Um Ambiente de Simulação de Fluido com Aplicações a Redes Multimídia

por

Kelvin de Freitas Reinhardt

UFRJ

Tese submetida para a obtenção do título de

Mestre em Ciências em Engenharia de Sistemas e Computação

ao Programa de Pós-Graduação de Engenharia de Sistemas e Computação

da COPPE/UFRJ

por

Kelvin de Freitas Reinhardt

Julho 2002

Um Ambiente de Simulação de Fluido com Aplicações a Redes Multimídia

Kelvin de Freitas Reinhardt

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Edmundo Albuquerque de Souza e Silva, Ph.D.

Prof. Valmir Carneiro Barbosa, Ph.D.

Dr. Jorge Moreira de Souza, Dr.

RIO DE JANEIRO, RJ - BRASIL

JULHO DE 2002

REINHARDT, KELVIN DE FREITAS

Um Ambiente de Simulação de Fluido com Aplicações a Redes Multimídia [Rio de Janeiro] 2002

XIX, 135 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2002)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Simulação de Fluido
2. Análise de Desempenho
3. Redes Multimídia
4. Ferramenta TANGRAM-II

I. COPPE/UFRJ II. Título (Série)

Dedico este trabalho aos meus familiares,

em especial, aos meus pais Lizarb e Inarcy.

Agradecimentos

Em primeiro lugar, gostaria de agradecer a Deus pelo fato de nossa existência, e por tão perfeita obra, na qual estamos inseridos. Gostaria de agradecer profundamente aos meus familiares, por tudo o que me ensinaram. Hoje vejo que simples palavras não descrevem a complexidade das ações, que por muitas vezes eram incompreendidas, mas que com certeza me levaram ao caminho certo. Com o passar do tempo, minha compreensão permite-me enxergar mais longe e acumulo cada vez mais um sentimento de admiração por eles.

Para citar meus mestres e colegas, gostaria de contar um breve estória: Certa vez um de meus mestres, Valmir Barbosa, parabenizou-me pelo bom desempenho que tive em uma das disciplinas. Minha resposta na época foi: "Quando o professor é bom, a tarefa de se obter um bom desempenho fica bastante facilitada". Com o tempo percebi que raciocínio estava na verdade incompleto. Hoje eu diria: "Quando os mestres são bons, assim como os alunos que os cercam, a tarefa de se obter um bom desempenho fica bastante facilitada". Por isso eu gostaria de agradecer aos professores e colegas, que com suas qualidades excepcionais, apontaram-me e ajudaram-me a alcançar o caminho correto, o caminho do conhecimento, da verdade, da ciência. Ao lado deles, o mundo pareceu um pouco menos obscuro. Em especial gostaria de citar os mestres Edmundo e Rosa, por todo seu esfoço e dedicação ao trabalho de pesquisa em grupo e por tudo que fizeram por mim. Gostaria de agradecer profundamente a Adriane Cardozo, que foi muito mais do que colega, sendo uma amiga de valor inestimável. Agradeço também aos pais dela, Nilta e Narciso, bem com a todo o pessoal do LAND que direta ou indiretamente me ajudaram no desenvolvimento desta tese: Flávio, Ana Paula, Magnos, Ratton, Sidney, Sadoc, Allyson, GD e em especial ao Bruno, que dedicou muito de seu tempo em prol desta obra. Por fim gostaria de lembrar algumas pessoas que fizeram parte da minha vida nesta época, como a Marluce, Ana Cristina, Rodrigo, César, Paulo André, Roberta e Augusto.

Agradeço ao Governo Federal do meu adorado Brasil, em especial à CAPES, pela bolsa de estudos que recebi. Meus agradecimentos também à Lucent e ao CPqD pela extensão da minha bolsa, pois sem ela dificilmente eu teria completado minha jornada.

Muito obrigado a todos!

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

Um Ambiente de Simulação de Fluido com Aplicações a Redes Multimídia

Kelvin de Freitas Reinhardt

Julho/2002

Orientador: Edmundo de Souza e Silva
Programa: Engenharia de Sistemas e Computação

Os métodos tradicionais de simulação, utilizados para modelagem de redes de computadores, enfrentam problemas em relação ao custo computacional, já que as redes atuais possuem uma grande capacidade de transmissão e em geral grandes dimensões. Para minimizar este problema surgiram diversas técnicas, entre elas, a simulação de fluido, onde o tráfego entre os nós de rede é tratado de forma contínua, em vez de unidades discretas. Sua principal vantagem está no fato de que somente as mudanças de taxas precisam ser tratadas. Neste trabalho, foi criado um paradigma de modelagem de fluido baseado em recompensas de taxa. De acordo com este paradigma, foi idealizado, projetado, construído e validado, um simulador de fluido com foco na área de redes. Este foi construído sobre o simulador da ferramenta TANGRAM-II, onde foram adicionadas diversas funcionalidades. O simulador de fluido herdou as características e o poder de modelagem do TANGRAM-II e de acordo com seus princípios, é genérico o suficiente para modelar praticamente qualquer tipo de sistema, ao mesmo tempo que possui objetos de rede prontos para o uso de forma rápida e simples.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A Fluid Simulation Environment Applied to Mutimedia Networks

Kelvin de Freitas Reinhardt

July/2002

Advisor: Edmundo de Souza e Silva
Department: Computer and System Engineering

The traditional simulation methods used to model computer networks have had problems with computational costs when modeling nowadays networks, since they have high bandwith and large size. To reduce the computational costs and to make feasible the modeling of bigger networks, new techniques were developed, such as fluid simulation. In this kind of simulation, the traffic is considered a fluid (instead of discrete packets) and the main advantage is concerned in the fact that only rate changes must be treated by the simulator. In this work, were created a fluid modeling paradigm based on rate rewards. Accordingly with this paradigm, a fluid simulator focused in computer networks was idealized, designed, developed and validated. It was developed using TANGRAM-II simulator where several features were added. The new fluid simulator inherited the facilities and the power of TANGRAM-II tool. According to the used paradigm, it is generic enough to model almost all kinds of systems and at same the time there are built-in network objects that enable network modeling in a simple and fast way.

Palavras-chave

1. Simulação de Fluido
2. Análise de Desempenho
3. Redes Multimídia
4. Ferramenta TANGRAM-II

Glossário

ATM : Modo de Transferência Assíncrono (*Asynchronous Transfer Mode*);

CP : Particionamento Completo de Espaço em Fila. (*Complete Partitioning*);

CR : Recompensa Acumulada. (*Cumulative Reward*);

CS : Compartilhamento Completo de Espaço em Fila. (*Complete Sharing*);

FCFS : O primeiro a chegar é o primeiro a ser servido. (*First Come First Serve*);

FIFO : O primeiro a entrar é o primeiro a sair. (*First In First Out*);

FLB : Regulador de Tráfego de Fluido. (*Fluid Leaky Bucket*);

IR : Recompensa Instantânea. (*Instantaneuos Reward*);

MMFS: Fonte de Fluido Modulada por Markov. (*Markov Modulated Fluid Source*);

MMPS: Fonte de Pacotes Modulada por Markov. (*Markov Modulated Packet Source*);

GPS : Divisão Personalizada do Processador. (*Generalized Processor Sharing*);

UPS : Divisão Uniforme do Processador. (*Uniform Processor Sharing*);

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

ESTE trabalho de tese abrange o projeto e implementação de um simulador de fluido no ambiente de modelagem e análise TANGRAM-II. O simulador de fluido fora concebido de forma a se enquadrar dentro do paradigma da ferramenta, encontra-se embutido no módulo de simulação e tem como foco a modelagem de redes de computadores.

Nas seções seguintes são apresentados: (1.1) a motivação e objetivo, que introduzem o trabalho como um todo e apresentam, dentre outros, um resumo das técnicas de resolução de modelos, uma breve descrição de simulação de fluido e a abrangência da tese; (1.2) a contribuição, onde o simulador é brevemente descrito e o resultado alcançado é apresentado; e (1.3) o roteiro que descreve a dissertação sob um ponto de vista macro.

1.1 Motivação e Objetivo

As redes de computadores continuam a desenvolver-se num ritmo acelerado, crescendo em tamanho e complexidade. Este desenvolvimento tem o suporte de técnicas de engenharia apoiadas em ambientes de modelagem e análise, onde dentre outras, encontram-se ferramentas de simulação e soluções analíticas. Estas permitem a análise de mecanismos relacionados ao comportamento de uma rede, seja esta préexistente ou em fase de desenvolvimento.

O comportamento dos sistemas é analisado através de resultados obtidos de modelos, cuja resolução pode ser feita com a utilização de métodos analíticos que produzem resultados exatos ou boas aproximações destes. Os métodos analíticos, podem ser muitas ordens de magnitude mais econômicos em termos computacionais que a simulação, além de freqüentemente permitirem um conhecimento mais profundo e exato de certas propriedades do sistema. O maior problema se dá pelo fato destes métodos exigirem condições nos modelos que por muitas vezes são difíceis de satisfazer, inviabilizando assim, a modelagem de grande parte dos sistemas. Existem também técnicas numéricas que se encontram numa faixa entre a simulação e os métodos analíticos, tanto em termos de restrições quanto custo computacional.

Entretanto, a técnica mais utilizada para resolução de modelos é a de simulação, pela sua capacidade em representar com detalhes qualquer mecanismo de um sistema. Apesar de todo este poder, existe um preço a pagar que está na complexidade de programação e nos custos computacionais de tempo.

Na simulação de sistemas de computação, um paradigma dos mais usados é a orientação a eventos, ou seja, o sistema reage sempre que há a ocorrência de um evento e o tempo de simulação salta de evento em evento. Como exemplos de evento numa simulação de redes, cita-se a geração de pacote, uma mudança de estado de alguma fonte de tráfego, o término de serviço de um pacote, dentre outros. Desta forma, cada mensagem a ser transmitida é representada no modelo, assim como sua evolução entre os diferentes nós da rede.

Num modelo de rede de alta velocidade, uma simulação usando o paradigma tradicional, orientado a eventos, tem problemas de custo computacional. Um exemplo deste problema pode ser mostrado num modelo de rede ATM, onde são geradas e transferidas entre os nós milhões de células por segundo. Supondo-se que a medida de interesse a ser calculada, seja a probabilidade de descarte de células na entrada de um roteador, e sabendo-se que em geral essa probabilidade é baixa, da ordem de 10^{-6}, pode-se deduzir que centenas de eventos de perda de células necessitarão ser gerados para que se possa obter um mínimo de precisão no resultado, o que exige a geração e manipulação de centenas de bilhões de células. Isto porque o descarte de células é um evento raro e ocorre em média (neste exemplo) 1 a cada 1 bilhão de células geradas.

Para lidar com o problema de simulação de eventos raros pode-se utilizar técnicas como por exemplo *importance sampling* e *spliting/RESTART*, estudadas em [14]. Nestes casos é possível obter uma boa precisão nos resultados com menos esforço computacional. O módulo de simulação do TANGRAM implementa a técnica de spliting/RESTART. Outra técnica utilizada consiste em um tipo de simulação híbrida onde o tempo é discretizado em pedaços de igual tamanho. Esta técnica, apresentada em [21], é chamada TSHS (*Time-stepped Hybrid Simulation*) e pode ser mais econômica em termos computacionais, em troca de uma perda de precisão das medidas. A perda de precisão das medidas, assim como o ganho de velocidade, estão relacionados com a granularidade em que os eventos são analisados. Ainda existem outras técnicas, que são brevemente descritas em [23], tais como: cálculos preliminares (*preliminary calculations*), agregação de tráfego (*traffic aggregation*), métodos de extrapolação (*extrapolation methods*) - que estimam eventos raros - e paralelismo. Entretanto, mesmo que a medida a ser calculada não esteja diretamente relacionada à ocorrência de um evento raro, caso o tempo para se atingir o estado estacionário seja grande em relação ao intervalo de tempo do sistema, um número muito elevado de eventos deve ser gerado para que se possa obter a medida de interesse com um intervalo de confiança razoável.

O objeto de estudo deste trabalho concentra-se em uma técnica distinta das mencionadas acima, baseada na simulação de modelos de estado contínuo, que são os chamados modelos de fluido. Nesta abordagem, o fluxo de unidades discretas que viaja através dos canais e fica armazenado nas filas é substituído por fluidos que se movem de um recipiente para outro. Deste modo classificam-se como eventos apenas as mudanças de taxa na transferência dos fluidos. Assim sendo, em vez de modelar cada célula individualmente, apenas modela-se eventos de mudança de taxas de transmissão dos fluidos. Como em geral estas mudanças de taxas ocorrem com frequência bem inferior em relação ao intervalo entre transmissão de pacotes, não fica difícil perceber o ganho desta abordagem em termos de esforço computacional.

Cabe salientar que os modelos de fluido são uma abstração de mais alto nível em relação aos modelos tradicionais, o que requer um atenção especial em relação aos tipos de medidas de interesse que podem ser obtidos e à precisão destas medidas.

Baseado no fato de que a modelagem de sistemas têm um papel importante na engenharia de computação, procurou-se através do estudo aprofundado da técnica de fluido, uma forma de minimizar um dos principais problemas da simulação: o custo computacional. Este estudo mostrou que a técnica pode realmente trazer enormes ganhos em termos de custo computacional.

O objetivo maior deste trabalho, foi o desenvolvimento de um simulador de redes de computadores capaz de trabalhar de acordo com a técnica de fluido, cuja principal vantagem é oferecer resultados de simulação de boa qualidade, a um custo bem menor que o simulador tradicional.

1.2 Contribuição

Criou-se um paradigma de modelagem de fluido baseado em recompensas onde, o comportamento dos fluidos é modelado através do uso de recompensas de taxa. Esta associação mostrou-se bastante interessante, por sua simplicidade, flexibilidade e capacidade de representar com exatidão o comportamento dos fluidos.

Baseado neste paradigma, um simulador de fluido com foco na área de redes de computadores, foi idealizado, projetado, construído e validado. O simulador de fluido tem como base o simulador de eventos discretos da ferramenta TANGRAM-II. Nele, vários recursos foram inseridos, de forma que o simulador pudesse operar de acordo com a técnica de fluido, além é claro, de operar os modelos tradicionais. Os diversos elementos de uma rede foram estudados e modelados em forma de objetos. Os modelos de fluido são construídos através da composição destes elementos primários, e os resultados da simulação são obtidos através de medidas de interesse especificadas para cada objeto.

Alguns pontos importantes que distinguem a ferramenta TANGRAM-II em relação a outros trabalhos publicados na literatura cabem ser ressaltados. O novo simulador herdou todas as características de facilidade e poder de modelagem do simulador do TANGRAM-II e de acordo com o mesmo paradigma, é genérico o suficiente para modelar praticamente qualquer tipo de sistema. Ao mesmo tempo possui objetos de rede previamente construídos, que permitem a modelagem de sistemas de forma rápida e até mesmo por indivíduos com conhecimento menos apurado sobre o assunto. O simulador também conta com interessantes recursos de apresentação de resultados onde, além de medidas sobre os pontos de interesse, há opções de geração de *traces*, que mostram o comportamento do modelo. Estes podem ser visualizados em formato de gráficos, diretamente através da interface gráfica da ferramenta.

A contribuição inclui também a elaboração e implementação de novos recursos para a ferramenta TANGRAM-II, baseados no paradigma de modelos com recompensa. A ferramenta implementa uma série de soluções analíticas para modelos markovianos com recompensa [17, 13, 18, 19] e portanto o simulador desenvolvido neste trabalho complementa os trabalhos anteriores fornecendo ao analista um largo espectro de opções para o desenvolvimento e solução de modelos, não encontrados em outras ferramentas existentes.

1.3 Roteiro

O capítulo 2 descreve o ambiente de modelagem TANGRAM-II. O Capítulo 3 mostra a teoria de fluido através do estudo dos componentes básicos que compõem uma rede de computadores. No capítulo 4 é descrita a implementação do simulador dentro do TANGRAM-II. Os resultados obtidos e a validação do recursos implementados aparece no capítulo 5 e o 6 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Ambiente de Modelagem TANGRAM-II

NESTE capítulo o ambiente de modelagem TANGRAM-II é descrito. Este ambiente permite a construção de modelos e a sua solução tanto pela utilização de métodos analíticos como através de simulação.

2.1 Ambiente de Modelagem

A ferramenta TANGRAM-II possui um poderoso ambiente de modelagem desenvolvido para fins de pesquisa e ensino que possibilita a descrição de sistemas gerais através de uma interface amigável [5, 35]. Apesar de possuir um paradigma genérico o suficiente para suportar a descrição de sistemas gerais, o objetivo principal é fornecer suporte à descrição de sistemas de computação e comunicação. A descrição gráfica dos modelos é feita através da utilização de uma ferramenta de domínio público denominada, TGIF (*Tangram Graphic Interface Facility*) [6] que tem como paradigma o trabalho com gráficos vetoriais.

O paradigma de descrição de modelos foi proposto em [3] e é orientado a objetos. Ele suporta modelos genéricos o suficiente para descrever modelos de sistemas de computação e obter medidas de confiabilidade e de desempenho.

Uma primeira versão rudimentar da ferramenta foi implementada em linguagem Prolog em 1991 [11]. Em 1997 uma versão com novos recursos foi implementada em C/C++ [5] e desde então vem sendo aprimorada. Em 1998 parte do código foi reimplementado e otimizado de forma a produzir um ganho de desempenho além de facilitar ainda mais o uso da ferramenta. Nos anos seguintes a ferramenta recebeu inúmeras novas funcionalidades, além de ganhar uma interface gráfica escrita em Java que permite o uso facilitado dos diversos módulos do TANGRAM-II. Atualmente, dentre as funcionalidades, destaca-se um ambiente de engenharia de tráfego de redes [35]. Desde outubro de 2000 a ferramenta é distribuída gratuitamente pela internet [22].

De acordo com o paradigma da ferramenta [3], os modelos são representados por um conjunto de objetos que podem interagir entre si através do envio de mensagens. O estado interno de um objeto é armazenado por um conjunto de variáveis inteiras e o comportamento de cada um é definido por eventos que possuem uma condição e um conjunto de ações, e também por ações associadas à chegada de mensagens.

Os eventos são habilitados para o disparo sempre que a condição associada seja satisfeita. O tempo de disparo de cada evento, obedece a uma distribuição de probabilidade. A ferramenta possui as seguintes distribuições: exponencial, determinística, erlangiana, gaussiana, lognormal, uniforme, pareto, FBM, FARIMA, weibull e *file* onde as amostras são lidas de um arquivo. As amostras de tempo de disparo dos eventos são geradas pela ferramenta, desde que a condição associada esteja satisfeita no tempo atual, e a execução do evento só acontecerá se esta condição permanecer válida no tempo de disparo. As condições para o disparo de eventos podem ser descritas em função do estado interno do objeto.

Mensagens são abstrações criadas para possibilitar a interação entre os objetos e são trocadas em tempo zero. Quando um evento ocorre ou uma mensagem é recebida, um conjunto de ações é executado com uma determinada probabilidade. O tempo de execução destas ações também é zero. No final da execução de uma ação o objeto pode ter seu estado alterado e mensagens podem ter sido enviadas a outros objetos. Estas mensagens serão tratadas pela ferramenta ainda em tempo zero. Durante esta execução o sistema pode passar por um conjunto de estados evanescentes. Estes estados evanescentes não pertencem ao espaço de estados do modelo. Um estado evanescente é uma abstração para facilitar a modelagem. Estes estados podem ocorrer enquanto existirem mensagens sendo trocadas entre os objetos. Após o tratamento de todas as mensagens o sistema estará num novo estado válido, denominado tangível.

Para criar um modelo, o usuário pode contar com objetos pré-construídos que acompanham a distribuição da ferramenta, ou ainda, pode iniciar a descrição de um objeto particular através do uso de um objeto "vazio" chamado *obj_template*. Faz-se necessária a especificação dos eventos (subentende-se a especificação da sua condição e ações associadas), das ações associadas ao recebimento de mensagens, das variáveis de estado, e outras especificações, que ao todo irão compor a funcionalidade do objeto. Este processo exige conhecimentos do sistema que está sendo modelado, bem como das facilidades e características do módulo de modelagem da ferramenta que é apresentado em mais detalhes na próxima seção.

Após a descrição do modelo, o usuário precisa resolvê-lo para que possa obter as medidas de interesse desejadas. Este processo pode ser feito de duas formas possíveis: através de solução analítica ou através da simulação.

2.1.1 Solução Analítica

Para resolver o modelo através do uso de métodos analíticos se faz necessária, em geral, a geração de todo o espaço de estados do modelo, bem como a montagem da matriz que representa as transições entre estes.

Dado um estado inicial apontado pelo usuário, o gerador identifica todos os eventos habilitados e então um evento é escolhido para ser executado. A ação correpondente é executada, podendo ocasionar uma mudança de estado no objeto e o envio de um conjunto de mensagens, o que na prática é representado por uma lista de mensagens que devem ser entregues no próximo passo. Devido a existência de mensagens a serem processadas, o estado em questão é dito evanescente.

No próximo passo cada uma das mensagens é entregue e sua ação correspondente é executada. Esta ação por sua vez pode mudar o estado do objeto em questão e enviar mais mensagens. Este processo é repetido até que não existam mais mensagens a serem tratadas. Neste instante tem-se um estado tangível, que pertence ao espaço de estados do sistema.

Durante este processo de busca por um estado tangível, as taxas de transição também são calculadas. Este algorítmo é repetido recursivamente, fazendo um busca em profundidade nos estado tangíveis [14]. A partir do momento em que a geração do espaço de estados e a da matriz de transição de estados é concluída, o modelo pode ser resolvido.

No TANGRAM-II existe um poderoso módulo de solução analítica, que é capaz de resolver algumas classes de modelos, através do uso de diversos métodos de solução, onde podem ser feitas análises em estado estacionário e transiente dos modelos em questão. Em [27] existe uma descrição do módulo e a lista dos métodos utilizados.

Vários métodos para resolução de modelos Markovianos podem ser encontrados, tais como GTH, GAUSS-SIEDEL, JACOBI, SOR e POWER que trabalham com solução em estado estacionário. Para a solução transiente, vários métodos estão implementados baseados na técnica de uniformização [20]. Por exemplo, é possível a obtenção das probabilidades de estado no tempo t, assim como o cálculo da distribuição da recompensa acumulada num intervalo de tempo $(0, t)$. Estes métodos permitem a solução de modelos onde o tempo de disparo de um evento é dado por uma distribuição exponencial, isto é, modelos que podem ser representados por uma cadeia de Markov.

Ainda existe um método que permite a solução de uma classe de modelos não Markovianos [12]. Este método pode resolver um subconjunto de modelos que possuam eventos exponenciais e determinísticos. Uma restrição implica na impossibilidade de lidar com modelos onde exista mais de um evento determinístico habilitado ao mesmo tempo.

2.1.2 Simulação

A outra forma de resolução de um modelo é a simulação, onde é possível resolver modelos com distribuições genéricas predefinidas ou até mesmo modelos cujo intervalo entre disparos de eventos é dado por valores lidos de arquivos de trace, como descrito em [27]. Além desta vantagem, cita-se que na simulação o espaço de estados do modelo não precisa ser armazenado.

A simulação progride de forma semelhante à descrita na seção 2.1, onde é descrito o processo de geração do espaço de estados, com a diferença de que o único estado armazenado em memória é o estado presente e não é preciso determinar todo o espaço de estados. Apenas os estados pertencentes ao caminho amostral sendo gerado são obtidos. Os resultados vão sendo coletados e mantidos em estruturas de dados em memória e posteriormente escritos em arquivos, ou são gravados diretamente, durante o processo de simulação, quando a opção gerar *traces* estiver selecionada.

Após esta explanação sucinta sobre o funcionamento interno da ferramenta, retoma-se o processo de criação de modelos que é apresentado de forma mais detalhada na próxima seção. Estes passos precisam ser compreendidos pois são cruciais para que o leitor possa compreender o trabalho desta dissertação.

2.1.3 Criação de modelos

O primeiro passo a ser seguido na criação de um modelo, é a definição de seus objetos e a forma de interação entre estes. Para aplicar a devida funcionalidade a um objeto o projetista precisa definir as suas variáveis de estado, suas recompensas e quais são seus eventos, bem como em quais condições estes estarão habilitados. O projetista precisa também escrever o código representando a ação referente a cada um destes eventos, assim como a ação associada ao recebimento de cada mensagem.

A seguir, mais detalhes são apresentados através do uso de um exemplo bastante simples em caráter ilustrativo, onde pressupõe-se que um usuário queira modelar uma fila M/M/1/k. Neste modelo, uma fonte emite pacotes a uma taxa λ de acordo com uma distribuição exponencial. Estes pacotes são enviados para um objeto fila-servidor que é responsável por armazená-los (caso haja espaço) e serví-los de acordo com uma taxa μ cuja distribuição também é exponencial. Ao armazenar um pacote, o objeto fila-servidor incrementa uma variável que representa o número de pacotes na fila e ao servir a decrementa.

Para construir o modelo, inicialmente, o usuário deve chamar o módulo de modelagem, Figura 2.1. Esta interface foi criada e desenvolvida durante os estudos desta tese e tem como objetivos principais facilitar e guiar o usuário na utilização da ferramenta.

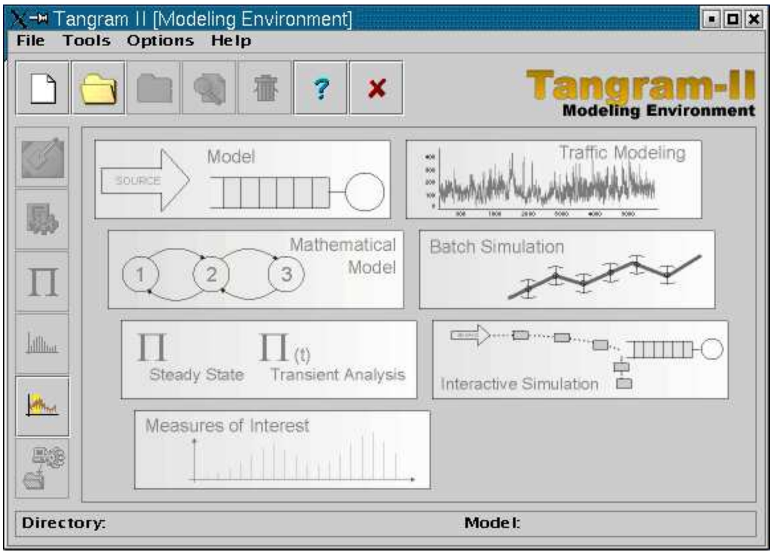


Figura 2.1: Interface Principal do Ambiente de Modelagem

Inicialmente, o ambiente de modelagem aparece com todas as figuras em preto e branco, e à medida que o projetista avança no processo de modelagem as figuras vão ficando coloridas, simbolizando as etapas já completadas. Nesta seção apenas é coberto o primeiro passo: a criação do modelo.

O usuário deve pressionar o botão *new* e escolher o nome para o novo modelo a ser criado. Após isto, o botão de chamada da ferramenta TGIF encontrar-se-á habilitado, indicando o próximo passo. Toda a especificação do modelo ocorrerá dentro do TGIF, cuja interface é ilustrada na Figura 2.2.

A definição de um objeto consiste basicamente da descrição de seis atributos,

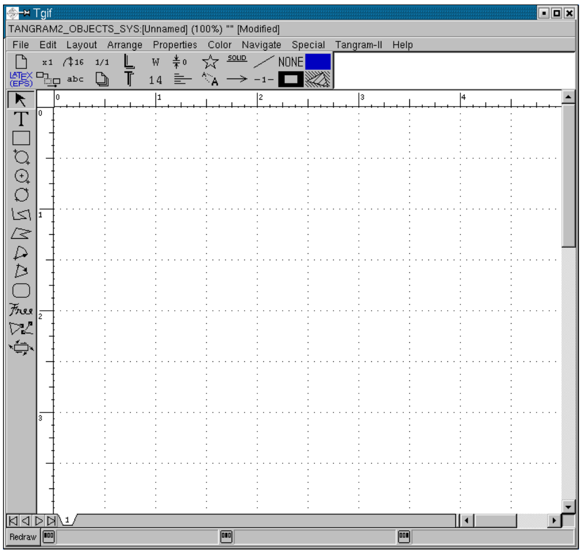


Figura 2.2: TANGRAM Graphic Interface Facility - TGIF

citados abaixo:

- Declaration (Declarações)
- Initialization (Inicializações)
- State Variables (Variáveis de Estado)
- Events (Eventos)
- Messages (Mensagens)
- Rewards (Recompensas)

Cada um destes atributos é apresentado em mais detalhes a seguir, juntamente com a descrição, em caráter ilustrativo, do objeto fila-servidor.

Declaration

Este atributo é responsável pela declaração das variáveis, constantes e parâmetros usados no objeto. Os três tipos de declaração são: variáveis (*Var*), constantes (*Const*) e parâmetros (*Param*), onde, os seguintes tipos podem ser usados:

- State: usado para declarar variáveis que representam o estado interno dos objetos.
- Integer, Float: usados para especificar constantes numéricas ou parâmetros.
- Object: usado para referenciar outros objetos do modelo.
- Port: define portas usadas na comunicação entre objetos.

O objeto fila-servidor precisa de uma variável de estado para armazenar o valor da fila (*fila*), uma constante inteira (*tam_fila*) para representar o tamanho máximo da fila, uma constante real (*taxa_serv*) para representar a taxa de serviço e uma variável do tipo porta *porta_ent*) usada para o recebimento dos pacotes.

```
Declaration=
 Var
   State   : fila;
 Const
   Integer: tam_fila;
   Float   : taxa_serv;
   Port    : porta_ent;
```

Initialization

Este atributo é responsável pela inicialização das variáveis de estado e pela definição das constantes. Supondo que a fila-servidor tem capacidade de servir 20 pacotes por unidade de tempo, que o tamanho do *buffer* é de 100 pacotes e que inicialmente a fila está vazia, o atributo seria como segue:

```
Initialization=
 fila      = 0
 tam_fila  = 100
 taxa_serv = 20.0
 porta_ent = conector
```

State Variables

Este atributo é utilizado no simulador interativo. As variáveis que forem nele indicadas, terão seu valor atualizado a cada passo de simulação. Pode-se ainda alterar o seu valor e com isto forçar uma mudança no caminho amostral. Maiores detalhes sobre este atributo podem ser encontrados no manual da ferramenta [27].

Events

Este atributo é utilizado para declaração e definição dos eventos do objeto. Um evento precisa ter pelo menos uma condição e uma ação associadas a ele. Em se tratando do evento de serviço da fila-servidor, sabe-se que a condição para que este esteja habilitado é a existência de algum pacote a ser servido, e a ação associada é o decremento do número de pacotes na fila. De acordo com as regras da ferramenta, para que se possa alterar o valor de uma variável de estado, é necessário o uso de uma função especial *set_ st* que deve ser chamada sempre no final da ação.

```
Events=
 event = servico (EXP, taxa_serv)
   condition= (fila > 0)
   action= {
             int f;

             f = fila - 1;

             set_st("fila", f);
          };
```

Messages

Neste atributo, todas as mensagens recebidas pelo objeto são tratadas. Ações são associadas à chegada de mensagens, e quando uma mensagem específica é recebida estas ações são executadas. As mensagens são identificadas pela constante *Port*, que indica por qual porta a mensagem está sendo recebida. No exemplo em questão, cada mensagem recebida representa a chegada de um novo pacote e a ação associada tem como função incrementar o número de pacotes na fila, caso esta não esteja cheia.

```
Messages=
 msg_rec = porta_ent
   action= {
             int f;
```

```
        f = fila;
        if (fila < tam_fila)
           f = fila + 1;

        set_st("fila", f);
     };
```

Rewards

Neste atributo são especificadas as recompensas de taxa e de impulso. Estes conceitos e a sintaxe merecem uma explicação mais detalhada e são apresentados separadamente na próxima seção.

Durante esta breve explanação foram apresentados os passos para a criação de um único objeto do modelo M/M/1/k, o fila-servidor. A fonte de pacotes poisson é muito simples e pode ser rapidamente deduzida se os conceitos acima apresentados forem levados em consideração.

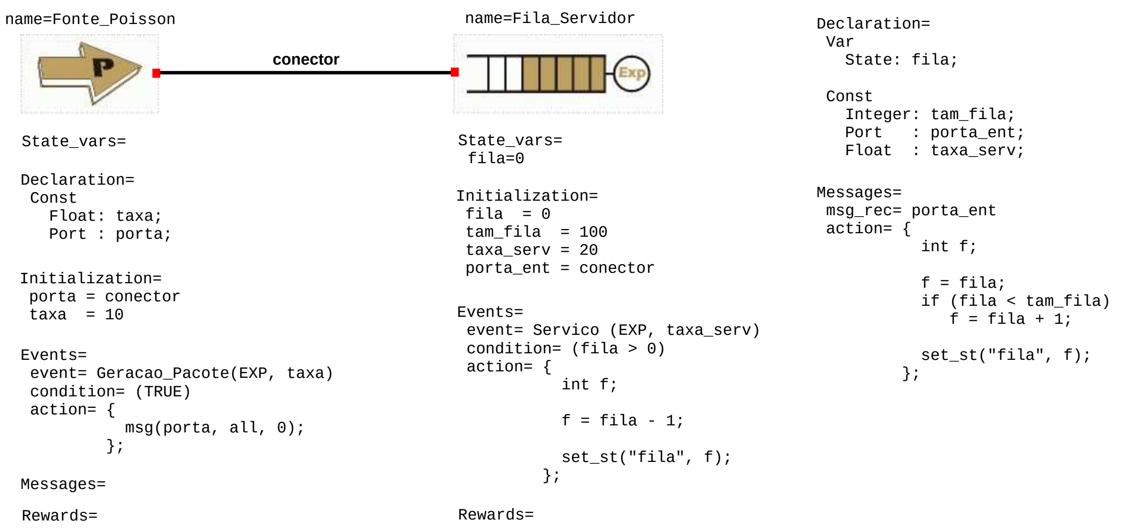


Figura 2.3: Modelo de Fila M/M/1/k no Ambiente de Modelagem

A Figura 2.3 apresenta o modelo completo e dispensa maiores comentários. Exemplos mais complexos e maiores detalhes sobre a ferramenta podem ser encontrados em [27].

2.1.4 Uso de Recompensas

Existe, pelo menos, duas formas de definir recompensas em um modelo. Seguem as definições de recompensa de taxa e recompensa de impulso.

Definição de Recompensa de Taxa Acumulada

Seja um sistema com espaço de estados finito $S = \{s_i, i = 1, \dots, M\}$. A cada estado $s \in S$ está associada uma taxa de recompensa de um conjunto $R = \{r_{c(s_i)}, i = 1, \dots, M\}$, onde $r_{c(s_i)}$ representa a recompensa ganha por unidade de tempo em que o sistema se encontra no estado s_i e $c(s_i)$ é uma função que mapeia o índice i do estado s_i no índice de recompensa associada a s_i. Seja $IR(t)$ a variável aleatória que representa a recompensa instantânea no tempo t. A variável aleatória recompensa acumulada é definida, durante o intervalo $[0, t]$, por:

$$CR(t) = \int_0^t IR(\tau)d\tau \tag{2.1}$$

A taxa média da recompensa acumulada em um intervalo de tempo $[0, t]$ é dada por:

$$ACR(t) = \frac{CR(t)}{t}, \tag{2.2}$$

e a média no tempo da recompensa acumulada é:

$$ATC(t) = \frac{\int_0^t CR(\tau)d\tau}{t} \tag{2.3}$$

Diversas medidas de interesse podem ser calculadas a partir do conceito de recompensa de taxa associada. Por exemplo, o valor médio de uma fila em $[0, t]$ pode ser obtido, através da criação de uma recompensa de taxa que acumule o tamanho da fila, ou seja, IR igual ao tamanho atual da fila. Neste exemplo a medida ACR seria a variável aleatória desejada. Outros exemplos, inclusive de recompensas que descrevem filas, são apresentados no decorrer da tese.

De forma análoga, segue a definição formal de recompensa de impulso.

Definição de Recompensa de Impulso Acumulada

Seja $\sigma_n = (s', s)$ a n-ésima transição feita pelo sistema e que ocorre do estado s' para o estado s. Define-se a variável ρ_{σ_n} como sendo o valor da recompensa de impulso ganho quando o sistema transiciona de s' para s. Sabendo-se que N(t) é o número de transições feitas até o tempo t, tem-se que a recompensa de impulso acumulada no período $[0, t]$ é

$$CI(t) = \sum_{n=0}^{N(t)} \rho_{\sigma_n} \tag{2.4}$$

Analogamente, dividindo o valor da recompensa acumulada pelo tempo de observação, obtem-se a média

$$ACI(t) = \frac{CI(t)}{t} \tag{2.5}$$

Assim como também é possível definir a média no tempo da recompensa acumulada

$$TCI(t) = \frac{\int_0^t CI(\tau)d\tau}{t} \tag{2.6}$$

Como exemplo de utilização deste conceito, cita-se a definição de uma recompensa de impulso acumulada associada a uma transição de estado indicando falha de transmissão. Desta forma, a medida $CI(t)$ traria a informação sobre o número de falhas de transmissão ocorridas no sistema em $[0, t]$.

Recompensas no TANGRAM-II

Pode-se fazer uso dos conceitos de recompensa acumulada no ambiente de modelagem da ferramenta TANGRAM-II, onde é permitida a definição de recompensas tanto de taxa quanto de impulso. Para manter a generalidade das definições, o significado das medidas e sua interpretação fica a cargo do usuário. Isto faz com que a ferramenta seja flexível e poderosa, sendo ao mesmo tempo, fácil de usar. Exemplos de medidas de interesse que podem ser obtidas são: utilização, vazão, tamanho médio de filas, tempo médio de espera, número de falhas, tempo operacional, tempo de reparo, dentre muitas outras.

Recompensa de Taxa no TANGRAM-II

Tendo em vista os conceitos formais apresentados acima, sabe-se que para se obter medidas de interesse através do uso de modelos com recompensas de taxa, precisa-se atribuir a cada estado do sistema um valor que indique a quantidade de recompensa que a medida está acumulando por unidade de tempo em que permanecer no dito estado. Por exemplo, num modelo onde deseja-se obter a medida de desempenho vazão (*throughtput*) de um dado sistema, a recompensa precisa determinar o desempenho que o sistema está obtendo em cada momento (estado). Desta forma, medidas como o desempenho médio do sistema podem ser facilmente obtidas.

No TANGRAM-II as recompensas estão associadas aos estados dos objetos através de condições. Um objeto pode ter diversas recompensas definidas, de forma que várias medidas possam ser obtidas simultaneamente. Assim sendo, pode-se descrever a sintaxe da ferramenta, salientando que tudo aqui referenciado encontra-se dentro do atributo *Rewards* dos objetos. Inicialmente o usuário precisa definir um nome para sua recompensa que represente a medida de interesse. Após a definição do nome, devem ser definidos pares *condição / valor*, onde a *condição* define o conjunto de estados em que a recompensa de taxa irá acumular o *valor* indicado. Nos estados onde a condição não for satisfeita nada será acumulado.

No exemplo anterior (M/M/1/k), toma-se por base que a medida de interesse seja a utilização do objeto fila-servidor. Assim sendo, deve-se definir uma recompensa de taxa que acumule um valor unitário sempre que haja pacotes a serem servidos. A sintaxe da definição é apresentada logo abaixo:

```
Rewards=
 rate_reward = utilizacao
   condition = (fila > 0)
   value = 1;
```

Pressupõem-se agora um outro exemplo, onde existam N servidores para atender a demanda da fila. Para a obtenção da utilização, precisa-se associar o valor 1, quando todos servidores estiverem trabalhando e o valor $fila/N$ quando existirem servidores ociosos. Segue a sintaxe:

```
Rewards=
 rate_reward = utilizacao
   condition = (fila < N)
   value = fila / N;
   condition = (fila >= N)
   value = 1;
```

Ao permitir que o usuário especifique vários pares *condição / valor*, pode-se obter estados onde duas ou mais condições possam ser satisfeitas, caso não sejam mutuamente exclusivas. Nesta situação a ferramenta provê quatro opções de comportamento:

- média - define como valor da recompensa a média aritmética entre todos os valores das condições que foram avaliadas como verdadeiras
- máximo - define como valor da recompensa o máximo entre todos os valores avaliados
- mínimo - define como valor da recompensa o mínimo entre todos os valores avaliados
- erro - gera um erro avisando ao usuário que mais de uma condição foi satisfeita numa recompensa para o mesmo estado

Uma destas opções deve ser escolhida antes da geração do modelo matemático, possivelmente cadeia de Markov, ou antes do início de uma simulação. A interface provê este controle ao usuário e tem inicialmente marcada a opção erro.

Recompensa de Impulso no TANGRAM-II

Da mesma forma que é possível associar valores de recompensa de taxa aos estados do sistema, pode-se associar valores de recompensa às transições de estado. Com as recompensas de impulso, diversas outras medidas podem ser obtidas. Por exemplo, pode-se avaliar o número de falhas num roteador, associando-se uma recompensa de impulso ao evento "falha". Toda vez que o evento ocorrer, uma recompensa é acumulada, indicando mais uma falha. A medida $CI(t)$ desta recompensa fornece o número de falhas ocorridas até o instante t. As transições de estado podem ocorrer no disparo de eventos e no tratamento de mensagens, logo, as recompensas de impulso podem ser associadas a ambas. Uma transição se inicia quando ocorre o disparo de um evento. No entanto esta transição pode resultar em diversos estados diferentes, caso existam diferentes ações que possam ser tomadas. O envio de mensagens também pode dar origem a transições com destinos distintos. Desta forma, para que se possa referenciar exatamente a transição desejada, é possível a indicação da ação específica na qual a recompensa deverá ser associada, seja esta ação parte integrante do código de disparo de eventos ou do código de tratamento de mensagens.

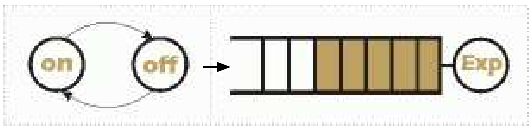
Da mesma forma que a recompensa de taxa, o usuário necessita especificar um nome único que irá representar a medida de interesse desejada. Em seguida deve-se especificar o evento ou mensagem a qual a recompensa estará associada e uma ou mais ações que, ao serem executadas, farão com que a recompensa acumule um dado valor. Em um caso onde haja a necessidade do conhecimento do número de pacotes transmitidos pela fonte no exemplo M/M/1/k, pode-se associar uma recompensa de impulso ao evento *geracao_pacote* da fonte. Sintaxe:

```
Rewards=
 impulse_reward = num_pacs
   event = geracao_pacote,1
   value = 1;
```

Cada evento (ou execução de mensagem) pode estar associado a uma ou mais ações. Neste exemplo nota-se que a ação associada foi a primeira dentro do evento *geracao_pacote*. Este número que indica a ação é relativo a ordem de aparição das ações no código do usuário e inicia com 1.

Outro exemplo do uso de recompensas de impulso é mostrado na Figura 2.4, que apresenta um modelo composto de uma fonte *On-Off*, que só transmite pacotes durante a permanência no estado *On*, e de um objeto fila-servidor que recebe os pacotes emitidos pela fonte. Neste exemplo, a fila não é modelada da maneira tradicional com uma variável de estado para representar seu tamanho, e sim, através

```
name=OnOff_fila_servidor
```




```
State_vars=
  status = 0

Initialization=
  status     = 0
  taxa_OnOff = 2
  taxa_OffOn = 2
  taxa_gera  = 2
  taxa_serv  = 1

Messages=

Rewards=
 impulse_reward=buffer
   cr_bounds= 0, 100
   event=Geracao, 1
   value= 1;
   event=Servico, 1
   value= -1;
```

```
Declaration=
 Var
  State: status;
 Const
  Float: taxa_OnOff, taxa_OffOn,
         taxa_gera, taxa_serv;

Events=
 event= OnOff (EXP, taxa_OnOff)
 condition= (status == 1)
 action= { ; set_st("status", 0); };

 event= OffOn (EXP, taxa_OffOn)
 condition= (status == 0)
 action= { ; set_st("status", 1); };

 event= Geracao (EXP, taxa_gera)
 condition= (status == 1)
 action= { /* nada */ ; };

 event= Servico (EXP, taxa_serv)
 condition= (TRUE)
 action= { /* nada */ ; };
```

Figura 2.4: Fila Modelada com Recompensa de Impulso

da especificação de uma recompensa de impulso, de onde pode-se obter as medidas desejadas com vantagens em termos de redução do número de estados do modelo e conseqüentemente no custo computacional para sua resolução. Pode-se associar impulsos aos eventos de transmissão de pacotes da fonte. Além disto, um novo evento que simboliza o serviço da fila está embutido no mesmo objeto, fazendo com que este na prática assuma todas as funcionalidades do sistema, passando a ser fonte, fila e servidor ao mesmo tempo. A este novo evento, associa-se uma recompensa negativa que representa o serviço de um pacote na fila. Logo uma recompensa chamada *buffer* irá descrever exatamente o comportamento da fila, recebendo uma unidade a cada pacote gerado e perdendo uma unidade a cada pacote servido. Entretanto uma fila não pode ter um número negativo de pacotes e sabe-se que na prática são limitadas em relação ao tamanho máximo que podem armazenar. Por este motivo existe a possibilidade de especificação de limites no valor acumulado da recompensa, através do uso da palavra reservada *cr_bounds* (*cumulative reward bounds*), onde podem ser especificados os limites inferior e superior que são aplicados ao valor acumulado. Assim $CI(t)$ conterá o número de pacotes na fila no tempo t e $TCI(t)$ a média de pacotes na fila até o mesmo instante.

Este recurso de limite pode ser aplicado tanto às recompensas de impulso ($CI(t)$) quanto às de taxa ($CR(t)$). Se omitido, o valor acumulado pode tender a infinito. Além disto, pode-se especificar apenas um dos limites, inferior ou superior, através do uso da palavra reservada INF num dos extremos, como mostra o exemplo a seguir, onde um buffer infinito é modelado.

```
Rewards=
 impulse_reward = buffer
   cr_bounds = 0, INF
   event = transmissao,1
   value = 1;
   event = servico,1
   value = -1;
```

O conceito de recompensas, bem como seu uso na ferramenta, é de vital importância para o entendimento deste trabalho, uma vez que os fluidos manipulados pelo simulador são representados através destas recompensas. Estes detalhes são apresentados no capítulo 4.

Capítulo 3

Características e Componentes Básicos de um Simulador Orientado a Fluido para Simulação de Modelos de Redes de Computadores

ESTE capítulo tem como objetivo apresentar de forma detalhada a descrição de modelos de fluido aplicados a redes de computadores, bem como seu comportamento durante o processo de simulação, através da especificação de seus principais componentes lógicos. O embasamento para as teorias descritas ao longo deste, no contexto de redes e teoria de filas, pode ser encontrado em [24, 42, 38, 4]. Já os conceitos e equações de fluido, provêm de trabalhos que utilizam a técnica [16, 34, 25, 43, 2, 31, 33, 1].

3.1 Introdução

Os simuladores orientados a eventos, reagem sempre que há a ocorrência de um evento no sistema sendo modelado e o tempo de simulação salta de evento em evento. A abordagem clássica para a modelagem de redes, de acordo com este paradigma, especifica eventos como geração de pacote, mudança de estado de fonte, serviço de um pacote, dentre outros.

No entanto, este método pode ter um alto custo computacional, uma vez que todas as mensagens (pacotes ou células) são representadas no modelo, assim como a evolução destas entre os diferentes nós da rede. Por isso faz-se necessário o uso de diferentes técnicas de simulação, de forma que se possa obter resultados com menos esforço computacional. Uma das técnicas possíveis consiste na simulação de modelos de estado contínuo, que são os chamados modelos de fluido.

Estes modelos de fluido são uma abstração dos modelos convencionais discretos, onde os dados que trafegam na rede são tratados de forma contínua. Em modelos de redes de computadores que usam esta abordagem, o fluxo de unidades discretas que viaja através dos canais de comunicação e fica armazenado nas filas é substituído por fluidos que se movem de um recipiente para outro. Deste modo classificam-se como eventos apenas as mudanças de taxa na transferência dos fluidos. Assim sendo, em vez de um evento ser gerado cada vez que um pacote é transmitido, apenas é necessário representar eventos relativos a mudanças das taxas de transmissão. Para certos modelos, como por exemplo aqueles que descrevem redes de alta velocidade, onde os intervalos de mudanças de taxas de transmissão, são ordens de grandeza maiores que o intervalo entre geração ou término do serviço de pacotes, fica evidente o ganho desta nova abordagem que necessita tratar apenas eventos de mudanças de comportamento dos fluxos em vez da enorme quantidade de eventos que representam a atividade dos pacotes. Cabe ressaltar que os modelos de fluido também possuem estados discretos, além dos contínuos que modelam os fluidos.

Ao longo deste capítulo, é apresentado um estudo descritivo sobre a teoria dos modelos de fluido. Para mostrar a dinâmica de funcionamento destes modelos, optou-se pela apresentação de alguns dos principais componentes lógicos que compõem uma rede modelada sob o paradigma de fluido. Obviamente, estes componentes são abstrações dos componentes reais que compõem as redes multimídia atuais.

Questões relativas ao ganho em termos de custo computacional e em relação à qualidade das medidas de interesse obtidas, são enfocadas no final do capítulo.

Cabe salientar ainda que mesmo tendo como enfoque principal os modelos de redes de computadores, este trabalho descreve a técnica de simulação de fluido que é bastante genérica e pode ser aplicada a outras áreas, além da computação. Um exemplo, no campo da física, seria a modelagem de um sistema de vasos comunicantes, onde a água armazenada numa caixa d'água de um bairro, localizada numa altitude maior, é transferida para caixas d'água e cisternas residenciais, localizadas numa altitude menor. A transferência da água acontece, sempre que uma bóia reguladora de nível indique que uma determinada caixa d'água necessita ser enchida. O modelo poderia ser estendido às torneiras, chuveiros e demais saídas de uma residência, representando o consumo de água das residências do bairro inteiro.

3.2 Fontes de Tráfego

Estes objetos representam os geradores de tráfego da rede. Por exemplo, tráfego proveniente dos mais variados tipos de servidores, onde cita-se www, e-mail, troca de arquivos, até os mais modernos serviços como distribuição de vídeo sob demanda, teleconferências e transmissão de voz sobre IP. Todas estas fontes geradoras de tráfego podem ser representadas através de modelos chamados MMPS (*Markov-Modulated Packet Source*) que são as fontes de pacotes moduladas por Markov. Estas fontes são modeladas por uma cadeia de Markov de tempo contínuo com n estados. A cada estado tem-se associada uma taxa de transmissão γ_i, onde γ_i representa a taxa de um processo de Poisson responsável pela geração de pacotes (ou células) no mesmo. Em outras palavras, quando a fonte encontra-se em um estado i, pacotes são gerados em intervalos exponenciais, com média γ_i. Existem taxas de transição entre os estados, que são responsáveis pelo comportamento da fonte. A Figura 3.1 mostra um exemplo de fonte MMPS do tipo nascimento e morte, onde existem taxas λ_i para transições de nascimento e taxas μ_i para mortes. Sendo assim, percebe-se que o funcionamento deste tipo de fonte é bastante simples, pois durante a evolução do processo, ou seja, enquanto a cadeia vai mudando de estado, a fonte emite pacotes a diferentes taxas, podendo variar de 0 à ∞.

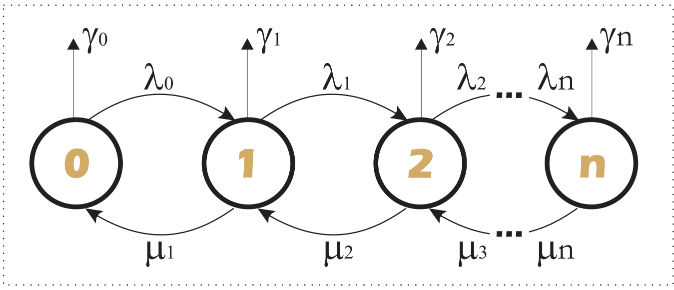


Figura 3.1: Fonte de Pacotes (MMPS) - Nascimento e Morte

As fontes não necessariamente precisam ser do tipo nascimento e morte e podem assumir qualquer comportamento descrito por uma cadeia de Markov, onde as transições entre os estados são descritas por uma matriz de taxas Q com espaço de estados $S = \{1, 2, 3, \ldots, n\}$.

Como o enfoque deste trabalho está dentro do paradigma de fluidos, utilizam-se fontes ligeiramente modificadas, onde, para cada estado associa-se a emissão de fluido a uma taxa constante γ_i, em vez da geração de pacotes individuais. No que diz respeito a transição de estados não existem diferenças. Estas fontes de fluido, utilizadas no decorrer deste trabalho, são denominadas MMFS (*Markov-Modulated Fluid Sources*) fontes de fluido moduladas por Markov.

De forma a ilustrar a diferença entre os tipos de fontes citados, são apresentadas duas figuras. A Figura 3.2 contém uma fonte do tipo *On-Off* e mostra a diferença de abstração entre pacotes e fluido. Na Figura 3.3 observa-se o comportamento do tráfego acumulado (em *bytes*), gerado em um período de atividade da fonte. Estas fontes *On-Off* são amplamente utilizadas por serem bastante simples e terem a capacidade de representar diversas fontes de tráfego reais.

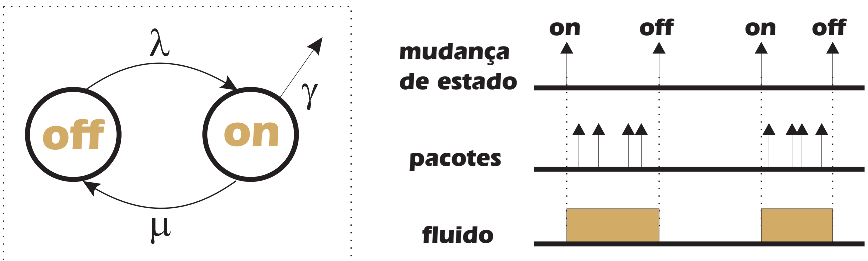


Figura 3.2: Fonte On-Off de Pacotes (MMPS) e Fluido (MMFS)

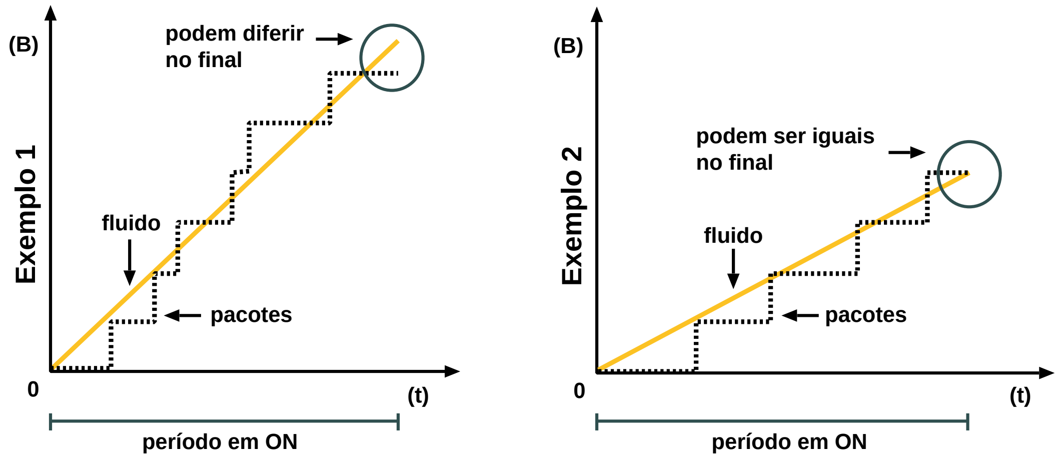


Figura 3.3: Tráfego Acumulado em Fontes On-Off de Pacotes e Fluido

Sendo a fonte do tipo MMFS nota-se que durante o estado *On* existe a emissão constante de fluido à uma taxa γ, entretanto se esta for considerada MMPS, percebe-se que durante este estado pacotes são emitidos de acordo com um processo de Poisson de média γ. Em ambos os tipos, não existe emissão alguma no estado *Off*, onde a fonte permanece sem atividade. Além disto, as transições do estado *Off* para o estado *On* e do estado *On* para o estado *Off* ocorrem segundo processos Poisson com médias λ e μ respectivamente.

As fontes MMFS descritas acima são utilizadas em praticamente todos os trabalhos relacionados a modelos de fluido e podem ser encontradas em [33, 16, 1, 2].

3.3 Representação das Filas de uma Rede de Computadores

Esta seção apresenta os principais componentes lógicos de um sistema de rede, a fila, suas disciplinas de atendimento e gerenciamento e o roteamento que pode ocorrer junto a esta. As filas, ou *buffers* podem estar presentes em diversos equipamentos, que podem ser caracterizados como um nó de rede, tais como *bridges*, *switches*, roteadores, dentre outros, cuja descrição pode ser encontrada em [26]. De forma simplificada, pode-se dizer que cada nó de uma rede tem pelo menos uma fila, onde os pacotes são armazenados, e possui uma disciplina de atendimento que será utilizada para estabelecer a ordem em que estes pacotes serão retransmitidos. Ainda existem outros fatores a serem considerados, tais como a forma de gerenciamento do espaço em fila. Ou ainda, o roteamento que geralmente é feito neste componente. Estes fatores são abordados ao final desta seção, que começa pela definição da fila em si e da descrição de duas disciplinas de atendimento.

Enfatiza-se que numa rede de fluido não existem pacotes chegando e sendo armazenados, e sim fluidos escoando em direção a um nó e sendo acumulados em um reservatório. Em cada reservatório existe uma saída para o líquido armazenado, que escoa, em direção a outro nó, a uma determinada taxa.

3.3.1 Recipiente de Armazenamento

O primeiro ponto a ser considerado é o próprio *buffer*, que representa o espaço disponível para o armazenamento dos fluidos. Este espaço, que na prática é constituído por células de memória, é representado por sua capacidade B, que varia, em teoria, entre 0 e ∞. Esta capacidade é uma medida de volume, e comporta o armazenamento de igual volume de fluidos.

Análogo ao funcionamento de um *buffer* de pacotes discretos, onde pacotes são perdidos caso não exista espaço suficiente para guardá-los, o volume de fluido que exceder o volume máximo de armazenamento é descartado no momento de sua chegada na fila, sendo o processo de perda irreversível. Na outra extremidade, o fluido é servido, ou seja retransmitido, o que contribui para a diminuição de volume do fluido armazenado.

3.3.2 Disciplinas de Atendimento

Um ponto de extrema importância a ser considerado é a disciplina de atendimento da fila de um nó de rede. No entanto para que as disciplinas possam ser definidas formalmente, juntamente com a explanação sobre estas, são considerados outros fatores intrínsecos. Um deles é o *buffer* em si, outro, são os fluxos, de entrada e saída, que caracterizam a movimentação dos líquidos e são representados por vetores, além de outros detalhes cujo conjunto compõe o sistema em análise.

FIFO

Seja um nó com uma fila finita de capacidade $B \leq \infty$, uma taxa de serviço constante c e uma disciplina de atendimento conservadora de trabalho FIFO (*First In First Out*), onde o serviço ocorre de acordo com a ordem de chegada. Esta disciplina também é conhecida por FCFS (*First Come First Serve*), ou seja, o primeiro a chegar é o primeiro a ser servido.

Seja $A(t) \in [0, \infty)$ a taxa do total de fluido que chega na fila no tempo $t \geq 0$. Supondo que $A(t)$ seja uma função degrau contínua à direita, pode-se dizer que a integral $\int_0^t A(u)du$ é contínua e linear por partes. Impondo esta restrição para $A(t)$, pode-se simplificar as equações de forma a facilitar a análise matemática do comportamento dos fluidos. Esta premissa é bastante plausível já que esta condição não é difícil de ser satisfeita. Como exemplos cita-se fontes *On-Off* não Markovianas ou ainda toda a classe de fontes Markovianas MMFS recentemente apresentada.

Seja $Q(t)$ o volume de fluido na fila no tempo $t \geq 0$. A dinâmica de $Q(t)$ pode ser descrita de forma exata pela seguinte equação:

$$Q(t) = Q(0) + \int_0^t (A(u) - c)\mathbf{1}(u \in \sigma)du, \quad t \geq 0, \tag{3.1}$$

onde, $\mathbf{1}$ representa uma função indicadora que assume os valores 0 se $u \ni \sigma$ ou 1 se $u \in \sigma$. (a) Para um buffer de capacidade infinita σ é dado por $\sigma = \{u \geq 0 | A(u) > c$ ou $Q(u) > 0\}$ e (b) para um buffer finito, $\sigma = \{u \geq 0 | (A(u) > c$ e $Q(u) < B)$ ou $(A(u) \leq c$ e $Q(u) > 0)\}$.

Para as funções degrau contínuas à direita, esta integral resulta em:

$$Q(T_{n+1}) = min\{B, [Q(T_n) + (A(T_n) - c)(T_{n+1} - T_n)]^+\} \tag{3.2}$$

onde, T_n representa a n-ésima transição de $A(t)$ e $(y)^+ = max(0, y)$. De acordo com estas equações, percebe-se que o caminho amostral resultante de $Q(t)$ é linear por partes, com inclinações definidas por:

$$\frac{dQ}{dt} = (A(t) - c)\mathbf{1}(t \in \sigma) \tag{3.3}$$

Estas mudanças de inclinação ocorrem no instante em que a fila enche, esvazia ou nas transições T_n de $A(t)$.

A taxa de saída pode ser descrita pela seguinte equação:

$$D(t) = c\mathbf{1}(Q(t) > 0 \ ou \ (A(t) > c) + A(t)\mathbf{1}(Q(t) = 0 \ e \ A(t) \leq c). \tag{3.4}$$

Partindo da suposição inicial, onde A é degrau contínua à direita, pode-se deduzir que $D(t)$ também se classifica da mesma forma. Outro aspecto importante a ser considerado é o fato de as filas não serem bloqueantes, ou seja, o fluido que chega numa fila finita cheia é perdido. Esta perda acumulada durante o intervalo $[0, t]$ pode ser calculada pela integral $\int_0^t (A(u) - c)\mathbf{1}(u \in \varsigma)du$, onde ς representa todos os períodos de sobrecarga (fila cheia) e é dado por $\varsigma = u \geq 0 | A(u) > c \ e \ Q(u) = B$. Salienta-se ainda que dado um tempo t, chama-se $t_0(t)$ o início do próximo período de fila vazia, assim como $t_B(t)$ o início do próximo período de sobrecarga.

Até aqui, foi apresentado um nó que trata apenas um fluxo de fluido, entretanto o modelo acima descrito, pode ser aplicado a um caso mais geral onde a fila é alimentada por N fluidos. Este caso ocorre quando lida-se com diferentes fluxos de tráfego e pode-se pensar, em se tratando de fluidos, em líquidos de diferentes cores, um para cada fluxo, que possivelmente não se misturam.

Seja $a_i(t) \in [0, \infty)$ a taxa do i-ésimo fluido de entrada no nó e $\vec{a}(t) = \langle a_1(t); a_2(t); \ldots; a_N(t)\rangle$ o vetor que representa os fluxos de todos os fluidos de entrada. Desta forma, a taxa total é dada por $A(t) = \sum_{i=1}^{N} a_i(t)$. Se cada fluxo $a_i(t)$ for descrito por uma função degrau contínua à direita, continua-se a respeitar a condição de que $A(t)$ também seja degrau contínua à direita, uma vez que a soma de funções degraus contínuas à direita, resulta numa função de mesmo tipo. De forma semelhante, $\vec{d}(t) = \langle d_1(t); d_2(t); \ldots; d_N(t)\rangle$ representa o vetor de saída e $D(t) = \sum_{i=1}^{N} d_i(t)$ denota a taxa do fluxo total de saída do nó.

O fato de lidar com N fluidos, requer uma atenção especial sobre o comportamento da fila, quando se leva em consideração a disciplina de atendimento FIFO.

Seja τ_n a n-ésima transição do vetor de entrada $\vec{a}(t)$. Desta forma, uma mudança em $\vec{a}(t)$ que ocorra em $t = \tau_n$ levará $Q(\tau_n)/c \geq 0$ unidades de tempo para se propagar até a saída da fila. Isto ocorre pois quando trabalha-se de acordo com a disciplina de atendimento FIFO, o serviço ocorre de acordo com a ordem de chegada. O tempo de propagação é igual ao tempo que o servidor leva para transmitir a quantidade $Q(\tau)$ de fluido que já estava na fila no momento da mudança. Sendo assim, no tempo $\omega_n = \tau_n + Q(\tau_n)/c$, a proporção dos fluxos de saída deve ser a mesma dos fluxos de entrada no tempo τ_n. Logo, se $A(\tau_n) > 0$, então para cada fluxo i, $d_i(\omega_n)/D(\omega_n) = a_i(\tau_n)/A(\tau_n)$. Cabe ainda salientar que, uma mudança em $\vec{a}(t)$, pode produzir duas transições (não simultâneas) em $\vec{d}(t)$ caso $Q(\tau_n) > 0$, $A(\tau_n) < c$ e a fila esvaziar antes da próxima transição em $\vec{a}(t)$.

O comportamento do vetor de fluxos de saída $\vec{d}(t)$ é descrito pelas seguintes equações:

$$\vec{d}(\omega_n) = \begin{cases} \dfrac{D(\omega_n)}{A(\tau_n)}.\vec{a}(\tau_n) & \text{, se } A(\tau_n) > 0, \\ 0 & \text{, se } A(\tau_n) = 0 \text{ e } Q(\tau_n) = 0 \text{ (Caso em que } \omega_n = \tau_n\text{)}, \end{cases} \tag{3.5}$$

e

$$\vec{d}(t_0(\tau_n)) = \vec{a}(\tau_n), \quad \text{se } t_0(\tau_n) < \tau_{n+1} \text{ e } Q(\tau_n) > 0. \tag{3.6}$$

As equações que descrevem o comportamento dos fluidos na fila FIFO foram extraídas de [33, 25].

Para ilustrar o comportamento de uma fila FIFO, toma-se por base o modelo descrito na Figura 3.4. No exemplo existem dois fluxos provenientes de fontes MMFS, representados pelo vetor $\vec{a} = \langle a_1; a_2 \rangle$, além da fila FIFO e do vetor de saída $\vec{d} = \langle d_1; d_2 \rangle$. Salienta-se que os fluidos são de cores diferentes e não se misturam. Além disto eles são representados como se fossem blocos. Esta disposição em blocos, serve para diferenciar a proporção de chegada, e acúmulo de fluidos, nas diferentes fases da evolução do fluxo de entrada.

Evolução do sistema: No tempo $t = 0$ a fila está vazia e não há fluido chegando no nó, $\vec{a} = \langle 0; 0 \rangle$. Num instante seguinte $t = \tau_1$, a fonte 1 passa para o estado 1 e começa a transmitir a uma taxa $0,3c$. Neste instante, de acordo com a equação 3.5

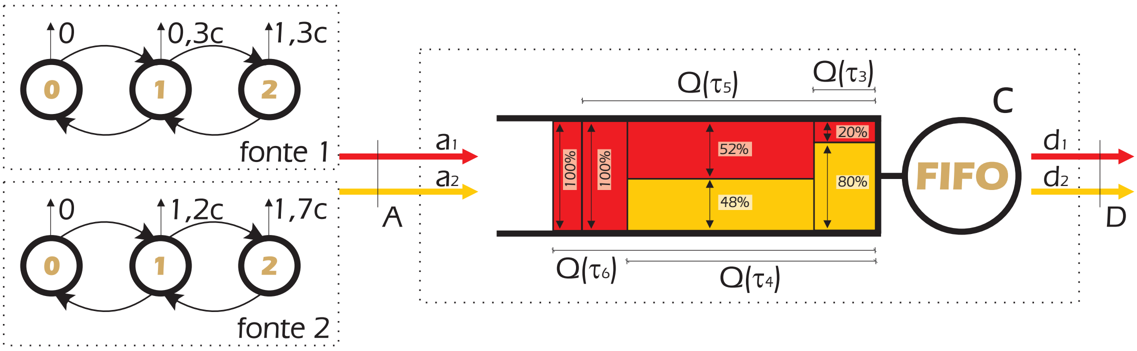


Figura 3.4: Exemplo de Fila FIFO

tem-se que $\vec{d}(\omega_1) = \langle 0,3c; 0\rangle$.

Imaginando então, que a fonte 2 passe para o estado 1, e comece a transmitir a uma taxa $1,2c$. Neste momento tem-se um fluxo de entrada $A(\tau_2)$ maior do que a capacidade de serviço c, portanto, passa-se a ter acúmulo de fluidos no interior do buffer, que se dá de acordo com a equação 3.2. Em $t = \tau_2$, tem-se $A(\tau_2) = 1,5c$, o serviço, dado pela equação 3.5, é $\vec{d}(\omega_2) = \langle 0,2c; 0,8c\rangle$ e observa-se um acúmulo no buffer de $0,1c$ e $0,4c$ unidades de volume por unidade de tempo para q_1 e q_2 respectivamente. Desta forma, se τ_3 ocorrer x unidades de tempo após τ_2, ter-se-á uma quantidade total de fluido acumulada na fila, de $x \times [A(\tau_2) - c] = x \times 0,5c$ unidades.

Supondo que em τ_3, a fonte 1 mude para o estado 2 e passe a transmitir numa taxa $1,3c$, ocasiona-se então a formação de um novo bloco, que pode ser observada na Figura 3.4. Neste bloco existe uma nova proporção de acúmulo e serviço para os fluidos, onde $\vec{a}(\tau_3) = \langle 1,3c; 1,2c\rangle$, $\vec{d}(\omega_3) = \langle 0,52c; 0,48c\rangle$ e um acúmulo de $0,78c$ e $0,72c$ unidades por unidade de tempo para q_1 e q_2 respectivamente. Sabe-se que esta saída $\vec{d}(\omega_3)$, só entrará em vigor, $Q(\tau_3)/c$ unidades de tempo após τ_3, pois este é o tempo que a mudança levará para se propagar até a saída (justamente o tempo de serviço de $Q(\tau_3)$).

A Figura 3.5 mostra o sistema sob o ponto de vista dos fluxos que entram, ficam e saem da fila, ou seja, mostra a evolução de $A(t)$, $Q(t)$ e $D(t)$.

Observando seu comportamento apartir de τ_4, quando a fonte 2 é desligada,

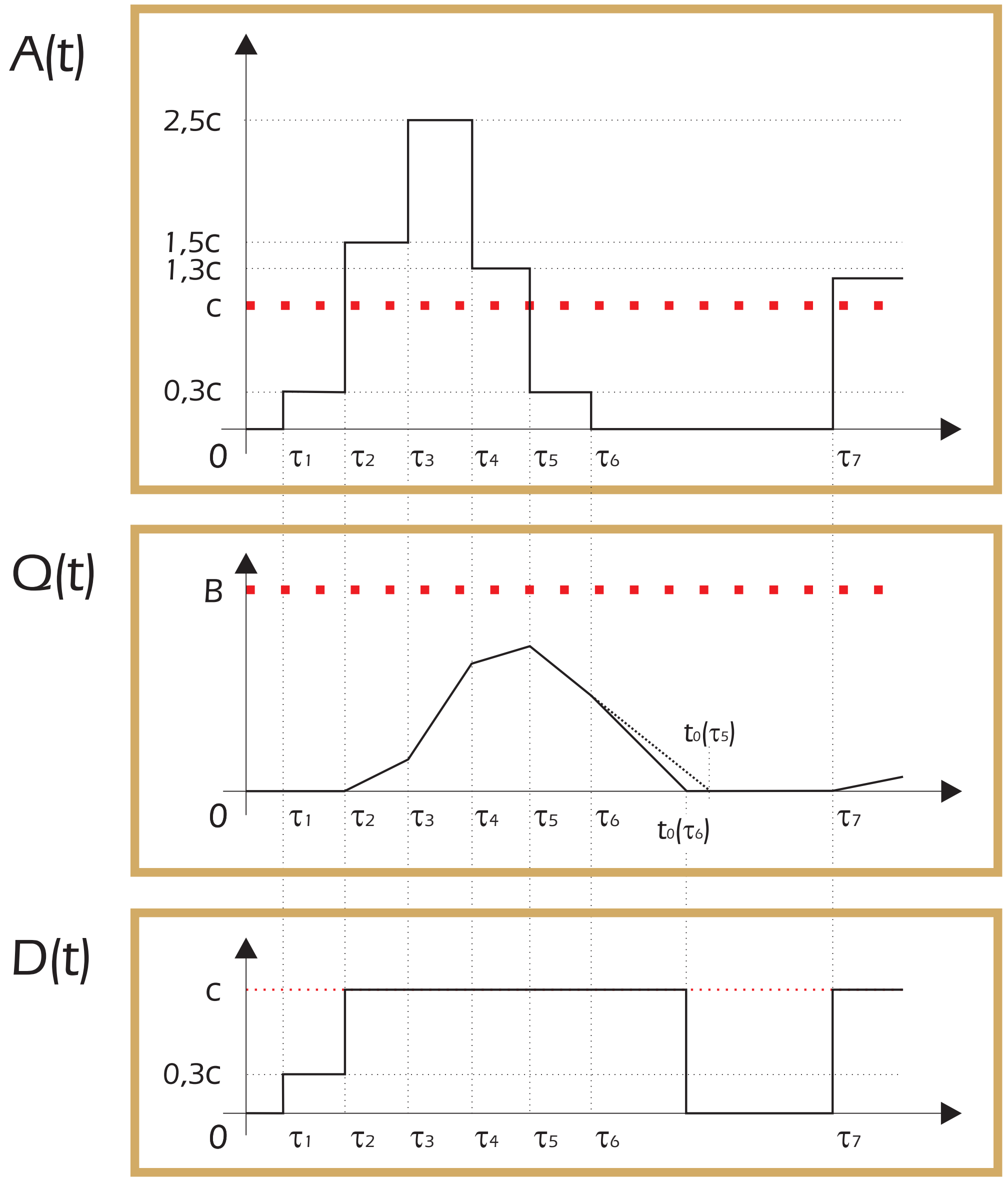


Figura 3.5: Evolução de A(t), Q(t) e D(t) na Fila FIFO

constata-se uma entrada de $A(\tau_4) = 1,3c$ e uma inclinação de $Q(\tau_4)$ com coeficiente angular $m = 0,3$. Em τ_5 a fonte 1 passa para o estado 1, quando tem-se $A(\tau_5) = 0,3c$ e $Q(\tau_5)$ com uma inclinação negativa (coeficiente angular $m = -0,7$). Neste instante, seria possível prever um esvaziamento do *buffer* no instante $t_0(\tau_5)$, desde que nenhuma outra mudança no fluxo de entrada ocorra. Entretanto, se ocorrer alguma mudança no vetor de entrada em $\tau_6 < t_0(\tau_5)$, o esvaziamento pode não ocorrer, sendo necessário o novo cálculo para prever o esvaziamento do *buffer* $t_0(\tau_6)$. É o que acontece neste exemplo, quando em τ_6 a fonte 1 desliga, fazendo com que o vetor de entrada tenha valores nulos. O fluido que está na fila continua a ser servido, vide $Q(t)$ e $D(t)$ da Figura 3.5. Observa-se que agora o ângulo de inclinação de Q mudou, o que deve acarretar no cancelamento do evento $t_0(\tau_5)$ previsto anteriormente. No mesmo instante um novo evento sinalizando o término de saída de fluxo é escalonado para $t_0(\tau_6)$.

Uma nova mudança no fluxo de entrada ocorre somente em $\tau_7 > t_0(\tau_6)$. Neste caso, o esvaziamento do *buffer* ocorre no tempo previsto ($t_0(\tau_6)$), não havendo o cancelamento deste tipo de evento como ocorreu em $t_0(\tau_5)$. Assim sendo, ter-se-á $D(t_0(\tau_6)) = A(\tau_6) = 0$.

Acima, foram apresentadas características da fila de fluido com disciplina de atendimento FIFO, que é apenas uma das formas existentes de se lidar com o fluxo de entrada de fluidos. Outra disciplina de interesse, a GPS, é caracterizada na próxima subseção.

GPS

A disciplina de atendimento GPS (*Generalized Processor Sharing*), surgiu de uma generalização natural da UPS (*Uniform Processor Sharing*), onde a capacidade do canal é dividida igualmente entre os fluxos. Já que a GPS é intrinsecamente contínua, uma versão discreta para lidar com pacotes foi proposta, com o nome PGPS (*Packet-by-Packet Generalized Processor Sharing*). A PGPS é uma aproximação da GPS e foi proposta originalmente por Demers, Shenker e Keshav sob o nome WFQ (*Weighted Fair Queueing*). Estes e outros detalhes podem ser encontrados em [32].

A GPS divide a capacidade do canal de saída em n partes, de acordo com os parâmetros de particionamento $\phi_1, \phi_2, \ldots, \phi_n$, satisfazendo as condições $\phi_1 + \phi_2 + \ldots + \phi_n \leq 1$ e $\phi_i \geq 0$. Cada parte, que recebe uma percentagem do serviço, pode ser interpretada como uma fila FIFO, e pode manusear um fluxo de fluido, ou um grupo de fluxos, desde que estes fluidos sejam misturados. A disciplina GPS é conservadora de trabalho e opera de forma que a banda alocada para uma parte ociosa, é distribuída para as demais na proporção de seus parâmetros de particionamento. Na Figura 3.6 observa-se a representação gráfica de uma fila com disciplina de atendimento GPS.

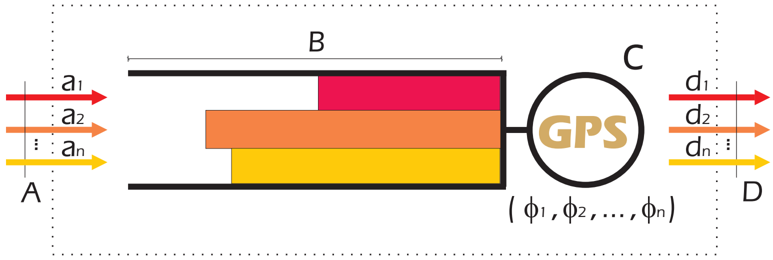


Figura 3.6: Nó de Rede com Fila de Fluido GPS

A dinâmica da fila é descrita pelas seguintes equações:

$$\begin{aligned} q_i(T_{n+1}) = q_i(T_n) &+ [(a_i(T_n) - d_i(T_n))\mathbf{1}(0 \leq q_i(T_n) < B_i) \\ &+(a_i(T_n) - d_i(T_n))^{-}\mathbf{1}(q_i(u) = B_i)] \times (T_{n+1} - T_n) \end{aligned} \tag{3.7}$$

$$\frac{d_i}{d_j} = \frac{\phi_i}{\phi_j}, \quad se \;\; Q_i > 0 \;\; e \;\; Q_j > 0 \tag{3.8}$$

onde, $i, j \in \{1, 2, 3, \ldots, n\}$, representam as filas FIFO do nó GPS e para cada uma delas, tem-se que B_i representa seu tamanho, $a_i(t)$ representa a entrada de fluido, $d_i(t)$ a saída e $q_i(t)$ o volume acumulado no *buffer*. T_n representa a n-ésima transição de $A(t)$, $(y)^{-} = min(0, y)$ e $\mathbf{1}$ é a função indicadora.

Uma equação equivalente à 3.8 é: para todo $i \in \{1, 2, 3, \ldots, n\}$,

$$d_i = \frac{\phi_i \mathbf{1}\{q_i > 0\}}{\Sigma_{j=1}^{n} \phi_j \mathbf{1}\{q_j > 0\}} c \tag{3.9}$$

assumindo-se $\frac{0}{0} = 0$.

Estas equações que descrevem o comportamento dos fluidos na fila GPS foram extraídas de [16].

3.3.3 Disciplinas de gerenciamento de fila

As disciplinas apresentadas acima, descrevem a maneira no qual o fluido é servido na fila de um nó de rede. Entretanto para que se possa garantir o senso de justiça, a robustez, e a qualidade de serviço (QoS), há também, a necessidade de um gerenciamento do espaço da fila, ou em outras palavras, uma política de ocupação de *buffer*. Sem este gerenciamento, as características das disciplinas de atendimento podem ficar comprometidas. O exemplo a seguir, mostra como a falta de gerenciamento do espaço pode acarretar em injustiças. Seja fila finita de tamanho B, que recebe 2 fluxos de fluidos, um amarelo e outro vermelho. A disciplina de atendimento é GPS, de capacidade c e pesos idênticos $\phi_{amarelo} = 0,5$ e $\phi_{vermelho} = 0,5$. Imagina-se um cenário, onde num dado instante a fila esteja cheia, com 50% de fluido de cada cor e o amarelo passe a chegar a uma taxa bastante alta, em relação a capacidade de serviço de sua classe, por exemplo. chegada de $1,25c$ sendo o serviço da classe amarela $0,5c$. Supõem-se também, que a taxa de chegada do

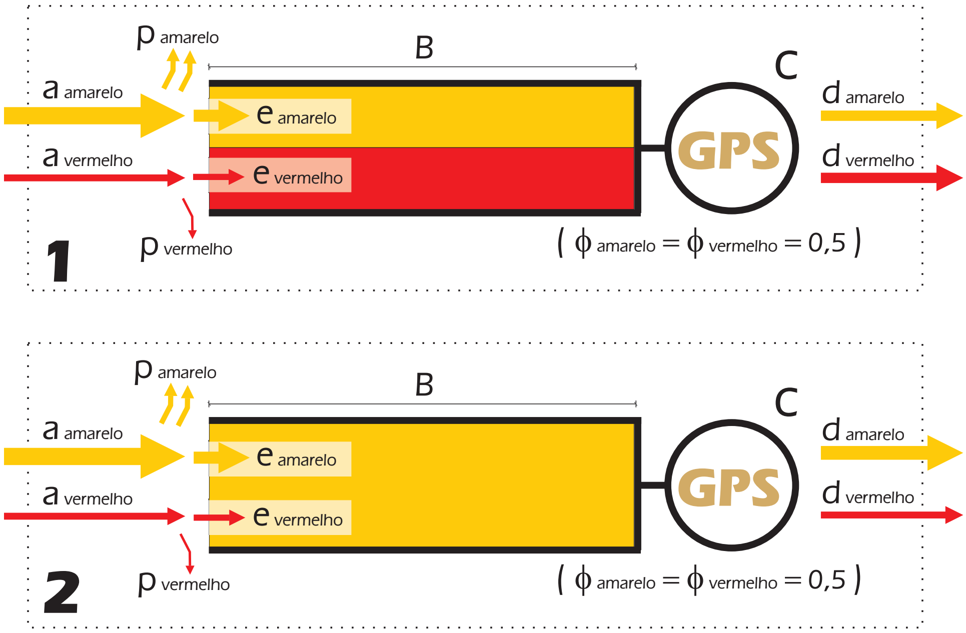


Figura 3.7: Injustiça em uma Fila sem Gerenciamento de Espaço

vermelho diminua para um valor menor que a capacidade do canal destinada a este, por exemplo $0,3c$. Neste cenário, a taxa do amarelo é maior que a do vermelho, $a_{amarelo} = 1,25c > a_{vermelho} = 0,3c$ e a soma da duas taxas de entrada é maior que a capacidade de serviço, $1,55c > c$ (Vide parte 1 da Figura 3.7).

Nesta situação, ocorrerão perdas na entrada da fila e apenas parte do fluxo conseguirá lugar no *buffer*, como mostra a equação:

$$A = P + E, \tag{3.10}$$

onde A é a chegada total, P a perda total e E a quantidade total que realmente entra na fila. $E = \Sigma_i e_i$, $A = \Sigma_i a_i$ e $P = \Sigma_i p_i$. A taxa de entrada de cada fluxo e_i é diretamente proporcional a sua taxa de chegada a_i, como mostra a seguinte equação:

$$e_i = \frac{a_i}{\Sigma_l a_l} c \ . \tag{3.11}$$

Sabe-se também que a cada unidade de fluido servida, uma nova unidade entra no *buffer*, sendo esta unidade de entrada, uma mistura de fluidos, de acordo com a proporção explicitada na equação 3.11.

Assim sendo, o fluido amarelo vai entrar em maior quantidade, por causa de sua maior avidez. $e_{amarelo} \cong 0,8c$ e $e_{vermelho} \cong 0,2c$. Como a quantidade de fluido vermelho que entra é menor que sua respectiva saída, $(e_{vermelho} \cong 0,2c) < (d_{vermelho} = 0,5c)$, a tendência é que ocorra o esvaziamento deste fluido no *buffer*. Assim como o amarelo tenderá a tomar todo o espaço da fila (Vide parte 2 da Figura 3.7). No momento em que o fluido vermelho se extingüir da fila, sua taxa de saída passará a ser igual a de entrada, $d_{vermelho} \cong 0,2c$, e sendo esta menor do que o $0,5c$, o vermelho deixará de usufruir parte da capacidade a que tinha direito. Parte esta que vai ser aproveitada pelo amarelo, já que a disciplina é conservadora de trabalho. Assim sendo, o amarelo vai ganhar maior banda, $d_{amarelo} \cong 0,8c$, por ser mais ávido, e o vermelho mesmo chegando a uma taxa menor do que sua banda nominal garantida pela disciplina de serviço GPS, terá perdas e não conseguirá usufruir da banda a que tem direito (50% da capacidade do canal). Caracteriza-se assim, uma injustiça em relação à qualidade de serviço.

Existem várias formas de resolver estas injustiças, dentre as quais destacam-se algumas bastante conhecidas, descritas em [41, 30]. Estas técnicas são aplicadas para os modelos que trabalham com pacotes, o que não impede a analogia com fluidos.

A seguir são apresentadas descrições de duas destas técnicas, as mais básicas, conhecidas como CP e CS. Um resumo de todas as técnicas apresentadas em [41, 30] pode ser encontrado no apêndice A.

Particionamento Completo (CP - Complete Partitioning)

Neste esquema a fila é permanentemente particionada em N filas menores, uma para cada fluxo de serviço. Não existe compartilhamento do recurso espaço em fila, entretanto nenhum fluxo é prejudicado por ações dos demais.

Esta técnica não faz um bom aproveitamento do espaço total do *buffer*. Entretanto garante justiça, robustez e com isto a qualidade de serviço.

Compartilhamento Completo (CS - Complete Sharing)

Neste esquema a fila é integrada e um pacote que chega é aceito se houver espaço disponível. Nesta disciplina existe o máximo aproveitamento do recurso espaço em fila, no entanto um fluxo pode monopolizar o serviço se for mais ávido, exatamente como o exemplo apresentado na Figura 3.7.

3.3.4 Roteamento

Uma das possíveis funcionalidades de um nó de rede é o roteamento de pacotes. Como, em geral, esta funcionalidade ocorre junto às filas, o assunto é abordado nesta seção. A idéia básica de um roteador é orientar os fluxos de entrada para seus devidos canais de saída.

Seja o seguinte nó de rede mostrado na Figura 3.8. O nó é representado por um retângulo e possui canais físicos de entrada E_i e saída S_i. Também estão representados os fluxos de entrada a_i e saída d_i. Neste exemplo de roteamento, os fluxos a_1 e a_2 chegam no nó através do canal físico E_1, assim como a_3 chega através de E_2 e $(a_4, \ldots, a_6)$ chegam através de E_3. Percebe-se no entanto, que independentemente do funcionamento interno, ou seja, das disciplinas de atendimento e do gerenciamento da fila, os fluxos d_i saem do nó por algum canal físico estipulado. No exemplo supra citado, o fluido 1 sai através do canal S_2. Assim como os fluxos 3 e 4 fluem através de S_1, 2 e 5 fluem através de S_3 e 6 sai por S_2.

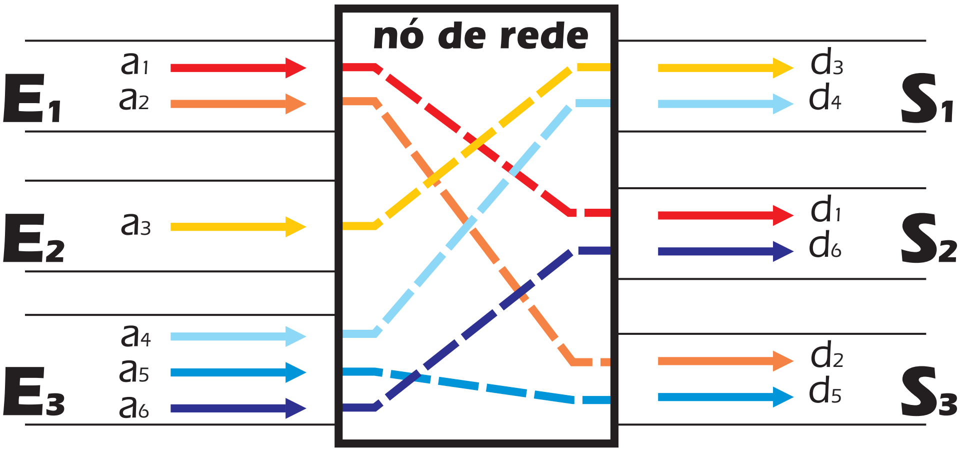


Figura 3.8: Roteamento Dentro do Nó de Rede

O roteamento de interesse neste trabalho é simplesmente o ato de redirecionar um fluxo de entrada a um canal específico de saída. Este redirecionamento pode ocorrer de acordo com o estado interno das filas, ou pode ser fixo, fazendo com que a rota seguida por um fluxo de fluido represente, por exemplo, um circuito virtual de uma rede ATM ou uma conexão, no nível da camada de aplicação, estabelecida pelo protocolo TCP em uma rede ethernet.

3.3.5 Resumo

As teorias apresentadas nesta seção 3.3 e no apêndice A descrevem a fila presente na maioria dos nós de rede físicos, bem como as disciplinas de atendimento e gerenciamento inerentes a esta. Também é descrito nesta seção, o esquema de roteamento que envolve o redirecionamento dos fluxos na(s) saída(s) da(s) fila(s). Os tópicos foram apresentados de forma genérica e servem de base para a modelagem de objetos de rede que se caracterizam como nó, tais como os citados no início desta seção. No capítulo 4 estão descritos alguns destes componentes que foram modelados durante o desenvolvimento desta tese.

3.4 Canais de Comunicação

O objetivo é representar, através deste tipo de objeto, o meio físico de comunicação entre dois nós. Como exemplos de meios cita-se: cabos coaxiais, fibras óticas, fios do tipo par trançado, canais de rádio frequência, dentre outros. Para efeitos de modelagem, apenas a característica tempo de propagação t_p é levada em consideração.

Assim sendo a única tarefa desempenhada pelo canal de comunicação é a inserção de um tempo de retardo no fluxo que está por este passando. Em outras palavras, uma mudança de taxa ocorrida em um tempo t na entrada de um canal, só será observada em seu outro extremo no tempo $t + t_p$. A Figura 3.9 mostra um componente canal de comunicação.

Figura 3.9: Canal de Comunicação

3.5 Reguladores de Tráfego

Outro componente de importância na estrutura de uma rede de computadores é o regulador de tráfego (*Leaky Bucket*). O regulador é um mecanismo que tem por finalidade o policiamento e a moldagem do tráfego que por ele atravessa. O esquema de um FLB *Fluid Leaky Bucket* é apresentado na Figura 3.10, e a descrição do seu funcionamento é dada a seguir:

Existem 3 parâmetros associados a um regulador:

- A capacidade do *buffer* de fluido - Onde o fluido, que representa o tráfego de

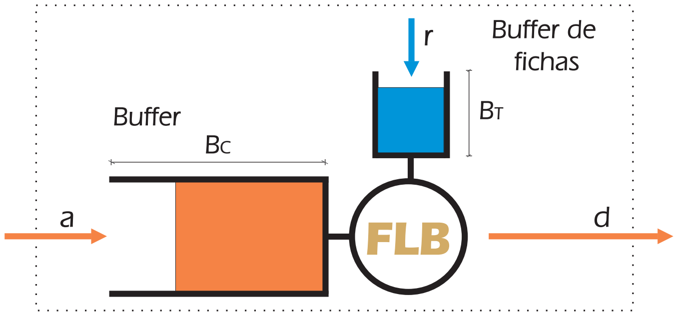


Figura 3.10: Regulador de Tráfego de Fluido (Fluid Leaky Bucket)

rede, fica armazenado caso não possa ser transmitido imediatamente. Equivalente ao *buffer* de pacotes em um regulador tradicional. Representada pela variável B_C.

- A capacidade do *buffer* de fichas - Onde o fluido proveniente da geração de fichas (*tokens) fica armazenado. Representada pela variável B_T.*

- *A taxa de geração de fichas - Define a quantidade de fluido que é gerado para o* buffer *de fichas por unidade de tempo. Representada por r.*

Além das variáveis acima apresentadas, considera-se $a(t)$ como a taxa de fluido que entra no regulador, $d(t)$ como a taxa de fluido que sai, $q(t)$ a quantidade de fluido armazenada no *buffer* e $f(t)$ a quantidade de fichas armazenada no *buffer* de fichas.

Para que uma unidade de fluido contida no *buffer* deixe o regulador, uma quantia igual de fluido contido no *buffer* de fichas precisa ser consumida. Desta forma o *buffer* só vai armazenar fluido caso a taxa de chegada $a(t)$ seja maior do que a taxa de geração de fichas e o *buffer* de fichas fique completamente vazio. Da mesma forma que o *buffer* de fichas só pode acumular fluido proveniente da geração de fichas se $a(t) < r$. O FLB opera de forma que $q(t).f(t) = 0$ para todo tempo t.

A dinâmica do regulador de tráfego de fluido é dada de forma precisa, pelas seguintes equações:

$$q(T_{n+1}) = [(a(T_n) - r)\mathbf{1}(0 < q(t) < B_C) + (a(T_n) - r)^+\mathbf{1}(q(t) = 0) +$$

$$+(a(T_n) - r)^{-}\mathbf{1}(q(t) = B_C)] \times (T_{n+1} - T_n) + q(T_n) \quad (3.12)$$

$$f(T_{n+1}) = [(r - a(T_n))\mathbf{1}(0 < f(T_n) < B_T) + (r - a(T_n))^{+}\mathbf{1}(f(T_n) = 0) +$$
$$+(r - a(T_n))^{-}\mathbf{1}(f(T_n) = B_T)] \times (T_{n+1} - T_n) + f(T_n) \quad (3.13)$$

$$d(t) = a(t)\mathbf{1}(q(t) = 0) + r\mathbf{1}(q(t) > 0) \quad (3.14)$$

onde, T_n representa a n-ésima transição de $a(t)$, $(y)^{-} = min(0, y)$, $(y)^{+} = max(0, y)$ e $\mathbf{1}$ é a função indicadora.

De acordo com o funcionamento deste mecanismo, todo o fluxo de tráfego que atravessa o regulador, é policiado, de forma que as rajadas (*bursts*) passam de forma controlada. Uma rajada que tem uma taxa de pico maior que a taxa de geração de *tokens* r passa pelo regulador às custas do consumo do *buffer* de fichas, contudo o tamanho máximo da rajada está limitado ao tamanho do *buffer* de fichas e sabe-se que após o consumo do *buffer* de fichas, somente uma taxa r de fluxo vai passar diretamente à saída. Nota-se também que podem haver perdas de fluido na entrada do FLB.

Esta característica de policiamento, faz uma amortização no fluxo de tráfego, e tem uma importância vital na garantia da qualidade de serviço. O uso do regulador de tráfego em conjunto com uma fila de serviço GPS, garante um tempo máximo de espera na fila e desta forma pode ser usado como garantia em um controle de admissão. Esta garantia é descrita em [32, 26] e é apenas um exemplo de utilização do *leaky bucket* na garantia de QoS.

Por um outro prisma, pode-se dizer que o *leaky bucket* efetua a moldagem do tráfego (*traffic shaping*) que por ele passa, como ilustrado na Figura 3.11.

O tráfego de saída do FLB encontra-se sempre dentro dos limites, onde tanto o inferior quanto o superior são determinados pela inclinação da taxa r de geração de fichas (*tokens*) com a diferença de que o superior é acrescido do tamanho do *buffer* de fichas B_T. A este comportamento, que modifica o tráfego, deixando-o sempre dentro dos limites denomina-se moldagem de tráfego.

Reguladores de tráfego, assim como as disciplinas FIFO e GPS, podem ser en-

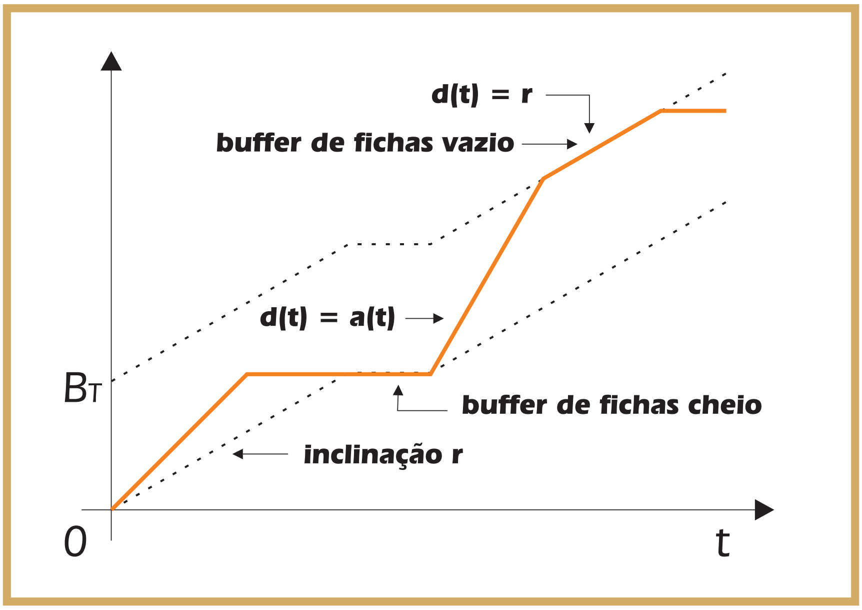


Figura 3.11: Tráfego Moldado por um Regulador de Tráfego de Fluido

contrados em roteadores como por exemplo, séries 1600,1700, 2500, 2600, 7500 a AS5300 da Cisco Systems [7, 8] ou no ASX-1000 da Fore Systems [15]. Em se tratando de reguladores, estes podem ainda ser encontrados em implementações de software em aplicações que necessitem policiamento e/ou moldagem, como por exemplo servidores (camada de aplicação) com garantia de QoS.

3.6 Aspectos Relativos ao Custo Computacional

A grande vantagem da técnica de fluido está no ganho em relação ao custo computacional que esta pode de fato prover. Este ganho se dá pelo fato de que um número reduzido de eventos precisa ser tratado para que um mesmo sistema seja analisado. Como visto no início deste capítulo, existe um conjunto de modelos onde o uso da técnica se torna vantajoso. Neste encontram-se justamente aqueles que possuem taxas de transição de estados das fontes ordens de grandeza menores que as taxas de geração de pacotes. Os modelos que descrevem as redes de alta velocidade atuais se encaixam neste perfil, e compõem apenas um exemplo que demonstra a utilidade da técnica.

Trabalhos como [2, 1], fazem uma análise de quando cada método é mais eficiente (pacotes x fluido), levando em consideração o número de eventos a serem processados. No entanto, existem alguns pontos que precisam ser considerados, como por exemplo o fato de que os eventos de fluido podem ser mais custosos de serem tratados. De acordo com a teoria de fluido, eventos podem ser escalonados, e por motivos peculiares pode ser que alguns destes venham a ser retirados da fila de eventos. A figura 3.5 mostra um exemplo onde existe a necessidade de que ocorra o desescalonamento de um evento. O processamento necessário para verificar a existência de um evento escalonado, e retirá-lo da fila de eventos, ou até mesmo reescaloná-lo para outro tempo, onera o custo do tratamento de eventos. Entretanto este custo é muito baixo se for considerado o custo de processamento para o tratamento completo de um evento. Assim sendo, este problema não invalida os benefícios desta forma de simulação.

Outro fator que deve ser observado, contudo previsto nos estudos [2, 1], é que em certos casos uma única mudança de taxa de fluido pode gerar 2 eventos. Este caso pode ser exemplificado pela ocorrência de um evento do tipo t_0 (quando um recipiente de fluido esvazia). Nesta situação uma mudança de taxa na entrada de uma fila, tratada pela ocorrência de um evento, ainda gera um segundo evento, que é necessário para que a nova taxa de saída seja calculada, no momento que o recipiente esvaziar.

Porém existe um fator que pode degradar bem mais o desempenho de uma simulação de fluido. É o chamado *ripple effect*, mencionado pela primeira vez em [16]. Este efeito é ocasionado por uma onda de mudanças de taxas que pode se propagar por todo o sistema. De forma a exemplificar o problema, apresenta-se uma rede

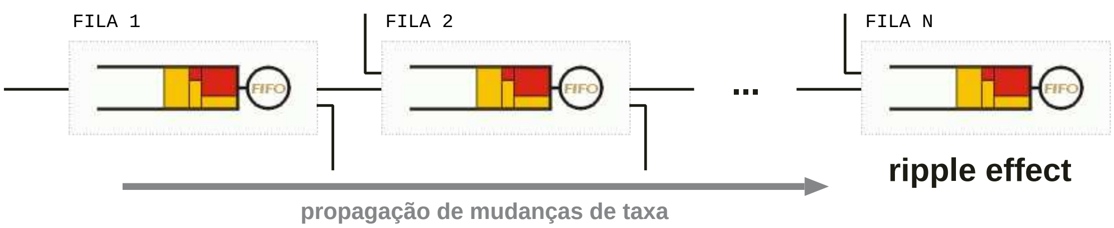


Figura 3.12: *Ripple Effect*

cuja topologia é descrita na figura 3.12. Considerando-se que uma mudança em alguma taxa de entrada na FILA 1 pode ocasionar alterações nas taxas de saída da mesma, observa-se que a FILA 2 será notificada desta modificação. O efeito pode ser semelhante na FILA 2 e ocasionar modificações nas filas que se seguem, formando uma onda que se propaga em direção ao final da rede.

Para certas mudanças de taxa, o efeito pode não aparecer, ou aparecer e não se propagar indefinidamente. No entanto, em modelos onde há realimentação nas filas, ele pode ser bastante prejudicial ao desempenho. Apesar da dificuldade em se avaliar o impacto do efeito, [1] mostra o seu comportamento para casos específicos, inclusive descreve que mesmo em modelos com realimentação, o sistema atinge a estabilidade, apesar de ter o desempenho bastante agravado.

Uma das técnicas utilizadas para diminuir o *ripple effect* é a agregação de fluxos. A técnica consiste em isolar o fluido de interesse e agregar os remanescentes em um único fluxo. Quando os fluxos estão agregados, uma única mudança de taxa pode

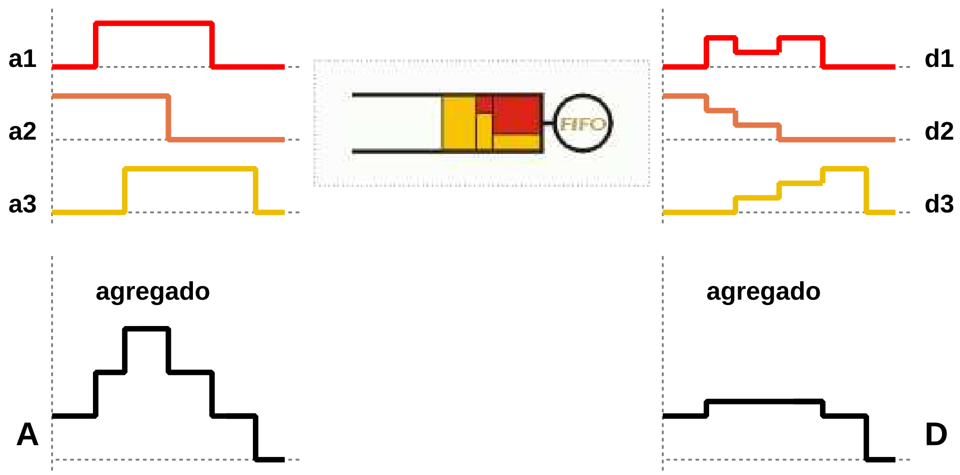


Figura 3.13: Agregação de Fluxos

representar diversas mudanças nos fluxos individuais. A figura 3.13 exemplifica o benefício da agregação de fluxos de fluidos. No exemplo, três fluxos $a1$, $a2$ e $a3$ alimentam uma fila FIFO, e $d1$, $d2$ e $d3$ representam suas respectivas saídas após o serviço. Nota-se que o número de mudanças de taxa do processo agregado é igual a soma do número de mudanças de cada fluxo individual, portanto não há economia, neste exemplo, em relação a agregação de fluxos de entrada. Entretanto, no processo de saída, considerando-se os fluxos individuais, ocorrem 11 mudanças de taxas, enquanto no agregado ocorrem apenas 3. Além da redução do número de eventos a serem processados, este tipo de técnica diminui efetivamente a propagação do efeito *ripple*, já que muitas mudanças no fluxo de entrada de uma fila são absorvidas e não geram mudanças de taxa de saída. Salienta-se que após a agregação de fluxos, não se pode mais diferenciar os fluxos agregados. Por isto, a importância de deixar isolado o fluxo, ou fluxos, de interesse.

Em [1], o número de eventos de uma simulação de pacotes é comparado com simulações de fluido, com e sem agregação de fluxos. O modelo estudado tem várias filas em série (*tandem network models*), e é semelhante ao apresentado na figura 3.12. O estudo mostra que para simulação de fluido com fluxos agregados, o comportamento em relação ao número de eventos é linear e assintoticamente igual a simulação de pacotes. Assim sendo a simulação de fluido é sempre mais rápida que a de pacotes para a resolução deste tipo de modelo, o que não seria verdade se não houvesse agregação de tráfego. O gráfico 3.14 relaciona a taxa de eventos que

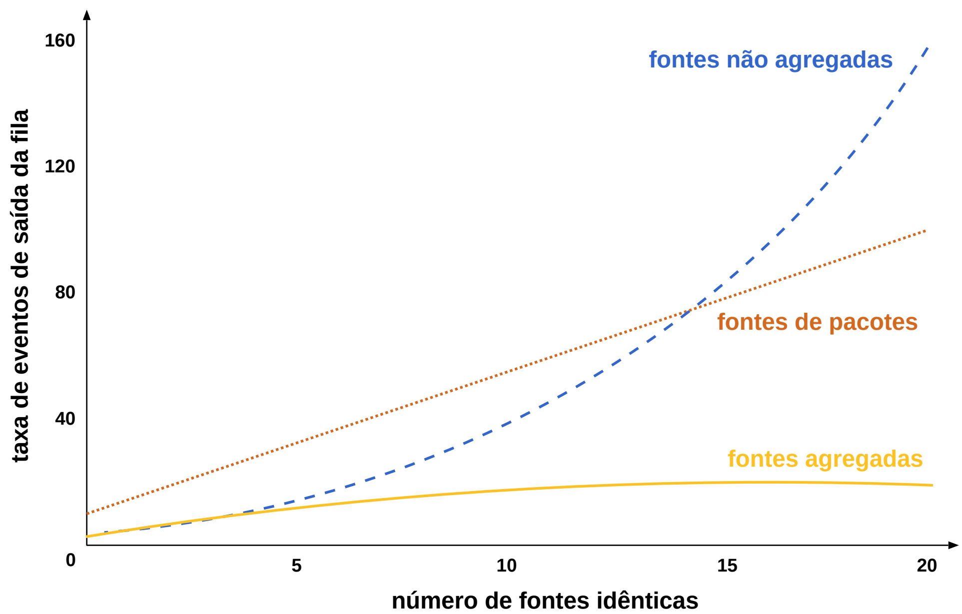


Figura 3.14: Taxa de Eventos para Fluxos Agregados e Não Agregados

devem ser processados na simulação de pacotes, de fluido com tráfego agregado e não agregado. Neste observa-se que a simulação de fluido com tráfego agregado é sempre mais vantajosa em termos de número de eventos a serem processados que as demais alternativas, em se tratando de modelos de rede com filas em série (*tandem network models*).

As filas com disciplinas FIFO são bastante afetadas pelo *ripple effect*. No entanto este efeito é amenizado quando se usa a disciplina GPS. De acordo com as características do GPS, há um certo grau de isolamento entre os fluxos, e é justamente isto que o dá uma certa imunidade ao efeito de propagação. Enquanto no FIFO, uma mudança de taxa de um fluxo de fluido que possua líquido no *buffer* provoca mudanças de taxas em todos os fluxos que também possuam fluido no *buffer*, no GPS, uma mudança de taxa, em condição semelhante (mas sem que o *buffer* esteja cheio), não altera a saída de nenhum fluxo se o fluxo em questão já está usando sua capacidade nominal. Ou então, altera somente o próprio fluxo se a banda usada for inferior a nominal e os que usavam a sua banda excedende.

Entretanto, contrastando com o benefício do isolamento, está a realocação de banda provinda do GPS, quando fluxos que chegam a taxas maiores que sua banda nominal utilizam a banda excedente de fluxos que não estão utilizando sua banda nominal por completo. Mas este efeito contrário provoca, em muitos casos, menor efeito de degradação do que os benefícios do efeito de isolamento, fazendo com que a simulação da política GPS seja, em geral, mais rápida que a da FIFO. Um fator que amplifica esta discrepância em relação ao número de eventos processados é a carga da fila, onde quanto menor a carga, menor o número de eventos processados por uma fila GPS e maior a diferença entre o número de eventos que devem ser processados entre GPS o FIFO.

3.7 Aspectos Relativos à Precisão das Medidas de Interesse

Os aspectos apresentados na seção anterior demonstram características relativas ao custo computacional da técnica de fluido, no entanto, sabe-se que esta técnica representa uma abstração de mais alto nível em relação às técnicas de modelagem tradicionais, portanto alguns pontos referentes à precisão das medidas de interesse, assim como quais podem ser obtidas, devem ser levados em consideração.

É sabido que para modelar um sistema é necessário que se faça várias simplificações, e que apenas as características mais importantes ou de real interesse devem ser levadas em consideração. De forma idêntica, os modelos de fluido são concebidos, e são boas aproximações, principalmente quando se trata de situações onde trafegam enormes quantidades de informações discretas (exemplo: pacotes). Nestes casos, onde há muitas partículas, pode-se olhar o sistema sob um ponto de vista mais distante, como se as partículas fossem puntiformes, e considerar-se então, que o sistema possui informação contínua, ao invés de discreta, trafegando pelo seu interior.

Grande parte das medidas de interesse que podem ser obtidas através da simulação de pacotes, também pode ser obtida na simulação de fluido, como por exemplo: tamanho médio da fila, taxa de perda, utilização, capacidade (*throughtput*), tempo de resposta, etc... Medidas como estas podem ser facilmente calculadas se for considerado que o espaçamento entre os pacotes é constante. O volume médio de fluido em uma fila descreve seu tamanho, o fluido perdido denota os pacotes perdidos e assim por diante. Entretanto, deve-se lembrar que o tratamento dos dados em forma de fluido pode diferir ligeiramente da realidade. Como exemplo cita-se o caso onde uma fila está cheia e a taxa de entrada passa a ser idêntica a capacidade de serviço. Na realidade, e nos modelos de pacotes, pode haver perda, justamente por causa da diferença de espaçamento entre pacotes. No entanto em um modelo de fluido a perda é zero nesta situação. Cabe salientar que certas medidas como o *jitter*, não podem ser obtidas através deste tipo de modelo.

De forma a avaliar o impacto das abstrações na qualidade das medidas de interesse, [31] fez um estudo, baseado em alguns modelos, que demonstrou a potencialidade da técnica. Este estudo, baseado em fontes e filas-servidoras, sugeriu que na maioria dos casos, a relação custo X benefício é favorável ao uso da técnica de fluido. O trabalho focou a comparação entre modelos de pacotes e de fluido, em diferentes escalas de tempo. Escalas $x = [1, 10, 100, 1000]$ foram usadas, onde o número médio de pacotes gerados em cada estado *on* das fontes era definido por aproximadamente $10x$.

Para modelos de fluido, mudanças na escala de tempo praticamente não alteram o número de eventos que devem ser tratados, enquanto que na de pacotes este cresce significativamente. Além disto, para maiores escalas de tempo, onde um número grande de informações é gerado, os modelos de fluido aproximam os de pacotes, e consequentemente as medidas tendem ao mesmo valor. Apenas para exemplificar, a figura 3.15 mostra os resultados obtidos no estudo, realizado sobre um modelo de uma única fila alimentada por uma fonte *on-off*. O gráfico denota que o erro

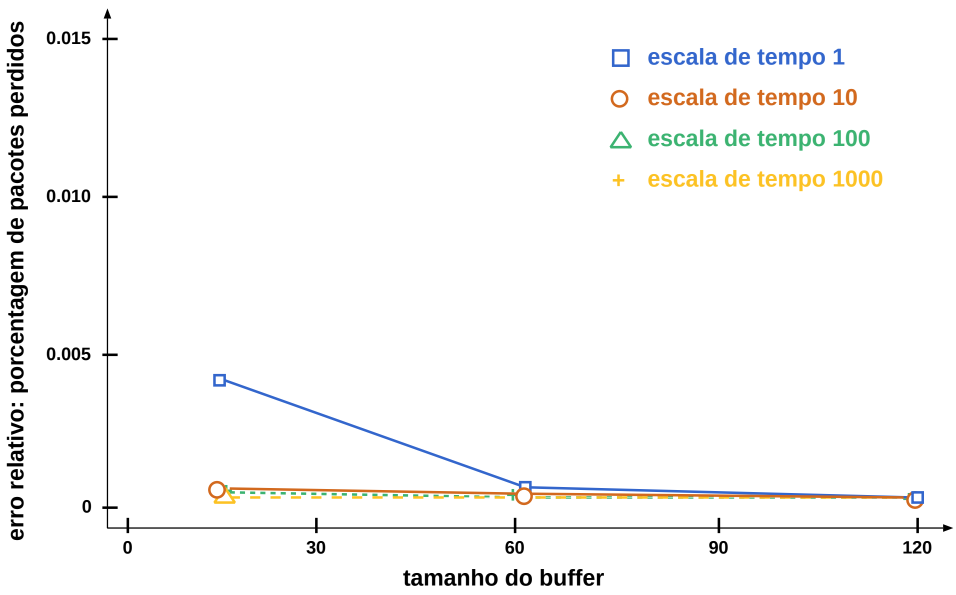


Figura 3.15: Erro Relativo para uma Fila com uma Fonte *on-off*

relativo tende a zero à medida que o tamanho do buffer cresce. Para escalas de tempo maiores, o erro já é muito pequeno mesmo para *buffers* pequenos. Este fato se justifica pois para *buffers* maiores, existe uma maior quantidade de níveis discretos que aproximam o comportamento ao contínuo, assim como a escala de tempo também contribui nos níveis de discretização, além de que nos *buffers* maiores ocorrem menos perdas, que também auxiliam na imprecisão das medidas.

[31] conclui que para a maioria dos casos o erro relativo entre os modelos de pacotes e de fluidos é muito pequeno, de até 5%, e que em casos piores, o erro não passa de 13%. Ainda assim, sabe-se que apenas alguns modelos foram criados, e que outros casos precisam ser analisados antes que se possa afirmar sobre a precisão destas medidas. Para complementar, o trabalho mostrou ganhos de até 2116 vezes

em termos de tempo de simulação, para escalas de tempo $x = 1000$, que afirmaram a eficácia e o custo X benefício favorável ao uso da técnica de fluido.

No capítulo 5 é apresentado um modelo de rede ATM, na seção 5.2.1, bastante simples, que compara a simulação de pacotes com a de fluido, mostrando a proximidade dos resultados e o enorme ganho em termos de custo computacional.

Capítulo 4

Simulação de Fluido no Ambiente de Modelagem do TANGRAM-II

ESTE capítulo descreve o funcionamento da simulação de fluido dentro do ambiente de modelagem TANGRAM-II, assim como os recursos implementados ao longo deste trabalho, que possibilitaram a realização deste tipo de simulação.

4.1 Princípios de Funcionamento

Em uma simulação tradicional, eventos a serem executados são colocados em uma lista ordenada por tempo. A simulação progride através da execução dos eventos, onde as ações associadas a estes são realizadas e novos eventos podem ser habilitados. O intervalo de tempo de um evento, desde o momento em que é habilitado até o seu disparo, é determinado por uma distribuição que é especificada no modelo.

De forma a ilustrar este processo, a Figura 4.1 mostra um modelo bastante simples, com uma fonte *on-off* e uma fila de serviço exponencial. Neste modelo tem-se eventos de mudança de estado da fonte, eventos de geração de pacotes e eventos de serviço de pacotes na fila. No disparo de quaisquer destes eventos, uma rotina de tratamento é executada e novos eventos são escalonados (inseridos no escalonador de eventos do simulador). O código das rotinas executadas e os eventos

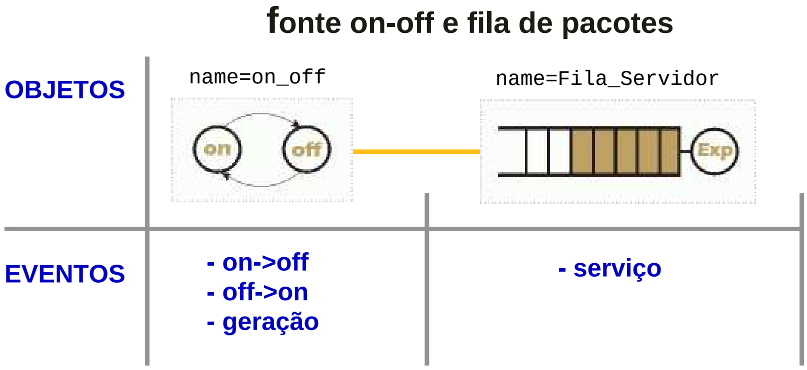


Figura 4.1: Eventos em um Modelo Tradicional

inseridos no escalonador definem o comportamento do modelo.

Na simulação de fluido, além dos eventos tradicionais, existem novos eventos relacionados com o comportamento dos fluidos. A Figura 4.2 mostra um modelo equivalente ao apresentado acima, onde salienta-se a necessidade da especificação de, por exemplo, eventos como o esvaziamento do *buffer* e o enchimento deste por completo. Estes eventos são diferentes dos tradicionais e são definidos por equações

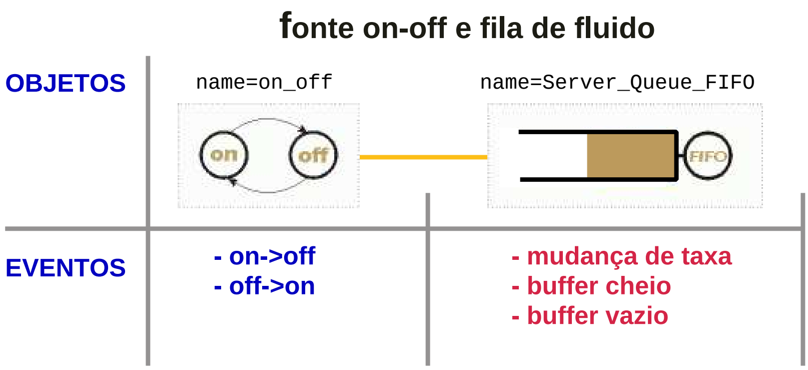


Figura 4.2: Eventos em um Modelo de Fluido

matemáticas, como demonstrado no capítulo 3. Para o escalonamento destes eventos faz-se necessária uma monitoração do nível de fluido do *buffer* e previsões de tempo de disparo para quando o nível do fluido atingir certos valores.

Objetivou-se neste trabalho, a construção de recursos que possibilitassem a manipulação destes eventos de fluido, de forma genérica e transparente para os usuários. Mesmo sendo perfeitamente possível a implementação de mecanismos que descrevem por completo o comportamento dos objetos de fluido apresentados no capítulo anterior de forma direta no código do simulador, optou-se pela criação de um paradigma genérico e de recursos que formassem uma base para a implementação de tais objetos. Ou seja, em vez de implementar toda a funcionalidade das equações de fluido (eventos de fluido, disciplinas de atendimento e etc.) para um subconjunto de objetos (por exemplo, diferentes tipos de filas) internamente ao simulador, optou-se pela implementação de um conjunto genérico de funcionalidades, onde apenas o tratamento de eventos de fluido está embutido no simulador, de forma a prover suporte à criação de modelos de fluido complexos.

Assim sendo, parte das funcionalidades dos objetos de fluido ficara fora do código do simulador, encontrando-se dentro do código dos modelos, de forma a prover flexibilidade e poder, uma vez que, por exemplo, novas disciplinas de atendimento e gerenciamento de espaço podem ser implementadas apenas com modificações no código dos objetos.

O paradigma genérico que serve de base para a construção dos modelos de fluido utiliza a idéia de representar os reservatórios de fluido através de recompensas. Desta forma, recompensas acumulam valores que indicam o volume de fluido armazenado em um determinado recipiente. No final do capítulo 2 há um exemplo onde um *buffer* é definido através de uma recompensa de impulso. Nele, a recompensa chamada *buffer* é definida de forma a receber uma unidade a cada pacote gerado e perder uma unidade a cada pacote servido. Também são especificados limites $(0, \infty)$. Com estas características sabe-se que um *buffer* infinito é modelado com exatidão.

Entretanto, a simulação de fluido manipula a informação de forma contínua, o que sugere a utilização de recompensas de taxa. Assim sendo, os valores instantâneos IR, definem o quanto de fluido vai ser acumulado por unidade de tempo. Valores positivos fazem com que o *buffer* cresça em volume e valores negativos fazem com que diminua em volume. Da mesma forma, os limites precisam ser especificados, ou seja, para a modelagem de um *buffer* de tamanho finito, especificam-se limites $(0, B)$. Através do uso destas recompensas de taxa acumuladas, pode-se modelar os recipientes de fluido, necessários para a construção dos objetos descritos no capítulo anterior.

Outro princípio importante para o paradigma é a definição de que as mensagens da ferramenta, devem transportar valores de taxas de fluxos de fluido, e a troca de mensagens, passa a representar mudanças nessas taxas. Assim sendo, uma mensagem enviada de um componente de rede A para outro B, especifica o quanto de fluido passa a escoar por unidade de tempo de A para B a partir do recebimento da mensagem.

Com o uso das recompensas de taxa acumulada para a descrição dos recipientes de fluido e o uso das mensagens para a notificação de mudanças de taxa, tem-se a base genérica para a implementação de objetos de simulação de fluido. Entretanto, a ferramenta TANGRAM-II, do modo como estava concebida, não suportava o uso do paradigma criado, nem possuía recursos que possibilitassem a implementação dos objetos de fluido em questão. Logo, havia a necessidade de incorporação de novos recursos. A descrição destes, bem como a dos problemas encontrados, compõem o restante deste capítulo.

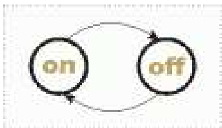
4.2 Recursos Implementados

De acordo com os princípios de funcionamento e com a necessidade de representar os componentes de rede apresentados no capítulo 3, surgiu a necessidade de criação e implementação de diversos recursos no módulo de simulação da ferramenta. Estes recursos formam a base genérica para a criação de modelos de fluido e implementam diversas funcionalidades dentre as especificadas pelas equações do capítulo 3. Em paralelo ao desenvolvimento dos recursos genéricos, os objetos de fluido foram trabalhados, de forma que implementassem as funcionalidades restantes, peculiares aos objetos. Ambas as implementações, internas ao simulador e o desenvolvimento dos objetos, são apresentadas a seguir e, sempre que possível, ilustradas com exemplos.

4.2.1 Mensagem Transportando Valor Contínuo

A primeira modificação ocorreu na estrutura da mensagem para que esta pudesse transportar um valor contínuo. Como dito, o módulo de simulação do TANGRAM é implementado em C++, portanto a modificação em questão fez com que a mensagem passe a carregar variáveis do tipo *double* ou *integer*. Assim sendo, a ferramenta ficou apta para o transporte das taxas de fluidos e com esta modificação as fontes de tráfego descritas em 3.2 puderam ser modeladas em sua plenitude. A Figura 4.3 mostra uma fonte de fluido *On-Off*.

```
name=fluid_on_off
```




```
Initialization=
  status     = 0
  OnOff_rate = 0.5
  OffOn_rate = 2
  port_out   = link1
  fluid_rate = 350

State_vars=
  status =

Messages=

Rewards=
```

```
Declaration=
  State Var
   Integer: status; /* 1 - ON; 0 - OFF */
  Const
   Float: OnOff_rate, OffOn_rate, fluid_rate;
   Port:  port_out;

Events=
 event= On_Off (DET, OnOff_rate)
 condition= (status == 1)
 action= {
           msg(port_out,all,0);
           set_st("status",0);
         };

 event= Off_On (DET, OffOn_rate)
 condition= (status == 0)
 action= {
           msg(port_out,all,fluid_rate);
           set_st("status", 1);
         };
```

Figura 4.3: Modelo de Fonte On-Off de Fluido no Ambiente de Modelagem

Neste objeto, chamado *fluid_ on_ off*, existe apenas 1 variável de estado, *status*, que indica o estado da fonte. Dois eventos foram especificados, *On_ Off* que desliga a fonte e *Off_ On* que a liga. Na ação destes eventos é enviada uma mensagem através da porta de saída denominada *port_ out* e é executado o comando *set_ st* que atualiza a variável de estado *status*. A mensagem enviada carrega em seu corpo o novo valor de transmissão.

Outras fontes são descritas no próximo capítulo e também podem ser encontradas no apêndice B.

4.2.2 Comandos: get_ir - set_ir - unset_ir

Até então, os valores instantâneos acumulados por uma recompensa eram definidos no momento em que a própria recompensa era definida, dentro do atributo *Rewards*, através dos pares *condição / valor*, tal qual como descrito na seção 2.1.4. No entanto, para representar os recipientes de fluido, os valores instantâneos precisam reagir de acordo com as mudanças de taxas dos fluidos que chegam ao recipiente. Como as mudanças de taxa acontecem durante o processo de simulação, e os valores instantâneos são calculados a partir dos novos valores das taxas, houve a necessidade de criação de comandos que definem novos valores instantâneos para as recompensas que representam os recipientes. A partir desta necessidade, foram criados 3 comandos:

- get_ir(*nome_da_recompensa*) - Efetua a leitura do valor instantâneo IR da recompensa especificada como argumento.

- set_ir(*nome_da_recompensa*, *valor*) - Quando executado, define um novo valor IR para a recompensa especificada, desabilitando a associação automática *condição / valor*. Antes do uso deste comando, os valores instantâneos eram definidos pelo par *condição / valor*, onde $IR = valor$ se *condição* for verdadeira. Após o uso do set_ir, o valor instanâneo da recompensa fica valendo indefinidamente, pois o mecanismo de associação automática se mantém desabilitado. O valor instantâneo só pode ser mudado através do uso de uma nova chamada de set_ir ou através de unset_ir.

- unset_ir(*nome_da_recompensa*) - Habilita a associação automática da recompensa especificada, fazendo com que o valor de IR passe a ser calculado automaticamente de acordo com os pares *condição / valor* especificados para a recompensa.

Estes comandos foram criados para o uso exclusivo do simulador e podem ser executados de qualquer lugar do código da ação de eventos ou mensagens.

O exemplo a seguir mostra o uso dos comandos: Toma-se como base que um objeto de fluido receba uma mensagem notificando mudança de taxa da fonte para 12Mb/s. Sendo a capacidade de serviço do objeto em questão igual a 10Mb/s, o recipiente deve acumular 2Mb/s. Dando ênfase apenas aos comandos aqui apresentados, e supondo *Mega bits* por segundo como unidade de informação, a linha de comando A especifica o novo valor IR da recompensa recipiente.

```
Messages=
 msg_rec = porta_de_entrada
   action= {
             ...
             set_ir( recipiente, 2 );      /* A */
             ...
             ...
             valor = get_ir( recipiente ); /* B */
             ...
             unset_ir( recipiente );       /* C */
           };
```

Os comandos B e C são meramente ilustrativos, e neste exemplo, a variável *valor* recebe 2 na linha B e em C o mecanismo de associação automática volta a vigorar novamente.

Nota: Os comandos *set_ir*, *unset_ir* e *set_cr* têm comportamento semelhante ao das variáveis de estado, em relação ao tempo. Como citado anteriormente, no capítulo 2, o valor de uma variável de estado só é atualizado ao final do código da ação, já que este instante de finalização caracteriza a mudança de estado. No caso dos comandos supracitados, mesmo sendo permitida sua utilização em qualquer parte do código, seu efeito em se tratando de atualização de recompensas, somente será notado ao final da execução do código da ação. Desta forma, qualquer mudança proveniente destes comandos somente terá efeito no cálculo das recompensas do estado seguinte. Cabe salientar que se estes comandos forem utilizados de forma replicada no código, prevalecerá o efeito do último comando executado.

4.2.3 Evento REWARD_REACHED

A ferramenta dispunha de várias distribuições, com o qual pode-se especificar diversas formas de espaçamento de tempo entre eventos. No entanto, não existia um mecanismo que pudesse notificar quando o valor acumulado de uma recompensa atingisse um valor específico. Na verdade, faz-se necessário o monitoramento dos valores acumulados, pois de acordo com a teoria dos fluidos, precisa-se saber quando o nível de fluido do reservatório atinge o valor zero, ou às vezes quando este nível atinge B, representando respectivamente o esvaziamento e enchimento completo do recipiente. Nestes casos ações devem ser tomadas, como por exemplo, no momento que um reservatório esvazia, a taxa de saída deve ser alterada, recebendo o valor igual ao da taxa do fluxo de entrada.

Com o intuito de inserir este mecanismo na ferramenta, criou-se um pseudo evento denominado *REWARD_REACHED*. Este evento especial escalona disparos para o instante em que o valor acumulado CR de uma recompensa alcança um determinado limite. Foram definidos 2 símbolos, /\ e \/ que ao serem usados na condição do evento, definem se o disparo deve ocorrer no momento em que o CR da recompensa alcança um dado valor de baixo para cima (/\) ou de cima para baixo (\/).

Os testes são realizados através de sentenças especiais do tipo "*get_cr(recompensa) símbolo limite*" onde *get_cr(recompensa)* representa o valor acumulado de *recompensa*, *símbolo* é um dos caracteres \/ ou /\ e *limite* representa o valor a ser alcançado. O cálulo do tempo para o próximo disparo do evento é realizado através da seguinte expressão:

$$\Delta t = \begin{cases} \dfrac{CR_i - L_i}{|IR_i|} & \text{, se } CR_i \ > \ L_i \text{ e } IR_i \ < \ 0 \text{ e } \textit{símbolo} = \backslash/ \ , \\[2em] \dfrac{L_i - CR_i}{IR_i} & \text{, se } CR_i \ < \ L_i \text{ e } IR_i \ > \ 0 \text{ e } \textit{símbolo} = /\backslash \ , \\[2em] \infty & \text{, demais casos} \end{cases} \tag{4.1}$$

onde CR_i e IR_i representam, respectivamente, o valor acumulado e o valor instantâneo da recompensa especificada na sentença i, assim como L_i representa o limite desta sentença e *símbolo* é o caracter especial que define o sentido por onde o valor limite será alcançado (por cima ou por baixo).

A condição do evento REWARD_REACHED suporta a presença de vários testes

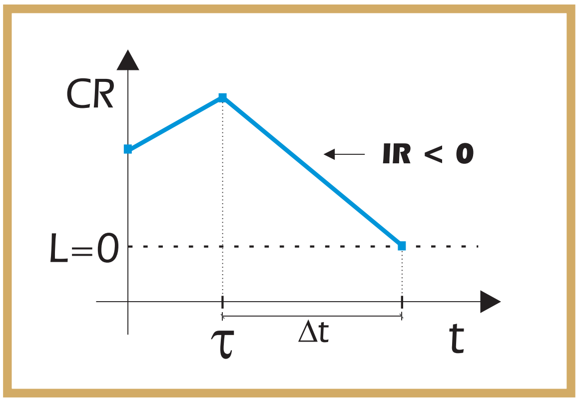


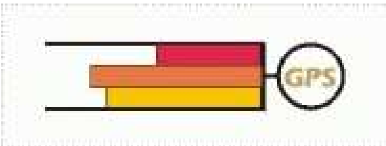
Figura 4.4: Cálculo do Tempo de Disparo do Evento REWARD_REACHED

baseados nos valores acumulados CR das recompensas, através das sentenças especiais "*get_cr(recompensa) símbolo limite* ", além de expressões aritméticas e testes sobre variáveis de estado e constantes, como ocorre nos demais eventos. No entanto existe uma restrição na qual não é permitido o uso de uma mesma recompensa mais de uma vez na condição de um evento. Caso exista a necessidade de observação de uma recompensa em diversos pontos, eventos diferentes devem ser criados, de forma que cada um deles possua somente uma sentença especial referenciando a recompensa em questão. Esta restrição é proveniente da implementação que utiliza uma estrutura de dados agregada ao evento para análise de cada recompensa, onde duas sentenças referenciando a mesma recompensa, causariam estados de inconsistência.

A condição do evento é avaliada em duas etapas: inicialmente todas as sentenças especiais são consideradas verdadeiras e a condição como um todo é analisada. Se a condição inteira for falsa, nenhum evento é escalonado. Caso seja verdadeira, o sistema analisa uma a uma as sentenças especiais e extrai o valor mínimo de Δt dentre todas elas, escalonando a ocorrência do evento para um tempo futuro, distante Δt do tempo presente de simulação. Salienta-se que, em certos casos, o evento pode ser escalonado para um tempo infinito, o que em termos práticos, equivale ao não escalonamento. Outro ponto de observação é o comportamento das sentenças especiais em relação aos operadores lógicos. Para estas sentenças não existe diferença entre os operadores OU (||) e E (&&), já que inicialmente todas as sentenças especiais são verdadeiras e no passo seguinte todas são analisadas, independentemente do operador, de forma que se possa obter o menor Δt. Tanto na sentença "*(get_ cr(fluido1) \/ 0) || (get_ cr(fluido2) \/ 0)" quanto na* "(get_cr(fluido1) \/ 0) && (get_cr(fluido2) \/ 0)", as duas condições seriam analisadas de forma idêntica e o tempo de disparo do evento seria escalonado para o menor tempo de alcance, não importando o operador utilizado.

A Figura 4.5 mostra um exemplo com uma fila de fluido GPS, onde o evento chamado t_0 está condicionado ao momento em que o valor acumulado de uma das recompensas, *fluido1*, *fluido3*, *fluido3* alcança o valor 0. No exemplo, sempre que

```
name=Server_Queue_GPS
```




```
Events=
 event= t0(REWARD_REACHED)
 condition=( (get_cr(fluido1)\/0) ||
             (get_cr(fluido2)\/0) ||
             (get_cr(fluido3)\/0) )
 action= {
           ...
         };
```

```
Rewards=
  rate_reward = fluido1
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido2
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido3
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
```

Figura 4.5: Evento REWARD_REACHED em uma Fila de Fluido GPS

um dos valores acumulados tocar o limite 0, o evento é disparado e conseqüentemente tem o código de sua ação executado.

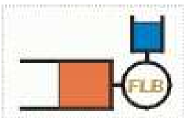
4.2.4 Inicialização do Valor Acumulado de uma Recompensa

Com os recursos implementados até o momento, o regulador de tráfego descrito em 3.5 poderia ser modelado quase que totalmente. No entanto um único ponto precisava ser resolvido. No modelo previsto para o regulador, o *buffer* de fluido inicia completamente cheio, entretanto no TANGRAM-II todas as recompensas acumuladas possuiam inicialmente o valor zero. Por isso foi criado um mecanismo que permite a inicialização de recompensas acumuladas.

Durante a especificação de uma recompensa, o usuário pode definir o seu valor

CR inicial através do prenchimento de *cr_init_val*. A figura 4.6 mostra o trecho do código do regulador, onde o *CR* da recompensa que representa o balde de fichas *TOKEN_BUCKET* é inicializado com o valor 100 (linha A). Caso a descrição da

```
name=Leaky_Bucket
```




```
Initialization=
 token_bucket_size=100
 ...
```

```
Rewards=

rate_reward = TOKEN_BUCKET
cr_bounds = 0,token_bucket_size
cr_init_val = token_bucket_size  // A
condition = (TRUE)
value = 0;
...
```

Figura 4.6: Inicialização do CR da Recompensa *TOKEN_BUCKET*

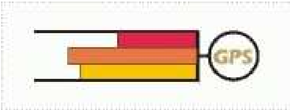
recompensa não contenha a atribuição de inicialização, seu valor é automaticamente preenchido com 0. O modelo completo do regulador encontra-se no apêndice B.

4.2.5 Variável de Estado Contínua e Vetor de Variável de Estado Contínua

Com o intuito de modelar os nós de rede descritos em 3.3, que são os componentes mais complexos dentre os estudados, diversos recursos foram projetados e implementados. O primeiro deles surgiu da necessidade de se armazenar os valores de entrada da fila $\vec{a}$, onde cada elemento representa a taxa de entrada de um fluxo de fluido. Foram criadas variáveis de estado do tipo contínuas de forma que os modelos pudessem ter espaço de estados contínuo. Com este novo recurso o TANGRAM passou a permitir modelagem com espaço de estados discreto e contínuo, ambos em cadeias de tempo discreto.

Também é possível a especificação de vetores de variáveis de estado contínuas, de forma análoga aos vetores de variáveis de estado discretas. O exemplo da figura 4.7 mostra mais um fragmento do objeto fila GPS, dando ênfase ao uso do recurso. A variável *afl[]* é declarada em A, no atributo *Declaration*. Esta deve ser inicializada em *Initialization*, como apresentado na linha B. Seu uso efetivo ocorre dentro do atributo *Messages*, onde ela armazena os valores das taxas de entrada de cada classe de fluido.

```
name=Server_Queue_GPS
```




```
Declaration=
 Const
   Integer: classes;
   ...

 State Var
   Float: afl[classes]; // A
   ...

Initialization=
  ...
  classes = 3

  afl = [0,0,0]  // B
  ...
```

```
Messages=
 msg_rec=port_in
 action= {

   float a[classes];/* taxa de chegada de cada fluido */    // C
   ...

   /*---- pega taxas de entrada antes da mudanca ---------------*/
   get_st( a,"afl[]" ); /* taxas de chegada */        // D
   ...
     // atualiza taxa de entrada que mudou, usando o conteúdo
     // da mensagem.

   /*---- distribuição da banda --------------------------------*/
   /*---- disciplina de atendimento: GPS -----------------------*/

   // cálculo das taxas de serviço usando o vetor a[i]
   ...

   /*---- atualização das recompensas (classes de fluido) ------*/
   ...

   /*---- atualiza taxas de chegada ----------------------------*/
   set_st( "afl[]",a );                               // E
 };
```

Figura 4.7: Uso de Variável de Estado Contínua em uma Fila de Fluido GPS

Seguindo a filosofia da ferramenta, um vetor local, neste exemplo chamado a, é especificado na linha C. O conteúdo da variável de estado *afl[]* pode ser recuperado através do comando *get_st(a, "afl[]")* encontrado na linha D, que efetua a cópia dos dados de *afl[]* para a. Neste ponto, a contém o vetor de entrada que antecede ao recebimento da mensagem que está sendo tratada. Assim sendo, a deve ser atualizado de acordo com a nova taxa recebida dentro do corpo da mensagem recém chegada. Em seguida, as taxas de saída podem ser calculadas, tomando-se como base o vetor de entrada a, os pesos das classes e a ocupação do *buffer* sendo que estes dois últimos estão omitidos no código do exemplo. Após o cálculo, resta a atualização dos IRs das recompensas e o armazenamento do vetor *afl[]* que ocorre na linha E.

4.2.6 Mensagem Transportando Vetores Inteiros e Contínuos

Além de transportar valores contínuos, fez-se necessário a implementação de mensagens que pudessem transportar vetores, contendo diversas informações numéricas. A partir desta, possibilitou-se que uma mensagem pudesse transportar um vetor de qualquer tamanho, tanto do tipo inteiro (int) quanto contínuo (double).

Uma das utilizações deste recurso é a indicação de qual fluido tem sua taxa alterada, onde diversos fluidos distintos são trabalhados em um mesmo objeto, como por exemplo em uma fila GPS. Neste caso, de forma a indicar quais fluxos de fluido tiveram mudança de taxa, mensagens são enviadas com vetores de duas posições, sendo que a primeira posição indica o número do fluxo de fluido e a segunda a nova taxa de escoamento.

O exemplo da Figura 4.8 enfatiza o uso do recurso. Dentro do atributo *Messages* do objeto *Server_ Queue_ GPS*, após o cálculo das taxas de saída *d*, existe um trecho de código responsável por enviar mensagens ao objetos posteriores, com a notificação de mudanças de taxas. Para isto, um vetor local *msg_ body[2]* é criado em A, em B sua primeira posição é preenchida com o número do fluido que teve alteração de taxa, em C a segunda é preenchida com o novo valor da taxa de saída e por fim, em D uma mensagem contendo estas informações é enviada ao objeto subseqüente. Por outro lado, o objeto *Server_ Queue_ GPS_ 2*, ao receber a mensagem, lê o


```
Messages=
 msg_rec=port_in
 action= {

   float msg_body[2];  // A

   ...

   // se a taxa do fluido 1 mudou
     msg_body[0] = 1;                          // B
     // sendo d[1] a taxa de saida do fluido 1
     msg_body[1] = d[1];                       // C
     msg( porta_saida, all, msg_body);// D

   // se a taxa do fluido 2 mudou
   ...

 };
```

```
Messages=
 msg_rec=port_in
 action= {

   float msg[2], a[classes]; // E
   int   indice;
   ...

   /*-- atualização da taxa de
        entrada que mudou.  ---*/

   get_msg_data( msg ); // F

   indice = (int) msg[0];         // G

   a[ indice ] = msg[1];          // H

   ...
 };
```

Figura 4.8: Transporte de Vetor em Mensagens (Fila de Fluido GPS)

conteúdo desta e atualiza o vetor de entrada $\vec{a}$ - em E, F, G, H - para em seguida proceder com suas demais tarefas, onde se encontra a de recalcular as novas taxas de saída do servidor. Caso houvessem mais objetos após o *Server_ Queue_ GPS_ 2*, o procedimento ocorreria de forma sucessiva, até que não houvesse mais modificações de taxas de saída ou até o último dos objetos. Este é o efeito de propagação denominado *ripple effect*.

Após a finalização desta implementação, tornou-se possível a modelagem do objeto *fila_servidor_GPS* descrito em 3.3.2, desde que utilizasse fila infinita ou com a disciplina de gerenciamento de *buffer* CP. As funções de roteamento descritas em 3.3.4 também foram cobertas em sua plenitude com o desenvolvimento deste recurso.

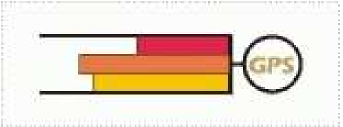
4.2.7 Soma de Recompensas Acumuladas e Instantâneas

O mais complexo dos recursos elaborados surgiu da necessidade de monitorar e controlar diversas recompensas de forma integrada. Para isto foi criado um mecanismo que controla a soma dos valores acumulados e instantâneos das recompensas de taxa. Com o auxílio deste controle tornou-se possível a modelagem por completo do objeto *fila_servidor_GPS* descrito em 3.3.2, utilizando qualquer disciplina de gerenciamento de *buffer* e com limite superior de tamanho da fila.

O mecanismo de controle implementado na forma de uma recompensa especial, denominada *rate_reward_sum*, permite a especificação de vários nomes de recompensas de taxa, através da qual irá acumular os valores CR e IR de todas elas. Além da monitoração destes valores, a *rate_reward_sum* exerce um efeito de controle sobre as recompensas associadas, no que diz respeito aos limites dos valores acumulados. Assim como os limites individuais sobre o CR de cada recompensa são respeitados, o limite especificado para a soma das recompensas também é controlado. A complexidade do mecanismo encontra-se justamente neste ponto. Para que a soma das recompensas seja condizente com a situação atual de cada recompensa individual, a partir do momento em que esta soma atingir um limite, faz-se necessário um controle refinado sobre a taxa de acumulo de cada uma das recompensas associadas.

A figura 4.9 exemplifica o uso da recompensa especial no objeto *fila_servidor_GPS*. Assim como as recompensas que definem cada fluxo de fluido, a recompensa especial, que representa o montante de fluido total do *buffer* compartilhado, é especificada dentro do atributo *Rewards*. No exemplo, uma soma denominada *buffer* é es-

```
name=Server_Queue_GPS
```




```
Declaration=
 Const
   Integer: classes, B;
   ...
Initialization=
  ...
  classes = 3
  B = 10000  // bits

  ...
```

```
Rewards=

  rate_reward_sum = buffer
    bounds  = 0,B
    rewards = fluido1 + fluido2 + fluido3

  rate_reward = fluido1
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido2
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido3
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
```

Figura 4.9: Soma de Recompensas Acumuladas (rate_reward_sum)

pecificada para manter o controle sobre outras 3 recompensas, $fluido1$, $fluido2$ e $fluido3$.

A figura 4.10 mostra dois exemplos simples, que ilustram o funcionamento do mecanismo.

Caso 1: Considera-se que os fluidos 1 e 2 possuem IR positivos de forma que a fila de fluido tem seu volume aumentado ao passar do tempo. Se nenhuma taxa mudar, *buffer* atingirá B em algum ponto no tempo, o que caracteriza a ocorrência de um evento t_B (*buffer* cheio). A partir deste ponto, aonde o limite da soma foi alcançado, as recompensas devem sofrer uma alteração em seu comportamento, de forma que a soma não ultrapasse o limite pré-estabelecido. Neste exemplo, os fluidos 1 e 2 apenas param de crescer, indicando que há perda de fluido para ambos os fluxos. Perda esta que pode ser calculada através da diferença do ângulo de inclinação do CR, entre seu crescimento natural e depois da atuação do mecanismo.

Caso 2: Entretando, situações complexas podem aparecer, como no exemplo em que os fluidos A e B têm IR positivos e C possui IR negativo, de forma que a soma seja positiva. Nesta situação, de forma a manter a soma *buffer* constante, os fluxos A e B devem sofrer mudanças de taxa de crescimento, já que C deve continuar com a mesma taxa de decréscimo. Nenhum fluido tem o CR congelado neste exemplo, já que o espaço liberado por C é ocupado pelos demais proporcionalmente a suas taxas

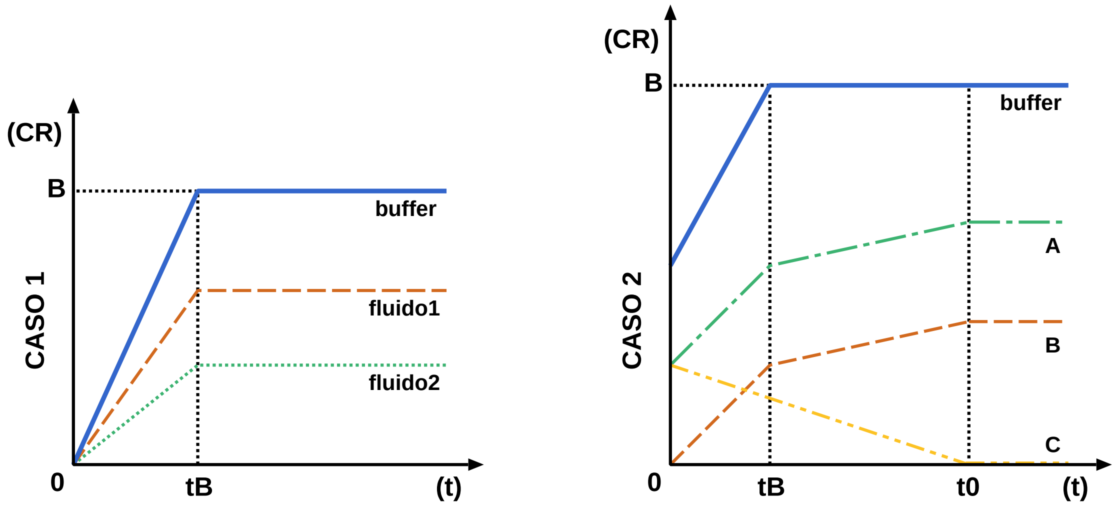


Figura 4.10: Exemplos do Controle dos CRs Baseados na Soma

de crescimento originais. Nota-se que existe perda de fluido para os fluidos A e B.

Para que o mecanismo fosse robusto o suficiente de modo a cobrir todos os casos, teve de ser criado um algoritmo que analisa uma a uma as mudanças de taxas de cada fluido individual ou da soma geral. Basicamente, a soma deve ser atualizada a cada mudança de taxa individual, uma vez que as equações dos fluidos são caracterizadas pelas seguintes equações de primeiro grau:

$$LI_i \leq CR_i \leq LS_i \tag{4.2}$$

$$CR_i(t) = IR_i(t).t + CR_i(t_{inicial}) \tag{4.3}$$

$$CR_{soma}(t) = \Sigma_{j=1}^{i} CR_j \tag{4.4}$$

$$LI_{soma} \leq CR_{soma} \leq LS_{soma} \tag{4.5}$$

onde, LI_i e LS_i representam o limite inferior e superior do valor acumulado da recompensa i, e CR_i o próprio valor acumulado. A equação 4.2 indica que o valor acumulado CR de uma recompensa i deve estar sempre dentro dos limites individuais desta. Os símbolos $CR_i(t_{inicial})$ e $IR_i(t)$ representam o valor acumulado no tempo inicial do intervalo e a recompensa instantânea atual da recompensa i no tempo t. Esta equação 4.3 é válida para um intervalo de tempo onde não haja mudança de taxa IR_i, neste exemplo $(t_{inicial}, t]$, e descreve o comportamento do valor acumulado da recompensa i. Em 4.4, CR_{soma} é o valor acumulado resultante da soma das demais recompensas e a equação 4.5 indica que o valor acumulado da soma deve estar dentro do intervalo (LI_{soma}, LS_{soma}) delimitado por seus limites inferior e superior, respectivamente.

A Figura 4.11 mostra as iterações do algoritmo de atualização das recompensas acumuladas, que atua a cada variação de taxa. No exemplo, as iterações ocorrem nos tempos: ut, $ut+1$, ..., $ut+n$, ..., até *tempo*. A Figura 4.12 apresenta a listagem

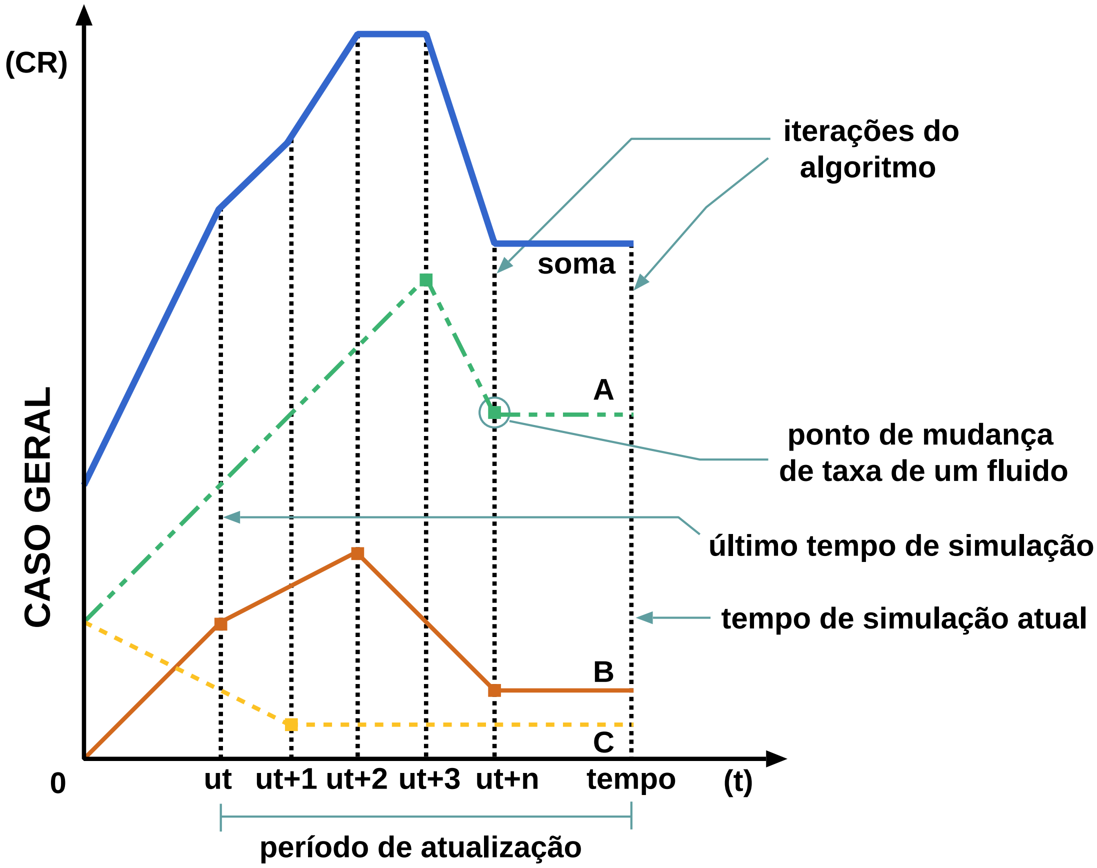


Figura 4.11: Iterações do Algoritmo de Atualização dos Valores Acumulados

do algoritmo em português estruturado, para n recompensas, salientando-se alguns pontos *a priori*:

- Supondo que a simulação progrediu no tempo, a missão do algoritmo é atualizar os valores acumulados CR_i de cada recompensa, bem como o valor acumulado da soma CR_{soma}, entre o último tempo de simulação (ut) e o tempo atual de simulação (*tempo*).

- Quando o valor de uma recompensa CR_i encontra-se dentro de seus limites, sua taxa de atualização é idêntica ao seu valor instantâneo IR_i. No entanto, caso haja a necessidade de um desvio de crescimento para que o CR_{soma} não extrapole os limites, o valor instantâneo de cada recompensa IR_i será modificado, podendo variar entre 0 e seu IR_i nominal. A este IR_i modificado, dá-se o nome $IRvirtual_i$. Este $IRvirtual_i$ é o valor instantâneo que efetivamente está sendo utilizado para a atualização da recompensa acumulada CR_i, levando em consideração todos as demais recompensas envolvidas na soma, seus limites e o valor acumulado da própria soma e seus limites, de forma que nenhum dos valores acumulados extrapole seus limites.

- Para recompensas que não estão associadas a nenhuma soma, a atualização do CR_i pode ter o comportamento de uma reta (sem mudanças de inclinação), caso em que o $CR_i(tempo)$ esteja dentro de seus limites, ou de duas retas, com apenas um ponto de quebra, justamente no ponto onde o limite foi alcançado. Para as recompensas associadas e suas somas, podem haver n quebras, como exemplificado na Figura 4.11.

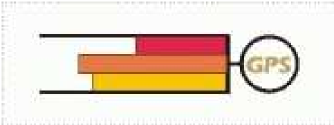
Cabe ressaltar que existe uma limitação na implementação, que proíbe a inclusão de uma mesma recompensa em mais de uma soma, portanto, uma recompensa de taxa, pode estar sob o controle de uma única soma, onde a inclinação de seu CR pode mudar por causa de mudanças em qualquer IR de recompensas pertencentes a soma, ou qualquer limite individual seja alcançado, ou ainda, no caso de o limite da própria soma ser atingido. Obviamente, também existe a possibilidade da recompensa ser livre de qualquer controle, de forma que somente suas mudanças de IR ou limites individuais possam alterar a inclinação de seu CR.

A partir do momento que a soma foi implementada, criou-se o comando *get_ cr_ sum(rate_ reward_ sum_ name)* de forma que pudesse ser usado, dentro da ação dos atributos *Messages* e *Events*, para ler o valor acumulado *CR* da soma de recompensas. Adaptou-se também, o evento especial *REWARD_ REACHED* para que este pudesse disparar quando o valor acumulado da soma de recompensas atinge algum limite, de forma análoga a utilizada para as recompensas de taxa. A Figura 4.13 mostra o evento t_B, no objeto *Server_ Queue_ GPS*, cuja condição prepara o disparo

```
A L G O R I T M O

1) atualize o valor acumulado entre ut e tempo de todas as recompensas que
   não estão atreladas a uma soma, levando em consideração seu IR e seus
   limites.
2) para cada soma de recompensas faça:
     3) veja se a soma alcançou algum limite.
        - caso negativo, faça para cada recompensa IRvirtual = IR
          (IR virtual recebe IR nominal).
        - caso positivo, calcule para cada recompensa o IRvirtual baseado
          em que a soma dos IRvirtuais deve ser nula. cada valor IR que se
          opor ao sentido da soma deve ser distribuído para os IRvirtuais
          no sentido da soma, em proporção a seu tamanho IR nominal.
     4) analize cada recompensa pertencente a soma entre ut e tempo levando
        em consideração seus valores IR e limites. pegue a soma dos CR e
        IRvirtuais, assim como o menor ponto de quebra de taxa dentre elas
        (denominado tquebra4).
     5) baseado na soma dos CR e IRvirtuais, calcule o comportamento da
        soma entre ut e tempo e veja se esta extrapola algum limite, e em
        qual tempo isto aconteceria (tquebra5).
     6) pegue o menor dos tempos: (tquebra4, tquebra5, tempo).
        - caso o menor valor seja tempo, atualize todas as recompensas da
          soma, inclusive ela, entre ut e tempo e vá para o passo 7.
        - caso o menor tempo seja tquebra4, atualize todas as recompensas
          da soma, inclusive a soma, entre ut e tquebra4. faça ut=tquebra4
          e vá para o passo 2.
        - caso o menor tempo seja tquebra5, atualize todas as recompensas
          da soma, inclusive a soma, entre ut e tquebra5. faça ut=tquebra5
          e vá para o passo 2.
7) fim de atualização.
```

Figura 4.12: Algoritmo de Atualização de Recompensas para o momento em que o *buffer* compartilhado encher por completo.

```
name=Server_Queue_GPS
```




```
Events=
 event= t0(REWARD_REACHED)
 condition=( (get_cr(fluido1)\/0) ||
             (get_cr(fluido2)\/0) ||
             (get_cr(fluido3)\/0) )
 action= {
           ...
         };

event= tB(REWARD_REACHED)
 condition=( get_cr_sum(buffer) /\ B )
 action= {
           ...
         };
```

```
Rewards=

  rate_reward_sum = buffer
    bounds  = 0,B
    rewards = fluido1 + fluido2 + fluido3

  rate_reward = fluido1
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido2
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido3
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
```

Figura 4.13: Soma de Recompensas na Condição de Evento Especial

A restrição do evento especial, onde uma mesma recompensa não pode ser utilizada mais de uma vez na mesma condição, é também estendida à soma de recompensas. Portanto, uma recompensa não pode estar na mesma condição onde sua soma é analisada.

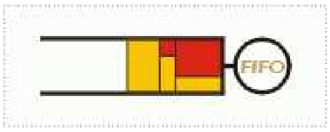
4.2.8 Variável de Estado de Tamanho Variável

Os recursos até então implementados permitiam a modelagem da maior parte dos objetos descritos no capítulo 3, faltando a implementação da fila FIFO e do canal de comunicação.

Estes elementos possuem características peculiares onde certas variáveis possuem tamanho variável. Como exemplo cita-se que a fila FIFO precisa armazenar blocos de fluido, de acordo com mudanças de taxas de entrada (cada mudança de taxa requer o armazenamento de um novo bloco). Estas taxas, em geral, variam de forma aleatória, tornando impossível prever em tempo de modelagem, quantos blocos precisam ser armazenados.

Para solucionar este problema, foram criadas variáveis de estado com capacidade de alocação dinâmica durante a execução da simulação. As variáveis de estado, denominadas *IntegerQueue* e *FloatQueue* são capazes de enfileirar quantos elementos forem necessários, dinamicamente. A figura 4.14 mostra trechos do código de um

```
name=Server_Queue_FIFO
```




```
Declaration=

  State Var
    FloatQueue: bloco[4]; // A
    ...

Initialization=

  bloco = [0,0,0,0]   // B
  ...
```

```
Messages=

 msg_rec=port_in
 action= {

   float d[3];          // taxas de 3 fluidos
   float prox_bloco[4]; // taxas 3 fluidos + temporizador
   float temporizador;
   int   i;
   ...

   // calculo das taxas de saida d[] para
   // a nova proporção de entrada

   // calculo do tempo de consumo para o bloco a
   // frente do que está se formando (temporizador).

   // preenchimento de prox_bloco
   prox_bloco[0] = temporizador;
   for (i=1; i<4; i++)
   {
      prox_bloco[i] = d[i-1];
   }

   // armazenamento de prox_bloco na estrutura dinamica
   save_at_tail( "bloco[]", prox_bloco ); // D

   ...
 };
```

Figura 4.14: Variável de Estado Dinâmica na Fila FIFO

objeto Fila_Servidor_FIFO, onde é necessário o uso de uma variável de estado de tamanho variável para o armazenamento dos blocos. No exemplo, que possui 3 classes de fluidos, é necessário que cada bloco guarde 4 informações: as taxas de saída de cada fluido para aquele bloco e o tempo necessário para que elas entrem em vigor. O seu uso pode ser observado na linha A, onde a variável *bloco* do tipo *FloatQueue* é declarada com dimensão 4 e em B onde ela é inicializada.

Do ponto de vista de leitura e escrita, estas variáveis dinâmicas se comportam de forma idêntica às variáveis de estado estáticas. Ou seja, as variáveis podem ser lidas com a função get_st() e atualizadas com set_st(). Nestas operações, o vetor acessado é denominado vetor de trabalho *trab*. A figura 4.15 mostra a estrutura de dados da variável de estado de tamanho variável e permite a observação do vetor de trabalho.

O dinamismo se dá, pela capacidade que elas possuem de enfileirar e desenfilei-

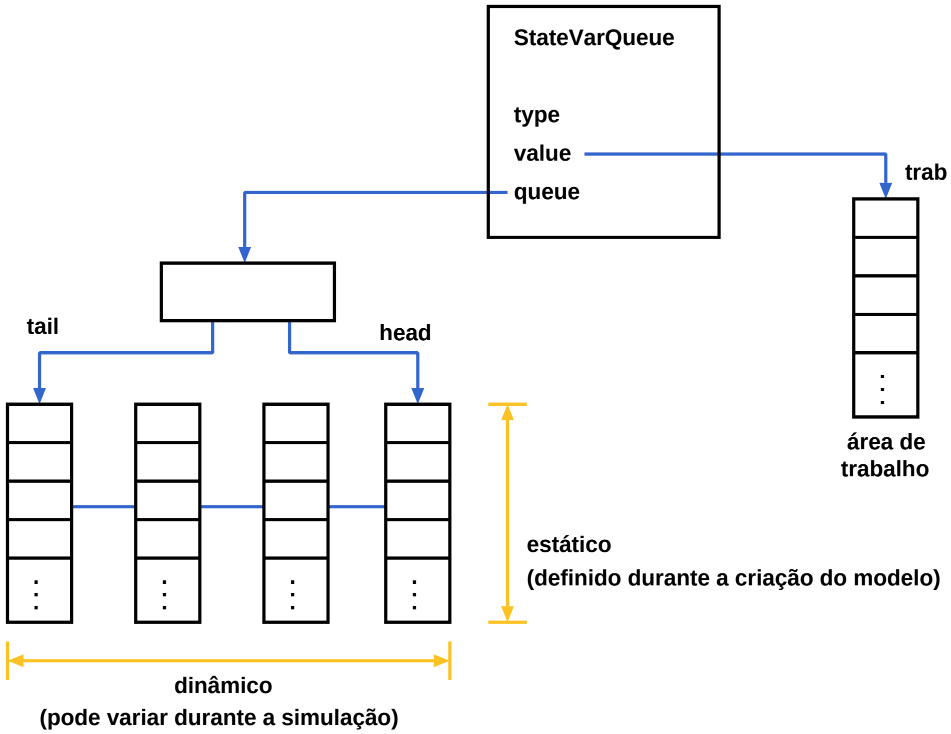


Figura 4.15: Estrutura de Dados da Variável de Estado de Tamanho Variável

rar vetores de igual tamanho e tipo de sua declaração (o próprio vetor interno *trab* ou algum vetor declarado dentro do código da ação desde que satisfaça a condição). Através dos comandos: *save_at_head()*, *save_at_tail()*, *restore_from_head()* e *restore_from_tail()* o vetor *trab* é posto ou retirado da fila, em sua cabeça ou cauda.

Também é permitido a inserção de um vetor local (declarado no código da ação) diretamente na cauda ou cabeça da fila, desde que seja do mesmo tamanho e tipo do vetor *trab*. A linha D, situada ao final do atributo *Messages*, apresentado na figura 4.14 mostra justamente o exemplo onde o conteúdo do vetor local *prox_bloco* é enfileirado na cauda da variável de estado dinâmica *bloco*. No exemplo, a cada mudança de taxa de entrada é armazenado um novo bloco que guarda em seu conteúdo as taxas de saída proporcionais àquele período e uma informação de tempo que indica quando estas entrarão em vigor.

De forma análoga, é permitida a retirada de um vetor alocado na fila, de maneira que seus dados sejam copiados diretamente para um vetor local, desde que este tenha o mesmo tamanho e tipo do vetor *trab*. Após a cópia o vetor é removido da fila, fazendo com que esta diminua em tamanho. A lista completa que mostra a sintaxe dos comandos pode ser encontrada no manual da ferramenta [27].

4.3 Erros de Cálculo Numérico e Problemas Enfrentados

Tendo em vista que este trabalho tem como base o uso de modelos de estado contínuo, é necessário que alguns comentários sejam feitos a respeito. O primeiro ponto a ser esclarecido refere-se a forma com que a informação é tratada dentro do computador, que mesmo sendo intrinsecamente discreta, pode ser considerada contínua devido a sua granularidade. Os números são representados por variáveis com milhares de níveis de discretização (tipo *double* da linguagem C) e podem ser considerados na prática, contínuos.

Esta discretização intrínseca aos computadores dá origem a erros de cálculo numérico computacional. Estes erros foram responsáveis por grande parte dos problemas enfrentados durante este trabalho de tese.

4.3.1 Comparação em Intervalos e Sistema de Aproximações

Os computadores digitais só conseguem armazenar números de forma discreta, portanto, para que os números reais (contínuos) sejam representados deve haver uma aproximação para o número discreto mais próximo do real em questão. Problemas mais graves ocorrem quando operações entre estes números são realizadas. Em muitos casos, as operações fazem com que o número perca precisão, justamente por causa das aproximações que devem ser feitas, já que o resultado da operação é armazenado em uma variável com número limitado de *bits*. Estes problemas são apresentados em [29].

Esta característica faz com que números reais armazenados no computador não possam ser comparados diretamente. A solução para este problema é usar intervalos de comparação. Um número x é considerado igual à outro y se diferir no máximo ϵ do valor de y.

$$x = y \quad se \quad (x - \epsilon) < y < (x + \epsilon). \tag{4.6}$$

onde ϵ é um valor bastante pequeno, da ordem de 10^{-6} ou menor. Este valor ínfimo deve ser suficientemente grande para evitar os erros de precisão númerica e pequeno o bastante para não alterar de forma significativa o resultado das simulações.

No TANGRAM-II, usou-se 2 números, $x\_epsilon$ e $y\_epsilon$, para comparações nos eixos das abscissas e ordenadas, respectivamente. Valores de tempo são associados ao eixo x e valores de recompensas ao eixo y. Nos testes realizados durante a implementação usou-se 10^{-10}, que se mostrou um valor seguro, para ambas as variáveis.

Além disto, estes números também são a base do sistema de aproximações implementado. O sistema de aproximações atrai valores muito próximos para determinados valores (pontos estratégicos), como por exemplo os limites de um valor acumulado, atuando de forma similar à força da gravidade. A Figura 4.16 exemplifica o sistema de aproximações. Em A um ponto é atraído para o limite zero no eixo y e para o tempo t no eixo x, fazendo com que sejam minimizados os erros de precisão numérica. Em B, há uma amostra, representada pela linha tracejada, do que poderia acontecer com um valor acumulado (CR) se não houvesse aproximação e do que ocorre (linha contínua) com o sistema de aproximações atuando.

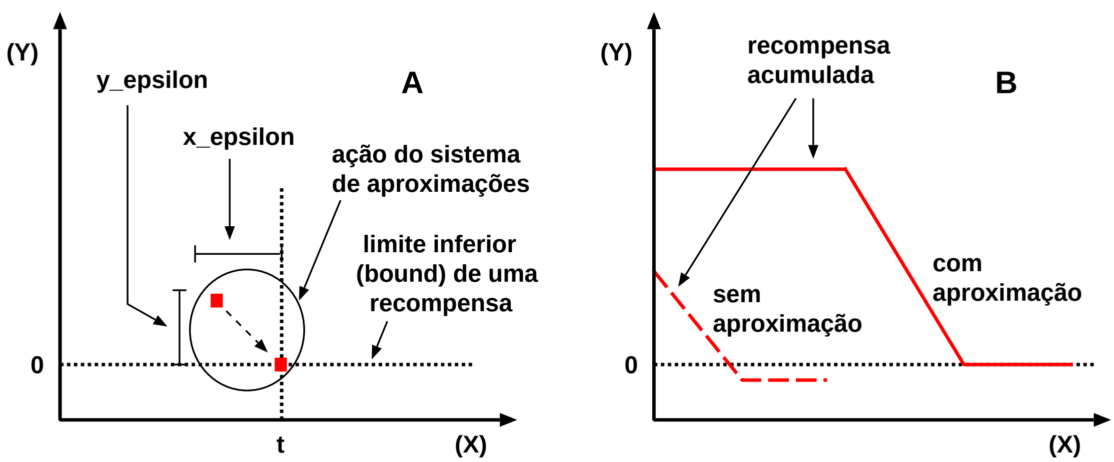


Figura 4.16: Sistema de Aproximações Utilizado no Simulador de Fluido

4.3.2 Detector de Falta de Progressão e Mecanismo de Atualização de Recompensas com Valores Pré-Definidos

Antes da implementação do sistema de aproximações e da comparação em intervalos, enfrentava-se o seguinte problema: O cálculo para o tempo de disparo de um evento $REWARD\_REACHED$ é realizado de acordo com a equação 4.1, e está exemplicado na figura 4.4. Suponha que o valor Δt calculado (para que CR alcance 0) seja um número com muitos dígitos significativos após a vírgula, como por exemplo: $2,5069076059$. Imaginando-se que no disparo do evento, o tempo de simulação fosse também um número com muitos dígitos significativos após a vírgula, como $12,5069076059$, a atualização das recompensas não encontraria o valor exato $CR = 0$, por causa dos erros de precisão numérica. Em determinadas situações, o valor CR permaneceria acima do limite $L = 0$ e um novo cálculo do Δt então seria acionado. Neste, também por causa da imprecisão, o Δt receberia o valor zero, e a simulação entraria em um laço infinito onde o tempo não mais progrediria.

Da análise e correção deste problema surgiram dois mecanismos: Um detector de falta de progressão de simulação, e o mecanismo de atualização de recompensas com valores pré-definidos. O detector monitora o tempo de simulação e inicia uma contagem do número de disparos de eventos especiais ($REWARD\_REACHED$), assim como dos demais eventos, toda vez que o tempo de simulação não progride. É perfeitamente possível, que ocorram diversos eventos em um mesmo tempo de simulação, no entanto pouco provável que isto ocorra por um número elevado de vezes (por exemplo 10^4). Se uma das duas contagens, de eventos especiais ou normais, ultrapassar este valor, o simulador imprime uma mensagem de aviso para o usuário indicando que algo pode estar errado, e que provavelmente o simulador está em laço infinito. A mensagem se repete em cada múltiplo do número, e cabe ao usuário cancelar a simulação. Além do sistema detector de laço infinito, foi criado um mecanismo de atualização de recompensas com valores pré-definidos. Quando um valor Δt é calculado, o sistema armazena o limite alcançado e qual recompensa será responsável pelo disparo do evento especial (por exemplo: a recompensa fluido1 alcançando o limite 0; neste caso o mecanismo armazenaria a recompensa fluido1 e o valor 0). No disparo de fato do evento, o mecanismo é acionado e utiliza os valores pré-definidos (previamente armazenados) para atualizar o valor exato da recompensa. Desta forma, o processo de cálculo do CR que pode inserir erros de precisão é dispensando, com isto tem-se uma melhora na precisão do resultado.

Tanto o mecanismo de atualização de recompensas com valores pré-definidos como o detector de falta de progressão, coexistem com a comparação em intervalos e o sistema de aproximações apresentados na subseção anterior. Inclusive, a atração em relação ao eixo x só ocorre por causa da existência do mecanismo de atualização de recompensas com valores pré-definidos. A Figura 4.17 apresenta um exemplo onde um evento especial possui 2 recompensas em sua condição - "*(get_cr(z) /\ B) || (get_cr(w) \/ 0)* " - e ambas alcançam seus limites em tempos muito próximos t e t'. O sistema de aproximações unifica ambos os tempos em t ou t' (o que for encontrado primeiro durante a avaliação da condição do evento) e armazena os valores dos limites, um para cada recompensa. Num deles será armazenado, z e B enquanto no outro w e 0. Quando o evento disparar, no momento da atualização

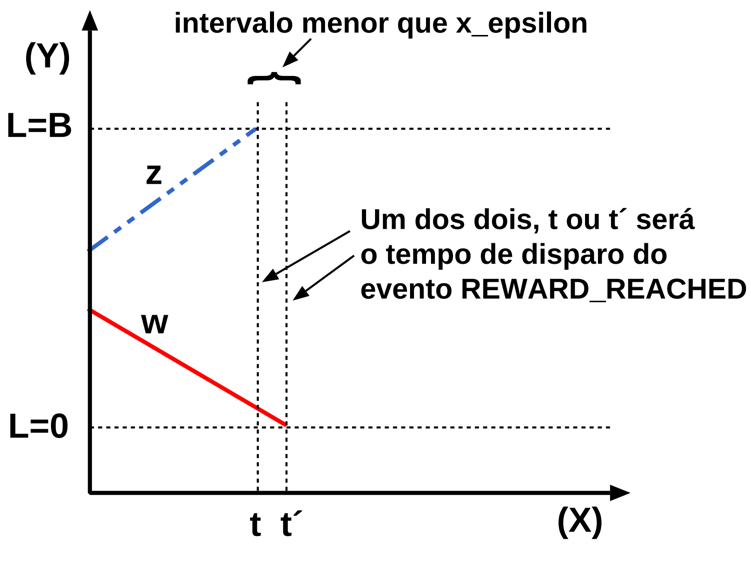


Figura 4.17: Atualização de Recompensas e Aproximação no Eixo das Abcissas

das recompensas, estas duas serão poupadas do cálculo tradicional e seus valores acumulados serão atualizados (carimbados) de acordo com os valores previamente armazenados.

4.3.3 Garantia de Ocorrência de Eventos Especiais

Mesmo depois da criação dos mecanismos acima descritos, ainda foram encontrados problemas. A Figura 4.18 ilustra uma situação onde um evento especial não dispara. Supondo-se uma recompensa w predestinada a alcançar seu limite B no

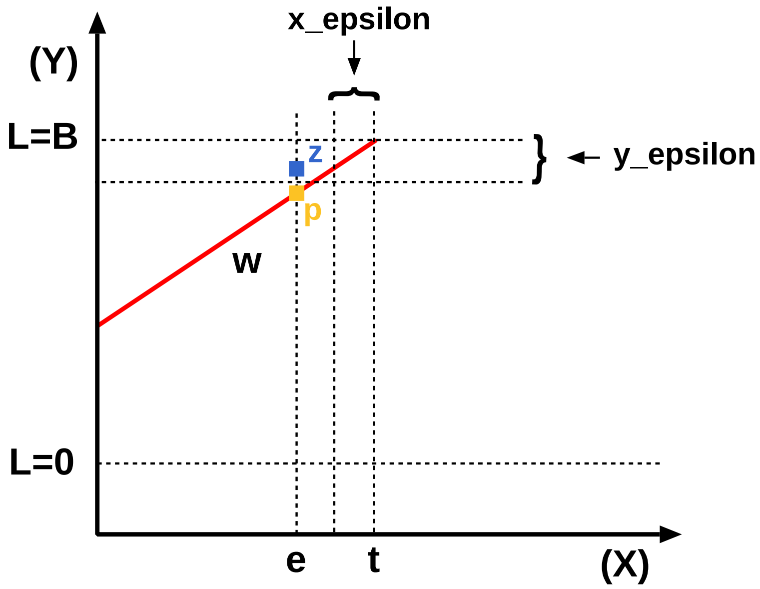


Figura 4.18: Problema de Falha de Evento Especial

tempo t. No entanto, assume-se que um evento dispara num tempo e menor que $t - x\_epsilon$. Durante a atualização das recompensas neste tempo e, é perfeitamente possível que em vez de calcular o ponto p, para a recompensa acumulada de w, o sistema calcule um valor próximo, como por exemplo z, dado que existem erros de precisão numérica. Nestas condições, o sistema de aproximações age, atraindo o valor acumulado de w para seu limite B. Deste modo, o disparo do evento w deixaria de acontecer, já que este alcançou seu limite e na avaliação da sua condição, Δt receberia ∞.

De modo a evitar este tipo de problema, foi criado um mecanismo que garante a ocorrência de um evento especial se porventura, a recompensa que prederminou seu tempo de toque alcance o valor limite durante a atualização das recompensas ocasionada por um disparo de outro evento. Neste mecanismo, são monitoradas todas as recompensas utilizadas em condições de eventos especiais. Durante o processo de atualização das recompensas, é feita uma verificação que indica se alguma recompensa alcançou um limite especificado em alguma condição. Se algum destes limites foi alcançado, e sabendo-se que o evento está escalonado num tempo muito próximo à frente, a única ação que precisa ser tomada, é o cancelamento do processo de avaliação da condição deste evento, evitando que este possa ser desescalonado. Assim existe a garantia de que o evento ocorrerá em seu devido tempo.

Cabe salientar, que os eventos especiais só tem suas condições avaliadas, quando alguma característica em sua condição é alterada, como por exemplo, uma mudança de estado ou uma mudança de IR de alguma recompensa que faz parte de sua condição. Além disto, esta avaliação só ocorre caso o mecanismo não tenha identificado que o evento deva ocorrer em virtude de que alguma condição tenha sido satisfeita durante o processo de atualização de recompensas.

4.3.4 Eventos Falsos e Verdadeiros

As condições dos eventos especiais *REWARD_REACHED* são analisadas no tempo presente de simulação t e antevêem o futuro de forma a identificar quando os valores acumulados alcançam algum dos limites. No entanto, quando o recurso de monitoração de soma de recompensas foi implementado, surgiram problemas de inconsistência com este mecanismo de previsão. A figura 4.19 exemplifica o caso onde um evento é escalonado para o tempo t' por possuir em sua condição a seguinte sentença: "*get_cr(fluido1)* $\bigwedge$ *LS1* ". Entretanto, neste exemplo o $fluido1$ pertence a soma de recompensas *buffer* e num dado momento, tem seu comportamento alterado por causa da soma, de forma a não mais tocar seu limite $LS1$ no tempo t'. Com isto o evento previamente escalonado para t' estaria incorreto.

Para resolver este problema o algoritmo descrito em 4.2.7 teve de ser incluído no processo de cálculo do tempo de disparo Δt. Os eventos que fazem parte de uma soma de recompensas passaram a não mais analisar o comportamento de um CR isoladamente, mas de todo o conjunto de recompensas associadas a esta soma

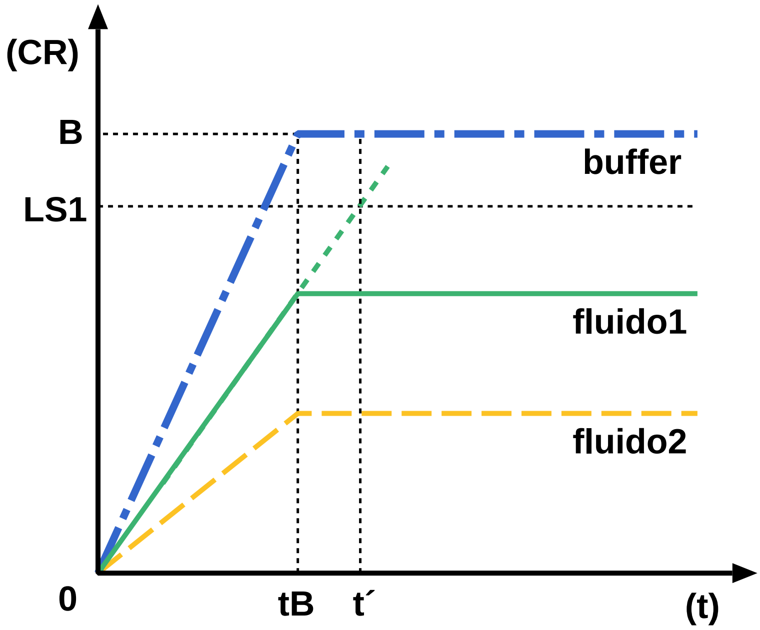


Figura 4.19: Problema de Escalonamento Inválido

($rate\_reward\_sum$). Além disto, pensou-se numa lógica onde os eventos escalonados erroneamente pudessem ser ignorados, evitando inconsistências na simulação. O mecanismo implementado escalona o disparo do evento, obtendo o menor dos tempos calculados de acordo com as seguintes regras:

1- Tempo em que o CR do fluido analisado irá tocar seu limite. Este cálculo é feito de forma isolada, como se só existisse este fluido.

2- O menor tempo de mudanças de taxas de acúmulo (IR) dentre todas as recompensas envolvidas na soma. Um exemplo destas mudanças de IR é encontrado na figura 4.11.

Se o menor tempo Δt não for o calculado pela regra 1, tem-se uma inconsistência. Portanto o evento deve ser ignorado no momento em que seu tempo de disparo for atingido. Para facilitar este trabalho, é utilizada uma variável indicadora de inconsistência. No tempo de disparo, ao identificar a inconsistência do evento através da variável indicadora, o simulador simplesmente ignora toda e qualquer ação que deveria tomar em condições normais, apenas sendo responsável pelo reescalonamento deste evento especial. Deste modo, alguns eventos escalonados no simulador do TANGRAM-II podem ser falsos (Não tem nenhum efeito). A figura 4.20 mostra a evolução do simulador de fluido no tempo, onde coexistem eventos verdadeiros e falsos .

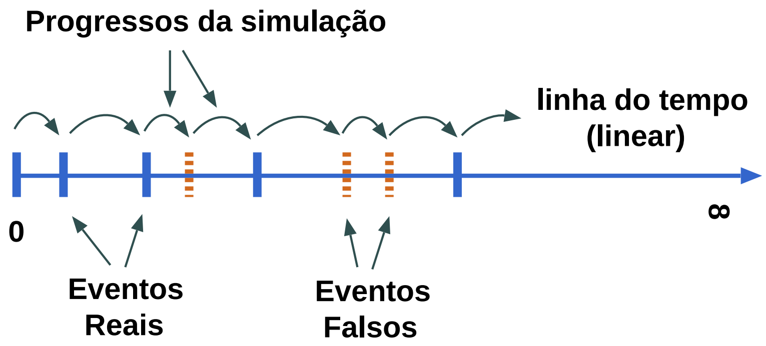


Figura 4.20: Linha do Tempo para o Simulador de Fluido

Esta opção foi escolhida em prol de ganho de velocidade. A solução mais natural, seria fazer toda a análise do comportamento futuro de um valor acumulador CR e escalonar o evento, em cuja condição está definido o limite de alcance da recompensa em questão, para o tempo correto de alcance, levando em consideração todas as recompensas envolvidas, de modo que não existissem eventos falsos. Nesta solução, todas as mudanças de taxas seriam analisadas entre o tempo atual de simulação e o provável tempo onde a recompensa alcançaria algum limite. No entanto, diversas vezes, eventos deste tipo são reescalonados por causa de mudanças de taxas de entrada, fazendo assim com que o tempo de processamento gasto para tal cálculo fosse totalmente perdido.

Na solução apresentada, apenas o tempo mínimo dentre todas as mudanças de IR é obtido e sendo o evento escalonado para este, a simulação progride com garantia de que nada está inconstente. Se durante o processo alguma taxa de entrada de fluido mudar, o evento deve ser reescalonado, e a perda de processamento é mínima. Se nada alterar o comportamento das taxas IR das recompensas associadas ao evento e este ocorrer no tempo indicado, o simulador irá saltá-lo, apenas com o trabalho de fazer um novo escalonamento para este evento. Podem haver casos onde um seqüencia de eventos falsos pode ocorrer, já que num evento falso, a nova avaliação pode incorrer em um novo evento falso, e assim por diante.

4.3.5 Considerações Gerais

Diversos problemas, em diferentes níveis, foram enfrentados durante a implementação deste trabalho, incluindo os erros supracitados de precisão númerica e complexidade dos recursos criados. No entanto obteve-se uma implementação que proporciona a coexistência de simuladores de técnicas distintas em um único módulo, de forma transparente para o usuário.

Sabe-se que esta característica pode prover vantagens sequer exploradas neste trabalho, como a criação de modelos híbridos, onde parte da informação é tratada como fluido e outra como elementos discretos (pacotes), de modo que se possa usar cada técnica onde esta for mais conveniente e/ou mais rápida. Este tipo de modelo pode ser construído com o uso de objetos conversores fluido-pacotes e pacotes-fluido, que são elementos bastante simples de serem implementados. Trabalho semelhante pode ser encontrado em [31].

Assim sendo, acredita-se que o TANGRAM-II, tenha se fortalecido ainda mais como uma ferramenta de apoio à engenharia de redes, permitindo a criação e avaliação de modelos de forma facilitada e com diversas possibilidades de resolução. Um pouco mais deste potencial é mostrado no capítulo seguinte, onde são apresentados alguns dos modelos criados.

Capítulo 5

Exemplos de Modelos de Fluido

DURANTE este trabalho de tese foram criados diversos modelos com objetivos de validar os recursos implementados, verificar os ganhos da técnica de fluido e mostrar o potencial da ferramenta. Neste capítulo são apresentados alguns destes exemplos, sub divididos por seus objetivos.

5.1 Validação dos Recursos Implementados

Uma das formas de validar os recursos e modelos implementados no simulador de fluido do TANGRAM-II, foi a comparação de resultados com outro simulador de fluido. Para tal tarefa, utilizou-se o *NetSimul*, brevemente descrito a seguir. Diversos modelos foram simulados em ambos os simuladores e o critério utilizado para verificação foi a comparação dos valores das medidas de interesse para os modelos análogos. Além disto, alguns modelos mais simples, foram comparados com suas resoluções analíticas, donde algumas foram calculadas através do próprio TANGRAM-II. Estas comparações provêm uma base ótima de comparação, justamente porque os valores das medidas de interesse dados pelo simulador são comparados com sua medida exata, calculada analiticamente.

5.1.1 Simulador de Fluido NetSimul

O NetSimul [1] foi desenvolvido no departamento de ciências da computação da universidade UMASS nos Estados Unidos, e está disponível na internet [9]. A ferramenta consiste em dois simuladores simples e independentes, um de pacotes e outro de fluido. Estes, criados para simular redes de computadores, possuem diversos componentes, tais como fontes, canais, roteadores e filas. Os modelos são especificados através da linguagem de programação C, onde se faz o uso dos objetos previamente construídos. Os simuladores foram construídos sobre o *framework* SSF (*Scalable Simulation Framework*), que reúne um conjunto de recursos para construção de simuladores, inclusive com possibilidade de programação paralela. Segundo os autores, o simulador foi amplamente testado e validado, através de resultados analíticos e de comparações com o simulador de redes mais utilizado pelo meio acadêmico, o NS (Network Simulator) desenvolvido pela universidade de Berkeley. Inclusive, deste trabalho de validação surgiu um relatório técnico [28] que indicara um erro de implementação na fonte *on-off* do NS.

3 Fontes On-Off e uma Fila Servidor

O primeiro exemplo apresentado possui 3 fontes do tipo *on-off* com parâmetros: $\lambda = 1$, $\mu = 1$ e $\gamma = 10$ (on-off / off-on / taxa de transmissão) e uma fila servidor GPS de parâmetros: $B = \infty$, $\phi_1 = \phi_2 = 0,5$ e $C = (15,1515; 15,7894; 16,6666)$ onde cada capacidade de serviço correponde a uma utilização específica $(0,99; 0,95; 0,90)$, respectivamente. A Figura 5.1 mostra o diagrama dos modelos. A única diferença do segundo exemplo em relação ao primeiro é a disciplina de atendimento da fila, que passa a ser FIFO. Salienta-se que sendo ambas as disciplinas conservadoras de trabalho e tendo os modelos os mesmos parâmetros, os valores médios das filas devem ser iguais, fato este que pode ser comprovado nos experimentos. As tabelas 5.1 e 5.2 mostram os resultados obtidos nas simulações com as duas ferramentas, o Tangram-II e o NetSimul. Elas mostram o valor médio da fila e o intervalo de confiança calculado em 95%, para as diferentes cargas. Para a obtenção dos resultados foram efetuadas 10 rodadas de simulação para o tempo 100.000 unidades. O testes foram

Figura 5.1: Modelos com 3 Fontes on-off e Fila Servidor

Utilização	**TANGRAM-II**	**NETSIMUL**
0,90	17,10 ± 0,3698	17,28 ± 0,3873
0,95	41,59 ± 1,5095	42,69 ± 1,2396
0,99	212,96 ± 20,6644	216,61 ± 18,4905

Tabela 5.1: Exemplo 1: Tangram-II X NetSimul - Tamanho médio da fila GPS

realizados em um Pentium IV 1.6GHz, 512MB Ram, barramento 400MHz, rodando o Red Hat Linux 7.1 kernel v2.4.18. A tabela 5.3 mostra o tempo médio de execução em segundos de cada simulação (10 rodadas de 100.000 unidades de tempo). Apesar de não provar que o funcionamento do simulador de fluido do TANGRAM-II é correto, os resultados encontrados validam a implementação, por apresentarem um comportamento muito semelhante. A medida de volume médio do *buffer* é um bom indicador, pois envolve muitas variáveis, onde salienta-se o comportamento da fonte, a manipulação das taxas de entrada da fila, o cálculo das taxas de saída para cada fluxo e a própria manipulação do recipiente que acumula os fluidos. Nos valores apresentados, observa-se a grande proximidade dos valores, e que estes encontram-se dentro dos intervalos de confiança, essenciais neste tipo de comparações.

De acordo com os resultados, nota-se que o simulador de fluido do TANGRAM-

Utilização	**TANGRAM-II**	**NETSIMUL**
0,90	17,50 ± 0,3594	17,24 ± 0,2375
0,95	42,48 ± 1,4251	42,56 ± 1,3823
0,99	214,52 ± 53,5291	219,53 ± 43,0326

Tabela 5.2: Exemplo 2: Tangram-II X NetSimul - Tamanho médio da fila FIFO

Modelo	**TANGRAM-II** (seg)	**NETSIMUL** (seg)
GPS	754	79
FIFO	2.146	55

Tabela 5.3: Tangram-II X NetSimul - Tempo de Execução

II, para este exemplo particular, é aproximadamente 22 vezes mais lento que o NETSIMUL (9,5 vezes para o GPS e 39 vezes para o FIFO). Isso é facilmente explicável pelo fato de o TANGRAM-II ter sido concebido com objetivo de prover flexibilidade e facilidade para o usuário, além de possuir um código fonte com alta legibilidade. Com estes objetivos, há um detrimento em relação à eficiência do simulador (*trade off*: eficiência *versus* legibilidade e flexibilidade). A diferença de desempenho entre os simuladores não está relacionada com a simulação de fluido e sim com os mecanismos de funcionamento internos ao TANGRAM-II.

Ainda assim cabe salientar que os modelos do NETSIMUL executados nesta comparação são extremamente rudimentares, sequer mostram os valores médios de cada fluxo de fluido dentro da fila, não calculam perdas, não geram *traces* e não foram construídos de forma a serem simulados em diversas rodadas. Logo os valores médios e o intervalo de confiança foram calculados manualmente, através do uso de técnicas descritas em [37].

As facilidades adicionais do TANGRAM-II em relação ao NETSIMUL também explicam em parte o maior tempo de execução. O TANGRAM-II tem todo um aparato de auxílio a modelagem, que começa desde a facilidade de modelagem propiciada por sua interface gráfica TGIF, especificação das medidas de interesse de forma objetiva e simples, possibilidade de geração automática de *traces*, além das facilidades que a interface (JAVA) da ferramenta proporciona. Além disto, os modelos executados acima são genéricos e possuem diversas recompensas, que a rigor não seriam necessárias para estes exemplos específicos. Como nestes modelos o tamanho do *buffer* é ilimitado, não haveria necessidade do uso da facilidade que fornece a soma de recompensas (*rate_reward_sum*), muito menos de recompensas para armazenamento das perdas. Algumas destas características, que indicam o poder e facilidade de uso da ferramenta são apresentadas no final deste capítulo em 5.3.

Filas em Série

De forma a executar um teste mais abrangente, foi criado um modelo mais complexo que possui filas em série (*tandem queue network*). O modelo é análogo ao apresentado em [1], e possui as seguintes características: Nove fontes on-off idênticas de parâmetros: $\lambda = 1$, $\mu = 0,5$ e $\gamma = 300$ (on-off / off-on / taxa de transmissão). Quatro filas-servidor GPS de parâmetros: $B = \infty$, $\phi_1 = \phi_2 = \phi_3 = \frac{1}{3}$ e $C = 315,7895$ (O valor de serviço tem como base uma utilização de 95% para a fila 1). Quatro canais de comunicação com $delay = 0,1$ e quatro comsumidores, responsáveis pela absorção do tráfego. A Figura 5.2 mostra a arquitetura do modelo. O tráfego gerado pela fonte 1 flui através de todas as filas e canais até ser consumido pelo consumidor 4. Além deste tráfego, denominado tráfego de interesse, cada fila recebe outras duas classes, cada uma alimentada por sua fonte. O fluxo destas 2 fontes extras, é imediatamente enviado para um consumidor, de forma que cada fila trabalhe com 3 fluxos (classes) de fluido.

Este modelo foi simulado no Tangram-II, enquanto sua contrapartida foi executada no NetSimul. Foram realizadas 10 rodadas de 100.000 unidades de tempo para a obtenção das medidas de interesse. O resultado em termos de valor médio das filas é apresentado na tabela 5.4.

Fila	**TANGRAM-II**	**NETSIMUL**
1	2.404,10 ± 142,3804	2.376,33 ± 156,3322
2	2.036,17 ± 108,8681	2.055,99 ± 126,1192
3	1.967,26 ± 103,8887	2.009,29 ± 111,4385
4	1.877,96 ± 132,4154	1.993,71 ± 91,4484

Tabela 5.4: Tangram-II X NetSimul - Média das Filas em Série

O simulador de fluido do Tangram-II manteve a mesma consistência apresentada em todas as demais comparações, estando os valores médios muito próximos aos encontrados pelo NetSimul e sempre havendo interseções nos intervalos de confiança.

5.1.2 Resolução Analítica

Além da comparação de resultados com outro simulador de fluido, alguns exemplos foram elaborados, de forma que seus resultados pudessem ser comparados com o resultado exato obtido através de solução analítica. Um destes exemplos, cuja solução analítica foi obtida através da própria ferramenta, é detalhado na próxima seção.

Histograma e 1 Fila

Um dos modelos criados, é composto de uma fonte do tipo histograma e uma fila, ambos modelados em um único objeto, como mostra a figura 5.3.

A fonte histograma fora obtida a partir de um *trace* de vídeo de uma partida de futebol, disponibilizado em [36]. Esta seqüência está codificada em MPEG-I e todos os três tipos de quadros (I,P,B) foram levados em consideração para a modelagem da fonte de tráfego. A parametrização fora feita conforme descrito em [40], onde a partir da análise do *trace* foram definidos valores para as taxas de transmissão em cada um dos oito níveis da fonte, apresentados na tabela 5.5. A fila-servidor

Nível	taxa de transmissão (bits/seg)
1	7024512
2	6098688
3	5177088
4	4251648
5	3330048
6	2404224
7	1482624
8	556800

Tabela 5.5: Taxas de Transmissão da Fonte de Histograma

possui $C = 1337368$ bits/s e $B = 100000$ bits. Mais detalhes sobre este exemplo podem ser encontrados em [39].

Os resultados analíticos foram obtidos através do próprio TANGRAM-II, com o uso do método de solução proposto em [10]. Este método calcula a função de distribuição complementar para uma medida de interesse especificada por uma recompensa, num dado tempo transiente. Esta distribuição é dada por vários pontos e seus respectivos valores de probabilidade, de onde pode-se extrair a média através da integral dos pontos, com o uso da técnica de aproximação por trapézios. O valor médio da fila, calculado analiticamente, fora $E[N] = 30707$.

Embora o mesmo modelo pudesse ser simulado por técnicas tradicionais, criou-se um modelo análogo com os objetos de fluido desenvolvidos neste trabalho, de forma que sua simulação pudesse ser comparada ao resultado exato previamente obtido. A figura 5.4 mostra o modelo, que fora construído com um objeto fonte histograma, de parâmetros idênticos ao da figura 5.3, e um objeto fila servidor GPS, que por ter apenas um fluxo de fluido em uso, torna irrelevante a disciplina de atendimento, comportando-se como o FIFO, idêntico ao modelado na figura 5.3. Este segundo modelo fora simulado em 10 rodadas de 100000 unidades de tempo, e o valor médio da fila encontrado fora $E[N] = 30562,85 \pm 418,48$ para um intervalo de confiança de 95%. Como esperado, o valor da medida de interesse encontrado na simulação fora semelhante ao seu valor exato, que encontra-se dentro do intervalo de confiança da simulação.

5.2 Amostra do Ganho Computacional

De forma a exemplificar o enorme ganho que pode ser obtido através do uso do simulador de fluido, é apresentado um modelo que descreve um comutador (*switch*) ATM. Além do ganho em termos de custo computacional, o exemplo apresentado mostra também que a diferença entre os resultados é bastante pequena, o que comprova as vantagens da técnica de fluido.

5.2.1 ATM

Foram criados na ferramenta TANGRAM-II dois modelos semelhantes, um de fluido e outro tradicional de células (pacotes de informação). A figura 5.5 mostra a topologia dos modelos. Os parâmentros escolhidos para a simulação de fluido foram: $\lambda = 1$(segundo), $\mu = 1$(segundo) e $\gamma = 348,4528$(1000 células/segundo) (on-off / off-on / taxa de transmissão) e uma fila servidor FIFO de parâmetros: $B = 18$(1000 células), $\phi_1 = \phi_2 = 0,5$ e $C = 366,79$(1000 células/segundo) onde a capacidade de serviço correponde a uma utilização de $0,95$. O modelo de células (pacotes), também têm mudanças de estado das fontes regidas por variáveis exponenciais, no entanto a geração de pacotes ocorre de forma determinística. Os parâmetros são idênticos aos do modelo de fluido, com excessão da capacidade de serviço C que necessita de um fator de correção de modo que a mesma utilização de $0,95$ seja mantida. Este fato se deve a uma diferença existente entre as fontes, onde o valor médio de cada uma varia de acordo com o paradigma utilizado.

A Figura 5.6 mostra a diferença entre uma fonte de fluido e uma fonte de pacotes com geração determinística. Na abstração de fluido, desde o exato momento em que ocorre uma transição para o estado *on*, até o momento onde a fonte é desligada, considera-se geração de tráfego, de forma exata. Na fonte de pacotes, ou células, em geral define-se que um evento de geração, representa o final de transmissão de um pacote, e com isto o primeiro evento de geração só ocorre depois de um tempo determinístico, justamente o tempo de geração de um pacote. O problema se dá ao final do período *on*, quando ao ser desligada, a fonte interrompe a geração de eventos de pacotes, fazendo com que o último pacote seja perdido. Deste modo as duas fontes têm um volume de tráfego gerado que pode diferir de até um pacote a cada período em que a fonte está ligada.

[28] demonstra esta diferença através de expressões analíticas. As equações 5.1 e 5.2 descrevem o cálculo da média de tráfego gerado pelas fontes de fluido e de pacotes respectivamente.

$$E[N]_{fluido} = \gamma . \frac{\lambda}{\lambda + \mu} \tag{5.1}$$

$$E[N]_{pacotes} = \frac{e^{-\frac{\frac{\lambda+\mu}{\lambda}}{\gamma}}}{1 - e^{-\frac{\frac{\lambda+\mu}{\lambda}}{\gamma}}} \tag{5.2}$$

onde $E[N]$ representa o número médio de pacotes (ou volume de tráfego) gerados pela fonte e λ, μ e γ as taxas de transição *on* para *off*, de *off* para *on* e de transmissão, respectivamente. No exemplo, a capacidade de serviço é calculada de acordo com:

$$C = \frac{2.E[N]}{U} \tag{5.3}$$

onde a capacidade de serviço do comutador C, é dada pelo dobro do valor médio de transmissão de cada fonte, dividido pela utilização U.

Assim sendo, a capacidade de serviço da fila de células, foi recalculada e a simulação foi executada com $C = 365,74$(1000 células/segundo). A tabela 5.6 mostra os resultados obtidos nas simulações obtidas através de 10 rodadas de 50.000 segundos cada.

Modelo	**Média da fila (1000 células)**	**Tempo de Execução (segundos)**
Células	6,9903 ± 0,03051	41.618
Fluido	6,8430 ± 0,01757	728

Tabela 5.6: Comutador ATM - Modelo de células X fluido

Neste exemplo os resultados diferem de $2,7\%$ e o simulador de fluido é 57 vezes mais eficiente, o que demonstra a ótima relação custo benefício do método. A unidade escolhida para a simulação foi 1000 células para que se pudesse simular o modelo de pacotes, em um tempo plausível. Se a unidade células fosse escolhida, o que seria mais adequado, em termos de corretude da simulação, o número de eventos para a simulação de pacotes seria acrescido em 3 ordens de grandeza. Enquanto que para o simulador de fluido, o tempo de execução seria o mesmo, fazendo com que a diferença de velocidade entre eles também fosse 3 ordens de magnitude maior.

Este é apenas um exemplo que mostra o enorme potencial da técnica. Por este pode-se imaginar que quanto maior forem os modelos, maiores serão os ganhos. O crescimento do número de eventos é linear para os modelos de pacotes, e será linear também para os modelos de fluido que não tiverem efeito *ripple*. Cabe salientar que o efeito *ripple* pode ser minimizado através do uso da técnica de agregação de tráfegos. Assim sendo, percebe-se que a técnica de fluido expande o limite das simulações atuais, em relação ao tamanho das redes modeladas, para algumas ordens de grandeza a mais, viabilizando a construção de modelos consideravemente maiores.

5.3 Usabilidade do Simulador

Nesta seção alguns exemplos são analisados de forma mais detalhada, para que se possa mostrar algumas das potencialidades que a ferramenta TANGRAM-II proporciona aos usuários, e que fazem dela uma excelente ferramenta de auxílio à engenharia e apoio didático.

5.3.1 ATM com perdas - Exemplo A

O primeiro exemplo apresentado nesta seção visa mostrar o poder e facilidade com que diversas medidas podem ser extraídas de um modelo. A Figura 5.7 apresenta um modelo onde são definidas recompensas para coletar as seguintes medidas de interesse: Tráfego sainte das fontes, tráfego entrante no consumidor, tráfego entrante e sainte na fila-servidor, média do buffer, utilização e perda. Outras medidas derivadas podem ser obtidas, como por exemplo o tempo médio de serviço, obtido através da lei de Little.

$$\bar{N} = \lambda.\bar{T} \tag{5.4}$$

onde, $\bar{N}$ é o número médio de elementos na fila, λ é a taxa média de chegada e $\bar{T}$ é o tempo médio de retardo.

Na Figura 5.8 podem ser observadas a especificação das recompensas no atributo *Rewards* e suas atualizações no atributo *Messages* do objeto *server_queue_GPS*.

O simulador apresenta três medidas distintas, $CR(t)$, $ACR(t)$ e $ATC(t)$ para cada recompensa especificada, como descrito na seção 2.1.4. Dependendo da maneira pela qual a recompensa é utilizada, alguma destas medidas fornece o valor de interesse. Por exemplo: a recompensa fluido1, recebe valores de recopensa instantânea (IR) que podem ser positivos ou negativos, assim sendo, o valor médio de fluido acumulado na fila é dado pela $ATC(t)$ (média de recompensa acumulada ponderada pelo tempo). A recompensa rew_a1 recebe somente valores positivos de IR, portanto as medidas $CR(t)$ e $ACR(t)$ informam o total de fluido entrante e a média de tráfego, respectivamente. Outras recompensas podem ser mais trabalhadas, como a utilização, que recebe o valor instantâneo 1 caso exista fluido sendo servido ou 0 caso contrário. Deste modo a medida $ACR(t)$ indica a proporção de tempo (entre 0 e 1) em que a fila foi utilizada. As somas de recompensas também fornecem estas medidas, e devem ser usadas de forma análoga.

Estes resultados são gravados no arquivo de saída da simulação. Um trecho do arquivo de saída é apresentado na Figura 5.9.

Após a execução de uma simulação de 30.000 segundos com 5 rodadas, os seguintes resultados foram obtidos para um intervalo de confiança de 95%: Além destes

Medida de Interesse	**Valor (intervalo)**	**Unidade**
Média de fluido1 na Fila	22,12 ± 2,38	Mb
Média de fluido2 na Fila	20,54 ± 2,47	Mb
Média da Fila	42,66 ± 0,61	Mb
Montante de Chegada de fluido1	2.007,91 ± 5,77	Tb
Montante de Chegada de fluido2	2.007,31 ± 8,63	Tb
Montante de Chegada	4.015,22 ± 5,24	Tb
Montante de Perda do fluido1	86,09 ± 16,08	Tb
Montante de Perda do fluido2	86,09 ± 16,08	Tb
Montante de Perda	172,18 ± 32,16	Tb
Utilização	82,65 ± 0,79	%
Medida Derivada	**Valor**	**Unidade**
Percentagem de Perda	4,29	%
Retardo Médio do fluido1	0,3999	μ seg
Retardo Médio do fluido2	0,3714	μ seg
Retardo Médio na fila	0,3856	μ seg

Tabela 5.7: Medidas de Interesse do server_queue_GPS (A)

resultados, existe ainda a opção de geração de *traces*. O simulador do TANGRAM-II gera automaticamente arquivos de *trace* para o valor instantâneo e acumulado de cada recompensa. Estes arquivos são gerados no padrão GNUPLOT, logo, além de fornecerem os valores ao longo de todo o caminho amostral para análises numéricas, podem ser usados diretamente na visualização de gráficos. As medidas podem ser combinadas de forma que se pode colocá-las em um mesmo gráfico, permitindo assim uma análise facilidada do comportamento das variáveis. As figuras a seguir apresentam trechos de gráficos obtidos apartir deste modelo.

Na Figura 5.10 pode-se observar o comportamento dos valores instantâneos e acumulado do fluido1. Na figura 5.11 observa-se as taxas de chegada $A(t)$ e saída $D(t)$ totais da fila-servidor. Já o gráfico da figura 5.12 mostra o nível de fluido acumulado na fila e as perdas acumuladas ao longo do tempo. Na figura 5.13 apresenta-se as taxas do fluido 1 na saída da fonte, na entrada da fila (nota-se que o comportamento é idêntico, mas defasado de 0,1 segundo, que é justamente o atraso provocado pelo canal) e na entrada do consumidor (idêntico à saída da fila). Estes são apenas alguns exemplos de gráficos, e salienta-se que qualquer combinação pode ser gerada. A interface provê uma tela onde o usuário escolhe os arquivos desejados e ao aperto de um botão, o GNUPLOT é chamado para desenhar os gráficos.

5.3.2 ATM com perdas - Exemplo B

Continuando a descrição do poder e facilidade da ferramenta, criou-se um novo exemplo, com as mesmas características do anterior exceto pelo fato de que reguladores de tráfego estão associados à entrada de cada fluxo de fluido na fila. O objetivo é mostrar apenas um exemplo do tipo de análises que podem ser feitas através dos modelos. A Figura 5.14 apresenta a arquitetura e os valores do novo modelo. Também foi realizada uma simulação de 30.000 segundos com 5 rodadas e intervalo de confiança de 95%, cujos resultados são apresentados nas tabelas 5.8, 5.9 e 5.10.

A partir destes resultados, pode-se fazer análises de comportamento de ambos os modelos, A e B. No exemplo A tem-se uma perda maior na fila, em relação ao B, já que neste existem reguladores de tráfego. A perda menor que ocorre em B tem o custo de maior espaço em *buffer*, já que os reguladores também possuem memória deste tipo. Com esta amostra, fica fácil perceber que diversos sistemas podem ser analisados de uma forma simples e muito rica em detalhes. Além de análises de engenharia, estes resultados podem auxiliar no aprendizado, já que os alunos podem acompanhar o desenvolvimento de sistemas de uma forma prática e totalmente visual.

Medida de Interesse	**Valor (intervalo)**	**Unidade**
Média de fluido1 na Fila	15,11 ± 0,36	Mb
Média de fluido2 na Fila	16,35 ± 0,53	Mb
Média da Fila	31,45 ± 0,80	Mb
Montante de Chegada de fluido1	2.189,32 ± 10,49	Tb
Montante de Chegada de fluido2	2.232,90 ± 6,35	Tb
Montante de Chegada	4.422,22 ± 16,80	Tb
Montante de Perda do fluido1	5,54 ± 0,59	Tb
Montante de Perda do fluido2	6,77 ± 0,51	Tb
Montante de Perda	12,31 ± 1,31	Tb
Utilização	97,40 ± 0,24	%
Medida Derivada	**Valor**	**Unidade**
Percentagem de Perda	0,2784	%
Retardo Médio do fluido1	0,2175	μ seg
Retardo Médio do fluido2	0,2255	μ seg
Retardo Médio na fila	0,2215	μ seg

Tabela 5.8: Medidas de Interesse do server_queue_GPS (B)

Medida de Interesse	**Valor (intervalo)**	**Unidade**
Média da Fila	46,77 ± 0,68	Mb
Montante de Chegada	2.179,51 ± 6,36	Tb
Montante de Perda	0,00 ± 0,00	

Tabela 5.9: Medidas de Interesse do leaky_bucket_1

Medida de Interesse	**Valor (intervalo)**	**Unidade**
Média da Fila	47,84 ± 0,91	Mb
Montante de Chegada	2.179,51 ± 7,36	Tb
Montante de Perda	0,00 ± 0,00	

Tabela 5.10: Medidas de Interesse do leaky_bucket_2

5.3.3 Outros Exemplos de Gráficos

Para finalizar, selecionou-se mais 4 gráficos que mostram o comportamento de um regulador de tráfego e de filas com disciplinas distintas. A Figura 5.15 mostra um gráfico gerado a partir de *traces* obtidos como resultado de uma simulação, onde aparecem as taxas de chegada *(ARRIVAL)* e saída *(DEPARTURE)* do regulador, o comportamento do balde de fichas *(BUCKET)* e da fila de dados *(BUFFER)*. Através das recompensas *LOWB_CR*, *UPPB_CR*, *D_CR* e *A_CR* pode-se observar a moldagem do tráfego, num gráfico gerado a partir do mesmo experimento que originou a figura anterior.

As figuras subsequentes mostram o comportamento do fluido1, fluido2 e do *buffer* para duas filas-servidor com tráfego de entrada idêntico (fontes idênticas e simuladas com o mesmo caminho amostral). No primeiro modelo, utilizou-se uma fila GPS e no segundo uma fila FIFO. Como resultado observa-se que o *buffer* é idêntico em ambas, já que são conservadores de trabalho, no entanto os volumes individuais de cada fluido varia de acordo com a disciplina de atendimento.

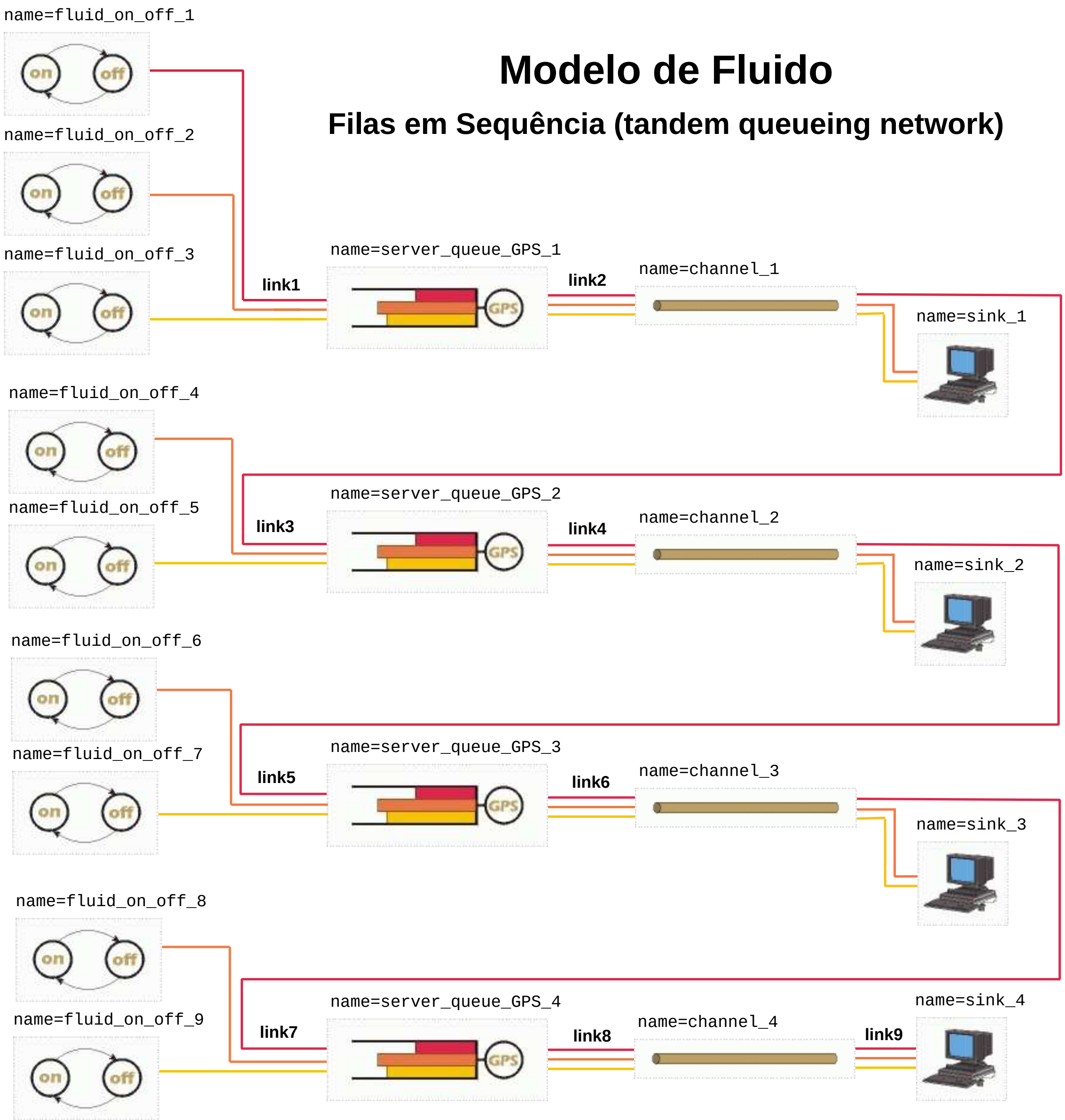


Figura 5.2: Filas em Série

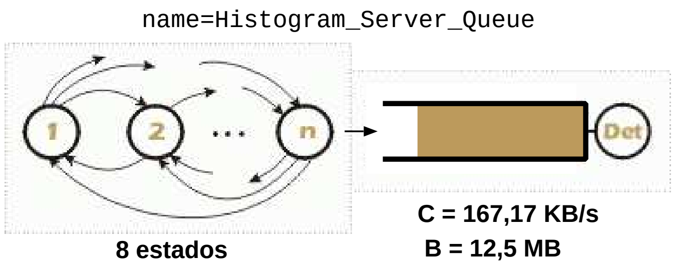


Figura 5.3: Modelo Fonte Histograma e Fila num Único Objeto

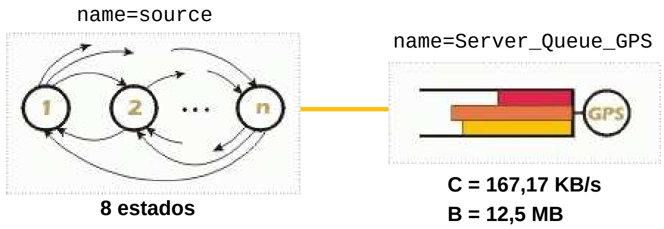


Figura 5.4: Histograma Montado com Objetos de Fluido

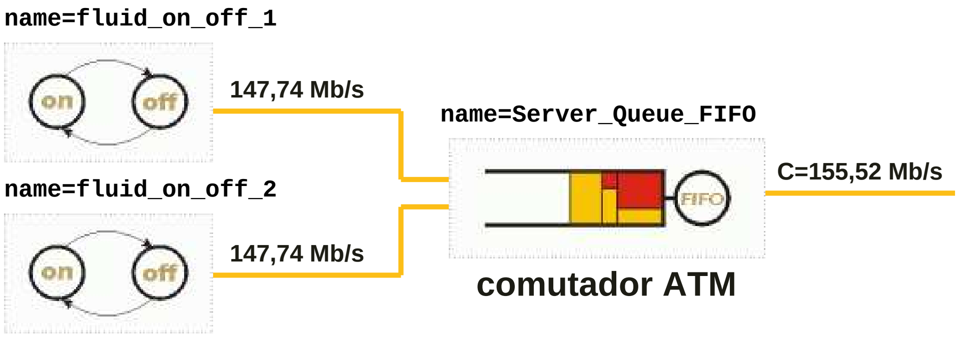


Figura 5.5: Modelo de um Comutador ATM

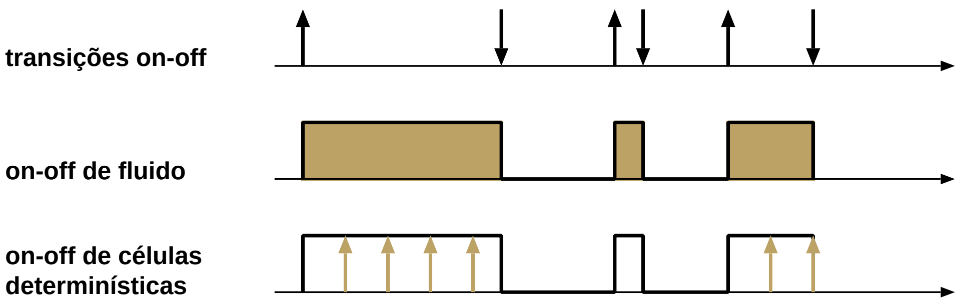


Figura 5.6: Diferenças Entre uma Fonte de Fluido e de Pacotes Determinística

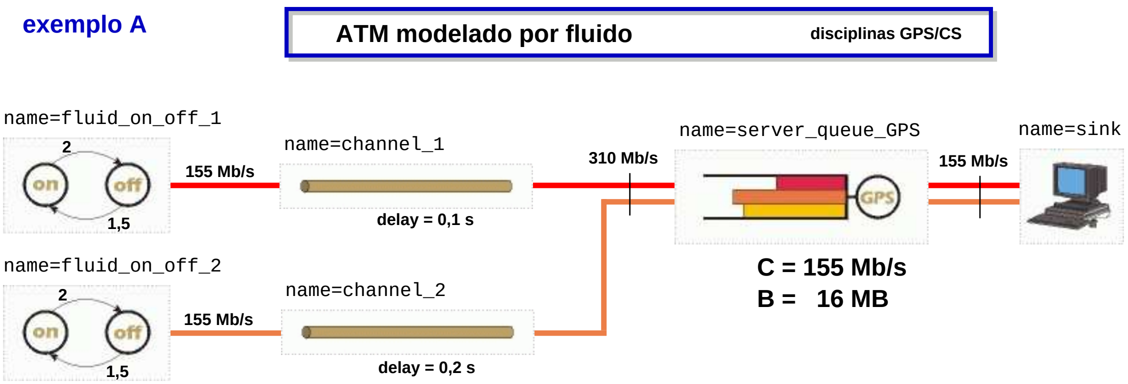
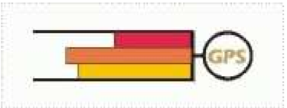


Figura 5.7: Exemplo A - Diversas Medidas de Interesse

```
name=server_queue_GPS
```




```
Messages=
 msg_rec=port_in
 action= {

   ...

   /*---- atualiza IR dos fluidos ----------*/
   valor = a[0] - d[0];
   set_ir( fluido1, valor );
   valor = a[1] - d[1];
   set_ir( fluido2, valor );
   valor = a[2] - d[2];
   set_ir( fluido3, valor );

   /*---- atualiza IR das perdas -----------*/
   valor = IR_perda[0];
   set_ir( perda1, valor );
   valor = IR_perda[1];
   set_ir( perda2, valor );
   valor = IR_perda[2];
   set_ir( perda3, valor );

   ...

   /*---- debug info ----*/
   valor = a_bck[0]; set_ir( rew_a1, valor );
   valor = a_bck[1]; set_ir( rew_a2, valor );
   valor = a_bck[2]; set_ir( rew_a3, valor );

   valor = a[0]; set_ir( rew_av1, valor );
   valor = a[1]; set_ir( rew_av2, valor );
   valor = a[2]; set_ir( rew_av3, valor );

   valor = d[0]; set_ir( rew_d1, valor );
   valor = d[1]; set_ir( rew_d2, valor );
   valor = d[2]; set_ir( rew_d3, valor );

   /*---- compute utilization --------------*/
   if ( valor > 0 ) {
     set_ir( utilization, 1 );
   }
   else {
     set_ir( utilization, 0 );
   }

   ...
 };
```

```
Rewards=
  rate_reward_sum = buffer
    bounds = 0,B
    rewards = fluido1+fluido2+fluido3;

  rate_reward = fluido1
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido2
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido3
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;

  rate_reward_sum = perda
    rewards = perda1+perda2+perda3;

  rate_reward = perda1
    condition = (FALSE)
    value = 0;
  rate_reward = perda2
    condition = (FALSE)
    value = 0;
  rate_reward = perda3
    condition = (FALSE)
    value = 0;

  /* debug info */

  rate_reward_sum = amount
    rewards = rew_a1+rew_a2+rew_a3;

  rate_reward = rew_a1
    condition = (FALSE)
    value = 0;
  rate_reward = rew_a2
    condition = (FALSE)
    value = 0;
  rate_reward = rew_a3
    condition = (FALSE)
    value = 0;

  rate_reward = rew_av1
    condition = (FALSE)
    value = 0;
  rate_reward = rew_av2
    condition = (FALSE)
    value = 0;
  rate_reward = rew_av3
    condition = (FALSE)
    value = 0;

  rate_reward_sum = amount
    rewards = rew_d1+rew_d2+rew_d3;

  rate_reward = rew_d1
    condition = (FALSE)
    value = 0;
  rate_reward = rew_d2
    condition = (FALSE)
    value = 0;
  rate_reward = rew_d3
    condition = (FALSE)
    value = 0;

  rate_reward = utilization
    condition = (FALSE)
    value = 0;
```

Figura 5.8: Exemplo A - Recompensas

```
Nome do arquivo: <nome_do_modelo>.SIMUL.<nome_dado_pelo_usuario>
Trecho:

---------------------------------------------------------------
                  R E W A R D S   R E S U L T S
---------------------------------------------------------------
 Simulation time : 30000.000000
 Number of runs  : 5
 Total simulation time : 150000.000000
 Confidence interval   : 95%
 Simulation execution real time : 1124731.033 milliseconds

----------------- Rate Rewards Results -----------------

 Measure: server_queue_GPS.fluido1
   - CR(t) : - mean = 1.3151422278e+01
             - var  = 2.2953023959e+02
             - interval = +/- 1.32797844e+01
               [-1.28362121e-01 , 2.64312067e+01]
   - ACR(t): - mean = 4.3838074261e-04
             - var  = 2.5503359955e-07
             - interval = +/- 4.42659480e-04
               [-4.27873736e-06 , 8.81040223e-04]
   - ATC(t): - mean = 2.2123590522e+01
             - var  = 7.3459178670e+00
             - interval = +/- 2.37571371e+00
               [1.97478768e+01 , 2.44993042e+01]

 Measure: server_queue_GPS.perda1
   - CR(t) : - mean = 8.6090957353e+04
             - var  = 3.3653827931e+08
             - interval = +/- 1.60800837e+04
               [7.00108737e+04 , 1.02171041e+05]
   - ACR(t): - mean = 2.8696985784e+00
             - var  = 3.7393142146e-01
             - interval = +/- 5.36002789e-01
               [2.33369579e+00 , 3.40570137e+00]
   - ATC(t): - mean = 4.2864098884e+04
             - var  = 7.9132518830e+07
             - interval = +/- 7.79737756e+03
               [3.50667213e+04 , 5.06614764e+04]
```

Figura 5.9: Arquivo de Resultados da Simulação

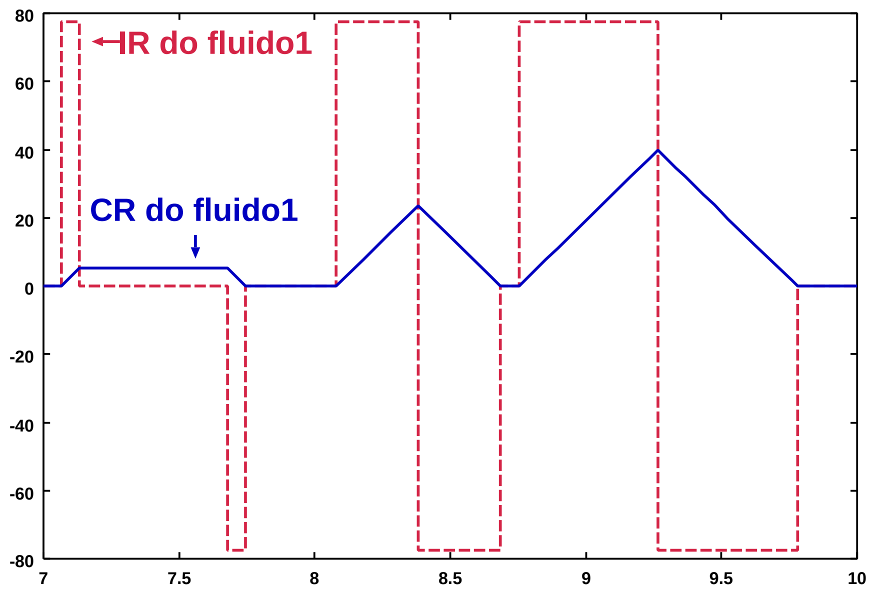


Figura 5.10: Gráfico - IR e CR do Fluido 1 no Tempo

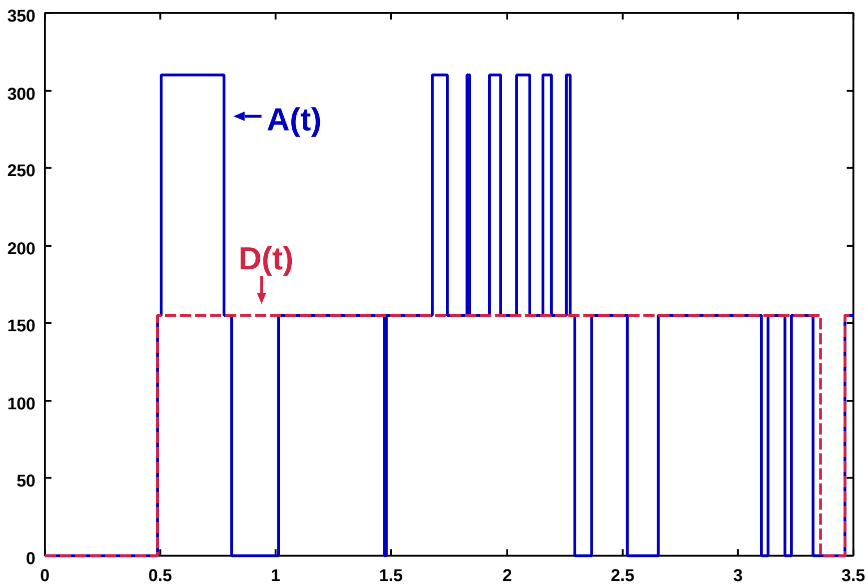


Figura 5.11: Gráfico - Chegada A(t) e Saída D(t) no Tempo

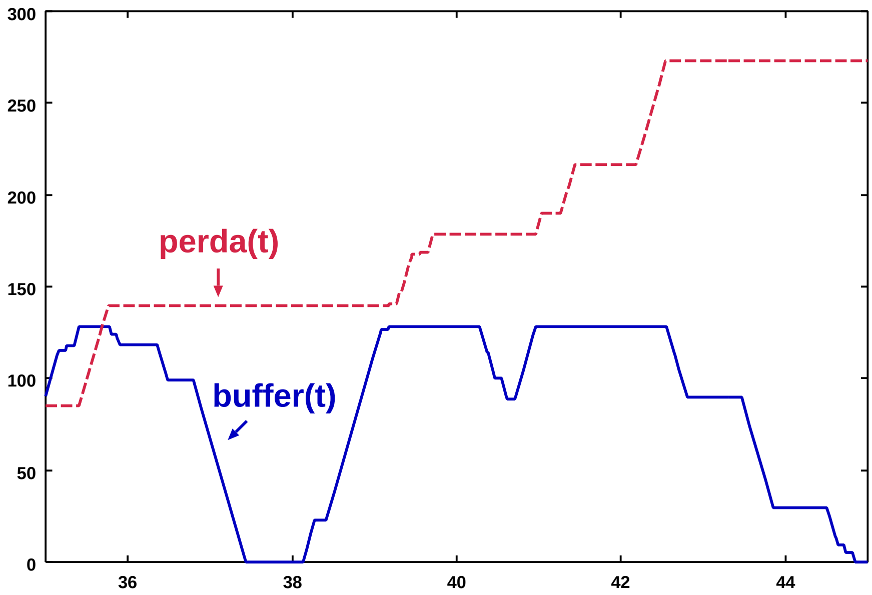


Figura 5.12: Gráfico - Buffer e Perdas no Tempo

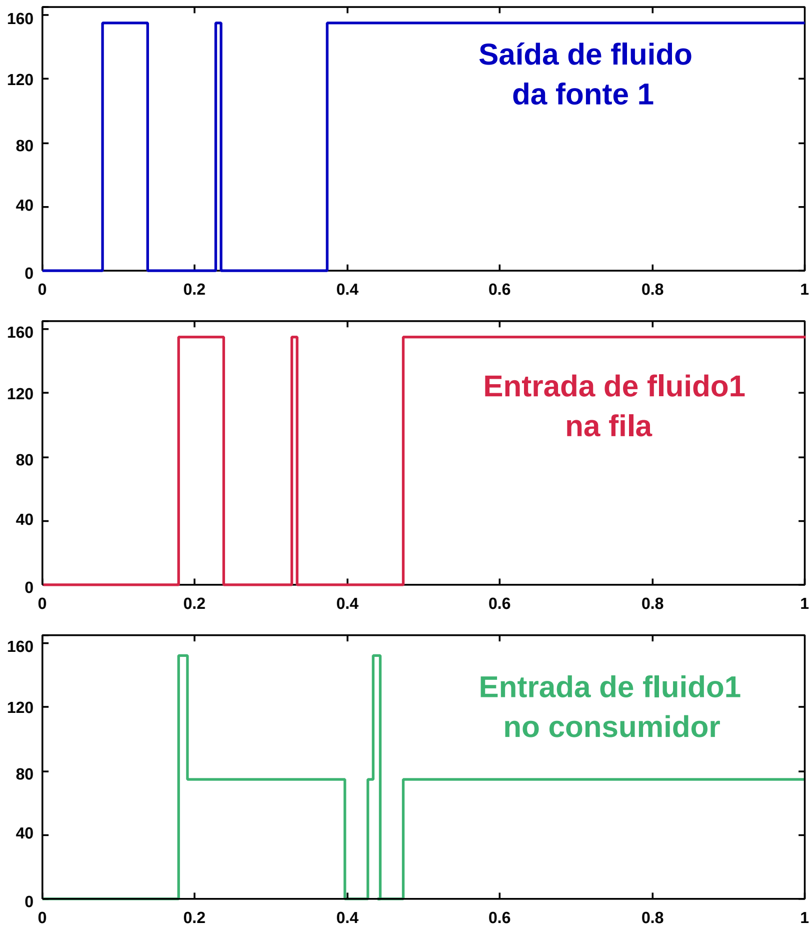


Figura 5.13: Gráfico - Taxas do Fluido1 ao Longo do Percurso

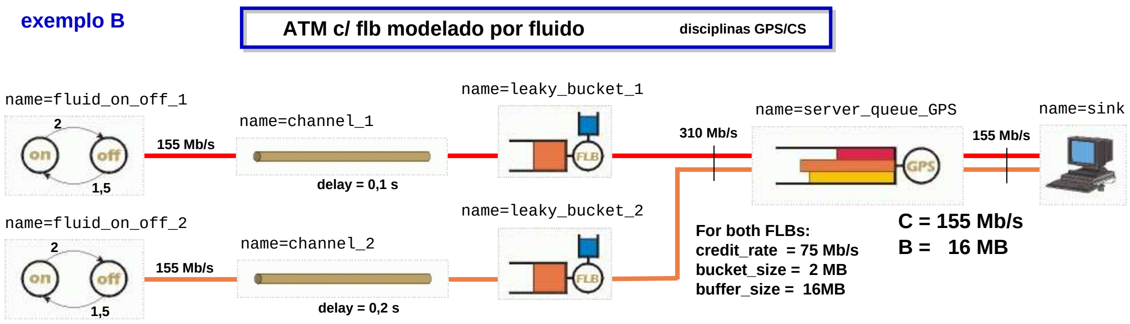


Figura 5.14: Exemplo B - ATM com Tráfego Regulado

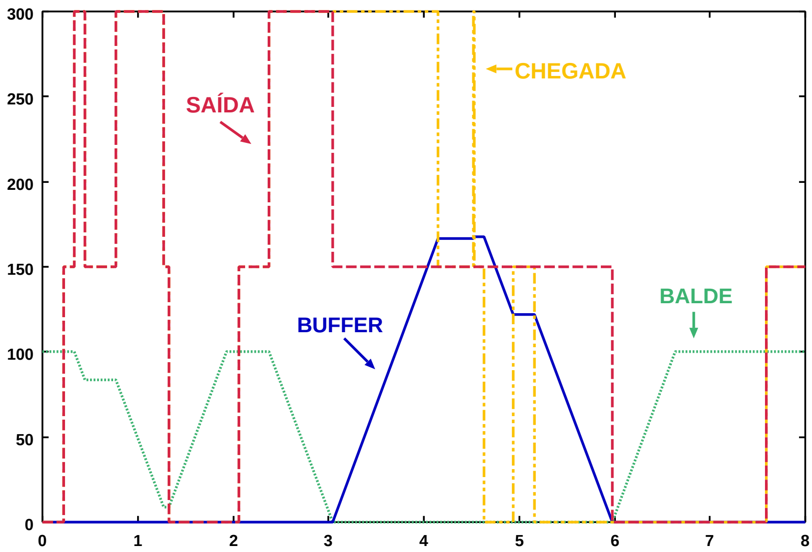


Figura 5.15: Comportamento do Regulador de Tráfego

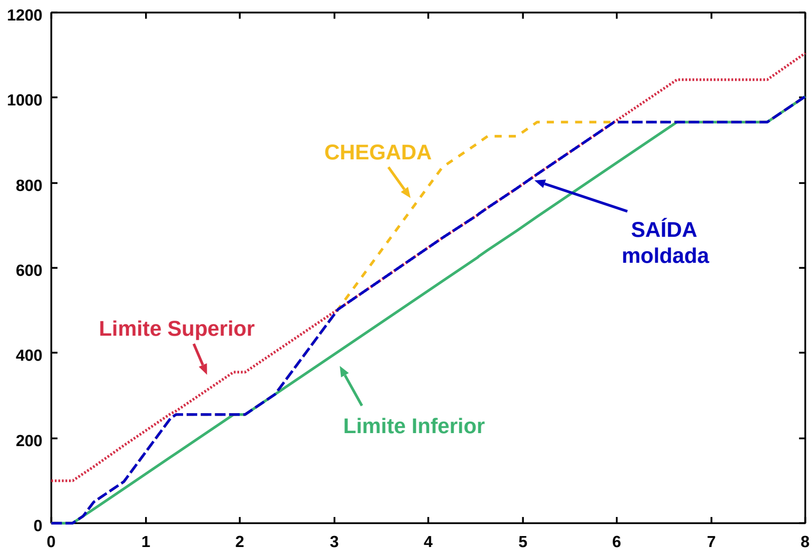


Figura 5.16: Moldagem no Regulador de Tráfego

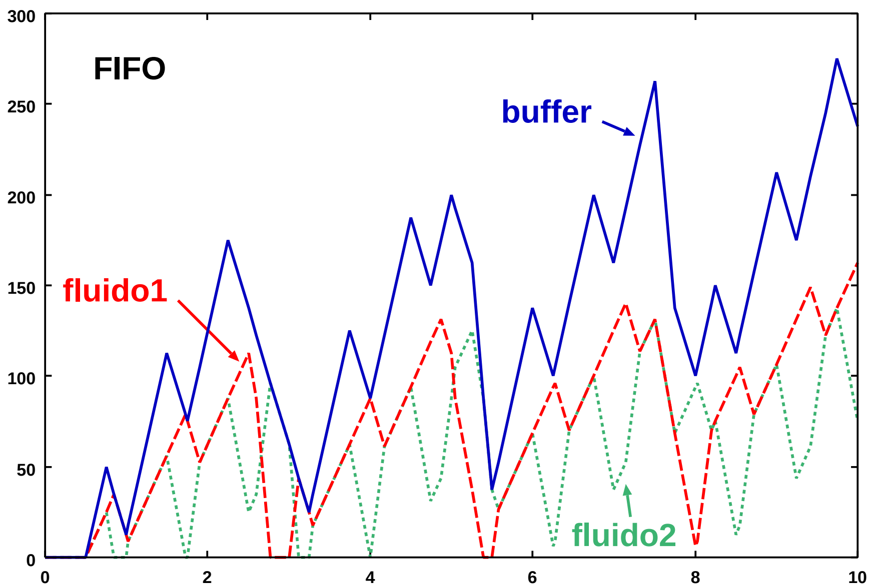


Figura 5.17: Buffer Atendido por FIFO

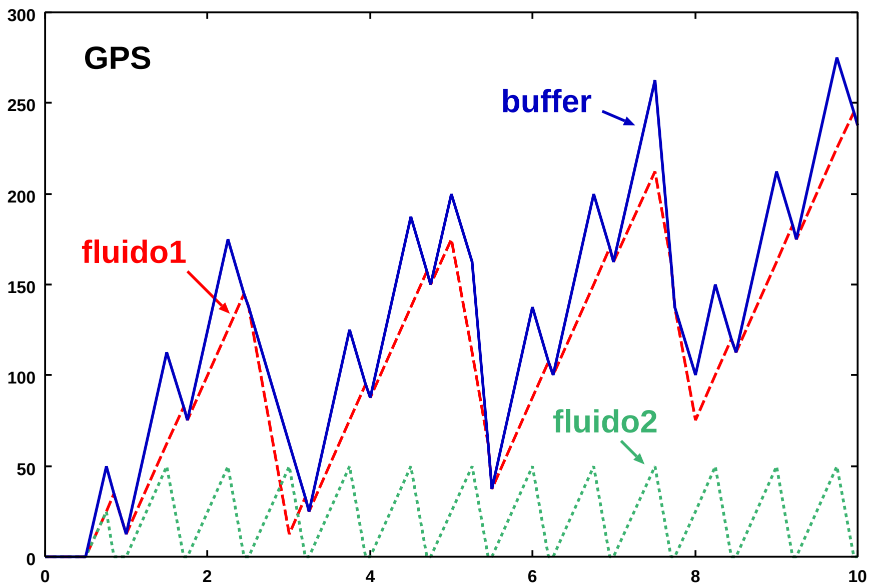


Figura 5.18: Buffer Atendido por GPS

Capítulo 6

Conclusão

Tendo-se como base a importância da modelagem no apoio à engenharia de sistemas, e sabendo-se que a simulação é o método mais abrangente e utilizado para resolução dos modelos, procurou-se uma forma de minimizar seu principal problema, relativo ao alto custo computacional, através da utilização da técnica de fluido, que é promissora nesta área.

Criou-se um paradigma de modelagem de fluido baseado em recompensas onde, o comportamento dos fluidos é modelado através do uso de recompensas de taxa. Esta associação mostrou-se bastante interessante, por sua simplicidade, flexibilidade e capacidade de representar com exatidão o comportamento dos fluidos.

Dentro deste paradigma e focalizado na área de redes de computadores, um simulador de fluido foi idealizado, projetado, construído e validado. Este simulador tem como base o simulador da ferramenta TANGRAM-II, na qual vários recursos foram inseridos, de forma que o simulador pudesse dar suporte a modelos de acordo com a técnica de fluido, além é claro, dos modelos tradicionais.

Para tal desenvolvimento, os diversos elementos de uma rede foram estudados e modelados em forma de objetos, através da aplicação das teorias envolvidas na técnica de fluido. Esta implementação é composta por duas camadas, uma genérica que provê a base para a implementação dos objetos, implementada dentro do código do simulador, e uma camada específica de acordo com cada objeto, implementada dentro do código dos modelos. Do estudo destes objetos e do comportamento dos modelos que são compostos por estes, revelou-se várias características, peculiaridades, vantagens e problemas da técnica.

Para os modelos onde as taxas de geração de pacotes são muitas vezes maiores que as taxas de transição de estados do sistema, a técnica mostrou-se extremamente eficiente. Salienta-se que neste caso enquadram-se a maioria das redes de computadores de alta velocidade da atualidade. Alguns aspectos relativos à qualidade das medidas de interesse são abordados, e na maioria dos casos o erro relativo é bastante pequeno, viabilizando o uso deste tipo de modelagem. Problemas como o chamado *ripple effect* podem degenerar o desempenho dos modelos de fluido em casos onde haja muitos elementos interligados em série ou quando exista realimentação, mas técnicas como agregação de tráfego o minimizam, e fazem com que os modelos de fluido sejam sempre mais velozes do que suas contrapartidas de pacotes.

Dependendo do modelo a ser resolvido, o uso da simulação tradicional pode se tornar inviável, por causa do elevado custo computacional. Nestas situações pode ocorrer de um único segundo de simulação levar horas para ser executado. De uma forma geral conclui-se, que a técnica de fluido expande este limite das simulações atuais, para algumas ordens de grandeza a mais, possibilitando a construção de modelos consideravelmente maiores.

O resultado alcançado neste trabalho foi bastante satisfatório, já que o simulador herdou todas as características de facilidade, poder de modelagem e apresentação de resultados do simulador do TANGRAM-II, e os recursos implementados no simulador do TANGRAM-II são genéricos o suficiente para suportar a modelagem de quaisquer sistemas que se enquadrem à técnica de fluido, podendo estes estarem fora da área de redes ou até mesmo fora da área de computação. Ao mesmo tempo objetos de fluido bem definidos foram criados e podem ser usados como elementos básicos na montagem de diversos tipos de modelos. Com o uso desta biblioteca de objetos, o tempo de aprendizado e construção dos modelos diminui sensivelmente, além de que estes objetos provêm várias medidas de interesse que disponibilizam para o usuário uma ótima visão das propriedades do sistema e permitem que se obtenha um conhecimento aprofundado sobre os elementos e o modelo como um todo.

6.1 Trabalhos Futuros

Três linhas de atuação podem estender o trabalho aqui apresentado: aperfeiçoamentos no simulador, análises da própria técnica de fluido e o uso dos recursos criados para o desenvolvimento de novos modelos.

A respeito de aperfeiçoamentos no simulador, cita-se:

- Vetorização de recompensas e de constantes: são necessárias para que objetos como o fila-servidor possam ser totalmente genéricos em termos de número de classes suportadas. Atualmente os modelos prontos possuem 3 classes e uma extensão neste número, apesar de simples, requer modificações em seu código fonte.

- Vetorização de objetos: para que não seja necessária a cópia de vários objetos idênticos, como por exemplo no caso onde existem várias fontes de tráfego iguais.

- Construção de novos objetos: definir e modelar outros objetos de rede e de sistemas computacionais para expandir a abrangência do simulador.

- Implementação de um cache de estados ou modificação na estrutura do simulador: atualmente para cada estado de simulação, são criados diversos objetos (instâncias de classes na linguagem C++), que representam uma cópia do estado interno da simulação, e após a execução dos passos de transição de estado, toda a estrutura antiga é desalocada. Este é o principal fator da lentidão do simulador e o principal responsável pela diferença de velocidade se comparado ao NetSimul. Este problema pode ser resolvido de duas formas: através da criação de um cache de estados de simulação, que forneceria uma solução boa em termos de desempenho e consumo de memória, ou idealmente, em termos de velocidade, a modificação da estrutura principal, de forma que todo o espaço de estados seja gerado antes do início da execução da simulação. Neste segundo caso o desempenho seria ótimo, mas haveria um maior consumo de memória.

Dentre os itens, os 2 primeiros são de fácil implementação, sendo o terceiro, dependente da complexidade do objeto a que se propõe a modelagem e apenas o último demandaria um trabalho mais apurado e custoso.

Outra linha a ser explorada é a própria pesquisa sobre os modelos de fluido. Dentre os trabalhos estudados, apenas [31] aborda diretamente a qualidade das medidas de interesse, enquanto a imensa maioria apenas aborda os aspectos de ganho computacional. Trabalhos na área de simulação de fluido são recentes, sendo que os primeiros datam de 1996 e o primeiro simulador de fluido de que se tem notícia data de 1999. Apesar de já existirem diversas respostas, a técnica de fluido necessita ainda ser explorada, para que se possa definir com mais exatidão, o quão precisa e rápida ela pode ser e em quais casos ela é melhor aplicada.

Além destes, cita-se como trabalho futuro o uso dos recursos do simulador para a criação de novos modelos de redes de computadores. Modelos mais exatos e mais requintados podem ser construídos, inclusive fora da área de redes. Com isto poder-se-á descobrir, inclusive, outras utilizações para estes novos recursos além das especificadas neste trabalho.

Referências Bibliográficas

[1] B. Liu, Y. Guo, J. K. D. T., e Figueiredo, D. R. A Study of Networks Simulation Efficency: Fluid Simulation vs. Packet-level Simulation. In *INFOCOM* (2001).

[2] B. Liu, Y. Guo, J. K. D. T., e Gong, W. Fluid Simulation of Large Scale Networks: Issues and Tradeoffs . In *PDPTA '99* (1999).

[3] Berson, S., de Souza Silva, E., e Muntz, R. R. *Numerical Solution of Markov Chains*. Marcel Dekker, Inc., 1991, ch. An Object Oriented Methodology for the Specification of Markov Models, pp. 11–36.

[4] Bertsekas, D., e Gallager, R. *Data Networks*, 2 ed. Prentice Hall, 1992.

[5] Carmo, R., de Carvalho, L., de Souza e Silva, E., Diniz, M., e Muntz, R. Performance/Availability Modeling with the TANGRAM-II Modeling Environment. *Performance Evaluation 33* (1998), 45–65.

[6] Cheng, W. C. URL http://bourbon.cs.umd.edu:8001/tgif/. TANGRAM Graphic Interface Facility.

[7] Cisco. Cisco IOS Quality of Service Solutions Configuration Guide. Configuration Guide.

[8] Cisco. URL http://www.cisco.com/univercd/home/home.htm. Technical Documents.

[9] Daniel Ratton e Yang Guo. URL http://gaia.cs.umass.edu/. Site do simulador de fluido FluidSim.

[10] Diniz, M. C. *Técnicas de Solução para Modelos Provenientes de Redes Multimídias.* PhD thesis, Programa de Pós-Graduação de Engenharia de Sistemas e Computação da COPPE/UFRJ, 2000.

[11] Diniz, M. C., e de Souza Silva, E. Especificação e Geração de Modelos Markovianos para Análise de Desempenho e Confiabilidade de Sistemas. *Revista Brasileira de Computação 6*, 3 (1991), 23–42.

[12] E. de Souza Silva, H. G., e Muntz, R. *Computations with Markov Chains - Efficient Solutions for a Class of Non-Markovian Models.* W.J. Stewart - Kluwer Academic Publishers, 1995.

[13] E. Souza Silva, H. G., e Muntz, R. Polling systems with server timeouts and their application to token passing networks. *IEEE/ACM Transactions on Networking 3*, 5 (1995), 560–575.

[14] Figueiredo, D. R. O Módulo de Simulação da Ferramenta TANGRAM-II: Suporte para Medidas com Recompensas, Recursos de Eventos Raros e Aplicações a Modelos de Redes Multimídia. Tese de Mestrado, COPPE/UFRJ, 1999.

[15] Fore. URL http://tactics.marconi.com/index.cgi. Fore Systems white papers, manuals and software downloads.

[16] G. Kesidis, A. Singh, D. C., e Kwok, W. W. Feasibility of Fluid Event-driven Simulation for ATM Networks. In *IEEE GLOBECOM* (1996).

[17] Gail, E. S. S. . H. Calculating cumulative operational time distributions of repairable computer systems. *IEEE Transactions on Computers (Special Issue on Fault Tolerant Computing) C-35*, 4 (April 1986), 322–332.

[18] Gail, E. S. S. . H. An algorithm to calculate transient distributions of cumulative rate and impulse based reward. *Communications in Statistics, Stochastic Models (ISSN=0882-0287) 14*, 3 (1998), 509–536.

[19] Gail, E. S. S. . H. *Computational Probability (ISBN 0-7923-8617-5).* W. Grassmann - Kluwer Academic Publishers, 2000.

[20] Gail, E. S. S. . H. *The Uniformization Method in Performability Analysis - Performability Modeling: Techniques and Tools (ISBN 0-471-49195-0).* John Wiley & Sons, 2001.

[21] Guo, Y., Gong, W., e Towsley, D. F. Time-stepped hybrid simulation (TSHS) for large scale networks. In *INFOCOM (2)* (2000), pp. 441–450.

[22] Kelvin F. Reinhardt, Adriana S. Silva e Adriane Q. Cardozo. URL http://www.land.ufrj.br/. Site do LAND.

[23] Kesidis, G., e Singh, A. An overview of cell-level ATM network simulation. In *High Performance Computing Systems Conf.* (1995), pp. 202–214.

[24] Kleinrock, L. *Queueing Systems* . Wiley-Interscience Publication, 1975.

[25] Kumaram, K., e Mitra, D. Performance and Fluid Simulations of a Novel Shared Buffer Management System. In *Proc. IEEE INFOCOM* (1998).

[26] Kurose, J. F., e Ross, K. W. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison Wesley, 2001.

[27] LAND - Laboratory for modeling, analisys and development of networks and computing systems. *TANGRAM-II user's manual*, April 2001.

[28] Liu, D. R. F. . B. On the specification of ns and other known on-off sources. In *Computer Science Technical Report 00-25* (2000).

[29] Lopes, M. A. G. R. . V. L. R. *Cálculo Numérico: Aspectos Teóricos e Computacionais.* Makron Books, 1996.

[30] Mutlu Arpaci and John A. Copeland. Buffer Management for Shared-Memory ATM Switches. *IEEE Communications Surveys & Tutorials* (2000).

[31] Nicol, D., Goldsby, M., e Johnson, M. Fluid-based simulation of communication networks using SSF, 1999.

[32] Parekh, A. K., e Gallager, R. G. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. In *IEEE/ACM Transactions Networking* (1993).

[33] Raymond Marie, José Incera, D. R. e. G. R. FluidSim: A Tool to Simulate Fluid Models of High-Speed Network. In *Proceedings of TOOLS 2000* (2000).

[34] Ridder, A. Fast simulation of markov fluid models, 1996.

[35] Rosa M.M. Leão, E. d. S. e. S., e de Lucena, S. C. A Set of tools for Traffic Modelling, Analysis and Experimentation. In *Proceedings of TOOLS 2000* (2000).

[36] Rose, O. URL http://nero.informatik.uni-wuerzburg.de/ rose/. MPEG traces.

[37] Ross, S. M. A. *A Course in Simulation.* Prentice-Hall, 1990.

[38] Schwartz, M. *Broadband Integrated Networks.* Prentice Hall, 1996.

[39] Silva, A. P. C. Métodos de Solução para Modelos Markovianos com Recompensa. Tese de Mestrado, COPPE/UFRJ, 2001.

[40] Skelly, P., Schwartz, M., e Dixit, S. A histogram-based model for video traffic behavior in an atm multiplexer. *IEEE/ACM Transactions on Networking 1*, 4 (August 1993), 445–459.

[41] T. Lizambri, F. D., e Wakid, S. Priority Scheduling and Buffer Management for ATM Traffic Shaping. In *Proceedings of 7th IEEE Workshop on Future Trends of Distributed Computing Systems - FTDCS '99* (1999).

[42] Trivedi, K. S. *Probability and Statistics with Reliability, Queuing and Computer Science Applications.* Prentice-Hall, 1982.

[43] Yan, A., e Gong, W. Fluid Simulation for High Speed Networks. In *IEEE Transactions on Information Theory* (1999).

Apêndice A

Disciplinas de Gerenciamento de Espaço em Fila

A seguir são apresentados resumos das principais técnicas de gerenciamento de *buffer* descritas em [41, 30].

Janelas Estáticas (ST - Static Thresholds)

1. **Particionamento Completo (CP - Complete Partitioning)** - Neste esquema a fila é permanentemente particionada em N filas menores, uma para cada fluxo de serviço. Não existe compartilhamento do recurso espaço em fila, entretanto nenhum fluxo é prejudicado por ações dos demais.

2. **Compartilhamento Completo (CS - Complete Sharing)** - Neste esquema a fila é integrada e um pacote que chega é aceito se houver espaço disponível. Aqui existe o máximo aproveitamento do recurso espaço em fila, no entanto um fluxo pode monopolizar o serviço se for mais ávido, vide exemplo *injustiça*.

3. **Compartilhamento com Tamanho Máximo de Fila (SMXQ - Sharing with Maximum Queue Lenghts)** - Também conhecido como compartilhamento restrito de buffer. Nesta disciplina a fila é compartilhada contanto que

um dado fluxo não ultrapasse um valor máximo pré-estabelecido k_i. A idéia é fazer uso do compartilhamento e ao mesmo tempo evitar que algum fluxo possa monopolizar o sistema. Para isto,

$$\Sigma_{i=1}^{N} k_i \geq M \ , \tag{A.1}$$

onde k_i representa a quantidade de buffer que um fluxo pode ocupar, e M é o tamanho total da fila. Apesar desta definição ser genérica, em geral usa-se apenas um parâmetro *threshold* para todos os fluxos, de acordo com a definição a seguir:

$$k_i = \alpha.M, \ i \ \in \ \{1, \ldots, N\} \tag{A.2}$$

$$SMXQ : \frac{1}{N} < \alpha < 1 \tag{A.3}$$

Este parâmetro α, que atua como um limite (*threshold*), define o grau de compartilhamento e pode deixar a disciplina equivalente ao CP se $\alpha = \frac{1}{N}$ ou ao CS se $\alpha = 1$.

4. **Compartilhamento com Alocação Mínima (SMA - Sharing with Minimum Allocation)** - Nesta disciplina cada fila individual $fluxo_i$ têm um mínimo de espaço reservado, sendo o restante do espaço compartilhado. Isto evita o monopólio do recurso.

5. **Compartilhamento com Tamanho Máximo de Fila e Alocação Mínima (SMQMA - Sharing with a Maximum Queue and Minimum Allocation)** - É a integração da SMA e SMXQ. Cada $fluxo_i$ tem acesso a um mínimo de espaço reservado e não pode ultrapassar seu limite k_i. Desfruta de vantagem sobre a SMXQ, pois mesmo sendo esta restritiva em relação ao tamanho máximo de cada fila, pode ocorrer o caso de vários fluxos estarem com carga alta e um fluxo com carga leve sofrer consequências por não haver espaço de armazenamento para suas células.

Até aqui todas as disciplinas apresentadas possuem um comportamento em comum: Um pacote que está chegando é descartado imediatamente se a fila estiver em um estado específico, onde haja necessidade para tal, de forma a manter o compromisso entre vazão e justiça com relação aos fluxos. No entanto existe a possibilidade de que a decisão de descarte seja incorreta, como por exemplo, um pacote descartado de forma a preservar a reserva de um outro fluxo, poderia estar sendo jogado fora em vão, se o outro fluxo não fizesse um da sua reserva. Por este motivo, foram criadas disciplinas de pós-descarte, que evitam este tipo de desperdício.

Pós-Descarte (PO - Push-Out)

Na filosofia pós-descarte, um pacote que chega nunca é descartado se existir espaço na fila como um todo. Se a fila estiver cheia ao chegar um pacote, a perda pode se dar tanto neste que está chegando, quanto num outro pacote previamente armazenado, de acordo com algum critério de decisão. A decisão pode basear-se simplesmente no estado da fila ou em diferentes classes de prioridade.

1. **Medida de Policiamento Posterior (DRP - Delayed Resolution Policy)** - De acordo com [30], a primeira disciplina baseada nesta filosofia foi proposta por Thareja e Agrawala sob o nome DRP (*Delayed Resolution Policy*). Seu funcionamento é idêntico ao descrito ateriormente e dá origem a filosofia pós-descarte.

2. **Descarte sob Demanda (DoD - Drop on Demand)** - Foi proposta por Wei *et al*, sob o nome original *Drop From the Longest Queue*. Segue as mesmas regras da classe PO, sendo que o critério para o descarte é a escolha do último pacote (mais recente) da fila mais longa. Esta disciplina trata todos os fluxos de forma igualitária, o que não permite o tratamento de tráfegos de diferentes prioridades.

3. **Pós-Descarte com Limite (POT - Push-Out with Threshold)** - Nasceu de uma generalização da disciplina DoD, onde diferentes tipos de tráfegos passam a ter diferentes prioridades. Uma idéia similar surgiu independentemente com o nome CSVP (*Complete Sharing with Virtual Partition*). Esta disciplina possui os seguintes atributos: N fluxos dividem o buffer total de

tamanho M, de forma que cada fluxo tem uma fila individual k_i, onde

$$\Sigma_{i=1}^{N} k_i = M , \tag{A.4}$$

Quando o buffer não está cheio, pacotes de qualquer tipo são aceitos. Caso contrário, existe duas possibilidades: Se o pacote que está chegando for do tipo i e sua respectiva fila estiver ocupando um espaço menor do que k_i, então com certeza existe uma fila j, ocupando mais espaço do que deveria. Assim a disciplina de admissão vai aceitar o novo pacote as custas de rejeitar um pacote do tipo j que havia sido previamente aceito. Por outro lado, se a fila do pacote que está chegando for maior do que k_i, então este será rejeitado.

Observa-se que se o pacote descartado for o da chegada, ou se a fila não estiver cheia, a disciplina comporta-se de forma equivalente à CS. No entanto, sob alta carga, a disciplina tende a operar como o CP.

Do ponto de vista teórico esta última classe é justa, eficiente e naturalmente adaptativa. Justa no sentido de que as filas pequenas podem crescer, às custas do descarte que ocorrerá nas mais longas. Eficiente pois permite que nenhum espaço em buffer fique vago quando houver algum pacote ávido por este. E naturalmente adaptativa no que diz respeito ao comportamento da disciplina em relação ao comprimento das filas. Quando muitas filas estão ativas, a rivalidade faz com que seus tamanhos sejam pequenos, no entanto quando apenas uma está ativa, é permitido que seu tamanho atinja grandes dimensões.

O problema da PO concentra-se no campo prático, pois é difícil implementar esta disciplina em roteadores modernos de alta velocidade. Quando o buffer está cheio, uma escrita tem um passo extra do descarte de um pacote, além de que o sistema terá que fazer um controle permanente dos tamanhos das filas e do fato que o descarte de pacotes localizados no meio da fila deixa buracos que não são fáceis de gerenciar.

Políticas Dinâmicas (DP - Dynamic Policies)

As disciplinas baseadas na filosofia de janelas estáticas presumem ambientes onde o comportamento do tráfego não varia significantemmente com o tempo. No entanto sabe-se que o número de fluxos (usuários) pode mudar, as rotas de tráfego podem ser alteradas, assim como a própria demanda de um fluxo pode variar com o tempo. A filosofia pós-descarte lida bem com estas mudanças, mas sabendo-se dos seus custos de implementação, buscou-se soluções alternativas onde o gerenciamento do buffer pode adaptar-se às mudanças e sempre garantir um desempenho ótimo ou próximo deste. Assim sendo, surgiram as disciplinas de gerenciamento dinâmicas que se adaptam às variações de tráfego.

1. **Controle Adaptativo (AC - Adaptative Control)** - Este esquema possui dois elementos chave: identificação e atuação. Identificação refere-se à medição do tráfego, bem como à análise do seu comportamento de forma a identificar a necessidade de correção das ações da disciplina de gerenciamento. A atuação é o ato de aplicar as correções que mudam o comportamento do sistema. A partir do princípio de que a identificação pode ser feita através de medições do tráfego, ou através de estimativas estatísticas e que a atuação é simplesmente o ato de efetuar uma nova alocação de filas, o problema concentra-se na criação de um procedimento de atualização. Em outras palavras, quando e como a atualização deve ocorrer durante a atividade da disciplina compõem o maior desafio a ser superado. Especificamente no controle adaptativo usa-se uma heurística para solucionar o problema citado. Mesmo não tendo uma solução definitiva para o problema, esta disciplina abriu caminho para o surgimento de outras através da sua idéia básica de dinamismo, e dos procesos de identificação e atuação.

2. **Limite Dinâmico (DT - Dynamic Threshold)** - O objetivo é obter uma disciplina de gerenciamento cujo esquema tem a simplicidade do SMXQ e a adaptabilidade do PO. Não existe a necessidade do monitoramento do tráfego de cada porta que ocorre no controle adaptativo. O esquema é baseado na seguinte idéia: o limite (*trheshold*) de cada fila em qualquer instante de tempo

é proporcional ao montante livre do *buffer* como um todo. Um pacote que chega é descartado caso a fila a qual se destina, esteja com um tamanho igual ou maior do que o seu limite k_i. A disciplina é formulada da seguinte maneira:

$$T(t) = \alpha.(M - Q(t)) \ , \tag{A.5}$$

onde $T(t)$ representa o *threshold* para o tamanho das filas, α é uma constante e $Q(t)$ é a soma dos tamanhos das filas individuais.

Com esta fórmula, o DT adapta-se às mudanças de carga que ocorrem no tráfego. Sempre que uma mudança ocorre, o sistema entra em um estado transiente. Passa-se a um exemplo onde um fluxo que está operando com carga baixa aumenta sua atividade. Sua fila individual tende a crescer, fazendo com que a fila global também cresça. Este crecimento vai provocar uma diminuição do limite (*threshold*) e fai fazer com que algumas filas, que possivelmente tenham ficado acima do seu limite, passem a frear os pacotes na entrada enquanto são drenadas pelo servidor. Este processo faz com que exista mais espaço livre para os pacotes do fluxo que acabou de aumentar.

A maior vantagem do DT é a robustez em relação à mudanças de carga no tráfego, o que não ocorre nas disciplinas de janelas estáticas.

Apêndice B

Objetos de Fluido do TANGRAM-II

Neste apêndice são apresentados os objetos de fluido construídos durante o trabalho de tese. Para tal, optou-se por mostrar os objetos dentro de modelos onde estão inseridos, sendo estes modelos, em alguns casos, os mesmos apresentados no capítulo 5. As figuras mostram a imagem representativa do modelo por completo, entretanto somente o objeto em evidência têm seu código fonte listado, em forma de atributos que encontram-se espalhados pelo desenho.

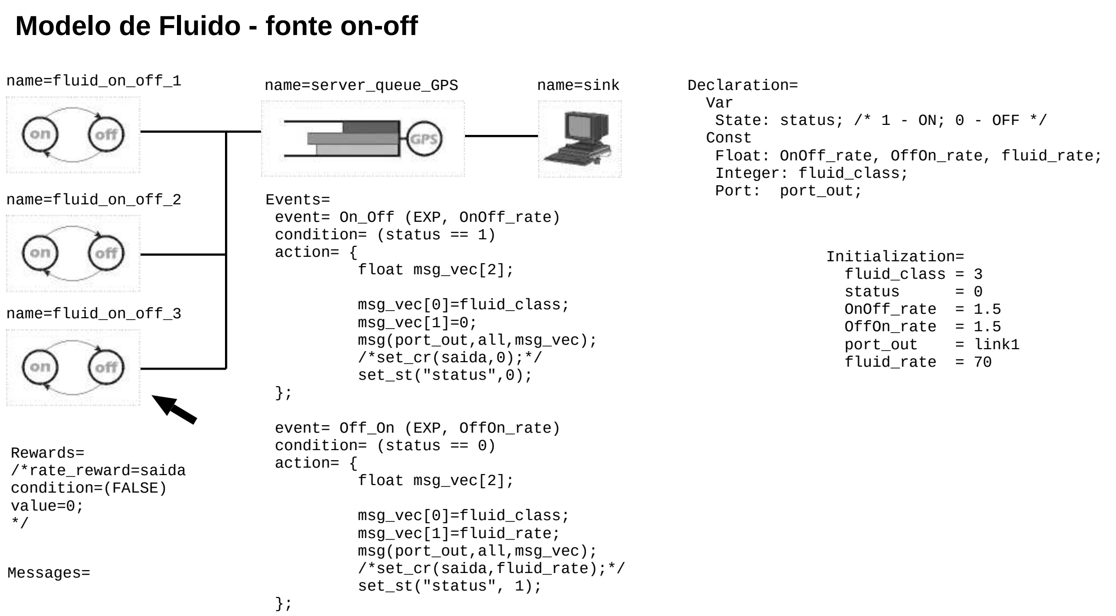
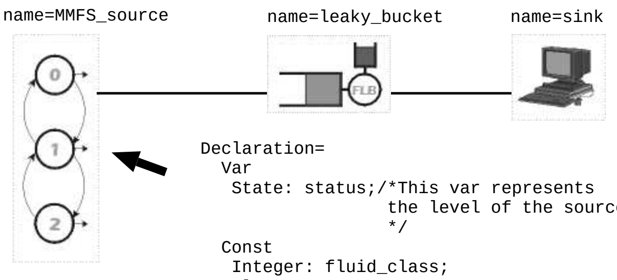


Figura B.1: Fonte on-off

Modelo de Fluido - fonte MMFS




```
Declaration=
  Var
   State: status;/*This var represents
                 the level of the source.
                 */
  Const
   Integer: fluid_class;
   Float  : rate_0, rate_1, rate_2,
            trans_0_1, trans_1_0,
            trans_1_2, trans_2_1;
   Port : port_out;

  Messages=

  Rewards=
  rate_reward=output
  condition=(FALSE)
  value=0;
```

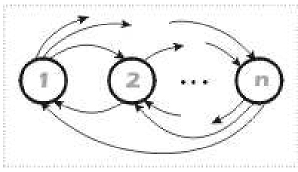
```
Initialization=
 status=0
 fluid_class=1
 rate_0=0
 rate_1=150
 rate_2=300
 trans_0_1=1.5
 trans_1_0=1.5
 trans_1_2=3.0
 trans_2_1=3.0
 port_out=link1
```

```
Events=
 event= Transition_0_1 (EXP, trans_0_1)
 condition= (status == 0)
 action= {
          float msg_vec[2];
          msg_vec[0]=fluid_class;
          msg_vec[1]=rate_1;
          msg(port_out,all,msg_vec);
          set_cr(output,rate_1);
          set_st("status", 1);  };

 event= Transition_1_0 (EXP, trans_1_0)
 condition= (status == 1)
 action= {
          float msg_vec[2];
          msg_vec[0]=fluid_class;
          msg_vec[1]=rate_0;
          msg(port_out,all,msg_vec);
          set_cr(output,rate_0);
          set_st("status", 0);  };

 event= Transition_1_2 (EXP, trans_1_2)
 condition= (status == 1)
 action= {
          float msg_vec[2];
          msg_vec[0]=fluid_class;
          msg_vec[1]=rate_2;
          msg(port_out,all,msg_vec);
          set_cr(output,rate_2);
          set_st("status", 2);  };

 event= Transition_2_1 (EXP, trans_2_1)
 condition= (status == 2)
 action= {
          float msg_vec[2];
          msg_vec[0]=fluid_class;
          msg_vec[1]=rate_1;
          msg(port_out,all,msg_vec);
          set_cr(output,rate_1);
          set_st("status", 1);  };
```

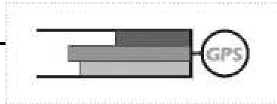
Figura B.2: Fonte MMFS de 3 Estados

Modelo de Fluido - fonte histograma

name=histogram_source



name=server_queue_GPS



name=sink

```
Initialization=

  fluid_class = 1
  uniform_rate = 30
  level    = 1
  port_out = link1

  /* bits/seg */
  tx_1 = 7024512
  tx_2 = 6098688
  tx_3 = 5177088
  tx_4 = 4251648
  tx_5 = 3330048
  tx_6 = 2404224
  tx_7 = 1482624
  tx_8 =  556800

Declaration=
Const
 Integer: fluid_class;
 Float: uniform_rate;
 Float: tx_1,tx_2,tx_3,tx_4,
        tx_5,tx_6,tx_7,tx_8;
 Port:  port_out;

State Var
 Integer: level;

Messages=

Rewards=
/*rate_reward = sent
bounds= 0, 100000
condition = (level == 1)
value = tx_1 - tx_servico;
condition = (level == 2)
value = tx_2 - tx_servico;
condition = (level == 3)
value = tx_3 - tx_servico;
condition = (level == 4)
value = tx_4 - tx_servico;
condition = (level == 5)
value = tx_5 - tx_servico;
condition = (level == 6)
value = tx_6 - tx_servico;
condition = (level == 7)
value = tx_7 - tx_servico;
condition = (level == 8)
value = tx_8 - tx_servico;*/
```

```
Events=
 event= transition_1 (EXP, uniform_rate)
 condition= (level == 1)
 action=
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_1;
     msg(port_out,all,msg_vec); set_st("level",1); }: prob = 0.9671848014;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_2;
     msg(port_out,all,msg_vec); set_st("level",2); }: prob = 0;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_4;
     msg(port_out,all,msg_vec); set_st("level",4); }: prob = 0.0276338515;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_5;
     msg(port_out,all,msg_vec); set_st("level",5); }: prob = 0.0034542314;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_6;
     msg(port_out,all,msg_vec); set_st("level",6); }: prob = 0.0017271157;

 event= transition_2 (EXP, uniform_rate)
 condition= (level == 2)
 action=
   {float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_2;
     msg(port_out,all,msg_vec); set_st("level",2); }: prob = 0.9661904762;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_5;
     msg(port_out,all,msg_vec); set_st("level",5); }: prob = 0.0004761905;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_6;
     msg(port_out,all,msg_vec); set_st("level",6); }: prob = 0.0004761905;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_7;
     msg(port_out,all,msg_vec); set_st("level",7); }: prob = 0.0033333333;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_8;
     msg(port_out,all,msg_vec); set_st("level",8); }: prob = 0.0295238095;

 event= transition_3 (EXP, uniform_rate)
 condition= (level == 3)
 action=
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_3;
     msg(port_out,all,msg_vec); set_st("level",3); }: prob = 0.9667176291;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_6;
     msg(port_out,all,msg_vec); set_st("level",6); }: prob = 0.0009408444;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_7;
     msg(port_out,all,msg_vec); set_st("level",7); }: prob = 0.0054098553;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_8;
     msg(port_out,all,msg_vec); set_st("level",8); }: prob = 0.0269316712;

 event= transition_4 (EXP, uniform_rate)
 condition= (level == 4)
 action=
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_1;
     msg(port_out,all,msg_vec); set_st("level",1); }: prob = 0.0000378601;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_4;
     msg(port_out,all,msg_vec); set_st("level",4); }: prob = 0.9667587930;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_6;
     msg(port_out,all,msg_vec); set_st("level",6); }: prob = 0.0001135804;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_7;
     msg(port_out,all,msg_vec); set_st("level",7); }: prob = 0.0073827282;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_8;
     msg(port_out,all,msg_vec); set_st("level",8); }: prob = 0.0257070382;
```

Figura B.3: Fonte Histograma de 8 Estados

```
event= transition_5 (EXP, uniform_rate)
 condition= (level == 5)
 action=
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_1;
     msg(port_out,all,msg_vec); set_st("level",1); }: prob = 0.0000331367;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_3;
     msg(port_out,all,msg_vec); set_st("level",3); }: prob = 0.0000165684;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_4;
     msg(port_out,all,msg_vec); set_st("level",4); }: prob = 0.0000165684;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_5;
     msg(port_out,all,msg_vec); set_st("level",5); }: prob = 0.9665153423;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_7;
     msg(port_out,all,msg_vec); set_st("level",7); }: prob = 0.0060805885;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_8;
     msg(port_out,all,msg_vec); set_st("level",8); }: prob = 0.0273377957;

 event= transition_6 (EXP, uniform_rate)
 condition= (level == 6)
 action=
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_2;
     msg(port_out,all,msg_vec); set_st("level",2); }: prob = 0.0000245604;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_3;
     msg(port_out,all,msg_vec); set_st("level",3); }: prob = 0.0000573075;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_4;
     msg(port_out,all,msg_vec); set_st("level",4); }: prob = 0.0000081868;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_5;
     msg(port_out,all,msg_vec); set_st("level",5); }: prob = 0.0000081868;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_6;
     msg(port_out,all,msg_vec); set_st("level",6); }: prob = 0.9667125127;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_7;
     msg(port_out,all,msg_vec); set_st("level",7); }: prob = 0.0009742280;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_8;
     msg(port_out,all,msg_vec); set_st("level",8); }: prob = 0.0322150178;

 event= transition_7 (EXP, uniform_rate)
 condition= (level == 7)
 action=
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_1;
     msg(port_out,all,msg_vec); set_st("level",1); }: prob = 0.0000057739;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_2;
     msg(port_out,all,msg_vec); set_st("level",2); }: prob = 0.0000692865;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_3;
     msg(port_out,all,msg_vec); set_st("level",3); }: prob = 0.0002309549;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_4;
     msg(port_out,all,msg_vec); set_st("level",4); }: prob = 0.0011259051;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_5;
     msg(port_out,all,msg_vec); set_st("level",5); }: prob = 0.0021478804;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_6;
     msg(port_out,all,msg_vec); set_st("level",6); }: prob = 0.0005831611;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_7;
     msg(port_out,all,msg_vec); set_st("level",7); }: prob = 0.9701952724;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_8;
     msg(port_out,all,msg_vec); set_st("level",8); }: prob = 0.0256417659;

 event= transition_8 (EXP, uniform_rate)
 condition= (level == 8)
 action=
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_1;
     msg(port_out,all,msg_vec); set_st("level",1); }: prob = 0.0000186232;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_2;
     msg(port_out,all,msg_vec); set_st("level",2); }: prob = 0.0000695265;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_3;
     msg(port_out,all,msg_vec); set_st("level",3); }: prob = 0.0002917631;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_4;
     msg(port_out,all,msg_vec); set_st("level",4); }: prob = 0.0008454922;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_5;
     msg(port_out,all,msg_vec); set_st("level",5); }: prob = 0.0020423417;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_6;
     msg(port_out,all,msg_vec); set_st("level",6); }: prob = 0.0049053446;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_7;
     msg(port_out,all,msg_vec); set_st("level",7); }: prob = 0.0054988032;
   { float msg_vec[2]; msg_vec[0]=fluid_class; msg_vec[1]=tx_8;
     msg(port_out,all,msg_vec); set_st("level",8); }: prob = 0.9863281056;
```

Figura B.4: Fonte Histograma (continuação)

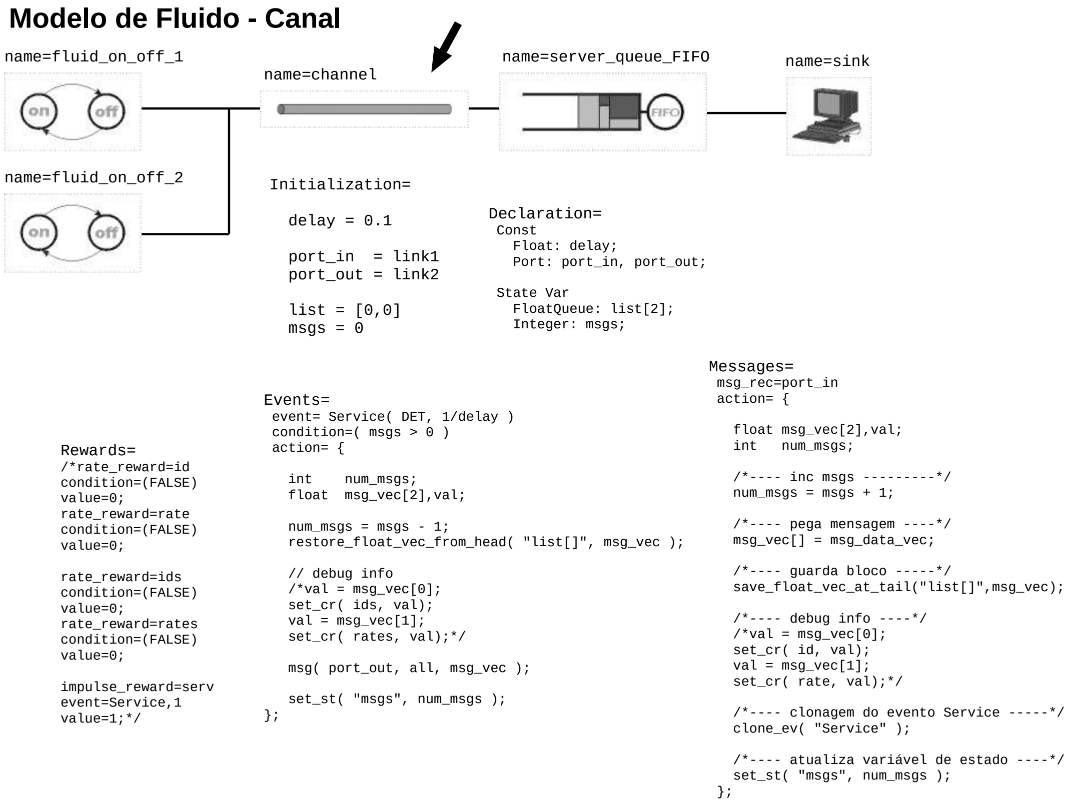


Figura B.5: Canal de Comunicação

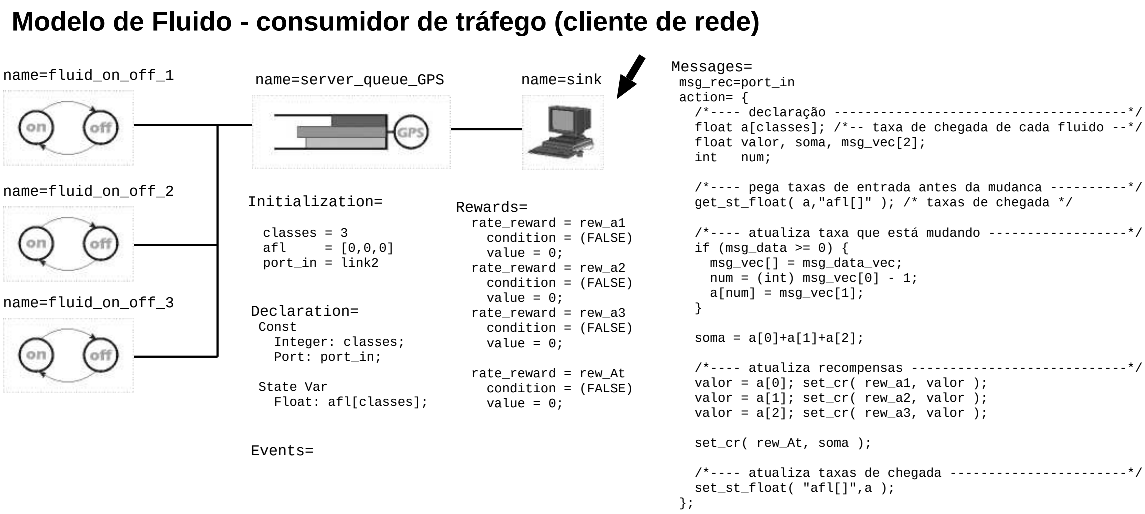
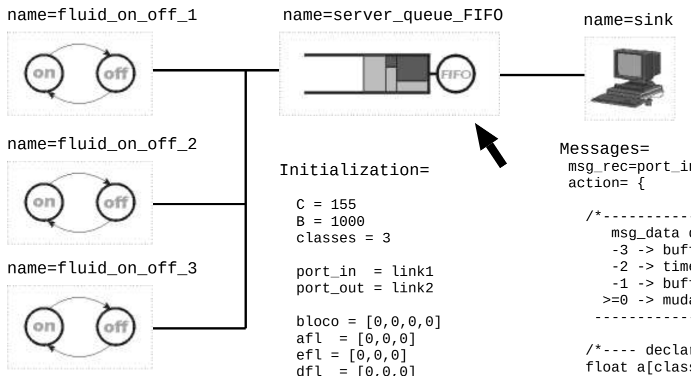


Figura B.6: Consumidor de Tráfego

Modelo de Fluido - disciplina FIFO/CS




```
Declaration=
 Const
   Integer: classes;
   Float: C, B;
   Port: port_in, port_out;
   Object: myself;

 State Var
   FloatQueue: bloco[4];
   Float  : afl[classes],
            dfl[classes],
            efl[classes],
            prox_serv[classes],
            tempo_parcial;
   Integer: ponteiro;

     Events=
      event= t0(REWARD_REACHED)
      condition=( get_cr_sum(buffer) \/ 0 )
      action= {
                msg( port_in, myself, -1 );
              };

      event= tB(REWARD_REACHED)
      condition=( get_cr_sum(buffer) /\ B )
      action= {
                msg( port_in, myself, -3 );
              };

      event= muda_taxa_saida(REWARD_REACHED)
      condition=( get_cr(timer)\/0 )
      action= {
                msg( port_in, myself, -2 );
              };

    Rewards=
      rate_reward_sum = buffer
        bounds = 0,B
        rewards = fluido1+fluido2+fluido3;

      rate_reward = fluido1
        cr_bounds = 0,B
        condition = (FALSE)
        value = 0;
      rate_reward = fluido2
        cr_bounds = 0,B
        condition = (FALSE)
        value = 0;
      rate_reward = fluido3
        cr_bounds = 0,B
        condition = (FALSE)
        value = 0;

      rate_reward = timer
        condition = (TRUE)
        value = -1;

      rate_reward_sum = perda
        rewards = perda1+perda2+perda3;

      rate_reward = perda1
        condition = (FALSE)
        value = 0;
      rate_reward = perda2
        condition = (FALSE)
        value = 0;
      rate_reward = perda3
        condition = (FALSE)
        value = 0;
```

```
Initialization=

  C = 155
  B = 1000
  classes = 3

  port_in  = link1
  port_out = link2

  bloco = [0,0,0,0]
  afl  = [0,0,0]
  efl = [0,0,0]
  dfl  = [0,0,0]
  prox_serv = [0,0,0]

  myself  = server_queue_FIFO

  ponteiro = 0
  tempo_parcial = 0
```

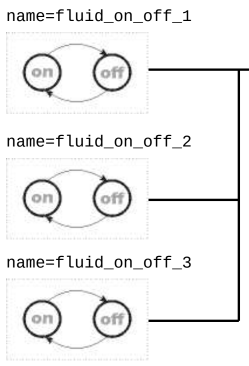
```
Messages=
 msg_rec=port_in
 action= {

   /*-----------------------------------------------------
      msg_data define o tipo de atuação.
      -3 -> buffer cheio (=B)
      -2 -> timer disparou, indicando novas taxas de saída
      -1 -> buffer vazio (=0)
     >=0 -> mudança de taxa de entrada
    ----------------------------------------------------*/

   /*---- declaração ------------------------------------------*/
   float a[classes];     /* taxa de chegada de cada fluido */
   float d[classes];     /* nova taxa de saída de cada fluido */
   float IR[classes];    /* indica o valor da recompensa */
   float IR_perda[classes]; /* taxas de perda */
   float d_tmp[classes]; /* taxas de saida que serão armazenadas*/
   float d_ant[classes]; /* taxas de saida antigas */
   float e[classes];     /* taxas de entrada no buffer */
   int   b[classes], i, bloco_atual, num;
   float At, Et, Xt, At_bck, valor;
   float t1, sobra, epsilon;  /* outras */
   float prox_t, z0, z1, msg_vec[2];
   float dt1,dt2,dt3, tmp_prox[classes], tmp_bloco[4];
   float tmp_tempo_parcial;

   /*---- inicialização ---------------------------------------*/
   epsilon = 1e-9;
   bloco_atual = ponteiro;
   get_st_float ( tmp_prox,"prox_serv[]");
   tmp_tempo_parcial = tempo_parcial;
   prox_t = 0;

   /*---- pega taxas de chegada, entrada e saída ---------------*/
   get_st_float( a, "afl[]"); /* taxas de chegada */
   get_st_float( e, "efl[]"); /* taxas de entrada no buffer */
   get_st_float( d, "dfl[]"); /* taxas de saída */
   get_st_float( d_ant, "dfl[]"); /* taxas de saída antigas */

   /*---- pega instantaneous reward ----------------------------*/
   IR[0] = get_ir ( fluido1 );
   IR[1] = get_ir ( fluido2 );
   IR[2] = get_ir ( fluido3 );

   /*---- valores totais -------------------------------------*/
   Xt = get_cr_sum( buffer );
   At = a[0]+a[1]+a[2];
   Et = e[0]+e[1]+e[2];

   At_bck = At;

   /*---- verifica se evento timer ocorreu ( ==-2 ) ------------*/
   if( msg_data==-2 ){
     do {
       /*---- pega taxas de saída --------*/
       for( i=0; i < classes; i++){
         d[i] = tmp_prox[i];
       }

       bloco_atual = bloco_atual - 1;

       /*---- se existir outro bloco, carregá-lo em tmp_prox -*/
       if ( bloco_atual >= 1 ){
         restore_from_head("bloco[]");
         tmp_prox[0] = bloco[1];
         tmp_prox[1] = bloco[2];
         tmp_prox[2] = bloco[3];
         tmp_prox[3] = bloco[4];
         prox_t = bloco[0];
         set_cr(timer,prox_t);

         tmp_tempo_parcial = tmp_tempo_parcial - prox_t;
       }
     } while( (bloco_atual > 0) && (prox_t <= epsilon) );
   }
   else {
     if ( msg_data != -3 ){

       /*---- atualiza taxa que está mudando ---------------------*/
       if (msg_data >= 0) { /* se a tx < 0 eh t0 ou timer --*/
         msg_vec[] = msg_data_vec;
         num = (int) msg_vec[0] - 1;
         a[num] = msg_vec[1];
       }

       /*---- valores totais --( recalcula At pois a[i] mudou )---*/
       At = a[0]+a[1]+a[2];
```

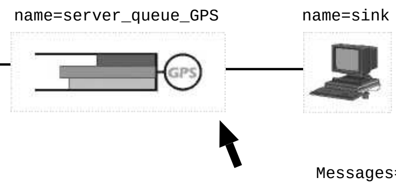
Figura B.7: Fila FIFO / CS

```
      /*---- distribuicao da banda ------------------------------*/
      /*---- disciplina de atendimento: FIFO --------------------*/
      if( Xt <= epsilon ){
        /*--- caso onde d[i] vai ser usado imediatamente ---*/
        if (At <= C){
          d[0]=a[0]; d[1]=a[1]; d[2]=a[2];
        }
        else {
          d[0] = (a[0]/At)*C;
          d[1] = (a[1]/At)*C;
          d[2] = (a[2]/At)*C;
        }
      }
      else
      {
        /*--- caso onde d[i] vai ser guardado ---*/
        if (At == 0){
          d_tmp[0]=0; d_tmp[1]=0; d_tmp[2]=0;
        }
        else {
          d_tmp[0] = (a[0]/At)*C;
          d_tmp[1] = (a[1]/At)*C;
          d_tmp[2] = (a[2]/At)*C;
        }
      }
      /*---- neste momento temos d[] segundo FIFO. ----------------*/

      /*---- msg_data carregando nova taxa ( msg_data >= 0) ------*/
      if (msg_data != -1){ /* entra aqui somente quando mudou a[i]*/
        if ( Xt > epsilon ){
          if (bloco_atual == 0) {
            /*--- tratamento da fila ---*/
            t1 = Xt/C;
            if ( t1 < epsilon ){
              t1 = epsilon;
            }

            /*--- guarda bloco ---*/
            set_cr( timer, t1 );

            tmp_prox[0] = d_tmp[0];
            tmp_prox[1] = d_tmp[1];
            tmp_prox[2] = d_tmp[2];
          }
          else{
            z1 = tempo_parcial;
            z0 = get_cr(timer);
            t1 = Xt/C - z1 - z0;
            if ( (t1 < epsilon) || (At_bck == 0) ){
              t1=0;
            }
            /*--- guarda bloco ---*/
            tmp_tempo_parcial = z1 + t1;
            tmp_bloco[0] = t1;
            tmp_bloco[1] = d_tmp[0];
            tmp_bloco[2] = d_tmp[1];
            tmp_bloco[3] = d_tmp[2];
            save_float_vec_at_tail("bloco[]",tmp_bloco);
          }

          bloco_atual = bloco_atual + 1;
        }
      }
    } /* fim de != -3 */

    /*---- caso o buffer esteja cheio ---------------------------*/
    if ( Xt >= (B - classes*epsilon) ) {
      if ( At > C ){
        for ( i=0; i < classes; i++ ){
          e[i] = a[i] / At * C;

          /*---- calculo da perda de fluido Xt==B ----*/
          IR_perda[i] = (a[i]-e[i]);
        }
      }
      else{
        for ( i=0; i < classes; i++ ){
          e[i] = a[i];

          /*---- calculo da perda de fluido Xt==B ----*/
          IR_perda[i] = 0;
        }
      }
    }
    else{

      /*---- calculo da perda de fluido Xt < B ----*/
      for ( i=0; i < classes; i++ ){
        e[i] = a[i];
        IR_perda[i] = 0;
      }
    }

    /*---- atualiza IR das perdas -------------------------------*/
    valor = IR_perda[0];
    set_ir( perda1, valor );
    valor = IR_perda[1];
    set_ir( perda2, valor );
    valor = IR_perda[2];
    set_ir( perda3, valor );
  }

  /*---- atualiza IR dos fluidos -------------------------------*/
  valor = e[0] - d[0];
  set_ir( fluido1, valor );
  valor = e[1] - d[1];
  set_ir( fluido2, valor );
  valor = e[2] - d[2];
  set_ir( fluido3, valor );

  /*---- envia novas taxas de saida ------*/
  for( i=0; i < classes; i++){
    if (!((d_ant[i]<=d[i]+epsilon) && (d_ant[i]>=d[i]-epsilon))) {
      msg_vec[0] = i + 1;
      msg_vec[1] = d[i];
      msg( port_out, all, msg_vec );
    }
  }

  /*---- atualiza variáveis de estado ----*/
  set_st( "ponteiro",bloco_atual );
  set_st_float( "afl[]",a );
  set_st_float( "efl[]",e );
  set_st_float( "dfl[]",d );
  set_st_float( "prox_serv[]",tmp_prox );
  set_st_float( "tempo_parcial",tmp_tempo_parcial );

};
```

Figura B.8: Fila FIFO / CS (continuação)

Modelo de Fluido - disciplinas GPS/CS






```
Initialization=

  C = 155 /* Mb/s */
  B = 64  /* 8MB  */

  classes = 3

  afl = [0,0,0]
  dfl = [0,0,0]

  fi1 = 0.333333
  fi2 = 0.333333
  fi3 = 0.333334

  port_in  = link1
  port_out = link2
  myself = server_queue_GPS
```

```
Declaration=
 Const
   Integer: classes;
   Float: C, B;
   Float: fi1, fi2, fi3; /* pesos cuja soma=1 */

   Port: port_in, port_out;
   Object: myself;

 State Var
   Float: afl[classes], dfl[classes];
```

```
Events=
 event= t0(REWARD_REACHED)
 condition=( (get_cr(fluido1)\/0) || (get_cr(fluido2)\/0) ||
             (get_cr(fluido3)\/0) )
 action= {
          msg( port_in, myself, -1 );
        };

 event= tB(REWARD_REACHED)
 condition=( get_cr_sum(buffer)/\B )
 action= {
          msg( port_in, myself, -1 );
        };
```

```
Rewards=
  rate_reward_sum = buffer
    bounds = 0,B
    rewards = fluido1+fluido2+fluido3;

  rate_reward = fluido1
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido2
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;
  rate_reward = fluido3
    cr_bounds = 0,B
    condition = (FALSE)
    value = 0;

  rate_reward_sum = perda
    rewards = perda1+perda2+perda3;

  rate_reward = perda1
    condition = (FALSE)
    value = 0;
  rate_reward = perda2
    condition = (FALSE)
    value = 0;
  rate_reward = perda3
    condition = (FALSE)
    value = 0;

  /* debug info */

  rate_reward = rew_a1
    condition = (FALSE)
    value = 0;
```

```
Messages=
 msg_rec=port_in
 action= {

   /*---- declaração -------------------------------------------*/
   float x[classes];     /* buffer p/ cada fluido - nível-CR(t) */
   float a[classes];     /* taxa de chegada de cada fluido */
   float a_bck[classes]; /* backup de a[] */
   float d[classes];     /* nova taxa de saída de cada fluido */
   float IR[classes];    /* indica o valor da recompensa */
   float d_ant[classes]; /* taxas de saída antigas */
   float peso[classes];  /* fi */
   float IR_perda[classes]; /* taxas de perda */
   float av[classes];    /* taxas de entrada no sistema */
   int   b[classes], num, i;
   float At, Xt, soma, valor, sobra;  /* outras */
   float epsilon, msg_vec[2];

   /*---- inicialização ---------------------------------------*/
   epsilon = 1e-9;

   /*---- pega pesos ------------------------------------------*/
   peso[0] = fi1; peso[1] = fi2; peso[2] = fi3;

   /*---- pega taxas antes da mudanca -------------------------*/
   get_st_float( a,"afl[]" ); /* taxas de chegada */
   get_st_float( d_ant,"dfl[]" ); /* taxas de chegada antigas */

   /*---- pega instantaneous reward ---------------------------*/
   IR[0] = get_ir ( fluido1 );
   IR[1] = get_ir ( fluido2 );
   IR[2] = get_ir ( fluido3 );

   /*---- atualiza taxa que está mudando ----------------------*/
   if (msg_data >= 0) {  /* se msg_data < 0 é t0 ou tB! */
     get_msg_data_float_vec( msg_vec );
     num = (int) msg_vec[0] - 1;
     a[num] = msg_vec[1];
   }

   for( i=0; i < classes; i++){
     a_bck[i] = a[i];
   }

   /*---- lê rewards que descrevem o buffer -------------------*/
   x[0] = get_cr ( fluido1 ); /* cumulative reward */
   x[1] = get_cr ( fluido2 ); /* idêntico a get_rew */
   x[2] = get_cr ( fluido3 );
   Xt = get_cr_sum( buffer );

   /*---- caso o buffer esteja cheio --------------------------*/
   if ( Xt >= (B - classes*epsilon) ) {

     soma = a[0]+a[1]+a[2];

     if ( soma > C ){
       for ( i=0; i < classes; i++ ){
         av[i] = a[i] / soma * C;
         /*---- calculo da perda de fluido Xt==B ----*/
         IR_perda[i] = (a[i]-av[i]);
       }
     }
     else{
       for ( i=0; i < classes; i++ ){
         av[i] = a[i];
         /*---- calculo da perda de fluido Xt==B ----*/
         IR_perda[i] = 0;
       }
     }

     /*---- para manter a compatibilidade ----*/
     a[0] = av[0]; a[1] = av[1]; a[2] = av[2];
   }
   else{
     /*---- calculo da perda de fluido Xt < B ----*/
     for ( i=0; i < classes; i++ ){
       IR_perda[i] = 0;
     }
   }
```

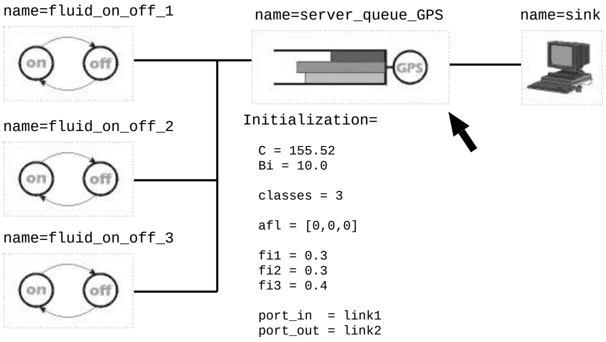
Figura B.9: Fila GPS / CS

```
rate_reward = rew_a2
   condition = (FALSE)
   value = 0;
 rate_reward = rew_a3
   condition = (FALSE)
   value = 0;

 rate_reward = rew_av1
   condition = (FALSE)
   value = 0;
 rate_reward = rew_av2
   condition = (FALSE)
   value = 0;
 rate_reward = rew_av3
   condition = (FALSE)
   value = 0;

 rate_reward = rew_d1
   condition = (FALSE)
   value = 0;
 rate_reward = rew_d2
   condition = (FALSE)
   value = 0;
 rate_reward = rew_d3
   condition = (FALSE)
   value = 0;

 rate_reward = rew_IR1
   condition = (FALSE)
   value = 0;
 rate_reward = rew_IR2
   condition = (FALSE)
   value = 0;
 rate_reward = rew_IR3
   condition = (FALSE)
   value = 0;

 rate_reward = rew_At
   condition = (FALSE)
   value = 0;
 rate_reward = rew_Dt
   condition = (FALSE)
   value = 0;
```

```
  /*---- distribuição da banda --------------------------------*/
  /*---- disciplina de atendimento: GPS -----------------------*/
  for( i=0; i < classes; i++){
    b[i]=0;
    d[i]=0;
  }
  sobra = C;
  for( i=0; i < classes; i++){
    if ((a[i] > 0)||(x[i]>0)) b[i]=1;
  }


  /*---- atualiza IR dos fluidos ------------------------------*/
  valor = a[0] - d[0];
  set_ir( fluido1, valor );
  valor = a[1] - d[1];
  set_ir( fluido2, valor );
  valor = a[2] - d[2];
  set_ir( fluido3, valor );

  do { /*--- se bi==0 então o fluido i já está ok! ------*/

    soma = (peso[0]*b[0] + peso[1]*b[1] + peso[2]*b[2] );

    if (soma > 0) {
      for( i=0; i < classes; i++){
        d[i] = d[i] + peso[i]*b[i] / soma * sobra;
      }
    }

   /* cálculo da sobra */
    sobra=0;
    for( i=0; i < classes; i++){
      if ( ((a[i]-d[i])<0) && (x[i]==0) && (b[i]==1) ) {
        sobra = sobra+d[i]-a[i];
        d[i] = a[i]; b[i] = 0;
      }
    }
  } while ( (sobra > 0) && ((b[0]==1)||(b[1]==1)||(b[2]==1)) );
  /*---- neste momento temos d[] segundo GPS. -----------------*/

  /*---- atualiza IR dos fluidos ------------------------------*/
  valor = a[0] - d[0];
  set_ir( fluido1, valor );
  valor = a[1] - d[1];
  set_ir( fluido2, valor );
  valor = a[2] - d[2];
  set_ir( fluido3, valor );

  /*---- atualiza IR das perdas -------------------------------*/
  valor = IR_perda[0];
  set_ir( perda1, valor );
  valor = IR_perda[1];
  set_ir( perda2, valor );
  valor = IR_perda[2];
  set_ir( perda3, valor );

  /*---- envia novas taxas de saída --(ROTEAMENTO)-------------*/
  for( i=0; i < classes; i++){
    if (!((d_ant[i]<=d[i]+epsilon) && (d_ant[i]>=d[i]-epsilon))) {
      msg_vec[0] = i + 1;
      msg_vec[1] = d[i];
      msg_float_vec( port_out, all, msg_vec );
    }
  }

  /*---- debug info ----*/
  valor = a_bck[0]; set_cr( rew_a1, valor );
  valor = a_bck[1]; set_cr( rew_a2, valor );
  valor = a_bck[2]; set_cr( rew_a3, valor );
  valor = a[0]; set_cr( rew_av1, valor );
  valor = a[1]; set_cr( rew_av2, valor );
  valor = a[2]; set_cr( rew_av3, valor );
  valor = d[0]; set_cr( rew_d1, valor );
  valor = d[1]; set_cr( rew_d2, valor );
  valor = d[2]; set_cr( rew_d3, valor );
  valor = IR[0]; set_cr( rew_IR1, valor );
  valor = IR[1]; set_cr( rew_IR2, valor );
  valor = IR[2]; set_cr( rew_IR3, valor );

  valor = a_bck[0]+a_bck[1]+a_bck[2];
  set_cr( rew_At, valor );
  valor = d[0]+d[1]+d[2];
  set_cr( rew_Dt, valor );

  /*---- atualiza taxas de chegada ----------------------------*/
  set_st_float( "afl[]",a_bck );
  set_st_float( "dfl[]",d );
};
```

Figura B.10: Fila GPS / CS (continuação)

Modelo de Fluido - disciplinas GPS/CP




```
Initialization=

  C = 155.52
  Bi = 10.0

  classes = 3

  afl = [0,0,0]

  fi1 = 0.3
  fi2 = 0.3
  fi3 = 0.4

  port_in  = link1
  port_out = link2

  myself  = server_queue_GPS
```

```
Declaration=
 Const
   Integer: classes;
   Float: C, Bi;
   Float: fi1, fi2, fi3; /* pesos cuja soma=1 */

   Port: port_in, port_out;
   Object: myself;

 State Var
   Float: afl[classes];
```

```
Events=
 event= muda_taxa_t0(REWARD_REACHED)
 condition=( (get_cr(fluido1)\/0)||(get_cr(fluido2)\/0)||(get_cr(fluido3)\/0) )
 action= {
          msg( port_in, myself, -1 );
        };
```

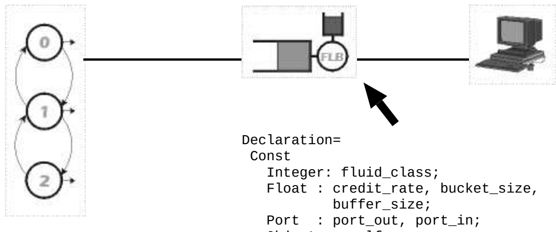
```
Rewards=
  rate_reward_sum = buffer
    rewards = fluido1+fluido2+fluido3;

  rate_reward = fluido1
    cr_bounds = 0,Bi
    condition = (FALSE)
    value = 0;
  rate_reward = fluido2
    cr_bounds = 0,Bi
    condition = (FALSE)
    value = 0;
  rate_reward = fluido3
    cr_bounds = 0,Bi
    condition = (FALSE)
    value = 0;

  /* debug info */
  rate_reward = rew_a1
    condition = (FALSE)
    value = 0;
  rate_reward = rew_a2
    condition = (FALSE)
    value = 0;
  rate_reward = rew_a3
    condition = (FALSE)
    value = 0;

  rate_reward = rew_d1
    condition = (FALSE)
    value = 0;
  rate_reward = rew_d2
    condition = (FALSE)
    value = 0;
  rate_reward = rew_d3
    condition = (FALSE)
    value = 0;

  rate_reward = rew_IR1
    condition = (FALSE)
    value = 0;
  rate_reward = rew_IR2
    condition = (FALSE)
    value = 0;
  rate_reward = rew_IR3
    condition = (FALSE)
    value = 0;
```

```
Messages=
 msg_rec=port_in
 action= {

   /*---- declaração -------------------------------------------*/
   float X[classes];     /* buffer p/ cada fluido - nível-CR(t) */
   float a[classes];     /* taxa de chegada de cada fluido */
   float d[classes];     /* nova taxa de saída de cada fluido */
   float IR[classes];    /* indica o valor da recompensa */
   float d_ant[classes]; /* taxas de saída antigas */
   float peso[classes];  /* fi */
   int   b[classes], i, num;
   float At, soma, valor, sobra;  /* outras */
   float epsilon, msg_vec[2];

   /*---- inicialização ----------------------------------------*/
   epsilon = 1e-9;

   /*---- pega pesos -------------------------------------------*/
   peso[0] = fi1; peso[1] = fi2; peso[2] = fi3;

   /*---- pega taxas de entrada antes da mudanca ---------------*/
   get_st_float( a,"afl[]" ); /* taxas de chegada */

   /*---- pega instantaneous reward ----------------------------*/
   IR[0] = get_ir ( fluido1 );
   IR[1] = get_ir ( fluido2 );
   IR[2] = get_ir ( fluido3 );

   /*---- calcula taxas de saída antes da mudança -------------*/
   for( i=0; i < classes; i++){
     d_ant[i] = a[i] - IR[i];
   }

   /*---- atualiza taxa que está mudando -----------------------*/
   if (msg_data >= 0) {  /* se a taxa for negativa eh t0!!! */
     get_msg_data_float_vec( msg_vec );
     num = (int) msg_vec[0] - 1;
     a[num] = msg_vec[1];
   }

   /*---- lê rewards que descrevem o buffer --------------------*/
   X[0] = get_cr ( fluido1 ); /* cumulative reward */
   X[1] = get_cr ( fluido2 ); /* idêntico a get_rew */
   X[2] = get_cr ( fluido3 );

   /*---- distribuição da banda --------------------------------*/
   /*---- disciplina de atendimento: GPS -----------------------*/
   for( i=0; i < classes; i++){
     b[i]=0;
     d[i]=0;
   }
   sobra = C;
   for( i=0; i < classes; i++){
     if ((a[i] > 0)||(X[i]>0)) b[i]=1;
   }

   do { /*--- se bi==0 então o fluido i já está ok! ------*/

     soma = (peso[0]*b[0] + peso[1]*b[1] + peso[2]*b[2] );
     if (soma > 0) {
       for( i=0; i < classes; i++){
         d[i] = d[i] + peso[i]*b[i] / soma * sobra;
       }
     }
     /* cálculo da sobra */
     sobra=0;
     for( i=0; i < classes; i++){
       if ( ((a[i]-d[i])<0) && (X[i]==0) && (b[i]==1) ) {
         sobra = sobra+d[i]-a[i];
         d[i] = a[i]; b[i] = 0;
       }
     }
   } while ( (sobra > 0) && ((b[0]==1)||(b[1]==1)||(b[2]==1)) );
   /*---- neste momento temos d[] segundo GPS. -----------------*/

   /*---- atualiza IR dos fluidos ------------------------------*/
   valor = a[0] - d[0];
   set_ir( fluido1, valor );
   valor = a[1] - d[1];
   set_ir( fluido2, valor );
   valor = a[2] - d[2];
   set_ir( fluido3, valor );

   /*---- envia novas taxas de saída --(ROTEAMENTO)-------------*/
   for( i=0; i < classes; i++){
     if (!((d_ant[i]<=d[i]+epsilon) && (d_ant[i]>=d[i]-epsilon))) {
       msg_vec[0] = i + 1;
       msg_vec[1] = d[i];
       msg_float_vec( port_out, all, msg_vec );
     }
   }

   /*---- debug info ----*/
   valor = a[0]; set_cr( rew_a1, valor );
   valor = a[1]; set_cr( rew_a2, valor );
   valor = a[2]; set_cr( rew_a3, valor );
   valor = d[0]; set_cr( rew_d1, valor );
   valor = d[1]; set_cr( rew_d2, valor );
   valor = d[2]; set_cr( rew_d3, valor );
   valor = IR[0]; set_cr( rew_IR1, valor );
   valor = IR[1]; set_cr( rew_IR2, valor );
   valor = IR[2]; set_cr( rew_IR3, valor );

   /*---- atualiza taxas de chegada ----------------------------*/
   set_st_float( "afl[]",a );

 };
```

Figura B.11: Fila GPS / CP

Modelo de Fluido - Regulador de Tráfego
Fluid Leaky Bucket

name=MMFS_source name=leaky_bucket name=sink




```
Declaration=
 Const
   Integer: fluid_class;
   Float : credit_rate, bucket_size,
           buffer_size;
   Port  : port_out, port_in;
   Object: myself;

 Initialization=
  fluid_class=1
  credit_rate=150
  bucket_size=100
  buffer_size=500
  port_in=link1
  port_out=link2
  myself = leaky_bucket

 Rewards=

 rate_reward = BUCKET
 cr_bounds = 0,bucket_size
 cr_init_val = bucket_size
 condition = (TRUE)
 value = 0;

 rate_reward = BUFFER
 bounds = 0,buffer_size
 condition = (FALSE)
 value = 0;

 rate_reward = ARRIVAL
 condition = (FALSE)
 value = 0;

 /* debug info */

 rate_reward = DEPARTURE
 condition = (FALSE)
 value = 0;

 rate_reward = A_CR
 condition = (FALSE)
 value = 0;

 rate_reward = D_CR
 condition = (FALSE)
 value = 0;

 rate_reward = LOWB_CR
 condition = (FALSE)
 value = 0;

 rate_reward = UPPB_CR
 cr_init_val = bucket_size
 condition = (FALSE)
 value = 0;

 impulse_reward = num_ev
 event = empty,1
 value = 1;
```

```
    Events=
    event= empty(REWARD_REACHED)
     condition=( (get_cr(BUFFER) \/ 0 ) || (get_cr(BUCKET) \/ 0 ) )
     action= {
              msg( port_in, myself, -1 );
            };
    event=full_bucket(REWARD_REACHED)
     condition=( get_cr(BUCKET) /\ bucket_size )
     action= {
            float Dp;

            Dp = get_cr(DEPARTURE);
            if (Dp > credit_rate){
              set_ir(LOWB_CR,credit_rate);
              set_ir(UPPB_CR,credit_rate);
            }
            else {
              set_ir(LOWB_CR,Dp);
              set_ir(UPPB_CR,Dp);
            }
    };

Messages=
 msg_rec=port_in
 action= {

          /*----- declaração ---------------------------------*/
          float CRt, CRp; /* nivel-CR(t) */
          float IRt, IRp; /* taxas de chegada de cada fluido   */
          float Ap,Dp;    /* Chegada e saida da fila de pct.   */
          float Dt_prev, Dp_prev; /* taxas de saida anteriores */
          float msg_vec[2], epsilon;

          /*----- inicialização ------------------------------*/
          epsilon = 1e-9;

          /*----- leitura dos rewards ------------------------*/
          CRp = get_cr ( BUFFER ); /*CR(t)*/
          CRt = get_cr ( BUCKET );
          Dp_prev = get_cr ( DEPARTURE );
          IRp = get_ir ( BUFFER ); /*IR(t)*/
          IRt = get_ir ( BUCKET );

          /*----- pega taxa de entrada -----------------------*/
          if (msg_data >= 0){
            /* se a taxa for -1 veio do evento vazio*/
            msg_vec[] = msg_data_vec;

            Ap = msg_vec[1];
            set_cr( ARRIVAL, Ap );
          }
          else Ap = get_cr ( ARRIVAL );
          set_ir(A_CR,Ap);

          /*----- cálculo das taxas do LB -----*/
          if (CRt > 0){
             Dp = Ap;
             IRt = credit_rate - Ap;
             IRp = 0;
          }
          if (CRt == 0){
            if (Ap > credit_rate ){
               Dp = credit_rate;
               IRt = 0;
               IRp = Ap - credit_rate;
            }
            else {
              if ( CRp == 0 ){
                 Dp = Ap;
                 IRt = credit_rate - Ap;
                 IRp = 0;
              }
              else {
                 Dp = credit_rate;
                 IRt = 0;
                 IRp = Ap - credit_rate;
              }
            }
          }
          set_ir (BUFFER,IRp);
          set_ir (BUCKET,IRt);

          /*----- aqui envia msg notificando mudanca de Dp --*/
          if ( !((Dp_prev<=Dp+epsilon) && (Dp_prev>=Dp-epsilon)) ) {
            msg_vec[0] = fluid_class;
            msg_vec[1] = Dp;
            msg( port_out, all, msg_vec );
          }

          /*----- daqui p/ baixo so info de debug -----*/
          set_cr (DEPARTURE,Dp); /* taxa de saida */
          set_ir (D_CR,Dp); /* Integral de Dp */
          if (CRt == bucket_size)
            if (Dp > credit_rate){
              set_ir(LOWB_CR,credit_rate);
              set_ir(UPPB_CR,credit_rate);
            }
            else {
              set_ir(LOWB_CR,Dp);
              set_ir(UPPB_CR,Dp);
            }
   };
```

Figura B.12: Regulador de Tráfego

Apêndice C

Parâmetros Ajustáveis

No apêndice B os objetos de fluido foram apresentados na íntegra. Entretanto, são necessárias algumas instruções para que o usuário possa utilizar o objeto sem a necessidade de analisar e compreender seu código fonte, como se fosse uma caixa preta de comportamento bem definido. Esta descrição, escrita em língua inglesa, é apresentada a seguir.

C.0.1 Fluid Objects Parameters

In addition to the provided fluid examples, there are several ready to use objects in TGIF fluid domain. This section describes the parameters that should be set to garantee the correct work of each component in the model context.

on-off source

Five parameters must be specified in Initialization attribute.

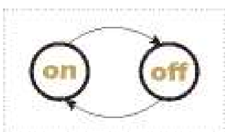


Figura C.1: on-off

- fluid_class - Indicates the class number that will receive the fluid traffic generated by the source in the subsequent queue server object. Typical values range between 1 and 3, in the presented examples.
- OnOff_rate - Indicates the transition rate from on state to off state.
- OffOn_rate - Indicates the transition rate from off state to on state.
- fluid_rate - Indicates the rate volume of the fluid that will be generated in on state.
- port_out - Responsible for the connection between the source and the subsequent object. A string name should be specified, and it must be the same of the port_in in the next downstream object. Two equal names indicates the direct association of the objects.

three state MMFS source

Five parameters must be specified in Initialization attribute.

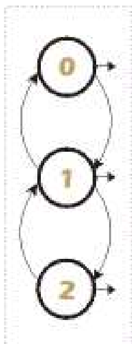


Figura C.2: 3 state MMFS

- fluid_class - Indicates the class number that will receive the fluid traffic generated by the source in the subsequent queue server object. Typical values range between 1 and 3, in the presented examples.
- rate_0 - Indicates the rate volume of the fluid that will be generated in 0 state.

- rate_1 - Indicates the rate volume of the fluid that will be generated in 1 state.
- rate_2 - Indicates the rate volume of the fluid that will be generated in 2 state.
- trans_0_1 - The transition rate from state 0 to state 1.
- trans_1_0 - The transition rate from state 1 to state 0.
- trans_1_2 - The transition rate from state 1 to state 2.
- trans_2_1 - The transition rate from state 2 to state 1.
- port_out - Responsible for the connection between the source and the subsequent object.

channel

Three parameters must be specified in Initialization attribute.

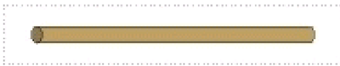


Figura C.3: channel

- delay - Indicates the the propagation delay in the channel. Values should be greater than 0.
- port_in - A string name that indicates where the fluid came from.
- port_out - String responsible for the connection with the subsequent object.

sink

One parameter must be specified in Initialization attribute. Note: This object can support up to 3 classes of fluid.

- port_in - A string name that indicates where the fluid came from.

Figura C.4: sink

server_queue_FIFO - CS

Five parameters must be specified in Initialization attribute. Note: This object can support up to 3 classes of fluid.

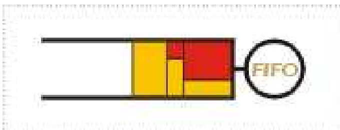


Figura C.5: server_queue - FIFO

- B - A float number representing the shared buffer size.
- C - A float number representing the service rate capacity.
- port_in - A string name that indicates where the fluid came from.
- port_out - String responsible for the connection with the subsequent object.
- myself - This string must contain the object name.

Important: The routing is done by this object, so you must set in the Messages attribute the output port and the fluid class for each departure rate. This part of the code is labeled by "/*—- send new departure rates — (ROUTING) —– */". By default all messages are sent to the port_out and the fluid class remais the same (fluid 1 is send to fluid class 1 of the subsequent queue).

server_queue_GPS - CS

Eight parameters must be specified in Initialization attribute. Note: This object can support up to 3 classes of fluid.

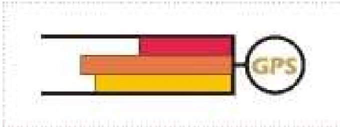


Figura C.6: server_queue - GPS

- B - A float number representing the shared buffer size.
- C - A float number representing the service rate capacity.
- fi1 - This is a float number that represents the weight of the fluid class 1.
- fi2 - This is a float number that represents the weight of the fluid class 2.
- fi3 - This is a float number that represents the weight of the fluid class 3.
- port_in - A string name that indicates where the fluid came from.
- port_out - String responsible for the connection with the subsequent object.
- myself - This string must contain the object name.

Note: The sum $fi1 + fi2 + fi3$ must be equal to 1.
Important: The routing is done by this object, so you must set in the Messages attribute the output port and the fluid class for each departure rate. The same as server_queue_FIFO - CS object.

server_queue_GPS - CP

Eight parameters must be specified in Initialization attribute. Note: This object can support up to 3 classes of fluid.

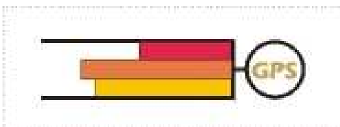


Figura C.7: server_queue - GPS

- Bi - A float number representing the buffer size for each class.
- C - A float number representing the service rate capacity.
- fi1 - This is a float number that represents the weight of the fluid class 1.
- fi2 - This is a float number that represents the weight of the fluid class 2.
- fi3 - This is a float number that represents the weight of the fluid class 3.
- port_in - A string name that indicates where the fluid came from.
- port_out - String responsible for the connection with the subsequent object.
- myself - This string must contain the object name.

Note: The sum $fi1 + fi2 + fi3$ must be equal to 1.
Important: The routing is done by this object, so you must set in the Messages attribute the output port and the fluid class for each departure rate. The same as server_queue_FIFO - CS object.

fluid_leaky_bucket

Seven parameters must be specified in Initialization attribute. This object can support just one fluid class.

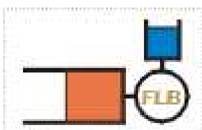


Figura C.8: Fluid Leaky Bucket

- fluid_class - Indicates the class number that will receive the fluid traffic released by the flb in the subsequent queue server object. Typical values range between 1 and 3, in the presented examples.
- credit_rate - A float number that indicates the rate of credit generation for the bucket.

- bucket_size - Indicates the size of the bucket as float type constant.
- buffer_size - Indicates the size of the data buffer as float type constant.
- port_in - A string name that indicates where the fluid came from.
- port_out - String responsible for the connection with the subsequent object.
- myself - This string must contain the object name.