.QueueLastEntity(s.SymbolNumber("Queue1"))
Do While PercorreDentro <> 0
        CLP1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("CLP"))
```

```
        QoS1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("QoS"))
        NumFila = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
        If (NumFila >= Capacidade) And (QoS1 = 1) And (CLP1 = 1) Then
            'seta J para receber o registro de localização, caso encontre
            variável para descartar
            s.IndexVariable = RankDentro
        End If
        If s.IndexVariable <> 0 Then
            Exit Do
        End If
        PercorreDentro = s.QueuePreviousEntity(PercorreDentro)
        RankDentro = RankDentro - 1
Loop
If s.IndexVariable <> 0 Then
        Exit Do
End If

'Loop para percorrer a fila em busca de uma célula com QoS=1 e CLP=0
RankDentro = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
PercorreDentro = s.QueueLastEntity(s.SymbolNumber("Queue1"))
Do While PercorreDentro <> 0
        CLP1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("CLP"))
        QoS1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("QoS"))
        NumFila = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
        If (NumFila >= Capacidade) And (QoS1 = 1) And (CLP1 = 0) Then
            'seta J para receber o registro de localização, caso encontre
            variável para descartar
            s.IndexVariable = RankDentro
        End If
        If s.IndexVariable <> 0 Then
            Exit Do
        End If
        PercorreDentro = s.QueuePreviousEntity(PercorreDentro)
        RankDentro = RankDentro - 1
Loop
If s.IndexVariable <> 0 Then
        Exit Do
End If

'Loop para percorrer a fila em busca de uma célula com QoS=0 e CLP=1
RankDentro = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
PercorreDentro = s.QueueLastEntity(s.SymbolNumber("Queue1"))
Do While PercorreDentro <> 0
        CLP1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("CLP"))
        QoS1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("QoS"))
        NumFila = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
        If (NumFila >= Capacidade) And (QoS1 = 0) And (CLP1 = 1) Then
```

```
                    'seta J para receber o registro de localização, caso encontre
                    variável para descartar
                    s.IndexVariable = RankDentro
                End If
                If s.IndexVariable <> 0 Then
                    Exit Do
                End If
                PercorreDentro = s.QueuePreviousEntity(PercorreDentro)
                RankDentro = RankDentro - 1
            Loop
            If s.IndexVariable <> 0 Then
                Exit Do
            End If

            'Loop para percorrer a fila em busca de uma célula com QoS=0 e CLP=0
            RankDentro = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
            PercorreDentro = s.QueueLastEntity(s.SymbolNumber("Queue1"))
            Do While PercorreDentro <> 0
                CLP1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("CLP"))
                QoS1 = s.EntityAttribute(PercorreDentro, s.SymbolNumber("QoS"))
                NumFila = s.QueueNumberOfEntities(s.SymbolNumber("Queue1"))
                If (NumFila >= Capacidade) And (QoS1 = 0) And (CLP1 = 0) Then
                    'seta J para receber o registro de localização, caso encontre
                    variável para descartar
                    s.IndexVariable = RankDentro
                End If
                If s.IndexVariable <> 0 Then
                    Exit Do
                End If
                PercorreDentro = s.QueuePreviousEntity(PercorreDentro)
                RankDentro = RankDentro - 1
            Loop
            If s.IndexVariable <> 0 Then
                Exit Do
            End If
            PercorreFora = s.QueuePreviousEntity(PercorreFora)
        Loop
        ' Se não encontrou nada para retirar, então descarta essa entidade para que não
        ative o remove
        If s.IndexVariable = 0 Then
            s.EntityDispose (s.ActiveEntity)
        End If
End Sub
```