UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS E TECNOLOGIA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

UM SIMULADOR PARA AVALIAÇÃO DE DESEMPENHO DE REDES LOCAIS

ANTONIO CAUPER FILHO

CAMPINA GRANDE – PB

DEZEMBRO / 1987

ANTONIO CAUPER FILHO

UM SIMULADOR PARA AVALIAÇÃO DE DESEMPENHO DE REDES LOCAIS

Dissertação apresentada ao Curso de **MESTRADO EM SISTEMAS E COMPUTAÇÃO** da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do Grau de Mestre.

JOSÉ ANTÃO BELTRÃO MOURA

Orientador

JACQUES PHILIPPE SAUVÉ

Co-orientador

C373s Cauper Filho, Antonio
Um simulador para avaliacao de desempenho de redes locais / Antonio Cauper Filho. - Campina Grande, 1987.
136 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Redes de Computadores 2. Avaliacao de Desempenho 3. Simulacao 4. Eventos 5. Simulacao Acionada por Eventos 6. Dissertacao I. Moura, Jose Antao Beltrao II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.7(043)

Para Rozangela, Lucas e Silas

AGRADECIMENTOS

Agradeço ao Dr. José Antão Beltrão Moura pela atenção dispensada, bem como pelas contribuições essenciais ao desenvolvimento deste trabalho além da paciência em ler esta dissertação várias vezes.

Agradeço também ao Dr. Jacques Philippe Sauvé por sua atenção assim como pelas contribuições substanciais a este trabalho.

Quero agradecer também a Dra. Maria Izabel Cavalcanti Cabral e ao Prof. José Homero Feitosa Cavalcanti por suas contribuições.

E agradeço a Deus que me deu força e disposição para chegar ao fim deste trabalho.

UM SIMULADOR PARA AVALIAÇÃO DE DESEMPENHO DE REDES LOCAIS

ANTONIO CAUPER FILHO

DISSERTAÇÃO APROVADA EM 22/12/87

JOSÉ ANTÃO BELTRÃO MOURA - Ph.D
- Orientador -

JACQUES PHILIPPE SAUVÉ -Ph.D
- Co-orientador -

JORGE CHAVES DE MEDEIROS - Dr.
- Componente da Banca -

MARIA IZABEL CAVALCANTI CABRAL - Dr.
- Componente da Banca -

MARCOS ANTONIO GONÇALVES BRASILEIRO - Dr.
- Componente da Banca -

CAMPINA GRANDE - Pb
DEZEMBRO - 1987

UM SIMULADOR PARA AVALIAÇÃO DE DESEMPENHO DE REDES LOCAIS

RESUMO

É sempre muito importante saber se um recurso está sendo ou vai ser bem utilizado.

No caso de redes locais não é diferente. Portanto, desenvolvemos aqui um Simulador que ajuda a descobrir se uma rede local, a ser instalada, atenderá a demanda.

Este simulador foi desenvolvido na linguagem de programação C e é aberto a modificações. Esta facilidade foi implementada graças ao modelo de simulação que utilizamos.

Finalmente, para validar este simulador comparamos os seus resultados com resultados de [MOUR 86].

ABSTRACT

It's always very important to know if a resource is being used or it's going to be well used.

In case of locals NETWARE (LAN) is not different. However, we develop here a simulator that helps to find out if a LAN, to be installed, will atend the demand.

This simulator was developed in a programation language C and it is opened to receive modifications. This facility was implement because of a simulation model that we use, thanks to it.

Finaly, to validate this simulator, we compare the results with the results of [MOUR 86].

SUMÁRIO

FIGURAS

TABELAS

CAPÍTULO 1

INTRODUÇÃO

A organização de uma rede local (RL), estabelece a ligação de vários usuários à sub-rede de comunicação (sistema de comunicação) através de estações (veja figura 1.1).

Figura 1.1 Organização de uma Rede Local

A sub-rede de comunicação é composta das estações e, geralmente, de um único meio de transmissão que é o responsável pelas transferências de informação entre as estações. O meio de transmissão é comumente disposto numa topologia em anel (veja a Figura 1.2) ou em barra (veja a Figura 1.3) [IEEE 802].

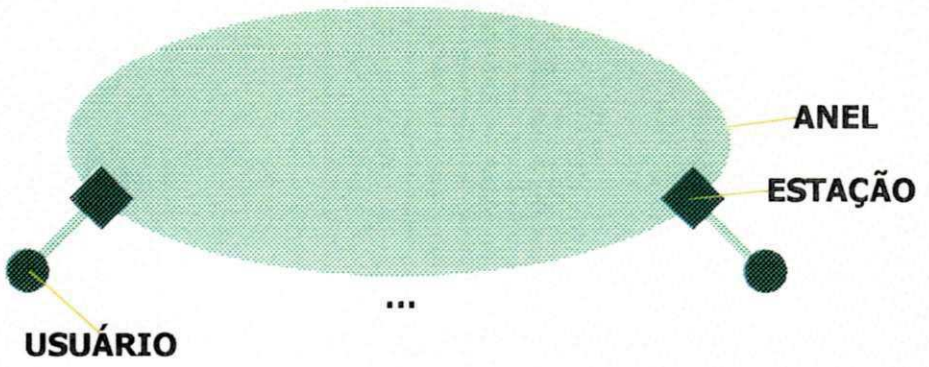

Figura 1.2 Anel

Figura 1.3 Barra

A fim de ordenar o uso do meio de transmissão pelas várias estações na sub-rede de comunicação, as estações seguem as regras do chamado "protocolo de acesso ao meio", ou simplesmente "protocolo de acesso". Em cima deste protocolo, definem-se outros protocolos que possibilitam a comunicação entre usuários da RL e que viabilizam as suas aplicações distribuídas. É de interesse avaliar o desempenho dos protocolos de redes locais para que se possa determinar a adequação de possíveis arquiteturas alternativas para o atendimento de determinadas aplicações.

1.1. MOTIVAÇÃO

Muito se tem feito em avaliação de desempenho de redes locais. Dentre os muitos trabalhos desenvolvidos nesta área podemos citar:

a) um estudo do desempenho da sub-rede de comunicação em anel e em barra através de Simulação Digital em [MOUR 83];

b) estudos analíticos de sub-redes locais em [BUX 81] e [MOUR 83/84];

c) em [MOUR 85] é estudado o desempenho de protocolos de transporte;

d) [WONG 83] mostra uma maneira analítica de se estudar o desempenho de redes locais a nível de aplicação.

Como podemos notar nos exemplos de trabalhos de avaliação de desempenho descritos anteriormente, todos os estudos disponíveis baseiam-se em duas importantes técnicas para avaliação de desempenho: a simulação e as ferramentas analíticas.

As ferramentas analíticas baseiam-se em **processos estocásticos** e **teoria das filas**. Os problemas existentes nesta técnica (ferramentas analíticas), são a sua dificuldade de manuseio, e as simplificações dos modelos em estudo, a partir das hipóteses feitas, que diminuem o poder "prático" destas ferramentas.

A simulação pode ser manual, usada quando o sistema a ser simulado é pequeno e/ou simples, ou digital quando feita em computador. Neste trabalho usaremos a técnica de simulação digital. Por brevidade, adotaremos apenas o termo "simulação".

A simulação é bem mais simples que as ferramentas analíticas e sempre funciona, servindo também para validar aproximações destas últimas. Mas é cara, em termos de desenvolvimento por necessitar de mão de obra especializada e de execução por que é demorada. Além disso, como os simuladores são escritos para resolverem problemas específicos, geralmente são descartados após o uso. Acrescente-se que a construção "ad hoc" de simuladores leva também à dificuldade de reusá-los em problemas ligeiramente diferentes. Nestes casos, é necessário desenvolvê-los por completo. Esta dissertação apresenta um simulador que minimiza estas limitações.

Faz-se necessário fazermos a distinção entre dois tipos de simulação: a gatilhada no tempo e a gatilhada ou acionada por eventos. Uma boa definição de evento é a encontrada em [MOUR 86]: "é uma perturbação no

estado do sistema" – a chegada de pacote para uma fila na interface da sub-rede de comunicação.

A simulação gatilhada no tempo é aquela onde o tempo (por exemplo: o relógio da máquina) vai passando e à medida que o tempo passa os eventos vão ocorrendo. Já na simulação acionada por eventos o relógio é simulado e quando os eventos ocorrem o relógio é adiantado para o instante desta ocorrência, assim o relógio não tem qualquer relação com um relógio real, como o relógio interno da máquina, por exemplo.

Na simulação gatilhada por eventos podemos simular dias de funcionamento de um sistema, em questão de minutos, já na gatilhada no tempo não, temos que esperar os dias passarem. A execução do simulador acionado por eventos é muito mais eficiente, por isto esta é a escolha – simulação acionada por eventos – neste trabalho.

É objeto desta dissertação, um simulador, acionado por eventos, para avaliação de desempenho de redes locais que seja fácil de modificar para novas aplicações, que auxilie o projetista de RL no dimensionamento de novas redes e alterações em redes já instaladas.

1.2. CONTRIBUIÇÕES DESTA DISSERTAÇÃO

O simulador deveria ter ainda as seguintes características:

a) fácil interação com o usuário, facilitando seu uso por leigos em informática;

b) fácil integração com várias alternativas para diferentes níveis de protocolos da rede;

c) modelar a camada de enlace do IEEE-802 [IEEE 802] e [GIOZ 86];

d) modelar a sub-camada de acesso do IEEE 802 [IEEE 802] e [GIOZ 86]:

- anel com passagem de ficha;
- barra com passagem de ficha;
- barra com CSMA-CD;

e) geração de tráfego de acordo com a aplicação;

f) possibilidade de incluir detalhes do protocolo de transporte; e

g) levantamento de estatísticas.

A contribuição deste trabalho vem da importância que ele tem para o usuário deste simulador interessado em medir o desempenho de RL e principalmente para o projetista de RL, pois reduz o tempo de desenvolvimento de ferramentas para avaliação de desempenho, por oferecer um simulador pronto, ou necessitando de poucas modificações para se adequar a várias arquiteturas ou aplicações de RL.

É importante notar que nosso objetivo não é avaliar o desempenho de RL já existentes (isto já foi feito), mas sim, apresentar uma ferramenta útil para a implementação e alteração de RL.

1.3. ORGANIZAÇÃO DA DISSERTAÇÃO

O conteúdo do restante desta dissertação está organizado como segue:

O capítulo 2 contém a concepção e a descrição de um modelo genérico para a construção de um simulador de RL, acionado por eventos.

O capítulo 3 contém a descrição das estruturas de dados usadas pelo simulador, além de vários comentários em torno de sua implementação.

O capítulo 4 contém um pequeno manual de uso e alterações do simulador aqui desenvolvido.

No capítulo 5 apresentamos três estudos de casos, para validação deste simulador.

Finalmente, no capítulo 6 apresentamos as nossas conclusões e as sugestões de continuidade deste trabalho.

CAPÍTULO 2

MODELO GENÉRICO DE SIMULAÇÃO PARA UMA REDE LOCAL

2.1. INTRODUÇÃO

Como um modelo é uma abstração do sistema em estudo [MOUR 86], o que tentaremos fazer neste capítulo é abstrair sucintamente o máximo de informação sobre as redes locais mais conhecidas, para chegarmos ao modelo proposto pelo título deste capítulo. Para ilustrarmos o procedimento adotado, consideremos uma aplicação de controle de estoque distribuído.

Suponhamos que, por necessidade de agilização de manutenção, o almoxarifado de uma Universidade X, funcione da seguinte forma: um almoxarifado central, com materiais de uso geral, e um almoxarifado específico por setor (por exemplo: no Departamento de Computação existe o almoxarifado de suprimentos para computadores). O almoxarifado central é quem repõe os estoques de todo o sistema de Almoxarifado da Universidade. E para isto, mantém as informações de todo o sistema. Este sistema caracteriza a aplicação de controle de estoque distribuído, do qual falamos anteriormente.

Suponhamos agora que esta aplicação seja automatizada e funcione em uma rede local, com a seguinte configuração (a nível global):

a) Um concentrador de dados (servidor de arquivos), localizado no almoxarifado central;

b) Um microcomputador por setor com processamento local e ligado à rede (caracterizando um usuário do servidor de arquivos).

Cada setor mantém, também, o seu controle de estoque, enviando um arquivo com as informações atualizadas de seu estoque, para o almoxarifado central, no fim do expediente.

No caso de falta de um item de um almoxarifado setorial, este setor pergunta ao almoxarifado central se outro almoxarifado o tem em estoque, e se afirmativo, solicita o devido remanejamento, através de um pedido eletrônico de material, que é, na realidade, um arquivo contendo todas as informações inerentes ao item em falta.

Como os buffers são limitados, os arquivos são transmitidos por partes, denominadas pacotes. Cada pacote recebe então um cabeçalho que contém as informações: destino, fonte, dígitos de verificação, comprimento do pacote e outras informações que compõem a chamada identificação do pacote. Estas informações que são acrescidas aos dados de cada pacote são chamadas de overhead. O acréscimo do overhead aos pacotes é feito pela Aplicação do usuário que deseja transmitir, ou seja, aplicação fonte (veja a Figura 2.1).

Depois que o overhead é acrescido ao pacote, este é submetido à **estação fonte** que providenciará a sua transmissão para a **estação remota**, obedecendo as regras do protocolo de acesso ao meio de transmissão (por exemplo: o protocolo em anel com passagem de ficha, veja a sua descrição no item b da seção 2.3).

A estação remota ao receber um pacote, pode mandar um aviso de recebimento (pacote de reconhecimento) deste pacote para a estação fonte, se assim o usuário desejar, e aciona a aplicação remota para receber este pacote. A aplicação remota por sua vez, ao receber o último pacote, toma as providências solicitadas pela aplicação fonte, podendo até transmitir um arquivo de volta dizendo por exemplo, que o item de estoque solicitado já foi alocado e que está à disposição do usuário fonte (almoxarifado setorial que solicitou o material). Neste instante a aplicação fonte passa a ser remota e vice-versa e o processo de transmissão de arquivos se reinicia.

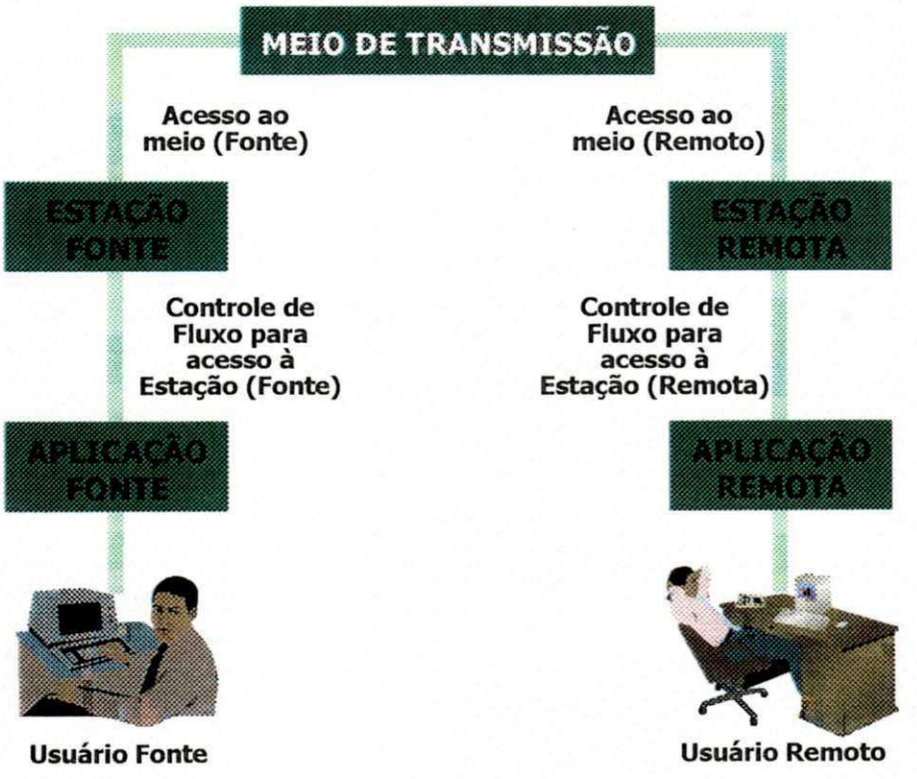

Figura 2.1 Ligação de um usuário fonte a um remoto em um ambiente de rede local

2.2. CONCEPÇÃO DO MODELO

A arquitetura de uma RL é geralmente composta das seguintes partes principais (do ponto de vista de avaliação de desempenho):

a) meio de transmissão;

b) protocolo de acesso ao meio;

c) estação;

d) controle de fluxo entre estações;

e) controle de fluxo para acesso à estação;

f) aplicação.

A Figura 2.2 ilustra um modelo (genérico), de uma forma sucinta, que captura detalhes das partes listadas acima. Observe que as figuras 2.1 e 2.2 são semelhantes.

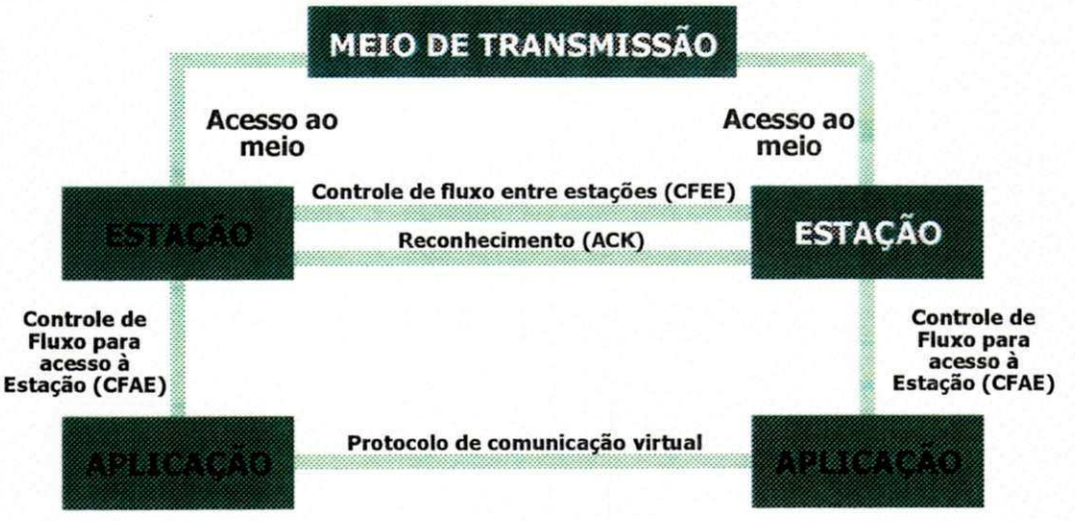

Figura 2.2 Modelo genérico para uma rede local

2.3. ENTIDADES E SUAS FUNÇÕES

Entidade é uma parte do modelo (da Figura 2.2) que tem uma função própria (específica) e bem definida. Pode ser física (por exemplo: meio de transmissão) ou lógica (por exemplo: aplicação). As entidades relacionam-se através de eventos (veja a definição de evento na seção 2.3). O modelo ilustrado na Figura 2.2 corresponde a uma entidade uma das partes listadas na seção 2.2. Vejamos agora a descrição das entidades do modelo.

a) **Meio de transmissão**

Tem a função de transmitir as informações de uma estação fonte a uma estação remota. Para simplificar, no nosso modelo, a entidade **meio de transmissão** é modelada por um intervalo de tempo gerado através de uma distribuição de probabilidade, acrescido de um overhead. Esse intervalo de tempo corresponde ao tempo de transmissão dos bits de um pacote de dados; o overhead corresponde à transmissão dos bits no cabeçalho e na calda do pacote. Esse overhead é produzido pelos protocolos da sub-rede

para endereçamento, identificação do tipo de pacote (dados, controle, ack, etc....) e controle de erro e de fluxo.

b) **Acesso ao meio**

Tem a função de controlar o acesso ao meio de transmissão, decidindo qual estação, em dado instante, tem permissão para transmitir. Podemos citar como exemplo o modelo do protocolo de acesso ao meio de transmissão em anel com passagem de ficha. Neste protocolo, uma ficha de controle (representada por uma seqüência especial de bits) é passada seqüencialmente ao redor do anel. Qualquer estação, ao receber a ficha, pode removê-la do anel, transmitir um pacote e então passar a ficha para a próxima estação.

c) **Estação**

Tem a função de transmitir e receber informação.

Para uma melhor distribuição modular do nosso modelo, desmembramos a **estação** em duas entidades, que são:

1) **processamento de transmissão de pacotes; e**

2) **processamento de recebimento de pacotes.**

O **processamento de transmissão de pacotes** é modelado como uma entidade que aceita pacotes (da **aplicação**) a transmitir e para cada pacote aceito emite um aviso de aceitação deste pacote para a **aplicação**.

O **processamento de recebimento de pacotes** é modelado como uma entidade que, para cada pacote de dados recebido, gera um pacote de reconhecimento (pacote ACK) a ser transmitido, se assim o usuário desejar. Encarrega-se, ainda, de passar o pacote recebido para a aplicação solicitada pelo usuário fonte; atualizar os recursos de controle de fluxo e descartar uma possível retransmissão do pacote de dados correspondente, para o caso em que o pacote recebido é ACK.

d) **controle de fluxo entre estações**

Tem a função de gerenciar os pacotes que serão transmitidos, mantendo uma fila destes pacotes em cada estação e atualizando os recursos de controle de fluxo utilizados. É modelado como uma entidade que aceita pedidos de transmissão e, dependendo das condições estabelecidas pelo controle de fluxo (ex.: tamanho da janela, créditos etc. [MOUR 86], libera uma autorização para (tentar) transmitir. Por exemplo, no controle de fluxo por janela, a estação autoriza transmissões enquanto a janela estiver aberta. Quando a janela fechar, a estação só dará uma nova autorização para transmitir, quando receber um pacote de reconhecimento para algum pacote de dados previamente enviado.

e) **controle de fluxo para acesso à estação**

Controla os pacotes oriundos da **aplicação** para acesso à estação, mantendo uma fila destes pacotes e atualizando os recursos utilizados. É modelado como uma entidade que aceita pacotes a transmitir em função das condições que a estação estabelece para gerir estes pacotes (por exemplo: tamanho da fila). Ao aceitar um pacote, a entidade sinaliza a **aplicação**, informando o aceite.

f) **Aplicação**

É um módulo do simulador que serve ao propósito de fazer a interface do pacote (simulador) com o usuário (ser humano) através de uma linguagem de entrada, onde este usuário pode escolher que aplicações ele quer simular. É definida ou escolhida pelo usuário e modelada como uma entidade que gera pacotes a transmitir, de acordo com uma distribuição de probabilidade. Para cada pacote de dados gerado, recebe a mensagem de aceitação deste pacote (veja a Figura 2.3).

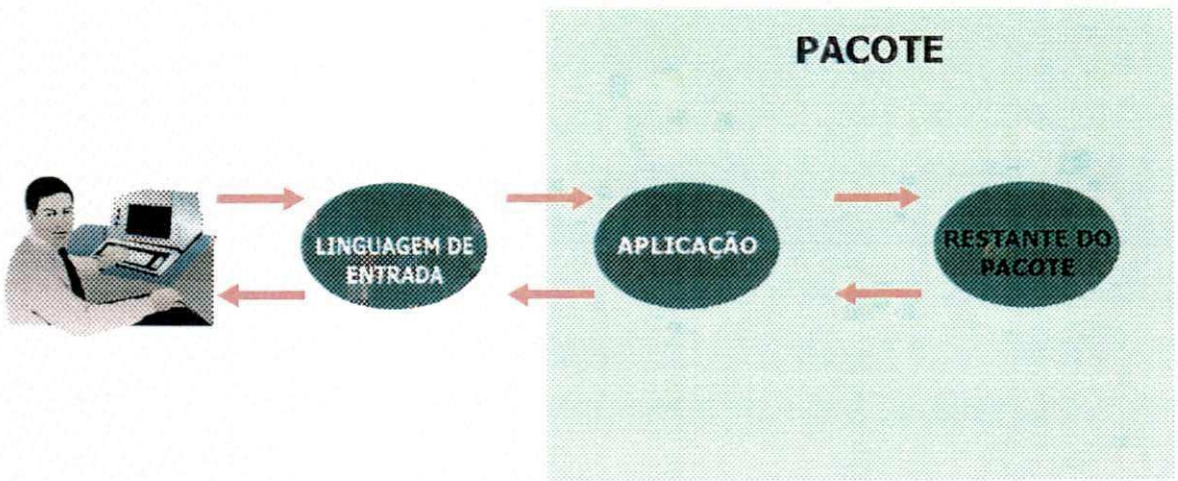

Figura 2.3 Aplicação

2.4. RELAÇÕES ENTRE AS ENTIDADES E SEUS EVENTOS

Um evento é uma "perturbação" instantânea que muda o estado do sistema (e por conseguinte, do modelo) [MOUR 86].

É necessário salientar que neste texto as palavras produz e escalona são sinônimas (isto quando estamos nos referindo a eventos).

Nas seções a seguir, usaremos as convenções da Figura 2.4.

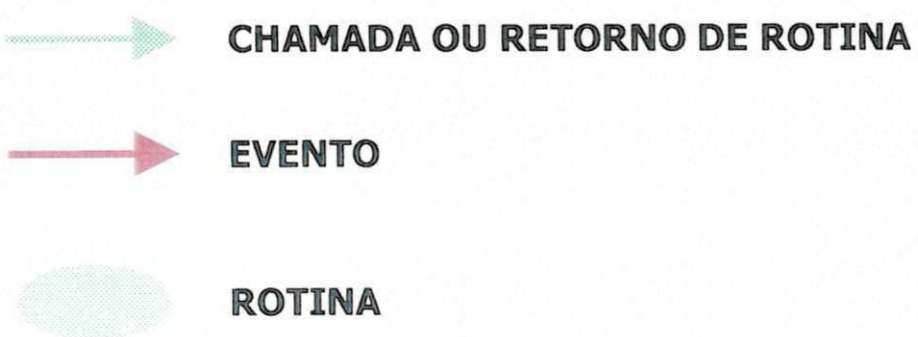

Figura 2.4 Convenções

A interpretação das convenções é oferecida pela Figura 2.5 onde a seta (vermelha) do evento GANHA_ACESSO indica que este evento é tratado pela entidade meio de transmissão, e esta por sua vez, chama a rotina **duplica_pacote** (veja a seta verde em sua direção). Quando **duplica_pacote** retorna o controle à entidade acima citada (veja a seta verde saindo da rotina **duplica_pacote** e indo a direção à entidade meio de transmissão), escalona o evento FIM_DE_TRANSMISSÃO (veja a seta vermelha do referido evento). Este significado se estende às setas de outros eventos e chamadas de rotinas.

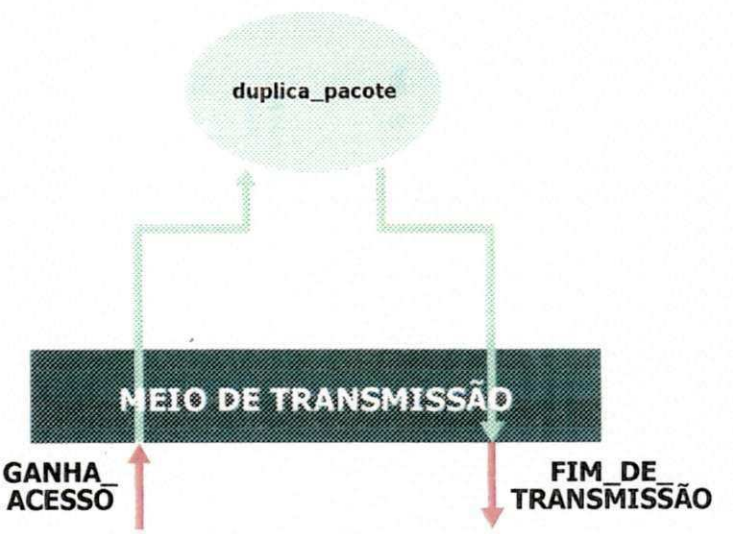

Figura 2.5 Ilustração do significado das convenções

Vejamos a seguir como as entidades do modelo genérico (listadas anteriormente) se relacionam através de eventos e chamadas a funções (rotinas). Por brevidade, nem todas as rotinas do pacote serão mencionadas aqui. Discutiremos apenas aquelas que forem chamadas por alguma entidade, ou acionadas por algum evento. No próximo capítulo mostraremos uma lista com todas as rotinas do simulador.

Neste modelo usaremos as seguintes entidades: **meio de transmissão**, **acesso ao meio**, **processamento de transmissão de pacotes**, **controle de fluxo entre estações**, **controle de fluxo para acesso à estação**, **processo de aplicação** e **processamento de recebimento de pacotes**.

a) Entidade **meio de transmissão**

Esta entidade é acionada pelo evento **GANHA_ACESSO** (escalonado pela entidade **acesso ao meio**), que por sua vez aciona a rotina **duplica_pacote**. Para que o pacote possa ser retransmitido no caso de erro de transmissão, esta rotina duplica o pacote a transmitir (ou melhor, a estrutura de dados que representa o pacote) e em seguida escalona o evento **REC_PAC**, associando a ele o pacote ora duplicado. Escalona ainda o evento **FIM_DE_TRANSMISSÃO**, conforme mostrar a Figura 2.6. Os eventos **REC_PAC** e **FIM_DE_TRANSMISSÃO** são escalonados para ocorrerem simultaneamente, só que o primeiro é escalonado para a estação remota e o segundo para a estação fonte. O evento **REC_PAC** aciona a entidade (remota) processamento de recebimento de pacotes. O evento **FIM_DE_TRANSMISSÃO** aciona a entidade (fonte) acesso ao meio.

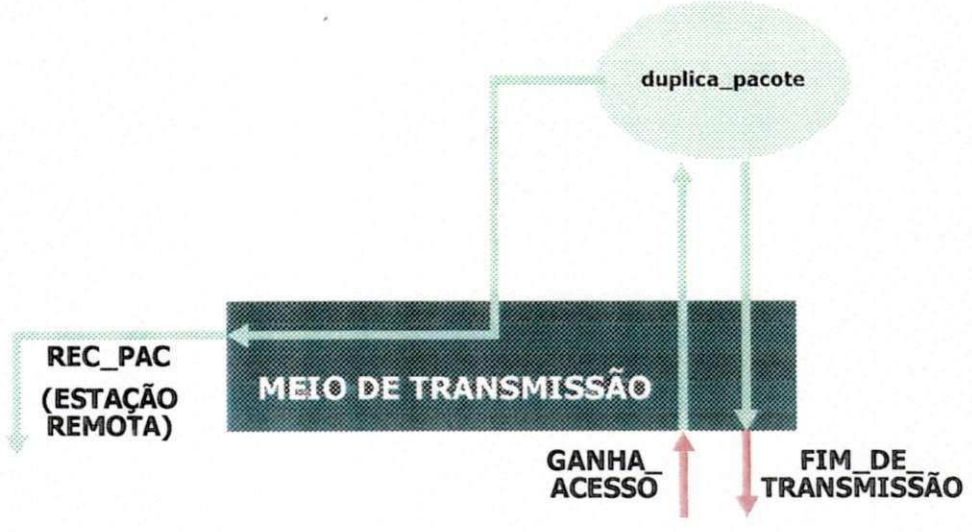

Figura 2.6 Modelo da entidade meio de transmissão

Fazendo uma analogia da entidade acima descrita com a aplicação de controle de estoque distribuída da seção 2.1, podemos dizer que esta entidade simula o tempo de transmissão de um pacote (pedaço do pedido de material) do almoxarifado setorial fonte para o almoxarifado setorial destino, através de um meio de transmissão (meio físico), por exemplo um cabo coaxial.

Quando um pacote da estação fonte tem acesso ao meio de transmissão (evento GANHA_ACESSO), ele leva um certo tempo para chegar à estação destino por isto existe a necessidade de se escalonar o evento REC_PAC que é um aviso da chegada do pacote transmitido na estação destino. O evento FIM_DE_TRANSMISSÃO é também um aviso de que o pacote chegou à estação destino, só que ele é consumido pela estação fonte, que assim fica sabendo que já pode tentar enviar um outro pacote.

A analogia feita acima é repetida para as entidades processo de aplicação e processamento de recebimento de pacotes. Só não repetimos em todas as entidades a seguir para não ficar tão repetitivo, mas creio que não será difícil, para o leitor, imaginá-las.

b) Entidade **acesso ao meio**

Em todo protocolo de acesso ao meio existem ações inerentes ao próprio protocolo (por exemplo: as ações requisitadas por um protocolo de passagem de ficha são diferentes das requisitadas por um protocolo com inserção de registro) e ações que são idênticas às de outros protocolos (por exemplo: enfileirar e desenfileirar pacotes na entidade acesso ao meio). Por esta razão, e para não quebrar a modularidade do nosso modelo, dividimos esta entidade em duas partes: a parte dependente do protocolo de acesso ao meio e a parte independente, respectivamente. Assim, caso o usuário necessite de um protocolo não implementado neste simulador basta mudar a parte dependente, diminuindo assim o seu trabalho (Veja a Figura 2.7).

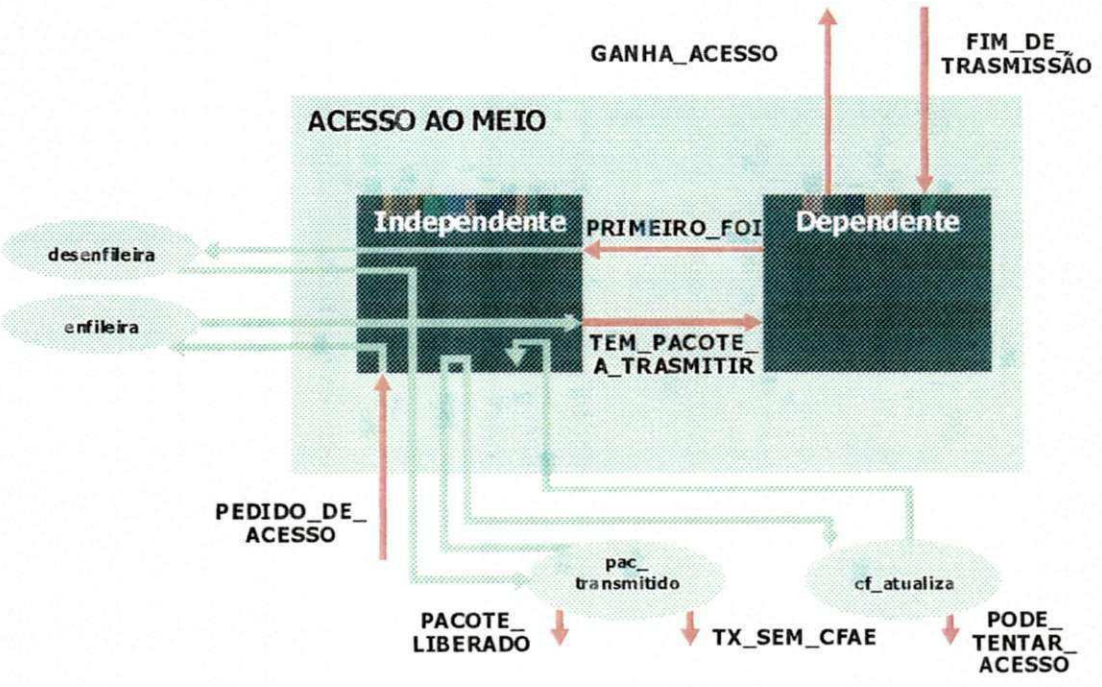

Figura 2.7 Modelo da entidade acesso ao meio

b.1) **Parte independente**

É acionada pelo evento **PEDIDO_DE_ACESSO**, que chama a rotina **enfileira**, responsável por colocar em uma fila o pacote a transmitir. Em seguida, se o acesso ao meio não estiver bloqueado, escalona o evento **TEM_PACOTE_A_TRANSMITIR**, bloqueando em seguida o acesso ao meio, para que cada estação tenha um, e somente um, pacote em transmissão (isto só é válido para os protocolos de acesso ao meio do IEEE 802, a que nos dispomos implementar).

É acionada também pelo evento **PRIMEIRO_FOI** (que é escalonado pela parte dependente) o qual dispara a rotina **desenfileira**, responsável por retirar o pacote, recém transmitido, da fila. Caso a fila não fique vazia, escalona um novo evento **TEM_PACOTE_A_TRANSMITIR** para o pacote que se encontra na frente da fila da entidade acesso ao meio. Em seguida, a rotina **pac_transmitido** é acionada e poderá escalonar o evento

TX_SEM_CFAE, no caso do pacote necessitar ser retransmitido, ou o evento **PACOTE_LIBERADO**, no caso em que o pacote foi transmitido normalmente.

Retornando da rotina **pac_transmitido**, a parte independente chama a rotina **cf_atualiza**, que atualiza os recursos de controle de fluxo utilizados pelo pacote ora transmitido. Em seguida, **cf_atualiza** escalona o evento **PODE_TENTAR_ACESSO**, dependendo das condições de controle de fluxo.

b.2) **Parte dependente**

É acionada pelo evento **TEM_PACOTE_A_TRANSMITIR** que por sua vez escalona o evento **GANHA_ACESSO** em função das condições estabelecidas pelo protocolo de acesso ao meio. É acionada também pelo evento **FIM_DE_TRANSMISSÃO** escalonando em seguida, o evento **PRIMEIRO_FOI**.

Vale salientar que a rotina **cf_atualiza** faz parte também da entidade **controle de fluxo entre estações (CFEE)**.

c) Entidade **processamento de transmissão de pacotes**

Esta entidade é acionada pelos eventos **PODE_TENTAR_ACESSO** e **TX_SEM_CFAE**, e pela entidade **controle de fluxo para acesso à estação** (através de chamada à rotina **enfileira**). Aqui temos um exemplo de que as entidades deste modelo não são acionadas exclusivamente por eventos, mas também por rotinas que neste caso representam eventos secundários.

Sendo acionada pelo evento **PODE_TENTAR_ACESSO**, a entidade chama a rotina **cf_ok** (da entidade **controle de fluxo entre estações**) que examina a estrutura de dados que contém pacotes (fila). Se a fila não estiver vazia (isto é, **há** pacotes a transmitir) a rotina **cf_ok** chamará a rotina **desenfileira** para retirar o pacote da fila e em seguida escalonará o evento **PEDIDO_DE_ACESSO**, associando a ele o pacote desenfileirado. No caso de fila vazia, retorna o controle para a entidade acima citada (veja a Figura 2.8).

Sendo ativada pelo evento **TX_SEM_CFAE** (escalonado pela rotina **pac_transmitido** ou pela rotina **rec_pac**), chama a rotina **enfileira** para enfileirar o pacote a ser transmitido e em seguida, chama a rotina **cf_ok**, a qual é avisada da chegada deste pacote. Se houver recursos para transmissão (por exemplo: crédito ou janela disponível), chama a rotina **desenfileira** para retirar o pacote da fila e logo após, escalona o evento **PEDIDO_DE_ACESSO**, associando a ele o pacote desenfileirado (veja a Figura 2.8).

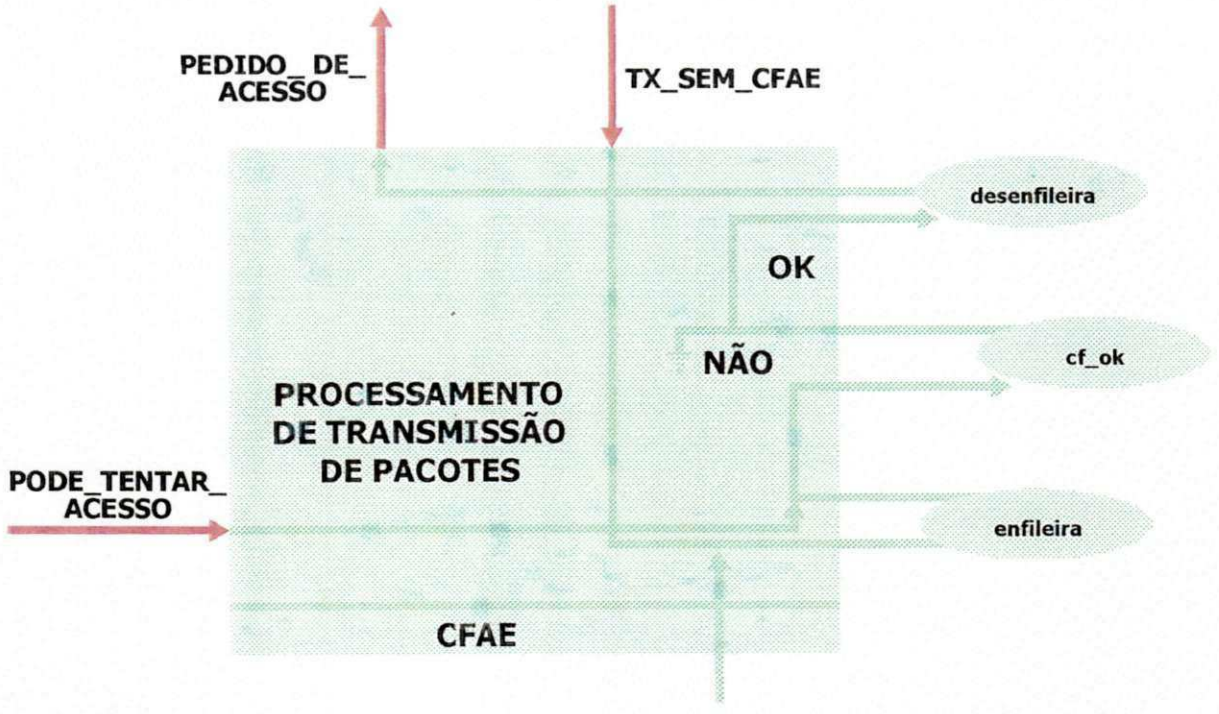

Figura 2.8 Modelo da entidade processamento de transmissão de pacotes

É importante salientar que esta entidade é relacionada com a entidade **controle de fluxo para acesso à estação (CFAE)**. Por isto, a Figura 2.8 mostra o **CFAE** como uma continuação da entidade **processamento de transmissão de pacotes.**

d) Entidade **controle de fluxo entre estações**

Esta entidade não é acionada por nenhum evento (veja a Figura 2.9). São chamadas a rotinas que a ativam: a chamada da rotina **cf_ok** proveniente da rotina **enfileira** e outra chamada da mesma rotina proveniente da entidade **processamento de transmissão de pacotes**. As ações que esta rotina (**cf_ok**) toma são descritas no item **c** desta seção.

É ativada também pela chamada da rotina **cf_atualiza,** que pode ser invocada pela entidade **acesso ao meio** e pela entidade **processamento de recebimento de pacotes.** As ações tomadas pela rotina **cf_atualiza**, estão descritas no item **b** desta seção.

Devemos observar que algumas aplicações não usam este controle de fluxo, o que torna esta entidade opcional, mas neste simulador ele é usado e não implementamos uma forma de desliga-lo.

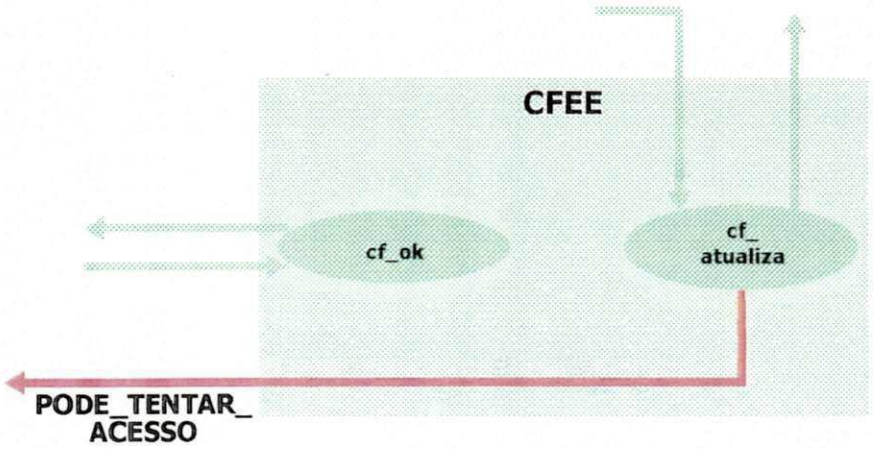

Figura 2.9 Modelo da entidade controle de fluxo entre estações

e) Entidade **Controle de fluxo para acesso à estação**

Esta entidade é acionada por dois eventos: **PACOTE_A_TRANSMITIR** e **PACOTE_LIBERADO**.

Quando é acionado pelo evento **PACOTE_A_TRANSMITIR** (veja a Figura 2.10), chama a rotina **cfae_b** que se encarrega de chamar a rotina

enfileira, que enfileira o pacote, na disponibilidade de vaga na fila. Neste caso, escalona o evento **PACOTE_ACEITO**. Caso não haja condição de enfileirar, o pacote ficará bloqueado, simulando o controle de fluxo entre estações (CFEE).

Quando esta entidade é ativada pelo evento **PACOTE_LIBERADO**, chama a rotina **cfae_db** que se encarrega de enfileirar o pacote que foi bloqueado anteriormente e, em seguida, escalona o evento **PACOTE_ACEITO**. Caso não haja um pacote bloqueado não faz nada (veja a figura 2.10).

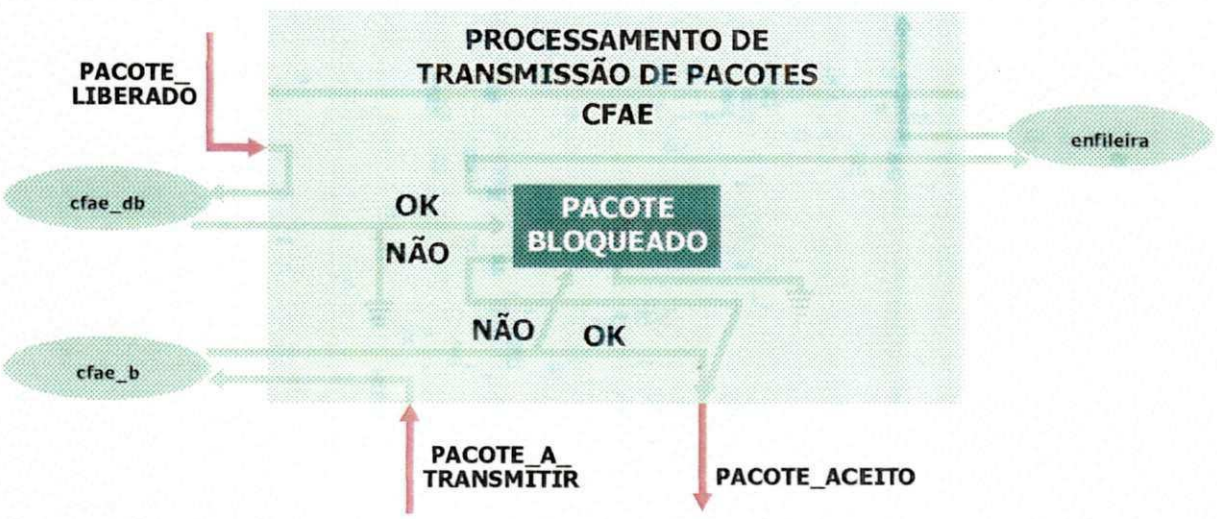

Figura 2.10 Modelo da entidade controle de fluxo para acesso à estação

O aparecimento da entidade **processamento de transmissão de pacotes** na Figura 2.10 é justificada no item **c** desta seção.

f) Entidade **processamento de aplicação**

É acionada pelo evento **PACOTE_ACEITO**, e na fase de inicialização (como já dissemos), pela aplicação escolhida e/ou definida pelo usuário (veja a Figura 2.11).

Para cada evento **PACOTE_ACEITO** tratado (por esta entidade), um evento **PACOTE_A_TRANSMITIR** é escalonado através de um gerador de

tráfego configurado pelo usuário, no caso em que a opção de bloqueio da aplicação pelo controle de fluxo para acesso à estação (**CFAE**) estiver ativa. Neste simulador esta configuração consiste em fornecer as médias da distribuição exponencial. Seria interessante que o usuário podesse escolher outras distribuições, mas este simulador não oferece esta facilidade. Neste caso, o gerador de tráfego fica bloqueado toda vez que gerar um pacote, sendo ativado exatamente quando ocorrer o aceite do pacote gerado anteriormente (evento **PACOTE_ACEITO**). Se a aplicação não for bloqueada pelo **CFAE**, os pacotes gerados terão que ser armazenados em uma fila (veja a figura 2.11). Esta fila modela a bufferização que o pacote é submetido na interface.

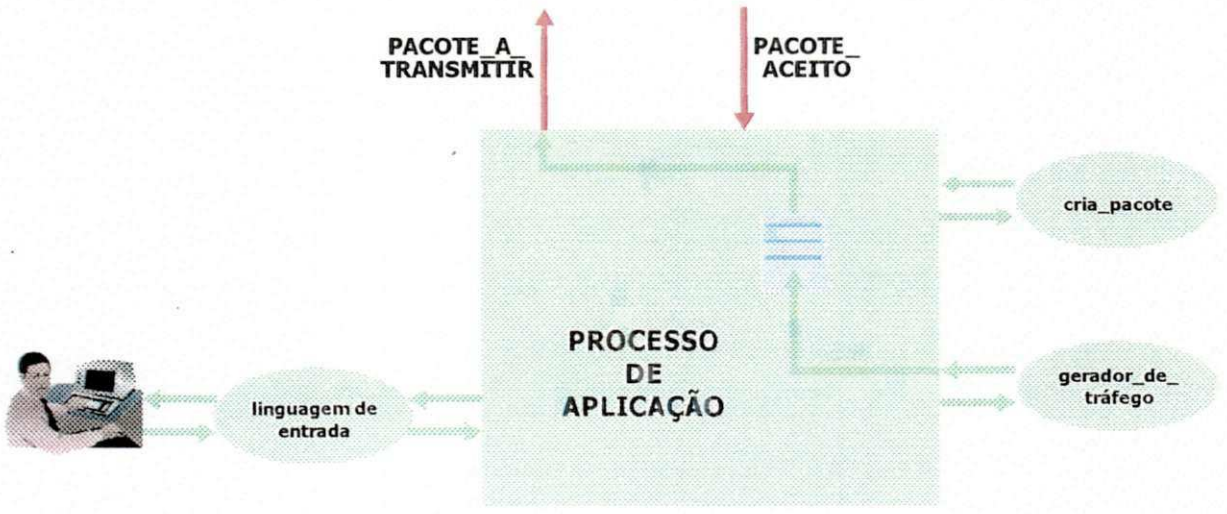

Figura 2.11 Modelo da entidade processo de aplicação

Fazendo uma analogia do exemplo de aplicação descrito na seção 2.1 com esta entidade, podemos dizer que esta entidade simula a aplicação (programa de controle de estoque distribuído) do almoxarifado fonte (estação fonte). Esta aplicação, que descrevemos acima, tem, geralmente, uma interface amigável com o usuário da aplicação, que pode ser o almoxarife (veja a Figura 2.11).

Depois de definido o pedido de material (podemos chamá-lo de mensagem), ele é quebrado em pacotes e é submetido à estação fonte

(computador fonte) através do evento PACOTE_A_TRANSMITIR (em termos de modelo de simulação). Se o pacote for recebido pela estação (evento PACOTE_ACEITO), a aplicação submete um outro pacote, até que toda a mensagem seja transmitida. Vale salientar que o gerador de tráfego é quem simula a criação (geração) da mensagem a ser transmitida.

g) Entidade **processamento de recebimento de pacotes**

É acionada pelo evento **REC_PAC** (recepção de pacote).

Quando o pacote recebido é de dados e o usuário pediu inclusão de tráfego de reconhecimento (ACK), um pacote ACK é gerado e associado a um evento **TX_SEM_CFAE**, que é escalonado de pronto. Em seguida, o pacote recebido é liberado. No caso em que o tráfego de ACKs é descartado, esta entidade só fará a coleta de estatísticas e a liberação do pacote recebido (veja a Figura 2.12).

Quando o pacote recebido for ACK, a entidade chama a rotina **cf_atualiza** que atualizará os recursos utilizados para controle de fluxo (por exemplo: abrir a janela). No caso de controle de fluxo por janela, escalona o evento **PODE_TENTAR_ACESSO**, em seguida, libera o pacote recebido.

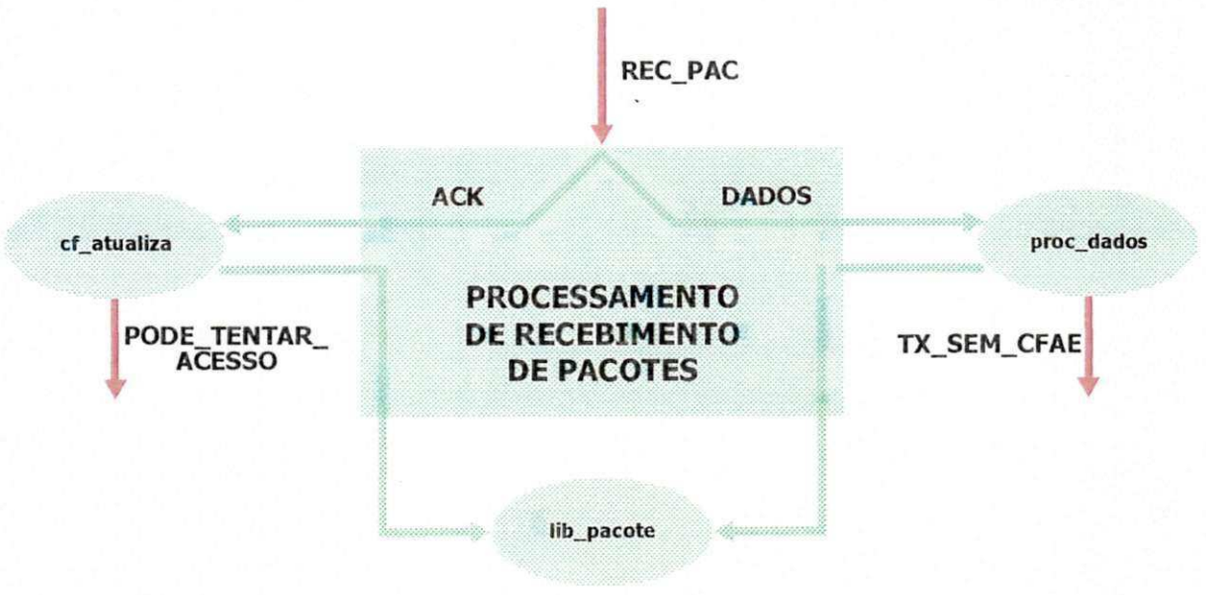

Figura 2.12 Modelo da entidade processamento de recebimento de pacotes

Traçando um paralelo da entidade acima (processamento de recebimento de pacotes) com o exemplo de aplicação de controle de estoque distribuído que apresentamos no início deste capítulo, podemos afirmar que esta entidade simula o almoxarifado setorial destino (estação destino) que vai atender ao pedido do almoxarifado setorial solicitantante (estação fonte). Quando o primeiro pacote de dados (da mensagem de solicitação de material) chega a esta entidade através do evento REC_PAC, a comunicação entre os dois almoxarifados setoriais é estabelecida e a estação destino então avisa à estação fonte que este pacote chegou através do envio de um pacote ACK (isto se o usuário do simulador assim desejar) e se repete até que todo o pedido de material seja transmitido do almoxarifado setorial fonte para o almoxarifado setorial destino.

Os Modelos das entidades acima, foram apresentados em um trabalho publicado no IV Simpósio Brasileiro de Redes de Computadores (veja [CAUP 86]) com poucas modificações.

2.5. VISÃO GERAL

Para melhorar o entendimento das relações entre as **entidades** e seus **eventos** de que trata a seção 2.4 deste capítulo, haja visto que o modelo genérico está desmembrado (particionado) relativamente ao contexto apresentado na Figura 2.2, resolvemos agrupar todas as entidades em uma só figura (veja a Figura 2.13).

Já traçamos alguns paralelos de partes do modelo aqui apresentado com a aplicação descrita na seção 2.1, mas agora faremos de maneira global com a figura 2.13. Quando um usuário de um almoxarifado setorial deseja um item de outro almoxarifado setorial, o seu pedido (que pode ser visto como uma mensagem) é quebrado em pacotes (na entidade processo de aplicação) e cada pacote é associado a um evento **PACOTE_A_TRANSMITIR**. Dependendo das condições de controle de fluxo de acesso a estação, o evento **PACOTE_ACEITO** é escalonado. Ocorrendo o evento **PACOTE_ACEITO**, um novo evento **PACOTE_A_TRANSMITIR** é escalonado, realimentando, assim, a entidade processamento de transmissão de pacotes.

Uma vez que o pacote se encontre na estação, um evento **PEDIDO_DE_ACESSO** é escalonado para ele, dependendo do controle de fluxo entre estações. Ocorrendo o evento **PEDIDO_DE_ACESSO**, o pacote é enfileirado na entidade acesso ao meio e se não houver um outro pacote desta estação sendo transmitido, um evento **TEM_PACOTE_A_TRANSMITIR** é escalonado para o pacote que esta na frente da fila da entidade acima citada. Quando ocorrer o evento **TEM_PACOTE_A_TRANSMITIR**, havendo possibilidade de transmissão, um evento **GANHA_ACESSO** é escalonado para este pacote.

Quando o evento **GANHA_ACESSO** ocorre, escalona o evento **FIM_DE_TRANSMISSÃO** para que ele possa descartar o pacote que foi transmitido da entidade (fonte) acesso ao meio. Na ocorrência do evento **GANHA_ACESSO**, é escalonado também o evento **REC_PAC** para a estação remota. A estação remota por sua vez, ao receber este evento, envia um pacote de reconhecimento (pacote ACK) para a estação fonte, para que esta atualize os seus recursos de controle de fluxo entre estações (CFEE).

Esperamos com estes comentários melhorar o entendimento da Figura 2.13, a seguir.

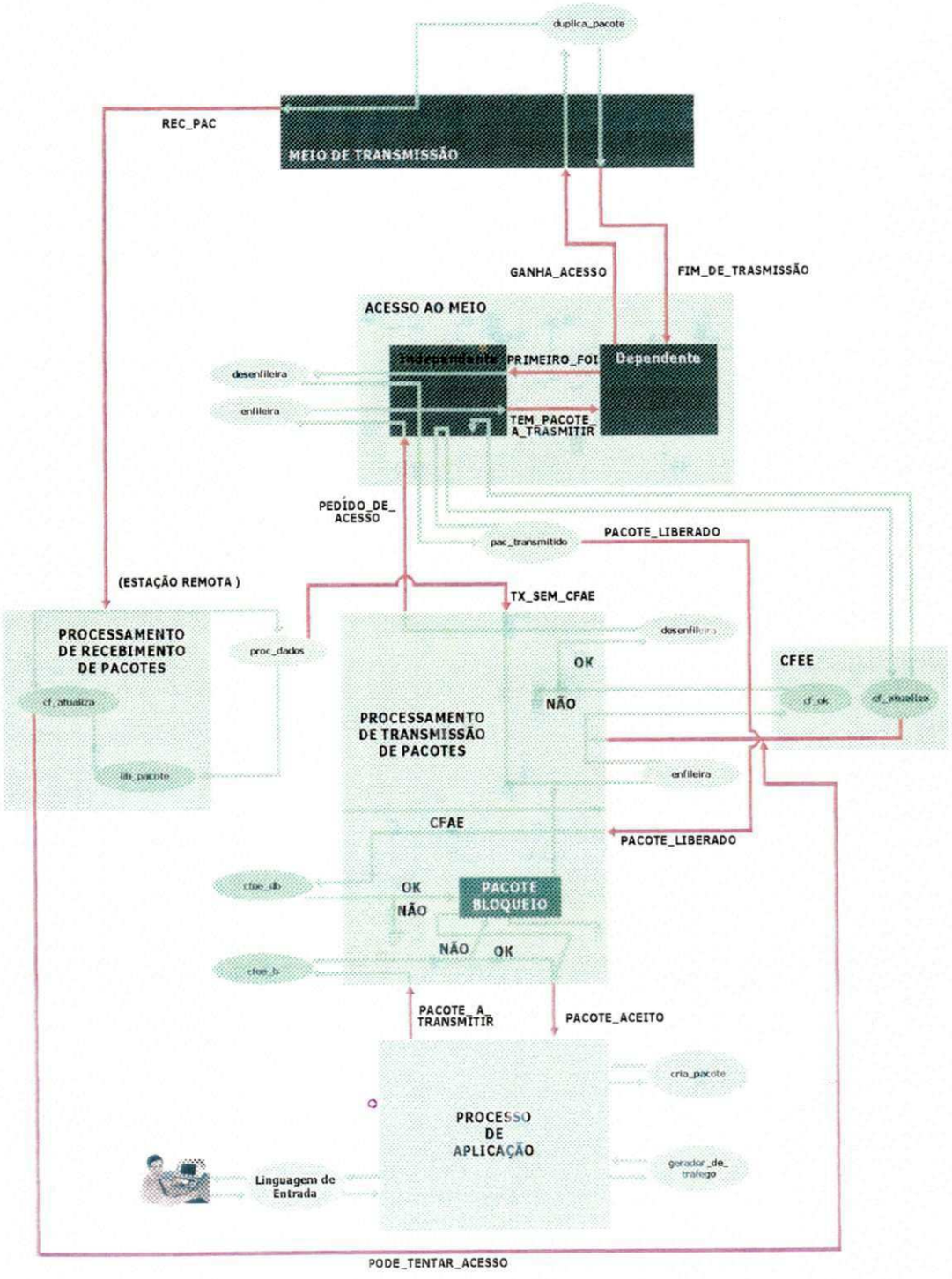

Figura 2.13: Modelo genérico de simulação para uma rede local

E para encerrar este capítulo, a Tabela 2.1, abaixo, resume o relacionamento das entidade com as rotinas que as simulam e seus eventos. Vejamos um exemplo de como se lê as Informações desta tabela: o evento **PACOTE_ACEITO** da entidade processo de aplicação é consumido por esta entidade, ou seja, pela rotina aplicação, que a simula.

Observe que a rotina **ac_m_d_an_fi** é quem simula a entidade acesso ao meio dependente para o protocolo de acesso em anel com passagem de ficha. Se fosse um outro protocolo esta rotina teria que ser substituída. Veja como fazer esta substituição no próximo capítulo.

TABELA 2.1: Relacionamento das entidades com as rotinas que as simulam e seus eventos

Entidade(s)	Rotinas(s) que a(s) simula(m)	Eventos	
		Nome	Ação
processo de aplicação	aplicacao	APLICACAO_TRANS_ARQUIVO	consome
		PACOTE_ACEITO	consome
		PACOTE_A_TRANSMITIR	produz
meio de transmissão	meio_de_trans	GANHA_ACESSO	consome
		FIM_DE_TRANSMISSAO	produz
		REC_PAC	produz
acesso ao meio (independente)	ac_m_ind	PEDIDO_DE_ACESSO	consome
		PRIMEIRO_FOI	consome
		TEM_PACOTE_A_TRANSMITIR	produz
		PACOTE_LIBERADO	produz
acesso ao meio (dependente)	ac_m_d_an_fi	PRIMEIRO_FOI	produz
		TEM_PACOTE_A_TRANSMITIR	consome
		GANHA_ACESSO	produz
		FIM_DE_TRANSMISSAO	consome
processamento de transmissão de pacotes	proc_trans	PODE_TENTAR_ACESSO	consome
		TX_SEM_CFAE	consome
		PACOTE_LIBERADO	consome
		PEDIDO_DE_ACESSO	produz
controle de fluxo entre estações (CFEE)	cf_ok e cf_atualiza	PODE_TENTAR_ACESSO	produz
controle de fluxo para acesso a estação (CFAE)	proc_trans (cfae_db e cfae_b)	PACOTE_A_TRANSMITIR	consome
		PACOTE_ACEITO	produz
processamento de recebimento de pacotes	proc_recepçao	REC_PAC	consome
		PODE_TENTAR_ACESSO	produz

CAPITULO 3

IMPLEMENTAÇÃO

3.1. CONSIDERAÇÕES GERAIS

A implementação do simulador baseia-se no modelo apresentado no capítulo 2 (veja a Figura 2.13).

Este simulador foi escrito na linguagem de programação C, inicialmente no sistema operacional UNIX do PDP-11/34 e posteriormente adaptado ao MS-DOS do micro computador compatível com o IBM-PC. Tem um total de 1485 linhas de código fonte, incluindo os comentários.

Vale salientar que o tamanho do simulador em bytes, é variável. Isto se dá em função da criação das estruturas de dados dinamicamente relocáveis em fase de execução. O tamanho estático do simulador (não em execução) é da ordem de 35 k bytes.

Uma observação importante é que nem tudo que foi planejado para o simulador foi implementado. A linguagem de entrada de dados (veja a letra f na página 12), por exemplo, ficou no projeto lógico da análise léxica e portanto nem apresentamos neste trabalho.

Este simulador é acionado por eventos (veja este tipo de simulação em [MOUR 86]). A Tabela 3.1. mostra uma lista dos eventos do simulador, informando quem consome (rotina que toma as providências relacionadas ao evento) e quem escalona cada evento.

TABELA 3.1
Lista dos eventos do simulador com seus consumidores e produtores

Evento	Rotina que consome	Rotina que escalona
APLICACAO_TRANS_ARQUIVO	aplicacao	aplicacao
PACOTE_A_TRANSMITIR	proc_trans	aplicaçao
PACOTE_ACEITO	aplicacao	proc_trans
PACOTE_LIBERADO	proc_trans	pac_transmitido
PEDIDO_DE_ACESSO	ac_m_ind	proc_trans
TEM_PACOTE_A_TRANSMITIR	ac_m_d_na_fi	ac_m_ind
PRIMEIRO_FOI	ac_m_ind	ac_m_d_na_fi
GANHA_ACESSO	meio_de_trans	pass_ficha
FIM_DE_TRANSMISSÃO	ac_m_d_na_fi	meio_de_trans
REC_PAC	proc_rerepcao	meio_de_trans
TX_SEM_CFAE	proc_trans	proc_recepcao
PODE_TENTAR_ACESSO	proc_trans	cf_atualiza

Obs.:

1) A rotina **ac_m_d_an_fi** (rotina que consome o evento **TEM_PACOTE_A_TRANSMITIR**), simula o protocolo de acesso ao meio em anel com passagem de ficha.

2) A terceira coluna da Tabela 3.1 (rotina que escalona), trata, na realidade, da rotina que chama a rotina que escalona o evento, por exemplo: a rotina **ac_m_d_an_fi** para escalonar o evento **PRIMEIRO_FOI**, chama a rotina **esc_prim_foi**. Na próxima seção este assunto ficará mais claro.

3) O único evento primário deste simulador é o **APLICACAO_TRANS_ARQUIVO**, os outros são todos secundários. Obs.: "Um evento primário é escalonado para ocorrer no simulador antes de sua ocorrência real. Um evento secundário ocorre em resposta a ocorrência de um evento primário" [MOUR 86].

4) Um outro evento primário que poderíamos ter neste simulador, é o de fim de simulação. Este evento seria escalonado para ocorrer num determinado instante da simulação e quando este instante chegasse a simulação seria encerrada. Isto se optassemos por executar o simulador por um determinado intervalo de tempo, mas neste caso optamos por parar a execução deste simulador quando

tivermos atingido um determinado tamanho de amostra que é fornecido pelo usuário do simulador no início da execução.

3.2. AS ROTINAS DO SIMULADOR

O simulador é composto de outras rotinas além das já discutidas até aqui e que estão contidas nas Tabelas 2.1. e 3.1.

A rotina principal (**main**) é composta da chamada de quatro rotinas, que sãos as quatro fases principais do simulador a seguir:

a) **leitura_ dados** – que, como o próprio nome diz, faz a leitura dos dados do simulador;

b) **inicialização** – que, também, como o próprio nome diz, faz a inicialização dos parâmetros e variáveis auxiliares do simulador, escalonando também o primeiro evento;

c) **scheduler** – é quem controla o simulador, atendendo os eventos da lista e chamando a rotina que trata cada evento, desencadeando assim, a chamada de todas as demais rotinas do simulador; e

d) **resultados** – imprime os resultados da simulação.

Temos ainda as rotinas de criação das estruturas de dados descritas na seção 3.3, que têm muita semelhança entre si, mudando somente o nome da estrutura de dados que está criando. Um exemplo é a rotina **cria_pacote** que cria a estrutura de dados **pacote**. Com características semelhantes temos as rotinas de escalonamento dos eventos, que atribuem as informações dos eventos aos membros das estruturas. Um exemplo é a rotina **esc_prim_foi** que escalona o evento **PRIMEIRO_FOI**.

Pensamos que uma listagem completa de todas as rotinas e o que cada uma delas faz seria cansativo e poucos benefícios traria. Assim, mostraremos de forma sucinta na Tabela 3.2, uma lista das outras rotinas do Simulador, e o que cada uma delas faz.

TABELA 3.2
Rotinas do simulador e sua funcionalidade

Rotina	Funcionalidade
cfae_db	Atualiza os recursos do CFAE, podendo desbloquear um pacote
cfae_b	Verifica os recursos do CFAE, e se a estação não pode consumir o pacote, ele fica bloqueado
aplicacao	Simula a entidade processo de aplicação, consumindo os eventos APLICACAO_TRANS_ARQUIVO e PACOTE_ACEITO
ger_pac_dados	Gera um pacote de dados
f_ascendente	Verifica se um pacote a ser enfileirado está em ordem ascendente em relação ao instante de tempo de geração
f_fim	Verifica se um pacote a ser enfileirado chegou ao fim da fila
enfileira	Enfileira pacotes
desenfileira	Desenfileira pacotes
col_estatistica	Coleta estatísticas
cf_ok	Verifica os recursos do CFEE, e se poder transmitir, atualiza-os
cf_atualiza	Atualiza os recursos do CFEE
debug	Imprime as mensagens de depuração
duplica_pacote	Duplica o pacote a ser transmitido
remove	Remove um nodo da lista duplamente encadeada
l_inicio	Verifica se um nodo a ser inserido está no início da listra
l_ascendente	Verifica se um nodo a ser inserido, na lista, está em ordem ascendente em relação ao instante de tempo de geração
insere	Insere um nodo em uma lista duplamente encadeada
pac_transmitido	Libera o pacote transmitido e se a opção CFEE estiver ligada chama cf_atualiza
random	Gera um número pseudo-aleatório para uma determinada seqüência de uma estação
gera_ack	Gera um pacote ack
proc_trans	Simula a entidade processamento de transmissão de pacotes, consumindo os eventos PODE_TENTAR_ACESSO, TX_SEM_CFAE, PACOTE_LIBERADO e PACOTE_A_TRANSMITIR
proc_recepcao	Simula a entidade processamento de recebimento de pacotes, consumido o evento REC_PAC
ac_m_ind	Simula a entidade acesso ao meio independente, consumindo os eventos PEDIDO_DE_ACESSO e PRIMEIRO_FOI
ac_m_d_an_fi	Simula a entidade acesso ao meio dependente com protocolo de acesso ao meio em anel com passagem de ficha
pass_ficha	Passa a ficha para a próxima estação que tem pacote a transmitir
mem_alloc	Aloca memória para uma estrutura de dados dinamicamente relocável
meio_de_trans	Simula a entidade meio de transmissão, consumindo o evento GANHA_ACESSO

Obs.: Nodo é um elemento de uma lista.

Informações detalhadas sobre cada uma das rotinas do simulador são oferecidas na listagem do programa no Apêndice B.

3.3. ESTRUTURAS DE DADOS

O simulador funciona em torno de uma lista de eventos duplamente encadeada. Esta lista é uma espécie de viga mestre para o simulador, o qual começa a sua execução colocando o primeiro evento nesta lista e a partir daí os outros eventos vão sendo gerados (escalonados) pelas rotinas que vão sendo acionadas pelo primeiro evento e inseridos na lista em ordem ascendente de instante de tempo de geração de cada evento. À medida que um evento é atendido (consumido) ele é removido da lista.

3.3.1. Estruturas de dados dos eventos

As estruturas de dados modelando os eventos contêm campos para:

a) apontador para o evento que o sucede;

b) apontador para o evento que o antecede:

c) apontador para a rotina que o consome;

d) parâmetros do evento em questão, que são: estação a quem o evento pertence e/ou um apontador para o pacote a ele associado:

e) tipo de evento; e

f) tempo de ocorrência do evento.

O item **d** não é comum a todos os eventos, pois nem todo evento necessita ter o apontador para um pacote. Por exemplo: o evento **PODE_TENTAR_ACESSO** não precisa de apontador para um pacote porque ele sempre ocorre para o primeiro pacote da fila da entidade processamento de transmissão de pacotes, necessitando somente da informação estação. Por esta razão e para minimizar o uso de memória, e, também, para otimizar a execução do simulador dividimos a estrutura de dados dos eventos em duas partes: a parte específica a cada evento que reúne as informações do item **d** acima e a parte comum a todos eventos, que reúne as demais informações dos eventos.

3.1.1.1 Parte Comum

Esta estrutura de dados é chamada **evento** (veja a Figura 3.1) e suas informações ficam armazenadas ns seguintes membros da estrutura (obedecida, é claro, a devida respectividade e substituindo a informação do item **d** por um apontador para os parâmetros do evento): **frente, tras, e_func, e_param, e_tipo,** e **e_t_ocorr**.

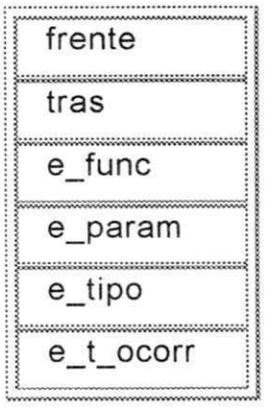

Figura 3.1. Estrutura evento

Os membros desta estrutura são usados para armazenar as informações relativas a cada evento. Por exemplo: o membro **e_tipo** pode conter a informação **PACOTE_ACEITO** se este for o evento que esta estrutura está representando.

3.3.1.2 Parte Específica

É anexada à parte comum através do apontador **e_param** (da estrutura evento) e contém os parâmetros do evento em questão.

Como todo evento tem sua parte específica distinta, uma hora, a parte comum está associada à estrutura de dados dos parâmetros (parte específica) do evento **REC_PAC**, e em outro instante à do evento **GANHA_ACESSO**, por exemplo.

Mostraremos nas sub-seções seguintes as estruturas de dados (parte específica) de cada evento, como elas foram chamadas e as figuras com os nomes dos membros de cada estrutura.

a)Estrutura de dados do evento **GANHA_ACESSO**

Nome: **g_acesso**

ga_estação
ap_g_acesso

Figura 3.2. Estrutura g_acesso

Onde **ga_estação** é um inteiro e **ap_g_acesso** é um apontador para uma estrutura de dados do tipo pacote.

b) Estrutura de dados do evento **FIM_DE_TRANSMISSÃO**

Nome: **fim_trans**

ft_estação
ap_fim_trans

Figura 3.3. Estrutura fim_trans

Onde **ft_estação** é um inteiro e **ap_fim_trans** é um apontador para uma estrutura de dados do tipo pacote.

c) Estrutura de dados do evento **RECEPÇÃO_DE_PACOTE**

Nome: **rec_pac**

rp_estação
ap_recpac

Figura 3.4. Estrutura rec_pac

Onde **rp_estação** é um inteiro e **ap_recpac** é um apontador para uma estrutura de dados do tipo pacote.

d) Estrutura de dados do evento **PRIMEIRO_FOI**

Nome: **prim_foi**

pf_estação
ap_p_foi

Figura 3.5. Estrutura prim_foi

Onde **pf_estação** é um inteiro e ap_p_foi é um apontador para uma estrutura de dados do tipo pacote.

e) Estrutura de dados do evento **TEM_PACOTE_A_TRANSMITIR**

Nome: **tem_pac_a_tx**

tpatx_estação
ap_tpatx

Figura 3.6. Estrutura tem_pac_a_tx

Onde **tpatx_estação** é um inteiro e **ap_tpatx** é um apontador para uma estrutura de dados do tipo pacote.

f) Estrutura de dados do evento **PEDIDO_DE_ACESSO**

Nome: **pedido_aces**

pa_estação
ap_pacess

Figura 3.7. Estrutura pedido_aces

Onde **pa_estação** é um inteiro e **ap_pacess** é um apontador para uma estrutura de dados do tipo pacote.

g) Estrutura de dados do evento **PODE_TENTAR_ACESSO**

Nome: **pode_t_acesso**

pta_estação

Figura 3.8. Estrutura pode_t_acesso

Onde **pta_estação** é um inteiro.

h) Estrutura de dados do evento **TRANSMISSÃO_SEM_CFAE**

Nome: **tx_sem_cfae**

tscfae_estação
ap_tscfae

Figura 3.9. Estrutura tx_sem_cfae

Onde **tscfae_estação** é um inteiro e **ap_tscfae** é um apontador para uma estrutura de dados do tipo pacote.

i) Estrutura de dados do evento **PACOTE_LIBERADO**

Nome: p**ac_liberado**

pl_estação

Figura 3.10. Estrutura pac_liberado

Onde **pl_estação** é um inteiro.

j) Estrutura de dados do evento **PACOTE_A_TRANSMITIR**

Nome: **pac_a_trans**

pat_estação
ap_patrans

Figura 3.11. Estrutura pac_a_trans

Onde **pat_estação** é um inteiro e **ap_patrans** é um apontador para uma estrutura de dados do tipo pacote.

k) Estrutura de dados do evento **PACOTE_ACEITO**

Nome: **pac_aceito**

pacto_estação
ap-paceito

Figura 3.12. Estrutura pac_aceito

Onde **pacto_estação** é um inteiro e **ap_paceito** é um apontador para uma estrutura de dados do tipo pacote.

3.3.2 Organização da lista de eventos

Como já foi mencionado esta lista é duplamente encadeada, e no começo da simulação o evento **APLICAÇÃO_TRANS_ARQUIVO** (se o simulador sendo executado for o que simula a aplicação transferência de arquivos, mencionada no início do capítulo 2 e estudada no próximo capítulo) é escalonado e colocado na lista. Este evento quando tratado pelo simulador, chama a rotina que o consome (aplicação) e esta rotina se encarrega de acionar o restante do simulador que desencadeia uma série de escalonamentos de outros eventos que também são colocados nesta lista em ordem crescente de ocorrência (obs.: o membro **e_t_ocorr**

da estrutura **evento**, da Figura 3.1., contém o tempo de ocorrência do evento, ou seja, instante em que o evento ocorrerá).

A Figura 3.13, abaixo dá uma idéia da organização da lista de eventos. O primeiro evento desta lista hipotética é o evento **PACOTE_ACEITO**. Compare a parte específica (apontada por **e_param**) com Figura 3.12. este é o evento a ser tratado pelo simulador neste instante. Os outros eventos serão atendidos em seguida. Note que o primeiro evento (**PACOTE_ACEITO**) e o último (**TRANSMISSÃO_SEM_CFAE**) têm um pacote associado a eles (veja a Figura 3.13). Já o segundo evento não possui pacote, por se tratar do evento **PODE_TENTAR_ACESSO** que obtém esta informação da fila de pacotes da entidade **processamento de transmissão de pacotes.**

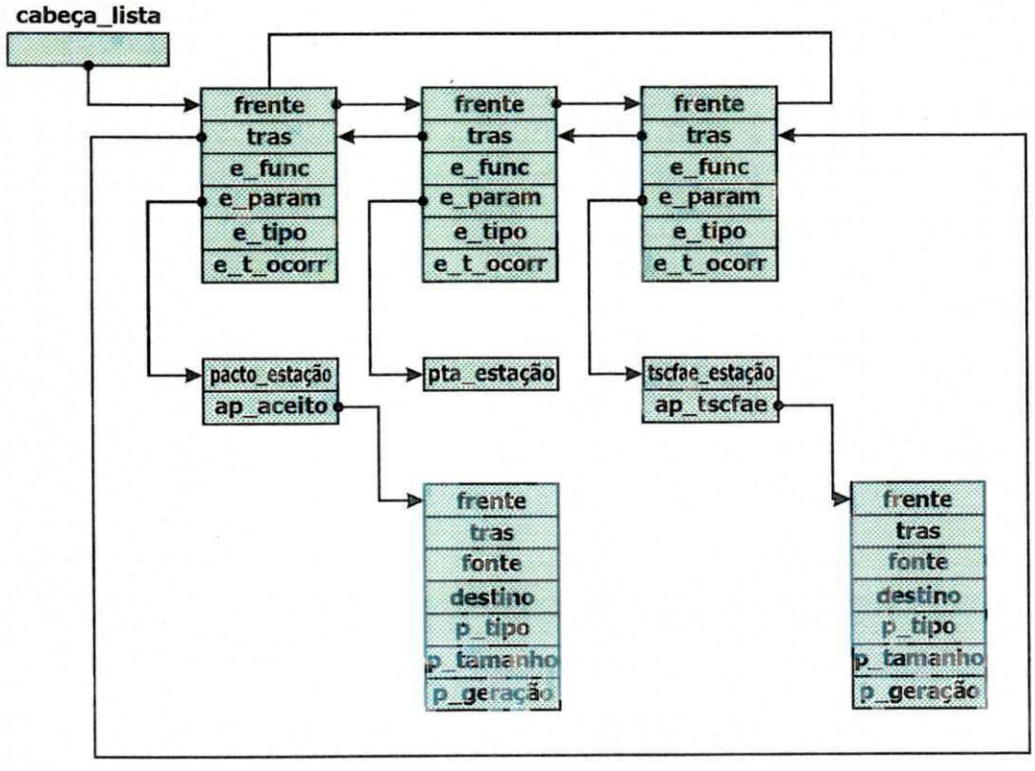

Figura 3.13 Exemplo hipotético da lista de eventos

3.3.3 Estrutura de Dados dos pacotes

Os eventos se relacionam com os pacotes através de um apontador (para a estrutura de dados do tipo pacote), que está na parte específica da estrutura de cada Evento (por exemplo: o apontador **ap_paceito** da estrutura do evento **PACOTE_ACEITO**, na Figura 3.13).

A estrutura de dados dos pacotes é chamada **pacote** e tem as seguintes informações (veja a Figura 3.14):

a) um apontador para o pacote que o sucede, cujo membro da estrutura é chamado **frente**;

b) um apontador para o pacote que o antecedem, chamado **trás**;

c) estação fonte, chamado **fonte**;

d) estação destino, chamado **destino**;

e) tipo do pacote, chamado **p_tipo**;

f) tamanho do pacote, chamado **p_tamanho**; e

g) tempo (instante) de geração do pacote, chamado **p_geração**;

Os membros desta estrutura são usados para armazenar as informações inerentes a cada pacote, por exemplo: o membro **p_tipo** de um determinado pacote pode conter a informação ACK se o pacote que representa for ack.

Figura 3.14 – Estrutura pacote

3.3.4 Organização das Filas de Pacotes

Cada estação tem três filas de pacotes:

a) Uma fila dos pacotes bloqueados pela entidade **controle de fluxo para acesso à estação**;

b) Uma fila da entidade **acesso ao meio**; e

c) Uma fila da entidade **processamento de transmissão de pacotes**.

Estas filas são apontadas pelo arranjo **frente_fila** que aponta sempre para o pacote que está na frente de cada fila. Por exemplo: o apontador **frente_fila[1][FILAACMEIO]**, aponta para o primeiro pacote da fila da entidade **acesso ao meio** da estação 1.

Estas são implementadas também, como listas duplamente encadeadas, usando assim, as mesmas rotinas de manipulação de listas, usadas pela lista de eventos.

A Figura 3.15 dá uma idéia de como funcionam as filas de uma estação **i**. A organização das demais filas é semelhante.

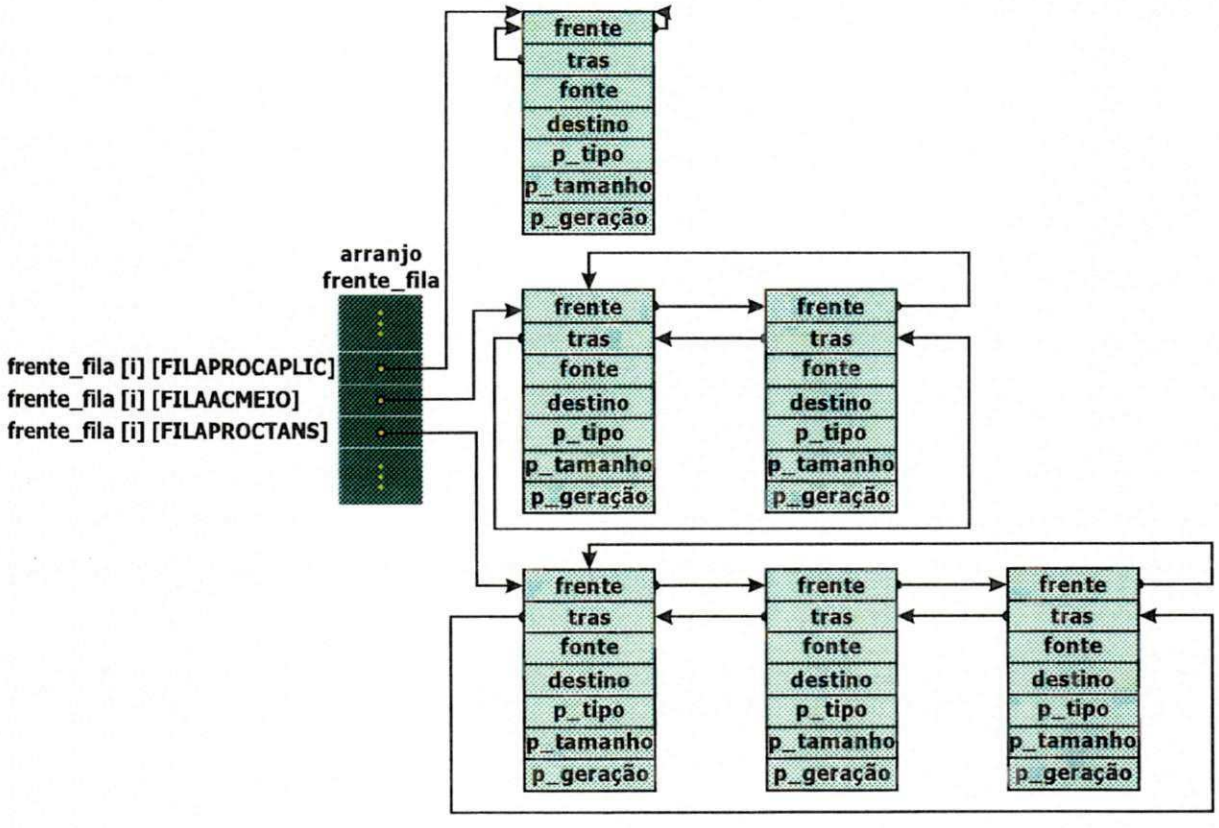

Figura 3.15 Exemplo hipotético das filas de uma estação i.

O esforço de trabalho empregado na implementação deste simulador foi dividido na Tabela 3.3 de acordo com o modelo apresentado no Capítulo 2, para que o leitor se desejar implementar algo semelhante tenha idéia de quanto tempo vai gastar. O relato do esforço constante na tabela 3.3 foi feito com base na experiência do programador na linguagem de programação utilizada que era de 3 anos.

TABELA 3.3

Força de trabalho empregada na implementação

Nome do segmento	Homens/hora
Meio de transmissão	58
Acesso ao meio – parte independente	90
Acesso ao meio – parte dependente: anel com passagem de ficha	65
Processamento de transmissão de pacotes	154
Controle de fluxo entre estações (CFEE) – por janela	63
Controle de fluxo para acesso à estação (CFAE)	73
Processamento de recebimento de pacotes com envio de ACK	99
Conexões de transporte, estatísticas e outros	316
Processo de aplicação – transferência de arquivos	70

CAPÍTULO 4

INTERFACE COM O USUÁRIO

Apesar de não termos implementados uma interface interativa com o usuário para este simulador, é muito simples usá-lo.

Neste capítulo supomos que o leitor conhece razoavelmente a linguagem de programação C.

4.1 PROCEDIMENTOS PARA UTILIZAÇÃO

Na sua forma atual, o uso deste simulador é feito a partir de um arquivo executável contendo o código objeto executável do Apêndice A ou B. Este arquivo executável é o resultado da compilação e linkedição do arquivo fonte que deve conter o programa do Apêndice A, caso você deseje o arquivo executável do primeiro estudo de caso (Veja o capítulo 5) ou do Apêndice B se para os demais estudos de casos.

O simulador usado aqui (veja Figura 4.1), simula a sub-rede em anel com passagem de ficha, onde os pacotes são gerados por um único gerador (função densidade de probabilidade exponencial) e cada pacote é alocado a uma determinada estação com probabilidade 1/n (n é o número de estações na sub-rede) e uniformemente. Veja mais detalhes sobre este simulador no primeiro estudo de caso do próximo capítulo.

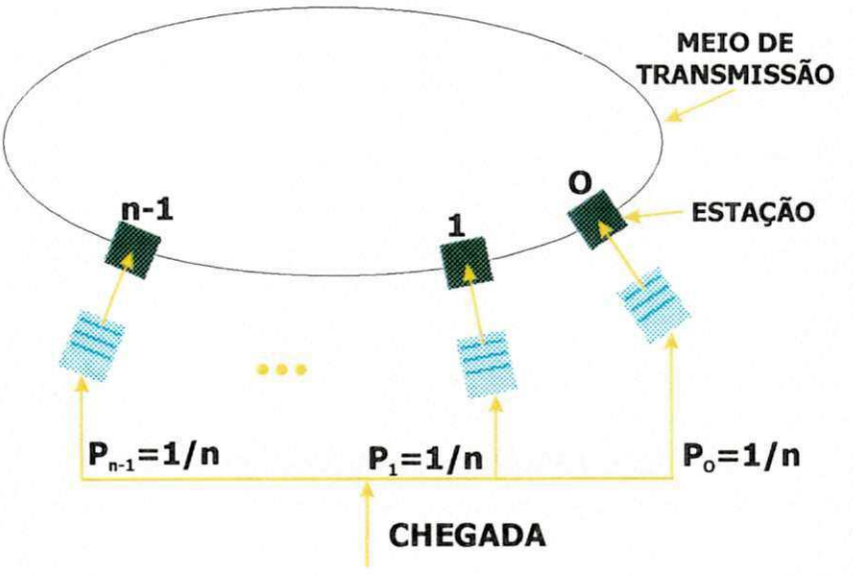

Figura 4.1 Rede do primeiro estudo de caso

Para gerar o arquivo executável do conteúdo do Apêndice A, por exemplo, usando um compilador C em um micro-computador compatível com o IBM PC e sistema operacional do tipo MS-DOS, siga os seguintes passos:

a) digite o conteúdo do Apêndice A num arquivo chamado, por exemplo, **SIMULADOR.C** (obs.: a extensão C significa que o conteúdo do arquivo **SIMULADOR.C** é código fonte na linguagem de programação C);

b) usando a seguinte linha de comando **cc simulador.c** você obterá o arquivo **SIMULADOR.EXE** que é o arquivo executável que estamos procurando.

Este arquivo executável (**SIMULADOR.EXE**) contém, portanto, o código executável do simulador do primeiro estudo de caso.

Para rodarmos este simulador devemos usar a seguinte linha de comando.

simulador a b c d e f

onde,

1) **a** é o tempo médio de Interchegada em uts;

2) **b** é o tempo de transmissão do anel em uts;

3) **c** é o número de estações;

4) **d** é o número de transmissões de pacotes simuladas;

5) **e** é a semente para a distribuição dos tempos de interchegada (exponencial); e

6) **f** é a semente para a distribuição de escolha da estação (uniforme).

Note que os sistemas operacionais do tipo MS-DOS não fazem distinção de comandos (linhas de comandos) em maiúsculas ou minúsculas, por isto não fizemos aqui esta distinção.

Fazendo a=10.000, b=1.000, c=16, d=15.000, e=13 e f=17, temos a seguinte linha de comando:

simulador 10000 1000 16 15000 13 17

O resultado (saída) desta linha de comando, acima, é o seguinte:

Resultado do Simulador: Estudo de Caso da Sub-Rede em Anel com Passagem de Ficha

ENTRADAS:

Tempo Medio de Interchagadas = 10000

Tempo de Transmissao no Anel = 1000

Numero de Interfaces = 16

Numero de Transmissoes Simuladas = 15000

Semente da Distribuiçao Exponencial = 13

Semente da Distribuiçao Uniforme = 17

Resultados:

Tempo Medio de Espera = 150.821667

Utilizacao do Meio de Transmissao = 0.095290

Compare o resultado desta simulação acima com os resultados apresentados no primeiro estudo de caso do próximo capítulo.

Obs.: 1) No simulador apresentado acima usamos a unidade de tempo simulado (uts) igual a 1 μs. Veja mais detalhes sobre uts no Capítulo 8 de [MOUR 86].

2) Antes de prosseguirmos com os próximos estudos de caso (2 e 3) é necessário definirmos TPDU (unidade de dados do protocolo de transporte) que é na realidade uma espécie de pacote da camada de transporte.

De modo similar você pode obter o arquivo executável do Apêndice B que simula o segundo estudo de caso e o terceiro do capítulo seguinte.

O simulador usado aqui (veja a Figura 4.2), simula conexões de transporte (veja o segundo estudo de caso do próximo capítulo) suportando a aplicação de transferências de arquivos em uma rede em anel com passagem de ficha (veja o terceiro estudo de caso do próximo capítulo).

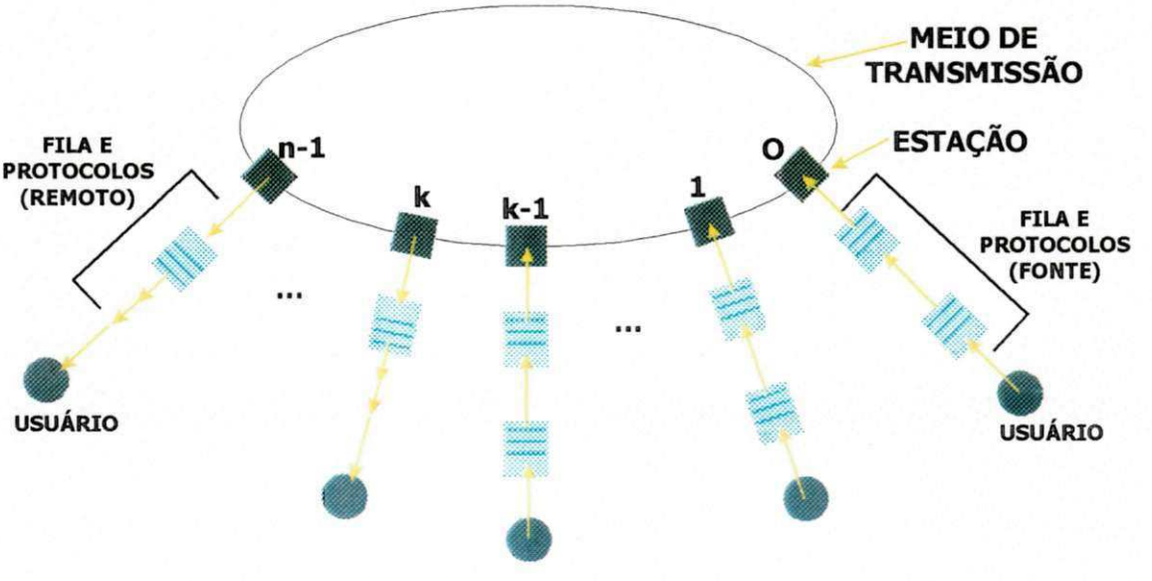

Figura 4.2 Rede do segundo estudo de caso e terceiro

Cada conexão de transporte é obtida com cada estação fonte i (0 <= i <= k –1) tendo sua estação (remota) par i+k, onde k=n/2.

Cada usuário (Figura 4.2) tem seus próprios geradores que são estatísticamente independentes.

O arquivo executável deste simulador pode ser obtido similarmente ao caso anterior. Para rodá-lo devemos usar a seguinte linha de comando:

simulador g h i j k l m < dados01.dat

1) **g** é o tempo médio de geração de uma TPDU em uts;
2) **h** é o tempo médio de absorção de uma TPDU em uts;
3) **i** é o tempo de duração da atividade independente em uts;
4) **j** é o tamanho da janela;
5) **k** é o tempo médio de transmissão de um pacote de dados em uts;
6) **l** é o número de transmissões de arquivos simuladas;
7) **m** é o número de conexões; e
8) **dados01.dat** é um arquivo contendo as sementes dos geradores (veja o Apêndice C).

Veja os exemplos da linha de comando acima no segundo estudo de caso e no terceiro do próximo capítulo.

No próximo capítulo apresentamos as estatísticas oferecidas por estes simuladores para comparação com resultados de [MOUR 86].

Temos falado de simuladores, neste capítulo, mas na realidade o que desenvolvemos neste trabalho foi um único simulador que separamos em duas partes (Apêndice A e B) por que algumas rotinas do primeiro estudo de caso tiveram que ser alteradas para atender aos demais estudos de casos. Veja que algumas rotinas se repetem (em nome e conteúdo) nos dois Apêndices.

Se você desejar fazer uso deste simulador para obter outros resultados não apresentados no capítulo 5 ou mesmo verificar os resultados que lá apresentamos, basta seguir as instruções que apresentamos anteriormente neste capítulo. Não precisando assim, de nenhuma alteração nos programas fontes apresentados nos Apêndices A e B.

4.2 ALTERAÇÕES E EXTENSÕES FEITAS PELO USUÁRIO

O usuário pode fazer alterações neste simulador para, por exemplo, obter novas estatísticas ou mesmo simular outras redes.

Suponhamos que a alteração seja para incluir um novo protocolo de acesso ao meio, como por exemplo: barra com **CSMA-CD**. Portanto, basta incluir a rotina (suponhamos que se chame **ac_m_d_csma-cd**) que simula este protocolo no Apendice A. Se você quizer rodar o primeiro estudo de caso para este protocolo de acesso e alterar o conteúdo da rotina **esc_tem_pac_a_tx** (que escalona o evento **TEM_PACOTE_A_TRANSMITIR**) a seguir:

```
/*
* Objetivo: escalonar o evento TEM_PACOTE_A_TRANSMITIR
*
* Parametros recebidos:
*          a) estação – estação que vai consumir o Evento
*          b) ap_pac  - apontador para o pacote associado a este
*                       evento
*
* Parametros retornados: nenhum.
*
*/

esc_tem_pac_tx (estação,ap_pac)
int estação;
struct pacote *ap_pac; {

    struct evento *p;
    struct tem_pac_a_tx *q;
    p = cria_evento( );
    q = cria_tem_pac_a_tx( );
    p→e_tipo -= TEM_PACOTE_A_TRANSMITIR;
```

```
    p→e_t_ocorr = relogio;
    p→e_func = ac_m_d_an_fi;
    p→e_param = q;
    q→tpatx_estação = estação;
    q→ap_tpatx = ap_pac;
    insere(p,&cabeça_lista,l_ascendente);
}
```

para o seguinte:

```
/*
 *  Objetivo: escalonar o Evento TEM_PACOTE_A_TRANSMITIR
 *
 *  Parametros recebidos:
 *          a) estação – estação que vai consumir o Evento
 *          b) ap_pac – para o pacote associado a este
 *                      Evento
 *
 *  Paramentos retornados : nenhum.
 *
 */

  esc_tem_pac_a_tx(estacao,ap_pac)
  int estação;
  struct pacote *ap_pac; {

        struct evento *p;
        struct tem_pac_a_tx *q;
        p = cria_evento( );
        q = cria_tem_pac_a_tx( );
        p→e_tipo = TEM_PACOTE_A_TRANSMITIR;
        p→e_t_ocorr = relogio;
        p→e_func = ac_m_d_csma-cd;
        p→e_param = q;
```

```
        q→tpatx_estacao = estacao;
        q→ap_tpatx = ap_pac;
        insere(p,&cabeça_lista, l_ascendente);
    }
```

Veja que a linha 24 da primeira rotina (**p→e_func = ac_m_d_an_fi**) mudou para **p→e_func = ac_m_d_csma-cd** na segunda rotina. E esta é a única alteração que o simulador necessita para atender a esta nova situação. Faltando somente gerar o arquivo executável para esta nova situação e rodar este simulador.

Observe que a rotina **ac_m_d_csma-cd** simula a entidade acesso ao meio dependente do protocolo de acesso ao meio, portanto, não esqueça das atribuições desta entidade se você fizer a modificação acima citada.

CAPÍTULO 5

ESTUDOS DE CASOS

Para validar o nosso simulador, selecionamos três estudos de casos, para comparar os seus resultados de [MOUR 86].

Para facilitar a execução do nosso simulador com relação à entrada e saída de dados, dividimos-o em dois, o que atende ao 1º estudo de caso (apêndice A) e o que atende ao 2º e ao 3º estudos de caso (apêndice B). O que pode levar o leitor a pensar que a implementação está distante da especificação, mas não está, isto se tornou necessário em função da limitação da memória do MS-DOS (640 k), pois com o uso de variáveis dinamicamente relocáveis o nosso simulador aumenta e diminue de tamanho durante a execução o que dependendo do tamanho da amostra a simular pode ser um problema. Vale salientar (como dissemos no capítulo 3) que este simulador foi escrito inicialmente para rodar no PDP-11/34 e em função da sua limitação de memória (64 k por processo) teve que ser re-escrito para rodar no MS-DOS, e agora não queriamos correr este risco.

5.1. PRIMEIRO ESTUDO DE CASO: Sub-rede em anel com passagem de ficha

Neste estudo de caso comparamos os resultados de uma parte do nosso simulador (veja a Figura 5.1.), que simula a sub-rede de comunicação

de uma rede local em anel com passagem de ficha, preocupados apenas com a transmissão de pacotes no anel e a coleta das estatísticas: atraso e vazão, da sub-rede de comunicação. Os resultados obtidos são comparados com os do simulador do capítulo 8 de [MOUR 86]. Apenas a entidade acesso ao meio (parte independente) teve que ser escrita (para o caso específico do anel com passagem de ficha).

Nestes dois simuladores, os instantes de tempo de geração de pacotes são espaçados segundo uma distribuição exponencial. Uma vez gerado, um pacote é alocado a uma determinada estação da sub-rede de comunicação segundo uma distribuição uniforme.

A Figura 5.1. ilustra que parte do nosso simulador foi usada para este estudo de caso.

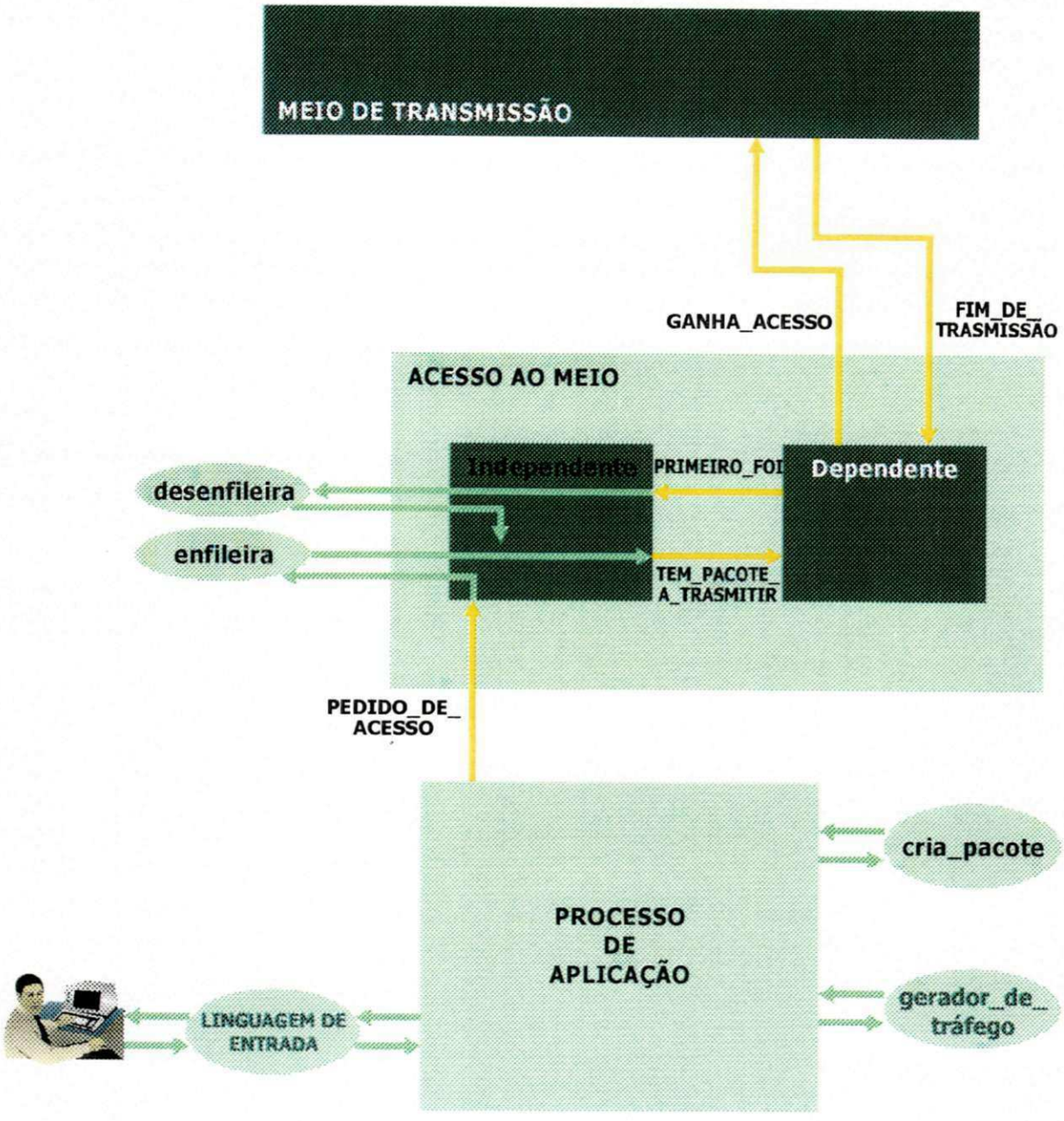

Figura 5.1 Modelo de simulação do primeiro estudo de caso

As Tabelas 5.1. e 5.2. comparam os resultados do nosso simulador com aqueles obtidos com o simulador em [MOUR 86]. Nas colunas "tempo médio de interchegadas" destas tabelas, estão os tempos de interchegadas de pacotes para acesso à sub-rede, em micro segundos. Por exemplo, na intersecção da primeira linha da Tabela 5.1. com a coluna acima citada temos o tempo médio de interchegada igual a 50.000 μs.

A Tabela 5.1. mostra os resultados do nosso simulador e os do simulador de [MOUR 86] para a estatística tempo médio de espera, ou seja, tempo médio que um pacote leva para ser transmitido, após ser gerado. Por exemplo, para o tempo médio de interchegadas de 10.000 μs, temos um tempo médio de espera por pacote igual a 151,081933 μs para o simulador de [MOUR 86] e de 150,821667 μs para o nosso simulador. Compare os outros resultados da Tabela 5.1.

TABELA 5.1

Tempo médio de espera nos dois simuladores

Tempo médio de interchegadas (em μs)	Tempo médio de espera (em μs)	
	Simulador de [MOUR 86]	Nosso simulador
50.000	64,752333	64,580733
10.000	151,081933	150,821667
5.000	228,165333	227,638400
3.333	371,409533	370,814333
2.500	549,232800	548,636400
2.222	656,430067	655,828933
2.000	762,308200	761,678200
1.750	959,986867	959,377667
1.500	1.453,103800	1.452,513600
1.250	2.728,566267	2.727,847933

A Tabela 5.2. mostra os resultados do nosso simulador e os do simulador de [MOUR 86] para a estatística utilização do meio de transmissão. Por exemplo, para a interchegada de 50.000 μs (veja a Tabela 5.2.), temos uma utilização do meio de transmissão do simulador de [MOUR 86] igual a 0,019058 e do nosso simulador de 0,019057. Compare com outros resultados da Tabela 5.2.

TABELA 5.2

Utilização do meio de transmissão nos dois simuladores

Tempo médio de interchegadas (em μs)	Utilização do meio de transmissão	
	Simulador de [MOUR 86]	Nosso simulador
50.000	0,019058	0,019057
10.000	0,095293	0,095290
5.000	0,190593	0,190588
3.333	0,285931	0,285927
2.500	0,381213	0,381213
2.222	0,428923	0,438923
2.000	0,476531	0,476531
1.750	0,544623	0,544623
1.500	0,635398	0,635398
1.250	0,762514	0,762514

Os valores apresentados nas duas tabelas acima são tão próximos (quando não são iguais) que, intuitivamente, podemos dizer (estatisticamente) que não há diferença entre os resultados dos dois simuladores. Com isto queremos atestar a validação do nosso simulador, pois os resultados apresentados pelo simulador de [MOUR 86] é um parâmetro verossímel para tal validação. Esta proximidade é atribuída aos seguintes fatores:

1) o tamanho de amostra (15.000) corresponde a 50% do ciclo do gerador de números pseudo-aleatórios;

2) a utilização do meio de transmissão nos simuladores é calculada pela soma dos tempos de transmissão dos pacotes no meio de transmissão dividido pelo tempo total de simulação. A soma mascara diferenças de números gerados individualmente;

3) A utilização é uma medida robusta que é menos dependente de flutuações estatísticas do simulador ao contrário do tempo de resposta.

Os outros dados utilizados por estes simuladores são:

a) tempo de transmissão no anel igual a 1000 μs;

b) número de estações igual a 16;

c) número de transmissões de pacotes simuladas iguais a 15.000;

d) semente para a distribuição exponencial igual a 13; e

e) semente para a distribuição uniforme igual a 17.

Cada um dos simuladores acima citados, é executado, por exemplo, usando os dados acima e o tempo médio de interchegada igual a 50.000 μs, com a seguinte linha de comando:

simulador 50000 1000 16 15000 13 17

Na linha de comando acima, "simulador" é um arquivo executável cujo conteúdo é o código objeto executável do simulador que se deseja executar.

Neste estudo de caso executamos os dois simuladores em micros computadores compatíveis com o IBM PC com clock de 4,7 Mhz. Os tempos de CPU gastos pelos dois simuladores são os seguintes:

a) para o simulador de [MOUR 86], o tempo gasto de CPU para cada tempo médio de interchegada foi em média de 10 minutos, sem o co-processador de ponto flutuante, e com o co-processador de ponto flutuante, e com o co-processador o tempo baixou para 1,3 minutos, proporcionando uma redução de 87%;

b) para o nosso simulador o tempo médio foi de 13 minutos sem o co-processador, proporcionando uma redução de 66% .

O nosso simulador gasta mais tempo por ser mais flexível (mais aberto a modificações) e por isto necessita de mais rotinas e eventos.

As rotinas no nosso simulador utilizadas neste estudo de caso são as seguintes: main, leitura_dados, inicialização, teste, cria_pacote, random, debug, f_ascendente, esc_pedido_aces, cria_evento, cria_pedido_aces, insere, l_ascendente, scheduler, ac_m_d_an_fi, ac_m_ind, enfileira, esc_tem_pac_a_tx, pass_ficha, desenfileira, remove, cria_tem_pac_a_tx,

sintaxe, esc_prim_foi, cria_prim_foi, esc_g_acesso, cria_g_acesso, meio_de_trans, mem_alloc, esc_fim_trans, col_estatistica, cria_fim_trans e resultados. Os conteúdos destas rotinas constam do Apêndice A.

5.2. SEGUNDO ESTUDO DE CASO: Protocolo de transporte de uma rede em anel com passagem de ficha

Neste estudo de caso comparamos os resultados de uma parte do nosso simulador que simula o protocolo de transporte em uma rede local em anel com passagem de ficha, com os resultados de ferramentas analíticas e simulação apresentados no capítulo 10 de [MOUR 86].

Não mostramos uma figura com o modelo desta parte do nosso simulador que simula o protocolo de transporte em uma rede local em anel com passagem de ficha, com os resultados de ferramentas analíticas e simulação apresentados no capítulo 10 de [MOUR 86].

Não mostramos uma figura com o modelo desta parte do nosso simulador por que este estudo de caso usa todo o modelo descrito no capítulo 2, com exceção da entidade processo de aplicação.

O nosso simulador neste estudo de caso tem ainda as seguintes características:

a) cada usuário (da rede) gera pedidos de transferências de arquivos com tamanho em TPDU's segundo uma distribuição geométrica;

b) a geração de TPDU's de cada servidor de arquivo é simulada através de um gerador de TPDU's com tempos de interchegada exponencialmente distribuídos;

c) a absorção das TPUD´s de cada usuário de arquivo é simulada através de um gerador de atraso com distribuição exponencial; e

d) para facilitar a implementação deste estudo de caso, incluímos duas informações na estrutura de dados dos pacotes que são: tempo de geração de arquivo e número de pacotes existentes no arquivo.

Para efetivarmos independência estatística entre os vários geradores (de arquivo, de TPDU´s, de atraso e etc.) atribuímos a cada usuário na rede simulada tantos geradores quantos necessários, cada um com sua própria semente, distinta dos demais. O código fonte para tal é mostrado abaixo (compare com os códigos das macros EXPONENCIAL e GEOMETRICA e da rotina random no Apêndice B).

```
/*
* Objetivo: gerar um número com distribuição exponencial
*
* Parâmetros que recebe:
*        a) x – Media da  distribuicao exponencial
*        b) y – Seqüencia dos Numeros pseudo-aleatorios
*        c) z – Estacao para quem vai gerar este numero
*
* Parametro que retorna: um número com distribuição exponencial
*
*/
#define EXPONENCIAL( x, y, z )      floor ( - ( x ) * \
                                    log ( 1.  – (random (y, z ) /32768. ) ) )


/*
* Objetivo: gerar um numero com distribuição geométrica
*
* Parametros que recebe:
*        a) x – media da distribuicao geometrica
*        b) y – seqüencia de numeros pseudo-aleatorios
*        c) z – estacao para quem vai gerar este numero
*
```

```
* Parametro que retorna: um numero com distribuicao geometrica
*
*/
#define GEOMETRICA(x, y, z )      floor (- ( x )* \
                                  log(1. – ( random ( y, z ) / 32768.) ) + 0.9999)


/*
* Objetivo: gerar um numero pseudo-aleatório entre 0 e 1
*
* Parametros que recebe:
*         a) no_seq – numeros da seqüencia de numeros pseudo-aleatorios
*         b) estacao- Estacao para que gerou o numero
*
* Parametros que retorna: um número pseudo-aleatorio
*
*/

int random(no_seq,estacao)
int no_seq;
int estacao; {
    srand(semte[no_seq][estacao]) ;
    semte[no_seq][estacao] = rand( ) ;
    return(semte[no_seq][estacao]) ;
}
```

Observações:

a) \ (caracter contra barra) é um caracter de continuação de linha (em C);

b) como os compiladores não aceitam acentuação não acentuamos as palavras do código acima; e

c) as macros EXPONENCIAL e GEOMETRICA não retornam parâmetros formalmente, mas as suas invocações (lugar onde

aparecem, por exemplo: dentro de uma expressão aritmética) são substituídas pelo valor que elas retornam.

Os geradores acima funcionam da seguinte forma:

1) se quisermos gerar um número com distribuição exponencial invocaremos a macro EXPONENCIAL com os parâmetros x, y e z que significam a média da distribuição exponencial, a seqüência de números pseudo-aleatórios (do gerador de TPDU's, por exemplo) e a estação, respectivamente; e

2) se desejarmos gerar um número com distribuição geométrica, devemos invocar a macro GEOMETRICA com os parâmetros x, y e z que significam a média da distribuição geométrica, a seqüência de números pseudo-aleatórios do gerador de arquivos e a estação respectivamente.

O nosso objetivo neste estudo de caso é verificar se o nosso simulador fornece alguns pontos próximos aos pontos da Figura 10.10 de [MOUR 86]. Assim, executamos o nosso simulador para 12 pontos da figura acima citada, com os dados da Tabela 10.1 de [MOUR 86], que repetimos parcialmente na Tabela 5.3 a seguir:

TABELA 5.3

Valores dos parâmetros usados no segundo estudo de caso

Parâmetro	Valor(es)
Número de usuários	60
Números de conexões simultâneas	1, 10, 20 e 30
Capacidade do anel	1 M bit / s
Tempo médio de transmissão de um pacote de dados	8 ms
Tempo médio de transmissão de um pacote de reconhecimento	0,1 ms
Tempo médio de geração de uma TPDU	0,2 s
Tempo médio de absorção de uma TPDU	0,2 s
Tempo médio de passagem de ficha entre estações	0
Tamanho da janela	1, 2 e 4

Como em [MOUR 86], o tempo de transmissão de um pacote de reconhecimento (0,1 ms) é adicionado ao tempo médio de transmissão de um pacote de dados (8ms). Obtivemos a estatística vazão por conexão (veja Tabela 5.4, a seguir) para várias configurações de rede local sugeridas na Tabela 5.3.

TABELA 5.4

Resultados do segundo estudo de caso

Número de conexões simultâneas	Vazão por conexão (arquivo / s)		
	janela = 1	janela = 2	janela = 4
1	0,1229	0,1570	0,1720
10	0,1182	0,1500	0,1680
20	0,1171	0,1480	0,1650
30	0,1156	0,1460	0,1640

Comparando os resultados da primeira linha da Tabela 5.4, acima, com os pontos correspondentes da Figura 10.10 de [MOUR 86], notamos que o ponto 0,1229 (janela = 1) da tabela acima coincide com o ponto correspondente do gráfico da Figura 10.10. Para janela = 2 existe uma diferença aproximada de 3% em relação aos resultados da simulação da Figura 10.10 de [MOUR 86] e para janela = 4, de 10%. Comparando os outros resultados, vemos que há uma tendência decrescente na Tabela 5.4 que se repete na Figura 10.10 de [MOUR 86].

As rotinas utilizadas neste estudo de caso estão no Apêndice e são as mesmas do terceiro estudo de caso. O que muda é a coleta de estatísticas, pois aqui não necessitamos coletar estatísticas da entidade processo de aplicação por isto o terceiro parâmetro (tempo médio de duração da atividade independente da linha de comando (abaixo), que é quem ativa a entidade acima citada, é zero (0). Aqui, o nosso simulador é executado, por exemplo, usando os dados para obter o ponto (vazão por conexão = 0,1720) da

interseção da primeira linha (linha o número de conexões simultâneas = 1) com a terceira coluna (janela = 4) usando a seguinte linha de comando):

simulador 200 200 0 1 8.1 500 1 < dados01.dat

Na linha de comando acima, **simulador** é também (como no primeiro estudo de caso) um arquivo executável, porém **dados01.dat** é um arquivo que contém as sementes usadas pelos geradores de números pseudo-aleatórios do simulador (veja o conteúdo deste arquivo no Apêndice C). Uma observação necessária é que usamos neste estudo de casos 1 uts igual a 1 ms.

Neste estudo de caso simulamos a transmissão de 10.000 TPDU´s por conexão de transporte, gastando uma média de 6 minutos de CPU, em micro-computador com as mesmas características do usado no primeiro estudo de casos e co-processador de ponto flutuante.

5.3 TERCEIRO ESTUDO DE CASO: Transferência de arquivos em uma rede em anel com passagem de ficha.

Neste estudo de casos comparamos os resultados do nosso simulador completo (veja a figura 2.13) simulando uma aplicação de transferência de arquivos numa rede em anel com passagem de ficha e controle de fluxo nas conexões de transporte (entre estações) por janelas deslizantes. Cada arquivo é segmentado em um número de unidades de dados do protocolo de transporte (TPDU´s). A estatística de interesse é o tempo médio de resposta para transferência de arquivos. Nossos resultados são comparados com os resultados de ferramentas analíticas e simulação apresentados no capítulo 11 de [MOUR 86].

Este estudo de caso é semelhante ao exemplo de aplicação apresentado no capítulo 2.

Neste estudo valem as observações dos itens a, b, c, e d do segundo estudo de caso, bem como, as observações sobre os geradores, além das seguintes observações:

a) O tempo médio de duração de atividade (independente) de cada usuário de arquivo, ou seja, o tempo em que o usuário de arquivo fica atendendo as atividades locais (fora da rede), é simulado, também, através de um atraso com distribuição exponencial;

b) O tempo médio de transmissão de um pacote de reconhecimento é igual a zero (0); e

c) Usamos 1 ms para 1 uts.

Faremos então a comparação de alguns resultados do nosso simulador para este estudo de caso descrito acima com os respectivos resultados da Figura 11.14 de [MOUR 86]. Para tanto, executamos o nosso simulador para 9 pontos da figura acima citada, com os dados da Tabela 11.3 de [MOUR 86], que repetimos parcialmente na Tabela 5.5 a seguir:

TABELA 5.5

Valores dos parâmetros usados no terceiro estudo de caso

Parâmetro	Valor(es)
Número de usuários de arquivos	10, 20 e 30
Números de servidores de arquivos	10, 20 e 30
Capacidade do anel	1 M bit / s
Tempo de transmissão no anel de um pacote de dados	10 ms
Tempo médio de geração uma TPDU	0,25 s
Tempo médio de absorção de uma TPDU	0,25 s
Número médio de TPDU´s / arquivo (distribuição geométrica)	25
Tempo médio de duração de atividade (independente)	10 s
Tamanho da janela	1, 2 e 4

Os resultados obtidos estão na Tabela 5.6.

TABELA 5.6

Resultados do terceiro estudo de caso

Número máximo de transferências simultâneas	Tempo médio de resposta (em segundos)		
	janela = 1	janela = 2	janela = 4
10	13,2109	10,2971	9,0251
20	13,3864	10,3124	9,1307
30	13,4899	10,3120	9,1470

Comparando os resultados da Tabela 5.6, acima, com os pontos correspondentes da Figura 11.4 de [MOUR 86] (ferramentas analíticas e simulação), concluímos, intuitivamente, que não há diferença estatística para os resultados com janela = 1 e janela = 2. Para janela = 4 existe uma diferença de ordem de 6%, aproximadamente, em relação aos resultados de [MOUR 86]. Tal diferença contudo encontra-se dentro dos intervalos de confiança reportados em [MOUR 86].

Neste estudo de caso o nosso simulador é executado (veja capítulo anterior), por exemplo, usando os dados para obter o ponto (tempo médio de resposta = 13,3864) da interseção da segunda linha (linha do número máximo de transferências simultâneas = 20) com a primeira coluna (janela = 1) da seguinte forma:

simulador 250 250 10000 25 1 10 10000 20 < dados01.dat

A linha de comando acima é semelhante à do segundo estudo de caso, já que usamos o mesmo simulador (simulador completo). A diferença se dá no terceiro argumento de entrada que não é zero (0), pois temos (neste estudo de caso) o tempo médio de duração de atividade Independente. Aqui também, os parâmetros da linha de comando estão em milésimos de segundo.

Uma observação importante é que simulamos uma transmissão de 12.500 TPDU´s (em média) por conexão de transporte, utilizando, em média, 7,5 minutos de tempo de CPU, em um micro-computador com as mesmas características do usado no primeiro estudo de caso e co-processador de

ponto flutuante. Assim para obter o ponto 13,3864, descrito acima, gastamos 150 minutos de tempo de CPU.

As rotinas do nosso simulador utilizadas neste estudo de caso são, também, todas as rotinas do Apêndice B.

CAPÍTULO 6

CONCLUSÕES E SUGESTÕES

6.1. CONCLUSÕES

No capítulo 1 apresentamos que o objetivo deste trabalho era a construção de um simulador acionado por eventos que contivesse as seguintes características: a interação com o usuário e a integração com várias alternativas para diferentes níveis de protocolo de rede fossem fáceis, modelasse a sub-camada de acesso ao meio (anel com passagem de ficha, barra com passagem de ficha e barra com CSMA-CD), gerasse tráfego de acordo com a aplicação, possibilitasse incluir detalhes do protocolo de transporte e que fizesse o levantamento de estatísticas.

Por razões de restrição de tempo, foi implementado um simulador modular acionado por eventos com as seguintes características: a interação com o usuário modela apenas a sub-camada de acesso ao meio em anel com passagem de ficha; gera tráfego para: a sub-rede de comunicação, o protocolo de transporte e para a transferência de arquivos de uma rede em anel com passagem de ficha além do levantamento de estatísticas para cada geração de tráfego.

Assim pensamos que este trabalho coloca à disposição de usuários e projetistas de redes locais, uma ferramenta valiosa para avaliação de desempenho dessas redes cuja funcionalidade pode ser estendida com adição de novos blocos.

A incorporação de novos blocos é possibilitada pela normalização das interfaces (em termos de eventos e chamadas a rotinas) entre os vários módulos do simulador. Isto para nós foi evidenciado quando estávamos implementando os estudos de casos apresentados no capítulo 5. Notamos também que a lista duplamente encadeada de eventos facilita a inclusão de novos eventos.

6.2. SUGESTÕES

Uma sugestão natural para estender este trabalho é a implementação de outros módulos (ver tabela 6.1) para atingir a totalidade dos objetivos anunciados. De acordo com os dados apresentados na Tabela 3.3 estimamos os seguintes esforços (Tabela 6.1) para estender o simulador segundo esta sugestão:

TABELA 6.1

Sugestões de extensão do trabalho

Extensão	Estimativa de Homens / Hora Necessárias
Acesso ao meio – parte dependente: barra com passagem de ficha	80
Acesso ao Meio - parte dependente: barra com CSMA-CD	100
Controle de fluxo entre estações por crédito	75
Processo de aplicação – transmissão de voz e dados	100

Como última sugestão apresentamos a implementação de uma melhor interface com o usuário que pode ser conversacional ou uma linguagem de entrada.

Esperamos, com este trabalho, ter contribuído para diminuir o tempo de desenvolvimento de ferramentas para avaliação de desempenho de redes locais.

APÊNDICE A

Código fonte do simulador para o primeiro estudo de caso no capítulo 5.

```
/* Inclusao  das bibliotecas de E/S e matemática */

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

/* Definições das Rotinas de Bibliotecas */
double log( );    /* logaritmo neperiano */
double floor( );  /* parte inteira de um numero real */
int       rand( )   /* retorna um numero aleatório entre 0 e 32767 */

/* Definicoes de Tipos  */
typedef long tempo_t;

/* definicao de marcos */
#define SIM                 1
#define NAO                 0
#define DADOS               0
#define ACK                 1
#define REM_PRIMEIRO(p)     remove (p,&p);
#define MAXESTACOES         20   /* Numero maximo de estações */
#define MAXFILAS            3   /* Numero maximo de filas por estacao */
#define FILAACMEIO          1   /* Fila do Acesso ao Meio */
#define FILAPROCTRANS       2   /* Fila do Processamento de Transmissao */
#define NUMSEQ              2   /* USADO PARA TESTE */
#define SEQFILA             1   /* IDEM  */
```

```
#define SEQCHEG          0  /* IDEM */
#define FILAPACCBLOQ     0  /* Fila dos pacotes bloqueados */

/* Variáveis e parametros do Simulador */
tempo_t  relogio
tempo_t  t_transmissao;
tempo_t  t_interchegada;
int      no_transmissao;
int      no_interfeces;
int      ack;
int      nada_a_trans;   /* sinalizacao de que nao ha pacotes a transmitir */
int      semte[NUMSEQ];      /* sementes para as sequencias aleatorias */
int      guarda_semte[NUMSEQ];/*Lembram das sementes iniciais da rodada */
int      sinal[MAXESTACOES];   /* Sinal das estações prontas a transmitir */
int     cfee_atual[MAXESTACOES]; /* Recursos utilizados pelo CFEE */
int     cfee_ent[MAXESTACOES];  /* Numero maximo de recursos do CFEE */
int      cfae_atual[MAXESTACOES] /* Recursos utilizados pelo CFAE */
int      cfae_ent[MAXESTACOES]   /* Numero maximo de recursos do CFAE */
int      aces_bloq[MAXESTACOES];   /*Bloquear o Acesso a estacao */

/*  Variaveis para coleta de Estatisticas */
int          no_amostras;
tempo_t      tempo_espera;
tempo_t      tempo_servidor_ocupado;

/* macros usadas para debug */
#define  SCHEDULER          0
#define  MEIO               1
#define  ACESSO_AO_MEIO     2
#define  RECEPÇÃO           3
#define  TRANSMISSAO        4
#define  CONTROLE_DE_FLUXO  5
#define  APLICAÇÃO          6
#define  ESTATISTICAS       7
```

```
#define  MAXDEBUG                8

/*  Estrutura de Dados para o debug */
struct debug {
        char  *d_nome ;
        int     d_flag ;
};

/* Estrutura para os apontadores frente e tras */
struct cab_lista {
        struct cab_lista  *frente ;
        struct cab_lista  *tras ;
};

/* Estrutura para a lista encadeada dos eventos  */
struct evento  {
        struct cab_list   ligações ;
        int     (*e_func)( ); /* apontador para a funcao associada ao evento */
        char      *e_param;
        int         e_tipo;
        tempo_t  e_t_ocorr;
};

/* Estrutura para os pacotes
struct pacote {
        struct cab_lista ligações;
        int   fonte;
        int   destino;
        int   p_tipo;
        int   p_tamanho ;
        tempo_t p_geração ;
};
```

```
/*  Definicao de Macros usadas pelos Eventos */
#define  GANHA_ACESSO                  0
#define  FIM_DE_TRANSMISSAO            1
#define  RECEP_DE_PACOTE               2
#define  PRIMEIRO_FOI                  3
#define  TEM_PACOTE_A_TRANSMITIR       4
#define  PEDIDO_DE_ACESSO              5
#define  TRANS_SEM_CFAE                6
#define  PACOTE_ACEITO                 7
#define  PAC_A_TRANSMITIR              8
#define  PODE_TENTAR_ACESSO            9
#define  PACOTE_LIBERADO               10

/* Declaração do arranjo de apontadores nome_evento */
extern char *nome_evento[ ];

/* Estrutura para o evento GANHA_ACESSO */
struct g_acesso  {
        int  ga_estacao;
        struct pacote *ap_g_acesso;
};

/*  Estrutura para o evento FIM_DE_TRANSMISSÃO  */
struct fim_ trans  {
        int  ft_estacao;
        struct  pacote  *ap_fim_trans;
};

/* Estrutura para o evento RECEPCAO_DE_PACOTE  */
struct rec_pac  {
        int rp_estacao;
        struct pacote  *ap_recpac;
};
```

```
/* Estrutura para o evento PACOTE_A_TRANSMITIR */
struct pac_a_trans {
        int pat_estacao;
        struct pacote *ap_patrans;
};

/* Estrutura para o evento PACOTE_ACEITO */
struct pac_aceito {
        int pa_estacao;
        struct pacote *pa_paceito;
};

/* Inicializacao das Rotinas de Criação das Estruturas */
struct evento       *cria_evento( );
struct g_acesso     *cria_g_acesso( );
struct fim_trans    *cria_fim_trans( );
struct rec_pac      *cria_rec_pac( );
struct prim_foi     *cria_prim_foi( );
struct tem_pac_a_tx     *cria_tem_pac_a_tx( );
struct pedido_aces      *cria_pedido_aces( );
struct pode_t_acesso    *cria_pode_t_acesso( );
struct tx_sem_cfae      *cria_tx_sem_cfae( );
struct pac_liberado     *cria_pliberado( );
struct pac_a_trans      *cria_p_a_trans( );
struct pac_aceito       *cria_aceito( );
struct pacote           *cria_pacote( );

/* Definicao de funções Macros */
                                /* distribuicao exponencial */
#define EXPONENCIAL(x,y)    floor (-(x)*log(1.-(random(y)/32768.)))

                                /* distribuicao uniforme */
#define UNIFORME(x)             (random(x)/32768.)
```

```
/* Inicializacao da rotina de números aleatorios */
int  random( );     /* retorna um numero inteiro entre 0 e 32767   */

/* Outras Rotinas  */
int  inicio( );
int  fim( );
int  l_ascendente( );
int  l_descendente( );
int  ganha_acesso( );
int  ac_m_ind( );
int  ac_m_d_an_fi( );
int  fim_trans( );
int  proc_recepção( );
int  proc_trans( );
int  aplicação( );

/* Declaração de apontadores para as estruturas */
struct evento *cabeca_lista;

/* Apontadores para as Filas de Pacotes */
struct pacote *frente_fila[MAXESTACOES][MAXFILAS];

/* inicializacao da Estrutura de Dados de debug */
struct debug tabdebug[ ] = {
        {"sched",0},
        {"meio",0},
        {"acmeio",0},
        {"recep",0},
        {"trans",0},
        {"cfluxo",0},
        {"aplic",0},
        {"estat",0},
};
```

```
/* Inicializacao da Estrutura de Dados dos eventos */
char *nome_evento[ ] = {
        {"ganha acesso"},
        {"fim de transmissao"},
        {"recepcao de pacote"},
        {"primeiro foi"},
        {"tem pacote a transmitir"},
        {"pedido de acesso"} ,
        {"transmissao sem CFAE"},
        {"pacote aceito"},
        {"pacote a transmitir"},
        {"pode tentar acesso"},
        {"pacote liberado"},
};

/* Declaração de um arranjo para debug
extern struct debug tabdebug[ ] ;

/* Leitura dos dados de entrada */
leitura_dados(argc,argv)
int     argc;
char    *argv[ ]; {
        tempo_t atol( ) ;  /* conversao ascii para long (biblioteca)  */
        int         atoi( ) ;   /* conversao ascii para long (biblioteca)  */
        int  i, aux ;
        while(argc  >  1 &&  argv[1][0] == '-'  {
                switch(argv[1][1])  {
                case 'd':
                        if(argc  >  2)  {
                                for (i = 0; i  < MAXDEBUG; i++)  {
                                    if(strcmp(tabdebug[i].d_nome,argv[2] == 0) {
                                            tabdebug[ i ].d_flag = SIM ;
                                            break;
                                    }
```

```
                        }
                        if(i  > = MAXDEBUG)  {
                                perror ("nivel de debug %s nao existe",
                                        argv[2]) ;
                        }
                        argc -= 2;
                        argv += 2;
                } else {
                        sintaxe( );
                }
                break;
        }
    }
    if(argc != NUMSEQ + 5) {
        perror ("Sintaxe: t_interchegada t_transmissao no_interfaces \
    no_transmissoes semente1 semente2\n");
        exit(0);
    }
    t_interchegada = atol(arg[1]);
    t_transmissao = atol(arg[2]);
    if( (no_interfaces = atoi(argv[3]))  >  MAXESTACOES)  {
        perror ("Voce excedeu %d, o numero maximo de estacoes \n",
            MAXEXTACOES);
        exit(0);
    }
    no_transmissao = atoi(argv[4]);
    for(aux = 0; aux < NUMSEQ; aux++)
        guarda_semte[aux] = semte[aux] = atoi(argv[5 + aux]);
}

sintaxe( ) {
}
```

```
/* Rotina que insere, em filas, em ordem ascendente */
f_ascendente(anodo,aatual)
struct pacote *anodo,*aatual; {
        return(anodo→p_geracao > aatual→p_geracao) ;
}

/* Rotina de enfileiramento */
enfileira(estacao,fila,ap_pac)
int estacao;
int fila;
struct pacote *ap_pac; {
        insere(ap_pac,&frente_fila[estacao][fila],f_ascendente);
}

/* Rotina de desenfileiramento */
desinfileira(estacao,fila)
int estacao;
int fila; {
        REM_PRIMEIRO(frente_fila[estacao][fila]);
}

/* Rotina provisória para coleta de estatisticas */
col_estatistica(ap_pac)
struct pacote *ap_pac; {
        tempo_espera +=relógio - (ap_pac→p_geracao);
        no_amostra++;
        tempo_servidor_ocupado += t_transmissao;
        debub(ESTATISTICAS,"no_amostras %d\n",no_amostras);
}

/* Rotina de inicializacao das variações globais */
inicializacao( )
{
        int i;
```

```
        relógio = 0;
        ack = 0;
        no_amostras = 0;
        tempo_espera = 0;
        tempo_servidor_ocupado = 0;
        nada_a_trans = SIM;
        for (i = 0; i < no_interfaces; i++) {
                sinal[ i ] = NÃO;
                frente_fila[ i ] [FILAACMEIO] = NULL;
                frente_fila[ i ] [FILAPROCTRANS] = NULL;
                frente_fila[ i ] [FILAPACBLOQ] = NULL;
                cfae_atual[ i ] = 0;

                /* para teste */
                cfae_ent[ i ] = 1;
                cfee_ent[ i ] 1;
                cfee_atual[ 1 ] = 0;
                aces_bloq[ i ] = NÃO;
        }

        /* para teste */
        teste ( );
}

/* Rotina de debug */
debug(nível,s)
int     nivel;
char    *s;
{
        if(!tabdebug[nivel].d_flag)  return ;
                printf("%s: ",tabdebug[nivel].d_nome) ;
                printf("%r",&s) ;
}
```

```
/* Rotina que remove um nodo de uma lista duplamente encadeada */
remove(anodo,acab)
struct  cab_lista *anodo;
struct  cab_lista **acab;
{
        if((*acab) == NULL) {    /* lista vazia */
                perror("Remove encontrou a lista vazia\n");
                exit(0);
        }
        anodo→tras→frente = anodo→frente;
        anodo→frente→tras = anodo→tras;
        if(*acab == anodo) {     /* remove o primeiro da lista  */
                *acab = anodo→frente;
        }
        if(*acab == anodo) {     /* lista ficou vazia  */
                *acab = NULL;
        }
}

l_ascendente(anodo,aatual)  /*inserir em ordem ascendente  */
struct evento  *anodo, *aatual ;
{
                return(anodo→e_t_ocorr  >  aatual→e_t_ocorr);
}

/* Rotina que insere um nodo em uma lista duplamente encadeada  */
insere(anodo,acab,comp)
struct cab_lista   *anodo;
struct cab_lista   **acab;
int     (*comp)( );
{
        struct cab_lista *an;
        an = *acab;
        if(an == NULL)  {  /* Lista vazia  */
```

```
            anodo→frente = anodo→trás = *acab = anodo;
            return;
        }
        while((*comp)(anodo,an)  &&  (an = na→frente)  != *acab);
        anodo→frente = an;
        anodo→tras = an→tras;
        an→tras→frente = anodo;
        an→tras = anodo;
        /* Seta cabecalho */
        if ((*comp)(anodo,*acab) == 0)  {
            acab = anodo;      /* anodo na frente  */
        }
}

/* Rotina principal */
main(argc,argv)
int     argc;
char  *argv[ ];
{
        leitura_dados(argc_argv) ;
        inicializacao( );
        scheduler( );
        resultados( );
        exit(0);
}

/* Rotina que gera um numero pseudo-aleatorio */
int random(no_seq)
int no-seq;
{
        srand(semte[no_seq]);
        semte[no_seq] = rand( );
        return(semte[no_seq]);
}
```

```
/* Rotina que dispara os eventos da lista duplamente encadeada */
scheduler( )
{
        struct evento  *ap_evento ;
                /* Nao  eh fim de simulação */
        while(no_amostras < no_transmissao)  {
                if(cabeca_lista == NULL)  {
                        perror("Scheduler não achou evento na lista\n");
                        exit(0);
                }
                ap_evento = cabeca_lista;
                REM_PRIMEIRO(cabeca_lista);
                relógio = ap_evento→e_t_ocorr;
                debug(SCHEDULER,"tempo %ld, evento tipo %s\n",relogio,
                                nome_evento[ap_evento→e_tipo]);
                (*ap_evento→e_func)(ap_evento);
                free(ap_evento→e_param);
                free(ap_evento);
        }
}

/* Rotina que escalona o primeiro evento */
teste( )  {
        /* para teste */
        struct pacote *ap_pac ;
        ap_pac = cria_pacote( );
        ap_pac→fonte = (int)(UNIFORME(SEQFILA)  * no_interfaces);
        if(ap_pac→fonte > (no_interfaces – 1)  {
                perror("Numero de ap_pac→fonte %d muito grande \n",
                        ap_pac→fonte) ;
                exit(0);
        }
        ap_pac→destino = 0;
        ap_pac→p_tipo = "dados";
```

```
        ap_pac→p_tamanho = 512;
        esc_pedido_aces(ap_pac→fonte,ap_pac);
}

/* Rotina que simula a Entidade Acesso ao Meio Independente */
ac_m_ind(ap_ev)
struct evento *ap_ev; {
        struct prim_foi  *p;
        struct pedido_aces *q;
        debug(ACESSO_AO_MEIO,"evento tipo %s\n",
                nome_evento[ap_ev→e_tipo]);
        switch(ap_ev→e_tipo) {
                case PEDIDO_DE_ACESSO:
                        q = ap_ev→e_param;
                        enfileira(q→pa_estacao,FILAACMEIO,q→ap_pacess);
                        if(!aces_bloq[q→pa_estacao]) {
                            esc_tem_pac_a_tx(q→pa_estacao,q→ap_pacess);
                            aces_bloq[q→pa_estacao] = SIM;
                        }
                        teste( );
                        break ;

                case PRIMEIRO_FOI:
                        p = ap_ev→e_param;
                        desenfileira(p→pf_estacao, FILAACMEIO);
                                /*para teste  */
                        /* pac_transmitido(p→pf_estacao,p→ap_p_foi); */
                        free(p→ap_p_foi);
                        aces_bloq[p→pf_estacao]  =  NÃO;
                        if(frente_fila[p→pf_estacao][FILAACMEIO] != NULL) {
                                esc_tem_pac_a_tx(p→pf_estacao,
                                        frente_fila[p→pf_estacao][FILAACMEIO]);
                                aces_bloq[p→pf_estacao]  = SIM;
                        }
```

```
                    break;
        }
}

/* Rotina de escalonamento do evento PRIMEIRO_FOI */
esc_prim_foi(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct prim_foi *q;
        p = cria_evento( );
        q = cria_prim_foi( );
        p→e_tipo = PRIMEIRO_FOI;
        p→e_t_ocrr = relogio;
        p→e_func = ac_m_ind;
        p→e_param = q;
        q→pf_estacao = estacao;
        q→ap_p_foi = ap_pac;
        insere(p,&cabeca_lista,l_ascendente);
}

/* Rotina de escalonamento do evento TEM_PACOTE_A_TRANSMITIR */
esc_tem_pac_a_tx(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct tem_pac_a_tx *q;
        p = cria_evento( );
        q = cria tem_pac_a_tx( );
        p→e_tipo = TEM_PACOTE_A_TRANSMITIR;
        p→e_t_ocorr = relogio;
        p→e_func = ac_m_d_an_fi;
        p→e_param = q;
        q→tpatx_estacao = estacao;
```

```
        q→ap_tpatx = ap_pac;
        insere(p,&cabeca_lista,l_ascendente);
}

/* Rotina de escalonamento do evento PEDIDO_DE_ACESSO */
esc_pedido_aces(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct pedido_aces *q;
        p = cria_evento( );
        q = cria_pedido_aces( );
        p→e_tipo = PEDIDO DE ACESSO;
        p→e_t_ocrr = relogio +
                (tempo_t)(EXPONENCIAL(t_interchegada, SEQCHEG));
        p→e_func = ac_m_ind;
        p→e_param = q;
        ap_pac→p_geração = p→e_t_ocorr;
        q→pa_estacao = estacao;
        q→ap_pacess = ap_pac;
        insere (p,&cabeca_lista,l_ascendente);
}

/* Rotina de escalonamento do evento GANHA_ACESSO */
esc_g_acesso(estacao)
int estacao;
        struct evento *p;
        struct g_acesso *q;
        p = cria_evento( );
        q = cria g_acesso( );
        p→e_tipo = GANHA_ACESSO;
        p→e_t_ocorr = relogio;
        p→e_func = meio_de_trans;
        p→e_param = q;
```

```
        q→ga_estacao = estacao;
        q→ap_g_acesso = frente_fila[estacao][FILAACMEIO];
        insere(p,&cabeca_lista,l_ascendente);
}

/* Rotina que simula a Entidade Acesso ao Meio Dependente:
 * em Anel com Passagem de Ficha
*/
ac_m_d_an_fi(ap_even)
struct evento  *ap_even;  {
        struct tem_pac_a_tx *p;
        struct fim_trans  *q;
        debug(ACESSO_AO_MEIO,"evento tipo %s\n",
                nome_evento[ap_even→e_tipo]);
        switch(ap_even→e_tipo)  {
                case FIM_DE_TRAMISSAO:
                        q = ap_even→e_param;
                        pass_ficha( );
                        esc_prim_foi(q→ft_estacao,q→ap_fim_trans);
                        break;
                case TEM_PACOTE_A_TRANSMITIR:
                        p = ap_even→e_param;
                        sinal[p→tpatx_estacao] = SIM;
                        if(nada_a_trans == SIM)  {
                                nada_a_trans  = NAO;
                                pass_ficha( );
                        }
                        break;
        }
}

/* Rotina que passa a ficha para a próxima estacao */
pass_ficha( )  {
        static int ficha = 0;
```

```
        int   i;
        for(ficha = (ficha + 1) % no_interfaces, i = 0;
                sinal[ficha] == NÃO && i < no_interfaces;
                        ficha = (ficha + 1) % no_interfaces, i++)
                        ;
        if(sinal[ficha] == NÃO)  {
                nada_a_trans = SIM;
                return;
        } else  {
                esc_g_acesso(ficha);
                sinal[ficha] = NÃO;
        }
}

/* Rotina que aloca espaço na Memória e imprime mensagem
 * de falta da mesma
 */
char *mem_alloc(num,tamanho,msg)
int     num;
int     tamanho;
char   *msg;
{
        char   *q;
        if((q = calloc(num,tamanho)) == NULL)  {
                perror("Falta Memória para cirar %s\n",msg);
        } else  {
                return(q);
        }
}

/* Rotina que cria a estrutura de dados pacote */
struct pacote *cria_pacote( )  {
        return((struct pacote *)mem_alloc(1,sizeof(struct pacote), "pacote"));
}
```

```
/* Rotina que cria a estrutura de dados evento */
struct evento *cria_evento( )  {
        return((struct evento *)mem_alloc(1,sizeof(struct evento),"evento"));
}

/* Rotina que cria a estrutura de dados g_acesso */
struct g_acesso *cria_g_acesso( ) {
        return((struct g_acesso *)mem_alloc(1,sizeof(struct g_acesso),
                        "g_acesso"));
}

/* Rotina que cria a estrutura de dados fim_trans */
struct fim_trans *cria_fim_trans( ) {
        return((struct fim_trans *)mem_alloc(1,sizeof(struct fim_trans),
                        "fim_trans"));
}

/* Rotina que cria a estrutura de dados prim_foi  */
struct prim_foi  *cria_ prim_foi( ) {
        return((struct prim_foi *)mem_alloc(1,sizeof(struct prim_foi),"prim_foi"));
}

/* Rotina que cria a estrutura de dados tem_pac_a_tx */
struct tem_pac_a_tx  *cria_tem_pac_a_tx( ) {
        return((struct tem_pac_a_tx *)mem_alloc(1,
                sizeof(struct tem_pac_a_tx),"tem_pac_a_tx"));
}

/* Rotina que cria a estrutura de dados  pedido_aces  */
struct pedido_aces  *cria_pedido_aces( ) {
        return((struct pedido_aces  *)mem_alloc(1,sizeof(struct pedido_aces),
                        "pedido_aces"));
}
```

```
/* Rotina de escalonamento do evento FIM DE TRANSMISSAO  */
esc_fim_trans(estacao,ap_pac)
int estacao;
struct pacote *ap_pac;  {
        struct evento *p;
        struct fim_trans *q;
        p = cria_evento( );
        q = cria fim_trans( );
        p→e_tipo = FIM_DE_TRANSMISSAO;
        p→e_t_ocorr = relogio + t_transmissao;
        p→e_func = ac_m_d_an_fi;
        p→e_param = q;
        q→ft_estacao = estacao;
        q→ap_fim_trans = ap_pac;
        insere(p,&cabeca_lista,l_ascendente);
}

/* Rotina que simula a Entidade Meio de Transmissao  */
meio_de_trans(ap_ev)
struct evento *ap_ev;
{
        struct pacote *p;
        struct g_acesso *q;
        q = ap_ev→e_param;
        debug(ACESSO_AO_MEIO,"a estacao %d ganhou acesso",
                                q→ga_estacao);
        /* para teste:  excluir os dois comandos a seguir  */
/*      p = duplica_pacote(q→ap_g_acesso);
        esc_rec_pac(p); */
        esc_fim_trans(q→ga_estacao,q→ap_g_acesso);
        col_estatística(q→ap_g_acesso);
}
```

```
/* Rotina que imprime os resultados da simulacao */
resultados( ) {
    printf("Resultados do Pacote Simulador: Estudo de caso\n");
    printf("da Sub-Rede em Anel com Passagem de Ficha\n");
    printf("\n");
    printf("ENTRADAS:\n");
    printf("Tempo Medio de Interchegadas = %ld\n",t_Interchegada);
    printf("Tempo de Transmissao no Anel = %ld\n",t_transmissao);
    printf("Numero de Interfaces = %d\n",no_interfaces);
    printf("Numero de Transmissoes Simuladas = %d\n",no_amostras);
    printf("Semente Distribuição Exponencial = %d\n",guarda_semte[0]);
    printf("Semente Distribuição Uniforme = %d\n",guarda_semte[1];
    printf("\n");
    printf("RESULTADOS:\n");
    printf("Tempo Medio de Espera = %f\n",
                        (double)tempo_espera/no_amostras);
    printf("Utilizacao do Meio de Transmissao = %f\n",
                        (double)tempo_servidor_ocupado/relogio);
}
```

APÊNDICE B

Código fonte do Simulador para o segundo estudo de caso e terceiro apresentados no capítulo 5.

```
/* Inclusão das bibliotecas de E/S e matemática */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <io.h>

/* Definições das Rotinas de Bibliotecas */
double log( );   /* logaritmo neperiano */
double floor( ); /* parte inteira de um número real */
int rand( ); /* retorna um número aleatório entre 0 e 32767 */

/* Definições de Tipos */
typedef long tempo_t;
typedef double tempo_col_est;

/* Definição de macros */
#define SIM                 1
#define NAO                 0
#define PRIMEIRO            1
#define DADOS               0
#define ACK                 1
#define REM_PRIMEIRO(p)     remove (p,&p);
#define MAXESTAÇÕES         200 /* Número máximo de estações */
#define MAXFILAS            3  /* Número máximo de filas por estação */
```

```
#define FILAACMEIO          2  /* Fila do Acesso ao Meio */
#define FILAPROCTRANS     1 /* Fila do Processamento de Transmissão */
#define FILAPROCAPLIC       0 /* Filas dos processos de aplicação */

        /* Usado para teste */
#define NUMSEQ        4      /* Número de arrays de seqüências
                                de números aleatórios */
#define SEQCONSF      0      /* Seqüência do gerador de pacotes
                                das estacoes fontes */
#define SEQCONSR      3      /* Seqüência do sorvedouro de pacotes
                                das estações remotas */
#define SEQGEO        1      /* Seqüência do gerador dos números de
                                        pacotes */
#define SEQATIV       2      /* seqüência de ativação das estações */

/* Variáveis e parâmetros do simulador*/
tempo_col_est relógio;
tempo_col_est t_transmissão;
tempo_t t_interchegada;
int  no_arquivos;
int  no_conexoes;
int  ack;
int  nada_a_trans;          /* Sinalização de que nao há pacotes a transmitir */
int  semte[NUMSEQ][MAXESTACOES]   /* Semelhante para as seqüências
                                                aleatórias */
int  guarda_semte[NUMSEQ][MAXESTACOES];  /*Lembram das sementes
                                                iniciais da rodada */
int sinal[MAXESTAÇÕES];  /* Sinal das estações prontas a transmitir */
int cfee_atual[MAXESTACOES];  /* Recursos utilizados pelo CFEE */
int cfee_ent[MAXESTACOES];  /* Número máximo de recursos do CFEE */
int cfae_atual[MAXESTACOES];  /* Recursos utilizados pelo CFAE */
```

```
int cfae_ent[MAXESTACOES]; /* Número máximo de recursos do CFAE */
int aces_bloq[MAXESTACOES]; /* Bloquear o Acesso a estação */
int pode_retrans[MAXESTACOES] ;
int no_pac_bcfluxo[MAXESTAÇÕES] ;
int no_m_pac;
tempo_t t_fonte;
tempo_t t_sorvedouro;
tempo_t t_m_ativ_aplic;
int tam_janela;
struct pacote *pac_bloqueado[MAXESTACOES];
int no_prim_pac;

/* Variáveis para coleta de Estatísticas */
int no_amostras;
tempo_col_est   tempo_espera;
tempo_col_est   tempo_servidor_ocupado;

/* Macros usadas para debug */
#define SCHEDULER                0
#define MEIO                     1
#define ACESSO_AO_MEIO           2
#define RECEPÇÃO                 3
#define TRANSMISSÃO              4
#define CONTROLE_DE_FLUXO        5
#define APLICAÇÃO                6
#define ESTATISTICAS             7
#define MAXDEBUG                 8

/* Estrutura de Dados para o debug */
struct debug {
        char *d_nome;
```

```
        int d_flag;
};

/* Estrutura para os apontadores frente e tras */
struct cab_lista {
        struct cab_lista *frente;
        struct cab_lista *tras;
};

/* Estrutura para a lista encadeada dos eventos */
struct evento {
        struct cab_lista ligações;
        int   (*e_func)( );  /* apontador para a função associada
                                ao evento */
        char * e_param;
        int  e_tipo;
        tempo_t e_t_ocorr;
};

/* Estrutura para os pacotes */
struct pacote {
        struct cab_lista ligações;
        int fonte;
        int destino;
        int p_tipo;
        int p_tamanho;
        int p_no_pac;
        int p_ult_pac;
        tempo_t p_geração;
};
```

```
/* Definição de macros usadas pelos eventos */
#define GANHA_ACESSO                    0
#define FIM_DE_TRANSMISSÃO              1
#define REC_PAC                         2
#define PRIMEIRO_FOI                    3
#define TEM_PACOTE_A_TRANSMITIR         4
#define PEDIDO_DE_ACESSO                5
#define TX_SEM_CFAE                     6
#define PACOTE_ACEITO                   7
#define PACOTE_A_TRANSMITIR             8
#define PODE_TENTAR_ACESSO              9
#define PACOTE_LIBERADO                 10
#define APLICAÇÃO_TRANS_ARQUIVO         11

/* Declaração do arranjo de apontadores nome_evento */
extern char *nome_evento[ ];

/* Estrutura para o evento GANHA_ACESSO */
struct g_acesso {
        int ga_estação;
        struct pacote *ap_g_acesso;
};

/* Estrutura para o evento FIM_DE_TRANSMISSÃO */
struct fim_trans {
        int ft_estação;
        struct pacote *ap_fim_trans;
};

/* Estrutura para o evento REC_PAC */
struct rec_pac {
```

```
        int rp_estação;
        struct pacote *ap_recpac;
};

/* Estrutura para o evento PRIMEIRO_FOI */
struct prim_foi {
        int  pf_estação;
        struct pacote *ap_p_foi;
};

/* Estrutura para o evento TEM_PACOTE_A_TRANSMITIR */
struct tem_pac_a_tx {
        int  tpatx_estacao;
        struct pacote *ap_tpatx;
};

/* Estrutura para o evento PEDIDO_DE_ACESSO */
struct pedido_aces {
        int pa_estacao;
        struct pacote *ap_pacess;
};

/* Estrutura para o evento PODE_TENTAR_ACESSO */
struct pode_t_acesso {
        int pta_estacao;
};

/* Estrutura para o evento TX_SEM_CFAE */
struct tx_sem_cfae {
        int tscfae_estacao;
        struct pacote *ap_tscfae;
```

```
};

/* Estrutura para o evento PACOTE_LIBERADO */
struct pac_liberado {
        int pl_estacao;
};

/* Estrutura para o evento PACOTE_A_TRANSMITIR */
struct pac_a_trans {
        int pat_estacao;
        struct pacote *ap_patrans;
};

/* Estrutura para o evento PACOTE_ACEITO */
struct pac_aceito {
        int pacto_estacao;
        struct pacote *ap_paceito;
};

/* Estrutura para o evento APLICACAO_TRANS_ARQUIVO */
struct trans_arquivo {
        int apta_estacao;
};

/* Inicializacao das Rotinas de Criação das Estruturas */
struct evento           *cria_evento( );
struct g_acesso         *cria_g_acesso( );
struct fim_trans        *cria_fim_trans( );
struct rec_pac          *cria_rec_pac( );
struct prim_foi         *cria_prim_foi( );
struct tem_pac_a_tx     *cria_tem_pac_a_tx( );
```

```
struct pedido_aces        *cria_pedido_aces( );
struct pode_t_acesso      *cria_pode_t_acesso( );
struct tx_sem_cfae        *cria_tx_sem_cfae( );
struct pac_liberado       *cria_pliberado( );
struct pac_a_trans        *cria_p_a_trans( );
struct pac_aceito         *cria_aceito( );
struct pacote             *cria_pacote( );
struct trans_arquivo      *cria_t_arquivo( );

/* Definição de funções Macros */
                                  /* distribuicao exponencial */
#define EXPONENCIAL(x, y, z)  floor(-(x)*log(1. - (random(y, z)/32768.)))
                                  /* distribuicao uniforme */
#define UNIFORME(x, y)        (random(x, y)/32768.)
                                  /* distribuicao geométrica */
#define GEOMETRICA(x, y, z)   floor(-(x)*log(1. - (random(y, z)/32768.))+0.9999)

/* Inicializacao da rotina de números aleatórios */
int random( );            /*retorna um número inteiro entre 0 e 32767 */

/* Outras Rotinas */
struct pacote  *ger_pac_dados( );
int inicio( );
int fim( );
int l_ascendente( );
int l_descendente( );
int meio_de_trans( );
int ac_m_ind( );
int ac_m_an_fi( );
int fim_trans( );
int proc_recepcao( );
```

```
int proc_trans( );
int aplicacao( );

/* Declaração de apontadores para as estruturas */
struct evento *cabeca_lista;

/* Apontadores para as Filas de Pacotes */
struct pacote  *frente_fila[MAXESTACOES][MAXFILAS];

/* Apontadores para os pacotes bloqueados pelo CFAE */
struct pacote  *pac_bloqueado[MAXESTACOES];

/* Inicializacao da estrutura de dados de debug */
struct debug tabdebug[ ] = {
        {"sched",0},
        {"meio",0},
        {"acmeio",0},
        {"recep",0},
        {"trans",0},
        {"cfluxo",0},
        {"aplic",0},
        {"estat",0},
};

/* Inicializacao da estrutura de dados dos eventos */
char *nome_evento[ ] = {
        {"ganha acesso"},
        {"fim de transmissao"},
        {"recepcao de pacote"},
        {"primeiro foi"},
        {"tem pacote a transmitir"},
```

```
        {"pedido de acesso"},
        {"transmissao sem CFAE"},
        {"pacote aceito"},
        {"pacote a transmitir"},
        {"pode tentar acesso"},
        {"pacote liberado"},
};

/* Declaração do array tabdebug */
extern struct debug tabdebug[ ];

/*
 * Obs.: as rotinas ou partes de rotinas que estão como comentários
 * foram desligadas para rodarmos o segundo e terceiro estudos de caso
*/

/* Rotina que atualiza os recursos do controle de fluxo de acesso
 * a estacao podendo liberar um pacote bloqueado */
cfae_db(estacao)
int estacao; {
        debug(CONTROLE_DE_FLUXO, "Recursos de CFAE usados pela
                estacao %d = %d \n",estacao,cfae_atual[estacao]);
        if(pac_bloqueado[estacao]) {
                cfae_atual[estacao] --;
                pac_bloqueado[estacao] = NAO;
                return(SIM);
        } else {
                cfae_atual[estacao] --;
                return(NAO);
        }
}
```

```
/* Rotina que verifica se um pacote pode ou não ter
 * acesso a estação */
cfae_b(estacao,ap_pac)
int estacao;
struct pacote  *ap_pac; {
        if(cfae_atual[estacao] < cfae_ent[estacao]) {
                cfae_atual[estacao] ++;
                return(NAO);
        } else {
                pac_bloqueado[estacao] = SIM;
                return(SIM);
        }
}

/* Leitura dos dados de entrada */
leitura_dados(argc,argv)
int     argc;
char   *argv[ ];
{
        tempo_t atof( );  /* conversao ascili para long (biblioteca) */
        int     atoi( );    /* conversão ascili para int (biblioteca) */
        int i;

        while(argc > 1 && argv[1][0] == ' - ') {
                        switch (argv[1][1] ) {
                        case 'd':
                                if(argc > 2) {
                                        for (i = 0; i < MAXDEBUG; i++) {
                                                if(strcmp(tabdebug[i].d_nome,
                                                        argv[2]) == 0) {
                                                        tabdebug[i].d_flag=SIM;
```

```
                                        break;
                                }
                        }
                        if (i >= MAXDEBUG) {
                                perror ("nível de debug %s nao existe",
                                        argv[2]);
                        }
                        argc -= 2;
                        argv +=2;
                }else {
                        sintaxe( );
                }
                break;
        }
    }
    if(argc ! = 9) {
        perror ("Sintaxe: t_fonte t_sorvedouro t_m_ativ_aplic no_m_pac \
tam_janela t_transmissao no_arquivos no_conexoes \n");
        exit(0);
    }
    t_fonte = atol(argv[1]);
    t_sorvedouro = atol(argv[2]);
    t_m_ativ_aplic = atol(argv[3]);
    no_m_pac = atoi(argv[4]);
    tam_janela = atoi(argv[5]);
    t_transmissao = atof(argv[6]);
    no_arquivos = atoi(argv[7]);
    if ((no_conexoes = atoi(argv[8])) > MAXESTACOES / 2) {
        perror ("Voce excedeu %d, o numero maximo de estacoes \n",
            MAXESTACOES);
        exit(0);
```

```
        }
        for (i = 0; i < no_conexoes; i++) {
                cfee_ent[i] = tam_janela;
        }
        for (i = 0; i < no_conexoes; i++) {
                scanf("%d \n", &semte[0][i]);
                guarda_semte[0][i] = semte[0][i];
                scanf("%d \n", &semte[1][i]);
                guarda_semte[1][i] = semte[1][i];
                scanf("%d \n", &semte[2][i]);
                guarda_semte[2][i] = semte[2][i];
                scanf ("%d \n", &semte[3][i + no_conexoes]);
                guarda_semte[3][i + no_conexoes] = semte[3][i + no_conexoes];
        }
}

sintaxe( ) {
}

/* Rotina que simula a Entidade Processo de Aplicacao */
aplicacao(ap_even)
struct evento *ap_even; {
        struct trans_arquivo *q;
        struct pac_aceito *p;
        struct pacote *w;

        debug(APLICACAO,"evento tipo %s\n",nome_evento[ap_even→e_tipo]);
        switch(ap_even→e_tipo) {
                case APLICACAO_TRANS_ARQUIVO:
                        q = ap_even→e_param ;
                        w = ger_pac_dados(q→apta_estacao);
```

```
                w->p_ult_pac = (int)GEOMETRICA(no_m_pac,SEQGEO, \
                        q->apta_estacao);
                w->p_no_pac = 1;
                w->p_geracao = relogio;
                if(cfee_atual[q->apta_estacao] < cfee_ent[q->apta_estacao]) {
                        esc_p_a_trans(q->apta_estacao,w);
                } else {
                        pac_bloqueado[q->apta_estacao] = w ;
                }
                break;
        case PACOTE_ACEITO:
                p = ap_even->e_param;
                if(p->ap_paceito->p_no_pac == p->ap_paceito->p_ult_pac) {
                        esc_t_arquivo(p->pacto_estacao);
                } else {
                        w = ger_pac_dados(p->pacto_estacao);
                        w->p_no_pac = p->ap_paceito->p_no_pac + 1;
                        w->p_geracao = p->ap_paceito->p_geracao;
                        w->p_ult_pac = p->ap_paceito->p_ult_pac;
                        if(cfee_atual[p->pacto_estacao] <
                                        cfee_ent[p->pacto_estacao]) {
                                esc_p_a_trans(p->pacto_estacao,w);
                        } else {
                            pac_bloqueado[->pacto_estacao] = w;
                        }
                }
                break;
        }
}
```

```
/* Rotina de geracao de pacotes de dados */
struct pacote *ger_pac_dados(estacao)
int estacao;
{
        struct pacote *p;
        p = cria_pacote( );
        p→fonte = estacao;
        p→destino = estacao + no_conexoes;
        p→p_tipo = DADOS;
        p→p_tamanho = 512;
        return(p);
}

/* Rotina que insere, em filas, em ordem ascendente */
f_ascendente(anodo, aatual)
struct pacote *anodo, *aatual;  {
        return(anodo→p_geracao > aatual→p_geracao);
}

/* Rotina que insere, em filas, no fim da fila */
f_fim(anodo, aatual)
struct pacote *anodo, *aatual; {
        return(1);
}

/* Rotina de enfileiramento */
enfileira(estacao, fila, ap_pac)
int estacao;
int fila;
struct pacote *ap_pac;  {
        insere(ap_pac,&frente_fila[estacao][fila],f_fim);
```

```
}

/* Rotina de desenfileiramento */
desenfileira(estacao, fila)
int estacao;
int fila; {
        REM_PRIMEIRO(frente_fila[estacao][fila]);
)

/* Rotina para coleta de estatísticas */
col_estatística(estacao, ap_pac)
int estacao;
struct pacote *ap_pac;
{
        tempo_espera + = relogio - ap_pa→p_geração;
        no_amostras ++;
        no_prim_pac ++;
        debug(ESTATISTICAS, "no_amostras %d\n", no_amostras);
}

/* Rotina que verifica se o controle de fluxo entre estacoes
 * tem recursos para transmitir pacotes */
cf_ok(estacao)
int estacao; {
        debug(CONTROLE_DE_FLUXO,"recursos de CFEE usados pela estacao \
%d %d\n", estacao, cfee_atual[estacao]);
        if(frente_fila[estacao][FILAPROCTRANS] ! =NULL) {
                if(cfee_atual[estacao] < cfee_ent[estacao]) {
                        cfee_atual[estacao] ++;
                        return(SIM);
                } else {
```

```
                return(NAO);
            }
        } else {
            return(NAO);
        }
}

/* Rotina que atualiza os recursos do controle de fluxo entre estacoes */
cf_atualiza(estacao)
int estacao; {
        cfee_atual[estacao] --;
        esc_pode_t_acesso(estacao);
}

/* Rotina de inicializacao das variáveis globais */
inicializacao( )
{
        int i;
        relogio = 0;
        ack = 0;
        no_amostras = 0;
        tempo_espera = 0;
        tempo_servidor_ocupado = 0;
        nada_a_trans = SIM;
        no_prim_pac = 0;
        for(i = 0; i < no_conexoes; i++)  {
            sinal[i] = NAO;
            frente_fila[i][FILAACMEIO] = NULL;
            frente_fila[i][FILAPROCTRANS] = NULL;
            frente_fila[i][FILAPROCAPLIC] = NULL;
            frente_fila[i + no_conexoes][FILAPROCTRANS] = NULL;
```

```
                    /* Para teste */
                /* cfae_atual[i] = 0;
                cfae_ent[i] = 1;
                cfee_ent[i] = 1; */

                cfee_atual[i] = 0;
                aces_bloq[i] = NAO;
                pac_bloqueado[i] = NÃO;
                pode_retrans[i + no_conexoes] = SIM;
                esc_t_arquivo(i);
        }
}

/* Rotina de debug */
debug (nível, s)
int     nível;
char    *s;
{
        if(!tabdebug[nível].d_flag) return;
        printf("%s: ",tabdebug[nivel].d_nome);
        printf("%r", &s);
}

/* Rotina que duplica o pacote na entidade meio de transmissao */
struct pacote *duplica_pacote(ap_pac)
struct pacote *ap_pac; {
        struct pacote *p;
        p = cria_pacote( );
        p→fonte = ap_pac→fonte;
        p→destino = ap_pac→destino;
        p→p_tipo = ap_pac→p_tipo;
```

```
        p→p_tamanho = ap_pac→p_tamanho;
        p→p_geracao = ap_pac→p_geracao;
        p→p_no_pac = ap_pac→p_no_pac;
        p→p_ult_pac = ap_pac→p_ult_pac;
        debug(MEIO, "duplicou pacote na estacao %d\n",p→fonte);
        return(p);
}

/* Rotina que remove um nodo de uma lista duplamente encadeada */
remove(anodo, acab)
struct cab_lista *anodo;
struct cab_lista **acab;
{
        if((*acab) == NULL) {   /* lista vazia */
                perror ("Remove encontrou a lista vazia \n");
                exit(0);
        }
        anodo→tras→frente = anodo→frente;
        anodo→frente→tras = anodo→tras;
        if(*acab == anodo) {   /* remover o primeiro da lista */
                *acab = anodo→frente;
        }
        if(*acab == anodo) {   /* lista ficou vazia */
                *acab = NULL;
        }
}
```

```
l_inicio(anodo, aatual)  /* inserir no inicio da lista */
struct cab_lista *anodo;
struct cab_lista *aatual;
{
        return(0) ;
}

l_ascendente(anodo, aatual)   /* inserir em ordem ascendente */
struct evento  *anodo, *aatual;
{
        return(anodo→e_t_ocorr  >= aatual→e_t_ocorr);
}

/* Rotina que insere um nodo em uma lista duplamente encadeada */
insere(anodo,acab,comp)
struct cab_lista  *anodo;
struct cab_lista  **acab;
int  (*comp)( );
{
        struct cab_lista *an;
        an = *acab;
        if(an == NULL) {   /* Lista vazia */
                anodo→frente = anodo→tras = *acab = anodo;
                return;
        }
        while((*comp)(anodo,an) && (an = an→frente) != *acab)
                ;
        anodo→frente = an;
        anodo→tras = an→tras;
        an→tras→frente = anodo;
        an→tras = anodo;
```

```
        /* Seta cabecalho */
        if((*comp)(anodo,*acab) == 0) {
                *acab = anodo;   /* anodo na frente */
        }
}

/* Rotina Principal */
main(argc,argv)
int     argc;
char    *argv[ ];
{
        leitura_dados(argc,argv);
        inicializacao( );
        scheduler( );
        resultados( );
        exit(0);
}

pac_transmitido(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        esc_p_liberado(estacao);
        if(!ack) {
                cf_atualiza(estacao);
        }
        free(ap_pac);
}
```

```
/* Rotina que gera um número pseudo_aleatorio */
int random(no_seq, estacao)
int no_seq;
int estacao;
{
        srand(semte[no_seq][estacao]);
        semte[no_seq][estacao] = rand( );
        return(semte[no_seq][estacao]);
}

/* Rotina que dispara os eventos da lista duplamente encadeada */
scheduler( )
{
        struct evento *ap_evento;
                        /* Nao eh fim de simulacao */
        while(no_amostras < no_arquivos) {
                if(cabeca_lista == NULL) {
                        perror("Scheduler NAO achou evento na lista\n");
                        exit(0);
                }
                ap_evento = cabeca_lista;
                REM_PRIMEIRO(cabeca_lista);
                relogio = ap_evento→e_t_ocorr;
                debug(SCHEDULER, "tempo %g, evento tipo %s\n", relogio,
                        nome_evento[ap_evento→e_tipo]);
                (*ap_evento→e_func)(ap_evento);
                free(ap_evento→e_param);
                free(ap_evento);
        }
}
```

```
/* Rotina que gera um pacote ack (desligamos para estes estudos de caso) */
/* gera_ack(fonte,destino)
int fonte;
int destino; {
        struct pacote *p;
        p = cria_pacote( );
        p→fonte = fonte;
        p→destino = destino;
        p→p_tipo = ACK;
        p→p_tamanho = 64;
        p→p_geracao = relógio;
        esc_tx_sem_cfae(p,fonte);
        return(p);
} */

/* Rotina que simula a entidade processamento de transmissao */
proc_trans(ap_even)
struct evento *ap_even; {
        struct pode_t_acesso *q;
        struct tx_sem_cfae    *p;
        struct pac_liberado   *w;
        struct pac_a_trans    *z;
        debug(TRANSMISSAO, "evento tipo %s\n",
                nome_evento[ap_even→e_tipo]);
        switch(ap_even→e_tipo) {
                /* Esta parte trabalha com a transmissao propriamente dita */
                case PODE_TENTAR_ACESSO:
                        q = ap_even→e_param;
                        if(cf_ok(q→pta_estacao)) {
                                esc_pedido_aces(q→pta_estacao,
                                        frente_fila[q→pta_estacao][FILAPROCTRANS]);
```

```
            desenfileira(q→pta_estacao,FILAPROCTRANS);
        }
        break;
case TX_SEM_CFAE:
        p = ap_even→e_param;
        /* enfileira(p→tscfae_estacao,FILAPROCTRANS,
            p→ap_tscfae);
        if(cf_ok(p→tscfae_estacao)) {
            esc_pedido_aces(p→tscfae_estacao,
                frente_fila[p→tscfae_estacao][FILAPROCTRANS]);
            desenfileira(p→tscfae_estacao,FILAPROCTRANS);
        } */
        cf_atualiza(p→tscfae_estacao);
        if(pode_retrans[p→ap_tscfae→destino] == NAO) {
            if(frente_fila[p→ap_tscfae→destino][FILAPROCTRANS]
                == NULL) {
                pode_retrans [p→ap_tscfae→destino] = SIM;
            } else {
                esc_tx_sem_cfae(p→tscfae_estacao,frente_fila\
                    [p→ap_tscfae→destino][FILAPROCTRANS]);
                desenfileira(p→ap_tscfae→destino,
                    FILAPROCTRANS);
            }
        }
        if(p→ap_tscfae→p_no_pac == p→ap_tscfae→p_ult_pac) {
            col_estatistica(p→tscfae_estacao,p→ap_tscfase);
        }
        if(pac_bloqueado[p→tscfae_estacao] != NULL) {
            esc_p_a_trans(p→tscfae_estacao,
                pac_bloqueado[p→tscfae_estacao]);
        }
```

```
                free(p→ap_tscfae);
                break;
                        /* Esta parte trabalha com o controle de fluxo de acesso
                         * a estacao (foi desativada para os estudos de caso */
        /* case PACOTE_LIBERADO:
                w = ap_even→e_param;
                if(cfae_db(w→pl_estacao)) {
                        enfileira(w→pl _estacao,FILAPROCTRANS,
                                frente_fila[w→pl_estacao][FILAPROCAPLIC]);
                        esc_pac_aceito (w→pl_estacao,
                                frente_fila[w→pl_estacao][FILAPROCAPLIC]);
                        pac_bloqueado[w→pl_estacao] = NAO;
                        desenfileira (w→pl_estacao,FILAPROCAPLIC);
                }
                break; */
        case PACOTE_A_TRANSMITIR:
                z = ap_even→e_ param;
                        /* Comandos desativados para teste */
                /* if(!cfae_b(z→pat_estacao, z→ap_patrans)) { */
                        enfileira(z→pat_estacao,FILAPROCTRANS,
                                z→ap_patrans);
                        if(cf_ok(z→pat_estacao)) {
                                esc_pedido_aces(z→pat_estacao,
                                        frente_fila[z→pat_estacao]
                                                [FILAPROCTRANS]);
                                desenfileira(z→pat_estacao,FILAPROCTRANS);
                        }
                        esc_pac_aceito (z→pat_estacao,z→ap_patrans);
                /*  } */
                break;
        }
}
```

```
/* Rotina que simula a entidade processamento de recebimento de pacote */
proc_recepcao(ap_ev)
struc evento *ap_ev; {
/*      struct pacote *p; */
        struct rec_pac *q;
        debug(RECEPCAO, "evento tipo %s\n", nome_evento[ap_ev→e_tipo]);
        q = ap_ev→e_param;
        switch(q→ap_recpac→p_tipo) {
                case DADOS:
                        /* substitui a rotina proc_dados */
                        /* if(ack)    {
                                p = gera_ack(q→ap_recpac→destino,
                                        q→ap_recpac→fonte);
                                esc_tx_sem_cfae(p→fonte,p);
                        } */
                        enfileira(q→rp_estacao,FILAPROCTRANS,q→ap_recpac);
                        tempo_servidor_ocupado += t_transmissao;
                        if(pode_retrans[q→rp_estacao]) {
                                esc_tx_sem_cfae(q→ap_recpac→fonte,
                                        frente_fila[q→rp_estacao][FILAPROCTRANS]);
                                desenfileira (q→rp_estacao,FILAPROCTRANS);
                                pode_retrans[q→rp_estacao] = NAO;
                        }
                        /*  free(q→ap_recpac); */
                        break;
                /* case ACK:
                        cf_atualiza(q→rp_estacao);
                        free(q→ap_recpac);
                        break;   */
        }
}
```

```
/* Rotina que simula a entidade acesso ao meio independente */
ac_m_ind(ap_ev)
struct evento *ap_ev; {
        struct prim_foi  *p;
        struct pedido_aces  *q;
        debug(ACESSO_AO_MEIO,"evento tipo %s\n",
                nome_evento[ap_ev→e_tipo]);
        switch(ap_ev→e_tipo) {
                case PEDIDO_DE_ACESSO:
                        q = ap_ev→e_param;
                        enfileira (q→pa_estacao,FILAACMEIO,q→ap_pacess);
                        if(aces_bloq[q→pa_estacao] == NAO) {
                                esc_tem_pac_a_tx(q→pa_estacao,
                                        frente_fila[q→pa_estacao][FILAACMEIO]);
                                aces_bloq[q→pa_estacao] = SIM;
                        }
                        break;
                case PRIMEIRO_FOI:
                        p = ap_ev→e_param;
                        desenfileira(p→pf_estacao,FILAACMEIO);
                                /* para teste */
                        /* pac_transmitido(p→pf_estacao,p→ ap_p_foi); */
                        aces_bloq[p→pf_estacao] = NÃO;
                        if(frente_fila[p→pf_estacao][FILAACMEIO] != NULL) {
                                esc_tem_pac_a_tx(p→pf_estacao,
                                        frente_fila[p→pf_estacao][FILAACMEIO]);
                                aces_bloq[p→pf_estacao] = SIM;
                        }
                        free(p→ap_p_foi);
                        break;
        }
```

```
}

/* Rotina de escalonamento do evento PRIMEIRO_FOI */
esc_prim_foi(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct prim_foi *q;
        p = cria_evento( );
        q = cria_prim_foi( );
        p→e_tipo = PRIMEIRO_FOI;
        p→e_t_ocorr = relógio;
        p→e_func = ac_m_ind;
        p→e_param = q;
        q→pf_estacao = estacao;
        q→ap_p_foi = ap_pac;
        insere(p, &cabeca_lista, l_ascendente);
}

/* Rotina de escalonamento do evento TEM_PACOTE_A_TRANSMITIR */
esc_tem_pac_a_tx(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct tem_pac_a_tx *q;
        p = cria_evento( );
        q = cria_ tem_pac_a_tx( );
        p→e_tipo = TEM_PACOTE_A_TRANSMITIR;
        p→e_t_ocorr = relógio;
        p→e_func = ac_m_d_an_fi;
        p→e_param = q;
```

```
        q→tpatx_estacao = estacao;
        q→ap_tpatx = ap_pac;
        insere(p, &cabeca_lista, l_ascendente);
}

/* Rotina de escalonamento do evento PEDIDO_DE_ACESSO */
esc_pedido_acess(estacao,ap_pac)
int estacao;
struct pacote *ap_pac;  {
        struct evento *p;
        struct pedido_aces *q;
        p = cria_evento( );
        q = cria_ pedido_aces( );
        p→e_tipo = PEDIDO_DE_ACESSO;
        p→e_t_ocorr = relógio;
        p→e_func = ac_m_ind;
        p→e_param = q;
        q→pa_estacao = estacao;
        q→ap_pacess = ap_pac;
        insere(p, &cabeca_lista, l_ascendente);
}

/* Rotina de escalonamento do evento GANHA_ACESSO */
esc_g_acesso(estacao)
int estacao; {
        struct evento *p;
        struct g_acesso *q;
        p = cria_evento( );
        q = cria_ g_acesso( );
        p→e_tipo = GANHA_ACESSO;
        p→e_t_ocorr = relógio;
```

```
        p→e_func = meio_de_trans;
        p→e_param = q;
        q→ga_estacao = estacao;
        q→ap_g_acesso = frente_fila[estacao][FILAACMEIO];
        insere(p, &cabeca_lista, l_ascendente);
}

/* Rotina que simula a entidade acesso ao meio dependente
 * em anel com passagem de ficha */
ac_m_d_an_fi(ap_even)
struc evento *ap_even; {
        struct tem_pac_a_tx  *p;
        struct fim_trans        *q;
        debug(ACESSO_AO_MEIO,"evento tipo %s\n",
                        nome_evento[ap_even→e_tipo]);
        switch(ap_even→e_tipo)  {
                case FIM_DE_TRANSMISSAO:
                        q = ap_even→e_param;
                        pass_ficha( );
                        esc_prim_foi(q→ft_estacao, q→ap_fim_trans);
                        break;
                case TEM_PACOTE_A_TRANSMITIR:
                        p = ap_even→e_param;
                        sinal[p→tpatx_estacao] = SIM;
                        if(nada_a_trans == SIM)  {
                                nada_a_trans == NÃO;
                                pass_ficha( );
                        }
                        break;
        }
}
```

```
/* Rotina que passa a ficha */
pass_ficha( ) {
        static int ficha = 0;
        int     i;
        for(ficha = (ficha + 1) % no_conexões, i = 0;
                sinal[ficha] == NAO && i < no_conexões;
                        ficha = (ficha + 1) % no_conexoes, i++)
                        ;
        if(sinal[ficha]  == NAO) {
                nada_a_trans = SIM;
                return;
        } else {
                esc_g_acesso(ficha);
                sinal[ficha] = NAO;
        }
}

/* Rotina que aloca espaço na memória e imprime
 * mensagem de falta da mesma */
charm   *mem_alloc(num,tamanho,msg)
int     num;
int     tamanho;
char    *msg;
{
        char    *q;
        if((q = calloc(num,tamanho)) == NULL) {
                perror("Falta memoria pra criar %s\n",msg);
        } else {
                return(q);
        }
}
```

```
/* Rotina  que cria a estrutura de dados pacote  */
struct pacote *cria_pacote( ) {
        return((struct pacote *)mem_alloc(1,sizeof(struct pacote),"pacote"));
}

/* Rotina  que cria a estrutura de dados evento  */
struct evento *cria_evento( ) {
        return((struct evento *)mem_alloc(1,sizeof(struct evento),"evento"));
}

/* Rotina  que cria a estrutura de dados g_acesso  */
struct g_acesso  *cria_g_acesso( ) {
        return((struct g_acesso *)mem_alloc(1,sizeof(struct g_acesso),"g_acesso"));
}

/* Rotina  que cria a estrutura de dados fim_trans  */
struct fim_trans  *cria_fim_trans( ) {
        return((struct fim_trans  *)mem_alloc(1,sizeof(struct fim_trans),"fim_trans"));
}

/* Rotina  que cria a estrutura de dados rec_pac  */
struct rec_pac  *cria_rec_pac( ) {
        return((struct rec_pac  *)mem_alloc(1,sizeof(struct rec_pac),"rec_pac"));
}

/* Rotina  que cria a estrutura de dados prim_foi  */
struct prim_foi *cria_prim_foi( ) {
        return((struct prim_foi *)mem_alloc(1,sizeof(struct prim_foi), "prim_foi"));
}
```

```
/* Rotina  que cria a estrutura de dados tem_pac_a_tx  */
struct tem_pac_a_tx *cria_tem_pac_a_tx( ) {
        return((struct tem_pac_a_tx *)mem_alloc(1,sizeof(struct tem_pac_a_tx),
                "tem_pac_a_tx"));
}

/* Rotina  que cria a estrutura de dados pedido_aces  */
struct pedido_aces *cria_pedido_aces( ) {
        return((struct pedido_aces *)mem_alloc(1,sizeof(struct pedido_aces),
                "pedido_aces"));
}

/* Rotina  que cria a estrutura de dados pode_t_acesso  */
struct pode_t_acesso  *cria_pode_t_acesso( ) {
        return((struct pode_t_acesso *)mem_alloc(1,sizeof(struct pode_t_acesso),
                "pode_t_acesso"));
}

/* Rotina  que cria a estrutura de dados tx_sem_cfae  */
struct tx_sem_cfae  *cria_tx_sem_cfae( ) {
        return((struct tx_sem_cfae *)mem_alloc(1,sizeof(struct tx_sem_cfae),
                "tx_sem_cfae"));
}

/* Rotina  que cria a estrutura de dados pac_liberado  */
struct pac_liberado  *cria_pliberado( ) {
        return((struct pac_liberado *)mem_alloc(1,sizeof(struct pac_liberado),
                "pac_liberado"));
}
```

```
/* Rotina  que cria a estrutura de dados pac_a_trans  */
struct pac_a_trans  *cria_p_a_trans( ) {
        return((struct pac_a_trans *)mem_alloc(1,sizeof(struct pac_a_trans),
                "pac_a_trans"));
}

/* Rotina  que cria a estrutura de dados pac_aceito  */
struct pac_aceito *cria_aceito( ) {
        return((struct pac_aceito *)mem_alloc(1,sizeof(struct pac_aceito),
                "pac_aceito"));
}

/* Rotina que cria a estrutura de dados trans_arquivo */
struct trans_arquivo *cria_t_arquivo( )  {
        return((struct trans_arquivo *)mem_alloc(1,sizeof(struct trans_arquivo),
                "trans_arquivo"));
}

/* Rotina para escalonar evento RECEPÇÃO_DE_PACOTE */
esc_rec_pac(ap_pac)
struct pacote *ap_pac;  {
        struct evento *p;
        struct rec_pac *q;
        p = cria_evento( );
        q = cria_rec_pac( );
        p→e_tipo = REC_PAC;
        p→e_t_ocorr = relógio + t_transmisssao;
        p→e_func = proc_recepcao;
        p→e_param = q;
        q→rp_estacao = ap_pac→destino;
        q→ap_recpac = ap_pac;
```

```
        insere(p, &cabeca_lista, l_ascendente);
}

/* Rotina para escalonar o evento FIM_DE_TRANSMISSAO */
esc_fim_trans(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct fim_trans *q;
        p = cria_evento( );
        q = cria_fim_trans( );
        p→e_tipo = FIM_DE_TRANSMISSAO;
        p→e_t_ocorr = relógio + t_transmissao;
        p→e_func = ac_m_d_an_fi;
        p→e_param = q;
        q→ft_estacao = estacao;
        q→ap_fim_trans = ap_pac;
        insere(p, &cabeca_lista, l_ascendente);
}


/* Rotina de escalonamento do evento PACOTE_A_TRANSMITIR */
esc_p_a_trans(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct pac_a_trans *q;
        p = cria_evento( );
        q = cria_p_a_trans( );
        p→e_tipo = PACOTE_A_TRANSMITIR;
```

```
            /* para teste */
        p→e_t_ocorr = relógio +
            (tempo_t)(EXPONENCIAL(t_fonte, SEQCONSF, estacao));
        p→e_func = proc_trans;
        p→e_param = q;
        q→pat_estacao = estacao;
        q→ap_patrans = ap_pac;
        insere(p, &cabeca_lista, l_ascendente);
}

/* Escalonamento do evento APLICAÇÃO_TRANS_ARQUIVO */
esc_t_arquivo(estacao)
int estacao;
        struct evento *p;
        struct trans_arquivo *q;
        p = cria_evento( );
        q = cria_t_arquivo( );
        p→e_tipo = APLICAÇÃO_TRANS_ARQUIVO;
        p→e_t_ocorr = relógio +
            (tempo_t)(EXPONENCIAL(t_m_ativ_apli,SEQATIV,estacao));
        p→e_func = aplicacao;
        p→e_param = q;
        q→apta_estacao = estacao;
        insere(p, &cabeca_lista, l_ascendente);
}

/* Escalonamento do evento PACOTE_LIBERADO */
esc_p_liberado(estacao)
int estacao; {
        struct evento *p;
        struct pac_liberado *q;
```

```
        p = cria_evento( );
        q = cria_pliberado( );
        p→e_tipo = PACOTE_LIBERADO;
        p→e_t_ocorr = relógio;
        p→e_func = proc_trans;
        p→e_param = q;
        q→pl_estacao = estacao;
        insere(p, &cabeca_lista, l_ascendente);
}

/* Escalonamento do evento PODE_TENTAR_ACESSO */
esc_pode_t_acesso(estacao)
int estacao;
        struct evento *p;
        struct pode_t_acesso *q;
        p = cria_evento( );
        q = cria_pode_t_acesso( );
        p→e_tipo = PODE_TENTAR_ACESSO;
        p→e_t_ocorr = relógio;
        p→e_func = proc_trans;
        p→e_param = q;
        q→pta_estacao = estacao;
        insere(p, &cabeca_lista, l_ascendente);
}

/* Escalonamento do evento TX_SEM_CFAE */
esc_tx_sem_cfae(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct tx_sem_cfae *q;
```

```
        p = cria_evento( );
        q = cria_cria_tx_sem_cfae( );
        p→e_tipo = TX_SEM_CFAE;
                /* Para teste */
        p→e_t_ocorr = relógio +
                (tempo_t)(EXPONENCIAL(t_sorvedouro,SEQCONSR,estacao));
        p→e_func = proc_trans;
        p→e_param = q;
        q→tscfae_estacao = estacao;
        q→ap_tscfae = ap_pac;
        insere(p, &cabeca_lista, l_ascendente);
}

/* Escalonamento do evento PACOTE_ACEITO */
esc_pac_aceito(estacao,ap_pac)
int estacao;
struct pacote *ap_pac; {
        struct evento *p;
        struct pac_aceito *q;
        p = cria_evento( );
        q = cria_aceito( );
        p→e_tipo = PACOTE_ACEITO;
        p→e_t_ocorr = relógio;
        p→e_func = aplicacao;
        p→e_param = q;
        q→pacto_estacao = estacao;
        q→ap_paceito = ap_pac;
        insere(p, &cabeca_lista, l_ascendente);
}
```

```
/* Rotina que atende ao evento GANHA_ACESSO.
 * Modelando, portanto, a entidade meio de transmissão */
meio_de_trans(ap_ev)
struct pacote *ap_ev;
{
        struct pacote *p;
        struct g_acesso *q;
        q = ap_ev→e_param;
        debug(ACESSO_AO_MEIO,"a estacao %d ganhou acesso",
                q→ga_estacao);
        p = duplica_pacote(q→ap_g_acesso);
        esc_rec_pac(p);
        esc_fim_trans(q→ga_estacao, q→ap_g_acesso);
}


/* Rotina que imprime os resultados da Simulação */
resultados( ) {
        int  i;
        printf("Resultados do Pacote Simulador: Estudo de caso\n");
        printf("da Transferencia de Arquivos\n");
        printf("\n");
        printf("ENTRADAS:\n");
        printf("Tempo Medio de Geracao de Pacotes = %ld\n",t_fonte);
        printf("Tempo Medio de Absorção de Pacotes = %ld\n",t_sorvedouro);
        printf("Tempo Medio de Ativação da Aplicação = %ld\n",t_m_ativ_aplic);
        printf("Tempo de Transmissao no Anel = %g\n",t_transmissao);
        printf("Tamanho da janela = %d\n", tam_janela);
        printf("Numero de conexões = %d\n",no_conexões);
        printf("Numero medio de pacotes por arquivo = %d\n",no_m_pac);
        printf("Numero de Arquivos Transmitidos = %d\n",no_amostras);
```

```
        for(i = 0;  i  <  no_conexões; i++) {
                printf("Semente do gerador de pacotes da estacao[%d] = %d\n",
                        i,guarda_semte[ 0 ][ i ]);
                printf("Semente do gerador de numero de pacotes da estacao[%d] = \
                        %d\n",i,guarda_semte[ 1 ][ i ]);
                printf("Semente do ativador da estacao[%d] = %d\n",
                        i,guarda_semte[ 2 ][ i ]);
                printf("Semente do sorvedouro de pacotes da estacao[%d] = %d\n",
                        i + no_conexoes, guarda_semte[ 3 ][ i + no_conexoes]);
}
printf("\n");
printf("RESULTADOS:\n");
printf("Tempo Medio de Resposta para Transferencia de Arquivos = %g\n",
        (double)tempo_espera/no_amostras);
printf("Utilizacao do Meio de Transmissao = %g\n",
        (double)tempo_servidor_ocupado/relogio);
printf("Numero de primeiros pacotes = %d\n", no_prim_pac);
printf("Vazao por Conexao = %g\n",
        (double)no_amostras/tempo_espera);
}
```

APÊNDICE C

Sementes usadas no segundo estudo de caso e terceiro.

Note que o simulador lê uma semente por linha, portanto a ordem em que as sementes estão neste apêndice é importante.

6427
7349
8263
9161
10069
5573
4721
3863
3041
2267
1487
1489
769
773
151
9739
9743
7001
7043
7109
7121
6143
6151
6199

6203
6211
6229
6247
6257
6271
6277
6287
6301
6317
6337
5501
5521
5531
5569
5581
5639
6481
6521
6547
6551
6563
6571
4789
4801
4813
4817
4831
21391
21397
21401
21407
21419
547

557
563
577
593
601
10663
10667
10709
10753
10771
10781
10799
10831
10847
10859
17623
17657
17659
17669
17737
17747
17749
17783
17839
21803
21817
21821
21839
21841
37
9739
13
641
643

49
701
719
733
47
3137
39
3163
3167
3169
3181
15817
15823
15859
15877
15881
31
1
3
5
11
21851
21859
21863
21871
21881
51
53

REFERÊNCIAS BIBLIOGRÁFICAS

[BUX 81] Bux, W., "Local Area Subnetworks: A Performance Comparison", IEEE Trans. on Commun., vol. COM-29, October, 1981.

[CAUP 86] Cauper, A. F., Sauvé, J. P. e Moura, J. A. B., "Um Pacote de Simulação para Avaliação de Desempenho de Redes Locais", Anais do IV SBRC, Recife, Abril, 1986.

[CHAN 78] Chandy, K. Mani e Yek, Raymund T., "Current Trends in Programming Methodology – Software Modelling", vol. 3, Prentice_Hall, Inc., Englewood Cliffs, Bew Hersey, 07632, 1978.

[GIOZ 85] Giozza, W. F., Araújo, J. F. M., Moura, J. A. B., e Sauvé, J. P., "Tendências de Padronização da Sub_Rede de Comunicação em Redes Locais de Computadores", GRC/UFPB, Relatório Técnico número RT_02/85, Fevereiro, 1985.

[GIOZ 86] Giozza, W. F., Araújo, J. F. M., Moura, J. A. B. e Sauvé, J. P., "Redes Locais de Computadores – Tecnologia e Aplicações", McGraw_Hill, Ltda., São Paulo, 1986.

[IEEE 802] IEEE Study Group on Local Área Network; Project 802.

[MOUR 79] Moura, J. A. B., "Loops and Ethernet: Evaluation and Comparison of Performance and Complexity", M. A. Sc. Thesis, Dept. of Electrical Engineering, University of Waterloo. Waterloo, Ontário, Canadá, 1979.

[MOUR 82] Moura, J. A. B., "Hierarchical Modelling of Local Area Networks under File Transfer". Phd. Thesis, Dept. Of Electrical Engineering, University of Waterloo. Waterloo, Ontário, Canadá, 1982.

[MOUR 83] Moura, J. A. B. E Sauvé, J. P., "Avaliação de Desempenho de Redes de Dados Locais", Anais do I SBRC, Porto Alegre, Maio, 1983.

[MOUR 83/84] Moura, J. A B. e Sauvé, J. P., "Análise Assintótica do Desempenho de Redes de Dados Locais", RBC, Rio de Janeiro, vol. 3, número 2, 1983/1984.

[MOUR 84] Moura, J. A. B., Sauvé, J. P., Giozza, W. F. e Araújo, J. F. M., "Técnicas de Simulação Digital", GRC/UFPB, Relatório Técnico número RT-18/84, dezembro, 1984.

[MOUR 85] Moura, J. A. B., Sauvé, J. P., Giozza, W. F. e Araújo, J. F. M., "Avaliação de Desempenho de Protocolos de Transporte", GRC/UFPB, Relatório Técnico número 09/85, 1985.

[MOUR 86] Moura, J. A. B., Sauvé, J. P., Giozza, W. F. e Araújo, J. F. M., "Redes Locais de Computadores – Protocolos de Alto Nível e Avaliação de Desempenho", McGraw_Hill, Ltda., São Paulo, 1986.

[SAGE 78] Sager, G. R. e Wong, J. W., "A First Course in Simulation", CS-78-03, Computer Science, University of Waterloo, Ontário, Canadá, 1978.

[TROP 81] Tropper, C., "Local Computer Network Tecnologies", Academic Press, Inc., Nova Iorque, 1981.

[WONG 83] Wong, J. W., Moura, J. A. B. e Field, J. A., "Hierarchical Modelling of File Transfers on Local Area Networks", Comput. & Elect. Engng., vol. 10, number 3, 1983, pp. 191-207.