Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

Medição de Desempenho de Hardware ATM no Nível de Célula

Leonardo Dutra Castanheira

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Ney Laert Vilar Calazans

Porto Alegre, janeiro de 2003

Aos meus maiores mestres, meus pais, Antonio Carlos Castanheira e Idanira Dutra Castanheira.

Agradecimentos

Aos amigos, em especial ao prof. João Batista pelo seu empenho em procurar resolver os problemas ao longo deste trabalho, na maioria das vezes impregando soluções "mágicas"para os nossos problemas acadêmicos de cada dia.

Aos colegas pelo companheirismo. Ao longo do tempo nos tornamos uma equipe, todos em busca de um objetivo comum. Nos tornamos soldados em uma batalha quase real. Valeu pessoal, pela "mira"certeira que muitas vezes evitou o recomeço. Mesmo aqueles que costumavam ficar grande parte do tempo "escondidos", foram fundamentais para cada vitória. Para cada um de nós existiu nesta batalha um "general", assim gostaria de agradecer ao colega Alexandre de Moraes Amory que em nenhum momento exitou em compartilhar o seu conhecimentos com os seus companheiros de grupo de pesquisa.

Aos mestres, por sua dedicação em cada fase do curso. Em especial ao Prof. Dr. Ney Calazans, pela orientar e motivar a criação deste trabalho.

Aos meus avós, que por estarem longe, me mostraram que a saudade é inevitável. Porém, fico mais confortado quando penso que cada vitória minha os deixam mais orgulhosos de seus filhos.

A minha irmã, pelo seu carinho e torcida. Sendo seu único e mais velho irmão, a tua presença em cada vitória na minha vida torna esta ainda mais importante.

Aos meus pais pelo carinho e incentivo. Obrigado pai, por ensinar-me a responsabilidade. Obrigado por ensinar-me quanto é fácil ser feliz com o que temos, sem jamais deixarmos de sonhar. Mãe, obrigado pelos ensinamentos da vida, por tornar o meu passo firme em cada jornada.

Ao meu anjo, minha noiva Sheila, pelo seu inestimável e incansável amor, por sua companhia e apoio. Obrigado por aquelas valiosas dicas, pela paciência, e sobre tudo, por ser uma excelente companheira na vida e no trabalho.

Abstract

The ATM technology has been chosen as the transfer mode for the B-ISDN. However, exploiting the potential benefits of this technology is a challenging task. ATM proposes simple, efficient traffic control techniques in order to optimize network utilization without compromising the applications quality of service, but in many cases these simple techniques can be difficult to apply. As a consequence, several tools have been proposed and developed to estimate the performance of ATM networks.

Today, network simulators are the main tools to measure the performance of ATM networks implementations. Simulation methods present several advantages when compared to analytical methods and to real networks experimentation methods. In many cases however, employing simulation can be inadequate. This is particularly true with regard to simulation time of complex networks and/or simulations of long time periods.

This work has as one of its objectives the investigation of environments used to measure the performance of ATM hardware modules. Also, it proposes a measurement method that employs FPGA-based hardware prototyping environments as a way to significantly reduce the time spent in measuring performance data and testbench development. This enables functional validation of ATM hardware modules at a lower cost. In order to validate the proposed method, a parameterizable traffic source has been designed and implemented. Additionally, an ATM layer parameterizable hardware for a given application context has been implemented. This implementation has served as a testbed for the proposed method. The first experimental results demonstrate the flexibility and usefulness of the method. This is particularly true with regard to the viability of applying the method and its execution time efficiency when compared to a simulation approach.

Resumo

A tecnologia ATM (*Asynchronous Transfer Mode*) foi escolhida como o modo de transferência para a RDSI-FL. Contudo, ainda hoje a exploração de todo o seu benefício impõem diversos desafios. Procurando otimizar a utilização da rede sem comprometer a qualidade de serviço das aplicações, a tecnologia ATM propõe métodos simples de controle de tráfego, mas de difícil aplicação na maioria dos casos. Como conseqüência, várias ferramentas para estimar o desempenho de redes ATM vêm sendo desenvolvidas.

Os simuladores são hoje as principais ferramentas para medição de desempenho em redes ATM. Estes apresentam uma série de vantagens quando comparados aos métodos analíticos e experimentais com redes reais. Contudo, em alguns casos o custo de execução torna sua utilização inadequada. Isto é particularmente verdade em simulações de longos períodos de tempo.

Este trabalho tem como um de seus objetivos estudar formas de se medir o desempenho de módulos de *hardware* ATM. Propõe-se também um método de medida que utiliza ambientes de prototipação de *hardware* baseados em FPGAs como forma de diminuir significativamente o tempo gasto no processo de medição de dados de desempenho e o tempo de desenvolvimento de *testbenchs*. O método provê uma validação funcional do *hardware* ATM a um menor custo. Como o objetivo de validar a proposta, o presente trabalho apresenta o projeto e implementação de uma fonte de tráfego. Ainda, uma implementação de camada ATM parametrizável e seu emprego em um contexto de aplicação específico foram conduzidos para servir ao teste do método de medida proposto. Resultados iniciais de experimentos práticos demonstram a flexibilidade e utilidade do método proposto, em particular no que concerne sua viabilidade de aplicação e sua eficiência em termos de tempo de execução, quando comparado com o uso de simulação.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Símbolos e Abreviaturas

RDSI-FE	Rede Digital de Serviços Integrados de Faixa Estreita	1
RDSI-FL	Rede Digital de Serviços Integrados de Faixa Larga	1
QoS	Quality of Service	1
FPGA	Field Programmable Gate Array	2
HDL	Hardware Description Language	2
UNI	User-Network Interface	2
NNI	Network-Network Interface	2
B-ICI	Broadband Intercarrier Interface	2
ITU-T	International Telecomunication Union	2
B-ISDN	Broadband Integrated Service Digital Network	3
PM	Physical Medium Sublayer	4
TC	Transmission Convergence Sublayer	4
HEC	Header Error Check	4
OAM	Operation Administration and Maintenance	4
VCI	Virtual Channel Identifier	4
VPI	Virtual Path Identifier	4
GFC	Generic Flow Control	4
PT	Payload Type	4
CLP	Cell Loss Priority	4
EFCI	Explicit Forward Congestion Indication	4
AAL	ATM Adaptation Layer	5

CS	Convergence Sublayer	5
SAR	Segmentation and Reassembly	5
SVC	Switched Virtual Connections	5
PVC	Permanent Virtual Connection	5
VPL	Virtual Path Link	6
VPC	Virtual Path Connection	6
VCL	Virtual Channel Link	6
VCC	Virtual Channel Connection	6
TM	Traffic Management	8
CAC	Connection Admission Control	9
PCR	Peak Cell Rate	13
CBR	Constant Bit Rate	13
rt-VBR	real-time Variable Bit Rate	13
nrt-VBR	non-real time Variable Bit Rate	13
ABR	Available Bit Rate	13
UBR	Unspecified Bit Rate	13
DBR	Determinists Bit Rate	13
SBR	Statistical Bit Rate	13
ABT	ATM Block Transer	13
CTD	Cell Transfer Delay	14
CDV	Cell Delay Variation	14
CLR	Cell Loss Rate	14
MBS	Maximum Burst Size	14
MCR	Minimum Cell Rate	15
LAN	Local Area Network	15
MP	Measurement points	16

MPT	Measurement Point at T_b	16
MPI	International Measurement Point	16
ITP	International Transition Portion	16
IIP	International Interoperator Portion	16
maxCTD	Maximum Cell Transfer Delay	17
p2pCDV	Peak-to-Peak Cell Delay Variation	17
CER	Cell Error Ratio	17
SECBR	Severely Errored Cell Block Ratio	17
CMR	Cell Misinsertion Rate	17
FLR	Frame Loss Ratio	18
CDVT	Cell Delay Variation Tolerance	22
BT	Burst Tolerance	22
UPC	Usage Parameter Control	25
NPC	Network Parameter Control	25
GCRA	Generic Cell Rate Algorithm	27
REM	Rate Envelope Multiplexing	36
RS	Rate Sharing	36
GMDP	General Modulated Deterministic Process	45
MMDP	Markov Modulated Deterministic Process	45
MMPP	Markov Modulated Poisson Processes	46
IPP	Interrupted Poisson Process	46
IBP	Interrrupted Bernoulli Process	46
UTOPIA	Universal Test & Operations PHY Interface for ATM	50
CAM	Content Addressable Memory	50
PMPP	Pareto Modulated Poisson Process	65
LFSR		69

Capítulo 1

Introdução

A evolução das aplicações e das redes de comunicação justifica a necessidade de uma nova tecnologia capaz de integrar vários serviços, com uma alta taxa de transferência e baixo custo. Com o objetivo de integrar diversas categorias de serviço, surgiu a chamada RDSI-FE (Rede Digital de Serviços Integrados de Faixa Estreita) e posteriormente a RDSI-FL(Rede Digital de Serviços Integrados de Faixa Larga). Os benefícios da tecnologia ATM a tornaram o *modo de transferência* escolhido para a RDSI-FL. O modo de transferência é uma coleção de mecanismos que são usados para chavear e multiplexar o tráfego em uma rede.

Contudo, a necessidade de integrar diversos serviços exige uma tecnologia flexível de forma a adaptar-se às necessidades impostas pelas diferentes aplicações. A tecnologia ATM permite que se alcance a garantia de *qualidade de serviço* (QoS - *Quality of Service*) exigida pelas aplicações com uma eficiente utilização dos recursos da rede. A otimização no uso dos recursos da rede é medida através do *ganho estatístico* e é obtida com a multiplexação estatística das fontes. Sabe-se que as fontes não transmitem à taxa de pico durante todo o tempo. Desta forma, quando várias fontes estão sendo multiplexadas o ganho estatístico pode ser obtido considerando-se a probabilidade de que estas fontes não atinjam suas taxas de pico simultaneamente e o tempo em que se manterão neste estado. Desta forma, mais conexões podem ser estabelecidas utilizando-se os mesmos recursos. Para isso, é necessário que a rede conheça previamente as características de cada fonte.

Em redes ATM, o fluxo de informação de uma conexão deve ser mapeado em uma *categoria de serviço* e os parâmetros destas categorias devem ser estabelecidos de acordo com as características da mesma. Contudo, para a maioria das categorias de serviço, garantir sua qualidade não é uma tarefa simples. Várias vantagens oferecidas pelas categorias de serviço não são exploradas devido à complexidade das técnicas envolvidas. Por outro lado, o sucesso das redes ATM depende do desenvolvimento de métodos efetivos para o gerenciamento de tráfego.

A qualidade de serviço resultante da aplicação das técnicas de gerenciamento de tráfego ao longo da rede ATM, é medida por um conjunto de parâmetros, denominados de *parâmetros de qualidade de serviço* pelo ATM-Forum ou de *parâmetros de desempenho* pelo ITU-T. A qualidade fim a fim de uma conexão combina o desempenho de todos os elementos ao longo da mesma.

A avaliação de desempenho de módulos ATM é uma fase importante no projeto da rede. Nesta fase é possível medir, a partir dos parâmetros de qualidade de serviço, o potencial de degradação da QoS do módulo. Posteriormente, este potencial pode ser somado ao potencial dos demais elementos de uma conexão e então comparados com as exigências da mesma. O processo de avaliação pode ser realizado em diferentes níveis de abstração e de resolução de

tempo. Os níveis de avaliação são abordados em mais detalhes na Seção 4.4.1.

Devido à necessidade de alto desempenho, muitos módulos ATM são implementados em *hardware*, criando novas dificuldades às ferramentas de avaliação. Por exemplo, a simulação atualmente é o método mais flexível, contudo, sua utilização é inviável por exemplo no nível de detalhamento envolvido em um projeto de *hardware* no nível de tranferência de registradores. As características do método de simulação são comentadas na Seção 4.4.2. Da mesma forma, as ferramentas que propõem uma avaliação no nível de *hardware* têm como grande desafio aumentar sua flexibilidade e seu compromisso custo benefício, pois em muitos casos a validação exige equipamentos de alto custo.

O aumento da capacidade dos FPGAs (*Field Programmable Gate Array*) vem tornando as plataformas de prototipação de *hardware* uma excelente ferramenta no auxílio a validação de projetos ATM complexos. Além disso, o aumento da qualidade das ferramentas de síntese lógica e de alto nível tornaram a combinação de FPGAs e linguagens de descrição de *hardware* (HDL), como VHDL e Verilog, largamente utilizados pelos projetistas de *hardware*. Desta forma, uma ferramenta de avaliação de desempenho de módulos de *hardware* ATM que explore a escalabilidade e a disponibilidade desta tecnologia caracteriza-se como uma excelente opção devido ao seu baixo custo de obtenção, de projeto, de uso e alta flexibilidade.

O trabalho descrito neste documento propõe o desenvolvimento de uma ferramental de apoio ao processo de avaliação de desempenho de projeto de módulos de *hardware* ATM, contemplando atividades de protocolo e de gerenciamento de tráfego executadas no nível de célula. Este ferramental contém suporte de *hardware* e *software* necessários a geração de estímulos (fontes de tráfego) e coleta de informações do *hardware avaliado* (HA). A proposta visa a utilização de plataformas de prototipação baseadas em FPGAs e HDLs para a especificação dos módulos de *hardware* envolvidos. As seções seguintes deste capítulo comentam diversos aspectos sobre redes ATM e FPGA relevantes ao trabalho realizado.

1.1 Arquitetura das redes ATM

Uma rede ATM é composta de *sistemas terminais* e chaves intermediárias, todos interligados por um meio físico. *Sistema terminal* neste contexto, significa qualquer equipamento ou rede, ligada a rede ATM em questão. A padronização ATM define duas interfaces entre os equipamentos: a UNI (*User-Network Interface*) e a NNI (*Network-Network Interface*)[COM02] [COM02a]. Existem dois tipos de UNI e NNI definidas pelo ATM Forum. Para a UNI são: UNI pública e UNI privada. A UNI pública é tipicamente utilizada para interconectar equipamentos terminais à chaves públicas e chaves privadas à chaves públicas e a UNI privada para interconectar equipamentos terminais à chaves privadas. Com a NNI não é diferente, existem as NNI públicas e NNI privadas. A NNI pública é utilizada para interconectar chaves públicas e a NNI privada para interconectar chaves privadas. A interface entre duas redes públicas é conhecida como B-ICI (*Broadband Intercarrier Interface*). A Figura 1.1 ilustra uma estrutura de rede ATM, com estas características.

1.2 Padronização em redes ATM

As organizações internacionais de padronização podem ser classificadas pelo seu enfoque técnico e por sua estrutura geográfica e política. As principais organizações que tratam sobre redes ATM são a ITU-T (*International Telecomunication Union*) e o ATM-Forum. O ITU-T

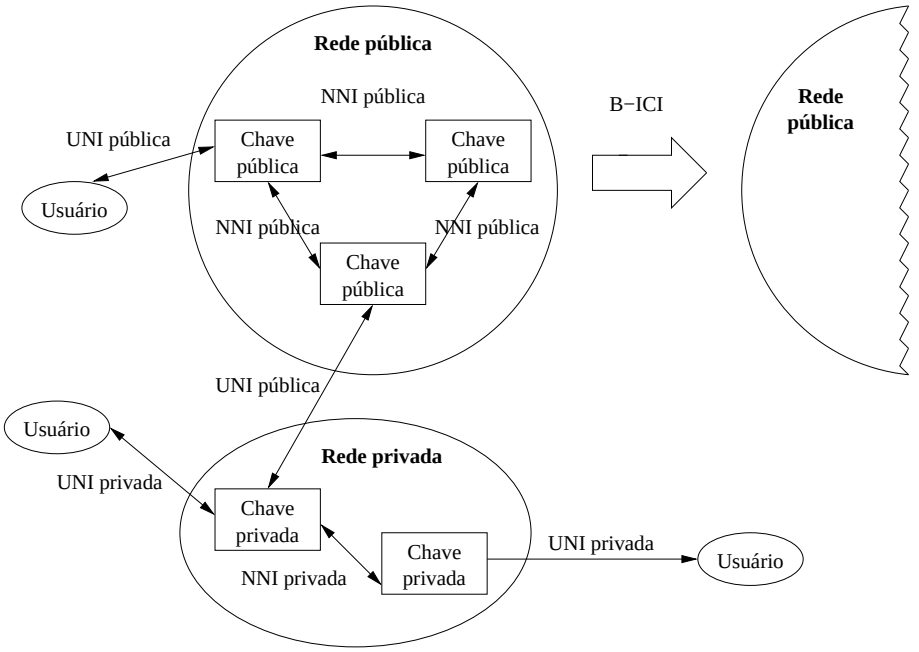


Figura 1.1: Ilustra as interfaces dos equipamentos de redes ATM. Chaves de uma rede podem ter até três tipos de interface. Em redes públicas: NNI pública (quando ligada a outra chave da mesma rede), UNI pública (quando ligada a outro equipamento fora da rede pública) e B-ICI (quando ligada a uma chave pertencente a outra rede pública). Em uma rede privada: NNI privada (quando ligada a outra chave da rede privada) e UNI privada (quando ligada a um equipamento externo a rede) e UNI pública (quando ligada a outra chave de uma rede pública).

é a organização responsável pelas recomendações relativas à RDSI-FL ou B-ISDN (*Broadband Integrated Service Digital Network*). Em 1988, o ITU-T foi a organização responsável por aprovar o ATM como o novo modo de transmissão para redes B-ISDN, através da recomendação I.121. Até hoje, foram produzidas inúmeras recomendações sobre a tecnologia ATM. A listagem das diversas recomendações produzidas pelo ITU-T pode ser obtida em [ITU01].

O ATM-Forum, por sua vez, é um consórcio de empresas de informática e de telecomunicações que tem como objetivo assegurar a interoperabilidade entre os equipamentos privados e os equipamentos das redes públicas de comunicação que estão em fase de desenvolvimento. O ATM-Forum aceita a associada de qualquer tipo de organização que esteja disposta a ajudar a acelerar a disponibilidade de soluções em ATM. Assim como o ITU-T, o ATM-Forum também produz recomendações em diversas áreas relacionadas ao ATM. A relação das recomendações produzidas pelo ATM Forum pode ser obtida em [COM01].

1.3 Protocolo ATM

A ITU-T definiu em [UNI91] o modelo de referência do protocolo da B-ISDN, como mostrado na Figura 1.2. O modelo é dividido em quatro camadas: física, ATM, AAL (*ATM Adaptation layer*) e camadas superiores. A Tabela 1.1 apresenta os serviços classificados de acordo com suas camadas.

A camada física é dividida em duas subcamadas: a subcamada dependente do meio físico

(*Physical Medium Sublayer* - PM) e a subcamada de convergência de transmissão (*Transmission Convergence Sublayer* - TC). A subcamada dependente do meio físico (PM), como o próprio nome diz, executa funções relacionadas com o meio físico, como por exemplo a temporização de bit (*bit timing*) e conversão eletro-óptica, enquanto a subcamada de convergência e transmissão é responsável pelo desacoplamento da taxa de células, controle de erros de cabeçalho, delineação de célula e adaptação de quadros de transmissão. Quando necessária, a função de desacoplamento da taxa de células insere células inúteis no meio físico para adaptar a taxa de fluxo de células ATM à taxa de transmissão do meio físico. O controle de erro de cabeçalho garante apenas a integridade do cabeçalho da célula inserindo um código CRC de 8 bits no campo de cabeçalho *Header Error Check* (HEC). Isto permite a verificação de erros múltiplos e a correção de erros simples, aqueles causados pela troca de um bit. A delineação de células identifica e delimita a célula ATM para permitir que o receptor recupere as fronteiras da mesma. Por último, a função de adaptação de quadros de transmissão encapsula as células ATM em quadros apropriados à transmissão no meio físico, se isso for necessário.

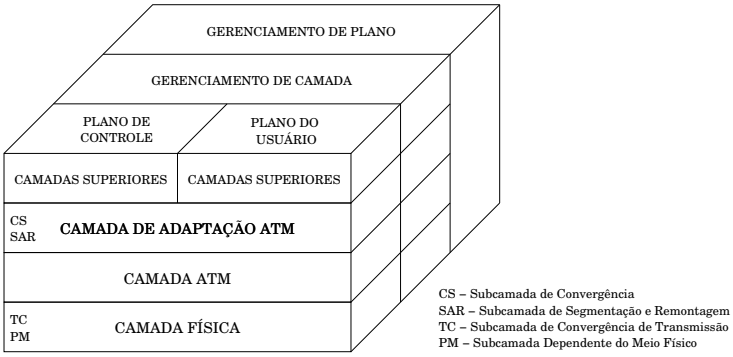


Figura 1.2: Modelo de referência do protocolo da B-ISDN.

Uma das funções da camada ATM é manter a qualidade de serviço para as conexões, garantindo que não haverá degradação da qualidade da transmissão de uma conexão devido ao mau comportamento de uma outra. A camada ATM também é responsável pela comutação de células nos chaveadores. Isso inclui não apenas a transferência da célula de um enlace de entrada para um enlace de saída adequado, mas também a atualização dos valores de VPI e VCI, se necessário. VPI e VCI são identificadores de conexão pertencentes ao cabeçalho da célula ATM. A Figura 1.3 mostra os campos de uma célula ATM

O mesmo formato de célula ATM é utilizado tanto para células de usuário como para células de gerenciamento (*Operation, Administration and Maintenance* - OAM) apenas com semântica diferente. A célula de usuário em uma interface UNI contém os campos VCI (*Virtual Channel Identifier*), VPI (*Virtual Path Identifier*), GFC (*Generic Flow Control*), PT (*Payload Type*), CLP (*Cell Loss Priority*), HEC (*Header Error Check*) e o campo de carga útil (*payload*). Como dito anteriormente, os campos VCI e VPI relacionam a célula a uma conexão. O uso desses campos é melhor explicado na Seção 1.4. O campo GFC é utilizado para controle da taxa de um terminal usando um controle de fluxo *stop-and-go*. O campo PT indica se o campo de carga útil (*payload*) contém dados do usuário ou informação de gerenciamento (células OAM). Para uma célula de usuário, o primeiro bit de PT assume o valor 0; o segundo bit é o *Explicit Forward Congestion Indication* (EFCI), que quando usado indica a existência de um congestionamento ao longo do caminho da conexão; o terceiro bit é utilizado pela camada superior. Atualmente, a AAL5 utiliza este bit para indicar a ultima célula de um quadro. Já

o campo CLP, indica a prioridade da célula. Célula com CLP=0 tem maior prioridade que uma célula com CLP=1. Isto significa que em caso de congestionamento, células com CLP=1 podem ser descartadas antes de células com CLP=0. O campo HEC, por sua vez, é utilizado pela camada inferior. A camada física preenche este campo com um código CRC de 8 bits, o que permite a verificação e correção de erros simples, ou seja, causados por erro de um bit. O último campo, carga útil ou *payload*, carrega a informação propriamente dita, seja ela de usuário ou de gerenciamento.

Os campos do cabeçalho da célula mudam nas interfaces UNI e NNI, mas o tamanho da célula permanece 53 bytes sendo 5 deles parte do cabeçalho. A diferença é que na interface NNI os bytes do campo GFC (*Generic Flow Control*), utilizado para controle de fluxo, são utilizados para aumentar o número de valores VPI. A multiplexação de células, por fim, é a inserção de várias conexões em um único caminho de transmissão.

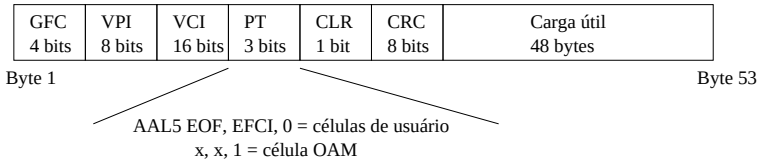


Figura 1.3: Formato da célula ATM. O mesmo formato de célula é utilizado tanto para células de usuário quanto para células de gerenciamento. Células de usuário e de gerenciamento são diferenciadas pelo valor de um bit do campo GFC.

A camada de adaptação ATM (AAL) é dividida em duas subcamadas: subcamada de convergência (*Convergence Sublayer* - CS) e a subcamada de segmentação e remontagem (*Segmentation and Reassembly* - SAR). Essas camadas realizam funções específicas, dependendo do tipo da AAL. Existem basicamente quatro tipos de AAL que visam implementar funções específicas para os diferentes tipos de tráfego. No entanto, é comum a todas as SARs a função de particionamento dos dados, recebidos de um nível superior na pilha de protocolo, em células de 53 bytes. No lado do receptor é feita a função inversa. Essa subcamada deve ser capaz de remontar os dados a partir das células enviadas. Maiores detalhes sobre as camadas físicas e AAL podem ser obtidos em [SOU01].

O modelo de referência é dividido em três planos: de usuário, de controle e de gerenciamento. O plano de usuário envolve a camada física, ATM e AAL (*ATM Adaptation layer*). Esse plano tem como objetivo prover facilidades de transmissão de dados fim a fim. O plano de controle é compartilhado pelas camadas física, ATM, AAL e protocolos de sinalização de alto nível. Este plano dá suporte à sinalização para estabelecimento, liberação de conexão e outras funções necessárias para controle de conexões quando SVCs (*Switched Virtual Connections*) são utilizadas. O plano de gerenciamento, por outro lado, tem como função facilitar a troca de informações entre o plano do usuário e de controle.

1.4 Conexões ATM

ATM é uma tecnologia orientada a conexão, apesar de permitir serviços não orientados a conexão [DUT95]. O padrão de rede ATM define dois tipos de conexão virtual: PVC (*Permanent* ou *Provided Virtual Connection*) e SVC (*Switched Virtual Connection*). PVCs são circuitos estabelecidos manualmente ou através de um protocolo de gerenciamento, enquanto

Tabela 1.1: Funções das camadas ATM.

Funções	**Camada/Subcamada**	
Funções de camadas superiores	níveis superiores	
Convergência	CS	AAL
Segmentação e remontagem	SAR	
Controle de fluxo genérico Interpretação de VPI/VCI Geração e extração de cabeçalho Multiplexação e demultiplexação de células	ATM	
Desacoplamento da taxa de células Controle de erros de cabeçalho (HEC) Delineação de células Adaptação do quadro de transmissão Geração e recuperação de quadros de transmissão	TC	Camada física
Inserção e extração de bit Temporização de bits Conversão eletro-óptica Acesso ao meio físico	PM	

em SVCs as conexões são estabelecidas sob demanda. Contudo, tanto em SVCs quanto em PVCs, antes dos dados serem efetivamente transmitidos, é necessário que uma conexão seja estabelecida. Em SVCs as conexões são criadas e liberadas utilizando-se sinais fora da banda. Isto significa que inicialmente existe uma banda reservada para a transmissão desses sinais. Contudo, é possível que seja alocada mais banda para transmissão desses sinais utilizando metassinalização. O processo de sinalização e metassinalização UNI estão descritos em [UNI96] e [UNI93] respectivamente.

Uma conexão consiste de concatenações de enlaces da camada ATM para fornecer uma capacidade de transferência fim a fim para os pontos de acesso. Durante este processo, um caminho é selecionado de acordo com os parâmetros de qualidade de serviço e identificadores (VCI e VPI) são escolhidos para esses caminhos. Além disso, recursos são reservados nos equipamentos da rede para garantir a qualidade de serviço exigida pelas aplicações. VPI significa *Virtual Path Identifier* e VCI significa *Virtual Connection Identifier*. Esses identificadores são carregados no cabeçalho de cada célula que trafega na rede. Os nós de comutação possuem tabelas que são atualizadas quando a conexão que o utiliza é criada ou desfeita. Assim, essa tabela é utilizada pelo nó com o objetivo de associar as células provenientes dos enlaces de entrada aos enlaces de saída. Esses nós podem comutar as células ou por VCI e VPI ou apenas por VPI.

Existem vários conceitos que estão diretamente relacionadas com os campos VPI e VCI de uma célula [UNI97]. São eles: VPL (*Virtual Path Link*), VPC (*Virtual Path Connection*), VCL (*Virtual Channel Link*) e VCC (*Virtual Channel Connection*). VPC é uma concatenação de dois ou mais VPLs. VPL é um enlace físico pertencente a uma VPC. Em um VPL o valor de VPI não é alterado. Percebe-se então que envolvidos no gerenciamento de uma VPC há no mínimo três equipamentos, sendo o elemento intermediário, uma chave VP. Uma chave VP pode ser implementada utilizando apenas uma tabela para cada enlace de entrada, onde o campo VPI da célula é usado como índice que identifica o enlace de saída e o próximo valor

do campo VPI da célula. A Figura 1.4 ilustra a utilização destas tabelas em uma chave VP.

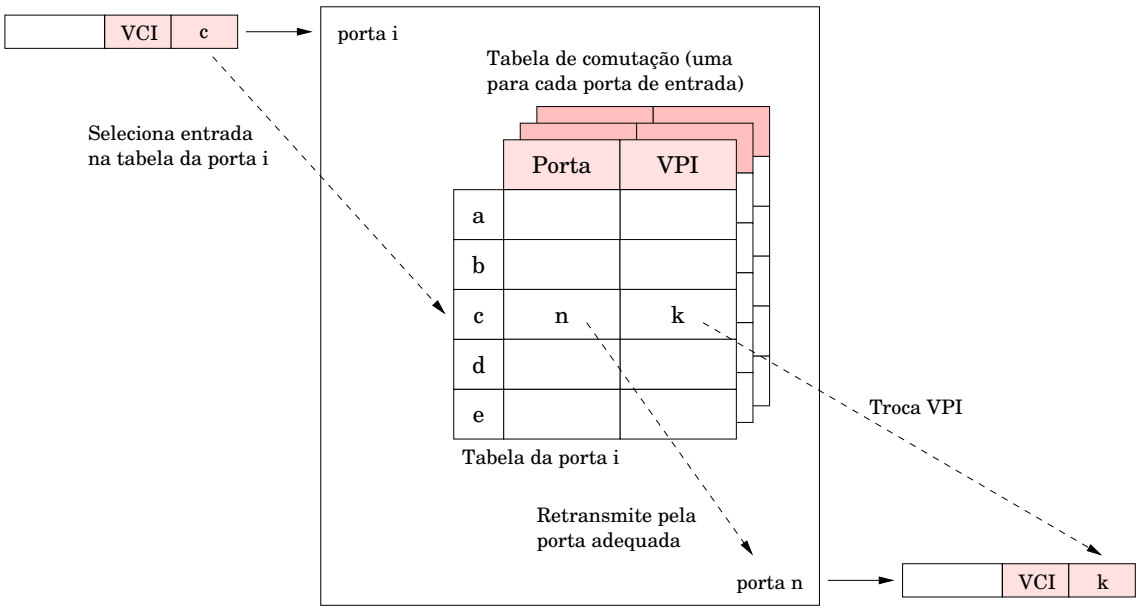


Figura 1.4: Tabela de roteamento de uma chave VP. Existe uma tabela para cada porta de entrada da chave ATM. Neste exemplo, uma célula que chega pela porta i deve ser roteada pela porta n. O campo VPI desta célula seleciona uma entrada na tabela da porta i. O primeiro campo da tabela indica o porta de saída da célula e o segundo, o próximo valor de VPI.

O objetivo de utilizar conexões VPC é diminuir o processamento nas chaves. Uma conexão VPC contém várias conexões VCC, como pode ser visto na Figura 1.5. Com isso, é possível diminuir a carga nos chaveadores, não apenas na execução do processo de chaveamento mas também na execução dos algoritmos que garantem a qualidade de serviço para as conexões de uma VPC. Em uma VPC, o valor de VCI não é modificado já que a célula é comutada utilizando somente o campo VPI.

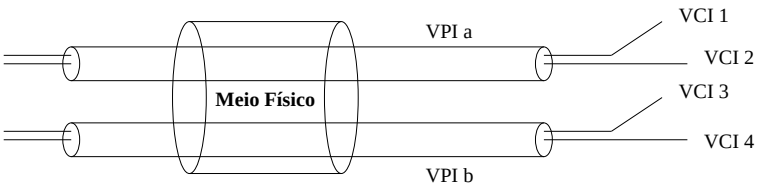


Figura 1.5: Conexões VCC agregadas em uma VPC. Várias conexões VCI podem ser agregadas em uma VPI, com o objetivo de diminuir o custo do chaveamento e da execução dos algoritmos de gerenciamento de tráfego.

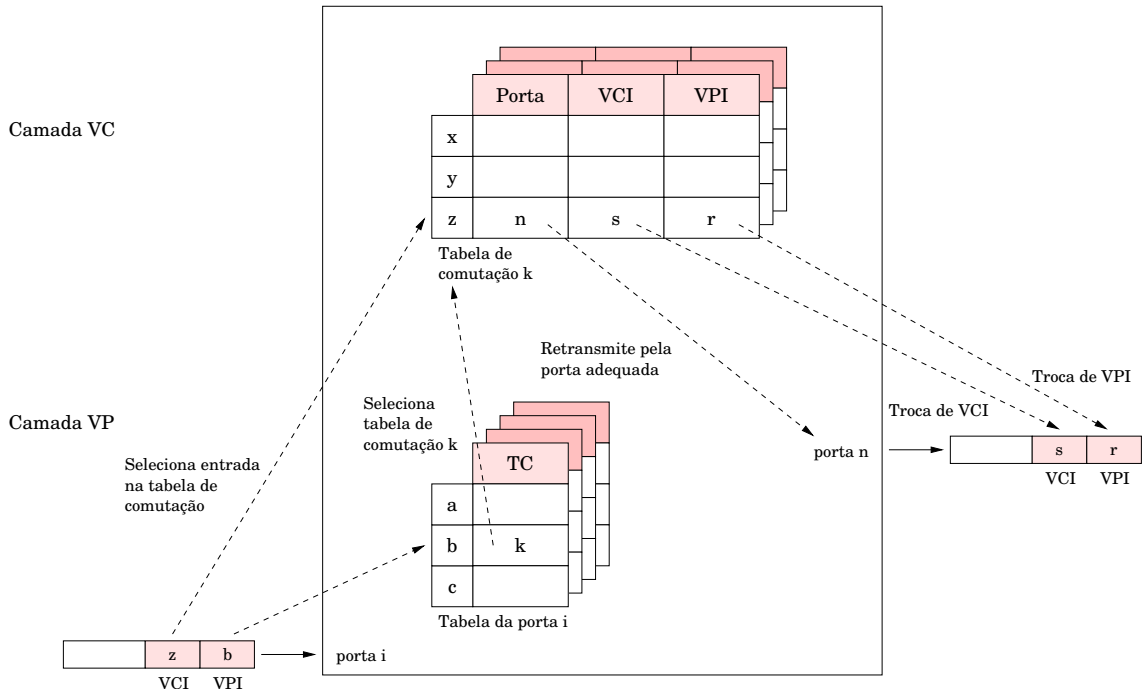


Figura 1.6: Tabela de roteamento de uma chave VP+VC. Em uma chave VP+VC existe uma tabela (TC) para cada porta de entrada e uma tabela para cada entrada desta tabela. Neste exemplo, uma célula que chega pela porta i deve ser roteada pela porta n. O campo VPI da célula seleciona a entrada na tabela TC. O campo da tabela TC seleciona a segunda tabela (tabela de comutação k). O campo VPC da célula seleciona a entrada nesta segunda tabela. O primeiro campo desta tabela indica a porta de saída, o segundo o próximo valor de VCI e o terceiro o próximo valor de VPI.

VCL é um enlace pertencente a uma VCC. Uma VCC, no entanto, é uma concatenação de VCLs e caracteriza-se por ser uma conexão fim a fim. Uma conexão VCC envolve uma chave VC+VP. Um chaveador VP+VC pode ser implementado utilizando-se duas tabelas para cada enlace de entrada. A primeira tabela usa o valor do campo VPI da célula como índice. Esta tabela identifica uma outra que deve utilizar o valor do campo VCI, também como índice. Como resultado do processamento da última tem-se o enlace de saída e os próximos valores dos campos VCI e VPI da célula. A Figura 1.6 mostra esta possível implementação. Então, um valor de VCI e VPI, na verdade, não caracteriza uma conexão fim a fim, e sim uma conexão em um determinado ponto da rede. Isso torna eficiente o reaproveitamento dos valores de VCI e VPI. A Figura 1.7 mostra os dois tipos de conexões (VPC e VCC) e suas relações com os valores dos campos VCI e VPI de uma célula.

1.5 Qualidade de Serviço

O compromisso de garantir a qualidade de serviço (*Quality of Service* - QoS) necessária para as aplicações impõe diversos desafios ao gerenciamento de tráfego (*Traffic Management* - TM) em redes ATM. O gerenciamento de tráfego pode ser dividido, segundo [GIR98], em um conjunto de tarefas específicas: contrato de tráfego, controle de admissão de conexão, policiamento, filas e escalonamento, controle de fluxo e controle de congestionamento. A

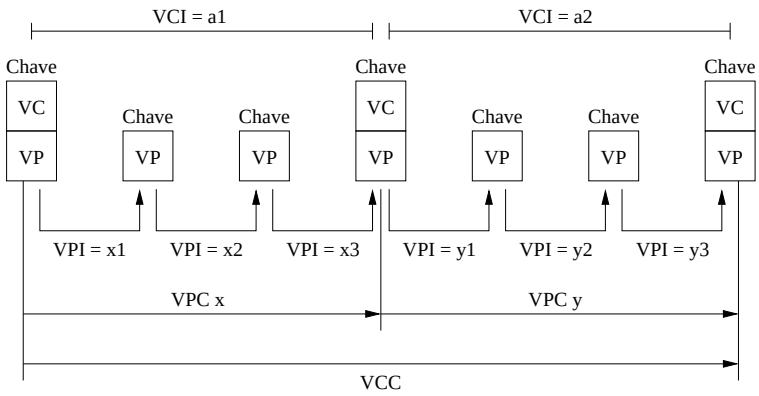


Figura 1.7: Relação entre os tipos de conexão, os identificadores de conexão e as chaves ATM. Os valores de VPI e VCI são alterados ao longo de uma VCC. Já em uma VPC, o valor de VCI não é alterado. Ou seja, o conjunto de identificadores VPI e VCI não identifica uma conexão ATM fim a fim.

Figura 1.8 mostra uma possível ordem de seqüência na aplicação das funções de gerenciamento de tráfego.

Uma aplicação que usa a rede ATM deve negociar o contrato de tráfego com a rede antes que a conexão virtual seja estabelecida. Essa aplicação deve mapear sua conexão virtual na categoria de serviços apropriada na camada ATM. Uma vez que o contrato de tráfego esteja definido para uma conexão, a rede necessita de um algoritmo de controle de admissão de conexão (*Connection Admission Control* - CAC) para determinar se a conexão pode ser aceita, com a garantia da QoS esperada pelo contrato, sem prejudicar as conexões previamente estabelecidas. Se a conexão for aceita, então as células podem iniciar seu fluxo na rede. Para evitar que uma conexão prejudique a QoS das demais, um ou mais mecanismos de policiamento pode ser usado. A função destes mecanismos é monitorar o tráfego da conexão para verificar se seu comportamento está de acordo com o contrato de tráfego estabelecido. Na maioria dos casos, juntamente com os mecanismos de policiamento, utiliza-se um mecanismo de formatação de tráfego (*traffic shaping*). Esse mecanismo é utilizado para diminuir o efeito da penalidade de causada pelo mecanismo de policiamento, resultando em um melhor aproveitamento da rede.

Eventualmente, conexões específicas precisam ser multiplexadas em determinados pontos da rede. Para se obter um ganho na execução desta tarefa, pode-se utilizar filas ou outro tipo de armazenamento temporário antes que o tráfego seja transmitido para os enlaces seguintes. Algoritmos de escalonamento são utilizado para procurar manter a qualidade de serviço almejada pelos diferentes tipos de conexões.

Apesar de todos os mecanismos descritos acima para garantir a qualidade em uma conexão, ainda existe o problema do congestionamento, causado pela sobreposição dos tráfegos em um determinado ponto da rede, que impõe um risco no compromisso com a QoS. Mecanismos de controle de congestionamento podem ser usados para garantir a QoS através do descarte seletivo de células. Também existem os mecanismos de controle de fluxo, que ao contrário do controle de congestionamento, procuram evitar que este último ocorra. Esses mecanismos utilizam o repasse de informação da rede para os sistemas terminais para que estes adaptem-se dinamicamente ao estado da rede. Esse controle de fluxo pode ser simples, utilizando apenas um bit no cabeçalho da célula (campos GFC ou EFCI), ou complexos, como aquele utilizado pela classe de serviço ABR, também chamado de *Controle de Fluxo Reativo*.

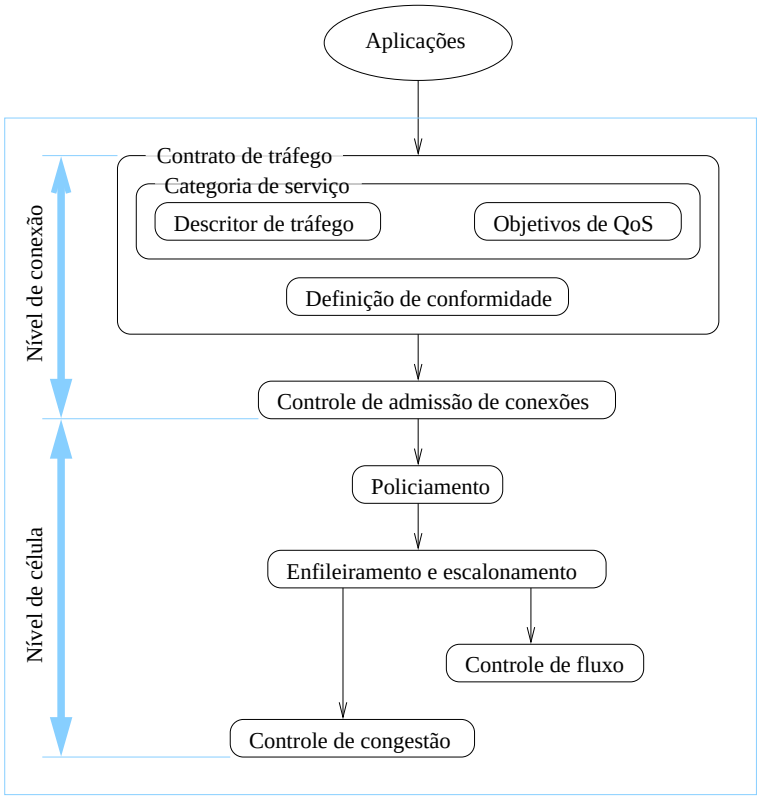


Figura 1.8: Funções de gerenciamento de tráfego. Fonte: [GIR98].

1.6 Motivação

O estudo sobre ATM vinha sendo realizado junto com o desenvolvimento de uma camada ATM como parte de um *driver*. Este tem como objetivos receber informações de uma fonte de voz e dados (rede Ethernet), formatá-los e introduzí-los em uma rede ATM. Durante a fase de projeto e desenvolvimento verficou-se verificou-se a necessidade de ferramentas para auxiliar o processo de validação funcional e de desempenho dos módulos que pertencentes a camada ATM e de adaptação ATM (AALs).

Como é mostrado na Seção 5 subsistemas, ou módulos, de redes são facilmente reaproveitados. Assim, durante a fase de projeto e desenvolvimento é necessário uma atenção especial a modularidade e capacidade de parametrização que serão dadas aos subsistemas. Por outro lado, quanto maior a parametrização e capacidade de parametrização destes subsistemas maior é custo teste. O tempo gasto em *testbenches* chega a ultrapassar 50% do tempo gasto em desenvolvimento. Um conjunto de ferramentas para auxiliar na validação funcional destes subsistemas pode dimininuir sensivelmente o tempo de desenvolvimento de *testbenches*.

Além de facilitar o teste dos módulos de *hardware*, um ferramental que forneça estímulos e colete as informações de forma adequada também pode diminuir o tempo necessário à medição de desempenho quando comparado à simulação. Como mostrado na Seção 4.2, o processo de avaliação de desempenho em redes ATM enfrenta uma série de dificuldades imposta pelas altas taxas de tranferência de células.

Devido a alta disponibilidade de plataformas de prototipação de *hardware*, um estudo que demostre a viabilidade de sua utilização na aceleração de um processo em particular pode apontar uma solução para processos de alto custo de execução semelhantes. Não ao acaso, o

processo de medição de desempenho em redes ATM foi escolhido como caso particular.

1.7 Objetivos

Com o aumento da capacidade de dispositivos de *hardware* reconfigurável tais como os FPGAs, o processo de prototipação utilizando estes é cada vez mais comum em ambientes de desenvolvimento de *hardware*. A flexibilidade das *linguagens de descrição de hardware* (HDL), combinada com o aumento da capacidade dos FPGAs e a evolução das plataformas de prototipação diminuem sensivelmente o custo e o tempo de implementação de um sistema. Utilizar esta forma de prototipação para construir e avaliar o desempenho de protótipos de módulos ATM pode causar uma sensível melhora na qualidade e uma redução no tempo de projeto.

A presente proposta de Plano de Estudos e Pesquisa consiste na especificação de um ambiente para medição de desempenho de módulos de *hardware* ATM, utilizando plataformas de prototipação de *hardware*, e opcionamente, *software*. O ambiente constituir-se-á de núcleos de propriedade intelectual do tipo soft (ou *IP soft cores*) projetados para a extração de informações de módulos ATM a nível de células. Ferramentas como *ChipScope* da *Xilinx* [XIL01] e *SignalTap* da *Altera* [ALT01] realizam funções semelhantes a desta proposta, qual seja, servir como apoio internamente ao circuito integrado para a extração de informações de módulos de hardware. Estas ferramentas comerciais permitem a extração de informações do ambiente de prototipação a nível de sinais lógicos (0s e 1s). contudo, a utilização destas ferramentas para extração de informações no auxílio a avaliação de desempenho de módulos ATM, não é uma boa opção devido ao nível de detalhamento oferecido pelas mesmas. Já o ambiente proposto, procurará oferecer um nível de abstração maior, onde as informações são tratadas a nível de células, tornando mais simples a utilização das informações no processo de avaliação do desempenho e validação dos módulos em questão.

1.8 Estrutura do Trabalho

O restante do trabalho está estruturado da seguinte forma. O capítulo 2 apresenta uma as principais características das categorias de serviços ATM definidas pelo ATM Forum. O capítulo seguinte discreve sobre os dois principais algoritmos utilizados na conformação de tráfego de acordo com os parâmetros de tráfego da conexão. O capítulo 4 traz um resumo sobre avaliação de desempenho em redes ATM. O mesmo também contém uma revisão bibliográfica onde são apresentadas algumas referências a alguns trabalhos relacionados e uma revisão sobre modelos de tráfego, em especial, os modelos suportados pela fonte proposta.

Os demais capítulos relatam os trabalhos realizados durante o curso. O capítulo 5 descreve a implementação de uma camada ATM de recepção que vinha sendo desenvolvido em cooperação com uma empresa. Este trabalho é útil para que se entenda o nível de detalhe de uma implementação em *hardware* e como a reutilização de módulos parametrizáveis pode diminuir sensivelmente o tempo de desenvolvimento. O capítulo 6 apresenta a proposta de uma ferramenta, ou ambiente, para avaliação de desempenho de módulos de *hardware* utilizando linguagens de descrição de *hardware* e ambientes de prototipação. O mesmo ainda descreve detalhadamente o projeto e implementação de uma fonte de tráfego como parte da ferramenta proposta. O capítulo seguinte demonstra a utilização da fonte na geração de diferentes tipos de tráfego. Por fim, o último capítulo conclui o trabalho realizado.

Capítulo 2

Arquitetura de serviços ATM

A introdução de novas categorias de serviços ATM aumenta os benefícios da rede, tornando a tecnologia conveniente para um conjunto maior de aplicações. Uma rede ATM pode prover VPCs ou VCCs com diferentes níveis de serviço. O conceito de negociação do comportamento esperado de uma camada ATM em termos de tráfego e eficiência para cada conexão permite ao usuário otimizar as capacidades da rede para torná-la conveniente aos requisitos da aplicação.

Um comportamento comum à maioria das primeiras gerações de redes ATM era reservar uma quantidade fixa de largura de banda para cada conexão durante a chamada, baseando-se na máxima taxa de emissão da fonte (*Peak Cell Rate* - PCR) e provendo uma única forma de garantia de qualidade de serviço. O parâmetro PCR é descrito na Seção 2.4.1. As categorias de serviços ATM tornam possível ao usuário selecionar combinações específicas de tráfegos e parâmetros de eficiência.

ATM é uma tecnologia destinada vários tipos serviço. Atualmente, a maioria dos requisitos que são especificados para uma dada aplicação pode ser encontrada em uma camada de adaptação ATM (AAL) apropriada. Contudo, de acordo com a definição do padrão ATM, o comportamento da camada ATM não deve confiar nos protocolos das AALs, uma vez que esses são serviços específicos, e nem nos protocolos das camadas superiores que são aplicações específicas.

Este capítulo aborda as categorias de serviço e suas relações com aplicações. Também relaciona essas categorias aos parâmetros utilizados para descrever o comportamento de seu tráfego (descritores de tráfego) e suas exigências de QoS (parâmetros de QoS). A Seção 2.1 descreve cada categoria de serviço e aplicações relacionadas. O conceito de *pontos de medição* são introduzidos na Seção 2.2. Pontos de medição nada mais são do que uma especificação da localização em que podem ser medidos os parâmetros de qualidade de serviço, que são descritos na Seção 2.3. Por último, são apresentados os diferentes parâmetros de tráfego e suas relações com as categorias de serviço na Seção 2.4.

2.1 Categorias de serviço e aplicações

Diferentes tipos de categorias de serviço foram definidas pelo ATM-Forum e pela ITU-T. O ATM-Forum definiu as seguintes categorias: *Constant Bit Rate* (CBR), *real-time Variable Bit Rate* (rt-VBR), *non-real time Variable Bit Rate* (nrt-VBR), *Available Bit Rate* (ABR) e *Unspecified Bit Rate* (UBR). Por outro lado, a definição de categorias de serviço da ITU-T inclui: *Deterministic Bit Rate* (DBR), *Statistical Bit Rate* (SBR), *Available Bit Rate* (ABR) e *ATM Block Transfer* (ABT). Apesar da denominação distinta, há similaridade entre as ca-

tegorias de serviço definidas pelo ATM-Forum e pela ITU-T, que pode ser vista na Tabela 2.1. Nesta seção serão abordadas as categorias de serviço definidas pelo ATM-Forum, apenas por serem mais utilizadas que as da ITU-T.

2.1.1 *Constant Bit Rate* (CBR)

A categoria de serviço CBR é usada para conexões que exigem uma quantidade estática de largura de banda, que permanece continuamente disponível durante o tempo de vida da conexão. A quantidade de largura de banda alocada baseia-se no parâmetro *Peak Cell Rate* (PCR) que será discutido na Seção 2.4.1. A fonte pode emitir células na taxa de pico ou abaixo de PCR em qualquer momento e durante o tempo que desejar (ou não emitir nada). Essa categoria é utilizada por aplicações de tempo real. Essas aplicações geralmente têm requisitos de atraso rigorosos (*Cell Transfer Delay* - CTD e *Cell Delay Variation* - CDV, ver Seção 2.3.2). No entanto, essa categoria pode ser perfeitamente utilizada por outras aplicações. O comprometimento básico feito com a rede é que uma vez a conexão estabelecida a QoS negociada é assegurada para todas as células que estão respeitando sua definição de conformidade. Conformidade para PCR é discutida na Seção 3.5.3.

Aplicações que utilizam esse serviço têm requisitos estritos com relação ao retardo, no entanto permitem uma pequena taxa de perda de células (*Cell Loss Rate* - CLR, Seção 2.3.1). Células com atrasos além do valor especificado por *Cell Transfer Delay* (CTD) não são desejáveis para essas aplicações. A variação do atraso de células (CTD) é definida na Seção 2.3.2. Típicas aplicações para CBR são aquelas que transferem imagens, dados ou textos que contenham informação suficientemente importante ou aquelas em que o tempo de resposta dos sistemas finais justifica a ocupação de um canal CBR. Alguns exemplos são: vídeo conferência, áudio interativo e distribuição de áudio ou vídeo.

Para telefonia ou serviços de voz sobre ATM, a solução baseada em AAL1 requer suporte ao serviço CBR para garantir limites de atraso e variações na taxa de emissão. Também na área de multimídia, uma solução para serviços residenciais prevê transferência do formato MPEG2 sobre AAL5 provido pela camada ATM com serviço CBR.

2.1.2 *Real-Time Variable Bit Rate* (rt-VBR)

A categoria de serviço *real-time* VBR é recomendada para o uso de aplicações de tempo real (aquelas com requisitos estritos de retardo), sendo apropriadas para aplicações de voz e vídeo. Neste caso, espera-se que as fontes transmitam taxas que variam com o tempo. Alternativamente, a fonte pode ser descrita como seqüências de rajadas ("*bursts*"). Células com atrasos além de CTD não são desejáveis para aplicações que utilizam esse serviço.

A categoria de serviço rt-VBR pode suportar multiplexação estatística de fontes de tempo real. Esta é a grande diferença entre o CBR e rt-VBR. *Multiplexação estatística* dá origem ao chamado *ganho estatístico*. Uma vez que os dados não são enviados continuamente, é possível alocar menos recursos utilizando filas para absorver a chegada de células. Ganho estatístico e multiplexação estatística são discutidos na Seção 3.6.1. Aplicações que utilizam a categoria de serviço rt-VBR não têm a mesma garantia de QoS dada em serviços CBR. No entanto, elas permitem uma melhor utilização do canal, alocando uma largura de banda baseada não apenas em PCR, como é feita para serviços CBR, mas também em *Sustainable Cell Rate* (SCR), discutido na Seção 2.4.2 e *Maximum Burst Size* (MBS), discutido Seção 2.4.3.

2.1.3 *Non-Real-Time Variable Bit Rate* (nrt-VBR)

A categoria de serviço *non-real time* VBR é recomendada para o uso de aplicações *non-real time* que tem a característica do tráfego em rajadas. A categoria nrt-VBR pode ser caracterizada em termos de PCR, SCR e MBS como em rt-VBR. Geralmente, as aplicações que utilizam esse serviço toleram uma pequena taxa de perda de células (CLR). Assim como rt-VBR esse serviço dá suporte a multiplexação estatística.

VBR é apropriado para qualquer aplicação em que o sistema final possa beneficiar-se da multiplexação estatística através do envio de informações a uma taxa variável e podendo tolerar ou potencialmente recuperar pequenas perdas aleatórias. Este é o caso de qualquer fonte CBR que tem sua taxa de transmissão variável. Nesse caso, o serviço VBR permite um melhor uso dos recursos da rede sem prejudicar sua eficiência. *Real-time* VBR, em particular, pode ser usado em transmissão de voz com supressão de silêncio, sendo apropriado a qualquer classe de comunicações multimídia. *Non-real time* VBR, por sua vez, pode ser usado na transferência de dados por aplicações que executam transações com tempo de resposta crítico (reservas aéreas, transações de banco, monitoração de processos) e por *frame relay interworking*, sendo *interworking* é a utilização da tecnologia ATM por outras redes.

2.1.4 *Available Bit Rate* (ABR)

Available Bit Rate (ABR) é uma categoria de serviço da camada ATM que permite uma variação dinâmica dos parâmetros que descrevem seu tráfego. Para isso, é utilizado um tipo de mecanismo de controle de fluxo que suporta vários tipos de realimentação (*feedback*) para controlar a taxa da fonte. Com o uso desse mecanismo, a fonte deve comportar-se segundo o estado de carga dos nós da rede. Muitas fontes (aplicações) têm a habilidade de reduzir ou incrementar sua taxa transmissão se a rede necessitar. Com o uso de ABR, espera-se que o sistema final possa adaptar seu tráfego de acordo com a realimentação, causando uma baixa taxa de perda de células (CLR), e que exista um compartilhamento mais justo da largura de banda disponível, de acordo com política de alocação específica da rede.

A variação no atraso de células (CDV) não é controlada nesse serviço. O serviço ABR não é recomendado para dar suporte a aplicações de tempo real. No estabelecimento de uma conexão ABR, o sistema final deve especificar à rede a largura de banda requerida e a mínima largura de banda utilizável. Estas especificações são representadas por PCR e *Minimum Cell Rate* (MCR), respectivamente. O MCR pode ser especificado como zero. A largura de banda disponível da rede pode variar, mas não deve se tornar menor que MCR.

Qualquer aplicação *non-real time* rodando sobre um sistema final capaz de variar sua taxa de emissão pode explorar o serviço ABR. Essa categoria provê um suporte econômico às aplicações que não exigem um *throughput* rígido, requisitos estritos de atraso e uma baixa taxa de perdas de células (CLR). Exemplos incluem serviços de *interworking* utilizando-se ATM. Esses estão tipicamente sobre pilhas de protocolos baseados em roteadores como TCP/IP, que podem facilmente variar sua taxa de emissão quando requisitado pelo mecanismo policiamento de controle da taxa ABR. Outro ambiente de aplicação recomendado para o uso da categoria de serviço ABR é a emulação de redes locais (*Local Area Network* - LAN). Outros exemplos são aplicações que executam transferências de dados críticas (serviço de banco), aplicações em supercomputadores e comunicações de dados, tal como chamadas de procedimentos remotos e serviços de arquivos distribuídos.

2.1.5 *Unspecified Bit Rate* (UBR)

A categoria de serviço *Unspecified Bit Rate* (UBR) é recomendada para aplicações *non-real time*, que não exigem um compromisso rigoroso de atraso e sua variação. Exemplos de tais aplicações são aquelas que realizam comunicação tradicional de computadores, como transferência de arquivos e *e-mails*.

Espera-se das fontes UBR que elas transmitam rajadas não contínuas. Para garantir isso, pode-se ser utilizar o mecanismo de *traffic shaping*. O serviço UBR dá suporte a um alto grau de multiplexação estatística entre fontes. Esse serviço não tem nenhuma garantia de qualidade de serviço. Especificamente, UBR não utiliza a noção de uma largura de banda negociada por conexão. O fluxo de uma conexão de tráfego UBR pode não ter nenhum comprometimento com uma taxa de perda de células ou qualquer relação com o atraso das mesmas.

De uma forma geral, a utilização de UBR não é conveniente à maioria das aplicações. Por outro lado, as aplicações de dados em uma estação de trabalho são muito tolerantes a atrasos e perda de células. Alguns exemplos dessas aplicações são: transferência e distribuição de textos, dados e imagens e terminal remoto. Esses serviços pode tirar vantagem de qualquer sobra de largura de banda a um baixo custo.

2.2 Pontos de Medição

O desempenho na transferência de células ATM é medido através dos pontos de medição. *Measurement points* (MP) são pontos ao longo de uma conexão e estão associados a eventos que permitem uma descrição do desempenho da rede [UNI96a]. De acordo com [UNI00], para a RDSI-FL os MPs estão localizados nas interfaces onde a camada ATM é acessível. De forma mais precisa, os MPs estão localizados na pilha de protocolo acima das funções de multiplexação e demultiplexação (VP ou VC), mas abaixo de funções (VP ou VC) como o policiamento da taxa de células.

A referência [UNI00] define dois tipos de MPs: *Measurement Point at* T_b (MPT) e *International Measurement Point* (MPI). MPTs são MPs próximos a equipamentos terminais da rede, enquanto MPIs são MPs entre chaves nas extremidades dos países. Com o objetivo de gerenciar o desempenho de conexões ATM, estas foram divididas em três porções de acordo com [UNI00]. *Porção nacional* é uma porção localizada dentro de um mesmo país entre um MPT e um MPI. As demais porções localizam-se entre dois MPIs, sendo que a *porção de trânsito internacional* ou *International Transition Portion* (ITP) está localizada dentro de um mesmo país enquanto que a *porção de interoperador internacional* ou *International Interoperator Portion* (IIP) abrange países diferentes. Essas porções podem estar presentes tanto em VCCs como em VPCs. Em uma ITP de uma conexão VP há chaves VPs, enquanto em uma conexão VC há chaves VCs. Para uma IIP de uma conexão VP não há a presença de chaves ATM, enquanto em uma conexão VC pode haver a presença de chaves VPs, mas jamais chaves VCs. Naturalmente não há presença de chaves VCs dentro de conexões VPs, então essas chaves também não estão presentes em porções de uma VP. As Figuras 2.1 e 2.2 ilustram os diferentes tipos de porções para uma conexão VC e VP, respectivamente.

2.3 Parâmetros de qualidade de serviço

A qualidade de serviço da camada ATM é definida por um conjunto de parâmetros que caracterizam os requisitos de uma conexão. Esses parâmetros são denominados *parâmetros de*

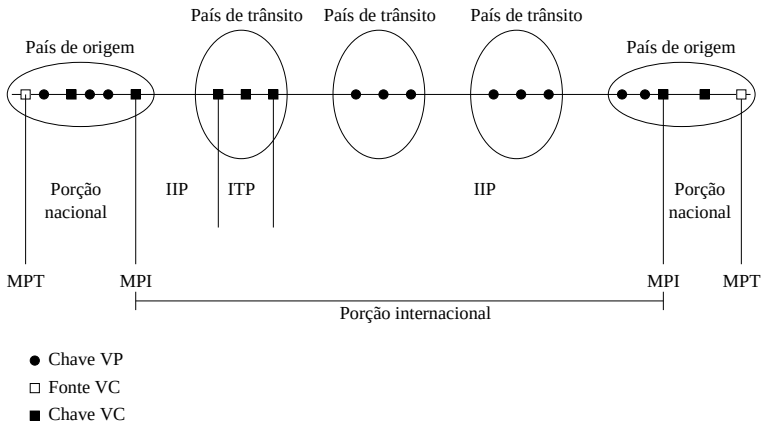


Figura 2.1: Pontos de medição em uma VCC. Ao longo de uma VCC existem chaves VP e VP+VC. A medida de desempenho em uma VCC pode ser feita nas diferentes porções que encontram-se entre duas chaves VCCs. Fonte: [UNI00].

qualidade de serviço pelo ATM-Forum. Já o ITU-T denomina-os *parâmetros de desempenho*. A qualidade fim a fim de uma conexão combina a qualidade de todos os elementos da rede. De acordo com [COM99], seis parâmetros são usados para medir a eficiência da rede para uma dada conexão. Três destes parâmetros podem ser negociados entre a rede e o sistema final. São eles: *Cell Loss Ratio* (CLR), *Maximum Cell Transfer Delay* (maxCTD) e *Peak-to-Peak Cell Delay Variation* (p2pCDV). Os parâmetros *Cell Error Ratio* (CER), *Severely Errored Cell Block Ratio* (SECBR) e *Cell Misinsertion Rate* (CMR) não podem ser negociados como parte do contrato de tráfego. Os parâmetros maxCTD e p2pCDV são discutidos na Seção 2.3.2.

2.3.1 Taxa de perda de células (*Cell Loss Ratio* - CLR)

O *Cell Loss Ratio* (CLR) é uma proporção de perda de células. Perdas de células ocorrem devido à chegada simultânea de rajadas de diferentes conexões. Para uma conexão, CLR é definido como sendo

$$CLR = \frac{\text{células perdidas}}{\text{total de células transferidas}},$$

onde segundo [GIR98] células perdidas incluem: o número de células que não chegaram ao destino, o número de células que foram recebidas com cabeçalho inválido e o número de células no qual o conteúdo foi corrompido por erros. Enquanto o total de células transferidas é o numero de células *conformes* (*conforming*) transmitidas durante um período de tempo. Uma célula *conforme* é aquela que está de acordo com os descritores de tráfego (Seção 2.4). O período de tempo citado acima não é padronizado mas pode ser entendido como o tempo de duração de uma conexão. O parâmetro CLR pode ser aplicado tanto ao fluxo CLP=0 quanto ao fluxo agregado CLP=0+1. Para o fluxo CLP=0 a rede faz distinção entre as células que tenham o valor do campo de cabeçalho igual a 0 e as que tenham o valor igual a 1. O campo CLP identifica a prioridade de uma célula sobre as outras. As células de CLP=0 tem maior prioridade sobre as células de CLP=1. Aquelas que trafegam com a indicação de menor prioridade (CLP=1) são células que em algum ponto da rede foram identificadas como não conformes. Então, para o cálculo de CLR, células de CLP=1 são consideradas como perdidas.

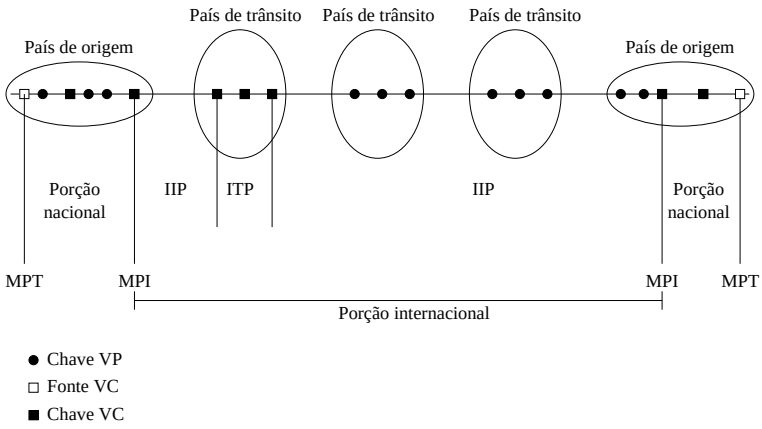


Figura 2.2: Pontos de medição em uma VPC. Ao longo de uma VPC existem apenas chaves VP. A medida de desempenho em uma VPC pode ser feita nas diferentes porções que encontram-se entre duas chaves VPCs. Fonte: [UNI00].

Quando considera-se o fluxo agregado (CLP=0+1), não há distinção entre as células, ou seja, o campo CLP é desconsiderado [UNI00].

Quando o descarte de quadros está sendo utilizado pela rede, o parâmetro CLR não é utilizado para medir o desempenho da rede. *Quadro* significa uma unidade de dados do protocolo AAL. Este tipo de descarte é utilizado com o objetivo de aumentar o desempenho da rede. Se uma célula que pertence a um quadro do nível superior for descartada, quando esse for remontado no destino, irá perceber-se a falta de informação, ou seja, a célula que foi descartada. Desta forma, todo o quadro deverá ser retransmitido. Quando o descarte de quadros é utilizado, ao invés de descartar-se apenas uma célula, descarta-se todo o quadro, evitando que as demais células, as mesmas que deverão ser retransmitidas como parte do pacote, consumam os recursos da rede. No caso desse descarte estar sendo utilizado, *Frame Loss Ratio* (FLR) é definido como sendo

$$FLR = \frac{\text{quadros perdidos} + \text{quadros corrompidos}}{\text{total de quadros transmitidos}}$$

O FLR é um parâmetro de QoS que não pode ser negociado com a rede.

2.3.2 Parâmetros de atraso

Segundo [GIR98], a medida de atraso de transferência de células (*Cell Transfer Delay* - CTD) é definida como o tempo entre o instante da partida de uma célula no sistema final até o momento de chegada no destino. O CTD entre dois pontos da rede é a acumulação de atrasos na transferência em cada nodo do caminho. O valor mínimo de CTD é obtido desconsiderando-se sua variação, ou seja, ele é medido utilizando-se apenas uma célula sem a presença de *buffers*. Para um dado nodo da rede, o valor mínimo, ou valor fixo, de CTD inclui o atraso de propagação no meio físico, o atraso do sistema de transmissão e o atraso de processamento. Quando esses nodos utilizam *buffers*, ocorrem variações no valor de CTD para as diferentes células. Esta variação é denominada de variação no atraso de célula (*Cell*

Delay Variation - CDV).

Como dito anteriormente, dois parâmetros de QoS para atrasos fim a fim podem ser negociados com a rede: *Peak-to-Peak Cell Delay Variation* (p2pCDV) e *Maximum Cell Transfer Delay* (maxCTD). A Figura 2.3 ilustra a função de densidade de probabilidade em serviços CBR e rt-VBR. O valor do *maximum Cell Transfer Delay* (maxCTD) representa a probabilidade 1-α, onde α é a probabilidade de que células excedam o valor máximo de atraso permitido, sendo essas células assumidas como perdidas ou inúteis. O parâmetro p2pCDV representa a diferença entre o máximo valor de CTD (maxCTD) e o mínimo CTD (parcela fixa do atraso). Essa medida permite a avaliação do máximo atraso possível entre duas células consecutivas que estão deterministicamente espaçadas. O valor de CDV é negociado com a rede através do descritor de tráfego CDVT, que será discutido na Seção 2.4. CDVT é um importante parâmetro utilizado na conformação e policiamento de tráfego, que serão discutidos no Capítulo 3.1. Dois métodos de medição de CDV são definidos em [UNI00]. São eles: *one-point CDV* e *two-point CDV*.

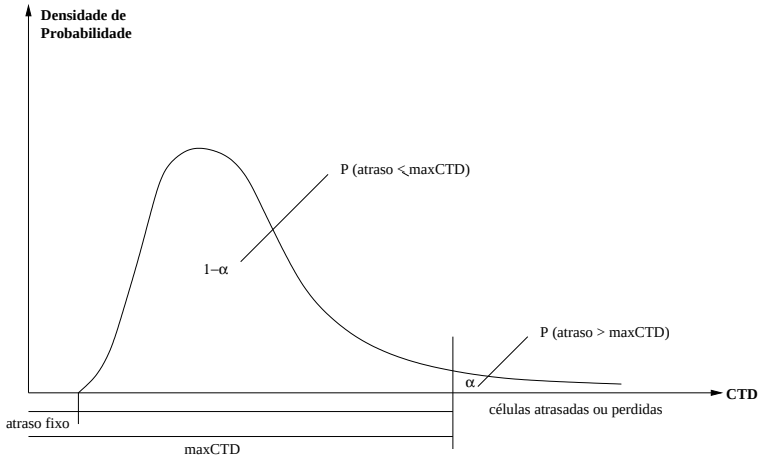


Figura 2.3: Modelo de probabilidade de CTD para categorias de serviços de tempo real. No modelo, α representa a probabilidade de que células excedam o valor de atraso máximo permitido. Fonte: [COM99].

2.3.2.1 One-point CDV

O *one-point CDV* para um célula, em um MP, é a diferença entre o tempo de referência de chegada da célula (c_k)e o real tempo de chegada da célula (c_k) no MP, ou seja, para célula k:

$$y_k = c_k - a_k,$$
$$c_0 = a_0 = 0,$$
$$c_k + 1 = \begin{cases} c_k + T \text{ se } c_k \geq a_k; \\ a_k + T \end{cases}$$

sendo c_k o tempo de referência, a_k o tempo real de chegada de célula, y_k o valor de *one-point* CDV e T um intervalo de tempo esperado. Geralmente, T assume o valor de 1/PCR (*Peak Cell Rate*). PCR é o valor da mais alta taxa de transmissão de uma conexão. Particularidades deste parâmetro serão discutidas na Seção 2.4. A Figura 2.4 ilustra um exemplo de cálculo de *one-point* CDV.

Valores positivos de *one-point* CDV correspondem a *cell clumping* ("chegada antecipada de células"); valores negativos correspondem a *gaps* ("chegada atrasada") no fluxo de células. O padrão de referência definido acima elimina o efeito de *gaps* na especificação e medição de *cell clumping*, já que somente quando $c_k \geq a_k$ o valor de referência c_k é utilizado no cálculo de c_{k+1} e não o tempo real de chegada da célula a_k.

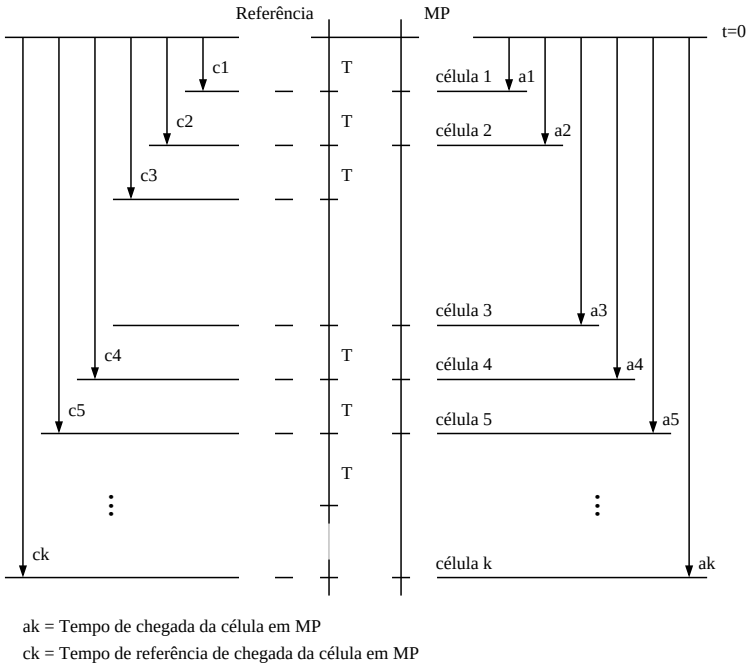


Figura 2.4: Exemplo de cálculo de valores de one-point CDV. Fonte: [UNI00].

2.3.2.2 Two-point CDV

O *two-point CDV* para uma célula entre dois pontos (MP1 e MP2) é a diferença entre o atraso real de transferência da célula entre os dois MPs e um valor referência para o atraso de transferência da célula entre os mesmos MPs. Para célula k:

$$x_k = a_{2k} - a_{1k},$$
$$v_k = x_k - d_{1,2},$$

sendo a_{2k} o tempo real de chegada em MP2, a_{1k} o tempo real de chegada em MP1, x_k o atraso real entre MP1 e MP2, $d_{1,2}$ o valor de referência para o atraso entre MP1 e MP2 e v_k o valor de *two-point* CDV. O método de medição deste valor baseia-se no conceito de *célula 0*. Segundo [UNI00], esse conceito não consta nas especificações atuais, podendo ser objetivo de estudos futuros. A Figura 2.5 mostra um exemplo de cálculo de *two-point* CDV.

Valores positivos de *two-point* CDV correspondem a um atraso maior do que o esperado, ou seja, maior que o valor de referência; valores negativos correspondem a atrasos menores do que o esperado.

2.3.2.3 Acumulação dos valores de QoS

A capacidade de estimar acumulação dos valores dos parâmetros de atraso (maxCTD e p2pCDV) ao longo dos vários nodos de uma conexão são de grande importância para decidir se

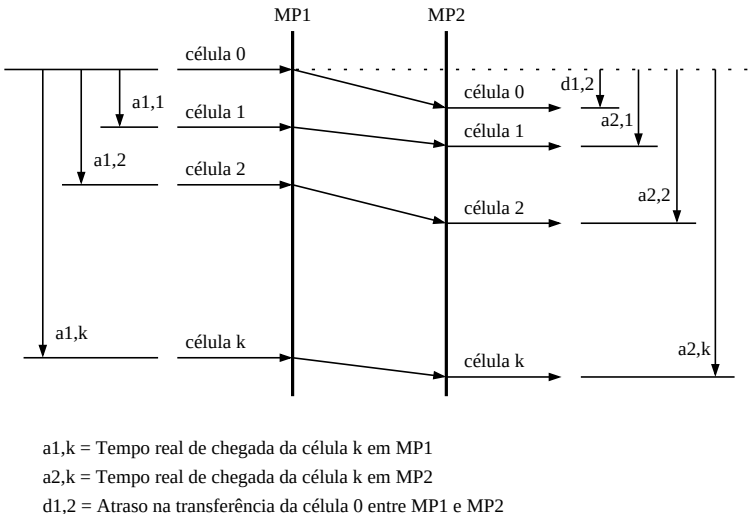


Figura 2.5: Exemplo de cálculo de valores do two-point CDV. Fonte: [UNI00].

a conexão poderá ou não ser estabelecida. Técnicas de acumulação mais sofisticadas permitem uma melhor estimativa de atraso fim a fim, utilizando o parâmetro p2pCDV, minimizando o número de conexões rejeitadas. Essas técnicas baseiam-se em padrões de chegada de células que podem colocar em risco a QoS da conexão. O parâmetro maxCTD é utilizado porque o valor de p2pCDV não expressa o atraso fixo em cada nodo. O acúmulo de maxCTD completa a informação de atraso utilizada pela rede.

2.3.3 *Cell Error Ratio* (CER)

O *Cell Error Ratio* (CER) para uma conexão é definido como sendo:

$$CER = \frac{\text{células com erro}}{\text{total de células}}$$

O *total de células* inclui células transferidas como sucesso e células com erro. No entanto, células transferidas com sucesso ou células com erro que pertençam a um bloco de células com erros severos não devem ser contabilizaras para o cálculo de CER.

2.3.4 *Severely Errored Cell Block Ratio* (SECBR)

Para uma dada conexão, *Severely Errored Cell Block Ratio* (SECBR) é definido como sendo:

$$SECBR = \frac{\text{blocos de células com erros severos}}{\text{total de blocos de células transmitidos}}$$

Bloco de células é uma seqüência de N células transmitidas consecutivamente em uma dada conexão. Na prática, um bloco de células corresponde ao número de células de informação de usuário transmitidas entre sucessivas células OAM [COM99]. Recomendações sobre os tamanhos de blocos de células podem ser obtidos em [UNI99].

2.3.5 *Cell Misinsertion Rate* (CMR)

Este parâmetro de qualidade de serviço é medido sob a forma de uma taxa (*rate*) e não de uma proporção (*ratio*), já que o número de células de conexão inseridas em outra conexão ocorre independentemente do número de células transmitidas ou recebidas em uma conexão [COM99]. A inserção de células em uma conexão particular é freqüentemente causada por um erro de cabeçalho não identificado. O CMR para uma conexão é definido como sendo:

$$CMR = \frac{\text{células mal inseridas}}{\text{intervalo de tempo}}$$

2.4 Descritores de tráfego

Além dos parâmetros de QoS existem também os chamados parâmetros de tráfego. Cada *parâmetro de tráfego* descreve uma característica de uma fonte de tráfego. Os parâmetros de tráfego descritos em [COM99] incluem *Peak Cell Rate* (PCR), *Sustainable Cell Rate* (SCR), *Maximum Burst Size* (MBS), *Minimum Cell Rate* (MCR) e *Maximum Frame Size* (MFS). Um conjunto desses parâmetros é denominado de *descritores de tráfego da fonte*. Os *descritores de tráfego da conexão* especificam as característica de uma conexão ATM. Esses são formados pelos descritores de tráfego da fonte, os parâmetros *Cell Delay Variation Tolerance* (CDVT) e *Burst Tolerance* (BT) e a especificação de conformidade das células de uma conexão (Capítulo 3.1). O parâmetro CDVT (*Cell Delay Variation Tolerance*) especifica quanto de variação pode ser tolerado na chegada de células em um determinado ponto da rede, devido à variação do atraso (CDV) causado por funções da rede. Pelo mesmo motivo, o parâmetro BT indica o quanto de variação no tamanho da rajada (*burst*) pode ser tolerado. Em geral, uma conexão não tem apenas um valor de CDVT. Diferentes valores podem ser aplicados ao longo do caminho da conexão. Descritores são utilizados por vários mecanismos de controle de tráfego, incluindo o controle de admissão de conexões e também na avaliação de conformidade de células da conexão. Os descritores de tráfego utilizados para definir uma conexão dependem do tipo de categoria de serviço que está sendo utilizada. Uma conexão bidirecional deve utilizar dois conjuntos de parâmetros (um para cada sentido) para descrever as características do tráfego.

Para que sejam atribuídos valores a esses descritores de tráfego, é necessário que se tenha um conhecimento prévio do comportamento da conexão. Essas características devem ser fornecidas pela aplicação. A referência [DIE00] propõe uma metodologia que permite determinar o valor dos descritores a partir de uma amostra inicial, mesmo quando a a aplicação não tem condições de especificar seus descritores. Junto com esta metodologia, a mesma referência sugere a utilização de um mecanismo de atualização adaptativo, baseado em uma renegociação dos parâmetros de tráfego para valores condizentes, caso estes estejam acima de valores de tolerância pré-definidos. A relação entre os descritores de tráfego e as categorias de serviço pode ser vista na Tabela 2.1.

2.4.1 *Peak Cell Rate* (PCR)

O parâmetro de tráfego *Peak Cell Rate* (PCR) especifica a maior taxa de transferência de uma conexão. Esse parâmetro permite que a rede reserve recursos para que se possa atingir os objetivos solicitados (por exemplo, *Cell Loss Rate*). O valor de PCR deve ser fornecido à rede em células/segundo. Desta forma, inverso da taxa de pico (1/PCR) representa o intervalo

teórico mínimo entre as chegadas de células de uma conexão em um determinado ponto da rede. Geralmente, este valor é utilizado pela variável T para o cálculo de atraso *one-point* CDV (Seção 2.3.2.1, Página 19). Este parâmetro é utilizado por todas as categorias de serviço definidas em [COM99].

2.4.2 *Sustainable Cell Rate* (SCR)

O parâmetro de tráfego *Sustainable Cell Rate* (SCR) representa um limite superior da taxa média de transmissão em um intervalo de tempo maior que o usado na definição de PCR. Também conhecido como taxa de bit sustentável, esse parâmetro permite à rede, através das funções de UPC, alocar os recursos necessários para garantir os objetivos de desempenho da rede. A quantidade de recursos alocada baseada em SCR é menor do que a baseada em PCR. Utilizado para descrever fontes VBR, o parâmetro SCR permite a multiplexação estatística do fluxo de tráfego de uma fonte. O inverso deste parâmetro, junto com o parâmetro *Burst Tolerance* (Seção 3.5.4) é utilizado pelo algoritmo GCRA para definir tráfegos VBR conformes.

2.4.3 *Maximum Burst Size* (MBS)

Maximum Burst Size (MBS) representa o fator de rajada associado à conexão. O MBS especifica o número máximo de células que podem ser transmitidas pela fonte na taxa de pico. No entanto, esta mesma rajada deve estar de acordo como o parâmetro SCR. As Figuras 2.6 e 2.7 mostram a aplicação dos parâmetro PCR, SCR e MBS para fluxos CLP=0 e fluxos agregados CLP=0+1, respectivamente.

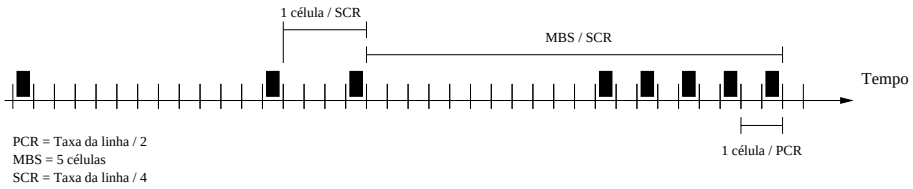


Figura 2.6: Exemplo de um fluxo conforme do ponto de vista dos parâmetros PCR, SCR e MBS para um fluxo CLP=0.

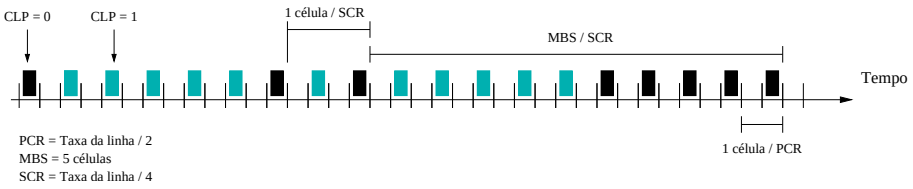


Figura 2.7: Exemplo de um fluxo conforme do ponto de vista dos parâmetros PCR, SCR e MBS para um fluxo agregado CLP=0+1.

2.4.4 *Minimum Cell Rate* (MCR)

O parâmetro *Minimum Cell Rate* (MCR) representa a mínima garantia de QoS dada pela rede à uma conexão. Esse parâmetro é utilizado apenas pelas categorias ABR e UBR. A

Tabela 2.1: Parâmetros e atributos das categorias de serviço.

ITU-T	DBR	SBR	ABT		ABR	
ATM Forum	CBR	rt-VBR	nrt-VBR	UBR	ABR	GFR
Parâmetro de tráfego						
PCR e CDVT	E					
SCR, MBS, CDVT	n/a	E		n/a		
MCR	n/a				E	n/a
MCR, MBS, MFS, CDVT	n/a					E
Parâmetros de QoS						
p2pCDV	E		NE			
maxCTD	E		NE			
CLR	E			NE	R	R
Outros atributos						
Feedback	NE				E	NE

E - Aplicado
NE - Não especificado
R - Específico da rede (pode ou não ser aplicado)
n/a - Não aplicado

utilização desse parâmetro para tráfego GFR depende das implementações específicas de cada rede. GFR significa *Generic Frame Rate*. Esta categoria provê um serviço de largura de banda sob demanda com uma garantia mínima da mesma. GFR caracteriza-se por trabalhar no nível de PDUs *(Protocol Data Unit)* das camadas de adaptação, em especial AAL5. A categoria de serviço GFR não está definida em [COM99].

2.4.5 *Maximum Frame Size* (MFS)

Maximum Frame Size (MFS) define o tamanho máximo de uma unidade de dados (PDU) do protocolo AAL que pode ser transmitida em uma conexão da classe de serviço GFR.

Capítulo 3

Conformidade de tráfego

No capítulo anterior discutiu-se, entre outras coisas, como uma conexão ATM é descrita aos elementos da rede. A partir desta descrição, os elementos da rede podem tomar diversas decisões sobre a conexão. Durante o período de estabelecimento da conexão, é possível decidir se a conexão pode ou não ser estabelecida, utilizando-se um mecanismo de admissão de conexões. Apesar da grande importância desse mecanismo, apenas com sua utilização não é possível garantir que não haverá uma degradação da qualidade do serviço durante o período de vida da conexão. Pode ocorrer que, intencionalmente ou acidentalmente, uma conexão passe a não se comportar mais como descrito no instante do seu estabelecimento. Se não tomada nenhuma atitude, essa conexão passará a consumir recursos que devem ser utilizados por outras conexões, impedindo que as últimas mantenham seus objetivos de QoS. Mecanismos como *formatação de tráfego* (*traffic shaping*), *policiamento de tráfego* (*traffic policing*) e *policiamento suave* (*soft policing*) podem ser utilizados para assegurar que o fluxo de células de uma conexão está conforme ao contrato de tráfego.

O mecanismo de conformação de tráfego é utilizado para deixar o tráfego de acordo com seus descritores. Os demais mecanismos (policiamento de tráfego e policiamento suave) têm como objetivo monitorar as células de uma conexão, identificando células não conformes e reagindo de alguma forma a essas células. Basicamente, eles diferem no tipo de reação tomada quando uma célula não conforme é encontrada. O conjunto de ações tomado pela rede para monitorar e forçar a conformidade do tráfego em UNIs e em NNIs é denominado de *Usage Parameter Control* (UPC) e *Network Parameter Control* (NPC), respectivamente.

A Seção 3.1 apresenta os parâmetros utilizados na verificação de conformidade de tráfego das categorias de serviço. A seguir, a Seção 3.2 descreve os mecanismos de policiamento, conformação de tráfego e os efeitos causados por eles. Na Seção 3.3 são discutidas as características e requisitos do *Usage Parameter Control* (UPC), enquanto na Seção 3.4 é apresentado o problema de conformidade em VPCs. Por último, são discutidos os algoritmos relacionados aos descritores de tráfego, que descreve formalmente estes algoritmos e apresenta a conformidade para PCR e SCR.

3.1 Definição de conformidade de tráfego

A definição de conformidade determina se os descritores de tráfego e os objetivos de QoS são aplicáveis somente a células de alta prioridade (CLP=0) ou fluxos agregados (CLP=0+1). Além disso, esta definição especifica quais as ações devem ser tomadas pela rede em caso de células não conformes. Como discutido na Seção 2.4, a definição de conformidade está

Tabela 3.1: Definição de conformidade para as categorias de serviço

Nome	Categoria(s)	PCR	SCR	MCR	*tagging*	CLP	Max-CTD/p2p-CDV
CBR.1	CBR	0+1	ns	ns	n/a	0+1	0+1
VBR.1	rt-VBR, nrt-VBR	0+1	0+1	ns	n/a	0+1	0+1(rt)
VBR.2	rt-VBR, nrt-VBR	0+1	0	ns	n/a	0	0(rt)
VBR.3	rt-VBR, nrt-VBR	0+1	0	ns	E	0	0(rt)
ABR.1	ABR	0	ns	0	n/a	0	n/a
GFR.1	GFR	0+1	ns	0	n/a	0	n/a
GFR.2	GFR	0+1	n/a	0	E	n/a	n/a
UBR.1	UBR	0+1	ns	ns	n/a	n/a	n/a
UBR.2	UBR	0+1	ns	ns	E	n/a	n/a

E - Aplicado
ns - Não especificado
R - Específico da rede (pode ou não ser aplicado)
n/a - Não aplicado

incluída no contrato de tráfego. Quando a garantia de QoS é aplicada a tráfegos agregados, o campo CLP do cabeçalho da célula é ignorado e as células com CLP=0 e CLP=1 são tratadas sem distinção. Neste caso a definição de conformidade é considerada *transparente ao CLP* (*CLP-transparent*). Quando apenas células com CLP=0 são consideradas, a definição de conformidade é denominada *CLP relevante* (*CLP-significant*).

Quando aplicada a definição de conformidade às categorias de serviço, essas são subclassificadas de acordo com os parâmetros utilizados para verificação de conformidade, ação sobre células não conformes e relevância de CLP. A Tabela 3.1 resume a definição de conformidade para as diferentes categorias de serviço.

3.2 Policiamento e conformação de tráfego

Apenas uma reserva coerente de recursos durante o período de estabelecimento de uma conexão não garante a QoS desejada pelas aplicações. É possível que após o estabelecimento da conexão, esta passe a ter características que ultrapassem os limites impostos pelos seus descritores de tráfego (Seção 2.4), prejudicando as demais conexões previamente estabelecidas. Sendo assim, é dever da rede garantir a QoS para as células conformes. O contrato de tráfego, que está associado a qualquer conexão ATM, especifica as condições de conformidade de tráfego das células e as penalidades a serem aplicadas pela rede, em caso de células não conformes.

Células não conformes podem sofrer penalizações severas (descarte ou degradação de prioridade) ou podem prejudicar consideravelmente a qualidade de serviço exigida pela aplicação. Nas especificações, tanto do ATM Forum como do ITU-T, não há uma exigência formal para que seja feita uma conformação de tráfego pela rede ou pelo usuário. Contudo, devido à penalização de células, uma boa alternativa é a utilização do mecanismo de formatação de tráfego. Como dito anteriormente, esse mecanismo é utilizado para tornar um tráfego não conforme em um tráfego conforme, desta forma evitando o efeito da penalização. Tanto a ação de policiamento quanto a de conformação podem utilizar o algoritmo *Generic Cell Rate Algorithm*

(GCRA) para verificar se as células estão conformes a um dado conjunto de descritores de tráfego. Dois GCRA são sugeridos em [UNI00a] e [COM99]: balde furado (*leaky bucket*) ou escalonamento virtual (*virtual-scheduling*). Para um mesmo padrão de chegada de células, esses algoritmos classificam as células da mesma forma. Esses dois algoritmos são discutidos em detalhes nas Seções 3.5.3 e 3.5.4, quando será tratada a conformidade de tráfego para os parâmetros de tráfegos PCR e SCR.

Os mecanismos de policiamento geralmente são aplicados nos pontos de entrada da rede através de funções de UPC. Por outro lado, a formatação de tráfego pode ser aplicada em todos os pontos de saída dos elementos da rede para amenizar o efeito de *jitters* causados pelo uso de *buffers* na multiplexação. *Jitter* é definido como uma pequena variação na fase de um sinal digital (grandes variações são chamadas de *wander*). Da mesma forma, uma VPC na interligação de duas redes poderá ser conformada na saída da primeira rede, antes de ser repassada para a segunda. A Figura 3.1 mostra os pontos em que estes mecanismos podem ser aplicados para um fluxo de sentido único. Os mecanismos destinados a controlar a conformidade de células podem ser resumidos da seguinte forma:

Policiamento de tráfego: identifica células não conformes (GCRA) e penaliza estas células. Como dito anteriormente, as penalidades a serem aplicadas sobre as células não conformes de uma conexão são especificadas pelo contrato de tráfego. Estas penalidades podem ser *descarte de célula* (*cell discarding*) ou *rotulamento de célula* (*cell tagging*). Ações tomadas pelos mecanismos de policiamento podem ser vistas com mais detalhes na Seção 3.3.

Policiamento suave: o *soft-policing* é uma alternativa ao mecanismo de policiamento de tráfego. Como um mecanismo de policiamento, a principal função do *soft-policing* é garantir a conformidade do tráfego em determinados pontos da rede, mostrados na Figura 3.1. No entanto, ao contrário do policiamento de tráfego, a primeira atitude tomada por esse mecanismo após identificar uma célula não conforme, não é penalizá-la, mas sim dar uma chance para que ela se torne conforme. Para tanto, é necessário a utilização de *buffers* para causar um atraso, até que esta célula possa ser considerada uma célula conforme.

Conformação de tráfego: esse mecanismo altera as características do tráfego de um fluxo de células em uma conexão para alcançar uma melhor eficiência da rede. O conformador de tráfego atrasa a emissão de uma célula até que ela seja considerada conforme pelo GCRA. Contudo, mesmo com a alteração da característica do tráfego, o mecanismo de formatação de tráfego deve manter a integridade da seqüência de células em uma conexão. Esse mecanismo pode ser utilizado para diminuição da taxa de pico, limitação do tamanho da rajada (*burst*) e redução do CDV através de um espaçamento conveniente entre células. Diferentemente dos mecanismos de UPC, a conformação de tráfego pode ser executada diretamente pelo sistema final para garantir que as células geradas pela fonte ou em uma interface UNI ou NNI estejam conforme ao contrato de tráfego.

3.3 *Usage Parameter Control* (UPC)

Como dito anteriormente, funções de UPC não pertencem ao escopo deste trabalho. O objetivo que se pretende atingir engloba apenas o estudo das funções de UPC. *Usage Parameter*

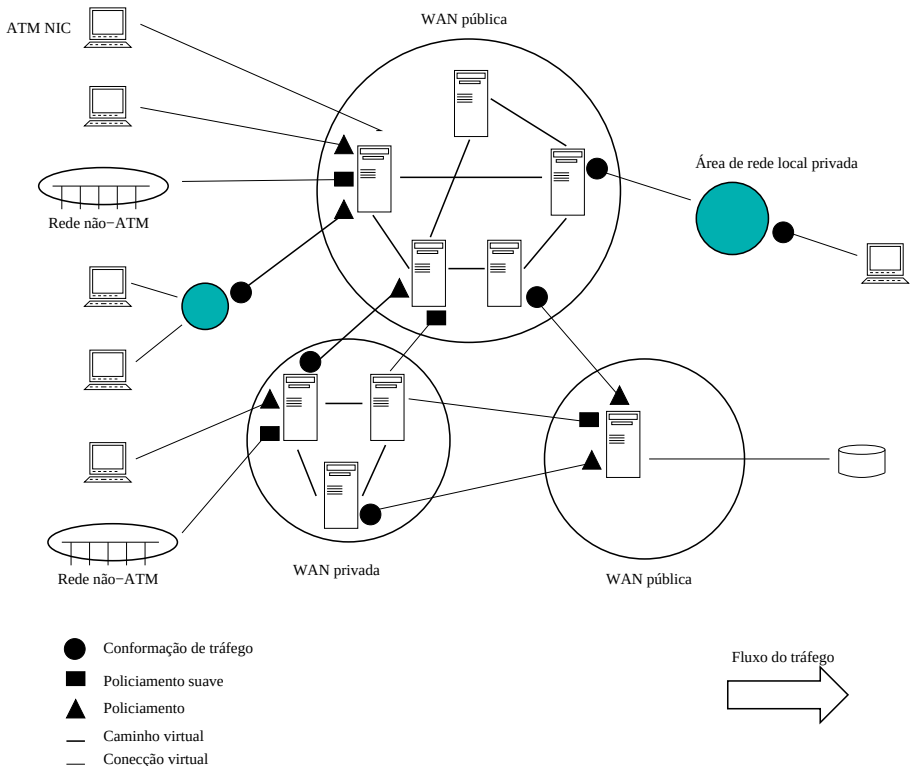


Figura 3.1: Possíveis localizações dos mecanismos de policiamento e conformação de tráfego. Os mecanismos de policiamento (policiamento de tráfego e policiamento suave) são utilizados nos pontos de entrada das redes, enquanto o mecanismo de conformação na saída de uma rede para outra ou na saída de um equipamento terminal. Fonte: [GIR98].

Control é um conjunto de ações tomados pela rede em UNIs para monitorar o tráfego e garantir que uma conexão respeitará o seu contrato de tráfego. Esse conjunto de ações age como um controle de tráfego preventivo, ou seja, procura evitar sobrecargas na rede, e não remediar esse problema depois que ele acontece. Seu principal objetivo é proteger os recursos da rede de conexões que excedam seus limites, consumindo mais recursos além daqueles que lhe foram garantidos pelo contrato de tráfego. O monitoramento é executado para todas as conexões de uma interface onde UPC está sendo usado. Contudo, métodos de monitoração de canais de meta-sinalização e fluxos OAM serão temas de estudos futuros do ATM-Forum [COM99]. O UPC pode ser aplicado tanto a VCCs com a VPCs. Quando aplicado a ambas, a função de UPC é, primeiramente, verificar se a células que chegam ao ponto de monitoração realmente pertencem a uma conexão. Só após isso, serão executadas funções que verificam se a célula está, ou não, conforme. Devido a isto, quando localizados em uma chave, as funções de UPC devem ser realizadas antes da função de chaveamento [COM99].

É uma obrigação da rede garantir a QoS àquelas conexões que estão de acordo com os descritores de tráfego. Os objetivos de QoS não são atingidos apenas com a utilização de UPC, mas muitas implementações de redes ATM utilizam as funções de UPC para garantir seu objetivo. Com isso, uma série de requisitos estão associados a este mecanismo. Um deles é a eficiência de UPC. O atraso causado pela execução das funções de UPC deve ser o menor possível, já que o CTD e CDV introduzidos por estas funções também fazem parte do CTD e CDV alocados pela rede. Outro requisito é quanto a localização de UPC. Recomenda-se que este esteja localizado em um *Virtual Path Link* (VPL) ou em um *Virtual Channel Link*

(VCL) que esteja conectado em um equipamento terminal. A Figura 3.2 mostra as possíveis localizações de UPC.

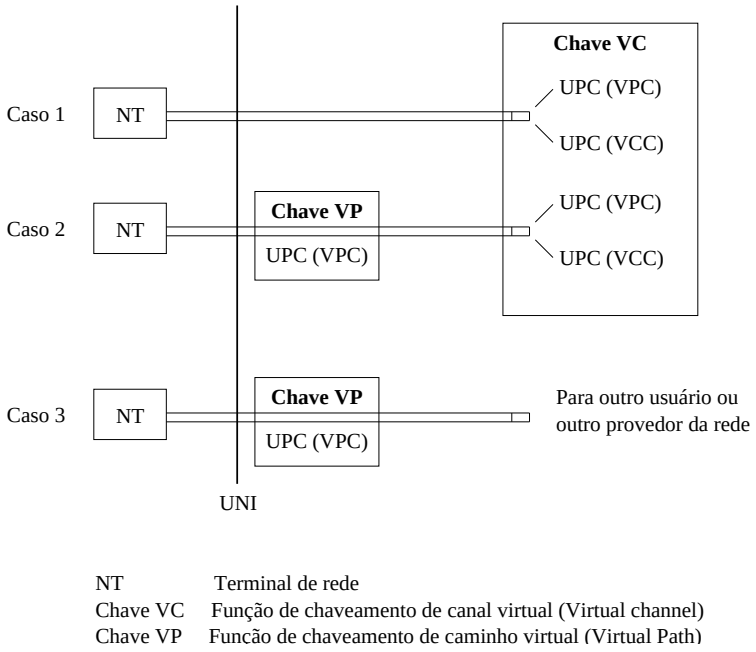


Figura 3.2: Localização das funções de UPC. Fonte: [COM99].

Por último, deve-se considerar as ações que UPC deve realizar sobre as células. De acordo com [COM99], três atitudes podem ser tomadas: não realizar nenhuma ação (*cell passing*), baixar a prioridade da célula (*cell tagging*) ou descartar a célula (*cell discarding*). A atitude de *cell-passing* é utilizada sobre células conformes, enquanto a ação de *cell tagging* e *cell discarding* é utilizada para células não conformes.

Se a opção de *cell tagging* está sendo usada por uma conexão, a função de UPC deve ser capaz de converter células CLP=0 em CLP=1. Essa atitude penaliza as células não conformes. Quando um fluxo agregado CLP=0+1 não está sendo utilizado, células CLP=1 tem menor prioridade do que células CLP=0, condição que prioriza o descarte dessas células em caso de congestionamento na rede. *Congestões* são causadas por sobrecargas em um ou mais ponto da rede. Se a opção de *tagging* está sendo utilizada no objetivo de MCR de uma conexão GFR.2, a função de UPC correspondente deve ser capaz de converter todas as células CLP=0 do quadro em células CLP=1 enquanto este quadro é identificado por UPC. Células que têm sua prioridade diminuída são chamadas de *células rotuladas* (*tagged cell*).

3.4 Problema de conformidade em VPCs

O bit CLP indica a prioridade da célula e é utilizado sem problemas em VCCs. No entanto, o uso deste bit em VPCs pode causar problemas. Várias VCCs com tráfego VBR (VCCs VBR) que contêm células CLP=0 e CLP=1 podem ser agregadas em uma VPC com tráfego VBR (VPC VBR). No ponto de agregação, células CLP=1 não podem ter o seu valor modificado.

O problema de conformidade em VPCs resume-se ao fato de que células com CLP=1 que são consideradas não conformes em uma VCC não são necessariamente consideradas não conformes em uma VPC. Os valores dos descritores da VPC devem caracterizar o comportamento

do tráfego agregado e não o comportamento de cada VCC. Por exemplo, várias VCCs VBR que transportam células CLP=0 e CLP=1 podem ser agregadas em uma VPC VBR. Se um mecanismo de formatação de tráfego for utilizado na VPC, o tráfego agregado deve ser formatado segundo os descritores da conexão VPC. Então, se utilizada alguma formatação de tráfego no ponto de agregação, células que antes eram consideradas não conformes na VCC, passam a ser conformes na VPC. Mas, o problema como foi dito antes, é que apesar destas células serem conformes na VPC elas continuam com o campo CLP=1, ou seja, com menor prioridade que as demais. Sendo assim, em caso de congestionamento estas células poderão ser descartadas antes das demais.

O uso de CLP *transparency* como solução para o problema de conformidade em VPCs é discutido em [GIR98]. Essa solução simplesmente desconsidera o campo CLP e assim, não prioriza o descarte de nenhuma célula em caso de congestionamento. A utilização CLP transparente deve ser lembrada no momento do estabelecimento da conexão. Uma vez que células CLP=0 e CLP=1 têm a mesma prioridade, a largura de banda alocada para CLP=1 deve garantir que estas células não irão afetar o fluxo de células CLP=0.

3.5 Descritores de tráfego e algoritmos relacionados

Funções da camada ATM (multiplexação de células, por exemplo) podem alterar as características da conexão, introduzindo uma variação no atraso das células (*Cell Delay Variation* - CDV). Quando células de duas ou mais conexões são multiplexadas, células de uma dada conexão podem ser atrasadas, enquanto células de outras conexões estão sendo inseridas na saída do multiplexador. Atrasos também podem ser causados devido a *overhead* da camada física ou inserção de células de gerenciamento (células OAM). A variação no atraso de células altera o tempo de chegada entre células consecutivas de uma conexão. As Figuras 3.5 e 3.6, a serem discutidas mais adiante, exemplificam a alteração em um padrão de chegada de células causada por funções executadas em uma chave ATM. O maior limite de *clumping* permitido ocorre quando a variação no atraso é igual *Cell Delay Variation Tolerance* (CDVT).

De acordo com [COM99], em uma UNI pública, o sistema final deve explicitamente ou implicitamente selecionar um valor de CDVT para uma conexão ATM a partir de um conjunto de valores suportados pela rede. Esse conjunto de valores ainda não foi definido e será tema de estudos futuros do ATM-Forum.

Como dito anteriormente, o parâmetro CDVT é importante no uso do *Generic Cell Rate Algorithm* (GCRA). O GCRA utiliza o CDVT para verificar a conformidade do tráfego. O GCRA atua como um algoritmo de decisão para cada célula que chega, ou seja, ele deve verificar se esta célula está conforme ou não. As funções de UPC podem utilizar este algoritmo para garantir a conformidade do tráfego. Apesar deste ser o algoritmo utilizado para definir a conformidade para os diferentes tráfegos, não se exige que a rede utilize este mesmo algoritmo nas funções de UPC.

Segundo [COM99], o GCRA é implementável seja como um algoritmo *Virtual Scheduling*, seja como um algoritmo *Leaky Bucket*. A definição destes algoritmos é apresentada nas Figuras 3.3(a) e 3.4(a), mediante um fluxograma simplificado. O GCRA é definido com os parâmetros de incremento (I) e de limite (L), e utiliza a notação GCRA(I,L). Os dois algoritmos GCRA (*Leaky Bucket* e *Virtual Scheduling*) possuem comportamento equivalente para qualquer intervalo de chegada de células.

Múltiplas instâncias de GCRA, possivelmente com diferentes parâmetros I e L, podem ser aplicadas a múltiplos fluxos (CLP=0 e CLP=0+1) da mesma conexão ou para o mesmo fluxo,

como é utilizado para tráfegos VBR. A conformidade para tráfegos VBR pode ser definida em função da conformidade para PCR e SCR. Então, para determinados casos (conformidade SCR para VBR.2 e VBR.3), uma célula será conforme somente se esta for considerada conforme por todas as instâncias do GCRA. A conformidade de PCR e SCR e suas relações com as categorias de serviço são discutidas abaixo.

3.5.1 Leaky Bucket

O algoritmo *Leaky Bucket* como o próprio nome diz, utiliza uma analogia com um balde furado. Assim, o balde tem uma vazão de entrada e uma vazão de saída. Na entrada, tem-se a chegada de células a diferentes intervalos de tempos (taxas de chegada variáveis). Como explicado acima, em uma rede real, intervalos constantes de chegada célula não ocorrem devido à variação no atraso de células. Na saída do balde, ou seja pelo furo, tem-se uma vazão I (incremento do GCRA). Então se a vazão de entrada é sempre igual ou menor que a vazão de saída do balde, este jamais transbordará.

O nível e o limite do balde são representados por B e L, respectivamente. O limite do balde (L) assume o valor do limite do algoritmo GCRA. Inicialmente, o balde está vazio, ou seja $B = 0$. A medida que as células chegam, um novo valor para B é calculado ($B = max(0, B - (t_a - LCT)) + I$), baseando-se no tempo de chegada da célula (t_a) e no tempo de chegada da última célula conforme (LCT), sendo este último inicializado com o tempo de chegada da primeira célula. O algoritmo decide se o balde eleva, baixa ou mantém seu nível, para cada célula que chega. Se o intervalo de tempo entre a célula n e a célula n-1 é igual a I, o balde mantém o seu nível. Se essa diferença é maior ou menor que I, o balde diminui ou eleva o seu nível proporcional ao valor da diferença. No entanto, percebe-se que o aumento ou diminuição do nível jamais será igual ao incremento (I), porque t_a nunca será igual à LCT. A medida que chegadas antecipadas de células ocorrem, com um pequeno ou grande intervalo entre elas, o nível do balde aproxima-se do limite L (CDVT). Quanto menor o intervalo entre as células antecipadas, mais rápido o nível do balde atingirá o seu limite. Se na chegada de uma célula o nível do balde (B) for maior que o seu limite, esta célula será considerada como não conforme. A Figura 3.3 mostra a definição do algoritmo e a analogia dos seus parâmetros com as dimensões de um balde furado.

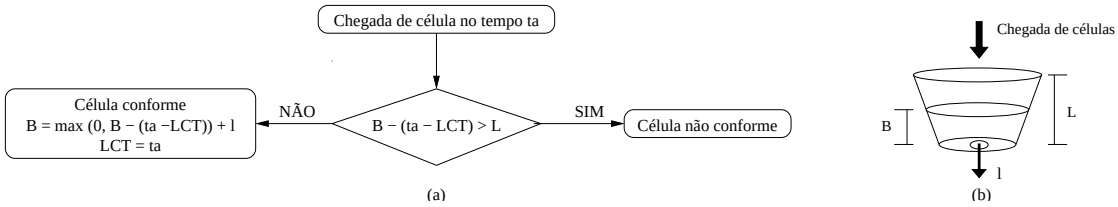


Figura 3.3: (a) Definição formal do algoritmo *Leaky Bucket*. (b) Analogia dos parâmetros do algoritmo *Leaky Bucket* com as dimensões de um balde furado.

3.5.2 Virtual Scheduling

O entendimento do algoritmo *Virtual Scheduling* é mais simples, quando comparado ao *Leaky Bucket*. O *Virtual Scheduling* calcula um tempo teórico para a chegada da célula (TAT), subtrai do tempo real de chegada de célula (t_a) e compara com um limite (L). Se a diferença

entre o tempo real de chegada da célula e o limite teórico para chegada de célula ($TAT - t_a$) é negativo, a célula é assumida como conforme, pois chegou depois do tempo previsto. Se essa diferença é positiva, mas menor que L, significa que a célula chegou antecipada mas não ultrapassou o limite de tolerância ($L > TAT - t_a$). No entanto, se a diferença é maior que L, a célula não só chegou antes do tempo previsto, como também ultrapassou o limite de tolerância. A Figura 3.4 mostra a definição do algoritmo e ilustra a relação entre o tempo real de chegada de célula e o tempo teórico calculado.

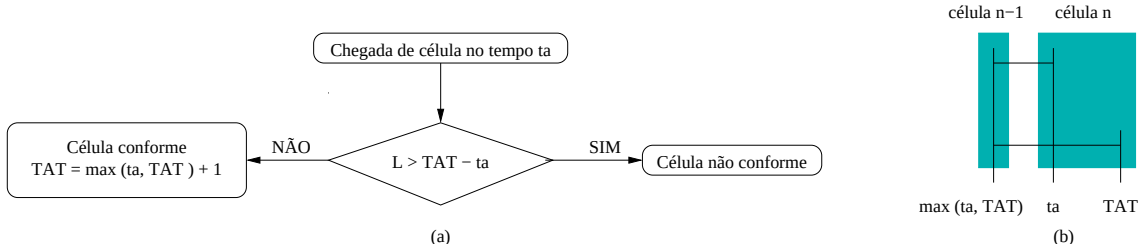


Figura 3.4: (a) Definição formal do algoritmo *Virtual Scheduling* (b). Relação entre o tempo real de chegada de célula e o tempo real calculado pelo algoritmo *Virtual Scheduling*.

3.5.3 Conformidade para PCR (GCRA(CDVT,1/PCR))

Um algoritmo que verifica a conformidade de uma fonte PCR pode ser bastante simples. Se o intervalo de chegada entre células é menor que 1/PCR, a célula é classificada, como conforme, caso contrário, ela é classificada como não conforme. Contudo, se múltiplas conexões são multiplexadas em um enlace, o padrão de tráfego inicial pode ser alteradom como causa da introdução de variação no atraso de células (*jitter*). A Figura 3.5 ilustra um exemplo de alteração no intervalo entre as células causado pela inserção de CDV. A inserção de CDV exige uma verificação menos rígida do que o algoritmo simples descrito acima. Sendo assim, um fator de tolerância é utilizado na verificação da conformidade. O fator de tolerância assume o valor do descritor de tráfego CDVT. O CDVT então passa a ser o limite do algoritmo GCRA (L) expresso em unidades de tempo. Como a conformidade está sendo verificada para PCR, o incremento assume o valor inverso de PCR, também em unidades de tempo. A conformidade de PCR é utilizada pelas categorias de serviço: CBR, rt-VBR, nrt-VBR e GFR.

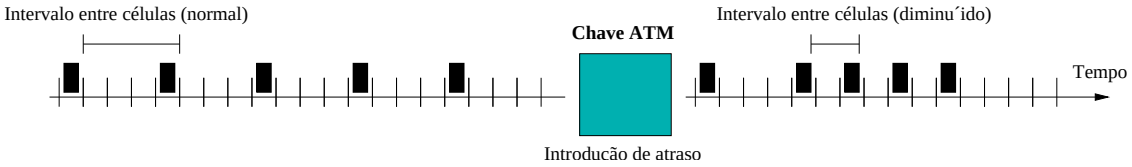


Figura 3.5: Exemplo de alteração no intervalo entre células causado pela inserção de CDV.

3.5.4 Conformidade para SCR (GCRA(CDVT+BT,1/SCR))

O algoritmo utilizado para verificar a conformidade para SCR é o mesmo algoritmo utilizado na verificação de conformidade para PCR. A única diferença está nos parâmetros utilizados

por estes algoritmos. Enquanto o GCRA para PCR utiliza o parâmetro CDVT como limite e 1/PCR como incremento, o GCRA para SCR, utiliza como limite CDVT adicionado ao fator de *Burst Tolerance* (BT) e 1/SCR como incremento. Assim como o atrasos causados a células pelas funções da rede ATM causam agrupamentos de células, este mesmos atrasos também podem causar um efeito de prolongação na rajada. Como discutido anteriormente, o parâmetro CDVT é utilizado como um fator de tolerância para a aproximação de células causada por inserção de atrasos. Da mesma forma, o parâmetro BT é utilizado como um fator de tolerância no aumento do tamanho da rajada. A Figura 3.6 ilustra um exemplo de aumento do tamanho da rajada de células, causado pela inserção de CDV. De acordo com [COM99], o BT pode ser obtido da seguinte forma:

$$BT = (MBS - 1).(1/SCR - 1/PCR)$$

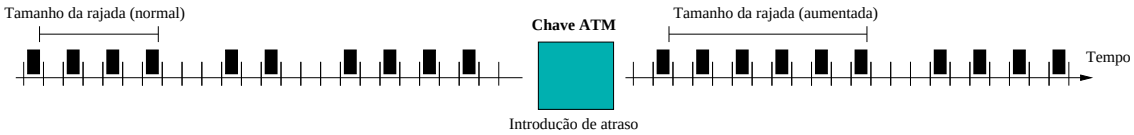


Figura 3.6: Exemplo de aumento do tamanho da rajada de células causado pela inserção de CDV.

A conformidade para SCR é utilizada pelas categorias de serviço rt-VBR e nrt-VBR. A conformação de tráfego VBR utiliza várias instâncias do algoritmo GCRA. Os algoritmos utilizados pela categoria rt-VBR e nrt-VBR são conhecidos como *Dual Leaky Bucket* e *Dual Virtual Scheduling*. O GCRA para VBR utiliza duas instâncias de *Leaky Bucket* ou *Virtual Scheduling*, uma para verificar a conformidade de PCR e outra para SCR. A descrição desses algoritmos para tráfegos VBR.1 pode ser vista nas Figuras 3.7 e 3.8.

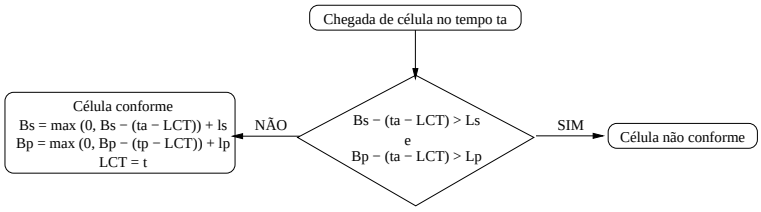


Figura 3.7: Definição formal do algoritmo *Dual Leaky Bucket* para tráfegos VBR.1. Fonte: [GIR98].

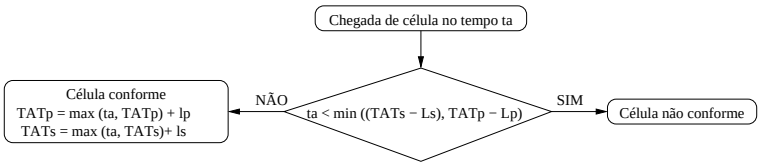


Figura 3.8: Definição formal do algoritmo *Dual Virtual Scheduling* para tráfegos VBR.1. Fonte: [GIR98].

Outra particularidade em tráfegos VBR é a diferença na utilização de fluxos CLP=0 na conformidade de SCR. VBR.2 e VBR.3 não utilizam fluxos agregados na conformidade de SCR. Sendo assim, é necessário que o algoritmo trate de forma diferenciada os diferentes tipos de células (células CLP=0 e CLP=1). Para células CLP=1 apenas a verificação de conformidade para PCR é utilizada, ou seja, se a célula for considerada conforme segundo a instância utilizada pelo algoritmo para validar conformidade PCR, esta célula é considerada conforme pelo algoritmo. Por outro lado, células CLP=0 devem ser consideradas conformes pelas duas instâncias do algoritmo (conformidade para PCR e SCR), e só assim, essas células poderão ser consideradas conformes pelo algoritmo. As Figuras 3.9 e 3.10 ilustram a definição dos algoritmos *Leaky Bucket* e *Virtual Scheduling*, para tráfegos VBR.2 e VBR.3, respectivamente.

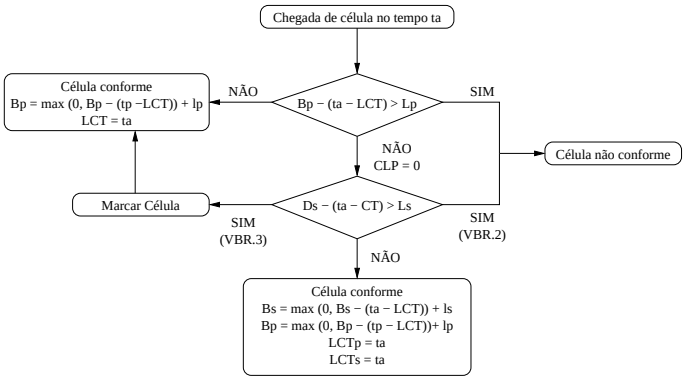


Figura 3.9: Definição formal do algoritmo *Dual Leaky Bucket* para tráfegos VBR.2 e VBR.3. Fonte: [GIR98].

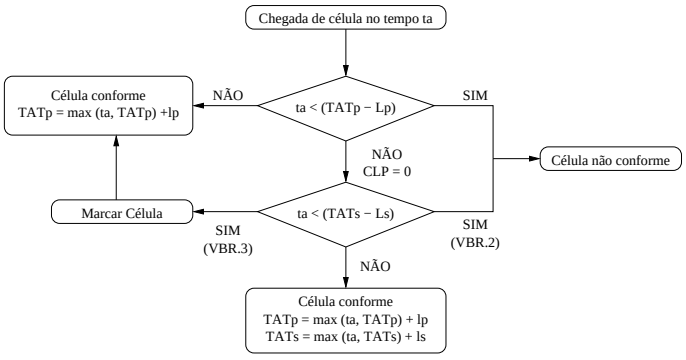


Figura 3.10: Definição formal do algoritmo *Dual Virtual Scheduling* para tráfegos VBR.2 e VBR.3. Fonte: [GIR98].

Como dito anteriormente, as funções de UPC podem ou não utilizar estes algoritmos para verificar a conformidade das diferentes categorias de serviço ATM. Da mesma forma, estes algoritmos também podem ser utilizados na formatação de tráfego.

3.6 Controle de Admissão de Conexões

No momento de estabelecer a conexão é necessário que os descritores de tráfego da conexão sejam analisados para que se determine o mínimo de recursos necessários para garantir a QoS

da mesma. O *Controle de Admissão de Conexões* (*Connection Admission Control* ou CAC) é definido como sendo o conjunto de ações utilizado pela rede no estabelecimento de SVCs. O mecanismo de CAC também é utilizado pelo gerenciamento da rede no estabelecimento de PVCs. Nos dois casos, o CAC é utilizado para verificar se uma conexão pode ou não ser aceita.

Uma conexão ATM tipicamente atravessa várias chaves e dentro delas várias filas. Para uma conexão ser estabelecida, cada elemento pertencente ao caminho da mesma deve verificar se ele possui ou não os recursos necessários para garantir a QoS desta conexão. Geralmente, estes recursos são medidos em termos de largura de banda e espaço alocados nas filas utilizadas como *buffers*.

Os algoritmos de CAC não são especificados pelo ATM Forum ou ITU-T, uma vez que este mecanismo depende da implementação da arquitetura dos escalonadores, da capacidade de processamento das filas e do tamanho dos buffers. Desta forma, uma implementação específica de CAC pode melhor adaptar-se às características específicas dos equipamentos. Além disso, não é necessário que ao longo da rede as chaves tenham a mesma implementação de CAC.

Os métodos de CAC são específicos a cada categoria de serviço. Para a categoria CBR por exemplo, existem diferentes métodos de CAC. Este pode ser bastante simples, decidindo se uma conexão poderá ser aceita baseando-se apenas em PCR. Assim, uma nova conexão será aceita se

$$\sum_i PCR \leq \text{Taxa da linha}$$

Este tipo de método é caracterizado por desprezar o CDV. A introdução de CDV faz com que as células agrupem-se ou afastem-se devido à utilização de *buffers* e intercalação de outros tráfegos no chaveamento. Então, quando células são agrupadas, a taxa de sua transmissão pode ultrapassar PCR, fazendo com que essa conexão consuma mais recursos que o previsto. Em *buffers* pequenos, esta introdução de atraso pode causar um transbordo de capacidade (*overflow*). Sendo assim, este método simples de CAC pode não ser suficiente para garantir a taxa de células perdidas (CLR) exigida pela conexão. Pode-se separar os métodos de CAC para CBR em dois tipos: CDV desprezível e CDV não desprezível.

Existem métodos de CDV desprezível mais precisos que o apresentado. Por exemplo, um desses métodos utiliza a probabilidade de que o *buffer* tenha sua capacidade excedida quando submetido a uma determinada taxa de células. Esta probabilidade é calculada com base em um modelo de fila M/D/1 ou nD/D/1 para cargas altas. Já os métodos que consideram o CDV procuram estimar o congestionamento em escala de rajada. Para isso, é necessário que seja estabelecida uma relação entre o tamanho da rajada e o tamanho do *buffer*. Sendo B o tamanho do *buffer* e BS_i o tamanho máximo da rajada de uma conexão i, a condição $\sum_i BS_i \leq B$ deve ser válida para que não haja perda de células em caso de congestionamento, devido à chegada simultânea de rajadas. Essa maneira pessimista de estimar a utilização dos buffers causa uma subutilização dos recursos da rede. Uma outra maneira, segundo [GIR98], é mapear o tráfego CBR em um equivalente VBR. Desta forma, o mesmo método utilizado para VBR poderá ser aplicado ao tráfego CBR.

A Seção 3.6.1 mostra que quanto maior o número de conexões, maior pode ser a exploração do chamado ganho estatístico. Um outro fator que influi significativamente no ganho estatístico é a *taxa de rajada* (*burstiness*) de uma conexão VBR. A taxa de rajada é definida como a relação entre a taxa de pico (PCR) e a taxa de pico sustentável (SBR). Quanto menor o valor da taxa de rajada (SCR/PCR $<<$ 1), maior será a possibilidade de exploração do ganho estatístico por um método de CAC. Se a condição SCR/PCR $<<$ 1 for válida, um método de

Tabela 3.2: Relação entre os métodos e técnicas de CAC utilizadas pela categoria de serviço VBR.

Método	Técnicas usadas	
	Taxas agregadas	Largura de banda efetiva
REM (*buffers* pequenos, i.e. rt-VBR)	Estima CLR para taxa de chegada de células e garante que a perda é estatisticamente aceitável. Ex. [ROB96].	Estima taxa agregada usando funções geradoras de momento. Ex. [KEL91].
RS (*buffers* grandes, i.e. nrt-VBR)	Acumula contribuições de QoS das conexões existentes e verifica se uma nova conexão viola os objetivos de QoS das já existentes. Ex. [BUF94].	Tolerante a perdas (Considera CLR). Ex. [ELW91] ou Intolerante a perdas (Não considera CLR explicitamente). Ex. [KEL91].

CAC baseado em PCR será considerado ineficiente e pessimista.

Desta forma, se o método de CAC adotar uma avaliação baseada em PCR, ela será pessimista e estará subutilizando os recursos da rede. Por outro lado, se este utilizar como base o valor de SCR, poderá estar comprometendo a qualidade de serviço das conexões. Sendo assim, os métodos de CAC utilizam a chamada *largura de banda efetiva*, *largura de banda equivalente* ou *largura de banda virtual* de uma conexão. O valor da largura de banda efetiva deve encontrar-se entre os valores de SCR e PCR. Quanto menor o seu valor, maior o ganho estatístico.

Para o tráfego rt-VBR, onde o tamanho do *buffer* deve ser pequeno devido aos requisitos temporais, métodos baseados na *multiplexação da envoltória de taxa* (*Rate Envelope Multiplexing* ou REM) podem ser utilizados. Estes métodos de CAC baseiam-se na chamada taxa de chegada agregada (*rate envelope*) para estimar um valor para CLR. Se o valor de CLR estimado for menor que o exigido, a conexão é admitida. Por outro lado, quando *buffers* de grandes dimensões forem utilizados, tipicamente para tráfego nrt-VBR, os métodos baseados na técnica de *compartilhamento de taxa* (*Rate Sharing* ou RS) são utilizados [CAS01]. A Tabela 3.2 relaciona os métodos e técnicas utilizadas no controle de admissão de conexões para a categoria de serviço VBR.

Para as categorias ABR e GFR, o único compromisso da rede é garantir, para cada conexão, a largura de banda mínima expressa em MCR. No entanto, para GFR o mecanismo de CAC deve considerar os parâmetros MFS e MBR, dependendo da largura de *buffer* disponível e da política de gerenciamento de *buffers* que está sendo utilizada. Por outro lado, o tráfego UBR não requer qualquer compromisso com relação à largura de banda. Sendo assim, o mecanismo de CAC para UBR admite conexões sem considerar a largura de banda disponível. Contudo, este método pode ser projetado para aceitar um número limitado de conexões UBR, para evitar a degradação excessiva da taxa das mesmas.

Uma chave ATM, por exemplo, não deve ser capaz de aceitar apenas conexões de uma mesma categoria. No mundo real, os recursos desta chave (largura de banda e filas) devem ser compartilhados por conexões de várias categorias de serviço, onde cada serviço pode estar utilizando sua própria fila. A Figura 3.11 ilustra a utilização de filas distintas de acordo com os

requisitos de QoS. Para garantir a QoS de um tráfego multiclasse, uma quantidade de largura de banda deve ser reservada para cada conexão. Uma maneira simples do mecanismo de CAC verificar se uma conexão pode ser admitida é utilizar a largura de banda efetiva estimada para cada categoria. Desta forma, sendo N_i o número de conexões da categoria i e α_i a largura de banda efetiva da mesma categoria, a conexão será admitida se

$$\sum_i N_i.PCR_i \leq \rho * \text{Taxa da linha}$$

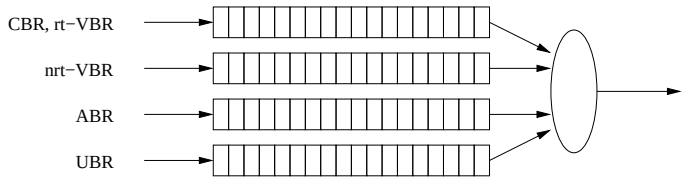


Figura 3.11: Um exemplo de utilização de filas individuais para cada categoria. Na verdade, uma fila pode agrupar uma ou mais conexões dependendo de seus requisitos de QoS. Uma estrutura de prioridades associadas às filas pode ser aplicada para garantir os requisitos de QoS.

Esta relação admite conexões considerando todas elas como sendo de mesma prioridade. O método apresentado em [BER98] considera conexões com diferentes prioridades. Métodos como este permitem que um maior número de conexões possam ser admitidas quando comparados ao método anterior.

3.6.1 Ganho estatístico

Ignorando o CDV, uma vez que a taxa máxima de emissão de células de uma conexão é expressa por PCR, o mecanismo de CAC não precisa alocar os recursos baseados em uma taxa maior que PCR. Contudo, uma vez que as conexões freqüentemente não enviam dados a uma taxa contínua, é possível que a alocação dos recursos seja baseada em uma taxa menor que PCR, quando várias conexões estão sendo multiplexadas em uma chave. Desta forma, o chamado *ganho estatístico* permite que menos recursos sejam alocados para cada conexão, e conseqüentemente um maior número de conexões seja admitido. Ganho estatístico pode ser definido como sendo:

$$\text{Ganho estatístico} = \frac{\text{Número de conexões admitidas com multiplexação estatística}}{\text{Número de conexões admitidas baseadas em PCR}}$$

Como dito anteriormente, um método de CAC eficiente deve alcançar o maior ganho estatístico possível sem colocar em risco o compromisso da rede com a qualidade de serviço das conexões. Como pode ser observado na equação acima, um maior ganho estatístico pode ser obtido com uma elevação na quantidade de conexões admitidas com multiplexação estatística. Geralmente, a multiplexação estatística é função do tamanho dos *buffers*, das características do tráfego e dos objetivos de QoS das conexões que estão sendo multiplexadas. Para aquelas conexões que não têm restrição quanto ao retardo, um maior ganho estatístico pode ser obtido com a utilização de *buffers* grandes. Sempre que se utilizam *buffers* podem ocorrer fenômenos de congestionamento. Esta pode ser de dois tipos fundamentais: congestionamento em escala

de célula e congestionamento em escala de rajada. O congestionamento em escala de célula ocorre em *buffers* pequenos devido à chegada simultânea de células de diferentes conexões. Por outro lado, o congestionamento em escala de rajada ocorre em *buffers* grandes quando ocorre chegada simultânea de rajadas de diferentes conexões.

Nas categorias CBR e rt-VBR, os requisitos de atrasos limitam o tamanho dos *buffers*. Para essas categorias, o congestionamento ocorre apenas em escala de célula. Com a utilização de *buffers* pequenos, é extremamente difícil de se obter um ganho estatístico. Por outro lado, tráfegos sem requisitos temporais, como o nrt-VBR, permitem um ganho estatístico maior com a utilização de *buffers* de maiores dimensões. Para essa categoria, congestionamento nas chaves ocorre não apenas no nível de célula, mas também no nível de rajada.

Capítulo 4

Avaliação de desempenho de módulos ATM

Esta Seção apresenta os principais fatores envolvidos na avaliação de desempenho em redes ATM e as dificuldades enfrentadas pelos métodos utilizados nesse processo. Ao final da Seção deverá ser possível enumerar um conjunto de problemas que devem ser resolvidos para viabilizar e tornar significativo o processo de avaliação, em particular, para módulos de *hardware*.

Redes com altas taxas de transmissão impõem uma série de desafios aos métodos utilizados no apoio de avaliação de desempenho. Em um pequeno período de tempo, pode ocorrer um número muito grande de eventos, dependendo da resolução de tempo utilizada. Cada evento cria uma mudança no estado da rede. Esta mudança depende da alteração de cada elemento da rede, que por sua vez depende da mudança das partes que o compõem, ou seja, dos seus módulos. Além disso, o processo torna-se mais complicado dependendo do nível de detalhe de cada módulo.

A Seção 4.1 apresenta a classificação para os diferentes níveis de avaliação de desempenho. A Seção 4.2 mostra os principais métodos utilizados no mesmo processo, incluindo suas vantagens e desvantagens. A Seção 4.3 enumera os principais fatores a serem considerados no projeto de uma ferramental de medição de desempenho de módulos de *hardware* ATM. Finalmente, a Seção 4.5 discute os modelos suportados pela fonte de tráfego implementada como parte deste trabalho (Seção 6).

4.1 Níveis de avaliação de desempenho

Devido à complexidade do processo, a avaliação de desempenho pode se dar em diferentes níveis de abstração e de resolução de tempo. O nível de abstração depende da liberdade dada pelo método e ferramenta de avaliação e do esforço de compilação envolvido. O *custo de compilação* é definido como o tempo de preparação necessário para a execução do processo de avaliação, enquanto o *custo de execução* é o tempo necessário para a execução do processo de avaliação.

Os níveis de resolução de tempo são definidos a partir dos fatores que caracterizam o tráfego ao longo do tempo. Segundo [COS94], o tráfego em uma rede ATM mostra um comportamento que pode ser caracterizado através dos seguintes níveis de resolução de tempo: nível de calendário, nível de conexão, nível de diálogo, nível de rajada e nível de célula.

O nível de calendário descreve a variação do tráfego de um dia, de uma semana ou de uma

estação do ano. Por sua vez, o nível de conexão descreve uma variação em um período mais curto de tempo. A variação do tráfego, no nível de conexão, é causada pelo estabelecimento e liberação de conexões. O período típico de duração de uma conexão é entre 100 e 1000s, dependendo do tipo de serviço da mesma [COS94]. O nível de diálogo descreve o comportamento dos usuários de cada serviço. No caso de serviço unidirecional, o nível de diálogo não existe.

O nível de resolução de rajada é um dos mais importantes na avaliação de desempenho. Isto porque a multiplexação estatística explora a variação das características do tráfego causada pelas rajadas para obter o ganho estatístico. As rajadas ocorrem devido à natureza dos serviços. Por isso, em ATM a modelagem de diferentes serviços geralmente é feita neste nível. Por último, o nível de célula descreve o comportamento na variação do tráfego causado a partir de cada célula. Por exemplo, para um enlace de 622Mbps uma avaliação no nível de célula deve ter uma resolução de $0,6819\mu s$ que corresponde ao intervalo mínimo entre células. Neste documento, o termo intervalo entre células corresponde ao tempo entre o início de células consecutivas.

A escolha do nível de resolução reflete em fatores como custo de execução e o nível de gerenciamento de tráfego que poderá ser avaliado. Em um nível de célula, significa que um evento ocorre a cada $0,6819\mu s$ se a taxa do enlace é de $622Mbps$. Um evento pode ser considerado a presença ou a ausência de uma célula. Neste caso, em um segundo ocorrem aproximadamente 1.466.490 eventos, onde cada evento causa uma mudança no estado dos módulos da rede. Técnicas de avaliação de desempenho como simulação computacional exigem um alto custo de execução quando submetidas a altas resoluções de tempo.

A Seção seguinte apresenta os métodos mais utilizados na avaliação de desempenho. Nela será comentada a limitação de cada método, considerando a execução dos mesmos em diferentes níveis de abstração e resolução de tempo.

4.2 Métodos de avaliação de desempenho

Segundo [POS98], a complexidade inerente aos sistemas de comunicação requer o projeto e avaliação dos aspectos sistêmicos em diferentes níveis de abstração. Isto significa que o processo de avaliação deve ser realizado em várias etapas da criação do sistema. O processo de avaliação de desempenho de módulos ATM, em geral, é uma tarefa trabalhosa. As soluções mais utilizadas para execução desta tarefa são as experimentações com redes ATM reais, métodos de modelagem analítica e simulações computacionais. Todas elas possuem vantagens e limitações, e embora elas não sejam mutuamente exclusivas, cada uma é melhor aplicável a uma determinada situação particular.

Experimentos com redes reais, por exemplo, é o único método que provê dados precisos para a avaliação de desempenho, já que utilizam, uma implementação física do módulo para tal, ou seja, um protótipo do produto final. Além disto, este método permite uma análise baseada no comportamento real do sistema, isto é, após a análise não há uma fase de implementação para se obter o produto final. Quando outros métodos são utilizados, a implementação pode apresentar um comportamento bastante diferente do modelo usado na fase de avaliação. O experimento com rede reais permite validar o desempenho de uma especificação que representa grande parte das características do módulo. Contudo, este método apresenta uma série de desvantagens, que em grande parte dos casos torna inviável a sua utilização na avaliação de desempenho. O alto custo é um dos principais motivos. Dependendo do nível de detalhe desejado, medidas diretas podem requerer um conhecimento completo do ambiente, equipamentos

caros e pessoal especializado. Além disto, o tráfego de uma rede real é variável e imprevisível, o que torna difícil a análise de um módulo da rede segundo um padrão específico de tráfego.

Assim como o experimento com redes reais, o método analítico pode apresentar resultados com grande precisão. Contudo, quando se trata de redes de alta velocidade, este método apresenta sérias restrições. A fim de manter os métodos analiticamente tratáveis, várias simplificações são feitas, o que pode produzir resultados bastante limitados em termos de confiabilidade. A teoria de filas é uma excelente ferramenta de auxílio aos métodos analíticos.

O método de simulação é o mais flexível deles. Algumas de suas vantagens são a modelagem tão realista quanto necessária de fontes de tráfego, a facilidade de realização de testes (desconsiderando o tempo necessário para tal) e a análise com diferentes níveis de abstração e de resolução de tempo. A principal limitação deste método é a demanda por alto desempenho de processamento. Dependendo do nível de detalhe, a simulação de um determinado modelo pode ser muito demorada, em alguns casos até mesmo inviável. Por exemplo, de acordo com [KUH95], operações de tempo real de um segundo no nível de transferência entre registradores correspondem a aproximadamente 100 dias de simulação em uma SUN SPARC 10/512 com 224 MB de RAM. Técnicas de aceleração podem ser utilizadas para diminuir o tempo de execução das simulações. Neste método, é o modelo do sistema que determina o nível de abstração e a precisão dos resultados. Para se obter resultados próximos da realidade, é necessária uma tradução precisa da especificação a ser avaliada para o modelo específico do simulador em questão. Desta forma, é possível dizer que método de simulação torna-se inadequado a medida que o nível de abstração do modelo diminui e a resolução de tempo aumenta, devido ao custo de compilação e execução.

Os métodos de simulação e modelagem analítica são os mais utilizados na avaliação do desempenho de módulos ATM. Por exemplo, [TSO00a], [TSO00] e [SON97] utilizam estes para avaliar suas propostas. Devido à grande demanda por estes métodos, existem muitos trabalhos na literatura que propõem modelos e técnicas para aprimorá-los. Uma alternativa a estes métodos é a utilização de equipamentos comerciais no processo de validação de uma implementação (protótipo). Equipamentos de teste como o HPE5200A e o HP4200B[PAC98] foram construídos especificamente para este fim. Apesar da qualidade, o preço a que são oferecidos estes equipamentos torna inviável a utilização dos mesmos em grande parte dos projetos, sobretudo em ambientes acadêmicos.

4.3 Avaliação de módulos de *hardware* ATM

Uma ferramenta para medição de desempenho de módulos de *hardware* ATM deve executar basicamente duas tarefas: a geração de estímulos ao *hardware* ATM (HA) e a coleta de informações que revelam o comportamento do HA segundo os parâmetros de qualidade de serviço.

Para que o processo seja válido é necessário que o HA esteja sendo avaliado, tanto quanto possível, sob as mesmas condições em que irá trabalhar. Assim, grande parte do esforço do projeto de uma ferramenta de avaliação é gasto no processo de geração de estímulos. Considerando que o HA não apresenta nenhuma abstração, ou seja, todos os detalhes de funcionalidade e temporização estão expressos nele, um alto nível de resolução de tempo é necessário em sua avaliação. Por exemplo, para um enlace de 622Mbps, uma fonte deve ser capaz de gerar uma célula a cada $0,6819\mu s$. Considerando que a interface comum para um elemento típico que trabalha a 622Mbps é de 16 bits, uma nova palavra deve ser gerada a cada $0,02573\mu s$. Para trabalhar em uma resolução de tempo tão alta é necessário que a geração de

células seja feita por *hardware*.

A geração de células sendo executada por *hardware* torna a ferramenta menos flexível, uma vez que a mesma deve ser capaz de gerar vários padrões de tráfego. Tipicamente, o padrão de geração de células de um tráfego é modelado segundo um processo específico ou um processo modulado. Detalhes sobre modelos de tráfego são apresentados em [COS94]. Na maioria das vezes, recursos de memória são utilizados para dar uma maior flexibilidade à fonte de tráfego. Algumas propostas encontradas na literatura são discutidas na próxima Seção.

4.4 Revisão bibliográfica

Esta Seção comenta algumas fontes utilizadas na pesquisa em modelagem de tráfego e em propostas de ferramentas destinadas à avaliação de desempenho de módulos de *hardware* ATM. A pesquisa realizada teve como principal objetivo investigar propostas de suporte de *hardware* e *software* para a geração de estímulos e coleta de informações de módulos de *hardware* ATM.

A referência [COS94] traz uma revisão sobre modelagem de tráfego de voz, dados e vídeo, apresentando detalhes sobre o comportamento estatístico de cada tráfego. Além disso, é apresentada uma série de critérios a serem considerados na escolha de um modelo de tráfego. Quanto à modelagem de fontes de tráfego, é apresentada uma classificação para os diferentes modelos e seus métodos de parametrização. Por último, relaciona-se uma série de serviços de uma rede de telecomunicações aos modelos de tráfego apropriados aos mesmos.

O trabalho de Leland, Taqqu, Willinger e Wilson [LEL93] causou um grande impacto na engenharia de redes. Este demonstrou que o tráfego em uma rede local Ethernet tem um comportamento em rajada para várias escalas de tempo, e que nenhum dos modelos de tráfego usados até aquele momento era capaz de capturar este comportamento. Um tráfego em rajadas em muitas ou todas as escalas de tempo pode ser descrito estatisticamente usando a noção de *auto-similaridade*. Um fenômeno *auto-similar* apresenta similaridades estruturais através de um longo intervalo de escalas de tempo. Trabalhos posteriores, como o apresentado em [SAH99], observaram a natureza auto-similar em vários outros tipos de tráfegos, tais como ATM, vídeo digital comprimido e tráfego na Web. A natureza auto-similar de muitos tráfegos reais colocou em dúvida os resultados obtidos com os modelos de tráfego tradicionais. Ainda, esta referência mostra que tal comportamento tem uma série de implicações para o projeto, controle e análise de redes de alta taxa de transferência. A referência [SAH99] traz um estudo específico sobre natureza auto-similar dos tráfegos, considerando suas causas e conseqüências. Segundo [SAH99], após a descoberta do fenômeno auto-similar, os teóricos de rede dividiram suas opiniões: uns afirmam que a teoria de redes deveria ser revista, enquanto outros afirmam que deveria ser descartada. Contudo, ainda hoje trabalhos vêm sendo realizados a fim de responder questões relevantes à comunidade de controle de tráfego sobre a auto-similaridade [SIL00].

A referência [KEI95] não discute modelos de tráfego, mas uma ferramenta para a análise de desempenho de chaves ATM. Esta descreve a teoria e aplicação da metodologia da geração do tráfego de teste que está sendo utilizada pela ferramenta. Esta ferramenta mede parâmetros de desempenho tal como CLR, CTD, CDV e CER. Também é descrito um algoritmo matemático que pode ser usado para a geração de vários padrões de tráfego, por exemplo para dados, voz e vídeo. A função fundamental do gerador de células de teste é criar níveis de variação e distribuição do fluxo de células ATM, baseado em uma distribuição estatística oferecida pelo usuário. O suporte em *hardware* oferecido por esta proposta dá uma boa flexibilidade ao gerador de tráfego, permitindo até mesmo a execução de processos modulados. Contudo,

este utiliza um tamanho de memória que pode ser considerado inapropriado para alguns casos. Por exemplo, a memória pode ser considerada insuficiente para representar alguns modelos de tráfego ou indisponível em alguns recursos de *hardware*, tais como FPGAs. Além disso, o projeto do *hardware* da fonte, apesar de permitir execução de processos modulados, não permite que na coleta das células sejam extraídas algumas informações no nível de rajada. Essas informações podem ser obtidas utilizando-se um número de seqüência de rajadas.

Esta é uma das contribuições da proposta descrita em [HON98]. Nesta referência, o projeto e a implementação de um gerador de tráfego são apresentados. O sistema proposto deve ser utilizado junto a um computador pessoal do tipo PC. Segundo o autor, foi realizada uma cuidadosa alocação das funções de *hardware* e *software* para que este possa trabalhar *on-line*, a 155,52Mbps e ainda permitir a emulação de diferentes padrões de tráfego. A utilização de interfaces padronizadas com a camada física, a capacidade de produzir 16 fluxos independentes de tráfego e a utilização de uma camada de software para auxiliar na tradução do modelo de tráfego se destacam entre as vantagens desta proposta.

Por último, [POS98] propõe a integração de um simulador de redes com um simulador VHDL e com uma placa de teste de *hardware* para compor um um ambiente de co-verificação a nível sistêmico. Esta proposta tem como vantagem, permitir a verificação funcional de modelos HDL através do reuso de padrões de teste e modelos de tráfego existentes nos simuladores de rede. Também permite o acesso à análise dos simuladores e aos vários níveis de abstração oferecidos pelos mesmos. A principal justificativa desta proposta é o tempo gasto na construção de *testbenches* para teste funcional do *hardware*. Este tempo chega a atingir mais de 50% do tempo de projeto.

A proposta de uma ferramenta para avaliação de módulos de *hardware* ATM descrita no Capítulo 6 procura combinar algumas vantagens das propostas apresentadas nesta Seção. Esta revisão bibliográfica mostra que algumas ferramentas são suficientemente qualificadas para ajudar no processo de avaliação de desempenho. Contudo, a disponibilidade das mesmas é um problema ainda não resolvido, devido a obrigatoriedade da utilização de *hardware*. Além de trazer outras vantagens, uma ferramenta de avaliação baseada em FPGAs e núcleos de propriedade intelectual (IP) do tipo *soft* é uma solução viável para o problema, combinando a disponibilidade de FPGAs à facilidade de distribuição de núcleos IP *soft*. A Seção seguinte apresenta um resumo sobre modelos tráfego, enfatizando aqueles suportados pela ferramenta proposta.

4.5 Modelos de Tráfego ATM

A caracterização de fluxos de células geradas por fontes ATM tem sido investigada no mínimo há dez anos. Várias propostas de modelos de fontes têm surgidos enfatizando características como simplicidade, realismo e precisão. Freqüentemente, esses modelos de fontes têm sido usados para avaliar algoritmos de policiamento de tráfego e controles de admissão de conexões ou para avaliar o desempenho de arquiteturas de chaveamento. Quanto mais precisa a caracterização, mais precisas serão as estimativas dos parâmetros de desempenho. Desta forma, mais previsível será o comportamento da rede.

Segundo [COS94], alguns critérios podem ser utilizados na escolha de uma fonte de tráfego:

Semelhança com as fontes reais: O modelo da fonte deve aproximar-se de uma fonte real de tal forma que as características relevantes do tráfego real sejam incorporadas no modelo da fonte.

Generalidade: Esta é a capacidade que o modelo tem de representar diferentes tipos de fontes reais, ou seja, de gerar tráfegos com diferentes características. Para ser genérico, um modelo deve trocar o comportamento estatístico do seu tráfego apenas mudando os valores de seus parâmetros sem trocar sua definição. Este é um critério importante a ser considerado no projeto de um ambiente para medição de desempenho em *hardware*, onde o custo de compilação para a troca de um modelo é alto.

Tratabilidade Analítica: De um ponto de vista analítico, este critério significa que o uso de um modelo de tráfego em várias aplicações leva a soluções que podem ser tratadas por uma computação numérica. Assim, não apenas a precisão do modelo é importante mas também a precisão com que o método de solução pode ser usado.

Método de Parametrização do Modelo: Todos os modelos estatísticos necessitam ser parametrizados. Modelos que têm N parâmetros exigem soluções de N equações. Assim, se $N > 2$, as equações de média e variância não são suficientes. Na análise de alguns modelos, a função de geração de probabilidades utilizada pelo modelo é freqüentemente utilizada para ajudar a derivar expressões de média, variância e autocorrelação.

Número de Parâmetros: O número de parâmetros é proporcional a complexidade do modelo, ou seja, modelos com uma quantidade excessiva de parâmetros são indesejáveis.

Os modelos de tráfego podem ser divididos de uma forma geral em duas classes: *dependência de curta distância* e *dependência de longa distância* [YUA02]. Os modelos de dependência de curta distância apresentam uma estrutura de correlação que decresce a uma taxa exponencial ou até mais rápido. Já, um modelo de tráfego de longa distância têm uma estrutura de correlação que decresce a uma taxa mais lenta que uma exponencial. *Estrutura de correlação*, como o próprio nome diz, estabelece uma relação entre as chegadas. Esta relação tende a diminuir com o passar do tempo. Dentre os modelos de dependência de curta distância estão os processos de renovação, processos de Markov, processos modulados por Markov e modelos de regressão. Modelos de dependência de longa distância incluem ARIMA, superposição de fontes ON-OFF, utilização de distribuições de Poisson moduladas por Pareto. A seguir são mostrado alguns modelos de tráfego para as duas classes. Esta Seção não tem o objetivo de conter uma revisão sobre fontes de tráfego, mas sim mostrar algumas características dos modelos de tráfego suportados pelo projeto de *hardware* da fonte proposta. Detalhes sobre os modelos de tráfego podem ser encontrados em [COS94, FIL96, ADA97, YUA02, ADU02].

4.5.1 Processos de Renovação

Em um processo de renovação, os tempos entre chegada de células são independentes e distribuídos identicamente. Assim, não existe nenhuma correlação entre os tempos de chegada de células, o processo é completamente definido pela distribuição de probabilidade usada.

Os modelos de renovação têm sido usados na modelagem de tráfego por sua simplicidade matemática. Existem dois casos importantes nos processos de renovação: *Poisson* e *Bernoulli*. Em um processo de Poisson, o tempo entre chegadas de células são distribuídos exponencialmente. O processo de Bernoulli é análogo ao processo de Poisson para tempo discreto. O tempo entre chegadas de células em um processo de Bernoulli é distribuído de forma geométrica.

O processo de Poisson pode ser caracterizado como um processo de renovação em que os intervalos entre chegadas de células A_n são exponencialmente distribuídos com parâmetros λ :

$PrA_n \leq t = 1 - e^{-\lambda.t}$ (*função de distribuição de probabilidade* ou PDF). De forma equivalente, este processo pode ser considerado como um processo de contagem satisfazendo $Pr\{N(t) = n\} = e^{-\lambda.t}.\frac{(\lambda.t)^n}{n!}$ (*função de densidade de probabilidade ou pdf*). Um processo estocástico $\{N(t), t \leq 0\}$ é chamado de *processo de contagem* se $N(t)$ representa o número total de eventos que ocorreram até o tempo t.

Para o processo de Bernoulli, a probabilidade de uma chegada em um slot de tempo é p e independente entre os *slots*. A probabilidade de que aconteçam k chegadas em n *slots* é dada por uma distribuição binomial onde $Pr\{N_k = n\} = \binom{k}{n} p^n (1-p)^{k-n}$ (pdf), com n entre 0 e k. O tempo entre chegadas é geométrico com parâmetros $p : PrA_n \leq j = p(1-p)^j$ (PDF), sendo j um valor inteiro não negativo.

Tipicamente, em estudos de desempenho de redes ATM, a geração de células para uma única fonte de dados ou voz era caracterizada por um processo de chegada de Poisson ou Bernoulli. Contudo, os processos de Poisson ou Bernoulli não são capazes de capturar a presença de rajadas em um ambiente de tráfego ATM e nem a correlação de tempo entre chegada de células.

4.5.2 Processos Modulados

Talvez o processo modulado mais simples seja o GMDP. GMDP significa *General Modulated Deterministic Process* ou seja, processo determinísticos modulados por uma distribuição qualquer. GMDP é baseado em uma máquina de estados finta com N estados. A Figura 4.1 ilustra um processo modulado com três estados ($N = 3$).

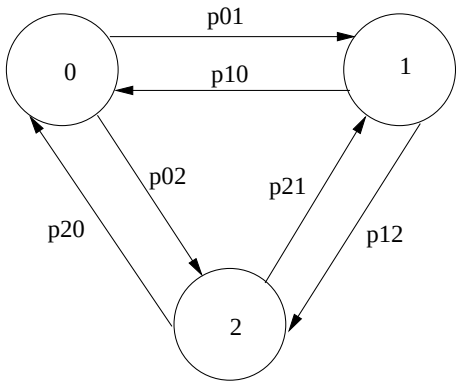


Figura 4.1: GMDP de três estados.

Em cada estado, células são geradas com tempo constante de intervalo entre as mesmas denominado de d_i, onde i identifica o estado. O número de células (X_i) que são emitidas em um estado i pode ter uma distribuição discreta qualquer. Em geral, os processos modulados incluem estados de silêncio, onde nenhuma célula é produzida. O tempo de permanência para esses estados pode ter uma distribuição discreta qualquer.

Um caso comum em GMDPs é utilizar uma distribuição geométrica com no mínimo uma célula para X_i. Neste caso, o processo é chamado de *processo determinístico modulado por Markov* ou *Markov Modulated Deterministic Process* (MMDP). Este tipo de processo modulado pode ser descrito como uma cadeia de Markov de tempo e espaço discreto. Modelos do tipo MMDP foram largamente utilizados para representação de tráfego gerado por fontes de vídeo [COS94].

Também são muito comuns os *Processos de Poisson Modulados por Markov* ou *Markov Modulated Poisson Processes* (MMPP). Neste caso, os processos de Poisson têm sua taxa de chegada modulada por uma cadeia de Markov de estados discretos e tempo contínuo. O tempo de permanência no estado i tem uma distribuição exponencial negativa. A utilização de MMPP permite uma modelagem de fontes variando no tempo e ao mesmo tempo mantém uma solução analítica baseada na teoria de filas [ADA97]. A referência [COS94] cita dois casos onde a utilização de MMPP emula o comportamento de fontes de voz (ON-OFF) com uma boa fidelidade na geração do tráfego, e conseqüentemente com uma boa estimativa de probabilidade de perda de células.

Um caso especial dos processos modulados é o modelo ON-OFF. Este é uma cadeia de Markov de dois estados com alternância entre os estados ativo e inativo (ou de silêncio). No estado ativo, células são emitidas com intervalos constantes. Nota-se que este é um caso especial de GMDP. Os modelos ON-OFF foram originalmente introduzidos com um domínio de tempo contínuo. Isto significa que a duração dos estados ativo e inativo são uma variável de tempo contínuo. O tempo de permanência nos estados ativo e inativo pode ser gerado por uma distribuição geométrica. Neste caso, o modelo ON-OFF equivale a um MMDP de dois estados, sendo um deles o estado de silêncio.

Talvez o modelo ON-OFF mais utilizado seja o *Interrupted Poisson Process* ou IPP. Este é um caso especial de MMPP. Neste modelo ON-OFF as chegadas ocorrem segundo um processo de Poisson durante um período de tempo distribuído de forma exponencial, estando o período de silêncio também distribuído de forma exponencial. A versão de tempo discreto é conhecida como *Interrrupted Bernoulli Process* ou (IBP). Processos de chegada de uma única fonte de voz podem ser aproximados por um IPP. No caso de várias fontes de voz, o padrão de chegada de células pode ser aproximado através da superposição de N IPPs idênticos, resultando em um MMPP de $N + 1$ estados. Um resumo sobre a parametrização de processos modulados pode ser obtida em [COS94].

4.5.3 Modelos de tráfego auto-similar

Estudos recentes de medidas de tráfego com alta qualidade e alta resolução têm revelado um fenômeno que atinge diretamente a atividade de modelagem, projeto e controle de redes de banda larga. Para isso, foi necessário a análise de milhões de pacotes em uma LAN Ethernet em um ambiente de pesquisa e desenvolvimento [LEL93]. Especificamente, padrões de tráfego em rajadas gerados por fontes de dados e aplicações de tempo real VBR, tal como vídeo e áudio comprimido, tendem a apresentar um certo grau de correlação entre chegadas, mostrando um tempo de dependência de longa distância (*efeito Joseph*). Nestes estudos, o tráfego de pacotes parece ser estatisticamente auto-similar. O fenômeno auto-similar faz com que o tráfego medido exiba uma estrutura similar sobre várias escalas de tempo.

Em tráfego de pacotes, a auto-similaridade manifesta-se independentemente do tamanho das rajadas. A mesma manifesta-se de diferentes formas: uma função de autocorrelação cujo somatório não tende a um valor finito, a variância da média de amostragem que decresce em função do tamanho da amostra mais rapidamente que $1/n$, etc [FIL96]. O parâmetro Hurst (H) é utilizado para capturar o grau de auto-similaridade em uma seqüência. Um resumo sobre o estudo de auto-similaridade pode ser encontrado em [SAH99].

Hoje já existem várias propostas de modelos para geração de tráfego auto-similar. Alguns deles como ARIMA têm sido utilizados. Apesar deles não serem facilmente tratáveis, provêem uma boa descrição do tráfego auto-similar usando poucos parâmetros [ROB97]. Por outro lado, estão surgindo várias propostas baseadas em cadeias de Markov para geração de tráfego

auto-similar. Além da simplicidade, a utilização de cadeias de Markov permite o reuso das técnicas analíticas desenvolvidas no passado, como teoria de filas, para avaliação de desempenho. Alguns exemples destas propostas são apresentadas em [ROB96, ROB97, KAS01]. Como será visto no Capítulo 6, a fonte de tráfego proposta dá suporte a utilização modelos baseados em cadeias de Markov, com isso permitindo a geração de tráfego auto-similar. A mesma também permite a utilização de modelos ON-OFF de Pareto para geração de tráfego auto-similar. Estudos apresentados em [TAQ97, WIL97] *apude* [SIL00] mostram, analiticamente e através de medidas, que a superposição de várias fontes ON-OFF com estrita alternância de períodos ativos e inativos, onde estes períodos exibem o *efeito Noé* (alta variabilidade), podem gerar tráfego agregado de rede *efeito Joseph*.

O trabalho [SIL00] apresenta um resumo sobre as comparações realizadas envolvendo as distribuições exponencial e de Pareto. Nele diz-se que a distribuição de Pareto exibe o efeito Noé, e possui características bem distintas da distribuição exponencial, tradicionalmente usada nos modelos ON-OFF. Mostra-se também que a distribuição de Pareto tende a gerar valores muito pequenos e valores muito grandes com probabilidade muito maior que a distribuição exponencial. A tendência da distribuição exponencial é gerar mais valores em torno da média. Esta tendência torna os modelos ON-OFF de Pareto bem distintos dos modelos ON-OFF com distribuições exponenciais. Os resultados deste mesmo trabalho comprovam que a utilização do modelo ON-OFF exponencial pode ser usado para fins de avaliação de desempenho e projetos de multiplexadores ATM somente quando o comprimento da rajada é da ordem do tamanho do *buffer*. Para *buffers* maiores, os altores alegam que os resultados obtidos com esse modelo se distanciam muito da realidade.

Capítulo 5

Projeto e Implementação da Camada ATM

O trabalho de medição de desempenho em módulos de hardware ATM teve como motivação a avaliação de desempenho de um *driver* em desenvolvimento no escopo de uma cooperação com empresa. O *driver* deve receber pacotes de dados de uma rede Ethernet, e de voz de um processador DSP, e formatá-los adequadamente para serem transmitidos em uma rede ATM, através de um meio físico. Para isso, a estrutura mostrada na Figura 5.1 foi proposta.

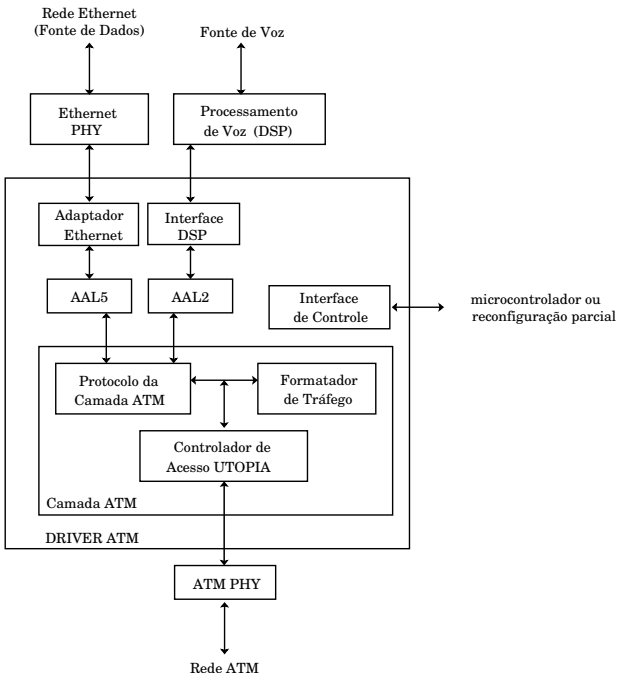


Figura 5.1: Diagrama de blocos do driver ATM.

Vários módulos que compõem o *driver* ATM foram implementados. O trabalho [SOU02] apresenta implementações em *software* e *hardware* para as diferentes camadas de adaptação ATM. Este Capítulo irá relatar a implementação da parte de recepçã de uma camada ATM, comentando a funcionalidade de cada módulo e suas interfaces. Esta implementação permite que se compreenda os requisitos e a necessidade de uma ferramenta de medição de desempenho de *hardware*. Uma ferramenta deste tipo, permite avaliar e validar um módulo de *hardware*

sem que seja necessário um conhecimento detalhado da estrutura e das funcionalidades do mesmo.

5.1 Implementação da Parte de Recepção ATM

A descrição da camada ATM de recepção é feita de forma *top-down*. O nível mais alto da hierarquia da estrutura é mostrado na Figura 5.2. Na Figura são mostrados um controlador UTOPIA, um controlador para uma memória CAM, uma ROM e o chamado núcleo da camada ATM, onde estão implementadas as funções de protocolo. A Figura também mostra os sinais que são utilizados para interação da camada ATM com as camadas de adaptação e com a camada PHY. A camada ATM de recepção implementada permite a utilização de 2^n pontos de acesso às camadas de adaptação através da parametrização do *soft core*, onde n pode variar de 0 a 8. Cada ponto de acesso permite a utilização de um barramento, o que possibilita compartilhar um mesmo ponto de acesso entre várias camadas de adaptação. Sendo n o número de bits utilizados para selecionar uma interface de acesso às AALs, 2^{8-n} é o número máximo de AALs em cada ponto de acesso.

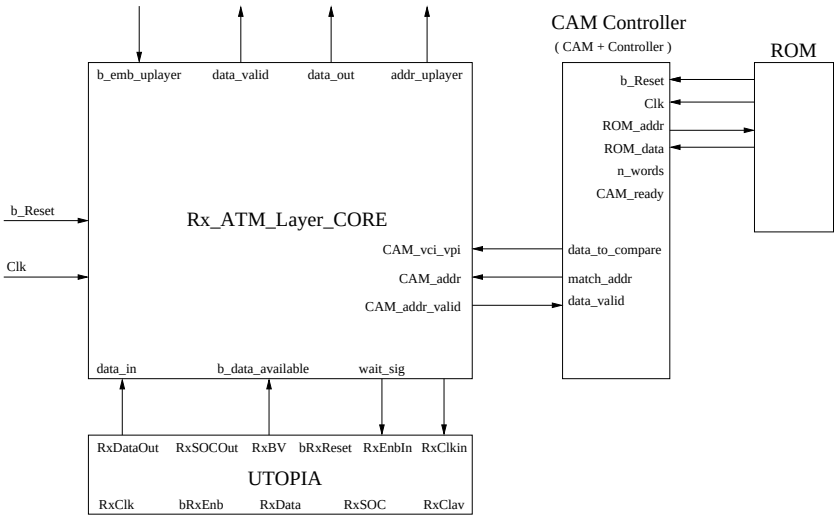


Figura 5.2: Diagrama de blocos do nível de hierarquia de mais alto da implementação da camada ATM de recepção.

5.1.1 O controlador da Interface UTOPIA

A camada ATM usa a interface padronizada UTOPIA para trocar dados com a camada PHY. Os sinais desta interface e o seu comportamento estão descritos em [COM95]. O controlador UTOPIA implementado deve ser utilizado apenas como parte integrante de uma camada ATM, devendo comunicar-se diretamente com uma camada PHY compatível com o padrão. Ele foi projetado para comportar-se como o mestre do barramento. Cabe salientar que o controlador implementado não é uma implementação completa da especificação, pois não possui, por exemplo, a capacidade de interagir com várias camadas físicas (MPHY).

A referência [SOU02] apresenta uma comparação entre os diferentes níveis da interface UTOPIA e a funcionalidade de cada sinal da interface de transmissão e recepção. Ainda, a

mesma referência apresenta de forma detalhada a implementação do controlador UTOPIA e a estratégia de teste utilizada. Assim, esta Seção irá assumir que a troca de sinais entre as camadas ATM e PHY está de acordo com a especificação UTOPIA nível 2 [COM95].

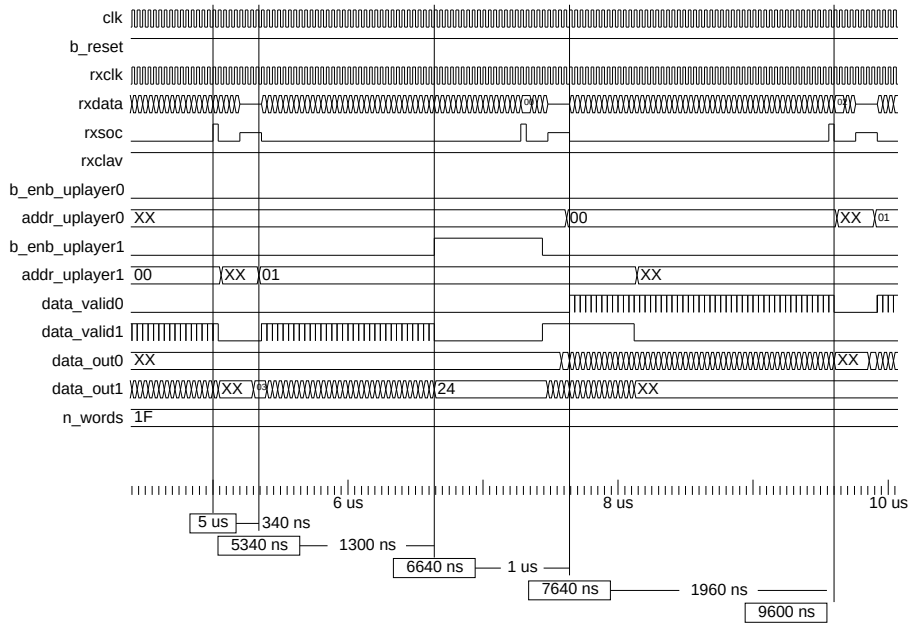


Figura 5.3: Sinais das interfaces da camada ATM. A Figura mostra a recepção de duas células consecutivas. O início de cada transmissão é marcada por um pulso no sinal `rxsoc` ($5us$). Quatro ciclos após, a camada causa uma parada na recepção para a decodificação do cabeçalho desativando o sinal `b_rxenb`. O *testbench* emula o comportamento de uma camada física multi-PHY, colocando o sinal de dados (rxdata) em *tri-state*. Durante a decodificação dos campos VPI/VCI da célula, a camada ATM identifica a qual ponto de acesso e a qual AAL pertence a conexão. Neste caso, a célula está destinada ao ponto de acesso 1. A camada ATM então seleciona a AAL (no endereço 01) no ponto de acesso através do sinal `addr_uplayer1` ($5340ns$) e começa a transmitir. O sinal `data_valid` que estava desativado durante a recepção do cabeçalho é ativado logo após o endereço da AAL ser colocado em `addr_uplayer`. O sinal `data_valid` é usado para indicar que dados válidos estão presente em `data_out`. Durante a recepção, a camada superior avisa à camada ATM que não será capaz de receber mais dados durante um determinado tempo mantendo `b_enb_uplayer1` ($6639ns$). Mesmo durante esta parada, a camada ATM continua recebendo os dados da camada PHY devido à presença de *buffers*. Em $7640ns$ uma nova célula começa a ser transmitida ao ponto de acesso 0 ao mesmo tempo que a célula anterior ainda está sendo transmitida ao ponto de acesso 1. Em $9600ns$ termina a recepção da célula do ponto de acesso 0 sem ocorrer nenhuma parada ($1960ns/40ns = 49$ ciclos de clock).

A Figura 5.3 mostra uma simulação funcional, ilustrando o comportamento dos sinais na interface da camada ATM para a seguinte configuração: interface UTOPIA com a camada física, 4 conexões e dois pontos de acesso a camada superior. A mesma também mostra a capacidade de independência entre os diferentes pontos de acesso configurados. Na Figura 5.3 os sinais estão divididos em quatro grupos. No primeiro estão apenas o sinais `clk` e `b_reset`. O prefixo "b_" é usado para avisar que o sinal é ativo em "0". No segundo grupo estão os sinais da interface UTOPIA [COM95] `rxclk`, `b_rxenb`, `rxdata` e `rxclav`. O terceiro grupo

é composto pelos sinais da interface com a camada superior `b_enb_uplayer`, `addr_uplayer`, `data_valid` e `data_out`. Como a configuração prevê dois pontos de acesso, existe um conjunto de sinais para cada interface, identificados como 0 e 1. No último grupo estão os sinais da interface como a memória ROM `n_words`, `rom_addr` e `rom_data`.

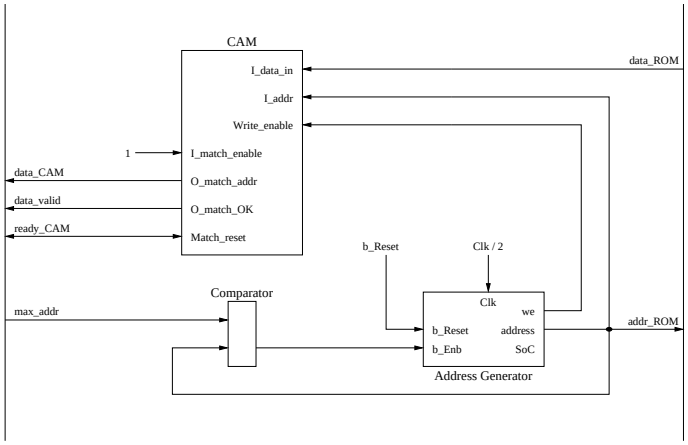


Figura 5.4: Diagrama de blocos do controlador CAM.

5.1.2 O Controlador da CAM

O primeiro nível da implementação, na Figura 5.2, mostra também uma interação entre o o controlador CAM e o núcleo da camada de recepção. Nesta implementação, a CAM é utilizada para resolver a questão de endereçamento das camadas superiores a partir dos valores de VPI e VCI. A memória CAM encontra-se implementada como parte do controlador CAM.

O controlador CAM é utilizado apenas para preencher a memória CAM com os valores que identificam as conexões virtuais, ou seja, o valor dos campos VCI e VPI da célula. Estes valores são copiados de uma ROM durante a inicialização do sistema. Esta implementação do controlador foi utilizada apenas para teste do sistema, contudo ela pode ser alterada para permitir a configuração dos valores de VCI e VPI a partir de diferentes dispositivos. A utilização de uma memória associativa de alto desempenho possibilita uma eficiente decodificação dos campos VCI e VPI do cabeçalho, embora a mesma exija uma grande quantidade memória para ser implementada. Este fato pode restringir significativamente o número de conexões, dependendo dos recursos de memória disponíveis. Por exemplo, para um dispositivo XCV800 é possível utilizar no máximo uma CAM de 3.584 *bits* com uma largura máxima de 16 x 224 bits ou com um número máximo de endereços 448 x 8 bits [BRE00]. Considerando que o tamanho do campo VPI é 8 *bits* e que toda a memória disponível no FPGA seria utilizada pela CAM, o número máximo de endereços em um dispositivo XCV800 é igual 149. A implementação foi baseada em um projeto de referência da Xilinx, descrito em [BRE00]. O diagrama de blocos do controlador é mostrado na Figura 5.4.

O funcionamento do controlador é simples. Para facilitar os testes do processo de configuração (escrita dos identificadores VCI e VPI na CAM), o número máximo de valores utilizados na inicialização da CAM é fornecido através de um sinal da interface (`max_addr`). O gerador de endereços, mostrado na Figura 5.4 como `address_generator`, fornece os endereços à ROM. Quando o endereço atinge o valor máximo, especificado através do sinal `max_addr`, o comparador ativa o sinal `finished`, tirando a lógica de comparação da memória CAM do estado

de *reset*. A Figura 5.5 mostra o resultado da simulação durante o período de inicialização da CAM.

Depois de copiados todos os identificadores de conexões para a CAM, o controlador avisa o núcleo, através do sinal `b_cam_ready`, que a CAM já pode ser utilizada. Este sinal funciona como um sinal de *set* para o núcleo da camada de recepção. Ou seja, quando o núcleo percebe o sinal `b_cam_ready` ativado, é dado início à execução das funções da camada ATM. A Figura 5.6 ilustra a troca de sinais entre o núcleo da camada ATM, o controlador UTOPIA e o controlador CAM na recepção de uma célula.

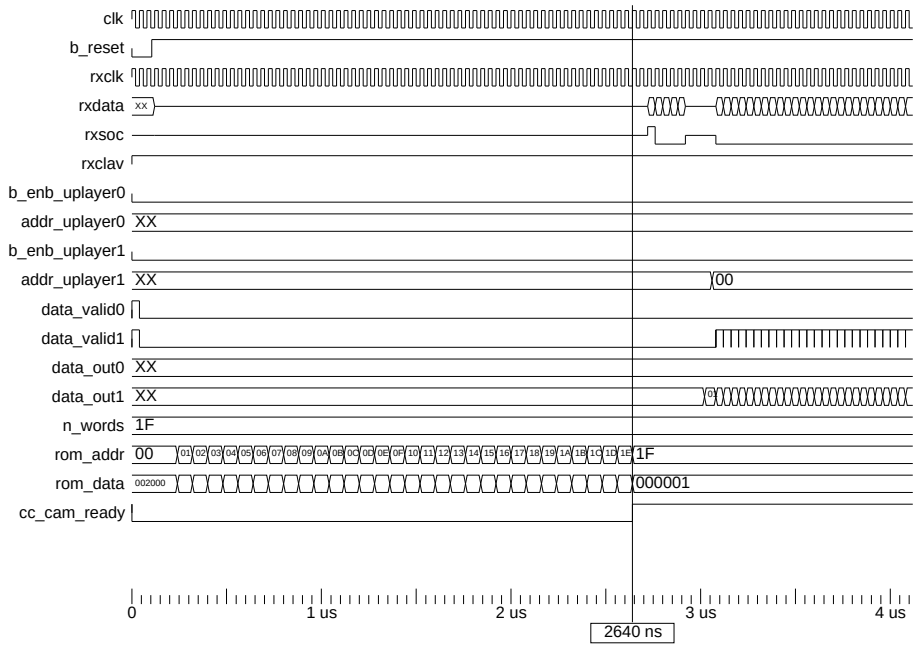


Figura 5.5: Inicialização da memória CAM. Durante os primeiros $2537ns$ o controlador utiliza os valores em uma memória ROM para preencher a memória CAM com os valores de configuração das conexões (VPI|VCI). Após o término da configuração, o controlador ativa o sinal `cc_cam_ready` fazendo com que o controlador UTOPIA e o núcleo da camada ATM saiam do estado de *reset*.

A Figura 5.6 ilustra uma importante funcionalidade da camada ATM, o descarte de células. A situação chamada de *inserção indevida de células* pode ocorrer nas chaves ATM devido a uma falha comutação, causada por um erro não detectado no cabeçalho. O descarte de células na camada ATM evita a retransmissão de pacotes pelas camadas superiores. A Figura 5.6 mostra duas situações diferentes de recepção. Quando a primeira célula chega à camada ATM, em $84345ns$, os valores de VPI e VCI são extraídos, concatenados e repassados à memória CAM através do sinal `core_vci_vpi`. Na memória CAM as conexões estão configuradas com os seguintes valores de VPI/VCI: 0/32, 16/32, 32/32, 48/32. O valor de `core_vci_vpi` (002020) em hexadecimal para a primeira célula corresponde ao valor 32/32 para VPI/VCI. O valor que retorna da CAM através do sinal `core_cam_addr` é utilizado para selecionar o ponto de acesso à camada superior e a AAL dentro de cada ponto. O valor 02 significa a seguinte configuração: ponto de acesso 0 e endereço de AAL 1. Este valor é interpretado como válido quando o sinal `core_cam_addr_valid` está ativo. Para a célula seguinte ($86640ns$), identificou-se que a conexão de valores $VPI = 64$ e $VCI = 34$ (002040) não havia sido previamente estabelecida. Neste caso, a célula é descartada. Nenhuma camada superior é

ativada através dos sinais `core_addr_uplayer` e `core_data_valid`

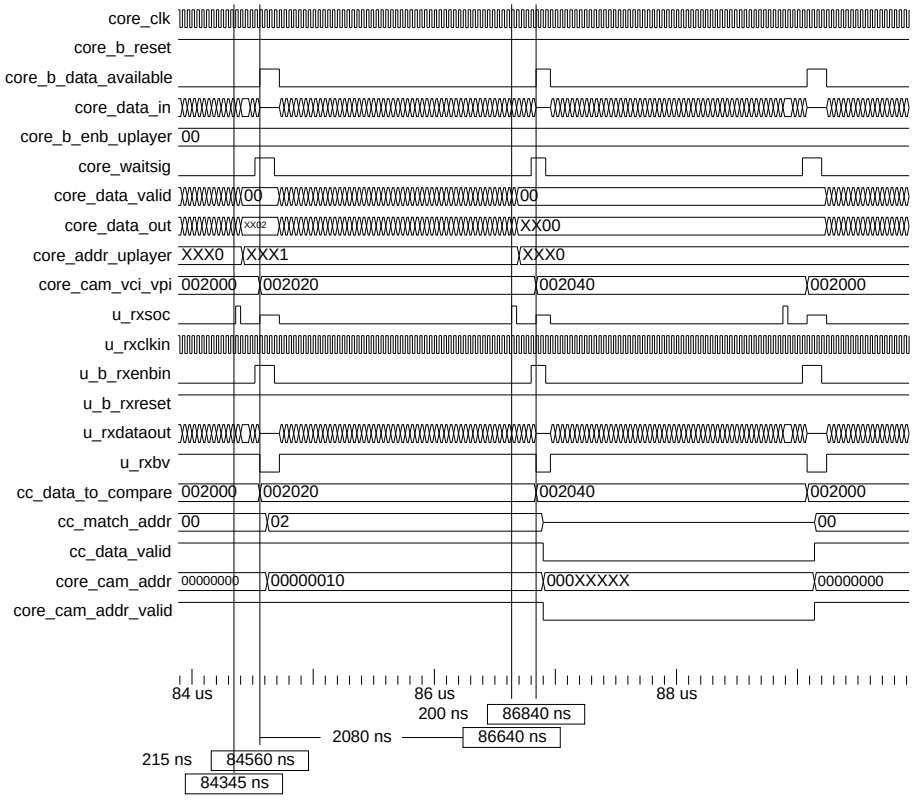


Figura 5.6: Simulação dos sinais do núcleo da camada ATM, do controlador UTOPIA e do controlador CAM.

5.1.3 Estrutura Interna do Núcleo

Em um segundo nível da hierarquia de projeto, o núcleo é constituído basicamente por dois módulos, chamados de *servidor* e *buffer de interface*. A Figura 5.7 ilustra a estrutura de interconexão entre estes dois módulos, identificados por *server* e *interface_ buffer*, respectivamente.

Somente o módulo servidor está ligado ao controlador UTOPIA, ou seja, toda as células que chegam na camada ATM são repassadas a ele. O servidor é responsável por separar os parâmetros da célula a serem passados à camada de adaptação e decodificar o cabeçalho das células para saber a que AAL as mesmas devem ser enviadas. Para saber isso é necessário saber a qual ponto de acesso a célula deve ser enviada e, aí então, qual AAL. Cada ponto de acesso tem relacionado a si um *buffer* circular, chamado *buffer de interface* ou `interface buffer` como identificado na Figura 5.7. Este é composto de uma memória dupla porta de tamanho parametrizável, adicionada de uma lógica de controle que permite a leitura correta da memória. Quando a lógica de controle percebe uma nova célula no *buffer*, ela a repassa automaticamente para a camada de adaptação destino. Nesta lógica de controle também está incluído um árbitro que rege o acesso à memória de dupla porta, evitando escritas em endereços que ainda não foram lidos ou leituras em endereços que ainda não forma escritos.

Cada um dos sinais nas interfaces dos módulos mostrados na Figura 5.7 será explicado mais adiante. Com a utilização do controlador UTOPIA, apenas três sinais são necessários para

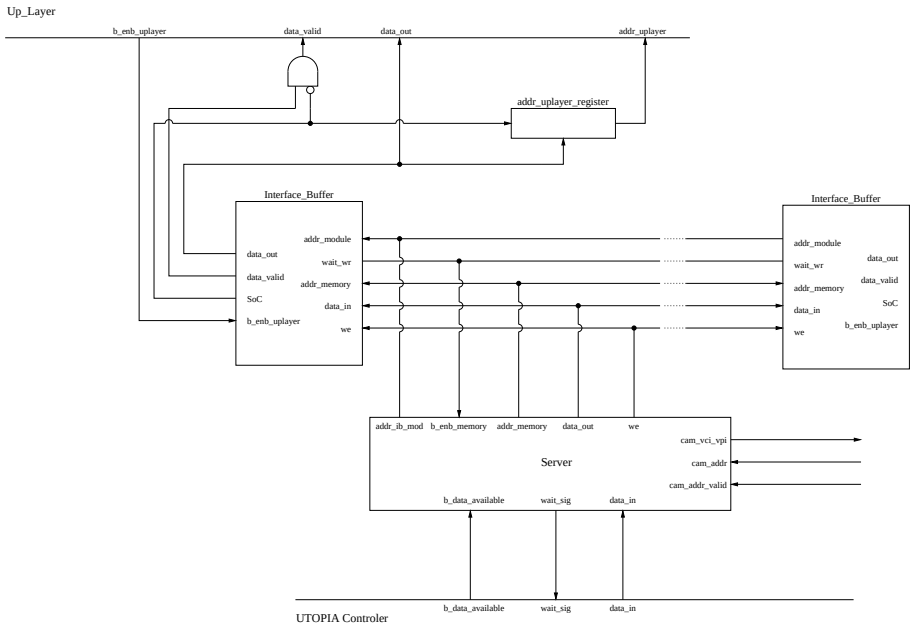


Figura 5.7: Diagrama de blocos da ligação entre o servidor e *buffer* de interface.

transferir os dados até o servidor. O servidor então trata o cabeçalho das células e identifica para qual dos pontos de acesso a célula deve ser enviada. Como o projeto do *driver* prevê apenas uma camada física, a utilização de um barramento para interligar o servidor a vários *buffers* de interface não prejudica o desempenho do sistema, já que não haverá disputa pela utilização do barramento. O primeiro *byte* enviado pelo servidor a um *buffer* de interface contém o endereço da camada superior, ou seja, de uma AAL. No *buffer* de interface, este byte é armazenado junto com os demais. Quando chegar a vez desta célula ser transmitida à camada superior, este primeiro byte é registrado pelo módulo identificado na Figura 5.7 como `uplayer_addr_register`. Este módulo trata de colocar o valor registrado nas linhas de endereços `addr_uplayer`. Durante o ciclo utilizado para registrar o valor do endereço da camada superior, a lógica ligada ao sinal `data_valid` desativa o mesmo, avisando à camada superior ativa que o dado durante este ciclo não é válido. A transmissão dos dados entre a camada ATM e AALs também se dá de forma síncrona. A Figura 5.8 permite observar, através de um resultado de simulação, a troca de sinais entre o servidor e os *buffers* de interface, entre o servidor e a interface com o controlador UTOPIA e entre os *buffers* e a interface com a camada superior. A mesma Figura mostra a chegada da primeira célula à camada ATM. Neste caso, tanto o *buffer* do ponto de acesso 0 como do ponto de acesso 1 estão vazios. A célula recebida tem como destino o ponto de acesso 1. Para facilitar a visualização os sinais do módulo `interface_buffer` do ponto de acesso 0 não estão sendo mostrados. Apenas o sinal `ib_data_valid0` está sendo mostrado para garantir que nenhum byte da célula em questão foi armazenada no *buffer* do ponto de acesso 0. Em $2719ns$ surge o início da célula. Do tratamento do cabeçalho, dois novos *bytes* aparecem em `s_data_out` no instante $3us$. O primeiro *byte*, 01, é o valor de retorno da CAM que será usado por uma lógica externa ao servidor para selecionar uma AAL no ponto de acesso 1. O segundo, 03, contém o valor dos parâmetros que devem ser repassados as AALs (campo PT da célula). No mesmo instante ($3us$), o servidor utiliza o sinal `s_addr_ib_mod` para selecionar o *buffer* de interface correspondente ao ponto de acesso desejado. Nota-se que apesar do endereço da AAL superior ter sido escrito no *buffer*

de interface, uma lógica externa garante que o mesmo não será interpretado como dado válido ($3080ns$).

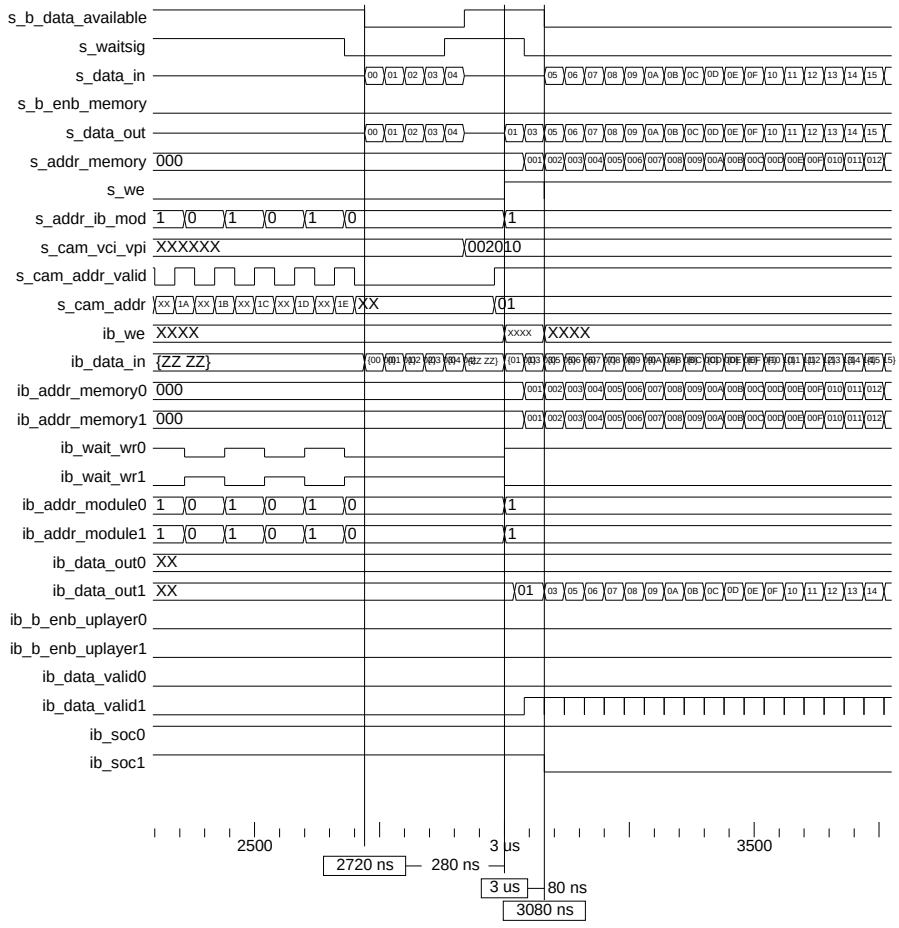


Figura 5.8: Simulação dos sinais entre o servidor e os *buffers* de interface.

As Figuras 5.9 e 5.11 ilustram os diagramas de blocos da lógica presente nos módulos `server` e `interface_buffer`, respectivamente. No servidor (Figura 5.9), a célula que chega através do controlador UTOPIA é recebida pelo *tratador de cabeçalho da célula*, mostrado na Figura 5.9 como `cell_head_manager`. Dois dos três sinais da interface do núcleo com a CAM são utilizados por este módulo. Após receber os campos VCI e VPI da célula, este módulo causa uma parada de apenas três ciclos na recepção para decodificar os identificadores da conexão. O resultado é retornado pelo sinal `cam_data`. Este sinal tem uma largura de 8 *bits*, sendo n bits utilizados para identificar o ponto de acesso à camada superior, e $8-n$ bits para identificar a AAL dentro do ponto de acesso.

Esta camada ATM de recepção também implementa a funcionalidade de descarte de células. Células que não pertencem a nenhuma conexão estabelecida por esta camada podem ser inseridas incorretamente no enlace por algum roteador devido a algum erro não detectado. Se estas células não forem descartadas na camada ATM, elas podem causar uma retransmissão nos níveis superiores. A funcionalidade de descarte de células inseridas incorretamente no enlace é implementada no tratador de cabeçalho da célula. Quando os valores de VCI/VPI de uma célula mal inserida são repassados à CAM, o sinal `cam_data_valid` avisa que nenhuma conexão foi estabelecida previamente com estes valores de VPI/VCI. Quando isto ocorre, o

tratador de cabeçalho de células desabilita todos os geradores de endereços, com isso desabilitando a escrita nos *buffers* de interface. De qualquer forma, este mantém o sinal `wait_sig` desativado, ou seja, a célula é recebida, mas não é repassada à camada superior.

O módulo tratador de cabeçalho da célula também retira os parâmetros que devem ser repassados à AAL. Estes parâmetros encontram-se no campo PT do cabeçalho da célula (4 *bits*). Quando retirados da célula, estes parâmetros são armazenados no *buffer* de interface como sendo o segundo *byte* da célula (o primeiro é o endereço da AAL).

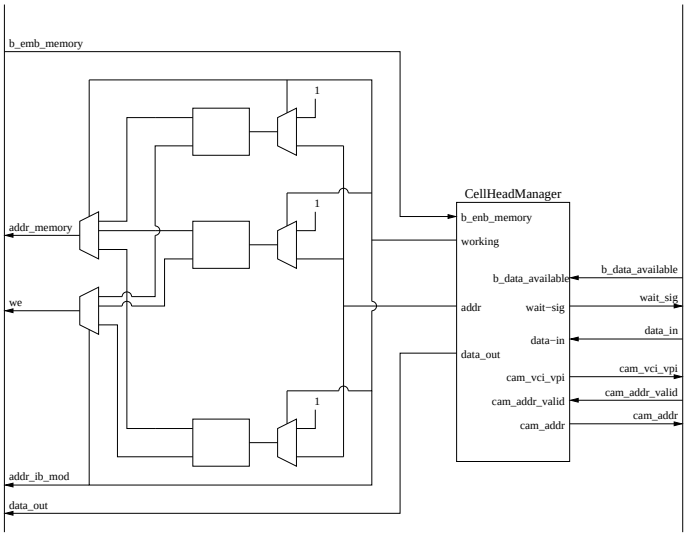


Figura 5.9: Diagrama de blocos do servidor.

A Figura 5.9 mostra que ligados ao `cell_head_manager` estão vários geradores de endereços. Cada um desses geradores está vinculado a um ponto de acesso, ou seja, a um *buffer* de interface. Os valores gerados por estes módulos são usados como endereços de escrita no *buffer* da interface correspondente. O tratador de cabeçalho da célula utiliza um barramento de endereços para ativar o módulo *address_ generator* correspondente à interface destino da célula. A saída desses geradores de endereços são multiplexadas utilizando *buffers tri-states*. Da mesma forma que o *buffer* de interface registra a saída de endereços, o tratador de cabeçalhos da célula registra o valor de retorno da CAM (sinal `cam_data`) e o coloca na saída de endereços (`addr`) para selecionar ao mesmo tempo um gerador de endereços e um *buffer* de interface (`addr_ib_mod`). Esses geradores recebem e geram sinais controle para a escrita no *buffer* durante uma transferência de dados síncrona. A Figura 5.9 mostra que o sinal que permite a escrita no *buffer* de interface selecionado (`b_enb_memory`) está ligado apenas ao tratador de cabeçalho da célula, e portanto é quem decide quando as células podem ou não serem escritas no *buffer* de interface. Assim os sinais de controle repassados aos geradores de endereço são baseados no estado do *buffer* e também no estado do tratador de cabeçalho da célula. A Figura 5.10 ilustra a troca de sinais internos ao módulo.

O diagrama de blocos do *buffer* de interface é mostrado na Figura 5.11. O módulo principal é chamado de *memória de transferência* (`transfer_memory`). Neste módulo, está implementado o *buffer* circular e seu árbiro. Como dito anteriormente, este árbitro é utilizado para evitar que escritas sejam feitas em endereços ainda não lidos e leituras sejam feitas em endereços que não forma escritos. O acesso a este *buffer* é feito de forma seqüencial, o que facilita a implementação do árbitro. Todo o controle é feito a nível de endereços e não de células. Isto permite que a célula comece a ser lida antes mesmo de estar totalmente gravada no *buffer*.

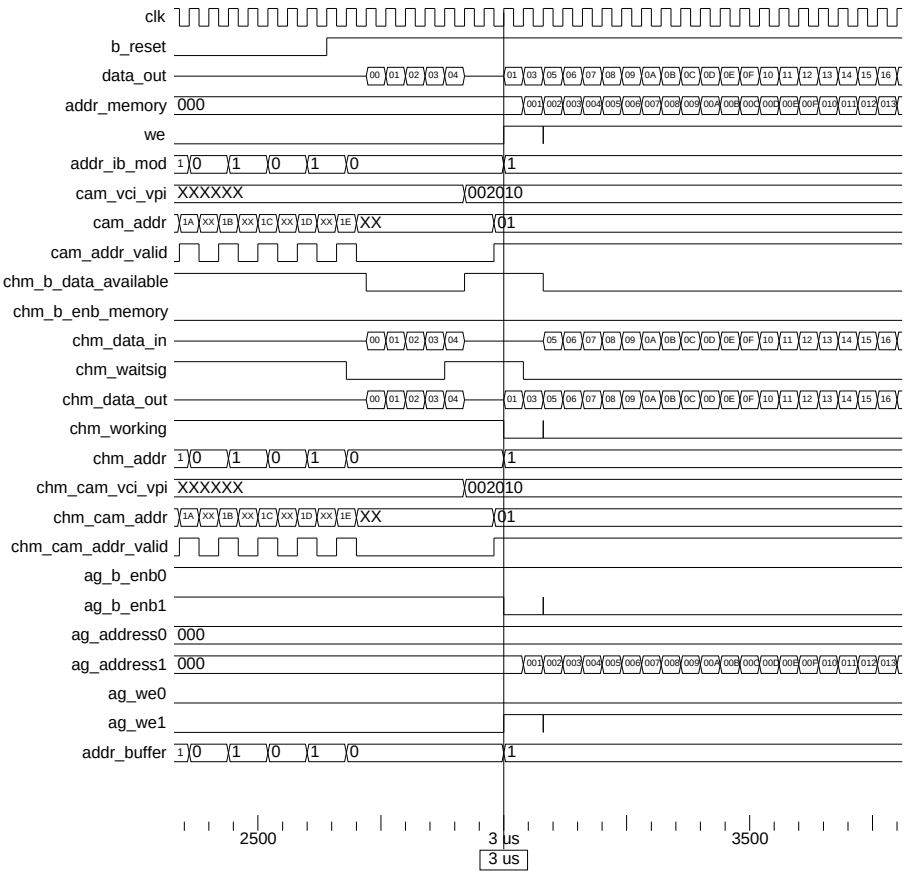


Figura 5.10: Simulação dos sinais internos ao servidor. O sinais com o prefixo "chm" pertencem ao módulo `cell_head_manager`. O tratador de cabeçalho de célula utiliza o sinal `addr_buffer` para selecionar um *buffer* de interface e um gerador de endereços ($3us$). Os geradores de endereços são identificados na forma de onda pelo prefixo "ag".

A memória de transferência foi implementada de forma a facilitar os testes em busca de um tamanho de *buffer* adequado a uma combinação de determinada carga de tráfego e taxa de serviço por parte das camadas de adaptação. O *buffer* circular presente neste módulo pode ser dimensionado simplesmente parametrizando o *soft core*. Este *buffer* apresenta uma ótima escalabilidade, podendo ser dimensionado a nível de células e com uma resolução de uma célula. Através da parametrização do *soft core* ainda é possível configurar o número de *bytes* por célula a serem gravados no *buffer* de interface, atualmente são 50 (48 *bytes* de carga útil da célula, 1 *byte* de endereço da camada superior e 1 *byte* com os parâmetros da AAL). Esta facilidade permite uma inclusão de novos parâmetros de maneira simples, ou até mesmo uma expansão na largura dos dados.

A lógica mostrada à direita da memória de transferência na Figura 5.11 é responsável por manter o sinal `address_wr` com o endereço da última escrita durante o período em que o módulo não está sendo utilizado pelo servidor. Isto é necessário para garantir uma leitura correta do *buffer*. A leitura é feita pela lógica mostrada à esquerda da memória de transferência na Figura 5.11. Esta lógica é composta basicamente por um gerador de endereços, idêntico aos utilizados no servidor e no contador da CAM. O sinal de controle que indica ao gerador de endereços que um novo dado pode ser transferido da memória para a camada superior é basicamente um "ou" lógico entre o sinal `b_enb_uplayer` da AAL ativa e o sinal `wait_rd` da

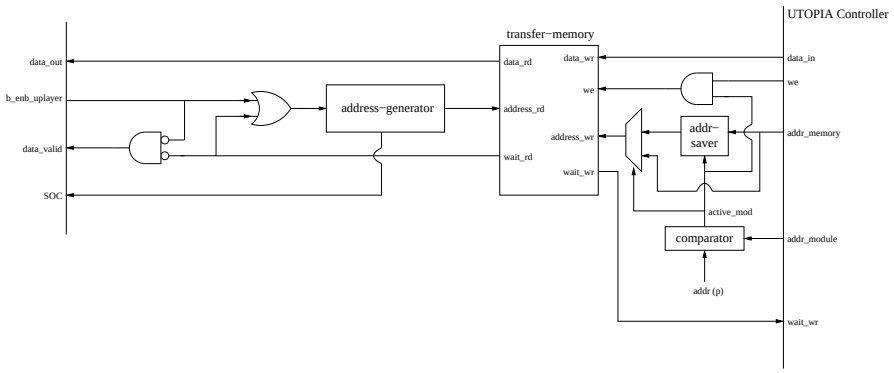


Figura 5.11: Diagrama de blocos do *buffer* de interface.

memória de transferência. O sinal `b_enb_uplayer` indica que a AAL ativa é capaz de receber mais uma palavra de dados, que neste caso é de 8 *bits*. Já o sinal *wait_rd* é provido pelo árbitro da memória e quando ativo indica que a memória não pode ser lida durante o atual ciclo de *clock*. A Figura 5.12 mostra a troca de sinais internos de um *buffer* de interface durante um período ativo e inativo, ou seja, quando a célula está sendo enviada ao seu ou a outro ponto de acesso.

5.2 Conclusões

Naturalmente existe uma série de detalhes de implementação que não foram discutidos durante este Capítulo, mas que não contribuem significativamente para o entendimento do sistema implementado. A seguir são resumidas as principais características da camada ATM de recepção desenvolvida neste projeto.

1. O sistema implementado tem um boa escalabilidade devido ao uso de um *soft core*, o cuidado dado a sua modularização e o nível de parametrização de cada módulo envolvido.

2. A parametrização do *soft core* permite, entre outras coisas, especificar o número de pontos de acesso à camada superior e o tamanho do *buffer* de interface em cada ponto de acesso.

3. Também é possível, através da simples parametrização do *soft core*, aumentar o número de parâmetros a serem gravados nos *buffers* de interface. O *buffer* está implementado no módulo chamado de *memória de transferência*. Este é composto basicamente de uma memória *dual port* e um árbitro que controla o acesso a ela. O tamanho da memória é parametrizável no nível de célula. Uma alteração deste módulo para aumentar o número de palavras para cada célula a ser gravada no *buffer* poderia consumir um tempo significativo. Prover um bom nível de parametrização deste módulo, permite diminuir o custo envolvido na alteração funcional dos demais módulos que compõem o sistema.

4. Sem considerar as restrições de tempo do circuito, esta camada ATM de recepção pode atingir uma taxa de 190Mbps utilizando uma interface de 8 *bits* a 25MHz. Um dos principais gargalos com relação à taxa de transferência é o tempo gasto para decodificação dos campos VPI e VCI. Para minimizar este custo utilizou-se uma CAM de alto desempenho.

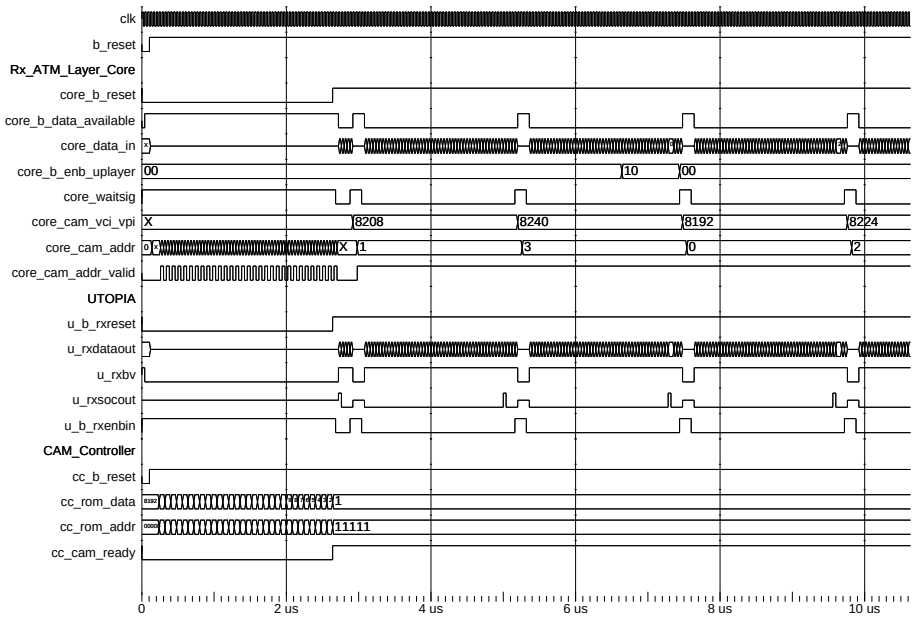


Figura 5.12: Simulação do uso dos *buffers* de interface. Os *buffers* de interface estão separados pelos seus respectivos sinais `clk`. O primeiro grupo de sinais está relacionado com o ponto de acesso 0 e o segundo grupo como o ponto de acesso 1. O comportamento dos sinais mostram que os *buffers* de interfaces estão ligados ao servidor por um barramento. Os sinais ligados ao servidor de ambos *buffer* de interface tem os mesmos valores. Desta forma, o *buffer* de interface é selecionado pelo endereço presente no sinal `addr_module`. No instante $3us$ o *buffer* de interface do ponto de acesso 1 é selecionado e duas células são escritas em seqüência no *buffer*. No instante $7560ns$ o mesmo acontece para o ponto de acesso 0. Em seguida, no instante $11840ns$, não ocorre a chegada de célula ou o servidor detectou uma célula mau inserida no enlace e a está descartando.

5. O projeto permite até 256 conexões simultaneamente ativas independentemente do número de pontos de acesso à camada superior. Contudo, na prática, a utilização da CAM de alto desempenho, na maioria dos casos, limita o número de conexões devido à grande quantidade de memória exigida.

6. A implementação não possui uma largura de dados parametrizável.

7. A parametrização do número de palavras para cada célula nos *buffers* e a forma com que o árbitro destes *buffers* foram implementados permite um uso efetivo da memória de dupla porta de 78%. Atualmente, são gastos 64 endereços para armazenar 50 palavras.

Capítulo 6

Ferramental Proposto

É notável a crescente utilização da combinação de FPGAs com linguagens de descrição de *hardware* tais como VHDL no projeto de diferentes tipos de sistemas. A utilização de ferramentas de simulação HDL e de síntese tem diminuído sensivelmente o tempo de implementação desses projetos. Neste Capítulo, apresenta-se uma proposta de ferramental que visa a utilização de FPGAs e HDLs para adquirir uma série de vantagens no processo de avaliação de desempenho. Algumas destas vantagens são:

1. Permitir uma fácil distribuição da ferramenta, já que o *hardware* que a compõe constitui-se de um ou mais *soft core*;

2. Minimizar o problema da falta de flexibilidade das fontes de tráfego. Sendo o *hardware* um *soft core* é possível redimensioná-lo de acordo com as necessidades, considerando os recursos disponíveis no FPGA.

3. Possibilitar o uso de recursos disponíveis em simuladores HDL para a simulação global do sistema, caso o *hardware* ATM também esteja descrito em HDL;

4. Permitir a avaliação de módulos com altas taxas de transmissão se todo o sistema estiver *on chip*. Neste caso, o sistema é composto dos módulos de medição e do HA.

5. Possuir uma boa escalabilidade, já que até seu *hardware* pode ser alterado facilmente;

6. Diminuir sensivelmente o tempo de desenvolvimento de *test benches*.

A proposta do ferramental tem como desafio explorar as vantagens oferecidas pelo uso de FPGA e *soft cores* na avaliação de desempenho de módulos ATM. O propósito do ferramental é gerar informações sobre o potencial de degradação da qualidade de serviço através da medição dos parâmetros de QoS. A princípio, os parâmetros de QoS que serão medidos são apenas os negociáveis com a rede.

O ferramental proposto é executa basicamente duas atividades: a *geração de estímulos* e a *coleta de informações do HA*. A geração de estímulos tem como objetivo fornecer ao HA diferentes padrões de chegada de células. Este processo é descrito na Seção 6.1. A coleta de informações, por sua vez, realizará a retirada das informações do HA úteis ao processo de avaliação. Ao final, é possível medir o potencial de degradação da qualidade de serviço do HA através dos valores calculados para os parâmetros de QoS. A coleta de informações é discutida na Seção 6.2. A Figura 6.1 ilustra os processos executados pela ferramenta.

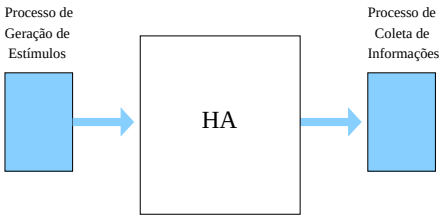


Figura 6.1: Processos executados pela ferramenta proposta.

Dada as dificuldades enfrentadas pelas ferramentas descritas na Seção 4, o primeiro fator considerado no projeto da ferramenta foi o nível de flexibilidade que poderia ser dado ao processo de geração de estímulos, considerando os recursos disponíveis em um FPGA atual. O ferramental proposto é composto de um estrutura de *software* e *hardware* que torna a sua utilização viável e a mesmo tempo torna transparente ao usuário a configuração do *hardware* para geração do padrão de tráfego desejado. Por fim, a Seção 6.3 descreve a implementação da fonte de tráfego proposta na Seção 6.1.

6.1 Processo de geração de estímulos

A geração de estímulos é o principal processo do ferramental. É ela que torna ou não significativas as informações coletadas do HA. Estas podem ser consideradas incorretas, se os estímulos gerados não conseguem reproduzir com fidelidade o padrão de tráfego desejado pelo usuário. Além disso, a quantidade de padrões de tráfego que o processo de geração de estímulos é capaz de reproduzir determina a versatilidade da ferramenta. Para se alcançar os requisitos de desempenho e versatilidade exigidos, o processo de geração de estímulos é composto de módulos de *software* e *hardware*.

Notou-se que apesar da proposta existente em [KEI95] mostrar algumas dificuldades para representar determinados tipos de tráfego, esta permite a utilização de diversos processos de distribuição com um único suporte de *hardware*. Com base nesta idéia, foi feito um novo projeto de *hardware* para a geração de estímulos. Este *hardware* será chamado de fonte de tráfego.

Dentro do processo de geração de estímulos, o projeto do *hardware* (fonte de tráfego) é que determina quão flexível poderá ser o processo. Duas soluções são possíveis. A primeira é criar um projeto de fonte para cada padrão de tráfego. Contudo, esta proposta apresenta uma série de problemas, onde os principais são a área do FPGA gasta com fontes de tráfego inativas ou um alto custo de compilação. O custo de compilação é atribuído a um novo processo de síntese do *hardware* para incluir uma nova fonte sempre que houver a necessidade da geração de um novo padrão de tráfego. A segunda é utilizar uma única estrutura de *hardware* capaz de dar suporte à geração de diferentes padrões de tráfego. Neste caso, o custo de compilação é determinado pelo tempo de reconfiguração da fonte, que é bem menor que o tempo de síntese. Uma estrutura em *software* foi projetada para executar parte do processo de tradução do modelo do tráfego para os parâmetros de configuração da fonte. A Figura 6.2 ilustra a estrutura composta de *software* e *hardware*, que foi proposta para a execução do processo de geração de estímulos.

Dois módulos compõem a parte de *software* do processo de geração de estímulos. Eles serão chamados de *interpretador* e *configurador*. O interpretador é responsável por converter um

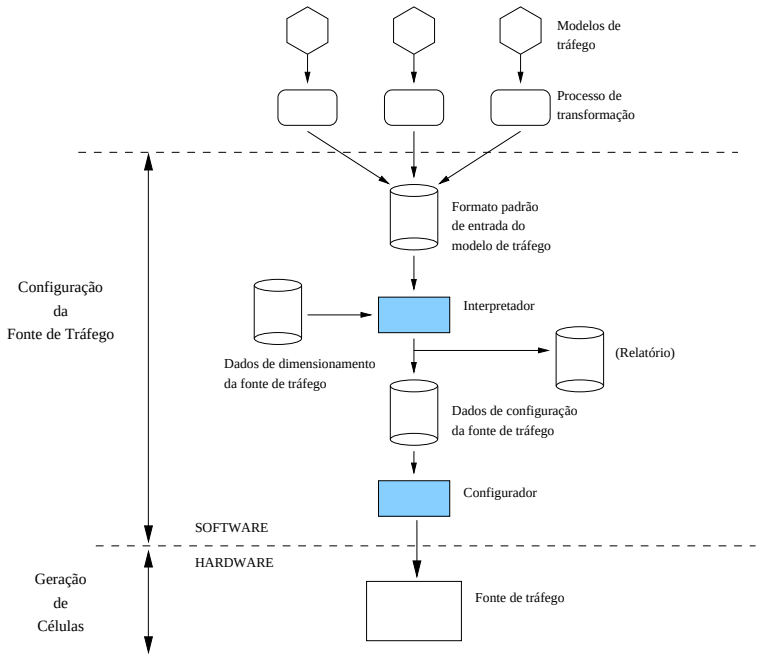


Figura 6.2: Estrutura de *software* e *hardware* proposta para geração de estímulos ao HA.

modelo específico de entrada em parâmetros de configuração da fonte de tráfego. O modelo de entrada é uma máquina de estados que permite especificação da distribuição utilizada para o tempo de permanência em cada estado e para o intervalo entre células. Após a leitura da máquina de estados, o interpretador deve, a partir dos parâmetros da distribuição da variável aleatória escolhida pelo usuário, calcular a probabilidade para um intervalo de possíveis valores da mesma. Este intervalo deve respeitar as dimensões das memórias de configuração da fonte de tráfego. A próxima tarefa do programa é converter estas probabilidades em parâmetros de configuração da fonte. A fonte de tráfego utiliza um conjunto de memórias e um procedimento de acesso a elas para gerar o padrão de tráfego segundo o processo de distribuição escolhido pelo usuário. Esta técnica é apresentada em [LAW91]. Diferente da proposta em [KEI95], esta permite uma melhor utilização da memória com um aumento insignificante de custo para o caso em questão. Neste caso, o custo é o tempo necessário para decidir o valor da variável aleatória. Por exemplo, para uma variável aleatória com n possíveis valores, são necessárias apenas duas memórias de n endereços, F_i e L_i. A largura de F_i dependerá da resolução da probabilidade desejada e a de L_i dependerá do tamanho do intervalo de valores que a variável poderá assumir. Kronmal e Peterson em 1979 propuseram um algoritmo eficiente para o preenchimento destas memórias [KEL91a]. O algoritmo de Kronmal e Peterson é apresentado abaixo.

1. Fazer $F_i = (n+1)p(i)$ para $i = 0, 1, ..., n$, onde i é o intervalo de valores assumidos pela variável aleatória.

2. Definir um conjunto G e S, onde $G = i : F \leq 1$ e $S = i : F_i \leq 1$.

3. Executar os seguintes passos enquanto S não estiver vazio:

   (a) Remover um elemento de G e outro de S, k e m respectivamente.

   (b) Fazer $L_m = k$ e trocar F_k por $F_k - 1 + /F_m$.

(c) Se $F_k < 1$, colocar k em S; caso contrário, colocar k em G

Esta técnica utiliza um acesso a cada memória para determinar o valor da variável aleatória. Assumindo que a variável aleatória possa ter um valor entre 0 e n, um valor j entre 0 e n é sorteado. O valor contido na posição j de F_i é comparado com um outro valor sorteado, que será chamado de p. Se o valor contido na posição j de F_i é menor que p, a variável aleatória passa a ter o valor j, caso contrário ela recebe o valor da posição j de L_i.

O segundo programa, o configurador, é responsável por configurar a fonte de tráfego a partir dos valores de F_i e L_i gerados pelo interpretador. A princípio, este programa pode utilizar qualquer interface com a placa de prototipação já que nesta parte do processo de geração de estímulos não há restrições temporais. No FPGA deve haver um módulo de *hardware* associado a cada interface da placa utilizada para configurar a fonte. Esta interface é responsável por receber o fluxo de dados gerado pelo configurador e, com isso, configurar a fonte de tráfego. Neste caso, configurar a fonte de tráfego significa preencher as RAMs da mesma de forma adequada. Assim, quando houver um redimensionamento da fonte, ou seja, de suas memórias RAMs, também será necessária uma nova parametrização dos módulos de interface. Uma outra proposta é a utilização de reconfiguração parcial para preencher a memória de configuração da fonte de tráfego. Apesar do processo de reconfiguração parcial depender da arquitetura do FPGA, sua utilização evita um acréscimo de hardware junto à fonte de tráfego.

O processo de geração de estímulos descritos até agora não exige nenhuma restrição de tempo rigorosa. Por isso, para diminuir o esforço de desenvolvimento e tornar a ferramenta mais flexível é possível que este processo seja executado em *software*. Contudo, a geração das células deve ser executada em *hardware* devido à largura de banda exigida pelos módulos ATM. A fonte de tráfego é composta basicamente por dois módulos: o gerador de células e seu modulador. O primeiro, como o próprio nome diz, tem o trabalho de gerar as células, utilizando um processo de distribuição para controlar o espaçamento entre as mesmas. O segundo é utilizado para modular o gerador de células, criando várias instâncias do mesmo. A máquina de estados traduzida pelo interpretador é utilizada para criar estas instâncias. Cada instância pode ter um padrão de geração de células com diferentes valores de cabeçalho para as células geradas. A estrutura da fonte permite uma tradução intuitiva de processos como o GMPP (*Generally Modulated Poisson Process*) ou GMDP (*Generally Modulated Deterministic Process*) para os seus parâmetros de configuração. A estrutura da fonte é apresentada na Figura 6.3.

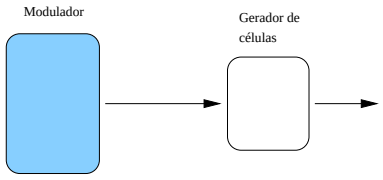


Figura 6.3: Estrutura da fonte de tráfego composta pelo gerador de células e seu modulador.

Ainda, segundo [KAS01a] é possível criar um tráfego pseudo auto-similar sobrepondo várias cadeias de Markov de dois estados com diferentes escalas de tempo. O autor utiliza o termo pseudo auto-similar para referir-se ao tráfego obtido com o seu modelo. Com exceção da cadeia de Markov de menor escala de tempo, as demais não teriam uma taxa de geração de células associado a cada estado, mas sim, uma outra cadeia de Markov. O Capítulo 7 deste trabalho

mostra um exemplo em que a fonte é configurada a partir de um modelo ON-OFF modulado por Pareto (PMPP).

Cada célula gerada pela fonte de tráfego carrega consigo informações que serão usadas para calcular os valores dos parâmetros de QoS. Estas informações são transportadas no campo de carga útil da célula. Na implementação atual, as informações utilizadas são o número de seqüência da célula, o identificador do estado do modulador, o valor de um temporizador que indica o instante em que a célula foi gerada e um código de correção de erros para o campo de carga útil da célula. A informação do número de seqüência da célula e o código de erros junto com os campos HEC e o CLP da célula permitem a detecção de perda de células, enquanto o valor do temporizador permite a medição do atraso de cada célula. O identificador do estado permite identificar em que estado estava o modulador quando foi causada a perda ou o atraso da célula.

O gerador de células também permite que os valores do cabeçalho sejam alterados. Com isso, cada estado do modulador pode gerar células com diferentes valores de cabeçalho. Esta característica permite a emulação de várias conexões em um único enlace.

Todo o processo de configuração da fonte e as funcionalidades da mesma foram implementados. A Seção 6.3 apresenta um projeto do *hardware* para a fonte de tráfego e uma estrutura composta de programas para viabilizar o processo de teste da fonte.

6.2 Processo de coleta de informações

O processo de coleta é realizado basicamente por módulos de *hardware*. Estes módulos têm o trabalho de retirar as informações da célula úteis ao processo da avaliação para que estas sejam posteriormente analisadas. Estas informações devem permitir o cálculo de número de células perdidas, o atraso constante e a variação do atraso para cada célula. Módulos de *software* são utilizados apenas para retirar as informações do FPGA e analisá-las.

O módulo chamado de *coletor* está em contato físico com as linhas de dados e de controle do HA. Ele é o responsável por controlar a retirada das informações do HA. O coletor utiliza uma interface síncrona para facilitar o alcance de altas taxas de coleta. A interface é composta de apenas dois sinais de controle. O primeiro é utilizado para avisar o início de uma nova célula. O outro sinal avisa ao coletor que dados válidos estão presente nas linhas de dados. A largura das linhas de dados depende da largura dados que está sendo utilizada pelo HA. Um exemplo de ligação entre os sinais do coletor e o HA é mostrado na Figura 6.4.

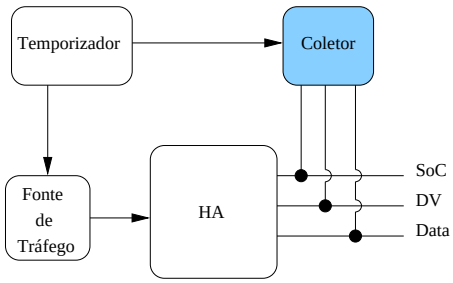


Figura 6.4: Ligação dos sinais de dados e controle entre o coletor e o HA.

Assim como a fonte de tráfego, o coletor também está ligado a um temporizador, como mostrado na Figura 6.4. Quando a célula é capturada pelo coletor, o mesmo deve consultar o

temporizador para registrar em que instante a chegada ocorreu. Subtraindo o valor consultado do valor de tempo de partida da fonte contido na célula é possível saber o atraso ocorrido entre a fonte e o ponto onde está colocado o coletor. Então, o coletor não apenas retira os dados úteis das linhas de dados do HA, mas também os processa para obter resultados iniciais. Isto também ocorre para os dados relacionados a detecção de perda de células. Na célula, o número de seqüência de células, o código de detecção de erros do campo de carga útil e o campo HEC do cabeçalho são utilizados na identificação de células perdidas. Quando estas informações são retiradas da célula pelo coletor, o mesmo as utiliza para gerar as seguintes informações relevantes ao processo de avaliação: o número de seqüência de células, um valor verdadeiro ou falso para a integridade dos campos da célula e a diferença entre o tempo de partida da célula e o tempo de chegada no ponto onde está o coletor. A Figura 6.5 ilustra os dados de entrada e de saída do coletor.

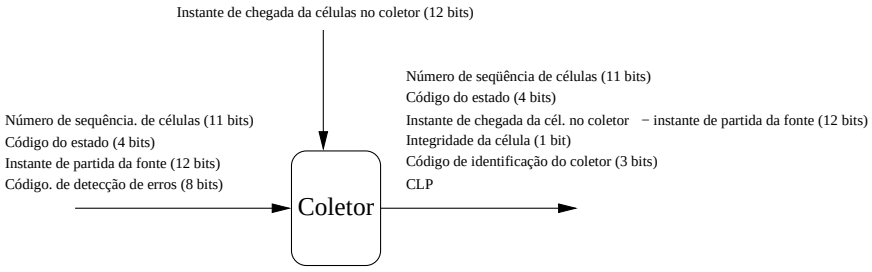


Figura 6.5: Produto gerado pelo coletor a partir dos dados extraídos da célula e do temporizador.

As informações produzidas pelo coletor devem ser retiradas do FPGA para que possam ser avaliadas pelos módulos de *software* em um *host*, por exemplo um computador pessoal do tipo PC. As operações realizadas pelo coletor diminuem o tempo de processamento do *software* e a taxa de transferência entre o FPGA e o PC. O tempo de processamento do *software* é reduzido, já que parte das operações estão sendo feitas em *hardware*. A redução da taxa de transferência pode ser vista comparando, na Figura 6.5, o número de *bits* de dados de entrada do coletor e seus dados de saída.

Duas formas de retirada de informações foram investigadas. A primeira utiliza uma memória para armazenar os dados. Nesta proposta, os dados produzidos pelos coletores durante o processo de execução do HA serão armazenados em uma memória até que esta atinja a sua capacidade máxima. Neste instante, dá-se fim ao processo de coleta e a amostra é retirada da memória para ser analisada no *host*. O grande problema apresentado por esta proposta é o tamanho da amostra. Por exemplo, se uma memória de $512K$ x $32bits$ fosse utilizada para armazenar os dados produzidos por um coletor trabalhando a 50MHz com uma interface com o HA de $16bits$, resultaria em aproximadamente $0,26s$ de execução do HA. Este é o pior caso, já que dificilmente o intervalo entre células se mantém igual a um. Já a segunda proposta visa a retirada dos dados produzidos pelo coletor durante o tempo de execução do HA. Esta proposta permite uma amostra de tamanho maior que a anterior. A princípio, a interface Ethernet da placa de prototipação é usada por prover uma largura de banda suficiente para a retirada dos dados do coletor. Contudo, a viabilidade da aplicação desta proposta ainda está sendo investigada.

Se for desejada a utilização de mais de um coletor, em ambas propostas, será necessária a utilização de *buffers* dentro de cada coletor para que não ocorra perda de informações devido à disputa pelo recurso, seja ela a interface Ethernet ou a memória. Neste caso, deve ser utilizado

um árbitro para gerenciar o acesso ao recurso. A Figura 6.6 mostra os diferentes módulos de *hardware* envolvidos no processo de coleta de dados. O número máximo de coletores e o tamanho dos *buffers* dos mesmos devem ser estabelecidos em função da taxa máxima de acesso ao recurso (Ethernet ou memória) e do padrão de tráfego gerado pela fonte. Considerando a complexidade da resolução do problema, está sendo utilizado um esquema simples para detectar perda de informação dentro dos coletores.

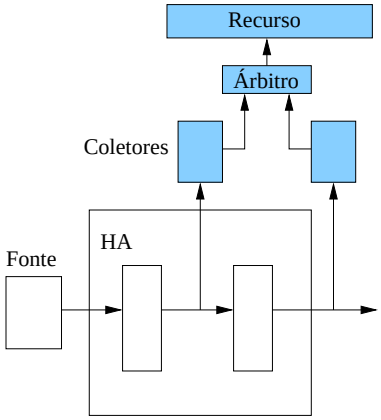


Figura 6.6: Estrutura geral dos módulos de *hardware* do processo de coleta.

No *host*, um software é responsável por receber as informações extraídas do *hardware* e gravá-las em disco para que posteriormente possam ser avaliadas. A princípio, com os dados produzidos pelos coletores podem ser medidos a taxa de perda de célula, o atraso constante e a variação no atraso de células.

A perda de células é medida através do parâmetro CLR. Como mostrado na Seção 2.3, a perda de células inclui o número de células que não chegaram ao destino, o número de células com cabeçalho inválido e o número de células nas quais o conteúdo foi corrompido por erros. A detecção de células que não chegaram ao destino é feita simplesmente observando o número de seqüência de células que foi incluído no campo de carga útil da mesma. O método usado permite a detecção da perda de no máximo 2^{11} células consecutivas. Isto ocorre devido ao número de bits utilizado na representação do número de seqüência de células.

Como dito anteriormente, o coletor gera um código de detecção de erros sobre os campos do cabeçalho e compara este código com o valor do campo HEC do cabeçalho da célula. Da mesma forma, o coletor gera um código de detecção de erros sobre o campo de carga útil é o compara com o valor do último byte da célula que também é um código de detecção de erros do campo de carga útil, só que gerado pela fonte. Desta forma é possível identificar todos os fatores envolvidos na medição da taxa de perda de células. O trabalho do *software* é apenas contabilizar a taxa de perda de células baseando-se nestes fatores e no campo CLP da célula. O campo CLP torna-se inútil no uso de fluxo não agregado, como descrito na Seção 2.3.1.

As medidas de atraso são feitas utilizando um valor de tempo relativo produzido pelo coletor. Este tempo é resultado da subtração entre o tempo de chegada da célula no coletor e o tempo em que a célula foi disparada pela fonte. Este método é conhecido como *two-point CDV* e está descrito na Seção 2.3.2.2. O trabalho do *software* é calcular o atraso entre os diferentes pontos de coleta e descobrir o maior valor da variação do atraso.

6.3 Implementação do processo de geração de estímulos

Como apresentado anteriormente, o processo de geração de estímulo permite a parametrização de um conjunto de valores de forma que o fluxo de células gerado pela fonte de tráfego seja capaz de representar o comportamento estatístico desejado pelo usuário. A fonte de tráfego gera células segundo um processo modulado o que permite representação de dependências de curta distância. O modelo utilizado como entrada para o processo de geração de estímulos permite especificar a probabilidade de transição entre os estados, a distribuição de probabilidade para o tempo de permanência e a distribuição para o intervalo entre células para cada estado. A Figura 6.7 ilustra os valores que podem ser parametrizados pelo usuário.

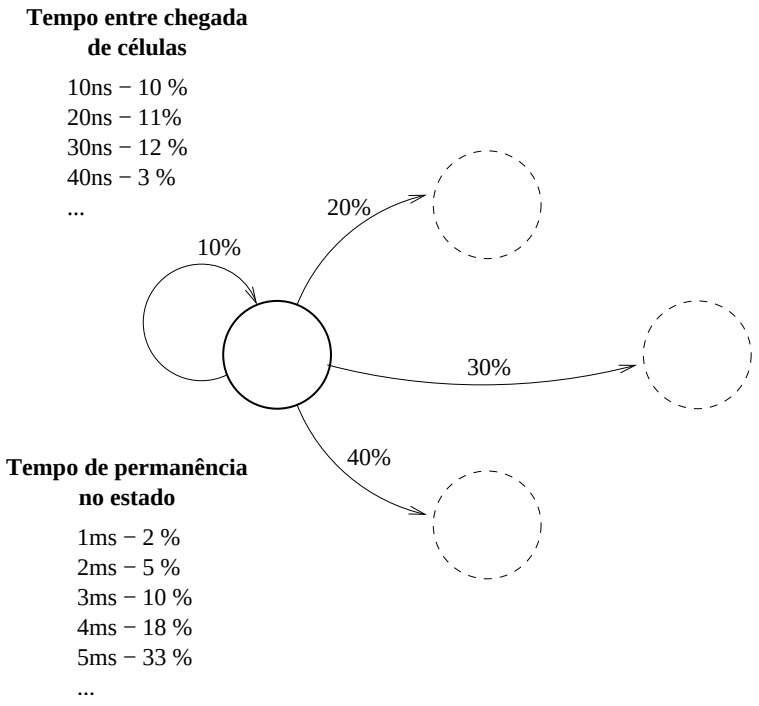


Figura 6.7: Exemplo ilustrando os valores parametrizáveis no modelo utilizado na geração de estímulos. São eles: a probabilidade de transição entre os estados, a distribuição utilizada para determinar o tempo de permanência em cada estado e a distribuição de probabilidade para o espaçamento entre células de cada estado.

A implementação do processo de geração de estímulos tem como objetivo facilitar o processo de teste da fonte e permitir uma comparação entre os resultados obtidos como diversas dimensões da mesma, além de permitir a medição de seu grau de confiabilidade e flexibilidade. A implementação feita permite uma dinâmica na troca de critérios e métodos utilizados ao longo do processo. A Seção 6.3.1 apresenta o modo proposto para a geração de variáveis aleatórias em *hardware*, enquanto a Seção 6.3.2 descreve o projeto de *hardware* da fonte de tráfego. Por último, a Seção 6.3.3 descreve as etapas realizadas na configuração da fonte. O contexto em que estão inseridos a fonte de tráfego e seu processo de configuração é mostrado na Figura 6.2. O processo de configuração da fonte de tráfego descrito na Seção 6.3.3 corresponde apenas ao módulo `interpretador`.

6.3.1 Geração de variáveis aleatórias em *hardware*

O modelo utilizado exige que a fonte de tráfego seja capaz de executar uma máquina de estados considerando o conjunto de valores parametrizáveis sem que para isso haja uma modificação na funcionalidade do *hardware*. Uma alteração deste tipo exigiria uma nova síntese do *hardware*, o que causaria um aumento no custo de compilação. Um dos principais fatores que tornou viável este projeto foi o método utilizado para geração das variáveis aleatórias a partir de uma distribuição qualquer. Como mostrado na Seção 6.1 este tem como principal vantagem a economia de memória. Além disso, sua implementação consome poucos recursos de *hardware*. Se considerarmos que, para cada estado, três variáveis aleatórias devem ser geradas e que os recursos limitados de memória dos FPGAs também serão utilizados pelo HA, a quantidade de memória exigida pela fonte torna-se um fator crítico ao processo. O projeto proposto para geração das variáveis aleatórias e sua ligação com as memórias F_i e L_i é mostrado na Figura 6.8.

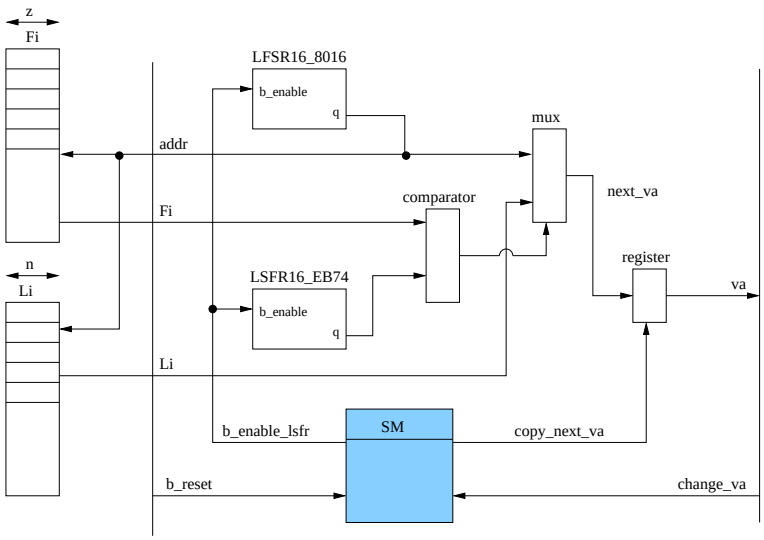


Figura 6.8: Diagrama de blocos do módulo `VA_generator`.

Este módulo, chamado de `VA_generator`, é capaz de gerar um valor de VA em um ciclo de *clock* após a solicitação, contudo é necessário que haja um intervalo de três ciclos entre solicitações. O sucesso do projeto e implementação deste módulo é vital para o funcionamento da fonte de tráfego, já que o mesmo carrega a responsabilidade de tentar capturar o comportamento estatístico do tráfego especificado através das distribuições de probabilidades. Este foi implementado e testado de forma independente do resto da fonte, permitindo seu reuso e diminuindo o tempo de projeto do *gerador de células* e do *modulador*.

A composição deste módulo requer basicamente dois geradores de números pseudo-aleatório (LFSR), um comparador, um multiplexador, um registrador e uma máquina de estados para controlar o processo. O projeto ainda permite parametrizar o número de *bits* que será usado para representar o intervalo de valores possíveis para a variável aleatória (`N_bits_va`) e o número de *bits* que será utilizado para representar os valores das probabilidades (`N_bits_probabilidade`).

O gerador de números aleatório identificado como `LFSR16_8016` tem uma interface de 16 *bits* e utiliza o polinômio $x^{15} + x^4 + x^2 + x$ (0x8016). Da mesma forma, o gerador de números aleatório `LFSR16_EB74` tem uma interface de 16 *bits* com o polinômio $x^{15} + x^{14} + x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3$ (0xEB74). Os N_bits_va menos significativos do

valor gerado por `LFSR16_8016` são utilizados como endereço das memórias F_i e L_i e como um possível valor para a variável aleatória. O valor gerado pelo `LFSR16_EB74` é repassado ao comparador, identificado na Figura 6.8 como `comparator`. Este é utilizado para identificar se o valor gerado por `LFSR16_EB74` é menor do que o valor de retorno da memória F_i. O resultado é utilizado como sinal de controle do multiplexador (`mux`) que tem como entrada o valor gerador pelo `LFSR16_8016` e o valor de retorno da memória L_i. Desta forma, a saída do multiplexador passa a ser o próximo valor da variável aleatória (`next_va`). Quando uma solicitação de geração é feita, o valor de `next_va` é escrito no registrador de saída. Todo este processo é controlado por uma máquina de estados que utiliza os sinais de controle `change_va` e `b_reset` e gera as saídas `b_enable_lfsr` e `copy_next_va`. O sinal `b_enable_lfsr` estimula os geradores de números pseudo-aleatórios a "escolherem"um novo valor, enquanto o sinal `copy_next_va` habilita a escrita no registrador de saída(`register`). A Figura 6.9 ilustra a máquina de estados utilizada no controle do processo.

A máquina é composta pelos estados `RESET`, `NEW_VA`, `GET_FI` e `WAIT_REQUEST_VA` (`NRVA`). No estado de `RESET` o processo está parado, ou seja, nenhum novo valor está sendo gerado ou copiado para o registrador de saída. Quando o sinal `b_reset` não está mais ativo, a máquina passa para o estado `NEW_VA`. Ela fica neste estado apenas um ciclo de *clock*. Neste estado, o signal `b_enable_lfsr` é ativado e um novo valor em cada gerador de números pseudo-aleatórios, ou LFSRs, é gerado. O próximo estado é o `GET_FI`. Este estado é utilizado para desativar os LFSRs e aguardar que os valores de retorno das memórias F_i e L_i estejam estáveis. No próximo ciclo de *clock* o valor do sinal `next_va` está estável e já pode ser copiado para o registrador de saída. Assim, a máquina de estado permanece em `WAIT_REQUEST_VA` até que haja uma solicitação de um novo valor de variável aleatória. Quando a solicitação é percebida, a máquina troca para o estado `COPY_VA` que habilita a escrita no registrador de saída. A Figura 6.10 mostra a forma de onda resultante de uma simulação do módulo `VA_generator`.

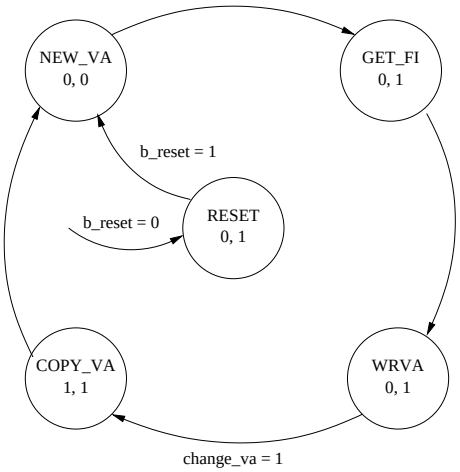


Figura 6.9: Máquina de estados do módulo `VA_generator`. A Figura mostra a máquina com o sinal de controle `change_va` e os sinais gerados `b_enable_lfsr` e o `copy_next_va` nesta ordem. `WRVA` significa `NEW_REQUEST_VA`.

A qualidade de geração das variáveis aleatórias depende diretamente da eficiência dos geradores de números pseudo-aleatórios. Os polinômios 8016 e EB74 utilizam possuem um mecanismo de detecção de zeros que permite que os mesmos gerem até 65536 (2^{16}) valores. A pesar de terem diferentes polinômios e diferentes sementes, a forma com que estes LSFRs

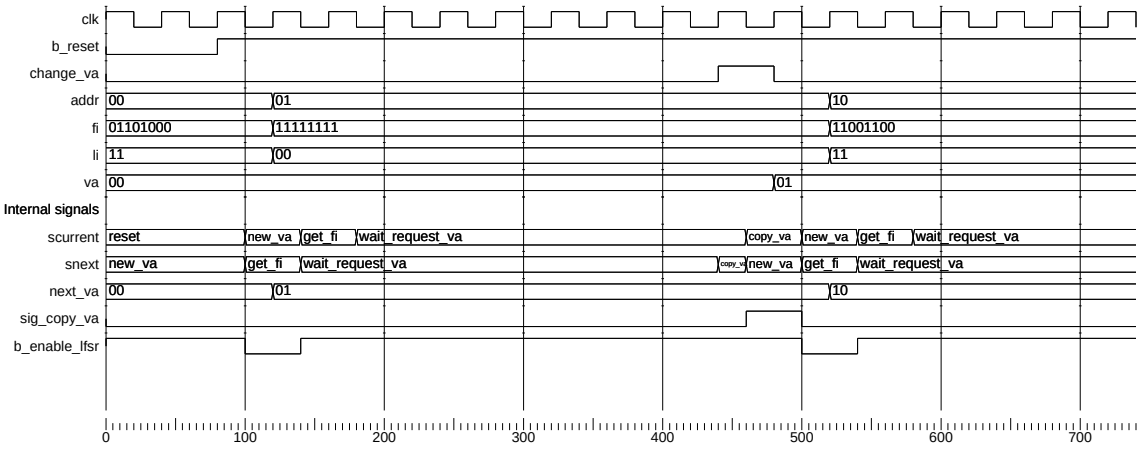


Figura 6.10: Simulação do módulo `VA_generator`. O processo tem seu início quando o sinal `b_reset` é desativado (entre 0 e $100ns$). Quando o `b_reset` ativo é percebido, a máquina troca para o estado `NEW_VA` ($100ns$). No próximo ciclo de *clock* a máquina já está no estado `GET_FI`. Na simulação, as memórias de F_i e L_i respondem instantaneamente à troca de `addr`. Neste instante já está decidido qual será o próximo valor para a VA. Assim, a máquina entra em um estado de espera pela requisição de um novo valor para variável aleatória. Esta requisição ocorre com quando o sinal *change_ va* é ativado entre $400ns$ e $500ns$. Quando a requisição é percebida, a máquina vai para o estado `COPY_VA` que habilita a escrita do valor de VA previamente gerado no registrado de saída do módulo através do sinal *copy_ next_ va*. No próximo ciclo, o novo valor de `va` já pode ser utilizado.

estão sendo utilizados causa o problema de *loops*. Os LFSRs geram uma seqüência de valores independentes de tamanho limitado, devido ao número finito de bits dos mesmos. Isto significa, que o gerador de variáveis aleatória irá gerar sempre a mesma seqüência de valores de tamanho 2^{16}. Para reduzir o efeito causado pelos *loops*, os LFSRs utilizam junto a eles um contador para detectar o período de duração de uma seqüência. Assim, quando o fim da seqüência é encontrado uma nova semente é gerada, adicionando 1 ao valor da semente antiga. Esta solução não elimina o *loop* formado pela combinação dos LFSRs mas diminui seu efeito, já que seu tamanho passa de 2^{16} para $2^{16} * 2^{16}$. A Figura 6.11 ilustra o tempo de validade de uma semente.

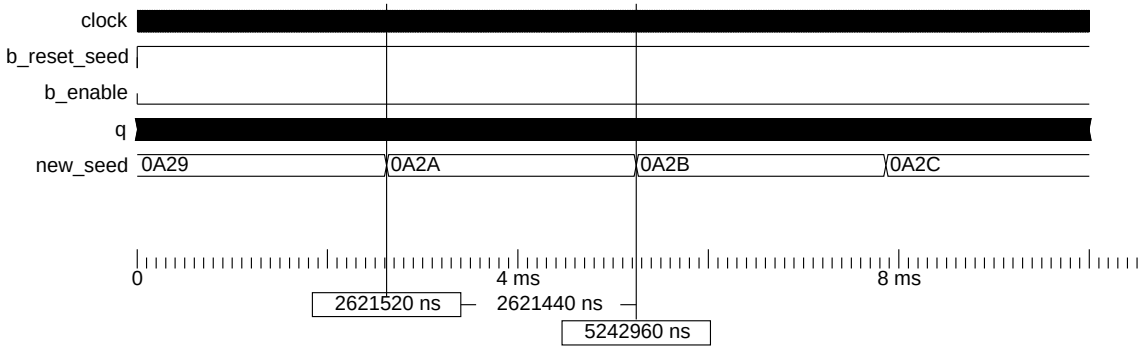


Figura 6.11: Simulação de um LFSR. A Figura mostra a medida do tempo de vida de uma semente.

Para armazenar uma distribuição de tamanho n são necessários duas memórias de 2^n endereços, uma com largura de n *bits* (L_i) e outra com largura de z *bits* (F_i), onde z determina a resolução da probabilidade (entre 0 e 100%). Contudo, partes não parametrizáveis do projeto do *hardware* combinadas ao método utilizado na geração de VAs causam algumas limitações ao processo de geração das variáveis aleatórias.

1. O valor máximo para a resolução da probabilidade é de $1,5.10^{-3}\%$. Isto ocorre devido à utilização do `LFSR16_EB74` que gera número de 16 *bits* ($100/2^{16}\%$).

2. O tamanho da distribuição não deve exceder 2^{16} valores. Da mesma foram que o item anterior, este limite existe devido à utilização de um gerador de números pseudo-aleatórios de 16 *bits* (`LFSR16_8016`).

3. Os valores da distribuição devem ser contíguos. Esta restrição é imposta pelo método.

A Figura 6.12 mostra o resultado de simulação do gerador de VA para um caso específico.

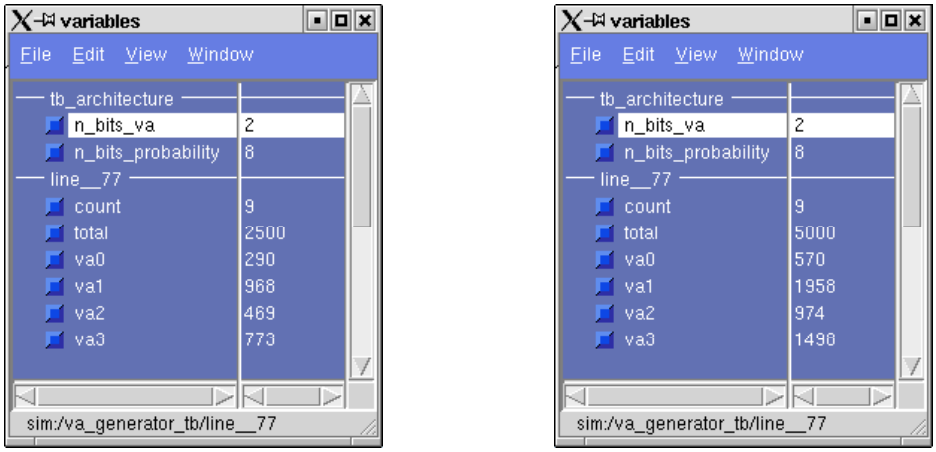


Figura 6.12: Simulação do comportamento estatístico do módulo `VA_generator`. A distribuição utilizada prevê uma probabilidade de ocorrência de 10% para $va0$, 40% para $va1$, 20% para a $va3$ e por fim 30% para a $va4$. O resultado da simulação mostra o número de ocorrências para cada va e o número total de vas geradas para os primeiros $1ms$ e $2ms$, respectivamente.

6.3.2 Estrutura de *hardware* da fonte de tráfego

Como apresentado na Seção 6.1 a fonte de tráfego é composta basicamente de dois módulos: o gerador de células e o modulador. O gerador de células tem como objetivo gerar a células ao HA com um espaçamento entre elas distribuido por uma função do usuário. Já o objetivo do modulador é criar várias instâncias do gerador de células. Cada instância é representada por um estado. O tempo de em cada estado e a transição entre eles também está distribuída de acordo com uma função do usuário. Ainda, é possível especificar a resolução de trabalho do gerador de células e do modulador para cada estado.

Com uma estrutura flexível, não é preciso uma nova síntese do *hardware*, para se mudar o comportamento da fonte, basta apenas uma nova configuração. Para isso, dois conjuntos de memórias são utilizado pela fonte. O primeiro é utilizado junto ao gerador de células. Um par de memórias F_i e L_i é utilizado para guardar as n distribuições usadas para gerar o

espaçamento entre as células, sendo n é o número de estados da fonte. Uma outra memória é utilizada para guardar o primeiro elemento significativo de cada distribuição e a resolução de trabalho do gerador de células para os n estado da fonte. A última memória utilizada pelo gerador de células armazena o valor do cabeçalho com que as células serão geradas em cada estado da fonte de tráfego. O resultado de saída do modulador determina em que estado se encontra a fonte de tráfego. Por isso, o mesmo é utilizado como parte do endereço de cada memória do gerador de células.

O segundo conjunto de memórias é utilizado junto ao modulador. Diferente do gerador de células, o modulador utiliza dois pares de memórias F_i e L_i. Um deles é utilizado para armazenar a distribuição para o tempo de permanência em cada estado da fonte, enquanto, o outro é utilizado para determinar a probabilidade de transição entre os estados. A unidade com que é contado o tempo de permanência em cada estado está armazenada em uma segunda memória, junto com o primeiro valor primeiro valor significativo da distribuição do tempo para cada estado. A Figura 6.13 ilustra a estrutura de interconexão dos princípais módulos da fonte de tráfego.

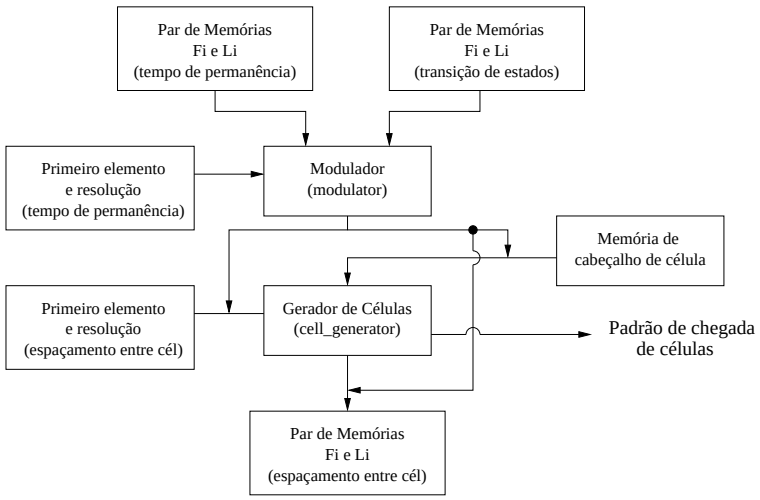


Figura 6.13: Estrutura de interconexão dos principais módulos da fonte de tráfego.

6.3.2.1 O gerador de células

O gerador de células deve gerar o espaçamento correto entre as células, e ainda montar as mesmas para serem repassadas ao HA. Como dito anteriormente, o espaçamento entre as células é gerado de acordo com uma distribuição específica dada pelo usuário. O módulo implementado para tal função é chamado de `cell_generator`. O diagrama de blocos para o projeto do *hardware* do módulo é mostrado na Figura 6.14.

Um dos módulos que compõem o projeto é chamado de `cell_maker`. Este módulo é responsável por montar a células a partir do valor de cabeçalho desejado pelo usuário e introduzir no campo de carga útil os dados úteis à avaliação de desempenho do HA. Este gera uma nova célula a cada pulso do sinal `trigger`. O restante do projeto é dedicado apenas a gerar e contar o tempo do intervalo entre células. O tempo entre células é gerado por um módulo `VA_generator` (Seção 6.3.1)e controlado pelo contador, ou `counter` como identificado na Figura 6.14. O deslocador, ou *shifter* como identificado na Figura 6.14 é utilizado para causar um deslocamento na distribuição do usuário. Isto permite uma economia de memória, já que

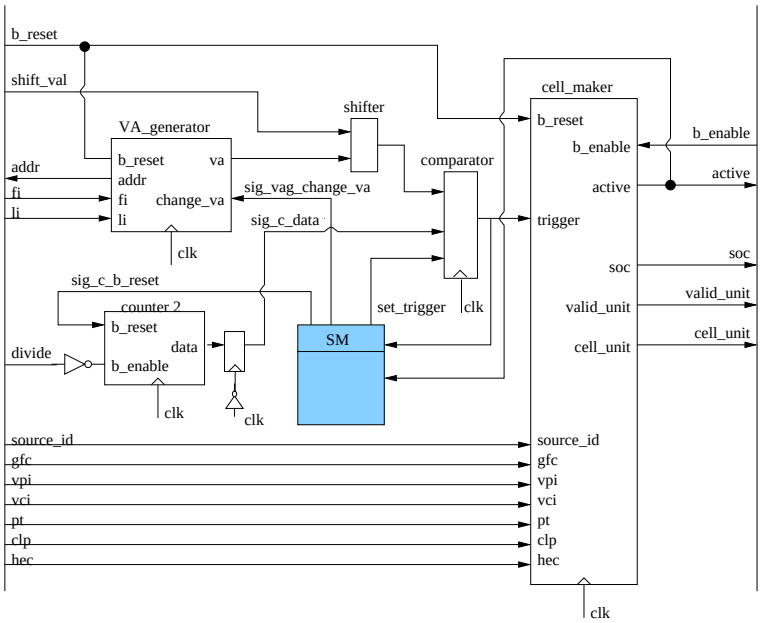


Figura 6.14: Diagrama de blocos do módulo `cell_generator`.

os valores iniciais da distribuição com probabilidade insignificante não precisam ser armazenados. No comparador (`comparator`), o sinal deslocado `sig_vag_va` é comparado com o valor do contador (`counter2`) deslocado de meio ciclo de *clock* (`sig_c_data`). O contador marca o tempo a partir da geração da última célula. Quando o comparador detecta que o tempo entre as células já atingiu o valor desejado (`sig_vag_va`), ele ativa o sinal `trigger` e uma nova célula é gerada. O processo é sincronizado por uma máquina de estados. Está máquina é ilustrada na Figura 6.15.

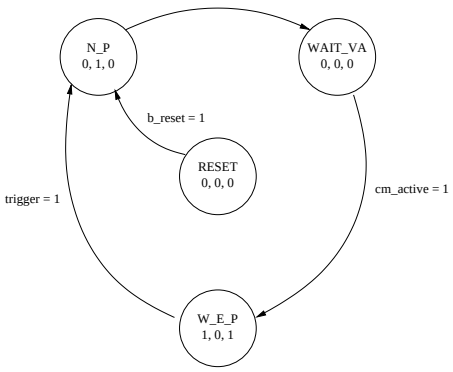


Figura 6.15: Máquina de estados do módulo `cell_generator`. A Figura mostra a máquina com o sinal de controle `trigger` e `sig_active` e os sinais gerados `sig_c_b_reset`, `sig_vag_change_va` e `set_trigger` mostrados nesta mesma ordem. `W_E_P` significa `WAIT_END_PROCESS`.

A máquina é composta pelos estados `RESET`, `NEW_PROCESS`, `WAIT_VA` e `WAIT_END_PROCESS`. Os estados `NEW_PROCESS` e `WAIT_END_PROCESS` são identificados na Figura 6.15 como `N_P` e `W_E_P`, respectivamente. No estado de `RESET`, todo o processo está parado em função do sinal

`b_reset`. Quando o sinal `b_reset` é desativado, a máquina vai para o estado `NEW_PROCESS`. Este estado marca o início de um novo processo de geração de célula. O sinal `sig_vag_change_va` é utilizado para estimular a geração de um novo valor para o tempo entre células. O próximo estado é utilizado apenas para esperar que o novo tempo entre células esteja disponível. A máquina pode permanecer neste estado por mais de um ciclo, caso alguma célula esteja sendo gerada pela fonte. O próximo passo é dar início a contagem do tempo entre as células. Isto é feito no estado `WAIT_END_PROCESS`. Este estado ativa a contagem do tempo através do sinal `sig_c_b_reset` e permite o disparo de uma nova célula através do sinal `set_trigger`. A máquina permanece neste estado até que o tempo entre células expire. Quando isto ocorre, o sinal `trigger` é ativado e um novo processo é iniciado. A Figura 6.16 mostra a forma de onda resultante de uma simulação do módulo `cell_generator`.

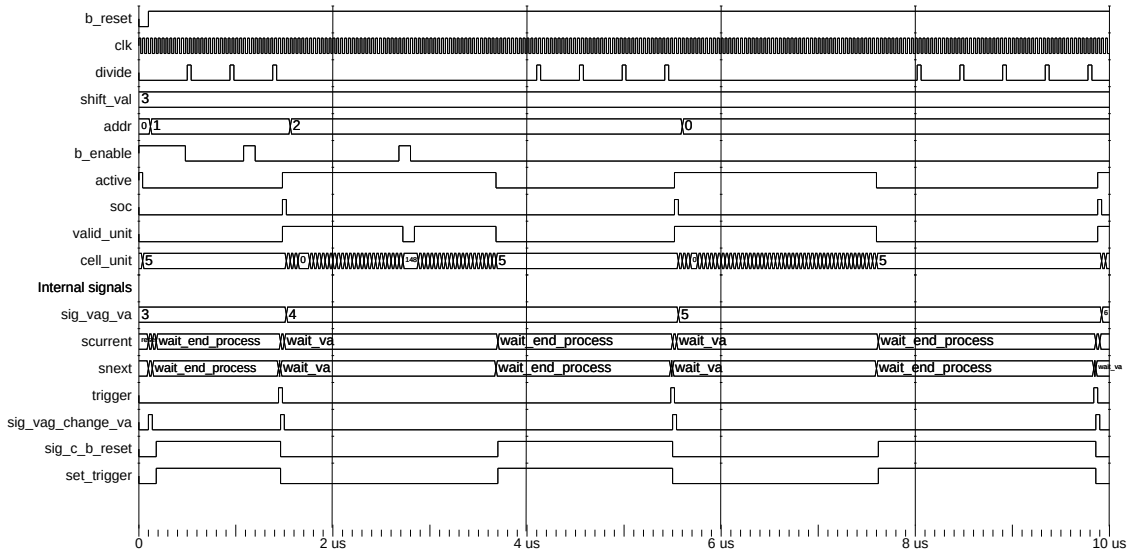


Figura 6.16: Simulação do módulo `cell_generator`.

Os sinais de interface com o HA são controlados pelo módulo `cell_maker`, mostrado na Figura 6.14. Este módulo é responsável por montar as células que serão transmitidas ao HA. A montagem pode ser feita de 8 em 8 *bits* ou de 16 em 16 *bits*, dependendo do número de *bits* da interface de dados escolhida pelo usuário. O mesmo também pode escolher se a célula montada terá ou não cabeçalho. Esta funcionalidade é útil quando se pretende utilizar a fonte para emular o comportamento de uma SAR (*Segmentation and Reassembly*) de uma camada de adaptação. Caso haja a utilização de cabeçalho, um gerador de CRC de polinômio $x^8 + x^2 + x + 1$ é usado para preencher o campo HEC. Depois de montado o cabeçalho, caso ele exista, dá-se início à montagem do campo de carga útil. O primeiro *byte* carrega o identificador da fonte e o segundo, o identificador da célula que é incrementado a cada célula gerada. O terceiro e quarto *bytes* são utilizados para marcar o tempo de partida da célula. A partir do quinto *byte*, com exceção do último, os demais são números aleatórios gerados por um LFSR. O último *byte* da célula carrega o CRC calculado a partir do campo de carga útil. O projeto de *hardware* proposto para que o módulo execute suas tarefas é mostrado na Figura 6.17.

O projeto é composto basicamente de dois contadores (`counter` e `update_cell_id`), um LFSR, um gerador de CRC, um multiplexador e duas máquinas de estados. O primeiro contador (`counter`) identifica a unidade da célula que está sendo transmitida ao HA. *Unidade da célula* é o fragmento da célula que é enviado ao HA em cada ciclo de *clock* através da

interface de dados, que pode ser de 8 ou 16 *bits*. O multiplexador tem como entrada a saída do LFSR, a saída do contador que atualiza o identificador da célula e a saída do CRC. Ele agrega suas entradas com o objetivo de formar unidades de 8 ou 16 *bits*, dependendo do número de *bits* da interface de dados com o HA. O valor de saída do multiplexador depende da unidade da célula que está sendo transmitida ao HA. Por exemplo, se a unidade da célula deve ser transmitida ao HA e se houver a presença de cabeçalho, a saída do multiplexador será o GFC e 4 *bits* do VPI para uma interface de 8 *bits* e GFC, VPI e 4 *bits* do VCI se a interface for de 16 *bits*. Em conjunto, duas máquinas de estados controlam o processo, inicializando e parando os contadores ou ativando e desativando sinais de controle da interface. Elas são mostradas na Figura 6.18.

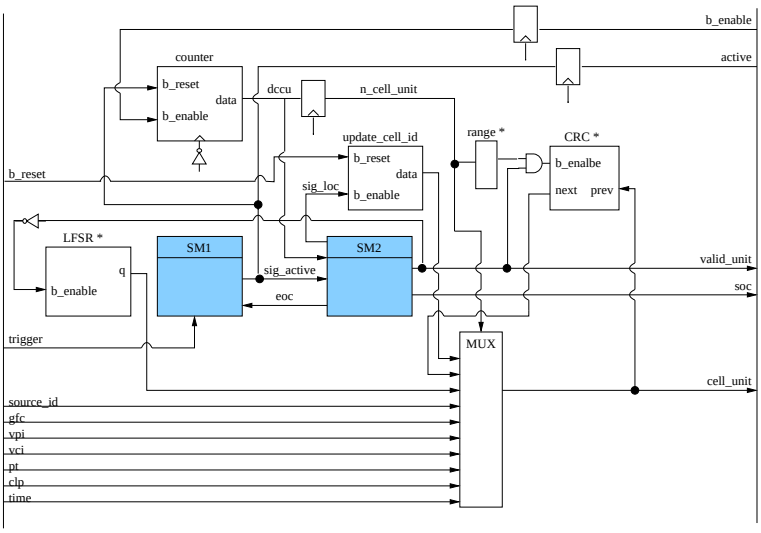


Figura 6.17: Diagrama de blocos do módulo `cell_maker`.

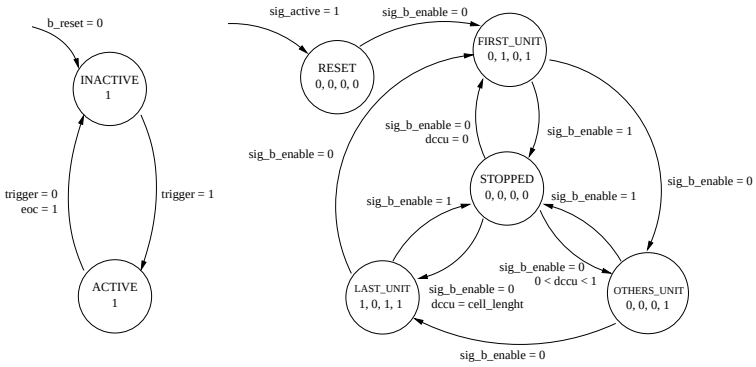


Figura 6.18: Máquinas de estados do módulo `cell_maker`. A Figura mostra a primeira máquina (`SM1`) com o sinal de controle `trigger`, `eoc` e `b_active` e o sinal gerado `sig_active`. O sinal `sig_active` é utilizado para acionar a segunda máquina (`SM2`). A segunda máquina tem os sinais de controle `sig_active`, `sig_b_enable` e `data_counter_cell_units` (`dccu`) e produz os sinais `eoc`, `valid_unit`, `soc` e `sig_loc` mostrados nesta mesma ordem na Figura.

O funcionamento das máquinas de estado mostradas na Figura 6.18 dá-se da seguinte forma. A primeira máquina (`SM1`) é inicializada no estado `INACTIVE`, e permanece neste estado

até que se solicite a geração de uma nova célula através do sinal `trigger`. Quando SM1 troca de estado, a segunda máquina (`SM2`) é ativada, saindo de seu estado de `RESET`. No estado `ACTIVE`, a máquina `SM1` utiliza o sinal `active` para indicar ao HA que uma célula está sendo transmitida. O HA pode causar uma parada no envio de células através do sinal `b_enable`. Assim, se este sinal está ativado, a máquina `SM2` sai do estado de `RESET` para `FIRST_UNIT`, ao contrário vai para o estado `STOPPED`. Isto não é uma particularidade do estado `RESET`. Estando em qualquer estado, ao detectar o sinal `b_enable` desativado, a máquina irá para o estado `STOPPED`, causando uma parada na transmissão da célula. O estado de `FIRST_UNIT` é utilizado para ativar o sinal `soc`, que indica ao HA o início de uma nova célula. No próximo ciclo de *clock*, a máquina deverá estar em `OTHER_UNITS`, se nenhuma parada for solicitada. Neste estado, as demais unidades da célula são transmitidas ao HA. Para transmitir a última unidade da célula ao HA, a máquina troca para o estado `LAST_UNIT`. Este estado ativa os sinais de controle `eoc` e `sig_loc`. O sinal `eoc` avisa à primeira máquina que a célula foi integralmente transmitida e que o módulo pode voltar ao estado `INACTIVE`. Já o sinal `sig_loc` habilita o contador `update_cell_id` para atualizar o valor de identificação de célula. Um exemplo de funcionamento do módulo `cell_maker` é mostrado através do resultado de simulação na Figura 6.19.

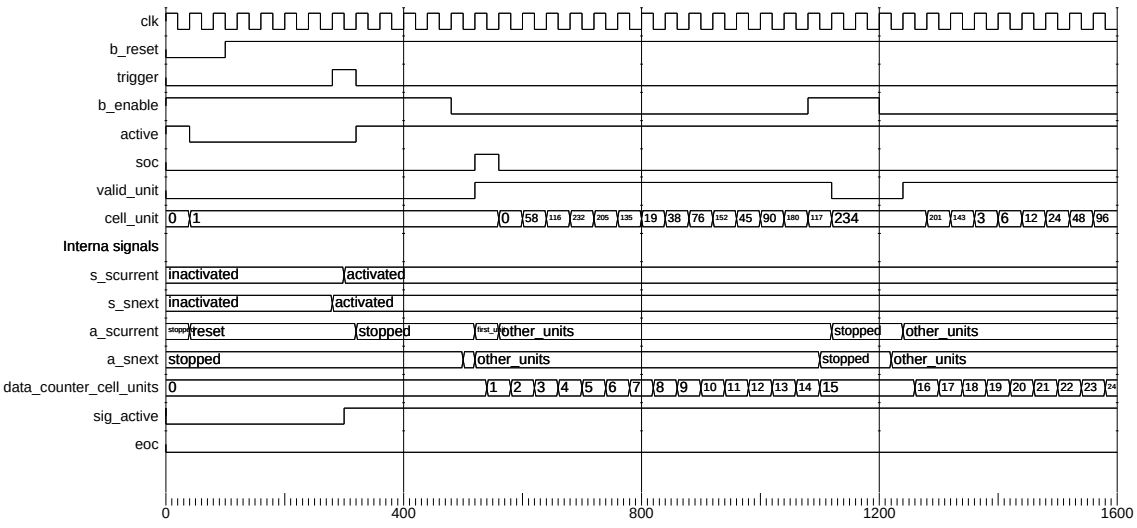


Figura 6.19: Exemplo de simulação do módulo `cell_maker`. Para o módulo `cell_maker` o processo inicia quando o sinal de entrada `trigger` é ativado (entre 0 e $400ns$). Quando isto ocorre a máquina de estados `SM1` vai para o estado `ACTIVATED`, colocando em 1 o sinal `active`. No próximo ciclo de *clock* a máquina de estados `SM2` troca para o estado `STOPPED` já que o HA está solicitando uma parada através do sinal `b_enable`. Quando o sinal `b_enable` é ativado, a máquina `SM2` vai para o estado `FIRST_UNIT` para transmitir a primeira unidade da célula (8 ou 16 bits) e ativar o sinal `soc`, para avisar o HA que a primeira unidade da célula está sendo enviada. O sinal `valid_unit` indica ao HA que a unidade de célula que está na interface de dados é válida. O próximo estado é chamado de `OTHER_UNITS`. Neste estado, se usado, o restante do cabeçalho é transmitido ao HA. Quando não há mais cabeçalho a ser transmitido os valores produzidos pelo gerador de números aleatórios são repassados ao HA.

6.3.2.2 O Modulador

Até agora, foi apresentado apenas o projeto dos módulos que compõem o gerador de células (módulo `cell_generator`). Apenas com a utilização deste módulo já é possível a geração de tráfego descrita por apenas uma distribuição de probabilidade para os intervalos entre células. Contudo, a sua utilização não permite a representação de nenhum tipo de dependência. Para permitir a representação de dependências curtas foi proposta a utilização de um módulo chamado de `modulator`, descrito na Seção 6.1. Este módulo tem como objetivo criar várias instâncias do gerador de células como em um processo MMDP ou MMPP. Cada instância do gerador implementa um estado do processo. O modulador deve decidir quanto tempo permanecer em um estado e para qual estado deve-se trocar, quando o tempo de permanência no estado atual expirar. O uso do modulador permite a utilização de uma máquina de estados como a da Figura 6.7 como um modelo de tráfego. Para permitir tal funcionalidade ao modulador, o projeto ilustrado na Figura 6.20 foi proposto, implementado e testado. Os estados no modelo de tráfego serão chamados de estados da fonte.

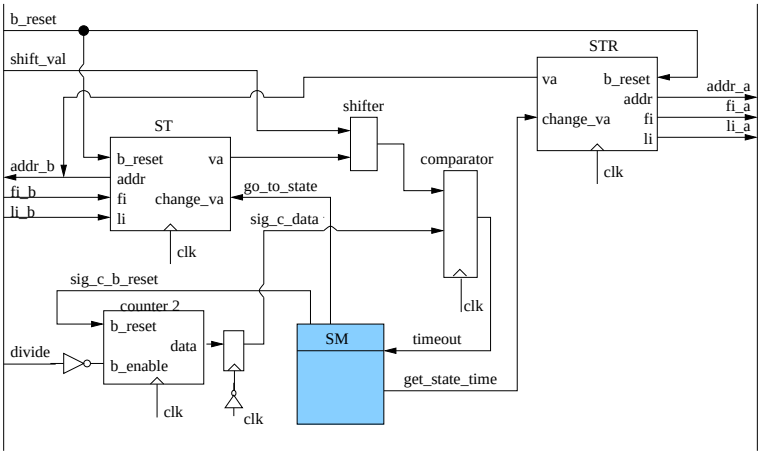


Figura 6.20: Diagrama de blocos do modulador.

Basicamente, o modulador é composto por dois geradores de VAs `VA_generator`, um contador, um deslocador, um comparador e uma máquina de estados para controlar o processo. Na Figura 6.20 estes módulos são identificados por `STR`, `ST`, `counter`, `shifter`, `comparator` e `SM` (*State Machine*), respectivamente. `STR` é um gerador de VA responsável por decidir qual será o próximo estado da fonte. O segundo gerador de VA, o módulo `ST`, determina, após a troca de estado da fonte, quanto tempo a fonte permanecerá neste novo estado. Por sua vez, o contador, o somador e o comparador são utilizados para determinar o instante em que o estado da fonte deve ser trocado. O tempo de permanência gerado pelo módulo `ST` (saída `va`) é adicionado à entrada `shift_va` pelo deslocador. O comparador, por sua vez, compara a saída do somador `sig_st_va` mostrada na Figura 6.20 com o valor do contador. Quando a saída do contador for igual a `sig_st_va` significa que o tempo de permanência no estado da fonte expirou e que este deve ser trocado. O sinal de entrada `divide`, também mostrado na Figura 6.20, é utilizado para determinar a unidade de tempo do contador. Este sinal é resultado da divisão do *clock* por um valor especificado pelo usuário. Este valor determina a unidade de tempo da distribuição do tempo de permanência em cada estado da fonte. O sinal `divide` foi adicionado à interface para permitir a utilização de diferentes unidades de tempo pelos estados da fonte. O processo de modulação é controlado pela máquina de estados `SM`

ilustrada na Figura 6.21.

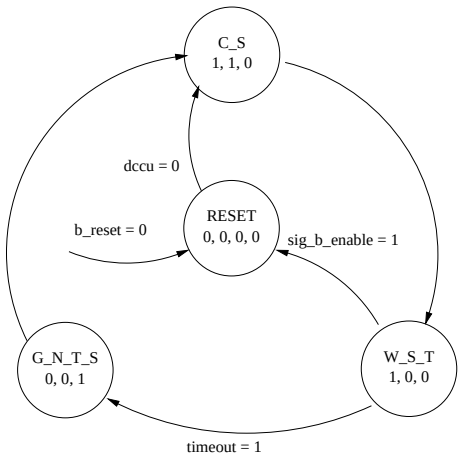


Figura 6.21: Máquinas de estados do módulo `modulator`. A Figura mostra a máquina (`SM`) com o sinal de controle `b_reset`, `timeout`, `dccu` e os sinais de saída `go_to_state`, `get_state` e `sig_c_b_reset`. Os estados `C_S`, `W_S_T` e `G_N_S_T` significam respectivamente `CHANGE_STATE`, `WAIT_STATE_TIME` e `GET_NEW_STATE_TIME`

.

A máquina que controla todo o processo no modulador é simples. Quando o sinal `b_reset` é desativado, inicia-se o processo. O primeiro passo é determinar qual deve ser o próximo estado da fonte. O novo estado é dado pela saída `va` do módulo `STR`. Então, para executar o primeiro passo, a máquina de estados `SM` ativa o sinal `go_to_state`, fazendo com que o módulo `STR` produza o valor do novo estado da fonte. Assim, este valor passa a ser o estado atual. O estado da fonte atual, junto com a saída `addr` do módulo `ST`, é utilizado como endereço das memórias que armazenam a distribuição do tempo de permanência em cada estado da fonte. Isto permite ao usuário especificar uma distribuição de probabilidades diferente para o tempo de permanência em cada estado da fonte. No próximo ciclo, a máquina `SM` estará no estado `GET_NEW_STATE_TIME` (`G_N_S_T`). Este estado habilita a geração de um tempo de permanência para o novo estado da fonte. No próximo estado, `WAIT_STATE_TIME`, a contagem do tempo de permanência é disparada através do sinal `sig_c_b_reset`. A máquina fica neste estado até que o sinal `timeout` avise que o tempo de permanência no estado da fonte atual expirou. Quando isto ocorre, o processo é reiniciado, voltando ao estado `CHANGE_STATE`. A Figura 6.22 mostra um exemplo de funcionamento do modulador.

6.3.2.3 O gerador de tráfego

O último módulo a ser descrito é utilizado para integrar o modulador (`modulator`) e o gerador de células (`cell_generator`). A Figura 6.23 ilustra através de um diagrama de blocos a integração dos módulos `cell_generator` e `modulator` e o dimensionamento suportado pela fonte de tráfego. A Figura 6.23 também apresenta três novos componentes. O primeiro, identificado na Figura 6.23 como `cell_head_memory`, guarda os valores dos campos de cabeçalho das células geradas em cada estado da fonte. A utilização de um valor de cabeçalho da célula pra cada estado, permite que a fonte de tráfego emule o comportamento de um fluxo de tráfego com várias conexões sendo multiplexadas. O segundo é um par memórias utilizadas para armazenar o valor de deslocamento da função e a unidade de tempo

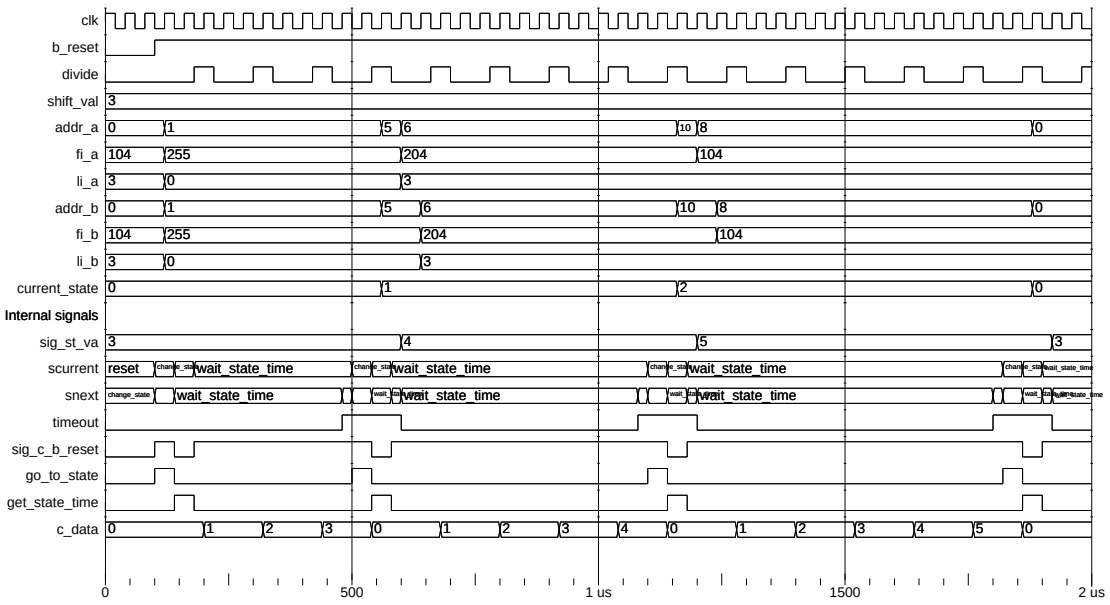


Figura 6.22: Exemplo de simulação do módulo `modulator`. Nos primeiros $100ns$ a máquina `SM` está no estado `RESET` devido ao sinal `b_reset`. Quando o sinal `b_reset` é desativado a máquina `SM` vai para os estado `CHANGE_STATE` ativando o sinal `go_to_state` para causar uma troca de estado da fonte. O modulador que estava no estado da fonte 0 (`current_state`) permanece neste estado mesmo após a solicitação feita através do sinal `go_to_state`. Isto ocorreu porque havia uma probabilidade de que o modulador se mantivesse neste estado. No próximo ciclo de *clock*, a máquina `SM` vai para o estado `GET_NEW_TIME_STATE` para gerar um valor para o tempo de permanência no novo estado da fonte. Feito isto, a máquina `SM` vai para o estado `WAIT_STATE_TIME` em que aguarda o tempo de permanência atribuído ao estado da fonte. Este tempo pode ser contado na Figura através do número de pulsos do sinal `divide`. Este número deve ser igual ao valor do sinal `sig_st_va`.

de trabalho (ou resolução de tempo) para cada estado. Estas memórias são utilizadas junto com o modulador e com o gerador de células. Elas são identificadas na Figura 6.23 por `modulator_F`, `modulator_R`, `cell_generator_F` e `cell_generator_R`. O último componente da fonte de tráfego é o gerador de pulsos de tempo ou divisor de tempo. Este componente é utilizado junto as memórias `modulator_R` e `cell_generator_R` de forma a ditar o ritmo de trabalho do modulador e do gerador de células. Os mesmos são identificados na Figura 6.23 como `modulator_divider` e `cell_generator_divider` respectivamente. Como as memórias `modulator_R` e `cell_generator_R` guardam um valor de unidade de tempo de trabalho para cada estado da fonte, o ritmo de trabalho do modulador e do gerador de células pode ser diferente para cada estado. Por exemplo, para um estado da fonte o seu tempo de permanência pode ser dado em ms enquanto para um outro estado o seu tempo pode ser contado em ns. Se a unidade de trabalho de um determinado estado da fonte for zero, nenhum pulso é gerado. Unidade de trabalho de valor zero não é válida para ser usado com o modulador, já que se nenhum pulso de trabalho for gerado a fonte permanecerá indefinidamente no mesmo estado da fonte. Contudo, a unidade de trabalho de valor zero pode ser utilizada junto ao gerador de células para emular o comportamento de um estado inativo de uma fonte ON/OFF. A Seção 7 apresenta alguns de exemplos de funcionamento da fonte e de sua parametrização.

Os módulos de *software* são responsáveis por preencher as memórias L_i e F_i e ajudar o

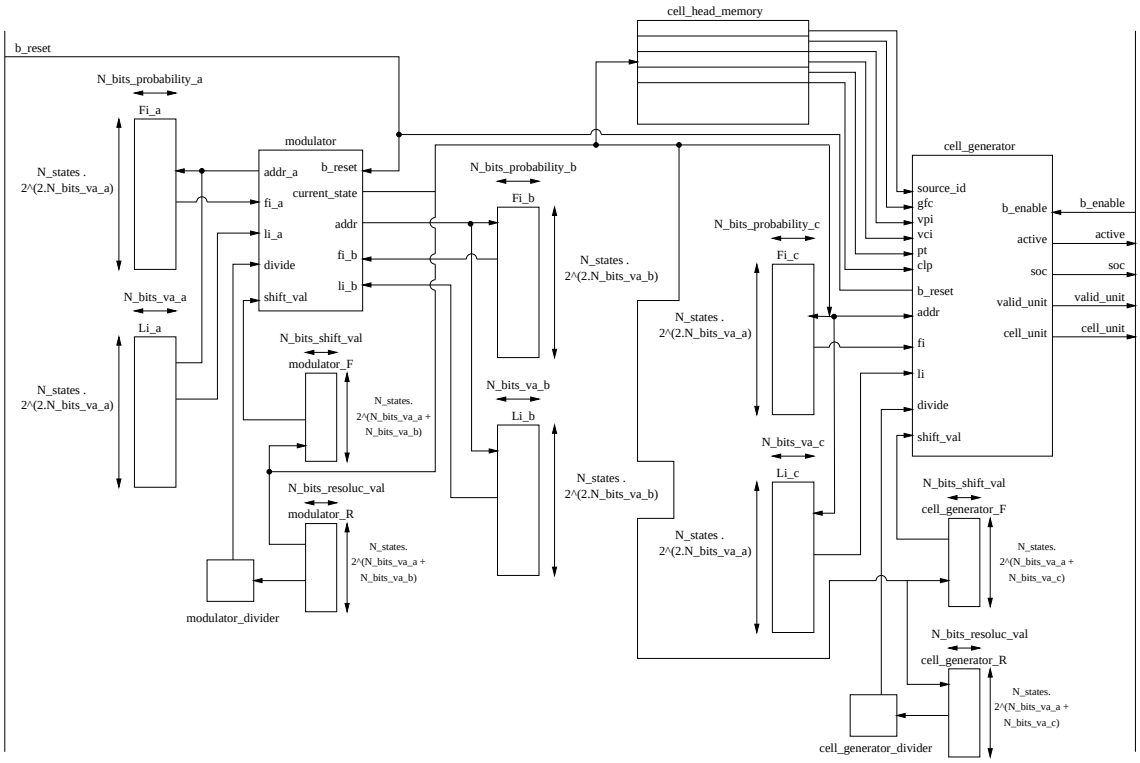


Figura 6.23: Diagrama de blocos do módulo `cell_source`.

usuário a determinar os valores adequados à parametrização da fonte de tráfego. Como dito anteriormente, a parte de *software* do processo de geração de estímulos é composta basicamente pelos módulos `interpretador` e pelo `configurador`. Para atingir os objetivos propostos, apenas o `interpretador` foi implementado. Este foi desenvolvido como um conjunto de programas que geram os dados úteis ao processo, e um *scripts* para guiar o processo de configuração. Esta forma de implementação permitiu uma ótima flexibilidade e escalabilidade na geração dos dados de configuração da fonte. A Seção seguinte descreve o fluxo de geração dos dados necessários à configuração da fonte utilizando os programas implementados.

6.3.3 Processo de configuração da fonte de tráfego

A Figura 6.2 mostra os módulos envolvidos no processo de configuração. A parte do processo de configuração da fonte descrita nesta Seção corresponde ao `interpretador`. Esta é composta basicamente de três etapas: a geração das distribuições de probabilidade utilizadas pelas variáveis aleatórias, a adaptação destes valores à quantidade de memória da fonte e a geração dos valores das memórias L_i e F_i utilizando o algoritmo de Kronmal e Peterson. Nota-se que as duas primeiras etapas do processo são fortemente dependentes do tipo de distribuição, o que exige boa escalabilidade da implementação do processo. Com isso, percebe-se que apenas a utilização de um único programa tornaria difícil aumentar o número de distribuições de probabilidade suportadas pelo processo. Assim, visando apenas a flexibilidade da realização dos testes, o processo foi implementado como um conjunto de programas controlados por um *script* que interagem com ferramentas proprietárias através de arquivos. A este *script* foi dado o nome `scpt`. Esta solução permite ao usuário incluir e retirar um tipo de distribuição apenas com a inclusão e remoção de programas, dando liberdade à utilização de novos critérios

durante o processo. A Figura 6.24 mostra os programas desenvolvidos como parte do trabalho, os programas proprietários e a interação entre eles através de arquivos na implementação do processo de configuração da fonte.

O *script* principal, chamado `scpt`, é uma das principais contribuições deste trabalho. Sua interpretação permite que se entenda o conjunto de passos necessários a configuração da fonte. O `scpt` é um *shell script* que pode trabalhar basicamente em dois modos: *default* ou interativo. No modo interativo, o *script* pergunta ao usuário todas as informações que precisa. No modo *default* os valores necessários são configurados no próprio *script*, de modo a facilitar o processo de teste. Um exemplo de parametrização do *script* é mostrado na Figura 6.25.

Este *script* irá utilizar um conjunto de programas para realizar as etapas de configuração da fonte. As ações realizadas em cada etapa devem ser separadas em programas para permitir uma boa escalabilidade no processo e uma boa agilidade na troca de critérios. Para atingir o nível de flexibilidade desejado, as seguintes questões foram respondidas:

1. Quais são as variáveis do processo?
2. Quais são as etapas do processo dependem destas variáveis?
3. Em cada etapa, quais são as ações que podem ou devem ser executadas de mais de uma forma?

A primeira questão tem como resposta o tipo e o número de elementos da distribuição e os valores de dimensionamento da fonte. Também é fácil perceber que as três etapas de configuração citadas no início da Seção estão diretamente relacionadas com as variáveis do processo, o que responde a segunda questão. A resposta da última questão requer alguma descrição sobre as ações realizadas em cada etapa.

A etapa de geração das distribuições de probabilidade é composta por apenas uma ação. Esta deve gerar os valores significativos da distribuição para cada uma das VAs envolvidas. Sendo esta ação uma possível resposta para a terceira questão, a implementação prevê um programa para gerar os valores significativos da distribuição para uma VA, enquanto o *script* trata de executá-lo para cada uma das variáveis aleatórias do processo. A Tabela 6.1 mostra o resultado produzido pelo programa utilizado para gerar uma distribuição de Poisson. Se o usuário já possui suas distribuições de probabilidade, esta etapa pode ser ignorada.

Para fins de teste, os programas `poisson.c`, `exp.c` e `pareto.c` foram implementados para geração de distribuições do tipo Poisson, exponencial e Pareto, respectivamente. Os parâmetros de entrada dos programas utilizados para geração de distribuições variam de acordo com o tipo de distribuição a ser gerada. Por exemplo, os programas `poisson.c` e `exp.c` recebem como entrada apenas a média da distribuição. Já o programa `pareto.c` foi implementado utilizando a distribuição de Pareto translada. Este tipo de distribuição permite associação de valores independente para a média e para o efeito Noé, representado por α. Assim, os parâmetros de entrada do programa `pareto.c` são a média e *alpha*. Estes programas geram apenas uma resposta, a distribuição desejada, também chamada de distribuição original. Esta é repassada a próxima etapa através de um arquivo com um nome escolhido pelo usuário. Estes programas encontram-se nos Apêndices A, B e C.

A próxima etapa é a adaptação dos valores da distribuição às dimensões da memória. Para se analisar as ações a serem tomadas nesta etapa é necessário avaliar as condições para cada estado isoladamente, já que a fonte de tráfego reserva quantidades idênticas de recursos para cada estado. Assim, analisando apenas o número de endereços da memória, o que deve ser feito se o recurso de memória da fonte para uma determinada variável aleatória de um estado

Tabela 6.1: Exemplo de distribuição gerada na primeira etapa do processo

Elemento	Probabilidade	Elemento	Probabilidade
1	0.000454	13	0.072908
2	0.002270	14	0.052077
3	0.007567	15	0.034718
4	0.018917	16	0.021699
5	0.037833	17	0.012764
6	0.063055	18	0.007091
7	0.090079	19	0.003732
8	0.112599	20	0.001866
9	0.125110	21	0.000889
10	0.125110	22	0.000404
11	0.113736	23	0.000176
12	0.094780		

não for suficiente para representar todos os valores significativos de uma distribuição? Neste caso, há duas opções: aumenta-se os recursos de memória da fonte no FPGA ou diminui-se o tamanho da distribuição. Para se manter a fidelidade do modelo utilizado, a melhor solução seria aumentar os recursos de memória, contudo isto nem sempre é possível já que os recursos de memória do FPGA são disputados entre a fonte e o HA. Também deve-se considerar que a fonte deve reservar quantidades iguais de recursos para seus estado, isto significa que para aumentar o tamanho da memória de um estado, é necessário que também se faça isto para os demais. Assim, deve-se considerar que a redução da distribuição pode ser uma solução para o problema. Contudo, o problema agora passa a ser como executar esta redução. Que critérios devem ser utilizados? Os critérios devem ser os mesmos para qualquer tipo de distribuição? Assim como a ação anterior, sendo a adaptação da distribuição com relação ao número de endereços da memória uma possível resposta para a terceira questão, esta ação deve ser executada isoladamente por um programa.

O programa desenvolvido para a ação de redução da distribuição foi chamado de `aproximar.c`. Este programa utiliza um critério muito simples para reduzir a distribuição. Ele divide o número de elementos da distribuição pela quantidade de memória disponível, criando intervalos de elementos da distribuição. Após, o programa soma as probabilidades dos elementos de cada intervalo e o resultado é atribuído a média aritimética dos elementos do intervalo e armazenado em uma posição da memória. Por exemplo, se a distribuição for 6 vezes maior que o espaço de memória disponível cada intervalor terá 6 elementos. Se um dos intervalos tiver os elementos $1, 2, 3, 4, 5 e 6$ com probabilidades de 1%, 2%, 3%, 4%, 5% e 6%, o resultado será um elemento de valor 3 ($\frac{1+2+3+4+5+6}{6}$) com probabilidade 21% (1%+2%+3%+4%+5%+6%). Este programa recebe como entrada a distribuição original, o espaço de memória disponível para a distribuição e o nome que serão utilizados pelos seus arquivos de saída. O mesmo produz dois arquivos de saída. O primeiro arquivo (com uma extensão $\_disc$) contém o resultado da redução, também chamada de distribuição reduzida. O segundo arquivo (com a extensão $\_rel$) contém uma distribuição semelhante à original, contudo todos os elementos que pertencem a um mesmo intervalo determinado pelo programa têm o mesmo valor de probabilidade, a média da soma de suas probabilidades na distribuição orignal. Esta distribuição representa a relação entre a distribuição original e a distribuição reduzida.

Toda a troca de informações entre os programas é feita através de arquivos. Os dados da distribuição devem ser gerados com o formato apresentado na Tabela 6.1. A utilização de um padrão como este permite a utilização de ferramentas como o `gnuplot` e `latex` para facilitar a visualização dos resultados produzidos. Assim, ao final é gerado um relatório com explicações das ações executadas ao longo do processo e gráficos que permitem a identificação de erros introduzidos devido à quantidade de recursos de memória e às restrições impostas pelo método utilizado na geração de VAs. A Figura 6.26 mostra um gráfico que permite comparar uma distribuição de Poisson com média 1000 com sua versão reduzida.

A próxima ação que deve ser realizada na etapa de adaptação é com relação à largura da memória F_i. Quantos *bits* devem ser utilizados para representar com fidelidade a probabilidade dos valores da distribuição? E ainda, sabe-se que os valores armazenados nas memórias da fonte não são os valores da distribuição, mas sim do resultado do algoritmo de Kronmal e Peterson. Contudo, é possível observar que o algoritmo de Kronmal e Peterson apresentado na Seção 6.1 gera os valores de F_i apenas com soma e produto das probabilidades da distribuição. Isto significa que se todos os valores da distribuição são representáveis pelo número de *bits* da largura de F_i, os valores gerados pelo algoritmo de Kronmal e Peterson também serão. Assim, verifica-se que a solução descoberta para adaptar o valor das probabilidades à largura da memória de F_i não depende do tipo de distribuição, mas sim da técnica utilizada na geração das variáveis aleatórias. Como o processo de geração de estímulos não utiliza nenhuma outra técnica para geração de VAs, a ação de adaptação não está entre as respostas da terceira questão, ou seja, não precisa ser isolada em um programa. Esta ação é executada no mesmo programa que as ações da etapa de geração dos valores das memórias L_i e F_i.

A próxima etapa é marcada pelas seguintes ações: geração efetiva dos valores de configuração da fonte, teste dos valores gerados e escrita do arquivo VHDL para a simulação. O primeiro passo para a geração dos valores de configuração da fonte é a execução do algoritmo de Kronmal e Peterson. Durante o trabalho, dois algoritmos para o preenchimento das memórias L_i e F_i foram estudados. O primeiro foi o algoritmo de Walker proposto em 1977. Os passos executados pelo algoritmo são mostrados abaixo.

1. $L_i = i$, $F_i = 0$, e $b_i = p(i) - 1/(n+1)$ para $i = 0, 1, ..., n$
2. Para i variando de 0 até n fazer
   (a) $c = min\{b_0, b_1, ..., b_n\}$, sendo k o índice do valor mínimo
   (b) $d = max(b_0, b_1, ..., b_n)$, sendo m o índice do máximo valor
   (c) se $\sum_{j=0}^{n} |b_j| < \epsilon$, parar o algoritmo
   (d) $L_k = m$, $F_k = 1 + c(n+1)$, $b_k = 0$ e $b_m = c + d$

Este algoritmo na prática apresenta apenas uma vantagem com relação ao algoritmo de Kronmal e Peterson. Ao contrário deste último, o algoritmo de Walker não causa nenhum tipo de falha devido a soma das probabilidades ser diferente de 1. Esta característica do algoritmo de Kronmal e Peterson torna sua utilização inviável em muitas aplicações práticas. Em particular, depois de algumas alterações feitas sobre a distribuições do usuário para que a mesma se adequasse aos recursos de *hardware* e a técnica de geração das VAs, esperava-se que a soma de probabilidade destas distribuições não atingisse exatamente 100%. Apesar desta restrição, o algoritmo de Kronmal e Peterson apresenta uma importante vantagem quando comparado ao algoritmo de Walker. O cálculo dos valores da memória F_i é feito utilizando apenas somas

Tabela 6.2: Comparação entre os algoritmos de Kronmal e Peterson.

Distribuições		**Erro produzido pelo algoritmo**	
Erro máximo	**Número médio de elementos**	**Walker**	**Kronmal e Peterson (alterado)**
1%	200,592	0,9023	0,0924
2%	66,542	2,7427	1,8763
3%	39,928	4,3759	3,1005
4%	28,11	6,2362	4,1955
5%	21,918	7,5827	5,0748
10%	10.24	16,578	8,1898

das probabilidades da distribuição e a constante 100%. Como dito anteriormente, esta característica permite garantir que os valores que serão gerados pelo algoritmo serão representáveis nas memórias apenas tornando os valores da distribuição em questão, também representáveis na memória. Adaptar os valores de F_i depois da execução do algoritmo adiciona ao processo um erro considerado inaceitável na maioria dos casos. O algoritmo de Kronmal e Peterson também apresenta-se mais eficiente que o algoritmo de Walker com relação ao seu tempo de execução.

Uma pequena alteração foi feita no algoritmo de forma que o mesmo pudesse produzir valores aproximados aos do algoritmo original, contudo eliminando o erro durante a execução. Para resolver o problema, várias simulações foram feitas. Foi observado que tanto para as distribuições com a soma das probabilidades maiores que 1 quanto para as menores que 1, o erro era causado por que o conjunto G esvaziava antes do conjunto S. No algoritmo original, apresentado na Seção 6.1, o conjunto S era resultado de todos os valores de i onde $F_i \leq 1$, sendo $F_i = F_k - 1 + F_m$. Notou-se que o erro causado pela imprecisão da soma das probabilidades poderia ser compensado fazendo $F_i = F_k - sum + F_m$, onde sum é a soma das probabilidades. A condição para pertencer ao grupo S foi trocada de $F_i \leq 1$ para $F_i \leq sum$. É possível notar que a alteração feita ao algoritmo original ainda gera valores representáveis em F_i, quando os valores da distribuição também são representáveis em F_i.

A Tabela 6.2 compara os resultados produzidos pelos algoritmos quando submetidos a diferentes variações de erros na soma das probabilidades da distribuição. Os erros de cada algoritmo foram medidos somando a diferença no valor da probabilidade obtida para cada elemento da distribuição. A primeira coluna da Tabela 6.2 mostra o intervalo de erro permitido. Por exemplo, para a última linha da tabela, as distribuições foram geradas com um erro de no máximo 10%, isto significa que a soma das probabilidade das mesmas variam entre 91.0% a 110%. A segunda coluna mostra o número médio de elementos das distribuições, enquanto as duas últimas colunas apresentam os erros introduzidos pelos algoritmos de Walker e de Kronmal e Peterson respectivamente. A Tabela 6.2 mostra que na prática os resultados obtidos com o algoritmo de alterado de Kronmal e Peterson apresentam uma menor introdução de erro.

Após a geração do conteúdo das memórias L_i e F_i da fonte de tráfego, devem ser gerados os valores das memórias de resolução e de primeiro elemento. A memória de resolução permite que a fonte possa trabalhar com unidades diferentes de tempo para cada estado na máquina de estados do usuário. Por exemplo, é possível utilizar uma distribuição em milisegundo para o tempo de permanência no estado 1, enquanto a distribuição para o tempo de permanência

no estado 2 esta expressa em segundos. O valor desta memória para cada estado é gerado pelo cálculo: *número de clocks de uma unidade da distribuição x intervalo entre os valores da distribuição*. Como dito anteriormente, a técnica utilizada para gerar as variáveis aleatórias exige que os valores da distribuição sejam contíguos. Contudo, a utilização da memória de resolução permite que os valores da distribuição não sejam contíguos, mas exige que os mesmos sejam um subconjunto de uma progressão aritmética.

Já a memória de primeiro elemento, por sua vez, permite causar um deslocamento à distribuição de cada estado da fonte. Por exemplo, para uma distribuição de Poisson com média 100, o primeiro valor com probabilidade significativa é o 67, considerando apenas os valores com probabilidades a cima de 0,01%. Sem a utilização da memória de primeiro elemento, seria necessário armazenar todos os valores iniciais da distribuição. Para o exemplo anterior, o tamanho da memória para armazenar integralmente a distribuição passaria de 140 endereços para 272 endereços. As memórias de resolução e de primeiro elemento são utilizadas junto as variáveis aleatórias de tempo de permanëncia no estado e de intervalo entre células.

A última ação desta etapa, ainda executada pelo mesmo programa, testa a qualidade dos valores de configuração gerados simulando o processo de geração de VAs para cada uma das variáveis aleatórias do processo, através de um teste exaustivo. Neste teste, todas as possíveis combinações dos geradores de números pseudo aleatórios são utilizadas. O resultados deste teste podem ser comparados às distribuições originais utilizadas pelo usuário, permitindo ao mesmo observar o erro introduzido pelo processo.

O programa que implementa todas estas ações foi chamado de `conf.c`. Este programa recebe como entrada o tamanho da distribuição, já reduzida ou não, o número de bits que será utilizado para armazenar as probabilidades de cada elemento da distribuição, o nome do arquivo que contém a distribuição e o nome que será utilizado pelos seus arquivos de saída. O mesmo produz cinco arquivos de saída. O primeiro arquivo (com a extensão `_fr`) guarda o valor do primeiro elemento significativo da distribuição. O segundo arquivo (com a extensão `_r`) trás o valor da resolução que deverá ser utilizado. No terceiro e quarto arquivos (com extensão `_F` e `_L`) contém os valores das memórias F_i e L_i gerados a partir de uma implementação do algoritmo de Kronmal e Peterson alterado. O último (com extensão `_test`), contém o resultado do teste de *software* realizado sobre os valores das memórias F_i e L_i.

Por fim, o *script* principal gera e grava no arquivo `dimensions.out` as sementes que serão utilizadas pelo par de LFSR dos módulos `va_generator` da fonte de tráfego. Um segundo *script*, chamado de `chbase`, é utilizado para transformar os valores de dimensionamento e de configuração gerados nas etapas anteriores em números binários. Para facilitar a utilização da fonte de tráfego, tanto para a simulação quanto para o síntese, o arquivo VHDL que a descreve traz "marcas" nos lugar dos valores de dimensionamento e de configuração da fonte. Assim, um outro *script*, chamado de `sub.awk`, é utilizado para substituir as "marcas" do arquivo VHDL da fonte pelos seus devidos valores de dimensionamento e configuração. Pronto, o arquivo VHDL gerado está pronto para ser simulado.

Ao final desta etapa, todas as adaptações às dimensões da fonte e a geração de seus valores de configuração devem ter sido feitas. Com as informações produzidas pelo processo e gravadas em arquivos, o *script* `scpt`, responsável por integrar os programas e guiar o processo, gera um relatório com gráficos e tabelas descrevendo cada passo realizado. Os programas utilizados no processo de configuração da fonte geram suas saídas em uma forma adequada para ser utilizada pela ferramenta `gnuplot`. O *script* principal utiliza o `gnuplot` para transformar as distribuições em gráficos. Outros três *scripts* (`i_report`, `p_report` e `e_report`) se encarregam de montar um texto, escrito em LaTeXque descreve as ações realizadas durante as etapas da configuração utilizando os gráficos gerados pelo `gnuplot`.

Abaixo são resumidas as principais atividades dos programas e *scripts* desenvolvidos.

scpt: *Script* principal utilizado para guiar o processo, chamando as ferramentas necessárias em cada etapa.

poisson.c: gera uma distribuição de Poisson a partir de sua média. Calcular as probabilidades dos elementos de uma distribuição de Poisson utilizando a equação $\frac{\alpha^i}{i!}e^{\alpha}$ resulta em erro de execução quando a média (α) é grande. Isto ocorre porque esta equação gera valores muito grande, não podendo ser representáveis pelo tipo de dado *double*. As multiplicações da equação foram transformadas em somas com aplicação de logarítmo.

exp.c: gera uma distribuição exponencial a partir de sua média.

pareto.c: gera uma distribuição de Pareto a partir de sua média e intensidade do efeito Noé, representado por α. Este programa utiliza a distribuição de Pareto transladada. Esta separa a intensidade do efeito Noé da média da distribuição.

aproximar.c: reduz uma distribuição. Este programa divide a distribuição em um número de intervalos desejado. Cada intervalo passa a ter um representate. Este representante é a média aritmética dos elementos do intervalo e sua probabilidade é igual a soma das probabilidades dos elementos do intervalo.

conf.c: este programa executa as ações que utilizam apenas um critério fixado pelas características da fonte de tráfego. O programa `conf.c` gera os valores necessários a configuração da fonte utilizando o algoritmo de Kronmal e Peterson alterado, executa o teste de *software* sobre os valores de configuração da fonte e, por fim, salva os resultados em arquivos.

chbase os valores de configuração da fonte são gerados pelos programas acima em base 10. Como os arquivos de simulação VHDL não permitem a entrada de valores em base 10, este *script* é utilizado trocar os valores de configuração da fonte de base 10 para base 2.

sub.awk este *script* substitui as "marcas" do arquivo VHDL pelos valores de configuração da fonte e seus parâmetros. Um exemplo de parâmetro é o número de *bits* utilizados para representar a probabilidade dos elementos da distribuição para o tempo de permanência em um estado (`ts_n_bits_prob`).

O Capítulo seguinte apresenta três estudos de caso de tráfegos gerados pela fonte.

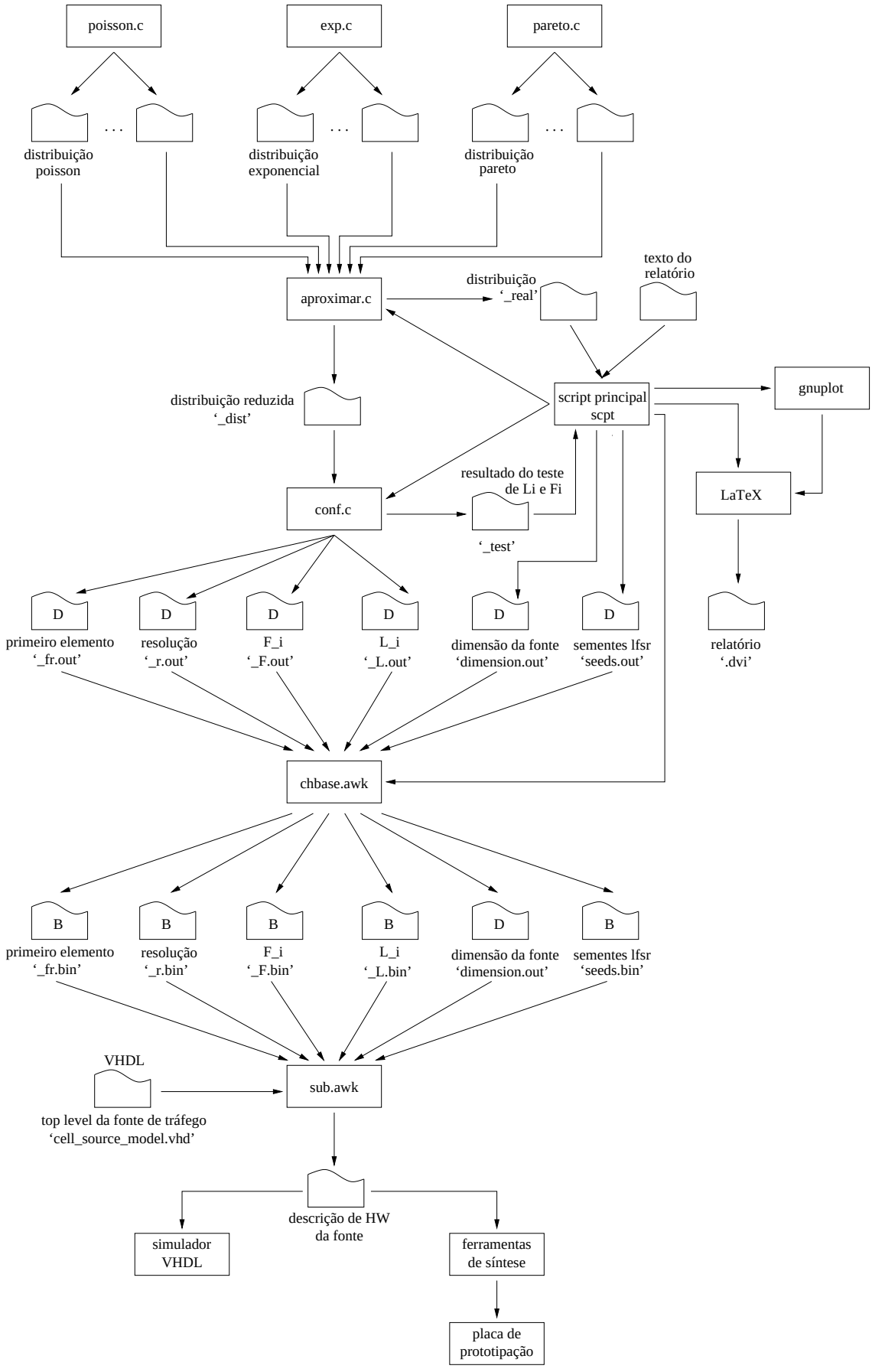


Figura 6.24: Fluxo de execução e programas envolvidos no processo de configuração da fonte de tráfego.

```
# default values - these values can be configured by you to use with option -d

# Voice source - talkspurt = 352ms (12 cells), silence interval = 650ms

n_states=2  # talkspurt and silence interval

celldist[1]=ex1_celldist1 # poisson - average = 29333 * 1us
statetime[1]=ex1_statetime1 # exp     - average = 35200 * 10us
statetrans[1]=ex1_statetrans1 # 2 = 100%
ic_time_unit[1]=25 # 1us
ts_time_unit[1]=250 # 10us

celldist[2]=inactive #celldist2 # inactive or silence
statetime[2]=ex1_statetime2 # poisson - average = 65000 * 10us
statetrans[2]=ex1_statetrans2 # 1 = 100%
ic_time_unit[2]=1 # no cell will be gerenerate
ts_time_unit[2]=250 # 10us

# configure values
ic_n_bits_addr_va=8
ic_n_bits_prob=16

ts_n_bits_addr_va=10
ts_n_bits_prob=16


st_n_bits_addr_va=1
st_n_bits_prob=2
```

Figura 6.25: Exemplo de parametrização do *script* principal (`scpt`). A variável `n_states` indica o número de estados da fonte de tráfego. As variáveis `celldist[`n`]`, `statetime[`n`]` e `statetrans[`n`]` armazenam o nome do arquivo que contém a distribuição para o intervalo entre células do estado n, o tempo de permanência do estado n e a matriz de transição do estado n, respectivamente. As variáveis `ic_n_bits_addr_va` e `ic_n_bits_prob`, indicam o número de *bits* a ser utilizado para representar os elementos e as probabilidades de uma distribuição, respectivamente. Os valores das variávies `ts_n_bits_addr_va` e `ts_n_bits_prob` são o número de *bits* utilizados para representar os elementos e as probabilidades da distribuição para o tempo de permanência nos estados. Da mesma forma, as variáveis `st_n_bits_addr_va` e `st_n_bits_addr_va` indicam o número de *bits* a ser utilizado para representar a matriz de transição entre os estados.

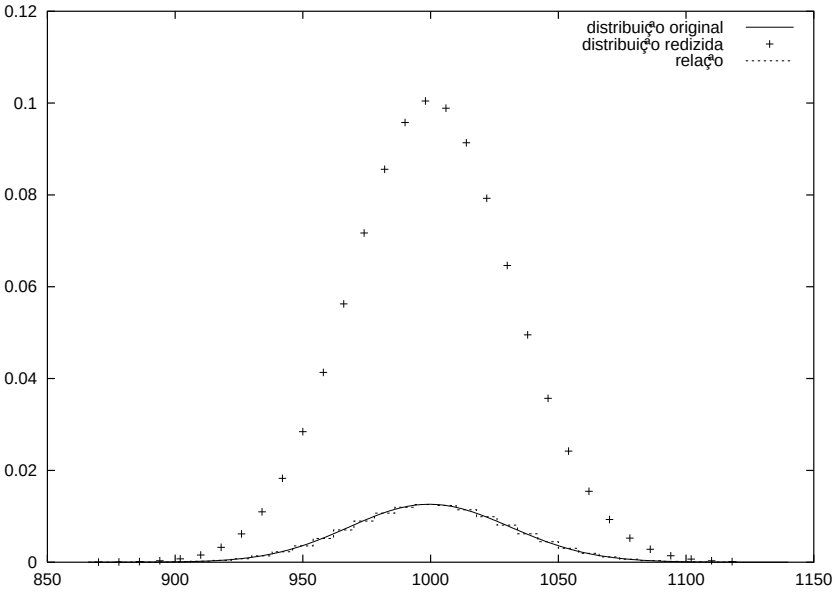


Figura 6.26: Distribuição de Poisson Reduzida. Este gráfico permite comparar a perda de informação quando uma distribuição de Poisson com média 1000 é sujeita a uma ação de redução. A distribuição original de 275 valores foi reduzida para 32 valores. A redução foi exagerada para facilitar a visualização do gráfico. O programa utilizado para redução agrupa uma quantidade de valores consecutivos e utiliza a médias destes valores e de suas probabilidades para gerar um novo valor na distribuição reduzida. O primeiro gráfico é a distribuição original. O segundo é a distribuição reduzida. O terceiro é gerado pelo programa redutor para permitir a visualização da relação entre a distribuição original e a reduzida

Capítulo 7

Estudos de caso

Este Capítulo apresenta alguns exemplos de utilização da fonte de tráfego proposta. Em cada exemplo, os resultados produzidos pelo processo de geração de estímulos são comentados e ilustrados. Neste Capítulo não são apresentados exemplos de medição em um *hardware* ATM real. O objetivo é mostrar que a ferramenta proposta é capaz de conservar o comportamento estatístico do tráfego capturado através do modelo utilizado.

Três exemplos da aplicação da fonte são apresentados. O primeiro, e mais simples, gera células ATM utilizando uma distribuição de Poisson para o intervalo entre células. Este tipo de modelo foi muito utilizado para emular a chegada de células de uma fonte de dados [COS94]. No segundo exemplo, a fonte é utilizada para gerar um padrão de tráfego descrito por um modelo ON-OFF que é atualmente empregado para representar o comportamento de fontes de voz. O último exemplo apresenta o tráfego gerado pela fonte segundo um modelo ON-OFF com tempo de permanência em cada estado distribuído segundo um processo de Pareto. Como dito anteriormente, o objetivo deste Capítulo está restrito apenas a descrever a utilização da fonte na geração de tráfego segundo modelos conhecidos na geração de padrões de tráfego, ou seja, em momento algum será discutida a eficiência do modelo na representação de tráfego. Informações sobre modelos de tráfego podem ser obtidas em [COS94, YUA02, SAH99, ADA97].

7.1 Estudo de caso Poisson

Neste exemplo, a fonte gera células ATM onde o espaço entre elas é distribuído segundo um processo de Poisson. Antes da geração do tráfego é necessário configurar a fonte. O primeiro passo para isso é a geração das distribuições que determinam o tempo de permanência nos estados da fonte e a distância entre as células para cada estado. Neste caso, a fonte terá apenas um estado, e neste estado da fonte, o espaço entre as células estará distribuído segundo um processo de Poisson com média 1000. A saída do programa *poisson.c* é mostrada na Figura 7.1.

O programa da Figura 7.1 acusa que 275 valores significativos foram gerados. Valores significativos neste caso, são aqueles com probabilidade maior que $0,0001\%$. Este valor foi escolhido arbitrariamente. Após a geração da distribuição, é necessário atribuir valores à

```
Written distribution... OK with 275 signficant values
```

Figura 7.1: Saída gerada pelo programa `poisson do exemplo 1`. O programa acusa que 275 valores significativos foram produzidos para uma média igual à 1000.

```
# default values - these values can be configured by you to use with option -d

n_states=1

celldist[1]=ex0_celldist # poisson - average = 1000 clk cycles
ic_time_unit[1]=1 # 40ns

ic_n_bits_addr_va=5
ic_n_bits_prob=16
```

Figura 7.2: Configuração do script do estudo de caso Poisson. A variável `n_state` é usada para informar o número de estados da fonte. A variável `celldist` armazena o nome do arquivo que contém a distribuição para o intervalo entre células. A variável `ic_time_unit` determina a resolução que será usada, neste caso 40ns ($1.1/25MHz$). Por último, as variáveis `ic_n_bits_addr_va` e `ic_n_bits_prob` permitem a configuração do número de *bits* que será utilizado no endereçamento das memórias L_i e F_i e a largura da memória F_i.

variáveis para a utilização do *script* principal (`scpt`) no modo *default*. A Figura 7.2 ilustra a atribuição de valores à variáveis do *script*.

O valores das variáveis de configuração influem diretamente na qualidade do tráfego a ser gerado. Valores muito pequenos para `ic_n_bits_addr_va` ou `ic_n_bits_prob` para uma determinada distribuição podem exigir uma redução exagerada da mesma ou um arredondamento grosseiro na representação das probabilidades em F_i.

O próximo passo é dado com a execução do *script* principal no modo *default*. O processo pode ser separado em duas grandes partes. Na primeira, o *script* principal chama os programas `aproximar.c` e `conf.c`. Estes são necessários para tratar os valores da distribuição, adaptá-los às dimensões do '*hardware* e gerar os valores de configuração da fonte. Na segunda parte, o *script* principal gera o arquivo de sementes (`seeds.out`) e o arquivo de dimensões (`dimensions.out`) e executa os *scripts* `chbase.awk` e `sub.awk` para auxiliar a escrita do arquivo VHDL que descreve a fonte de tráfego parametrizada. A saída produzida pelo *script* é mostrada na Figura 7.3.

Neste caso, o programa `aproximar.c` deve reduzir o número de elementos da distribuição de 275 para 32. A redução é mostrada na Figura 7.4.Apesar da significativa redução, o programa `aproximar.c` avisa que conseguiu reduzir $99,9913\%$ da distribuição, ou seja, a soma das probabilidades para os novos elementos da distribuição é igual a $0,999913$. Após ler o arquivo com a distribuição reduzida (de extensão `_disc`), o programa `conf.c` adapta os valores das probabilidades dos elementos da distribuição para permitir a representação da mesmas na memória com largura de 16 *bits*. Uma largura de 16 *bits* permite uma resolução de até aproximadamente $0,0015\%$. A soma dos erros de cada elemento é mostrado como saída do processo, neste caso, $-0,007629\%$.

Ainda pelo programa `conf.c`, é executado o algoritmo de Kronmal e Peterson adaptado para a geração dos valores das memórias L_i e F_i da fonte, também chamados de valores de configuração. Estes valores são armazenados nos arquivos `ex0_celldist_F.out` e `ex0_celldist_L.out`. Os valores de primeiro elemento e resolução também gerados pelo programa `conf.c` são armazenados nos arquivos `ex0_celldist_fr.out`. Por fim, é executado um teste exaustivo sobre os valores produzidos para verificar a qualidade dos mesmos. O resultado do teste é armazenado no arquivo `ex0_celldist_test.out` para ser utilizado no relatório do processo. Os nomes dos arquivos de saída assumem o mesmo do arquivo que

```
Default mode

-------------------------------------------
STATE 1


Interarrival cell time :

Configuration in use
  Distribution  : ex0_celldist
Buffer size : 32
Number of bits : 16

- Rounding the distribution

Message of aproximar program :
Reading distribution in ex0_celldist file... OK
Selecting signficant range... OK
Summaring distribution... OK with total 99.9913 %

- Creating configuration values

Message of conf program :
Reading distribution in out/ex0_celldist_disc file... OK
Rounding configuration values to 16 bits... OK with error of -0.007629 %
Generating configuration values by Kronman e Peterson...  OK
Saving the configuration values... Calculating and saving first and resolution values... OK
Testing the configuration values and saving result in out/ex0_celldist_test file...  OK

- Adding values to configuration files

- Saving the dimensions of source
- Generating the seeds values

- Translate configuration values to binary

Changing the interarrival cell time values to base 2

- Writting VHDL file

Writting intial values to Fs and Ls memory
Writting diminsion constants of cell source
Do you want report process? <y/n>
Report was generated.
To see this report go to report directory and type latex report.tex
```

Figura 7.3: Saída produzida na execução do *script* principal para o estudo de caso Poisson.

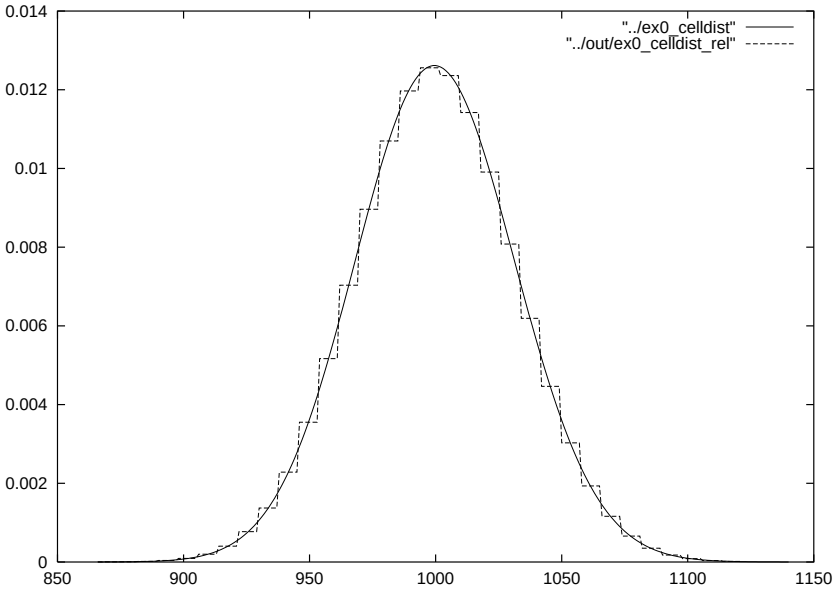


Figura 7.4: Comparação entre a distribuição original e a reduzida do estudo de caso Poisson. Esta Figura é gerada automaticamente pelo *script* principal para ser utilizada no relatório do processo.

contém a distribuição original, trocando apenas o sufixo. A Figura 7.5 compara através de gráficos a distribuição reduzida e o resultado do teste.

O segundo passo, como dito anteriormente, está relacionado com a escrita do arquivo VHDL, chamado `cell_source.vhd`. Primeiro, parâmetros como dimensões da fonte e semente para os geradores de números aleatórios são salvos nos arquivos `dimensions.out` e `seeds.out`, respectivamente, pelo *script* principal. Posteriormente, junto como os valores de configuração da fonte, estes são transformados em números binários pelo *script* `chbase.awk`. O *hardware* descrito em `cell_source_model.vhd` é implementado de forma a ser parametrizado pelo processo. "Marcas"no arquivo `cell_source_model.vhd` indicam o ponto em que os parâmetros devem ser escritos no arquivo. Por último, o *script* principal pergunta ao usuário se ele deseja criar um relatório do processo executado.

Para verificar a eficiência do processo, a descrição VHDL gerada foi simulada. No VHDL foi colocado um processo que será executado apenas para contar quantas vezes um determinado elemento da distribuição é escolhido como intervalo entre duas células consecutivas. O resultado da execução da fonte é mostrado na Figura 7.6.

A Figura 7.7 ilustra o padrão de chegada de células resultante. Sendo a resolução da fonte igual a $40ns$ e a média da distribuição igual a 1000, grande parte dos valores para o espaço entre células deve estar em torno de $40\mu s$.

7.2 Estudo de caso: tráfego de voz

Este estudo de caso mostra a fonte de tráfego sendo utilizada para emular o comportamento real de uma fonte de voz. Uma fonte de voz é representada tipicamente por um modelo IPP, onde a média do tempo de permanência no estado ativo é $352ms$ e o tempo médio de permanência no estado inativo, ou de silêncio, é $650ms$ [LIU99]. No estado ativo em média são gerados 12 pacotes, resultando em uma taxa média de 29333 celulas/segundo.

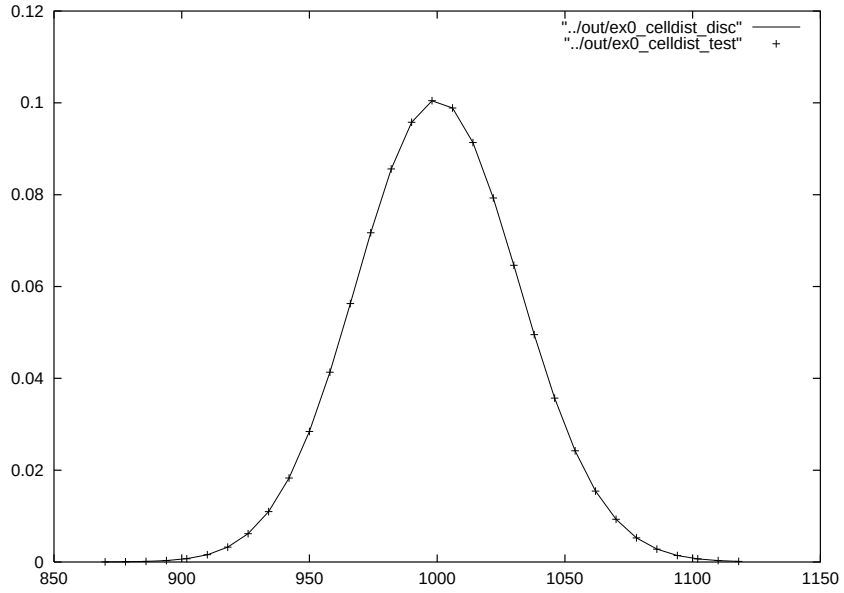


Figura 7.5: Comparação entre a distribuição reduzida e o resultado do teste gerado do estudo de caso Poisson. Esta Figura é gerada automaticamente pelo *script* principal para ser utilizada no relatório do processo.

O primeiro passo é a geração das distribuições que serão utilizadas pela fonte. Neste ponto deve-se considerar a resolução que será empregada. Usar uma resolução muito alta, como em μs (microsegundos) para este caso, exigirá distribuições com um grande número de elementos. Por exemplo, a distribuição de permanência no estado inativo teria 280008 elementos significativos e a distribuição do tempo no estado ativo teria 367531 elementos significativos. Excessivo número de elementos em uma distribuição também gera elementos com probabilidades muito pequenas, o que dificulta os processos de adaptação destes valores às dimensões da fonte. Os resultados gerados pelo processo são claramente inaceitáveis, devido aos erros introduzidos pelos arredondamentos. A Figura 7.8 mostra os gráficos resultantes do teste realizado por *software* para a configuração gerada. Neste caso, foi utilizado um espaço de endereçamento de 10 bits, ou seja, até 1024 elementos. Aumentar o espaço de endereçamento slém deste valor não trouxe uma melhora significativa.

Para diminuir o erro causado pelas distribuições com grande número de elementos, a resolução de trabalho da fonte foi diminuída. Com isso, a fonte perde em precisão, mas torna viável o processo e também diminui o tempo de simulação. Para esta demonstração foi utilizada uma resolução de $10\mu s$ (ic_time_unit[1] = 250) para o tempo de permanência nos estados. Para a distância entre células a resolução de $1\mu s$ foi mantida. As distribuições para o tempo de permanência no estado 1 e 2 foram geradas com médias $35200\mu s$ e $65000\mu s$, respectivamente. Desta forma, o número de elementos passou de 280008 para 117808 elementos para o estado 1 e de 367531 para 177668 elementos para o estado 2. Utilizando o mesmo espaço de endereçamento do caso anterior e o mesmo número de *bits* para representar as probabilidades na memória F_i, obteve-se o resultado mostrado na Figura 7.9.

O intervalo entre células do estado 1 não apresentou nenhum problema. Para uma resolução de $1\mu s$, a distribuição de Poisson resultante apresentou apenas 1349 elementos. O resultado da redução da distribuição e teste de *software* dos resultados gerados pelo processo é ilustrado na Figura 7.10.

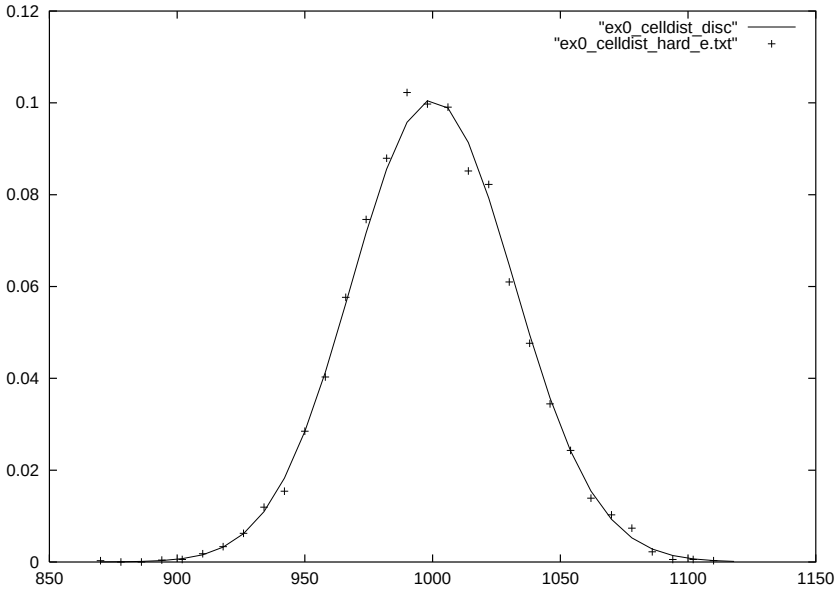


Figura 7.6: Resultado produzido pela execução da fonte de tráfego do estudo de caso Poisson.

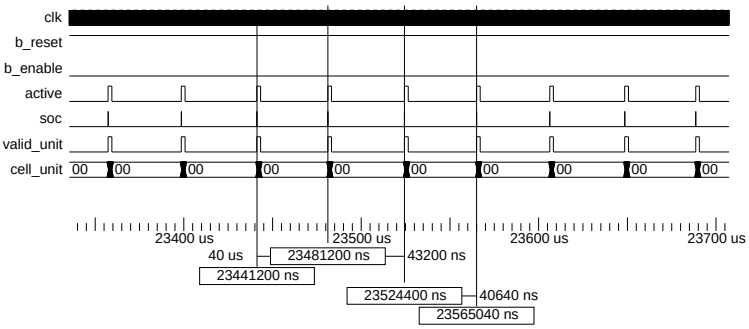


Figura 7.7: Padrão de chegada de células produzido pela fonte de tráfego.

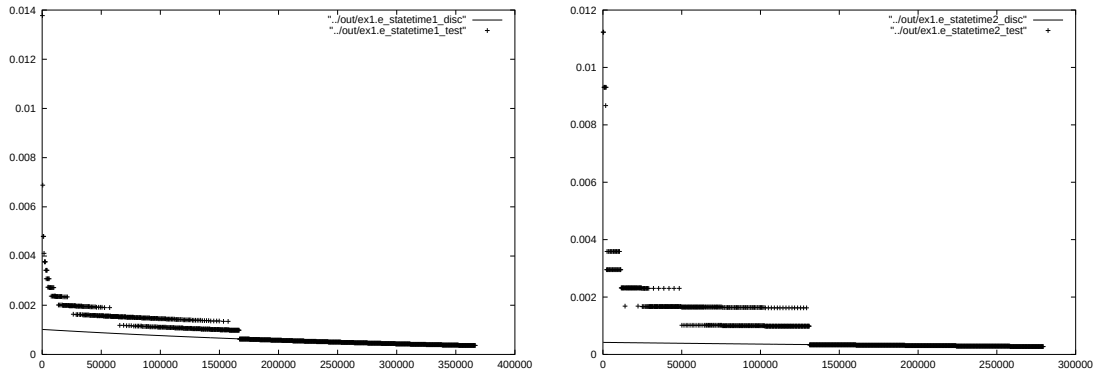
Mesmo tendo diminuído a resolução de trabalho da fonte, o tempo necessário para se ter um número de amostras satisfatórios tornou inviável obtenção de resultados por simulação do *hardware*. Assim, os resultados para o espaçamento entre células no estado 1, para o tempo de permanência no estado 1 e estado 2 não serão mostrados.

Por fim, a Figura 7.11 mostra a medição de um período de ON e OFF na forma de onda resultante da simulação da fonte.

7.3 Estudo de caso: Pareto

Este último estudo de caso demonstra a versatilidade da fonte emulando o comportamento de um processo de Poisson modulado por Pareto. A distribuição de Pareto tende a gerar valores mais afastados da média do que a distribuição exponencial. Devido a esta característica, algum esforço vem sendo aplicado para verificar a eficiência de fontes ON-OFF moduladas por Pareto na geração de tráfego auto-similar.

Apesar da importância da distribuição de Pareto na geração de tráfego auto-similar [SIL00, CHA01] esta Seção não tem a intenção de provar a eficiência do mesma nesta atividade. O que



(a) Tempo de permanência no estado 1. (b) Tempo de permanência no estado 2.

Figura 7.8: Resultado produzido pelo teste de *software* com o uso de grandes distribuições no estudo de caso fonte de voz. O gráfico 7.8(a) mostra o erro para o tempo de permanência no estado 1, enquanto o gráfico 7.8(b) ilustra o erro para o tempo de permanência no estado 2.

Tabela 7.1: Distribuição de Pareto. Fonte [SIL00].

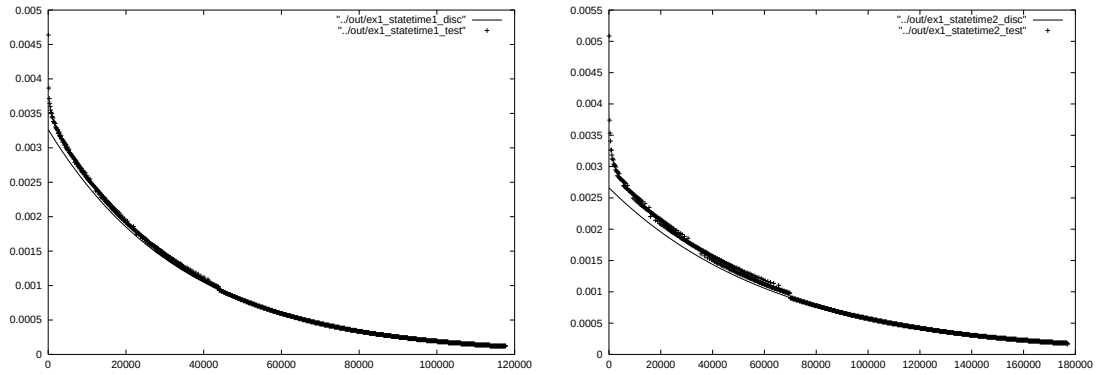
Parâmetro	α, $(\alpha > 0)$
Faixa	$1 \leq x \leq \inf$
fdp	$f(x) = 1 - x^{-\alpha}$
Média	$\alpha/(\alpha - 1)$, $\alpha > 1$

se pretende é mostrar a flexibilidade do processo de configuração e execução da fonte diante de uma nova distribuição. Para incluir uma nova distribuição no processo de configuração da fonte, foi preciso apenas gerar os valores segundo esta distribuição. O mesmo programa utilizado para reduzir o tamanho das distribuições de Poisson e exponencial foi usado para reduzir a distribuição de Pareto. O tempo de permanência no estado ativo, ou ON, segue uma distribuição de Pareto de média 2000. Utilizando um *clock* de 25MHz e um resolução 25 (ts_time_unit[1]=25), pretende-se gerar períodos de ON com um tempo médio de $2000\mu s$. Durante o período de ON as células estarão espaçadas segundo uma distribuição de Poisson com valor médio de $50\mu s$. O período de OFF também é determinado por uma distribuição de Pareto. Neste caso a média é de $6500\mu s$.

Em grande parte dos estudos para a geração de tráfego auto-similar utilizando a distribuição de Pareto, o interesse é determinar como os resultados são afetados pela mudança na intensidade do efeito de Noé para uma determinada média. A distribuição de Pareto simples, mostrada na Tabela 7.1, não permite a realização desse estudo, isto por que sua média varia de acordo com a intensidade de Noé.

Este problema pode ser resolvido utilizando-se uma distribuição de Pareto transladada. Esta permite que a associação de valores para a média e para o efeito de Noé, representado pelo parâmetro α, sejam feitos de forma independente. A distribuição de Pareto transladada é mostrada na Tabela 7.2.

O valor de θ pode ser obtido a partir da média, fazendo $\theta = \mu(\alpha - 1)$. Neste exemplo, o valor de α é 1.2 para ambas distribuições de Pareto utilizadas. Assim, gerando a distribuições



(a) Tempo de permanência no estado 1. (b) Tempo de permanência no estado 2.

Figura 7.9: Resultado produzido pelo teste de *software* com a redução da resolução no estudo de caso de fonte de voz. O gráfico 7.9(a) mostra o resultado do teste de *software* para o tempo de permanência no estado 1, enquanto o gráfico 7.9(b) mostra o resultado do mesmo teste para a o tempo de permanência no estado 2

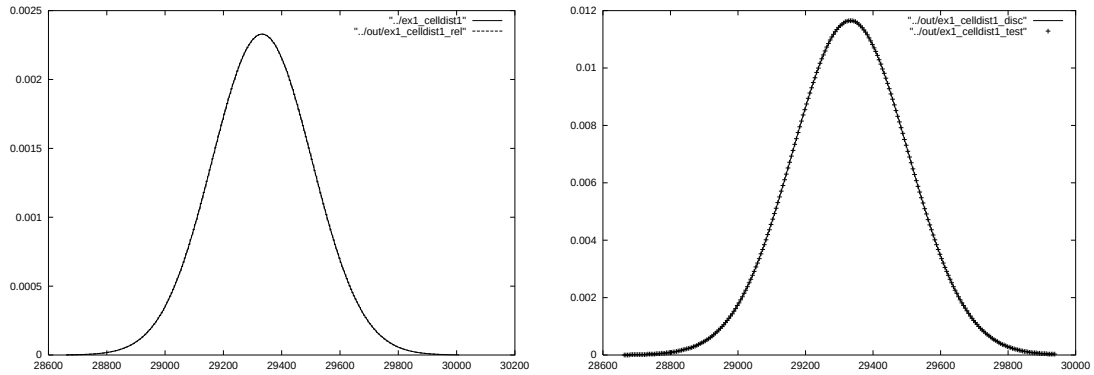
Tabela 7.2: Distribuição de Pareto transladada. Fonte [SIL00].

Parâmetro	α, $(\alpha > 0)$; θ
Faixa	$1 \leq x \leq \inf$
fdp	$f(x) = (\alpha/\theta)(\theta/x + \theta)^{\alpha} + 1$
Média	$\theta/(\alpha/1)$, $\alpha > 1$

necessárias e executando o processo a partir do *script* principal os seguintes resultados foram obtidos. As Figuras 7.12, 7.13 e 7.14 mostram os resultados do teste de *software* e *hardware* obtidos para a distância entre chegada de células, para o tempo de permanência no estado 1 e para o tempo de permanência no estado 2, respectivamente.

Por fim, a Figura 7.15 ilustra o resultado obtido pela simulação da fonte. Esta Figura exemplifica, de forma simples, o comportamento de um tráfego gerado por um PMPP.

O modo de implementação do ferramental de configuração da fonte tornou o processo fácil e rápido de ser executado. A combinação de vários programas feita através de um *script* tornou o processo transparente, flexível e escalável. Sabe-se que os exemplos apresentados acima não comprovam que a fonte de tráfego pode ser aplicada na maioria dos casos de medição de desempenho de sistemas ATM reais. Contudo, a valorização dada à flexibilidade durante a fase de projeto e o ferramental de configuração da fonte de tráfego desenvolvido possibilita a execução de de testes a uma baixo custo, considerando os fatores tempo e recursos.



(a) Redução da distribuição. (b) Teste de *software*.

Figura 7.10: Resultado da redução e teste de *software* para da distribuição do tempo entre células no estudo de caso fonte de voz. O gráfico 7.10(a) mostra a redução da distribuição original e o gráfico 7.10(b) mostra o resultado obtido pelo teste de *software* sobre os valores de configuração gerados.

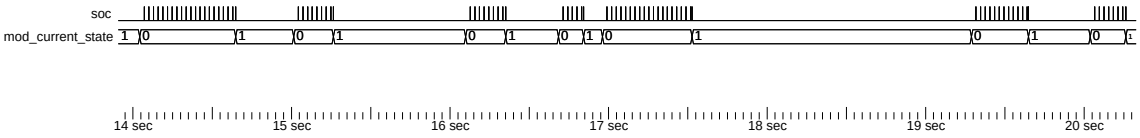
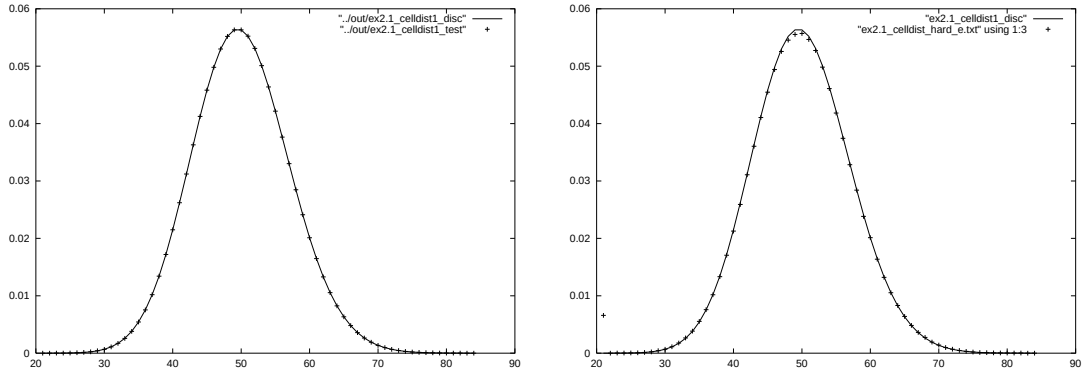
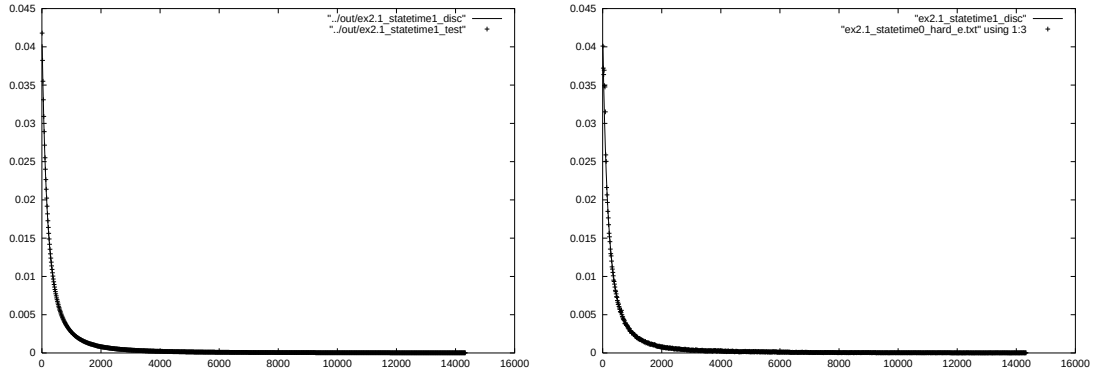


Figura 7.11: Medição do período de ON e OFF no estudo de caso fonte de voz.



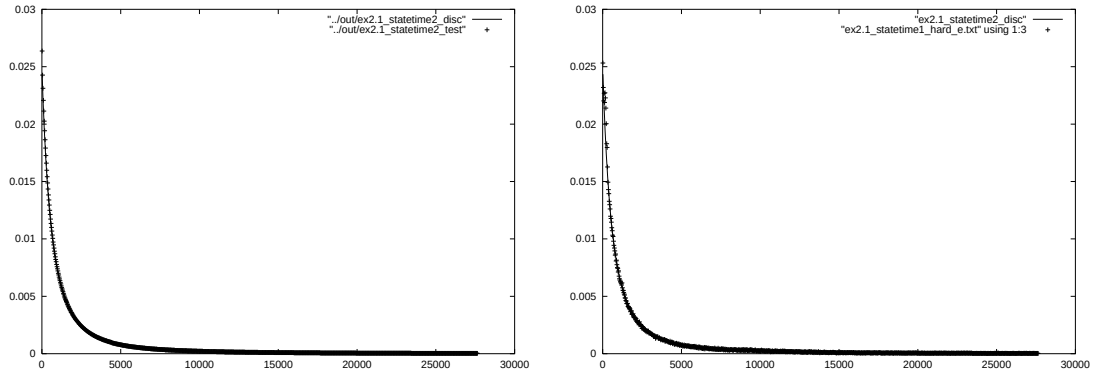
(a) Teste de *software*. (b) Simulação do *hardware*.

Figura 7.12: Resultados produzidos pelo processo de medição no *hardware* para a distância entre células do estado 1 do estudo de caso Pareto. O gráfico 7.12(a) mostra o resultado realizado por *software*, enquanto o gráfico 7.12(b) mostra o resultado de simulação do *hardware*.



(a) Teste de *software*. (b) Simulação do *hardware*.

Figura 7.13: Resultados produzidos pelo processo de medição no *hardware* para o estado 1 do estudo de caso Pareto. O gráfico 7.13(a) mostra o resultado realizado por *software*, enquanto o gráfico 7.13(b) mostra o resultado de simulação do *hardware*.



(a) Teste de *software*. (b) Simulação do *hardware*.

Figura 7.14: Resultados produzidos pelo processo de medição no *hardware* para o estado 2 do estudo de caso Pareto. O gráfico 7.14(a) mostra o resultado realizado por *software*, enquanto o gráfico 7.14(b) mostra o resultado de simulação do *hardware*.

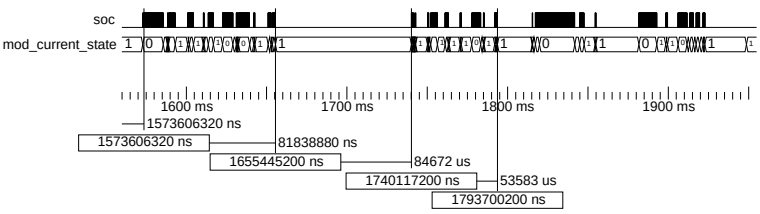


Figura 7.15: Padrão de tráfego gerado no estudo de caso Pareto.

Capítulo 8

Conclusões e Trabalhos Futuros

Durante os estudos realizados para o desenvolvimento deste trabalho, constatou-se que as ferramentas para medição de desempenho de redes ATM encontram-se maduras. Hoje, a classe de ferramentas mais popular é a dos simuladores, devido à facilidade de obtenção e aplicação. Contudo, mesmo para uma medição de desempenho em alto nível, estes se deparam com o problema crítico do custo de execução, o que os torna inadequados para simulação de sistemas complexos.

Uma forma de acelerar o processo de simulação é executá-lo em *hardware*. Contudo, é necessário um conjunto de ferramentas que torne o processo simples e transparente ao usuário e uma plataforma popular para aumentar a sua disponibilidade. Para o caso particular de redes ATM, o uso de *hardware* para medir o desempenho, além de acelerar o processo de avaliação, ainda permite uma verificação funcional da descrição do *hardware* ATM e diminui o tempo de desenvolvimento de *testbenches*. Isto é muito útil, uma vez que, devido a restrições de desempenho, grande parte dos módulos ATM deve ser implementados em *hardware*.

Este trabalho propôs-se a estudar formas para medir o desempenho de módulos de *hardware* ATM. Optou-se pela utilização de plataformas de prototipação e linguagens de descrição de *hardware* como forma de diminuir o problema de disponibilidade e flexibilidade apresentado por outras propostas. A implementação do método de medida descrito no Capítulo 6 permitiu que se tivesse uma visão mais precisa das dificuldades, vantagens e desvantagens da utilização de *hardware* na medição de desempenho.

Notou-se durante a execução de alguns testes da fonte que o uso de simuladores HDLs é inviável na obtenção de qualquer tipo de medida em um caso real. O estudo de caso com tráfego de voz, descrito na Seção 7.2, mostra a fonte sendo usada para emular o comportamento de uma fonte de voz real. Os resultados coletados após dois dias de simulação não permitiram qualquer conclusão sobre o comportamento estatístico da fonte, devido ao pequeno número de amostras de intervalo entre células e tempo de permanência nos estados ON e OFF. Esta observação é natural, já que simuladores em níveis mais abstratos também apresentam este problema. Contudo, a utilização dos simuladores HDLs são peças fundamentais no processo, permitindo ao usuário uma fácil verificação da ligação dos sinais, bem como a avaliação da funcionalidade do hardware.

O principal problema a ser resolvido na utilização de plataformas de prototipação é a quantidade limitada de memória. Este recurso deverá ser repartido entre o módulo que está sendo testado e os módulos introduzidos para a medição. A quantidade de memória disponível influi diretamente na qualidade da medição. Apesar das dificuldades oferecidas pelo ambiente de prototipação escolhido, este trabalho apresenta uma série de contribuições.

A primeira é o projeto e implementação de uma camada ATM parametrizável para a re-

cepção. Esta implementação suporta um número parametrizável de interfaces com as camadas superiores. Se este número é menor do que 256, a implementação ainda permite mais de uma camada superior por interface. O usuário pode selecionar uma gama ampla de compromissos entre estes valores.

A principal contribuição é a especificação, projeto e implementação em VHDL de uma fonte de tráfego parametrizável. Esta fonte de tráfego é capaz de suportar diferentes tipos de processos modulados. A geração das células é feita por *hardware*, permitindo a medição de desempenho de módulos ATM com altas taxas de transferência. Também foi desenvolvido um ferramental flexível e escalável para a configuração da fonte de tráfego. O complexo processo de personalização da fonte para diversas situações de tráfego foi facilitado pela implementação de *software* sob a forma de programas e *scripts* que guiam a geração dos arquivos de parametrização. A capacidade de expansão do ambiente para suportar outros modelos não diretamente disponíveis na implementação inicial é garantida pela exposição da estrutura interna de implementação do método de medida proposto em geral e pela estrutura da fonte em particular.

Por fim, a alteração do algoritmo de Kronmal e Peterson foi mais uma contribuição deste trabalho. O algoritmo original causa erros de execução quando a soma das probabilidades não é igual a 1. A condição de parada do algoritmo permite a utilização de índices de vetores menores do que 0. A alteração feita utiliza o próprio erro na soma das probabilidades para evitar o erro de execução do algoritmo.

No que diz respeito a trabalhos futuros, a utilização de plataformas de prototipação como forma de diminuir o custo de simulação de processos está vinculada a um conjunto de ferramentas dependente da plataforma que simplifique o uso da mesma. Desta forma, cria-se uma camada de abstração sobre a plataforma, facilitando o desenvolvimento de ferramentas específicas a um determinado processo, como por exemplo a medição de desempenho em redes ATM. A validação e a viabilidade de aplicação da presente proposta dependem de uma série de atividades a serem executadas.

Apesar dos testes realizados para o presente trabalho terem apresentado bons resultados, ainda é necessário verificar a eficiência da fonte na representação de tráfegos reais. Para isso, é preciso que a fonte seja capaz de executar modelos de tráfegos parametrizados, gerando-os com um comportamento estatístico próximo da realidade.

Para completar o ferramental proposto, é necessário projetar e implementar os módulos de *hardware* para a coleta de informações relevantes a avaliação de desempenho. O projeto destes módulos irá depender dos parâmetros de desempenho que se pretende medir. Somente com a utilização dos mesmos será possível comprovar a viabilidade de aplicação de plataformas de prototipação na avaliação de desempenho de módulos de *hardware* ATM. Para isso, é necessário uma descrição VHDL sintetizável dos módulos de *hardware* envolvidos. Para transformar a descrição da fonte de tráfego em um módulo VHDL é necessário substituir os vetores utilizados como memória por blocos de memória RAM. Vetores foram utilizados no lugar de blocos de memória real simplesmente para facilitar a etapa de parametrização da fonte de tráfego pelas ferramentas de configuração.

Por fim, é necessário alterar a descrição VHDL da fonte de tráfego, colocando os valores parametrizáveis em memória para que a mesma possa valer-se de reconfiguração parcial. O uso desta funcionalidade, disponível em alguns FPGAs atuais, permite uma diminuição significativa do tempo de compilação, pois evita uma nova síntese completa do *hardware*.

Durante o estudo realizado percebeu-se que a utilização de *hardware* na avaliação de desempenho de sistemas ATM, apesar de suas vantagens, não é simples de ser aplicada. Os motivos identificados são a dificuldade de acesso ao *hardware* necessário, a falta de flexibili-

dade e a dificuldade de aplicação do mesmo. Assim, o presente trabalho não torna o uso de módulos de *hardware* a melhor forma para se medir o desempenho de sistemas ATM, mas procura viabilizar tal processo utilizando uma plataforma comum aos projetistas para diminuir as limitações encontradas atualmente.

Apêndice A

Programa `poisson.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int poisson(int average, FILE *f)
{
int j=0, k=0;
double prob, acum_fat=0;

acum_fat = 0;
prob = exp(-average+j*log(average)-acum_fat);
while ((prob > 0.000001) || (j<=average))
{
if (prob > 0.000001)
{
fprintf(f, "%d %e\n", j, prob);
k++;
}
j++;
acum_fat+= log((double)j);
prob = exp(-average+j*log(average)-acum_fat);
}
return k;
}

int main(int argc, char *argv[])
{
FILE *f;
int average, sv;
char filename[40];

if (argc < 2)
{
printf("There are insufficient arguments\n");
```

```
return 1;
}

average = atoi(argv[1]);
sprintf(filename, argv[2]);

f = fopen(filename, "w");
if (f == NULL)
{
printf("Error opening file\n");
return 1;
}

printf("Writing distribution... ");

if ((sv = poisson(average, f)) == 0)
{
printf("FAIL\n");
return 1;
}
printf("OK with %d significant values\n", sv);
fclose(f);
return 0;
}
```

Apêndice B

Programa exp.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int expon(int average, FILE *f)
{
int j=1, k=0;
double prob;

printf("media = %d\n", average);
prob = (1.0/average) * exp(-1.0*j/average);
while ((prob > 0.000001) || (j<=average))
{
if (prob > 0.000001)
{
fprintf(f, "%d %e\n", j, prob);
k++;
}
j++;
prob = (1.0/average) * exp(-1.0*j/average);
}
return k;
}

int main(int argc, char *argv[])
{
FILE *f;
int average, sv;
char filename[40];

if (argc < 2)
{
printf("There are insufficient arguments\n");
return 1;
}
```

```
average = atoi(argv[1]);
sprintf(filename, argv[2]);

f = fopen(filename, "w");
if (f == NULL)
{
printf("Error opening file\n");
return 1;
}

printf("Writing distribution... ");

if ((sv = expon(average, f)) == 0)
{
printf("FAIL\n");
return 1;
}
printf("OK with %d significant values\n", sv);
fclose(f);
return 0;
}
```

Apêndice C

Programa `pareto.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int pareto(int average, double alpha, FILE *f)
{
int j=1, k=0;
double prob, teta;

teta = 1.0 * average * (alpha - 1.0);
prob = (alpha/teta) * pow(teta/(j + teta), alpha+1);
while ((prob > 0.000001) || (j<=average))
{
if (prob > 0.000001)
{
fprintf(f, "%d %e\n", j, prob);
k++;
}
j++;
prob = (alpha/teta) * pow(teta/(j + teta), alpha+1);
}
return k;
}

int main(int argc, char *argv[])
{
FILE *f;
int average, sv;
double alpha;
char filename[40];

if (argc < 2)
{
printf("There are insufficient arguments\n");
printf("The arguments are average, alpha, and filename\n");
```

```
return 1;
}

average = atoi(argv[1]);
alpha = atof(argv[2]);
sprintf(filename, argv[3]);

f = fopen(filename, "w");
if (f == NULL)
{
printf("Error open file\n");
return 1;
}

printf("Written distribution... ");

if ((sv = pareto(average, alpha, f)) == 0)
{
printf("FAIL\n");
return 1;
}
printf("OK with %d significant values\n", sv);
fclose(f);
return 0;
}
```

Referências Bibliográficas

[ADA97] ADAS, Abdelnaser. Traffic models in broadband networks. **IEEE Communications Magazine**, July 1997.

[ADU02] ADUBA, Chukwuemeka N.; SADIKU, Matthew N. O. Simulation and analysis of different traffic models for ATM networks. In: IEEE SOUTHEASTCON 2002, 2002. **Anais...** 2002.

[ALT01] ALTERA, Inc. **Signaltap embedded logic analyzer megafunction**. [S.l.: s.n.], 2001. (Data Sheet, Version 2.0).

[BER98] BERGER, A.W.; WHITT, W. Extending the effective bandwidth concept to networks with priority classes. **IEEE Communications Magazine**, August 1998.

[BRE00] BRELET, Jean-Louis. **Application note**: virtex series. [S.l.: s.n.], 2000. (XAPP204 (v1.2)).

[BUF94] BUFFET, E.; DUFFIELD, N.G. Exponential upper bounds via martingales for multiplexers with markiovian arrivals. **Journal of Applied Probability**, v.31, 1994.

[CAS01] CASTANHEIRA, Leonardo Dutra. Redes ATM: um estudo de algoritmos de gerenciamento de tráfego e qualidade de serviço. [S.l.: s.n.], 2001. (Pontifícia Universidade Católica do Rio Grande do Sul/Faculdade de Informática - PUCRS/FACIN, Porto Alegre, RS, Trabalho Individual I).

[CHA01] CHANDRA, K. Sarath; SHARMA, Manuj; GURU, D.S. Loss and delay behavior of an ATM multiplexer for aggregated pmpp (pareto-modulated poisson process) traffic sources. In: NINTH IEEE INTERNATIONAL CONFERENCE ON NETWORKS, 2001. **Anais...** 2001. p.344–348.

[COM95] COMMITEE, The ATM Forum Technical. **Utopia specification level 2**. [S.l.: s.n.], 1995.

[COM01] COMMITTE, The ATM Forum Techinical. **ATM technology, the foundation for broadband networks**. Disponível por WWW em www.atmforum.com/standards/approved.html (July 2001).

[COM99] COMMITTEE, The ATM Forum Technical. **Traffic management specification version 4.1**. [S.l.: s.n.], 1999.

[COM02] COMMITTEE, The ATM Forum Technical. **ATM user network interface(UNI) signalling specification version 4.1**. [S.l.: s.n.], 2002.

[COM02a] COMMITTEE, The ATM Forum Technical. **Private network-network interface specification v.1.1**. [S.l.: s.n.], 2002.

[COS94] COSMAS, John P.; PETIT, Guido H; LEHNERT, Ralf; BLONDIA, Chris; KONTOVASSILIS, Kimon; CASALS, Olga; THEIMER, Thomas. A review of voice, data and video traffic models for ATM. **European Transactions on Telecommunications**, v.5, n.2, April 1994.

[DIE00] DIEMER, Mouriac Halen. **DTA - discriminador de tráfego ATM**. [S.l.: s.n.], 2000. Dissertação de Mestrado.

[DUT95] DUTTON, Harry J. R.; LENHARD, Peter. **Asynchronous transfer mode (ATM)**: technical overview. New Jersey, USA: Prentice Hall PTR, 1995. (2nd ed.).

[ELW91] ELWALID, A.; MITRA, D.; WENTWORTH, R. H. A new approach for allocating buffers and bandwidth to heterogeneous, regulated traffic in an ATM node. **IEEE Journal on Selected Areas in Communications**, v.9, n.7, September 1991.

[FIL96] FILHO, Jorge Roberto Mendes; MORAES, Luís Felipe Magalhães de. **Modelos de fontes de tráfego para RDSI-FL**. [S.l.: s.n.], 1996.

[GIR98] GIROUX, Natalie; GANTI, Sudhakar. **Quality of service in atm networks**. [S.l.]: Prentice Hall, 1998.

[HON98] HONTAS, S.; TSELIKIS, G.; TOMPROS, S.; GIAMNIADAKIS, J.; ANDRITSOS, S.; LOUKATOS, D.; MITROU, N. ATM traffic generator card. an integrated solution. In: THIRD IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS, ISCC'98, 1998. **Anais...** 1998. p.161–165.

[ITU01] ITU-T. **Iternational telecommunications union**. Disponível por WWW em www.itu-t.com (July 2001).

[KAS01] KASAHARA, Shoji. Inernet traffic modeling: markovian approach to self-similar traffic and prediction of loss probability for finite queues. **IEICE Transaction Communication**, v.E84-B, n.8, August 2001.

[KAS01a] KASAHARA, Shoji. Internet traffic modeling: markovian approach to self-similar traffic and prediction of loss probability for finite queues. **IEICE Transactions Communication**, v.E84-B, n.8, August 2001.

[KEI95] KEISER, Gerd; FREEMAN, David; CARTER, Carrie. ATM test traffic generation algorithms. In: FOURTH INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS AND NETWORKS, 1995. **Anais...** 1995. p.462–469.

[KEL91] KELLY, F.P. Effective bandwidths at multi-class queues. **Queuing Systems**, v.9, 1991.

[KEL91a] KELTON, D.; LAW, A. **Simulation modeling and analysis**. USA: McGraw-Hill, 1991.

[KUH95] KUHN, Peter; STEFAN ECKART, Werner Bachhuber ans; HANSSON, Einar; KINDSMüLLER, Peter. A multiprotocol ATM/AAL network interface chip. In: PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON BROADBAND ISLANDS, 1995, Dublin, Ireland. **Anais...** 1995. ([Disp. em http://www.lis.e-technik.tu-muenchen.de/research/bv/e_publ.html]).

[LAW91] LAW, Averill M.; LELTON, W. David. **Simulation modeling & analysis**. [S.l.]: McGraw-Hill, 1991. (2nd ed.).

[LEL93] LELAND, W. E.; TAQQU, M. S.; WILINGER, W.; WILSON, D. V. On the self-similar nature of ethernet traffic. In: SIGCOMM'93, 1993. **Anais...** 1993. v.23, p.202–213.

[LIU99] LIU, Chunlei; MUNIR, Sohail; JAIN, Raj. Packing Density of Voice Trunking Using AAL2. In: IEEE GLOBAL TELECOMMUNICATIONS CONFERENCE, 1999. **Anais...** 1999. p.611–615.

[PAC98] PACKARD, Hewlett. **HP test & measurement 1999 catalog**. [S.l.: s.n.], 1998.

[POS98] POST, Guido; MüLLER, Andrea; GRöTKER, Thorsten. A system-level co-verification environment for ATM hardware design. In: DESIGN, AUTOMATION AND TEST IN EUROPE, 1998. **Anais...** 1998. p.424–428.

[ROB96] ROBERT, Stephan; LE BOUDEC, Jean-Y ves. On a markov modulated chain exhibiting self-similarities over finite timescale. **Performance Evaluation**, p.159–173, 1996.

[ROB97] ROBERT, Stephan; LE BOUDEC, Jean-Y ves. New models for pseudo self-similar traffic. **Performance Evaluation**, p.57–68, 1997.

[ROB96] ROBERTS, J.; MOCCI, U.; VIRTAMO, J. **Broadband network teletraffic**. Berlin: Heidelberg: Springer, 1996.

[SAH99] SAHINOGLU, Zafer; TEKINAY, Sirin. On multimedia networks: self-similar traffic and network performance. **IEEE Comunications Magazine**, v.37, n.1, January 1999.

[SIL00] SILVA, Fernanda Mendes Ascenção. **Multiplexação estatística de fontes heterogêneas em ATM usando o modelo on/off pareto**. [S.l.: s.n.], 2000. Dissertação de Mestrado.

[SON97] SONG, Hyjeong; JACOB, Lillykutty; KIM, Hyun-Gon; KWON, Boseob; CHUNG, Jai-Hoon; YOON, Hyunsoo. A simple and fast scheduler for input queued ATM switches. In: HIGH PERFORMANCE COMPUTING ON THE INFORMATION SUPERHIGHWAY, HPC ASIA'97, 1997. **Anais...** 1997. p.260–265.

[SOU02] SOUZA, Sheila Moreira; CASTANHEIRA, Leonardo Dutra; CALAZANS, Ney Laert Vilar; MORAES, Fernando Gehn. Design and prototyping of a utopia level 2 interface controller IP soft core. [S.l.: s.n.], 2002. (Pontifícia Universidade Católica do Rio Grande do Sul/Faculdade de Informática - PUCRS/FACIN, Porto Alegre, RS, Relatório Tecnico (em preparação)).

[SOU01] SOUZA, Sheila Moreira. Redes ATM: um estudo das camadas de protocolo e suas interfaces. [S.l.: s.n.], 2001. (Pontifícia Universidade Católica do Rio Grande do Sul/Faculdade de Informática - PUCRS/FACIN, Porto Alegre, RS, Trabalho Individual I).

[SOU02] SOUZA, Sheila Moreira. **Camadas de adaptação ATM para transmissão de dados e voz.** Porto Alegre, RS: Pontifícia Universidade Católica do Rio Grande do Sul, 2002. Dissertação de Mestrado.

[TAQ97] TAQQU, Murad; WILLINGER, Walter; SHERMAN, R. Proof of a fundamental result in self-similar traffic modeliing. **Computer Communications Review**, April 1997.

[TSO00] TSOU, Fu-Ming; CHIOU, Hong-Bin; TSAI, Zsehong. Design and simulation of an efficient real-time traffic scheduler with jitter and delay guarantees. **IEEE Transactions on Multimedia**, v.2, n.4, December 2000.

[TSO00a] TSOU, Fu-Ming; TSAI, Zsehong. A traffic shaper for supporting CBR and VBR service in ATM networks. **Performance Evaluation**, v.42, n.4, November 2000.

[UNI91] UNION, International Telecommunication. **B-isdn protocol reference model and its application**. [S.l.: s.n.], 1991. (Recommendation I.321).

[UNI93] UNION, International Telecommunication. **B-isdn meta-signalling protocol**. [S.l.: s.n.], 1993. (Recommendation Q.2120).

[UNI96] UNION, International Telecommunication. **B-isdn general network aspects**. [S.l.: s.n.], 1996. (Recommendation I.311).

[UNI96a] UNION, International Telecommunication. **Reference events for defining isdn and b-isdn performance parameters**. [S.l.: s.n.], 1996. (Recommendation I.353).

[UNI97] UNION, International Telecommunication. **Vocabulary of terms for broadband aspects of isdn**. [S.l.: s.n.], 1997. (Recommendation I.113).

[UNI99] UNION, International Telecommunication. **B-ISDN operation and maintenance principles and functions**. [S.l.: s.n.], 1999. (Recommendation I.610).

[UNI00] UNION, International Telecommunication. **B-ISDN ATM layer cell transfer performance**. [S.l.: s.n.], 2000. (Recommendation I.356).

[UNI00a] UNION, International Telecommunication. **Traffic control and congestion control in b-isdn**. [S.l.: s.n.], 2000. (Recommendation I.371).

[WIL97] WILLINGER, Walter; A; B; C; D; E; F. Self-similarity through high variability. **IEEE/ACM Transactions on Networking**, New York, v.5, n.1, p.71–86, February 1997.

[XIL01] XILINX, Inc. **Chipscope software and ILA cores user manual**. [S.l.: s.n.], 2001. (Manual, Version 4.1).

[YUA02] YUAN, Xiaohong; ILYAS, Mohammad. Modeling of traffic sources in ATM networks. In: IEEE SOUTHEASTCON2002, 2002. **Anais...** 2002.