Este exemplar corresponde à redação final da tese defendida por Maria Cristina Emiko Ussami e aprovada pela Comissão Julgadora em 27/05/93.

Orientador

Implementação e Análise de Desempenho de um Protocolo de Comunicação na Rede de Serviços Integrados RALFO

Maria Cristina Emiko Ussami

Orientador: Prof. Dr. Shusaburo Motoyama

Dissertação apresentada à Faculdade de Engenharia Elétrica da UNICAMP, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica

Maio/1993

Aos meus avós, Otikichi Marui e Shozaburo Ussami,
que atravessaram o mundo
e lutaram até o fim de suas vidas
em busca de seus ideais,
sempre com muita dedicação e lealdade aos seus princípios.

Agradecimentos

Agradeço a várias pessoas que viabilizaram, de muitas maneiras, este trabalho:

- Prof. Shusaburo Motoyama, pela orientação e oportunidade de trabalho;

- Prof. Akebo Yamakami e colegas do projeto que muito se empenharam para viabilizar a RALFO: Gorgônio Araújo, Gerson Saito, Paulo Pessoa e Cláudia Carneiro;

- Srs. Antônio Palma, Márcio Machado e Gustavo Chaves, do projeto CHILL, (CPqD/Telebrás), pelas orientações, consultas e fornecimento do compilador CHILL e suas ferramentas de apoio;

- Srs. José H. Zibelberg, Arthur, José Roberto, Arlindo e Victor Valenzuela (CPqD/Telebrás), pelo auxílio ao projeto;

- Srs. Sálvio Sobrinho e Jurandyr (CPqD/Telebrás), pelo fornecimento do SOP e todo apoio dado ao grupo;

- Amaury pelas sugestões sensacionais no uso dos pacotes SLAM e RESQ;

- Miguel Roszas, pela Administração dos equipamentos no laboratório do Departamento de Telemática, FEE, Unicamp;

- Srs. Marçal dos Santos e Osmaira Raeder do Centro de Computação da Unicamp por permitirem e incentivarem este trabalho;

- Meus amigos do Centro de Computação, pela amizade;

- Samira Telles, Hugo Lavalle, Sandra Shibuya, Sueli Turati, pelo apoio.

E outras pessoas que viabilizaram não só este trabalho, mas todos os passos que tenho dado até agora:

- Marcos, pelo seu amor e companheirismo que me acompanharam a cada minuto, cada letra deste trabalho; e a toda sua família que me aconchegou com muito carinho;

- Meus pais, pelo seus exemplos de determinação, por todos os seus imensuráveis esforços na criação e educação de seus filhos e por estarem ao meu lado em todos os momentos da minha vida;

- Minhas queridas irmãs, pelo grande incentivo, acreditando sempre em mim e em tudo que fiz;

- Meus tios Mário, Jorge e Emília, pelos seus ideais e por dedicarem grande parte de suas vidas ao amor às suas sobrinhas;

- Minhas famílias Marui e Ussami, pelo apoio;

- Todos os grandes amigos que consegui ao longo destes anos, em especial, Luciana Arantes e Vera Aoki;

- Cynthia, por sua orientação e incentivo, cujos resultados me fizeram chegar ao fim deste trabalho.

Sumário

O propósito deste trabalho é pesquisar os aspectos relacionados à implementação e avaliação de protocolos de comunicação, definindo um ambiente de desenvolvimento para a implementação de protocolos na rede RALFO, e um modelo para análise de desempenho. A RALFO (Rede de Área Local com Fibras Ópticas) é uma proposta de rede com integração de serviços de voz e dados, em desenvolvimento no Departamento de Telemática da Faculdade de Engenharia Elétrica da Unicamp, que utiliza topologia em anel e fibras ópticas como meio de transmissão.

Para cumprir estes objetivos, inicialmente é apresentado o projeto RALFO em detalhes: arquitetura da rede, hardware dos nós da rede, modelo de camadas. Dentro deste contexto é descrita a proposta do ambiente de desenvolvimento de software definido para a RALFO: a escolha do sistema operacional e da linguagem de programação, SOP e CHILL respectivamente. Detalhados os motivos que justificaram estas escolhas, apresenta-se a implementação da camada LLC, feita para um protótipo de rede, segundo a recomendação IEEE 802.2. Esta camada é responsável pela transmissão correta entre duas entidades conectadas através de um enlace lógico. Para avaliar o protocolo implementado são discutidos os aspectos da elaboração de dois modelos, analítico e de simulação e através deles a análise de desempenho da camada LLC.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A grande disseminação das redes locais de computadores deu origem a necessidade de interligação dessas redes por meio de uma outra rede de velocidade maior (média velocidade), abrangendo áreas geograficamente fechadas (como campus universitário ou indústria de grande porte). Estas redes de média velocidade, por sua vez, ocasionaram a necessidade de suas interconexões através de uma rede de alta velocidade, abrangendo agora áreas de dimensões metropolitanas, para conexões de equipamentos em locais bastante distantes. Estas redes são denominadas redes de áreas metropolitanas (MAN - Metropolitan Area Networks) e estão sendo objetos de padronização através do padrão IEEE 802.6 (Institute of Electrical and Electronics Engineers) e do padrão FDDI (Fiber Distributed Data Interface) do ANSI (American Nacional Standard Institute).

As redes metropolitanas estão sendo preparadas para transportar além de dados em geral, sinais de voz, aumentando sua eficiência permitindo o suporte à integração de serviços. Uma outra característica importante é que elas estão sendo desenvolvidas para ter máxima compatibilização com a RDSI-FL (Rede Digital de Serviços Integrados de Faixa Larga). Considera-se que a RDSI-FL será a rede que integrará definitivamente todas as redes existentes: as redes telefônicas, as LANs, as MANs e todas outras redes atualmente em desenvolvimento.

A rede que está sendo desenvolvida no Departamento de Telemática da Unicamp, denominada RALFO (Rede de Área Local com Fibras Ópticas), não segue as padronizações acima citadas (IEEE 802.6 ou FDDI), mas é bastante flexível para se compatibilizar com a emergente RDSI-FL. em concordância com a tendência internacional. Além disso, pode ser utilizada tanto em rede local como em rede metropolitana.

O propósito desta tese é, uma vez definida a arquitetura da rede RALFO e seu modelo de camadas, definir um ambiente de desenvolvimento para a implementação de protocolos da RALFO, implementar o protocolo LLC, bem como avaliar o desempenho do tráfego de dados na rede.

Com este objetivo, a pesquisa é dirigida através dos tópicos:

- Implementação de protocolos;
- Protocolo LLC (IEEE 802.2);
- Análise de desempenho do protocolo LLC.

Como produtos finais, obtidos neste trabalho, temos para um protótipo da RALFO desenvolvido no Departamento de Telemática, a implementação do protocolo IEEE 802.2 e a análise de desempenho do protocolo na rede.

Esta dissertação está organizada da seguinte maneira: inicialmente apresentamos a questão da integração de serviços e o projeto RALFO, no capítulo 2. A arquitetura e o hardware da rede são tópicos deste capítulo. No capítulo 3, serão discutidos os aspectos gerais relacionados à implementação de protocolos. São definidos o ambiente de desenvolvimento para a implementação dos protocolos da RALFO, bem como os softwares utilizados. O capítulo 4 explora as facilidades da linguagem CHILL para a implementação de protocolos de comunicação. A implementação do protocolo da camada LLC, na RALFO, será apresentada no capítulo 5. Implementado o protocolo, foi feita sua análise de desempenho: foram elaborados modelos de análise e coletados alguns resultados. Estes tópicos são o assunto do capítulo 6. Finalmente, no capítulo 7, seguem as conclusões deste trabalho.

Capítulo 2

Integração de serviços e a rede RALFO

2.1 Introdução

Neste capítulo são apresentados alguns resultados interessantes, encontrados na literatura, que justificam a integração de serviços e orientam como ela deve ser feita. Dentro deste contexto será introduzido o projeto RALFO: seus objetivos, sua arquitetura e seu modelo de camadas.

2.2 Integração de serviços

A disseminação do uso de mini e microcomputadores nos ambientes de escritórios e fábricas fez surgir a necessidade de interligá-los em uma rede de comunicação de dados, visando otimizar o uso destes equipamentos. Geralmente, nestes ambientes, existe uma rede comunicação de voz implementada através de uma central telefônica. A possibilidade de utilizar o mesmo meio físico para o transporte de voz e dados deu início às pesquisas de redes de serviços integrados.

Richer, Steiner e Sengoku investigaram alguns aspectos relacionados à comunicação de dados e voz em redes locais de escritórios como parte dos seus estudos sobre automação de escritórios [Richer 42]. Seus principais objetivos eram concluir se voz e dados deveriam ser tratados pelas mesmas técnicas, e se estes tipos de tráfegos deveriam ser combinados em uma única rede ou deveriam usar redes distintas, porém interconectadas.

Terminal	Taxa Transf. pico (kbps)	Utilização média (erlangs)	Vazão média (kbps)	Num. de term. em escritórios		
				pequeno	médio	grande
telefone	128.0	0.13	8.0	100	500	2000
proc. texto	2.4	0.4	0.048	5	50	200
term. dados	2.4	1.0	0.12	5	50	200
fac-símile	9.6	0.21	2.0	2	10	40
copiadora	48.0	0.036	1.7	2	10	40
impressora	9.6	0.36	3.5	2	10	40

Tabela 2.1: Características dos terminais em escritórios

Baseados em relatórios de pesquisa de mercado, primeiramente eles selecionaram alguns equipamentos que acreditaram ser os terminais que fariam parte desta suposta rede local e analisaram seus comportamentos. São eles: telefones, terminais de dados, processadores de textos, dispositivos de *fac-símile*, impressoras e copiadoras. A tabela 2.1 mostra as características e o número de terminais utilizados em escritórios de diferentes tamanhos, analisados pelos pesquisadores.

Para o cálculo destas taxas foram assumidas as seguintes premissas:

- Cada telefone trata 20 chamadas por dia (8 horas), cada chamada tem uma duração de 3 minutos. Foi considerado que durante todo o chamado telefônico o meio de comunicação é utilizado.
- Cada processador de texto é utilizado 200 minutos por dia. Porém em apenas 5% deste tempo é utilizada a comunicação de dados.
- Cada terminal de dados está ativo o dia todo. O tráfego de dados na rede é otimizado pelo uso de terminais inteligentes, portanto considera-se que cada terminal utiliza apenas 5% do tempo total em comunicação.
- Cada fac-símile produz 100 páginas em 100 minutos por dia, onde 100% deste tempo utiliza-se em comunicação de dados.
- Cada copiadora produz cópias de 1000 originais trafegados pela rede (48kbps). Cada original possui em média 50kbits.
- Cada impressora produz 100 relatórios de 20 páginas (50 kbits) cada um e possui uma taxa de transferência de 9.6 kbps.

Através destes dados, eles concluíram que o tráfego de voz é substancialmente maior que o tráfego de dados. Assim, se uma única rede de comunicação é utilizada para manipular voz e dados, ela deve ser otimizada para o tráfego de voz. Analisando vantagens e desvantagens sobre a utilização de uma rede integrada, estes pesquisadores chegaram às seguintes conclusões:

O uso de redes separadas tem vantagens em três aspectos:

- Redes separadas podem ser otimizadas de acordo com o tráfego de cada uma.
- Voz e dados têm diferentes necessidades para atraso, vazão, controle de erro e confiabilidade.
- No presente não existe muito relacionamento entre os tráfegos de voz e dados e os padrões de fluxo de ambos são diferentes Exemplo de aplicações integradas são correio eletrônico de voz, onde o remetente envia uma mensagem falada que chega ao receptor como uma mensagem escrita; e ditador, onde um ditado verbal possa ser armazenado em forma de dados para posterior transcrição e edição, por secretárias.

O uso de rede integrada tem as seguintes vantagens:

- Em uma rede integrada um cliente pode adquirir um único produto que oferece modularidade: pode-se começar com uma rede de voz e posteriormente adicionar facilidades de dados a um custo menor.
- É possivel aproveitar a instalação telefônica existente em uma construção para manipular o tráfego de dados.
- Como o volume de dados é geralmente bem menor que o de voz, fica barato prover as facilidades de dados em redes integradas.
- A confiabilidade necessária às redes de voz facilitará a manipulação do tráfego de dados.
- No futuro existirá maior relacionamento entre voz e dados e os padrões de fluxo de dados se tornarão parecidos com os da voz.
- O acesso aos dados externos (via redes de longa distância) seria mais direto em uma rede integrada, uma vez que os terminais de voz já possuem este acesso.

Portanto, as vantagens de redes separadas são superadas técnica e economicamente pelas redes integradas. As redes em ambientes de escritórios deverão ser baseadas nesta integração. No entanto, para que ela seja eficiente, a escolha de uma meio de transmissão de alta capacidade e confiabilidade torna-se fundamental.

2.3 Projeto RALFO

O projeto RALFO (Rede de Área Local com Fibras Ópticas), em desenvolvimento no Departamento de Telemática da Faculdade de Engenharia Elétrica, da Universidade Estadual de Campinas, tem como objetivo pesquisar os aspectos relacionados à concepção e implementação de uma rede de área local com integração de serviços de voz e dados utilizando como meio de transmissão fibras ópticas.

O uso de fibras ópticas nesta rede é conveniente pois associa à sua alta capacidade de transmissão, confiabilidade, imunidade às interferências e custo reduzido em futuro próximo.

Os tópicos de pesquisa deste projeto são:

- Arquitetura para a RALFO: Especificação de um modelo de camadas e seus protocolos de comunicação. Validação de protocolos.
- Método de acesso à RALFO: Proposição e avaliação de um método de acesso para a RALFO.
- Transmissão na RALFO: Estudos de esquemas de transmissão e códigos de linha a serem utilizados.
- Protótipo Experimental: Construção de um protótipo experimental para avaliação e verificação da praticabilidade de uma RALFO, bem como o estudo dos aspectos relacionados a implementação de protocolos de comunicação.

2.4 Arquitetura da RALFO

A RALFO apresenta topologia em anel duplo, tendo fibras ópticas como meio de transmissão. Os anéis tem fluxo em sentido inverso, conforme representado na Fig. 2.1.O anel duplo aumenta o grau de tolerância a falhas da rede. No caso de falha em um dos enlaces, faz-se uma reconfiguração da rede através de um caminho

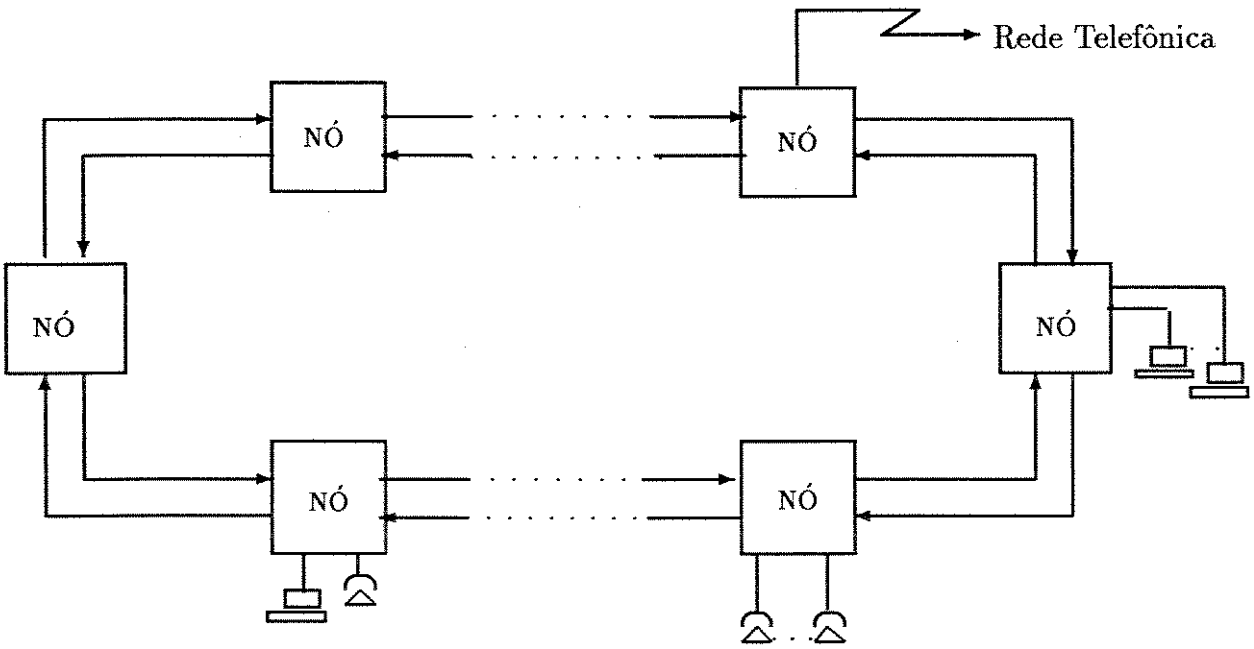


Figura 2.1: Topologia da rede RALFO

alternativo entre os nós onde o enlace se encontra danificado ("loop back"), e para o caso de falha em um dos nós, aplica-se a técnica de isolação do nó ("by pass").

A configuração em anel permite a rede operar tanto em distâncias curtas (rede local) quanto em distâncias alcançando regiões metropolitanas (rede metropolitana), pela introdução de regeneradores em cada nó da rede.

2.4.1 Nó da rede

O equipamento escolhido para funcionar como nó da rede foi o Processador Preferencial (PP), desenvolvido pelo CPqD da Telebrás [PP 41]. O PP utiliza o microprocessador iAPX 80286 da Intel operando no modo protegido. Esta escolha deve-se às facilidades oferecidas pelo PP para adicionar outras placas processadoras bem como a manipulação de seu barramento de dados.

A configuração do PP, adotada para a RALFO, apresenta cinco placas [Pessoa 39]:

- UPN: CPU (Unidade Central de Processamento) do PP, constituída de um microprocessador 286 com co-processador matemático (287), 512 Kbytes de memória RAM, 64 Kbytes de EPROM e duas interfaces seriais RS 232 e uma paralela;

- DIS: Controladora de disco, com capacidade de controlar quatro unidades de disco rígido (tecnologia winchester) de 5,25 polegadas com capacidade formatada até 288 Mbytes cada, quatro unidades de disco flexível de 8 e 5,25 polegadas e duas unidades de fita cartucho;

- COM: Controladora de vídeo, possui um processador próprio (8088), controla duas interfaces seriais RS 232 e uma interface paralela, 128 Kbytes de E-PROM , 192 Kbytes de RAM e um bloco de memória Dual-Port . Através destas memórias é possível um outro processador escrever internamente na placa COM;

- RAS: Placa rastreadora, de grande utilidade no desenvolvimento de projetos, pois através dela é posssível acompanhar o fluxo de informação no barramento do PP, simulando um analizador lógico. Apresenta um processador (8085) com até 32 Kbytes de EPROM e 8 Kbytes de RAM.

- SER: Placa controladora de interfaces seriais. Controla até oito interfaces seriais RS 232.

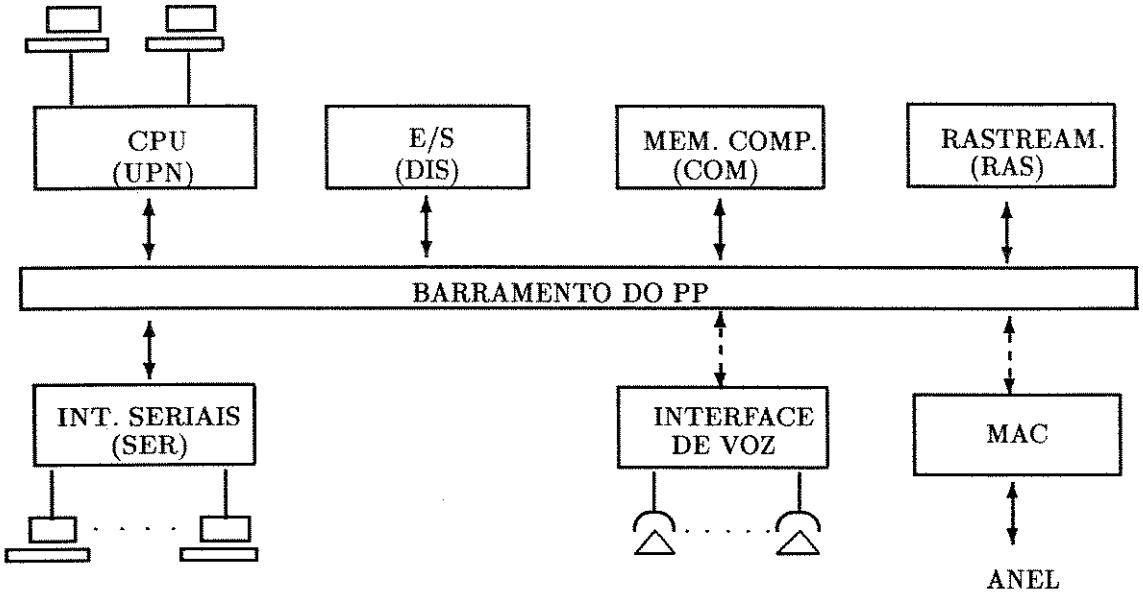


Figura 2.2: Arquitetura de hardware do PP

A esta configuração, serão acopladas a placa MAC e a Interface de Voz, através das quais faz-se o acesso à rede e aos terminais telefônicos respectivamente. Os terminais de dados serão conectados através das linhas seriais existentes nas placas do PP e das linhas da placa SER. Portanto, o nó da rede terá capacidade de conectar 10 terminais de dados e mais 8 terminais telefônicos.

A interface telefônica foi especialmente desenvolvida para a RALFO, como parte do projeto de tese de P. Pessoa [Pessoa 39], realizada no Departamento de Telemática da FEE, UNICAMP.

A comunicação do nó com o meio de transmissão (MAC) será feita através de uma interface de controle de acesso ao meio, em desenvolvimento, como parte do projeto de tese de G. Araújo [Araujo 3], no Departamento de Telemática da FEE, UNICAMP.

Estas placas são baseadas em microprocessadores 80286. Tanto a interface de voz quanto a MAC possuem seus próprios processadores, caracterizando o nó da rede com uma estrutura de multiprocessadores. A arquitetura do nó pode ser representada pela Fig. 2.2.

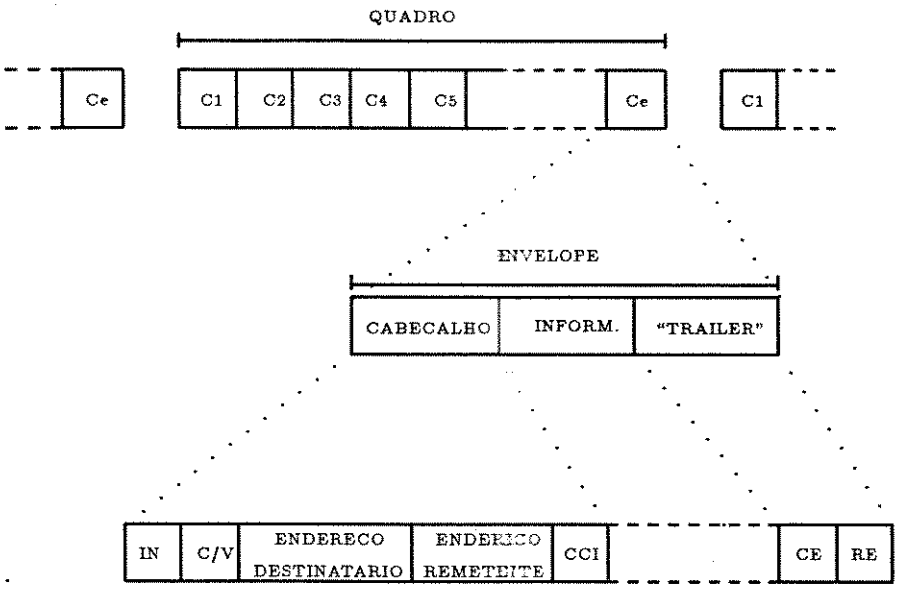


Figura 2.3: Estrutura de quadros e envelopes

2.4.2 Método de acesso e estrutura de quadros

Foi adotada na RALFO, a comutação de pacote tanto para voz como para dados; entretanto os pacotes de voz têm maior prioridade no acesso ao meio. O método de acesso adotado foi o "Empty Slot", modificado para incluir os sinais de voz [Guardiero 18]. A estrutura de quadros está representada na Fig. 2.3.

O primeiro bit IN (INício) é utilizado para o sincronismo de quadros na rede. O bit C/V (Cheio/Vazio) indica o estado do canal. O campo CCI (Conteúdo do Campo de Informação) indica se a informação é voz ou dados. O campo CE (Controle de Erro) é utilizado para garantir a confiabilidade da transmissão. O campo RE (REsposta) é utilizado no protocolo de acesso ao meio.

Quando um nó da rede deseja transmitir dados ele deve aguardar por um canal vazio. Assim que encontrar, ele sinaliza o canal como cheio e transmite seus dados ao canal. O nó deve desprezar o canal caso ele tenha sido o seu usuário no quadro anterior, evitando o monopólio do mesmo.

Para sinais de voz digitalizados, o método foi modificado para evitar atrasos consideráveis. Como os sinais de voz aparecem em surtos onde a continuidade deve ser mantida, neste esquema de acesso proposto um canal é alocado para um surto e

permanece alocado até que ocorra um novo intervalo de silêncio.

Para a recepção de dados, os nós ficam analisando continuamente os campos destinatários dos canais. Uma vez detectado seu próprio endereço, o nó faz uma cópia do conteúdo do canal. Em seguida o canal é retransmitido, porém com uma modificação indicando que o dado foi lido. O nó que enviou o dado é responsável por liberar o canal utilizado.

Na realidade a RALFO possui uma comutação híbrida: temos a comutação de pacotes para dados e comutação de circuitos para surtos de voz.

Este esquema de acesso proposto é muito conveniente, pois pode ser compatibilizado com a RDSI-FL (Rede Digital de Seriços Integrados - Faixa Larga), colocando-se as células da RDSI-FL no campo de informação dos envelopes da RALFO.

2.5 Modelo de camadas da RALFO

2.5.1 Modelo de Referência OSI

Em 1977 a ISO (Organização Internacional de Padronização) vislumbrando a necessidade de padrões para a interconexão de sistemas heterogêneos criou um subcomitê (SC16) para estudar o problema [Giozza 17].

No ano seguinte, o SC16 sugeriu uma arquitetura estratificada em camadas para que a partir dela se desenvolvesse um padrão para o Modelo de Arquitetura [ISO 29] que serviria de suporte para o desenvolvimento de protocolos-padrão. O modelo foi concluído em meados de 1979 e recebia o nome de Modelo de Referência para Interconexão de Sistemas Abertos (RM-OSI) [ISO 30]. No final deste ano, o comitê técnico 97 da ISO acatava o modelo, e em maio de 83 o RM-OSI era oficialmente aprovado pela ISO como padrão internacional de interconexão de sistemas abertos, através do documento ISO 7498 [ISO 31]. O CCITT (Comitê Consultivo Internacional de Telefonia e Telegráfos) acatou também o RM-ISO através da recomendação X.200 [CCITT 8].

A definição do modelo utilizou a técnica de estruturação e abstração, decompondo o problema de interconexão de sistemas abertos em módulos ou camadas mais simples e logicamente independentes. Cada camada utiliza os serviços providos pela camada imediatamente inferior, para fornecer um serviço à camada superior.

A vantagem desta arquitetura é que o esforço global de desenvolvimento é reduzido através de abstrações: não interessa para uma camada como as demais implementam o fornecimento de seus serviços. O que importa são os serviços propriamente especificados. Existe independência na implementação de cada camada.

Figura 2.4: Modelo RM-OSI/ISO

O modelo RM-OSI tem uma arquitetura de 7 camadas: física, enlace, rede, transporte, sessão, apresentação e aplicação, organizadas conforme mostra a Fig. 2.4. Resumidamente serão apresentadas as principais funções de cada camada [Giozza 17]:

- FÍSICA: Definir as características elétricas, mecânicas, e funcionais bem como os procedimentos para ativar, manter e desativar conexões físicas para a transmissão de bits entre as entidades da camada de enlace de dados.
- ENLACE: Detectar, e possivelmente corrigir, erros na camada Física e prover procedimentos para ativar, manter e desativar uma ou mais conexões de enlace de dados entre as camadas de rede.
- REDE: Fornecer uma trajetória de conexão entre um par de entidades da camada de transporte, passando possivelmente por nós intermediários. Pertencem a esta camada os procedimentos de roteamento e controle de congestionamento.
- TRANSPORTE: Fornecer o serviço de tranferência de dados, entre duas entidades da camada Sessão, de uma maneira transparente, isto é, as entidades não precisam conhecer os detalhes pelos quais é alcançada uma tranferência confiável (chaveamento de pacotes, roteamento e uso eficiente dos recursos de comunicação disponíveis).
- SESSÃO: Organizar e sincronizar o diálogo (ex: autenticação de um usuário na máquina), e gerenciar a troca de dados entre entidades da camada de a-

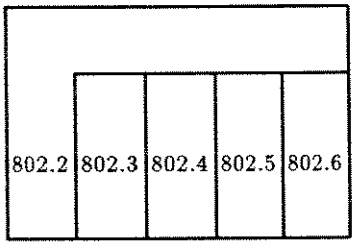


Figura 2.5: Padrões IEEE 802

presentação (ex: estabelecimento de comunicação do tipo "full-duplex", "half-duplex", etc).

- APRESENTAÇÃO: Fornecer os serviços que podem ser selecionados pela camada Aplicação para a interpretação da sintaxe dos dados trocados. Gerenciar a entrada, troca, apresentação e controle dos dados.
- APLICAÇÃO: Definir as interfaces entre os usuários no ambiente OSI, através das quais ocorrem todas as trocas de informação.

2.5.2 Arquitetura de redes locais

O projeto de padronização de redes locais teve início em fevereiro de 1980 com a criação do comitê 802 do IEEE [Giozza 17]. Os padrões deveriam ser compatíveis com o RM-OSI no nível dos protocolos de rede e deveriam também considerar os esforços de padronização para os protocolos de nível superior: transporte, sessão, apresentação e aplicação.

O comitê adotou as topologias de anel e barramento para redes locais e limitou a proposta de padrões às camadas física e enlace do RM-OSI, deixando a camada de rede vazia devido a não necessidade das funções de roteamento nestas topologias. Além disso, as camadas superiores (a partir da camada transporte até a aplicação) independem das características físicas da rede, e portanto, são aplicáveis às redes locais.

O comitê 802 propôs um conjunto de padrões atendendo às necessidades dos projetos de redes locais e às tendências existentes no mercado (Fig. 2.5). A proposta toma a camada de enlace do RM-OSI e a divide em duas subcamadas: Controle de Enlace Lógico (LLC) e Controle de Acesso ao Meio (MAC). O padrão para a subcamada LLC é denominado IEEE 802.2. Para a subcamada MAC foram estabelecidos

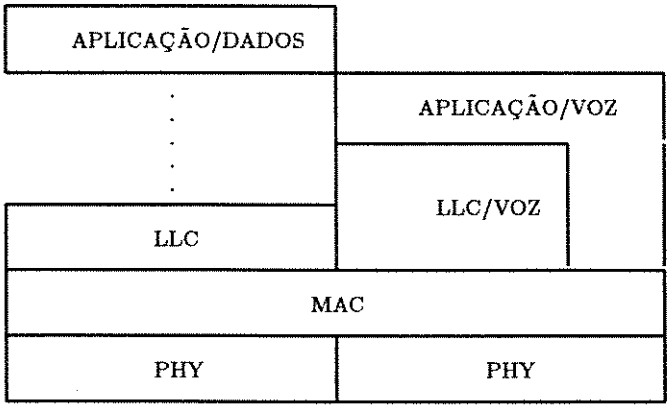


Figura 2.6: Arquitetura de camadas da RALFO

4 padrões de acordo com a topologia da rede e o método de controle de acesso ao meio utilizado:

- IEEE 802.3 - Topologia em barramento e controle CSMA/CD
- IEEE 802.4 - Topologia em barramento e controle por passagem de ficha
- IEEE 802.5 - Topologia em anel e controle por passagem de ficha
- IEEE 802.6 - Método de acesso para redes metropolitanas

A ISO decidiu adotar os documentos 802 em sua totalidade como "Draft Proposed Standards". A associação européia de manufaturadores de computadores (ECMA) que também trabalhou ativamente na padronização de redes locais, acatou oficialmente os padrões IEEE 802.

2.5.3 Arquitetura de camadas da RALFO

A estrutura em camadas, projetada para a RALFO está representada na Fig. 2.6. A distribuição das camadas nos processadores do nó da RALFO está de acordo com a Fig. 2.7.

- Camada PHY: responsável por ativar, manter e desativar conexões físicas para a transmissão de bits entre dois nós. Trata da conversão de pulsos elétricos para sinais ópticos e o procedimento inverso no recebimento de sinais ópticos.

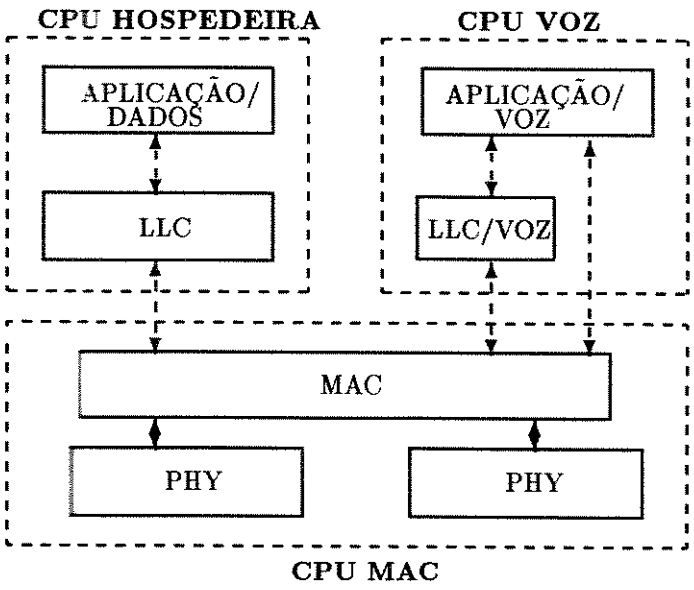


Figura 2.7: Localização física das camadas da RALFO

- Camada MAC: tem como função fazer a interface entre as camadas LLC, LLC/Voz e a camada física, provendo o controle de acesso ao meio físico que na RALFO corresponde ao duplo anel de fibra óptica.
- Camada LLC: reponsável pelo controle de enlace lógico de dados, e está implementada segundo as recomendações do padrão IEEE 802.2. Ela dará suporte às camadas superiores do modelo RM-OSI/ISO com objetivo de oferecer as facilidades de comunicação de dados.
- Camada LLC/Voz: responsável pelo controle de enlace de voz, cujo tratamento de sinalização tornou necessária a definição de um a camada apropriada.
- Camada Aplicação/Dados: Esta camada implementa as aplicações de transferência de dados.
- Camada Aplicação/Voz: responsável pelo tratamento das aplicações de terminais telefônicos.

Para testar a viabilidade da RALFO, foi definido um protótipo a ser implementado. Neste, a camada Aplicação/Dados comunica-se diretamente com a camada LLC,

sem intermédio das demais camadas do modelo RM OSI/ISO (REDE, TRANSPORTE, SESSÃO, APRESENTAÇÃO), para simplificar o escopo do trabalho.

2.6 Conclusões

Neste capítulo foi apresentado o projeto RALFO que propõe uma rede de serviços integrados de voz e dados. Ela utiliza como meio de transmissão, fibras ópticas em uma topologia de anel duplo, com o método de acesso próprio, baseado no "slotted ring" do Anel de Cambridge. O método de acesso prioriza o tratamento de voz, o que condiz com os estudos relacionados ao tratamento conjunto destes dois tipos de tráfego.

Para implementar esta rede foi definido um modelo de camadas baseado no modelo de referência OSI/ISO, onde duas camadas são definições próprias do projeto RALFO: camada MAC, implementa o método de acesso da RALFO; e a camada LLC/VOZ, responsável pelo tratamento da sinalização do transporte de voz.

Para testar a viabilidade desta rede será implementado um protótipo experimental que utiliza, como nó da rede, o processador PP. A ele serão acopladas mais duas placas processadoras: uma placa para tratamento dos terminais de voz (telefones); e outra placa para acesso ao meio físico (fibras ópticas). Serão implementadas as camadas PHY, MAC, LLC, LLC/VOZ, APLICAÇÃO/VOZ e a APLICAÇÃO/DADOS. A distribuição das camadas nos processadores do nó pode ser observada na Fig. 2.7. Para simplificar a implementação, a camada APLICAÇÃO/DADOS fará interface com a camada LLC, diretamente.

No próximo capítulo, serão apresentados os resultados dos estudos feitos sobre implementação de protocolos, e a definição de um ambiente de desenvolvimento para a implementação de protocolos da RALFO.

Capítulo 3

Aspectos gerais relacionados à implementação de protocolos

3.1 Introdução

Na implementação de aplicações de comunicações de dados (transferência de arquivos, correio eletrônico), em um determinado computador, geralmente pensa-se em adicionar às funções normais do seu sistema operacional, funções especializadas no tratamento de comunicação [Halsall 21]. Caso o sistema siga o modelo de camadas RM-OSI/ISO, a localização das camadas bem como a interação com o sistema operacional pode ser representada pela Fig. 3.1.

A implementação de todas as camadas de comunicação exige um esforço considerável de processamento e muitas vezes, para evitar sobrecarregar o computador, procura-se utilizar um sub-sistema de processamento dedicado para tratar os protocolos de uma determinada camada. Este sub-sistema de comunicação pode ser um cartão de circuito impresso conectado no barramento interno do processador hospedeiro.

Neste capítulo são discutidos os aspectos relacionados a implementação de protocolos: escolha do sistema operacional; escolha da linguagem de programação; e os requisitos da implementação de protocolos, bem como as escolhas feitas para o projeto RALFO.

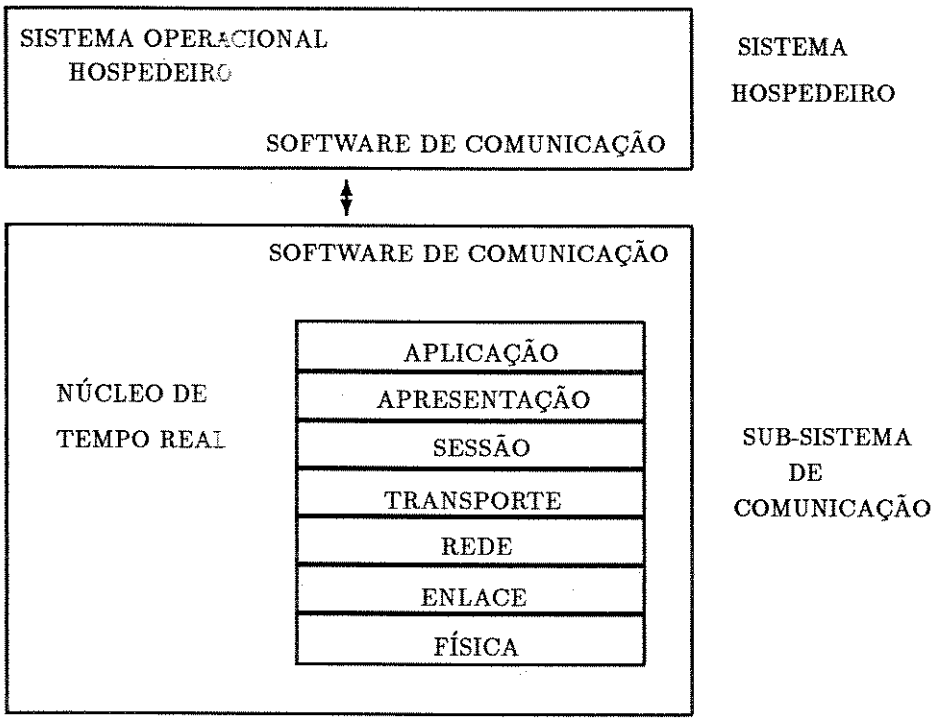


Figura 3.1: Interação das funções de comunicação - modelo OSI/ISO

3.2 Problemática

O modelo OSI/ISO não sugere como a implementação dos protocolos de comunicação deve ser feita, deixando esta decisão ao implementador. Entretanto, o modelo sugere que cada camada seja implementada como uma tarefa independente, que se comunica com suas tarefas vizinhas através da troca de primitivas. Portanto, os principais problemas a serem resolvidos durante a implementação de protocolos de comunicação são:

- Tarefas: Uma vez que o modelo RM OSI/ISO sugere uma implementação em camadas independentes é desejável que o sistema operacional e a linguagem de programação suportem o conceito de tarefas (processos);
- Comunicação de dados: As camadas se comunicam através de trocas de primitivas e PDUs (Protocol Data Unit). Portanto, é necessário que existam mecanismos que implementem a troca de mensagens entre as tarefas.
- Escalonamento de tarefas: As camadas, por serem independentes, exigem uma execução concorrente, e por terem restrições de tempo, relacionadas ao tratamento de mensagens e eventos, necessitam de um ambiente em tempo-real;
- Temporizadores: A maioria dos protocolos de comunicação estabele uma política de temporização de PDUs: caso a resposta a uma PDU não chegue em um intervalo de tempo estabelecido por um temporizador, determinadas atitudes são tomadas no protocolo. Logo, é desejável que o sistema operacional implemente facilidades no controle de tempo para implementação da temporização.

Na tese de M.C. Zabeu [Zabeu 49] estas questões são abordadas em detalhes. São discutidas também a necesssidade e a elaboração de um núcleo de tempo-real configurável, bem como sua aplicação na implementação de protocolos.

No livro de D.W. Davies [Davies 11] são discutidas as necessidades dos softwares para sistemas de comutação de pacotes: as funções, que deve ter um núcleo de um sistema multi-tarefa, relacionadas ao conceito de processos, compartilhamento de dados, escalonamento, tratamento de interrupções e controle de tempo. É apresentado também um projeto de software, de um núcleo de sistema operacional, para um computador, que é nó de uma rede com este tipo de comutação.

As primeiras decisões que devem ser tomadas para iniciar a implementação de protocolos é a escolha do sistema operacional, que gerenciará a execução de todas

as camadas no equipamento escolhido, e a escolha da linguagem de programação, que será utilizada na programação dos protocolos.

O sistema operacional escolhido deve suportar um ambiente multitarefa, através do conceito de processo. É desejável que ele ofereça algum mecanismo de troca de dados entre os processos. Além disso, o escalonamento destas tarefas deve obedecer a requisitos de um núcleo de tempo-real: compromisso com o tempo de execução e facilidades para manipulação de rotinas de interrupção. São necessários também, mecanismos de controle de tempo, para implementação de temporizadores.

Para a escolha da linguagem de programação, deve-se levar em conta o suporte a processos concorrentes e a comunicação de dados entre os processos. Muito se tem pesquisado nesta área e as soluções encontradas podem ser classificadas em três categorias [Madeira 37], conforme seus mecanismos de sincronização:

- Sincronização através de semáforos, eventos, condições e monitores: THE [Dijkstra 12], Modula [Wirth 48], [Brinch 4], [Hoare 26];
- Sincronização através de trocas de mensagens: [Brinch 5], [Harland 25] ;
- Sincronização através de "rendez-vous": CSP [Hoare 27], ADA [ADA 1].

Concluindo, o sistema operacional escolhido e a linguagem de programação utilizada devem suportar os mecanismos de concorrência necessários aos protocolos de comunicação. A seguir, serão apresentadas as escolhas feitas no projeto RALFO.

3.3 Escolha do Sistema Operacional

Determinada a arquitetura da rede RALFO (modelo de camadas e hardware), foi necessária a escolha dos produtos de software adequados para implementar os programas de comunicação.

Como mencionado anteriormente, a CPU Hospedeira (PP) será responsável pelas camadas LLC, Aplicação/Dados e opcionalmente todas as camadas intermediárias previstas no modelo OSI. São acopladas a ela, duas outras CPUs dedicadas:

- CPU MAC: implementa a camada MAC, sua comunicação com o PP será através de compartilhamento de memória e programas "drivers";
- CPU VOZ: implementa a camada Aplicação/Voz e a camada LLC/VOZ (Sinalização), sua comunicação com a MAC é feita por compartilhamento de memória.

Como o nó da rede possui três processadores, alguns responsáveis por várias camadas, ele necessita de um sistema operacional que atenda às seguintes características:

- oferecer um ambiente multitarefa em ambiente de processamento distribuído;
- ter características de um núcleo em tempo-real;
- permitir escalonamento de processos baseados em prioridades;
- possuir mecanismos de comunicação entre processos;
- permitir o tratamento de interrupções;
- ter mecanismos para gerenciamento de temporizadores.

A necessidade de características de tempo-real ocorre da necessidade dos softwares de comunicação controlarem vários tipos de hardware e manipularem procedimentos de controle que possuem restrições de tempo de execução. Uma falha em responder a um sinal em um tempo pré-determinado, pode causar a perda irrecuperável de informação ou outros danos. A natureza do hardware e do sistema operacional têm grande efeito sobre a performance do software de controle. [Davies 11].

O PP utiliza o microprocessador Intel 286, portanto, para a escolha de um sistema operacional que atendesse aos requisitos acima, foram analisadas as seguintes opções disponíveis no mercado:

- Xenix (Unix)
- PC-MOS (DOS Multiusuário)
- SOP (S.O. desenvolvido pelo CPqD da Telebrás)

Opção XENIX

O XENIX é a versão do sistema operacional UNIX feita pela empresa SCO (SC, USA) para processadores 80286. O UNIX foi projetado para ser um ambiente multi-tarefa e multi-usuário, "time-sharing". O escalonamento dos processos é feito alocando-se um intervalo de tempo para um processo, ao final do qual, ou por uma requisição de E/S, o processador é alocado ao próximo processo. Ele oferece um mecanismo de troca de informação entre processos, denominado "pipe" [Peterson 40].

Por não oferecer um ambiente de processamento em tempo-real e mecanismos adequados de comunicação entre processos, foi descartada sua utilização no projeto. Além disso, experiências demonstraram que o uso do XENIX em processadores 286, com mais de um usuário, degradam muito a performance oferecida a um usuário [Fiedler 15].

Opção PC-MOS

O PC-MOS é um sistema operacional multi-tarefa e multi-usuário, desenvolvido pela empresa The Software Link (GA, USA), disponível para a plataforma DOS, para processadores 80286 [Eglowstein 13]. Ele oferece basicamente um ambiente DOS e suporta aplicações DOS para cada terminal (limite de 25 terminais). Ele exige 1 MB para execução de uma única tarefa. Cada tarefa adicional é definida através da instrução ADDTASK, que aloca uma parte da memória extendida (acima de 1 MB) de tamanho configurável, e associa a uma linha serial, configurando-a com um tipo de terminal e a taxa de comunicação.

Ele oferece um mecanismo de troca de informações entre processos através de compartilhamento de memória. Define-se um dispositivo do tipo $PIPE.SYS, envia-se um dado para este dispositivo através da instrução COPY <arquivo.dados> <dispositivo>, retira-se um dado do dispositivo com o comando COPY <dispositivo> <arquivo.dados>. O PC-MOS não controla o acesso a memória compartilhada: se dois processos tentarem escrever ao mesmo tempo, as informações poderão ser escritas simultanealmente, o que exige do seu usuário, o desenvolvimentos de mecanismos para acesso exclusivo para garantir a integridade da informação.

O escalonamento dos processos é feitos por compartilhamento de tempo ("time-sharing"), ou seja, cada tarefa tem um limite de tempo configurável ("time-slice") para execução, ao final do qual é reescalonado para permitir a execução dos demais processos ativos.

O seu uso no projeto RALFO não se mostrou adequado devido a inexistência de características de processamento em tempo-real. Além disso, a troca de dados entre processos, para o caso de primitivas, exigiria o intermédio de uso de arquivos, o que causaria uma degradação na execução do protocolo. Não são oferecidas facilidades para tratamento da temporização.

3.3.1 Opção SOP

O sistema operacional SOP foi desenvolvido no CPqD da Telebrás pelo grupo do projeto PP-SO/P (Sistema Operacional para o PP operando no modo protegido) visando atender às seguintes necessidades dos projetos desenvolvidos no CPqD (centrais CPA[1] Telefônicas, Telex, etc..):

- controle de processos em tempo-real;
- ambiente multitarefa;
- eficiente esquema de comunicação e sincronismo entre processos;
- utilização em diferentes arquiteturas de sistema (distribuída, centralizada, multiprocessamento);
- controle de tempo (relativo/absoluto);
- proteção de software.

Os trabalhos da primeira versão tiveram início em 1988 e a última versão foi finalizada em julho de 1990.Atualmente seu código está implementado em Linguagem de Montagem do processador Intel 286, porém futuras versões do SOP, previstas no CPqD deverão ser desenvolvidas utilizando a linguagem C.Particularmente, ele suporta os mecanismos de concorrência da linguagem CHILL (CCITT High Interface Level Language).

Ele opera com o conceito de processo (usando o recurso "tarefa" do iAPX 286). Para atender os requisito de tempo-real, o SOP trata de forma eficiente interrupções geradas por interfaces de hardware e tem um sistema de prioridades, que permite uma política própria de escalonamento de processos. Para ser utilizado em sistemas distribuídos, ele suporta a comunicação entre processadores por meio de sinais, permite a implementação de programas que controlam interfaces de hardware específicas e suporta a implementação de um programa que controla a comunicação entre processadores. Possui mecanismos para detecção de falhas ("loops de programas", monopolização de recursos por um programa) e visualização das condições de funcionamento do sistema.

O SOP permite a execução de programas escritos em CHILL, implementando mecanismos de concorrência de forma análoga àquela definida implicitamente na

[1]Central por Programa Armazenado

linguagem. Ele é um sistema operacional configurável, isto é, pode-se decidir sobre a presença ou não de determinados módulos executáveis. O que é particularmente interessante no caso da CPU VOZ, que implementa duas camadas para uma aplicação dedicada e que não requer a complexidade necessária ao PP. Sua configuração completa exige um processador 80286 com 512 KB de memória RAM.

Diante das necessidades das aplicações da RALFO e das facilidades oferecidas pelo SOP, foi decidido que este seria o sistema operacional utilizado na rede.

3.4 Escolha da linguagem de programação

A linguagem de programação escolhida deveria atender aos seguintes requisitos:

- geração de código executável para o microprocessador Intel 286;
- suporte a execução concorrente de processos;
- suporte a comunicação entre processos;
- modularidade na programação.

Uma vez escolhida a máquina alvo e o seu o sistema operacional, a linguagem a ser utilizada deve explorar as características do ambiente definido: concorrência e multitarefa.

Analisamos algumas opções entre as linguagens concorrentes: Modula 2, Pascal concorrente, CHILL e consideramos também o uso de linguagens convencionais: C e Pascal. Estas últimas necessitavam do nosso compromisso de implementarmos a interface com as primitivas do sistema operacional SOP. Das linguagens concorrentes concluímos que a melhor opção era escolher a linguagem CHILL pois seus mecanismos de concorrência são suportados pelo sistema operacional SOP.

3.4.1 Linguagem CHILL

Em 1960 a CCITT percebeu que a tecnologia de telecomunicações deveria concentrar pesquisas na área de programação, dado o crescente uso de controle por programa armazenado (CPA) em sistemas de comutação [CHILL 10]. Em 1975, foram iniciados os estudos para a criação de uma linguagem de programação que atendesse a dois principais objetivos:

- Prover uma linguagem padrão atendendo a todos os projetistas e usuários de sistemas de controle por programa armazenado.

- Resolver todas as deficiências das linguagens existentes, utilizadas em sistemas CPA e consolidar em uma única linguagem as facilidades oferecidas por diversas linguagens.

Os trabalhos de elaboração da linguagem culminaram em uma proposta em 1979, aprovada pelo CCITT em novembro de 1980 (recomendação Z 200 - CCITT). Sua definição foi aprimorada no período de 1980 a 1984 e atualmente a mais recente versão é a aprovada na assembléia do CCITT em novembro de 1988. Está sendo liberada, de acordo com a revisão quadrienal feita pelo CCITT, a versao de 1992.

As principais características da linguagem são:

- suporta aplicações de tempo-real provendo mecanismos para comunicação e sincronização de processos e supervisão de tempo;

- permite programação concorrente em arquiteturas mono ou multiprocessadoras;

- possui mecanismos de compilação em partes;

- permite programação altamente modularizada e estruturada;

- permite o conceito de tipos abstratos de dados;

- oferece mecanismos de tratamento de exceções.

No próximo capítulo serão descritas, em detalhes, as facilidades da linguagem CHILL para a implementação de protocolos.

3.5 Ambiente de desenvolvimento de software

Uma vez escolhido o sistema operacional SOP e definido CHILL como a linguagem de programação, utilizamos as ferramentas oferecidas pelo Ambiente de Programação CHILL do CPqD (APCC). Estas ferramentas tem como objetivo capacitar o desenvolvimento de softwares em CHILL em duas plataformas: VAX (VMS), UNIX. As ferramentas são:

- Compilador: CHILL;

- Ligadores: LMP286, LNK286;
- Montadores: M286;
- Gerenciador de biblioteca: LIB286;
- Depurador Simbólicos: CSD, XCSD;
- Editor sensível a linguagem: ICE;
- Tradutor Pascal-CHILL: P2C.

CHILL

O compilador CHILL traduz programas fonte segundo as recomendações Z.200-1988 (APCC 4.0) para código objeto relocável destinado as plataformas 8086/88, 80186, 80286. É um compilador de múltiplos passos, capaz de otimizar o código gerado. Implementa também mecanismos de "compilação esperta", reduzindo o número de compilações das unidades de um programa, o que é muito útil para uma linguagem que incentiva a modularização.

Atualmente existem três ambientes de execução: Monitor CHILL (iAPX 286 operando no modo de endereçamento real); PP/SOP (iAPX 286 operando no modo virtual protegido); e SO88 (iAPX 8088).

LMP286

O LMP286 é um ligador que combina módulos objeto produzindo programas executáveis orientados para o processador 80286, operando no modo protegido. Os módulos objeto são arquivos contendo objeto relocável, produzidos pelo compilador CHILL e/ou montador M286.

LNK286

O LNK286 é um ligador que gera programa executável a partir da combinação de códigos objeto e bibliotecas. Ele deve ser utilizado quando se tem como máquina alvo o PP com o sistema operacional Monitor CHILL Modo Real, ou equipamentos compatíveis com PC com o Monitor CHILL Modo Real - Simplificado (não suporta programas concorrentes e alocação de memória via ALLOCATE). Deve ser utilizado quando o programa executável é gerado para o sistema operacional SO88 (8086/88)

M286

O montador M286 traduz programas escritos em um subconjunto extendido da linguagem de montagem do microprocessador iAPX 286 para código objeto relocável.

LIB286

LIB 286 é um gerenciador de bibliotecas que permite ao usuário criar, manter e examinar bibliotecas de módulos objeto, produzidos pelo compilador CHILL e/ou montador 286.

CSD

CSD (CHILL Symbolic Debugger) é um depurador simbólico disponível em plataformas UNIX e VAX/VMS. Ele permite que a execução de um programa CHILL seja conduzida e acompanhada de maneira interativa e controlada. A vantagem da utilização de um depurador está na possibilidade de manipulação dos objetos de um programas (constantes, variáveis, procedimentos, etc..) em um nível de abstração proporcionado pelo texto do programa fonte. Isto facilita a detecção e correção de erros. Ele permite a depuração de programas concorrentes e rastreamento de sinais. Esta facilidade é muito importante para o caso de implementação de protocolos.

XCSD

O XCSD é um depurador simbólico disponível para plataformas SPARC/SunOS, operando sob X Windows (V.11 r.4) ou OpenWindows, ligadas através de duas linhas seriais a um PP/SOP. Seu uso é totalmente orientado por janelas e a interação do usuário com este produto é feita através de "mouse" e teclado.

ICE

ICE (Interactive CHILL Editor) é um editor de texto direcionado para programação em CHILL. Ele permite a criação e alteração de programas CHILL e provê facilidades para a criação de programas corretos (sintática e semanticamente), formatados e compilados.

Ele está disponível em plataformas UNIX (X Windows V.11 r.4) e plataformas VAX/VMS. O trabalho de edição torna-se extremamente simples e o uso do ambiente X Windows, mais ainda uma vez que a edição é feita através de janelas (windows) e a seleção das opções, através do uso de "mouse".

P2C

O tradutor Pascal-CHILL, chamado P2C, traduz arquivos fonte em Pascal para os comandos CHILL correspondentes. Suporta pequenas extenções da linguagem implementadas pelo Pascal do VAX/VMS.

Configuração das máquinas

Dada a disponibilidade de estações de trabalho SPARC (SUN) para o desenvolvimento do software, foi dada a preferência ao uso das ferramentas desta plataforma. Portanto, o trabalho de edição , compilação e geração de código foi feito nestes equipamentos e o código executável era transportado para o PP através da conexão de uma linha serial da UPN a uma porta serial da estação. O software utilizado para a tranferência foi o KERMIT, cuja versão para o PP [KERMIT 33], foi desenvolvida pelo CPqD/Telebrás. Observe a configuração das máquinas conforme a Fig. 3.2.

3.6 Outros sistemas de execução para CHILL

De acordo com trabalhos publicados sobre CHILL, existem outras implementações de núcleos de sistemas operacionais que suportam as primitivas da linguagem CHILL. Por exemplo:

- RTS (Run Time System) [Hari 23]:desenvolvido no C-DOT[2], no projeto C-DOT DSS (Digital Switching System) para os nós da rede de equipamentos baseados no Motorola 680X0. Ele suporta o ambiente concorrente da linguagem.
- OSK (Operating System Kernel) [Hao 24]: desenvolvido no Nanjin Institute of Communication Engineering (R.P. China) para equipamentos compatíveis com PC (8088/8086). Suporta o ambiente concorrente da linguagem e todas as características de um núcleo em tempo-real.

3.7 Conclusões

Neste capítulo, foi apresentado o ambiente de desenvolvimento para a implementação de protocolos na RALFO. As primeiras decisões que devem ser tomadas

[2]Center for Development of Telematics, Bangalore, India

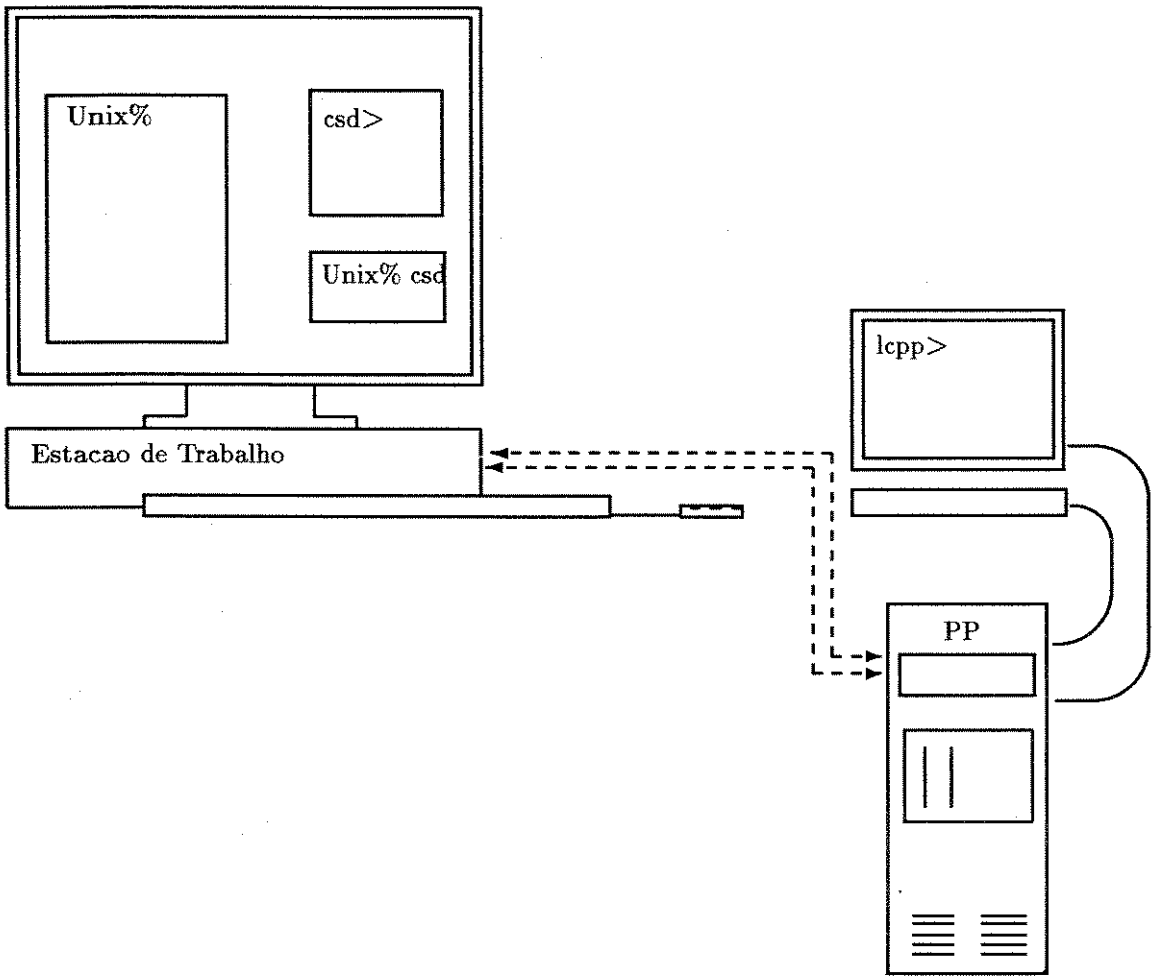


Figura 3.2: Interação da estação de trabalho com o PP

para iniciar a implementação de protocolos é a escolha do sistema operacional, que gerenciará a execução de todas as camadas no equipamento escolhido, e a escolha da linguagem de programação, que será utilizada na programação dos protocolos. O sistema operacional deve ter características de tempo-real, oferecer um ambiente multi-tarefa, fornecer mecanismos de comunicação de dados entre processos e mecanismos que permitam a implementação de temporizadores. A linguagem deve oferecer, preferencialmente, mecanismos de concorrência, ou seja, implementar conceitos de tarefas, comunicação entre tarefas e supervisão de tempo.

No projeto RALFO, optou-se pelo sistema operacional PP/SOP e a linguagem de programação CHILL. Por ter sido projetada pelo CCITT, com o objetivo de desenvolver sistema de comunicação, CHILL oferece recursos muito poderosos e parece ser uma ótima opção para a implementação de protocolos. Para implementar os protocolos da RALFO, foi utilizado o conjunto de ferramentas APCC, desenvolvido eficientemente pelo CPqD/Telebrás.

No próximo capítulo, serão explorados, em detalhes, os recursos da linguagem que facilitam a implementação de protocolos.

Capítulo 4

Implementação de protocolos usando CHILL

4.1 Introdução

Existem trabalhos que propõem a derivação de programas CHILL, a partir de especificações SDL (Specification and Description Language, CCITT). Venstad et al. em [Hallsteinsen 19], citam o projeto sobre ambientes de engenharia de sistemas do NTA-RD[1] em cooperação com ELAB-RUNIT[2], onde foi definida e está em implementação, uma ferramenta capaz de gerar código CHILL, partindo de especificações SDL e aplicando sucessivamente regras de transformação de programas [Hallsteinsen 20].

Outras ferramentas semelhantes são: DASOM [Johansen 32], que também é baseada em regras de transformação; e Melba [Kong 35], uma proposta que deixa a cargo do usuário muitos detalhes da derivação, o que torna seu uso um pouco restrito.

Estes trabalhos mostram que o uso da linguagem CHILL para a implementação de protocolos tem sido objeto de interesse na área de desenvolvimento de softwares de comunicação. CHILL oferece vários recursos que facilitam a implementação de protocolos [Sobrinho 43]: conceito de processos; mecanismos de comunicação de dados entre processos e facilidades de supervisão de tempo.

[1] Norwegian Telecommunication Administration, Research Department

[2] Research Lab of SINTEF, Foundation for Industrial and Technical Research at the Norwegian Institute of Technology

Neste capítulo, serão descritos em detalhes cada um destes recursos, bem como seus modos de utilização.

4.2 Processos

Quando se adota o modelo de camadas para especificar um protocolo de comunicação, assume-se que cada camada é uma entidade independente que oferece à camada superior um conjunto de serviços, e utiliza os serviços prestados por sua camada inferior.

Portanto, cada camada deve ser implementada como uma tarefa independente e o funcionamento do protocolo de comunicação baseia-se na execução de todas as tarefas (correspondentes às camadas) de modo concorrente.

A linguagem CHILL oferece o conceito de processo. Por definição, um processo é uma unidade de execução na qual os comandos que a constitui, são executados sequencialmente. Entretanto, um processo pode ser executado concorrentemente com outros processos. A sintaxe para a definição de um processo é:

```
<Nome do processo>:

 PROCESS ( <Lista de parametros>);

   <Comandos>

 END;
```

Sua definição é semelhante a de um procedimento, e possui mecanismos de passagem de parâmetros similares. A ativação de um processo é feita pelo comando START. O término do processo se dá quando ele executa todos os comandos de seu bloco, ou através de um comando STOP executado por outro processo.

Este recurso possui importantes facilidades: mecanismos de sincronização e comunicação de dados entre processos.

4.3 Comunicação e sincronização entre processos

As camadas de um protocolo de comunicação se comunicam trocando serviços. Portanto fazem-se necessários mecanismos que permitam a comunicação de dados entre processos. A linguagem CHILL oferece quatro opções:

- Evento: Sincroniza execuções dos processos através de instruções DELAY (aguardar por um evento) e CONTINUE (liberar os processos que aguardam por um determinado evento).
- Região: Organiza o acesso a uma área comum de dados compartilhada por vários processos. Uma vez que um processo utilize recursos pertencentes a uma região, esta é reservada e se outros processos tentarem o acesso a ela eles ficarão bloqueados até que a região seja liberada.
- Sinal: Provê sincronização e comunicação independente da localização dos dados. Ele permite a composição e a decomposição de um conjunto de valores a ser transmitido. A instrução SEND é utilizada para enviar uma lista de valores (de quanquer tipo) e a instrução RECEIVE CASE é utilizada para identificar um sinal e seus valores associados.
- "Buffer": Provê um meio de sincronização e comunicação através de um conjunto de elementos do mesmo tipo. As operações são utilizadas através das instruções SEND e RECEIVE CASE.

No caso de protocolos de comunicação o uso de sinais é muito interessante, pois pode-se utilizar o nome do sinal para identificar a primitiva, e o conjunto de informações associadas ao sinal para especificar os parâmetros da primitiva. O envio de sinal não causa a suspensão do processo e o sincronismo surge naturalmente: um processo é suspenso quando espera por sinais e sua fila está vazia; e automaticamente ativado quando recebe um sinal.

Além disso, o uso de sinais apresenta vantagens sobre os demais recursos porque: evento só permite sincronização; região necessita esforços adicionais para identificar o serviço e definição de estruturas de dados para os parâmetros; e finalmente os "buffers" tem a restrição de armazenar elementos do mesmo tipo.

Define-se um sinal da seguinte maneira:

```
DCL

 SIGNAL

 <Nome do sinal>=(Tipo do param. 1, Tipo do param. 2, ..., Tipo do param. N);
```

A camada *N* que necessita solicitar um serviço da camada *N-1* envia um sinal a ela indicando o serviço desejado:

```
  SEND  <Sinal> TO  <Processo>
```

A camada *N-1* que oferece este serviço identifica o sinal, como um serviço oferecido e providencia as atitudes apropriadas a cada sinal, da seguinte maneira:

```
RECEIVE CASE SET <Processo>;

( <Sinal> IN <lista de parametros> ): BEGIN
                                        <Comandos>
                                      END;
ELSE <Comandos>;

ESAC;
```

Além de permitir o sincronismo entre os processos, é possível associar a cada sinal um nível de prioridade que será levado em conta no tratamento deste. Isto é muito adequado para o caso onde algum serviço deve ser priorizado sobre outros. Exemplo: temporizações.

A linguagem também permite o envio de sinais a processos localizados em diferentes processadores. Para o projeto RALFO, que possui várias placas processadoras, isto é particularmente importante.

4.4 Temporizações

A linguagem CHILL oferece facilidades de temporização e agendamento nos mecanismos de comunicação. A finalidade básica do uso destes recursos é permitir

a ativação de processos e envio de sinais nos vencimentos de tempos solicitados [CHILL 9].

Pode-se associar temporização às ações: SLEEP (onde após o vencimento do tempo associado, o processo é ativado); DELAY e RECEIVE (uma exceção TIME-OUT é causada pelo vencimento do tempo); e SEND (um sinal é enviado ao final da temporização).

A função de agenda permite ativar processos (SLEEP) ou enviar sinal (SEND) no vencimento de uma data ou horário previamente agendado. Pode-se estabelecer intervalos de horários onde o sinal será enviado em intervalos pré-definidos.

Existem funções que permitem cancelar ou reiniciar uma temporização ou agendamento. A seguir, ilustraremos com um exemplo, a utilização de um sinal temporizado:

```
DCL
   id_A, id_B INSTANCE;

SIGNAL
   B TO Proc_B;


Proc_A:
PROCESS ();
 DCL
    tmr temporization,
    pt_timelocation ref_timelocation,
    var_timelocation time_location := [1,1,0];

  /* Configuracao do Temporizador */

  pt_timelocation:= -> var_timelocation;
  tmr:=[.tempkind: temp ,
        .timeloc: pt_timelocation,
        .pe: [.after: [.value:2, .unit:sec],
              .every: [.value:5, .unit:sec],
              .during:[.value:60,.unit:sec] ] ];

  Timed_Send (tmr);
```

```
  SEND B TO id_B;

END Proc_A;

Proc_B:
PROCESS ();
   DO FOR EVER;
    RECEIVE CASE
      (B): PUTF(TTY, 'RECEBIDO SINAL B');
    ESAC;
   OD;
END Proc_B;

id_A:=START Proc_A();
id_B:=START Proc_B();
```

Neste exemplo um sinal B será enviado pelo processo Proc_A ao processo Proc_B durante 60 segundos a cada 5 segundos, depois de 2 segundos da execução da instrução SEND.

No caso de protocolos de comunicação, onde é muito comum o uso de temporizadores, pode-se definir um processo temporizador que seria ativado quando recebesse uma instrução para ativar um determinado temporizador. A partir daí, enviaria ao processo solicitante, em intervalos definidos para cada temporizador, um sinal de expiração, até que este cancelasse o pedido de temporização.

4.5 Modularidade

A implementação de todas as camadas de um protocolo de comunicação pode exigir um esforço muito grande de programação e grande quantidade de linhas de código. Uma vez que cada camada funciona como uma tarefa independente, ela pode ser implementada de forma independente também. Para isso, são necessários recursos que garantam modularidade e integridade no desenvolvimento de software.

Módulos constituem a unidade básica de estruturação de programas CHILL. Eles oferecem controle de visibilidade dos objetos; encapsulamento e abstração de dados; decomposição de programas em partes que podem ser desenvolvidas de maneira segura e independente; e acesso mutuamente exclusivo dos processos concorrentes aos objetos declarados localmente [APCC 2].

O módulo define o escopo da visibilidade dos objetos. Os nomes criados dentro de um módulo não são visíveis fora dele e vice-versa. Essa visibilidade pode ser alterada através de instruções de importação (SEIZE) ou exportação (GRANT) de objetos.

A independência na programação é conseguida através do mecanismo de programação em partes: a idéia é que quando uma parte do programa vai ser desenvolvida separadamente, ela seja substituída por um módulo de especificação , que define as propriedades estáticas dos nomes exportados por ele. Por exemplo, considere um programa esquematizado da seguinte maneira:

```
Prog1: MODULE
          .
          .
          .

          Prog2: MODULE
                    .
                    .
                    .
                   END Prog2;
          END Prog1;
```

Suponha que se deseja desenvolver Prog2 separadamente. O programa anterior deve ficar da seguinte forma:

```
Prog1: MODULE
          .
          .
          .

          Prog2: SPEC MODULE
                    .
                    .
               .
                    END Prog2;
                  END Prog1;
```

O corpo do módulo Prog2, no programa original, só deverá conter os objetos visíveis ao módulo Prog1 (variáveis e definições de procedimentos, por exemplo) e

neste contexto é denominado módulo de especificação. Assim o módulo Prog1 pode ser desenvolvido em paralelo com o Prog2, uma vez que os relacionamentos já estão definidos. Pode-se compilar os dois módulos independentemente, pois a interface entre eles já está estabelecida através do módulo de especificação. A implementação efetiva do módulo Prog2 é feita através de outra definição do módulo, porém sem a palavra SPEC, indicando que é a implementação real. O contexto define os nomes que serão importados por este módulo e aqueles que serão exportados por ele.

```
CONTEXT
    .
    .
    .

FOR
Prog2: MODULE
          .
          .
          .
       END Prog2;
```

4.6 Conclusões

Para a implementação de protocolos, devemos procurar linguagens que ofereçam recursos para a implementação de tarefas (processos) de maneira modular e independente. Assim, cada camada poderá ser implementada como um processo. Para que as camadas troquem primitivas é preciso algum mecanismo de troca de dados entre os processos, por exemplo, envio de mensagens, compartilhamento de memória, etc.. Finalmente, para temporizar primitivas enviadas às camadas remotas, é desejável que a linguagem ofereça facilidades na supervisão de tempo.

Uma vez tendo o protocolo especificado através de diagramas de transições para cada estado do protocolo, a implementação fica imediata:

1. Define-se um processo para cada camada.

2. Para cada camada define-se um sinal para cada primitiva, e os parâmetros da primitiva formam o conjunto de dados associado ao sinal.

3. Para cada estado de uma camada define-se um procedimento que fará o tratamento dos sinais recebidos através de instruções do tipo RECEIVE CASE.

4. Todo o envio de sinais é feito através de instruções SEND.

5. Implementa-se as temporizações através de sinais temporizados.

Todos os aspectos da implementação da comunicação e sincronização dos processos ficam transparentes ao programador, tendo este que se preocupar exclusivamente com o protocolo de comunicação.

A linguagem CHILL se mostra como uma poderosa opção para a implementação de softwares de comunicação. Atualmente no Brasil, ela tem sido utilizada no desenvolvimento de projetos do CPqD/Telebrás, tais como Trópico RA [Franco 16], CETEX [Ferreira 14]. A sua utilização, na implementação de protocolos, será ainda mais facilitada com a disponibilidade de ferramentas capazes de derivar programas CHILL a partir de especificações formais, como os exemplos das ferramentas SDL-CHILL.

No próximo capítulo, será apresentada a implementação do protocolo LLC na RALFO, usando CHILL. Poderá ser visto como todos os recursos, aqui mencionados, foram aplicados na implementação.

Capítulo 5

Implementação da camada LLC da RALFO

5.1 Introdução

A seguir, será discutida a implementação da camada LLC[1] da RALFO utilizando a linguagem CHILL. Inicialmente a camada LLC será apresentada, segundo o protocolo IEEE 802.2 [CCITT 7], para em seguida detalhar a implementação feita.

5.2 Protocolo LLC

A camada LLC é reponsável pela transmissão correta entre duas entidades conectadas por um enlace de dados. Ela define dois tipos de operação para a comunicação de dados entre pontos de acesso de serviço (SAP): Tipo 1, onde os dados são transferidos entre as camadas LLC sem necessidade de procedimentos de estabelecimento de conexão, confirmação de recebimento, controle de fluxo de dados e recuperação de erros; e Tipo 2, onde existem os procedimentos de estabelecimento e término de conexão, controle de fluxo, recuperação de erros, oferecendo maior confiabilidade na operação.

Foi implementada a LLC Tipo 2. A escolha desta opção se deu pela garantia da confiabilidade de transmissão oferecida através de mecanismos de controle de fluxo e recuperação de erros, inexistentes na LLC Tipo 1. Portanto assume-se aqui que

[1] Daqui por diante os termos Aplicação e LLC sempre serão relacionados a dados, por se tratar da implementação da comunicação de dados.

PDU I:	I (Information)
PDUs S:	RR (Receive Ready)
	RNR (Receive Not Ready)
	REJ (Reject)
PDUs U:	UI (Unnumberede Information)
	DISC (Disconnect)
	SABME (Set Async. Bal.Mode Ext.)
	XID (Exchange Information)
	TEST (Test)
	UA (Unnumbered Acknowledgement)
	DM (Disconnected Mode)
	FRMR (Frame Reject)

Tabela 5.1: Tabela das PDUs do protoclo LLC

toda referência a LLC, neste trabalho, deve ser subentendida como o padrão IEEE 802.2 Tipo 2.

A camada LLC tem como camadas vizinhas as camadas Aplicação (camada superior) e MAC (camada inferior). Ela oferece 5 tipos de serviços (Fig 5.1):

- Estabelecimento de conexão: L_Connect;
- Transferência de dados: L_Data_Connect;
- Desconexão: L_Disconnect;
- Reiniciação do enlace: L_Reset;
- Controle de fluxo do enlace: L_Connection_FlowControl;

e utliza o serviço de envio de dados (MA_Data) oferecido pela camada MAC.

5.2.1 Estrutura da PDU da camada LLC

A PDU[2] é formada por quatro campos:

- Endereço DSAP: endereço do SAP destinatário;

[2]Unidade de Dados do Protocolo

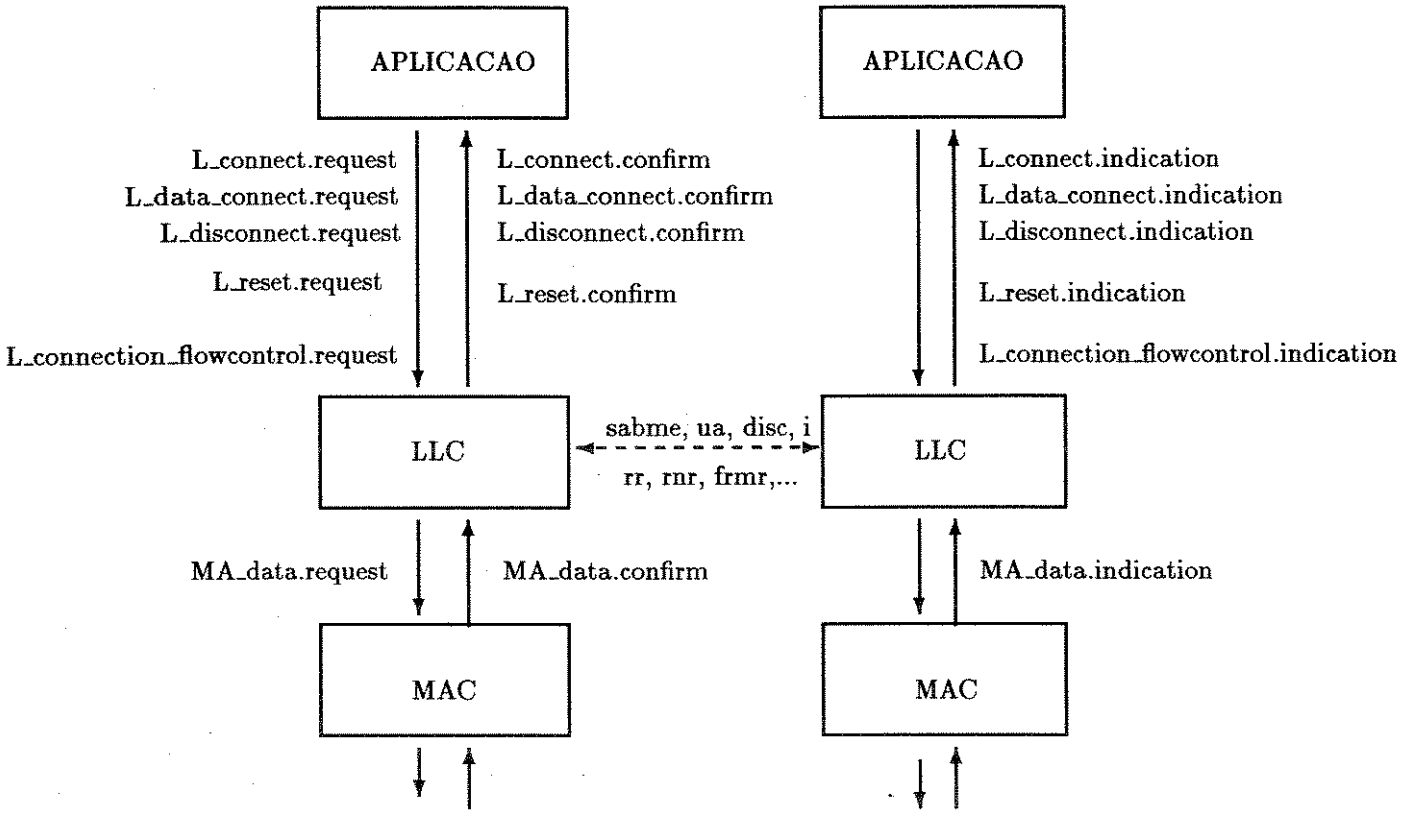


Figura 5.1: Serviços da camada LLC

- Endereço SSAP: endereço do SAP origem;
- Controle: campo que identifica a PDU;
- Informação: dados a serem transmitidos.

Existem três tipos de PDUs:

- I (Information): Indica que a PDU é um quadro de dados;
- S (Supervisory): PDU utilizada para supervisionar o enlace de dados: resposta a PDUs I, pedidos de retransmissão, pedidos de suspensão de envio de quadros I.
- U (Unnumbered): PDU utilizada para prover funções de controle de enlace adicionais, e transferência de informação sem sequenciamento.

Todas as PDUs existentes no protocolo podem ser vistas na tabela 5.1.

5.2.2 Diagrama de estados da LLC

Os diagramas de estados da LLC, representados pelas Fig.5.2, Fig.5.3, mostram todos os estados do protocolo relativos aos procedimentos de estabelecimento, término, reinicialização dos enlaces e tranferência de informação.

Segue uma breve descrição dos estados do protocolo:

ADM: (Asynchronous Disconnected Mode) Neste estado é possível receber um pedido de conexão de uma camada superior, ou de uma camada remota.

SETUP: Foi transmitida uma PDU SABME e aguarda-se por uma resposta.

NORMAL: Existe uma conexão de dados estabelecida entre o SAP LLC local e o remoto. O envio de quadros de informação e supervisão podem ser feitos.

D_CONN: Devido a um pedido da camada superior, foi enviado ao SAP remoto, um pedido de desconexão e aguarda-se uma resposta.

ERROR: Foi detectado um erro em uma PDU recebida e enviada ao SAP remoto, a PDU FRMR. Aguarda-se por uma resposta.

RESET: Devido a um pedido da camada superior, foi enviado uma PDU SABME, indicando que o enlace deve ser reinicializado. Aguarda-se por uma resposta.

AWAIT: Existe uma conexão de dados estabelecida entre o SAP LLC local e o remoto. O SAP local está aguardando uma resposta a um quadro com o bit P igual a 1. PDUs I podem ser recebidas mas não enviadas.

BUSY: Existe uma conexão de dados estabelecida entre o SAP LLC local e o remoto. O SAP local não pode receber quadros I. Somente enviá-los.

REJECT: Existe uma conexão de dados estabelecida entre o SAP LLC local e o remoto. Foi pedido a LLC local que reenvie os quadros a partir de uma determinada numeração, pois foi detectada uma falha na sequência dos quadros recebidos pela LLC remota.

AWAIT_BUSY: Existe uma conexão de dados estabelecida entre o SAP LLC local e o remoto. O SAP local está aguardando uma resposta a um quadro com o bit P igual a 1. PDUs I não podem ser recebidas mas não enviadas. Apenas os quadros de supervisão são processados.

AWAIT_REJECT: Existe uma conexão de dados estabelecida entre o SAP LLC local e o remoto. O SAP local está aguardando uma resposta a um quadro com o bit P igual a 1. PDUs I podem ser recebidas mas não enviadas. Além disso foi solicitado o reenvio de quadros a partir de uma determinada numeração, pois foi detectada uma falha na sequência dos quadros recebidos pela LLC remota.

O diagrama temporal do protocolo LLC é apresentado na Fig. 5.4. Através dele é possível ter uma idéia melhor do relacionamento das primitivas e as PDUs do protocolo. São apresentados os procedimentos de Estabelecimento de enlace (C); Tranferência de informação (T); Desconexão do enlace (D); Reinicialização do enlace (R); e Controle de fluxo de dados (F).

Os diagramas de transições e todos os detalhes do protocolo LLC podem ser encontrados em [CCITT 7].

O protocolo LLC possui alguns parâmetros importantes, relacionados à temporização, transmissão de informações (quadros I) e ao tamanho da PDU do protocolo. São eles:

- Temporizadores
  - Acknowledgement Timer: limite de tempo que a LLC deve esperar para receber uma resposta a uma PDU I ou PDU comando (P igual a 1);

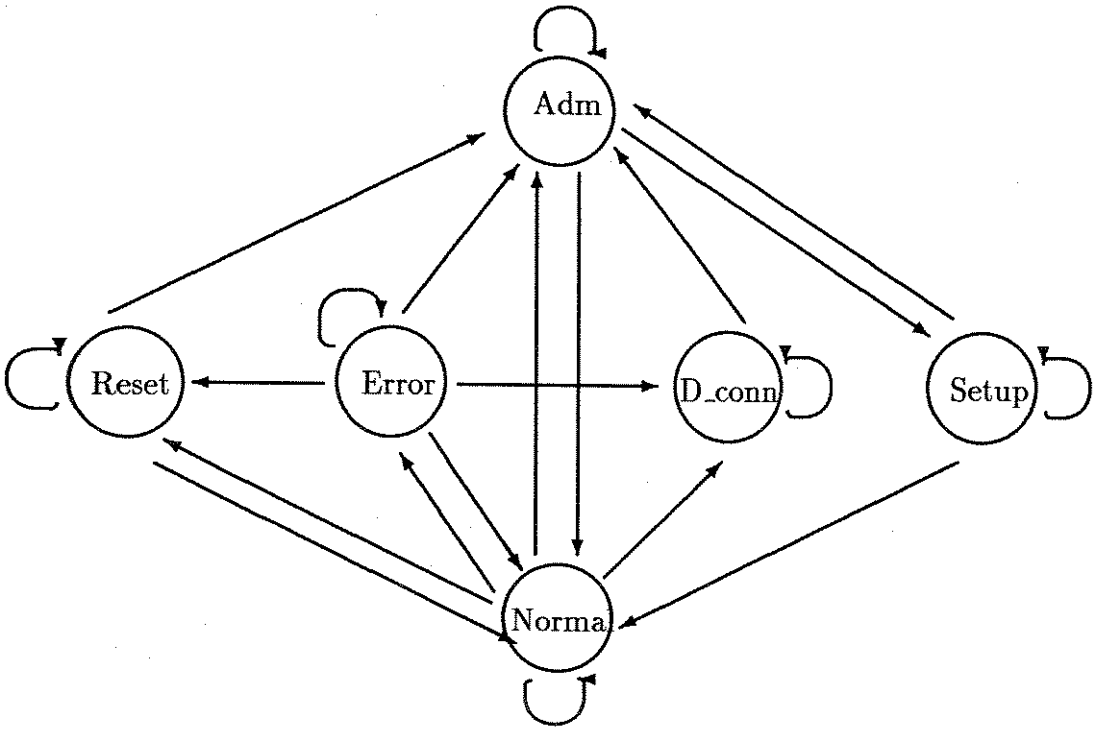


Figura 5.2: Diagrama de estados da LLC (Estabelecimento, término e reinicialização do enlace)

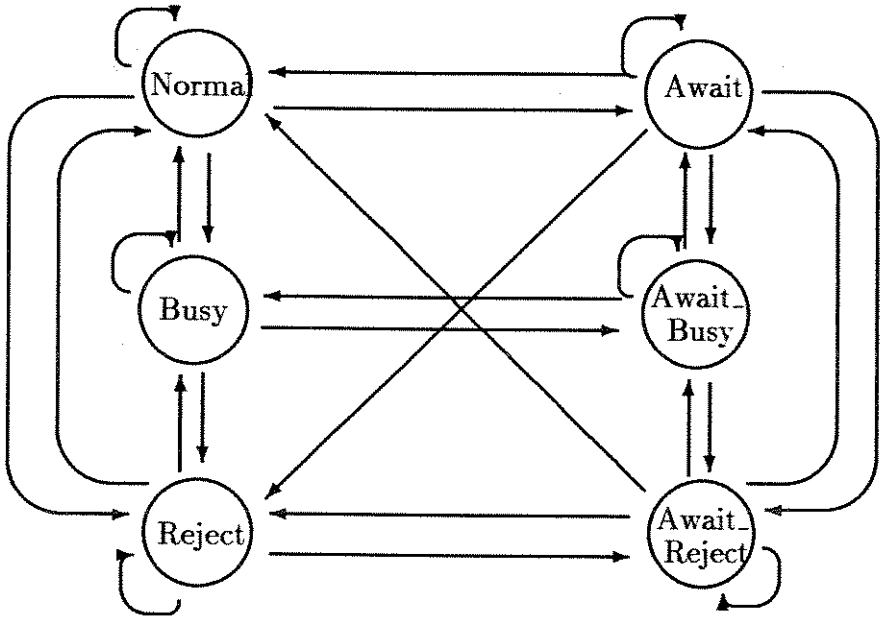


Figura 5.3: Diagrama de estados da LLC (Fase de transferência de informações)

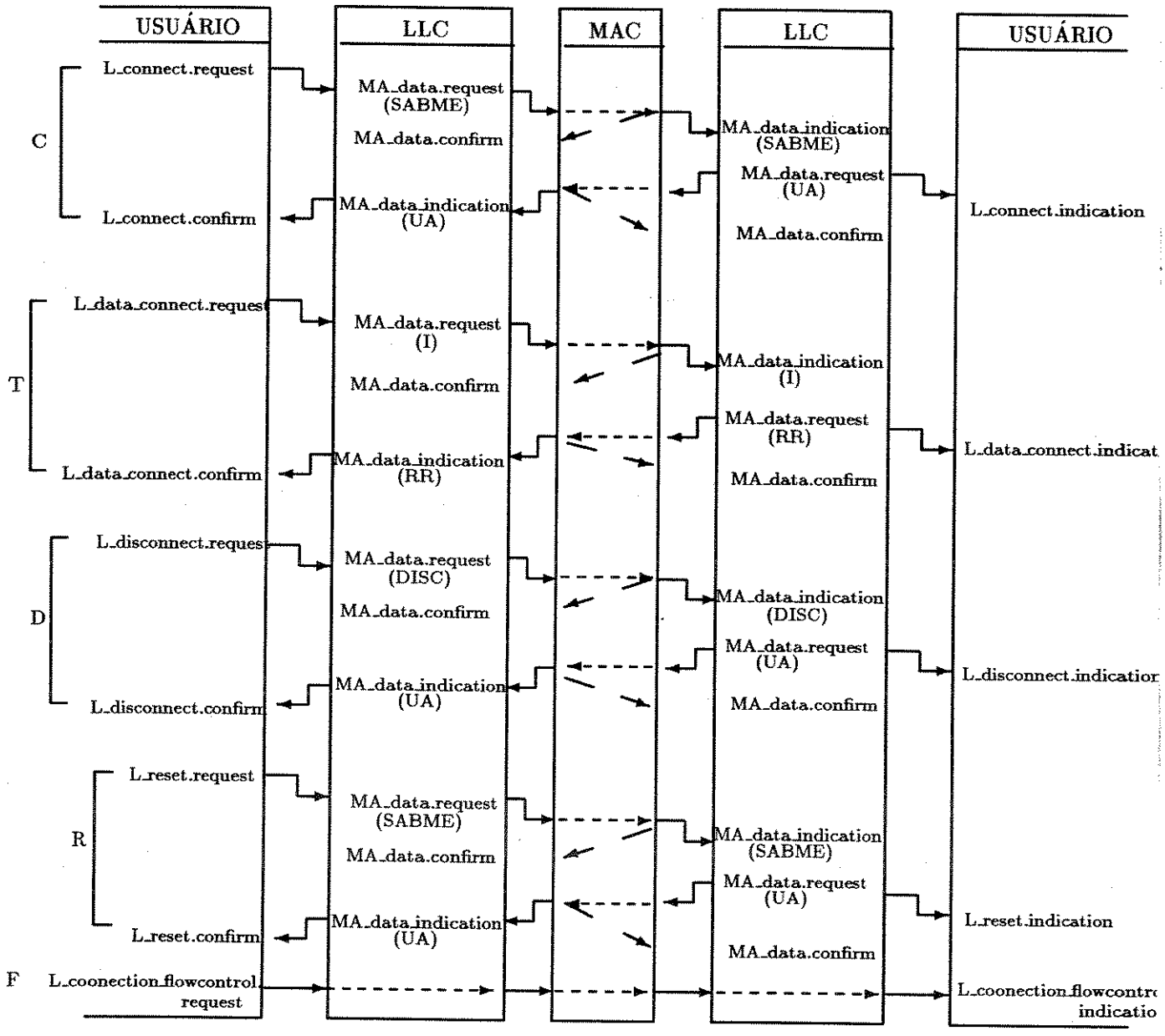


Figura 5.4: Diagrama temporal do protocolo LLC

  - P-bit Timer: limite de tempo que a LLC deve esperar para receber uma resposta F (igual a 1) a uma PDU comando (P igual a 1);
  - Reject Timer: limite de tempo que a LLC deve esperar para receber uma resposta a uma PDU REJ enviada;
  - Busy-State Timer: limite de tempo que a LLC deve esperar para receber uma indicação que a LLC remota saiu do estado BUSY;

- Número máximo de transmissões (N2): Número máximo de vezes que uma PDU deve ser retransmitida, após o vencimento de uma temporização;
- Número máximo de octetos em uma PDU I (N1): Este valor que depende da camada MAC, estabelece o tamanho do quadro de informações;
- Número máximo de PDUs I pendentes (K): Número máximo de PDUs I que podem estar aguardando por confirmação;
- Número mínimo de octetos em uma PDU: 3 ou 4 dependendo do tipo da PDU: U (3), S (4) e I (4).

Na próxima secção, serão explicadas as simplificações do protocolo feitas para a implementação da LLC no protótipo da RALFO, bem como todos os detalhes desta implementação.

5.3 Implementação

5.3.1 Simplificações feitas na implementação da RALFO

Dada a complexidade do protocolo LLC, quando observado em sua totalidade de estados e possíveis transições, algumas simplificações foram feitas de modo a facilitar o exercício da implementação, sem diminuir sua funcionalidade.

Estas simplificações foram obtidas tomando-se o protocolo original e eliminando os estados RESET e ERROR, impedindo o pedido de reinicialização do enlace e uma vez ocorrido um erro, o enlace é rompido, realizando o procedimento de desconexão. Além disso, o estado BUSY e seu correspondente AWAIT_BUSY também foram eliminados. Foi assumido que a capacidade de processamento de quadros I é ilimitada. Assim, os diagramas de estado puderam ser simplificados para os diagramas apresentados pela Fig. 5.5. A especificação do protocolo simplificado, em SDL, encontra-se no apêndice A.

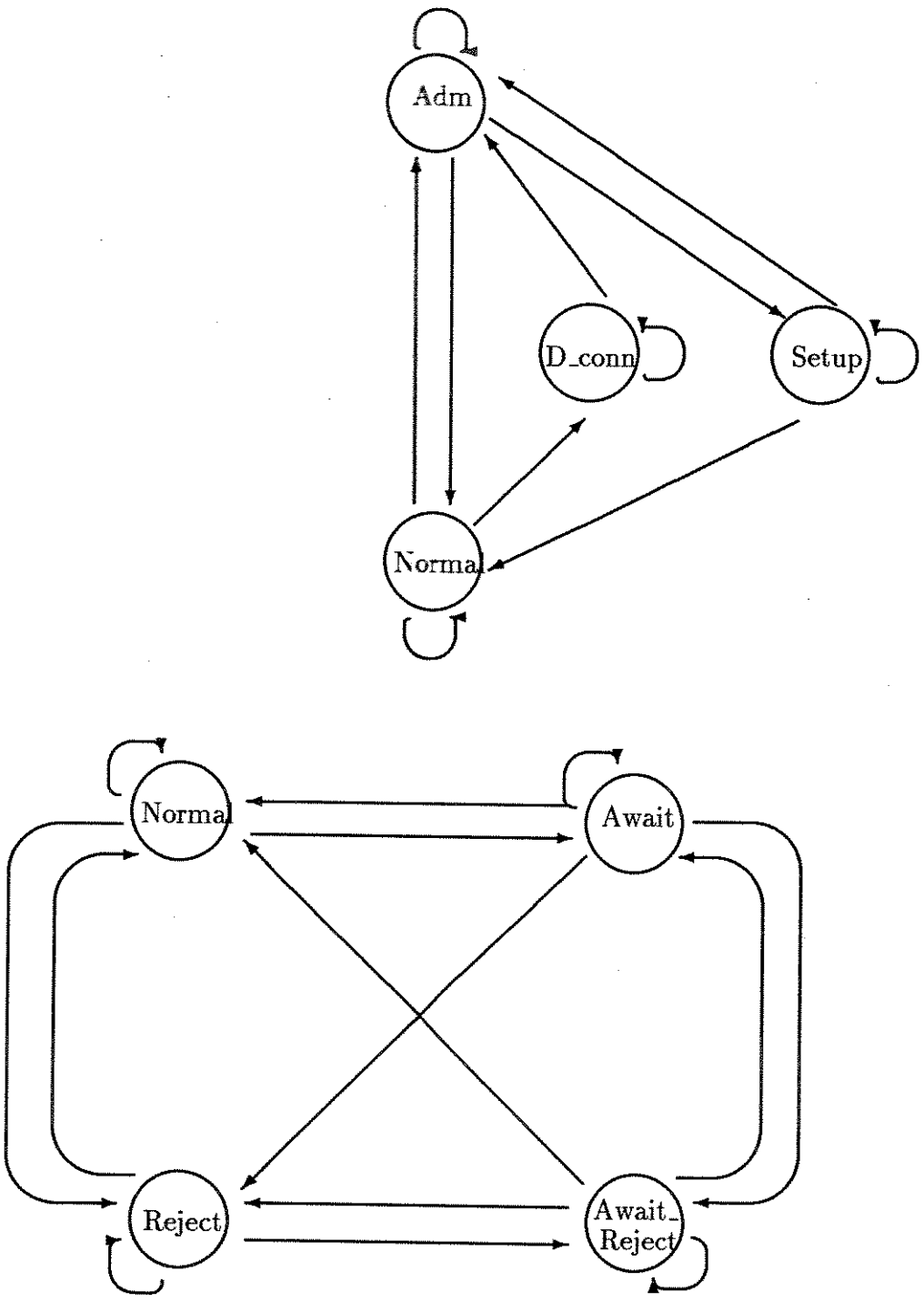


Figura 5.5: Diagrama de estados da LLC - RALFO

5.3.2 Estruturas de dados

Nesta secção serão discutidos as principais estruturas de dados utilizadas para implementar os aspectos mais relevantes de um protocolo de comunicação.

Processos

A camada LLC foi implementada como um processo. Sua função básica é funcionar como um "loop" infinito onde o processo aguarda por uma mensagem e quando recebe uma mensagem, o tratamento do enlace é dado de acordo com seu estado, conforme as especificações do protocolo [CCITT 7]. A seguir será apresentado um trecho do programa que mostra a implementação da LLC:

```
LLC:
PROCESS ();

/* Iniciar a tabela de enlaces da camada LLC                          */

Inicializa_Tabela ();

/* Definicao dos enderecos dos processos que executam camadas vizinhas
   a LLC                                                              */

Proc_APL:= Enderecos(Maquina).APL;
Proc_MAC:= Enderecos(Maquina).MAC;

/* Tratamento dos sinais recebidos                                    */

DO FOR EVER;
  RECEIVE CASE SET Proc_Origem;

/* Servicos solicitados pela camada superior                          */

  ( LCr   IN Lla, Lra, Lsc):
    BEGIN Evento := eLCr; la:= Lla; ra:= Lra; sc:=Lsc; END;
  ( LDCr  IN Lla, Lra, Ll_sdu):
    BEGIN Evento := eLDCr; la:= Lla; ra:= Lra; l_sdu:=Ll_sdu; END;
  ( LDr   IN Lla, Lra):
```

```
    BEGIN Evento := eLDr; la:= Lla; ra:= Lra; END;
  ( LRr   IN Lla, Lra):
    BEGIN Evento := eLRr; la:= Lla; ra:= Lra; END;
  ( LCFCr IN Lla, Lra, La ):
    BEGIN Evento := eLCFCr; la:= Lla; ra:= Lra; a:=La; END;

/*  Servico solicitado pela camada MAC                                  */

  ( MADi IN Lda, Lsa, Lm_sdu, Lrs, Lrsc):
    BEGIN Evento := eMADi; da:= Lsa; sa:= Lda; m_sdu:=Lm_sdu;
          rs:=Lrs; rsc:=Lrsc; END;

  ELSE  PUTFLN(TTY,'ERRO-LLC: Primitiva nao esperada ');
ESAC;

/* Procurar o enlace na tabela de enlaces do processo LLC. Status indica o
   sucesso da operacao.                                                 */

  Procura_Enlace(SSAP,DSAP,Enlace,Status);

/* Caso o enlace nao se encontre na tabela ele pode ser um candidato a en-
   lace                                                                 */

  IF NOT(Status) THEN Enlace.Estado:= ADM;   FI;

/* Tratamento do enlace segundo o seu estado no protocolo LLC          */

   CASE Enlace.Estado OF
   (ADM):    Estado_ADM (Enlace);
   (SETUP):  Estado_SETUP (Enlace);
   (NORMAL): Estado_NORMAL (Enlace);
   (ERROR):  Estado_ERROR (Enlace);
   (RESET):  Estado_RESET (Enlace);
   (D_CONN): Estado_DCONN (Enlace);
   ESAC;

/* Atualizacao do enlace na tabela de enlaces                          */
```

```
   Atualiza_Enlace(SSAP,DSAP,Enlace,Status);
   IF NOT Status THEN Erro (0); FI;

END LLC;
```

A seguir, um trecho do tratamento do estado ADM do protocolo, conforme seu diagrama de transições abaixo:

Cur. State	Event	Action(s)	Next State
ADM	CONNECT_REQUEST	SEND_SABME_CMD(P=X) P_FLAG:=P START_ACK_TIMER RETRY_COUNT:=0 S_FLAG:=0	SETUP
	RECEIVE_SABME_CMD(P=X)	CONNECT_INDICATE SEND_UA_RSP(F=P) V(S):=0 V(R):=0 P_FLAG:=0 REMOTE_BUSY:=0	NORMAL
	RECEIVE_DISC_CMD(P=X)	SEND_DM_RSP(F=P)	ADM
	RECEIVE_XXX_CMD(P=1)	SEND_DM_RSP(F=P)	ADM
	RECEIVE_XXX_CMD(P=0) or RECEIVE_XXX_RSP(F=X)		ADM

```
Estado_ADM:
PROC (Enlace Tipo_Enlace LOC);

DCL
   Status_Cria_Enlace BOOL,
   P Tipo_Flag,
   PDU  Tipo_m_sdu;

 CASE Evento OF
   (eMADi): BEGIN
```

```
            CASE Id_PDU OF
             (SABME): BEGIN
                      IF (Classe_PDU=CMD) AND (PF=B0) THEN
                       IF Aceita_Conexao() THEN /* E' possivel aceitar a
                                                   conexao */
                        /* Criar uma entrada na tabela de enlaces */
                        /* Atualiza a variavel local SSAP */
                        Cria_Enlace(SSAP,DSAP,Enlace,Status_Cria_Enlace);
                        IF Status_Cria_Enlace THEN
                          la.LSAP:=SSAP;           Enlace.VS :=0;
                          Enlace.VR :=0;           Enlace.P_Flag :=B0;
                          Enlace.Remote_busy:= B0; Enviar_LCi();
                          Enviar_UA(RSP,PF);       Enlace.Estado:=NORMAL;
                          Enlace.Subestado:=NONE;
                        ELSE Enviar_DM(RSP,PF);
                        FI;
                       ELSE  /* Nao e' possivel aceitar a conexao */
                        Enviar_DM(RSP,PF);
                       FI;
                      FI;
                      END;
               (DISC): BEGIN;
                      IF Classe_PDU=CMD THEN Enviar_DM(RSP,PF);
                      ELSE Erro(1); FI;
                      END;
               ELSE   BEGIN;
                      IF (Classe_PDU=CMD) AND (PF=B1)
                      THEN Enviar_DM(RSP,PF);
                      ELSE Erro(1);
                      FI;
                      END;
             ESAC;
            END;
   ELSE ;
 ESAC;
Estado_ADM;
```

Primitivas

Cada primitiva do protocolo, possui um conjunto de parâmetros associados. Uma vez que a linguagem CHILL implementa Sinal como um conjunto de valores a serem transmitidos, para realizar a comunicação entre processos, define-se cada primitiva como sendo um sinal, e os parâmetros das primitivas formam o conjunto de valores transmitidos por um sinal.

Para ilustrar esta idéia, serão mostrados trechos da implementação, onde são definidos os tipos de dados utilizados para a definição dos parâmetros e a definição dos sinais (primitivas):

```
/*=============================================================================*/
/*                          DEFINICAO DE TIPOS                                 */
/*(Definicoes das estruturas de dados dos parametros das primitivas LLC e MAC)*/
/*=============================================================================*/

SYNMODE

/* Tipo LSAP define o tipo utilizado para representar um SAP da camada LLC    */

  Tipo_LSAP     = SHORT_CARDINAL (0:255),

  Tipo_Endereco_LLC = STRUCT (LSAP Tipo_LSAP,
                              MSAP Tipo_MSAP),

/* Os tipos la,ra sao utilizados nas primitivas oferecidas pela camada LLC    */
/* e segundo especificacoes do protocolo LLC devem refletir de algum modo os  */
/* enderecos SAP das duas camadas LLC e MAC. Como definimos 8 bits para cada  */
/* um dos tipos de SAP, seu tipo devera' ser dois bytes.                      */

  Tipo_la       = Tipo_Endereco_LLC, /* la: Local address */
  Tipo_ra       = Tipo_Endereco_LLC, /* ra: Remote address */

/* O tipo l_sdu define uma PDU da camada acima da LLC. Sera' o campo de       */
/* de informacao da PDU da camda LLC e foi definido com este tamanho,         */
/* (8 bits) seguindo recomendacoes de performance em  estudos do funcio-      */
/*  namento da camada MAC.                                                    */
```

```
  Tipo_l_sdu    = BIT (16),

/* Parametros das primitivas da camada LLC                                  */

  Tipo_sc       = BIT (8),    /* sc : Service class */
  Tipo_s        = BIT (8),    /* s : Status */
  Tipo_r        = BIT (8),    /* r: Reason for disconnection, reset */
  Tipo_a        = BIT (8),    /* a: Amount of data the LLC is permitted to
                                  pass */
/* Tipo MSAP define o tipo utilizado para representar um SAP da camada MAC  */

  Tipo_MSAP     = SHORT_CARDINAL (0:255),

/* Os tipos da,sa serao usados para identificar a camada MAC de cada ma-    */
/* quina. Como cada maquina so' pode ter uma MAC, estes enderecos tambem    */
/* identificam as proprias maquinas.                                        */

  Tipo_da       = Tipo_MSAP,    /* da: Destination address */
  Tipo_sa       = Tipo_MSAP,    /* sa: Source address */

/* O tipo m_sdu define uma PDU da camada LLC. Seu campo de informacao e'    */
/* composto de 16 bits, seguindo recomendacoes do estudo de performance     */
/* da camada MAC                                                            */

  Tipo_m_sdu    = STRUCT (DSAP,SSAP BIT (8),
                          Controle BIT (16),
                          Info BIT (16) ), /* Segundo definicao do Gorgonio*/

/* Parametros das primitivas da camada MAC                                  */

  Tipo_rsc      = BIT (8),    /* rsc: Requested service class */
  Tipo_rs       = BIT (8),    /* rs: Reception status */
  Tipo_ts       = BIT (8),    /* ts: Transmission status */
  Tipo_psc      = BIT (8);    /* psc: Provided service class */
```

```
/*=============================================================================*/
/*                        DEFINICAO DAS PRIMITIVAS                             */
/* (Especificacao dos servicos da LLC e MAC atraves de sinais da linguagem     */
/*                                 CHILL)                                      */
/*=============================================================================*/

/* Servicos:       */
/*  LC    : L_Connect              */
/*  LDC   : L_Data_Connect         */
/*  LD    : L_Disconnect       */
/*  LR    : L_Reset        */
/*  LCFC  : L_Connection_FlowControl       */
/*  MAD   : MA_Data        */

/* Tipos de primitivas associadas a cada servico:       */
/*  r : request - passada do usuario N para a camada N       */
/*  i : indication - passada da camada N para o usuario N, este evento
*/
/*                   esta' relacionado com uma requisicao de servico remoto   */
/*  c:  confirm - passada da camada N para o usuario N, este evento       */
/*       fornece os resultados de uma requisicao anterior       */

/* Definicao das primitivas                                                    */
SIGNAL
/* Servicos oferecidos pela LLC a camada superior           */
  LCr    <> SID = 01 <> =( Tipo_la, Tipo_ra, Tipo_sc),
  LDCr   <> SID = 02 <> =( Tipo_la, Tipo_ra, Tipo_l_sdu),
  LDr    <> SID = 03 <> =( Tipo_la, Tipo_ra),
  LRr    <> SID = 04 <> =( Tipo_la, Tipo_ra),
  LCFCr <> SID = 05 <> =( Tipo_la, Tipo_ra, Tipo_a),
/* Servicos requisitados pela LLC para a camada superior               */
  LCi    <> SID = 06 <> =( Tipo_la, Tipo_ra, Tipo_s, Tipo_sc),
  LCc    <> SID = 07 <> =( Tipo_la, Tipo_ra, Tipo_s, Tipo_sc),
  LDCi   <> SID = 08 <> =( Tipo_la, Tipo_ra, Tipo_l_sdu),
  LDCc   <> SID = 09 <> =( Tipo_la, Tipo_ra, Tipo_s),
  LDi    <> SID = 10 <> =( Tipo_la, Tipo_ra, Tipo_r),
  LDc    <> SID = 11 <> =( Tipo_la, Tipo_ra, Tipo_s),
```

```
  LRi   <> SID = 12 <> =( Tipo_la, Tipo_ra, Tipo_r),
  LRc   <> SID = 13 <> =( Tipo_la, Tipo_ra, Tipo_s),
  LCFCi <> SID = 14 <> =( Tipo_la, Tipo_ra, Tipo_a),
/* Servico requisitado pela LLC para a camada MAC                    */
  MADr  <> SID = 15 <> =( Tipo_da, Tipo_m_sdu, Tipo_rsc),
/* Servicos oferecidos pela LLC a camada MAC        */
  MADi  <> SID = 16 <> =( Tipo_da, Tipo_sa, Tipo_m_sdu,  Tipo_rs,  Tipo_rsc),
  MADc  <> SID = 17 <> =( Tipo_ts, Tipo_psc);
```

Comunicação entre processos

A comunicação entre processos é feita através de envio de sinais entre processos. A sincronização se dá através dos comandos SEND e RECEIVE, conforme explicado no capítulo anterior. O uso do comando RECEIVE é feito no corpo principal do processo que implementa a LLC. O envio de sinais pode ser visto no trecho do programa, a seguir, que implementa o envio de um sinal MADr (MA_Data.request) para transmitir a PDU SABME:

```
Enviar_SABME: /* Quadro tipo U */
PROC (CR Tipo_Classe_PDU, PF Tipo_Flag);
  DCL PDU Tipo_m_sdu;

  PDU.DSAP:= Converte_INT_BIN(DSAP); PDU.SSAP:= Converte_INT_BIN(SSAP);
  IF CR=RSP THEN PDU.SSAP := PDU.SSAP OR B'1000_0000'; FI;
  IF PF=B1 THEN PDU.Controle:=B'1111_1110_0000_0000';
           ELSE PDU.Controle:=B'1111_0110_0000_0000';
  FI;
  PDU.Info:= (16)B'0';
  SEND MADr (da, PDU, Prioridade) TO  Proc_MAC;
END Enviar_SABME;
```

Tabela de enlaces

Para implementar a tabela de enlaces gerenciados pelo protocolo LLC, foi definida uma tabela, onde cada elemento é um conjunto de informações necessárias para o

protocolo, relativos a um determinado par de SAPs (SSAP, DSAP). A seguir, segue um trecho da implementação da tabela de enlaces:

```
/* Estruturas de dados utilizadas na definicao do enlace                    */

SYNMODE
 Tipo_Flag = BIT(1),
 Tipo_Count = SHORT_CARDINAL(0:127);

NEWMODE
 Tipo_Estado = SET (ADM, SETUP, NORMAL, RESET, ERROR, D_CONN),
 Tipo_Subestado = SET (AWAIT, AWAIT_BUSY, AWAIT_REJECT, REJECT,
                       BUSY, NONE),

/* Definicao do enlace                                                      */
/* Obs: A informacao do SSAP (local) sera' o proprio indice da tabela de en-
        laces                                                               */

 Tipo_Enlace = STRUCT ( Alocado       BOOL,          /* Indica se existe enlace
                                                        entre o SAP (indice) e
                                                        um DSAP remoto  */
                        SAP_Remoto    Tipo_LSAP,     /* DSAP remoto  */
                        Estado        Tipo_Estado,   /* Estado do enlace */
                        Subestado     Tipo_Subestado, /* Subestado do enlace */
                        VS            Tipo_Count,    /* Variavel do prot. */
                        VR            Tipo_Count,    /* Variavel do prot. */
                        P_Flag        Tipo_Flag,     /* Variavel do prot. */
                        S_Flag        Tipo_Flag,     /* Variavel do prot. */
                        Retry_count   Tipo_Count,    /* Variavel do prot. */
Cause_Flag    INT,            /* Variavel do prot. */
                        Remote_busy   Tipo_Flag);    /* Variavel do prot. */

/* Definicao do da tabela de enlaces da camada LLC                          */

SYN
 MAX_Enlaces = 15;
```

```
SYNMODE
  VAL_Enlaces = SHORT_CARDINAL(0:MAX_Enlaces);

DCL
 Tabela_Enlaces ARRAY (VAL_Enlaces) Tipo_Enlace, /* la de enlaces da LLC */
```

O conjunto de informações que define um enlace são o SSAP (no caso é o próprio índice da tabela) e DSAP (SAP_Remoto). O campo Alocado indica se determinado SAP local está alocado para um enlace corrente. As demais informações estão relacionadas ao protocolo: Estado (estado do protocolo LLC para um determinado enlace); Subestado (no caso do enlace estar no estado NORMAL); VS (variável que indica a numeração do próximo quadro I a ser enviado); VR (variável que indica a numeração do próximo quadro I a ser recebido); P_Flag (indica o valor do bit P/F); S_Flag (indica o valor do bit S); Retry_count (contador de retransmissões); Cause_Flag (indica a situação durante a fase de transferência) e Remote_busy (indica se a LLC remota está no estado BUSY).

Janela de quadros I

Para cada enlace da Tabela de enlaces, existe uma janela de quadros I associada. Sua função é armazenar quadros I enviados a LLC remota e que aguardam a confirmação de recebimento. Cada janela é definida como sendo um vetor circular com vários apontadores (Fig. 5.6): O número de posições deste vetor depende do parâmetro (K) da LLC.

- Ultimo_confirmado Aponta para o último quadro confirmado pela camada LLC remota;
- Proximo_livre Aponta para a próxima posição livre na janela de quadros;
- Ultimo_reenviado Aponta para o próximo quadro a ser remandado;

Os procedimentos relacionados a manipulação da janela de dados são:

- Inicializa_Janelas: Inicializa todos os vetores de quadros I, de cada enlace, como vetores vazios.

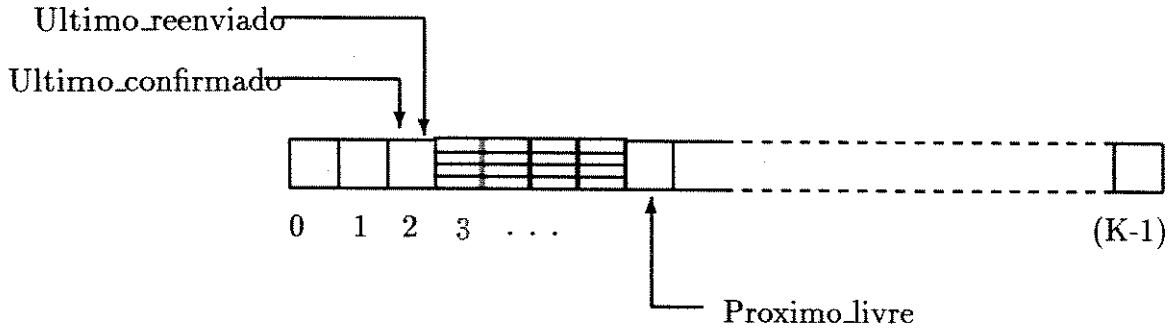


Figura 5.6: Janela de quadros I de um enlace

- Aguardar_confirmacao: Armazenar um quadro I na janela de quadros correspondente ao enlace, que foi enviado a LLC remota e aguarda a confirmação de recebimento.
- Reenviar_quadro_I : Reenvia a camada LLC remota, um quadro I (indicado pelo apontador Ultimo_reenviado) da janela de quadros correspondente ao enlace.
- Confirmar_quadros_I: Confirma N quadros I enviados, correspondente a um retorno recebido da camada LLC remota, através da variável NS. Os quadros são eliminados da janela.

Temporizadores

Na implementação do protocolo LLC para a RALFO, foram utilizados três temporizadores:

- P_timer;
- ACK_timer;
- REJ_timer.

Eles foram implementados da seguinte maneira: existe um processo (Temporizador) que é responsável pelas temporizações requisitadas pelo processo LLC. Cada

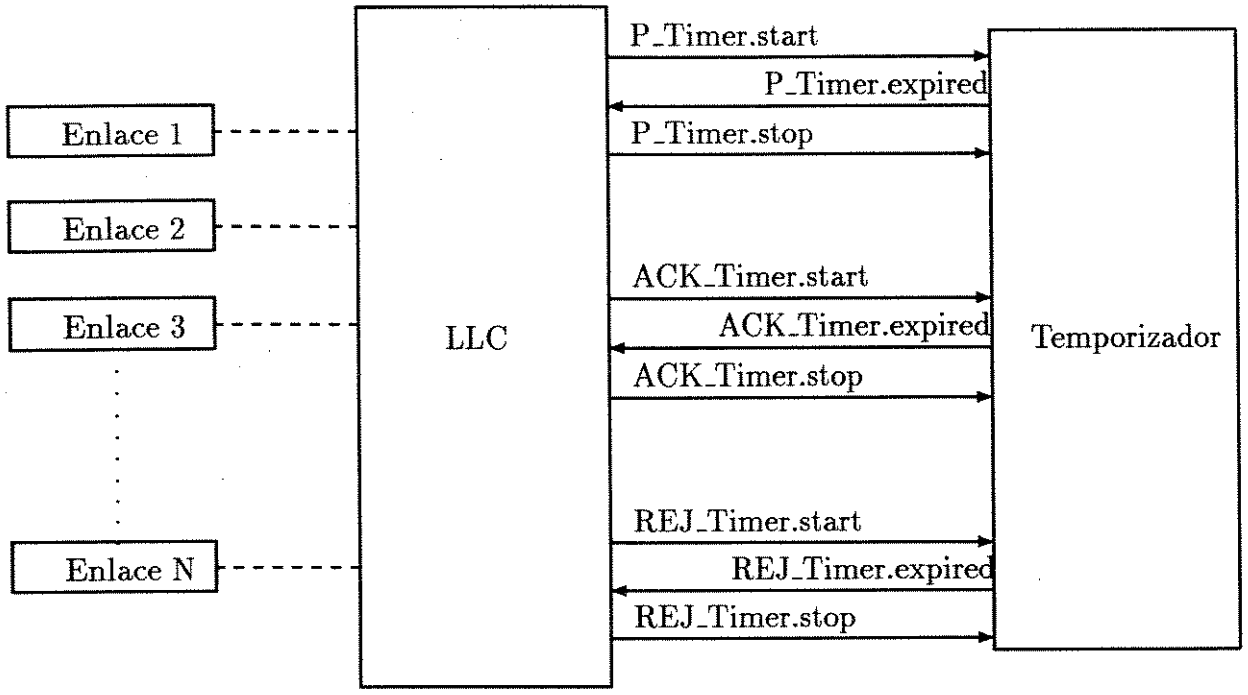


Figura 5.7: Funcionamento do processo Temporizador

vez que a LLC solicita uma das três temporizações, o processo Temporizador, envia periodicamente (o intervalo de tempo é parâmetro do protocolo LLC) um sinal, indicando o vencimento do temporizador correspondente. Isto se repete até que a LLC solicite o cancelamento de uma temporização.

Cada temporizador possui três sinais associados: start, stop, expired. O sinal start é enviado da camada LLC para o Temporizador para solicitar o início da temporização. O sinal stop é enviado da camada LLC para o Temporizador para solicitar o término da temporização. O sinal expired é enviado pelo processo Temporizador à camada LLC, uma vez ativada a temporização e a cada vencimento do temporizador correspondente, até que este processo receba um pedido de término de temporização (stop). O funcionamento deste processo pode ser ilustrado na Fig. 5.7.

Para possibilitar a identificação do enlace a ser tratado, cada sinal tem como parâmetros o par (DSAP, SSAP) para identificar o enlace tratado.

O módulo temporização pode ser definido como sendo um conjunto de estruturas de dados relacionados aos temporizadores e um conjunto de procedimentos de manipulação:

As estruturas de dados são:

- Sinais relativos ao temporizador P_Timer: P_Timer.start, P_Timer.stop, P_Timer.expired;
- Sinais relativos ao temporizador ACK_Timer: ACK_Timer.start, ACK_Timer.stop, ACK_Timer.expired;
- Sinais relativos ao temporizador REJ_Timer: REJ_Timer.start, REJ_Timer.stop, REJ_Timer.expired;
- Lista de enlaces temporizados por P_Timer;
- Lista de enlaces temporizados por ACK _Timer;
- Lista de enlaces temporizados por REJ_Timer;

Os procedimentos associados são:

- Inicializar_Temporizadores: Inicializa todas as listas de enlaces temporizados como listas vazias;
- Ativar_Temporizador (Parâmetros: Enlace, Temporizador): Este procedimento é utilizado pela LLC para solicitar a temporização de um determinado temporizador a um dado enlace;
- Cancelar_Temporizador (Parâmetros: Enlace, Temporizador): Este procedimento é utilizado pela LLC para solicitar o cancelamento de uma determinada temporização de um dado enlace;

O processo Temporizador funciona como um "loop" infinito que aguarda por sinais do tipo start ou stop, relacionados aos temporizadores. Uma vez recebido um pedido de temporização (start), ele insere o enlace na lista de enlaces, correspondente a um temporizador, e envia a cada vencimento do temporizador um sinal a LLC, indicando a expiração. Uma vez recebido um pedido de cancelamento de temporização (stop), ele cancela o envio de sinais (expired) a LLC e retira o enlace da lista de enlaces, correspondente a um temporizador.

A interface deste módulo para o ambiente externo será composta de

- Sinais relativos ao temporizadores;
- Procedimentos associados

Assim, as demais estruturas de dados e a implementação ficam restritas ao módulo e preservadas neste escopo.

Como pode-se observar, a implementação dos temporizadores fica bem facilitada devido ao suporte da linguagem CHILL a sinais temporizados, e a modularização, que garante a independência dos programas implementados.

5.3.3 Parâmetros do protocolo LLC

- Temporizadores: Acknowledgement Timer, P-bit Timer e Reject Timer têm seus tempos de vencimento estabelecidos no início da ativação do processo LLC;
- Número máximo de transmissões (N2): Estabelecido em 10;
- Número máximo de octetos em uma PDU I (N1): Foi fixado em 2 octetos, de acordo com os trabalhos da tese de G.B. Araújo [Araujo 3]. ;
- Número máximo de PDUs I pendentes (K): Estabelecido como sendo 20;
- Número mínimo de octetos em uma PDU: 3 ou 4 dependendo do tipo da PDU: U (3), S (4) e I (4).

5.4 Testes de funcionamento

Para testar o funcionamento da camada LLC um ambiente de teste foi definido no próprio PP/SOP, conforme esquematizado pela Fig. 5.8. Para isto, foi implementado parcialmente as camadas Aplicação e MAC,considerando principalmente as suas interfaces de comunicação com a camada LLC. A camada Aplicação foi implementada apenas com a função de utilizar as primitivas da camada LLC, para transportar dados. A camada MAC não foi completamente implementada na sua funcionalidade, ela apenas recebe uma primitiva de uma camada LLC e envia a outra LLC par. Ou seja, ela está servindo como um "curto-circuito" para possibilitar os testes da camada LLC.

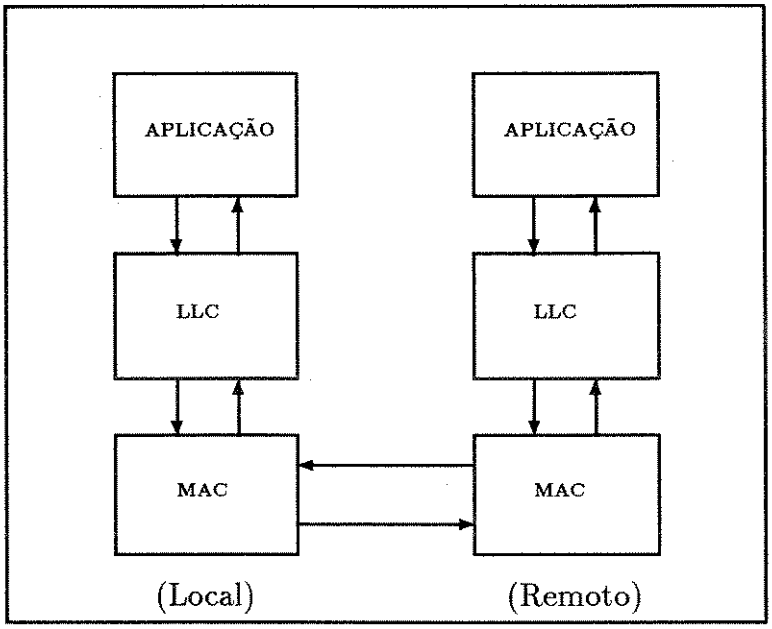


Figura 5.8: Ambiente de teste do protocolo LLC

Para simular dois nós em comunicação foram ativados dois processos para cada camada: um fazendo o papel de uma camada no nó local, e o outro, no nó remoto. Como o hardware e o software que implementam o controle de acesso ao meio físico não estavam prontos, foi estabelecido que a camada MAC local se comunicaria diretamente com a MAC remota. Assim, através das seis camadas em execução: Aplicação local e remota, LLC local e remota, MAC local e remota; foi possível testar todo o funcionamento da camada LLC sem ter que esperar pelo desenvolvimento das demais camadas.

5.5 Conclusões

Neste capítulo foi apresentada a implementação do protocolo LLC na RALFO. Inicialmente foi detalhada a especificação da LLC, segundo a recomendação 802.2: serviços, PDUs, diagramas de estados e diagramas temporais. Para iniciar os trabalhos de implementação, foi feita uma simplificação do protocolo: foram eliminados os estados RESET e ERROR.

Para ilustrar a utilização da linguagem CHILL, trechos do protocolo são discutidos mostrando a definição das primitivas, o tratamento de cada estado da LLC bem como a implementação dos temporizadores. São detalhadas também, a implementação dos enlaces e da janela de dados de cada enlace.

Para verificar a execução do protocolo foi definido um ambiente de testes conforme ilustrado na Fig. 5.8. Assim, a invocação do processo LLC em duas instâncias, simulando dois nós da rede, permitiu testar a operabilidade do protocolo, sem a necessidade da utilização de simuladores de protocolo.

No próximo capítulo, será abordada a análise de desempenho do protocolo implementado, seus modelos e seus resultados.

Capítulo 6

Análise de desempenho da comunicação de dados na RALFO

6.1 Introdução

Definido o ambiente de desenvolvimento de software e implementado o protocolo LLC da rede RALFO, a próxima atividade realizada foi a análise de desempenho do protocolo implementado. Este é o assunto deste capítulo que se inicia com uma abordagem geral da modelagem de protocolos. Em seguida, são apresentados os modelos de análise elaborados para a LLC e os resultados obtidos da análise realizada. Os resultados desta análise são apresentados através de gráficos do tamanho da fila de mensagens da camada LLC, em função do "throughput" da fila.

6.2 Modelagem dos protocolos de comunicação

A estrutura de camadas para protocolos de comunicação, proposta segundo o modelo de referência OSI/ISO, sugere que cada camada seja implementada como uma tarefa independente que se comunica logicamente com sua camada par, através da troca de primitivas entre suas camadas vizinhas: superior e inferior. A nível de modelamento de filas, isto equivale dizer que uma camada representa uma fila que recebe elementos das filas das camadas vizinhas, e que realimenta estas mesmas filas. Cada fila, conforme o protocolo de comunicação da camada que representa, estabelece as porcentagens de realimentação para as filas vizinhas.

Portanto, inicialmente o esquema das filas pode ser apresentado da maneira

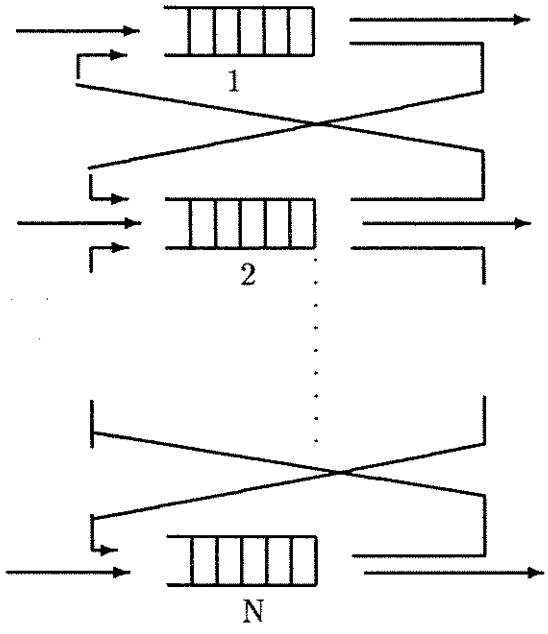


Figura 6.1: Esquema de filas para protocolos de comunicação

descrita pela Fig. 6.1. Para modelar o atendimento das filas é necessário considerar o ambiente de execução utilizado na implementação das camadas.

Para suportar a implementação em camadas independentes que se comunicam trocando mensagens, cada camada pode ser implementada em um processador próprio, onde existe a comunicação entre processadores para a troca de mensagens, ou várias camadas podem ser implementadas em um único processador que suporte um ambiente operacional multitarefa e concorrente. Em todos os casos, o método de escalonamento dos processos vai determinar a disciplina de atendimento das filas para modelamento do servidor de mensagens.

6.2.1 Múltiplos servidores

No caso de cada camada ser executada em um processador dedicado, existirá um servidor de fila específico para cada fila. A taxa de chegada de cada fila deverá levar em conta o tempo gasto para um processador entregar uma mensagem a outro

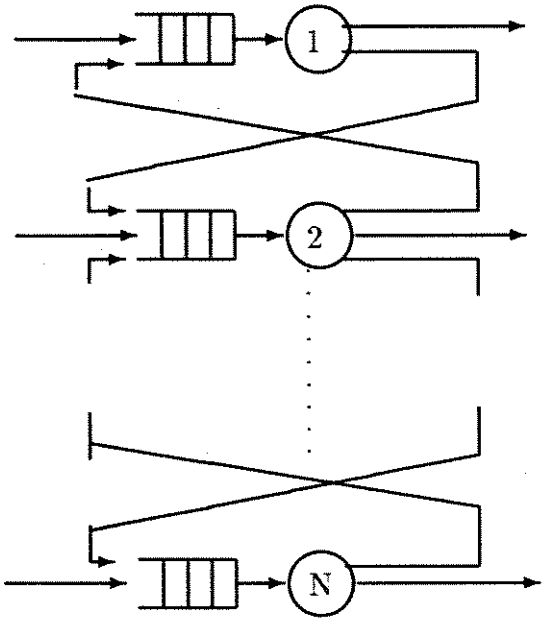


Figura 6.2: Múltiplos servidores

processador. O modelo de filas pode ser representado da maneira descrita pela Fig. 6.2.

Este modelo pode ser analisado pelos resultados do teorema de Jackson [Stuck 45], uma vez que as probabilidades de realimentação são determinadas pelo protocolo de cada camada.

6.2.2 Único Servidor

Para o caso onde várias camadas são executadas por um único processador, a nível de modelamento, existe um único servidor para atender várias filas. Neste caso, a política de escalonamento dos processos determinará a disciplina de atendimento do servidor. O modelo de fila pode ser representado pela maneira descrita pela Fig. 6.3. Esta política de escalonamento é imposta pelo sistema operacional e pelo tratamento das mensagens, dado na implementação dos protocolos.

Por exemplo, suponha que a CPU atenda cada processo exaustivamente, no que se refere às mensagens aguardando atendimento. Neste caso é possível aplicar os

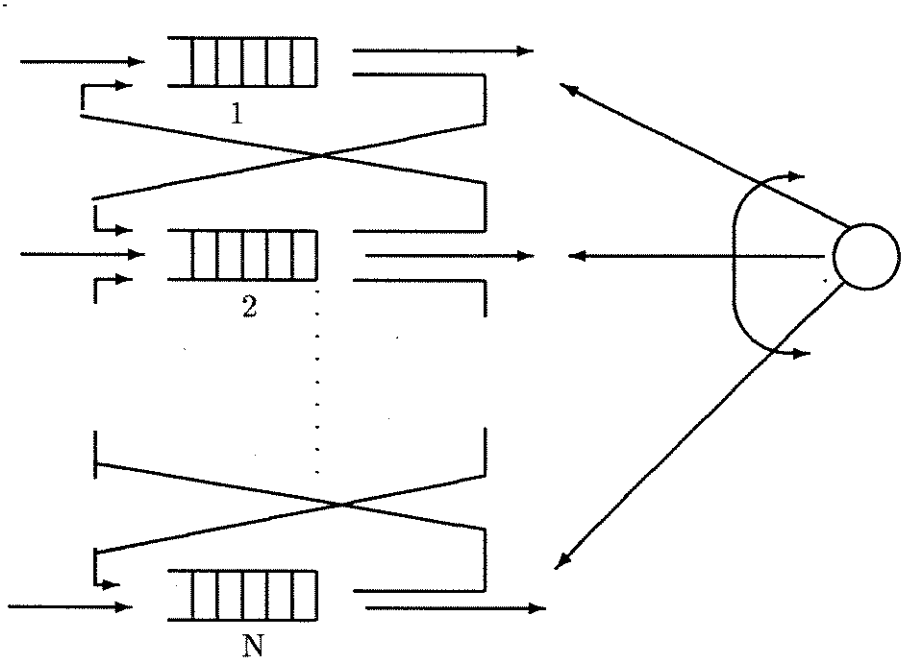


Figura 6.3: Único Servidor

resultados da análise de desempenho de filas com atendimento "Polling" exaustivo [Hammond 22].

6.3 Modelagem da LLC na RALFO

São apresentados nesta seção, os modelos para avaliação de desempenho da comunicação de dados da RALFO. Inicialmente o sistema é analisado em detalhes, para em seguida, apresentação do modelo simplificado, cuja solução analítica foi possível. Finalmente, um modelo aprimorado de simulação, definido e analisado através de um software da IBM chamado RESQ (Research Queueing Package) [Loewner 36].

6.3.1 Descrição do sistema

O processador principal do PP acomoda a comunicação de dados, ou seja, as camadas APLICAÇÃO, LLC e parte da MAC. Conforme mencionado em [SOP 44],

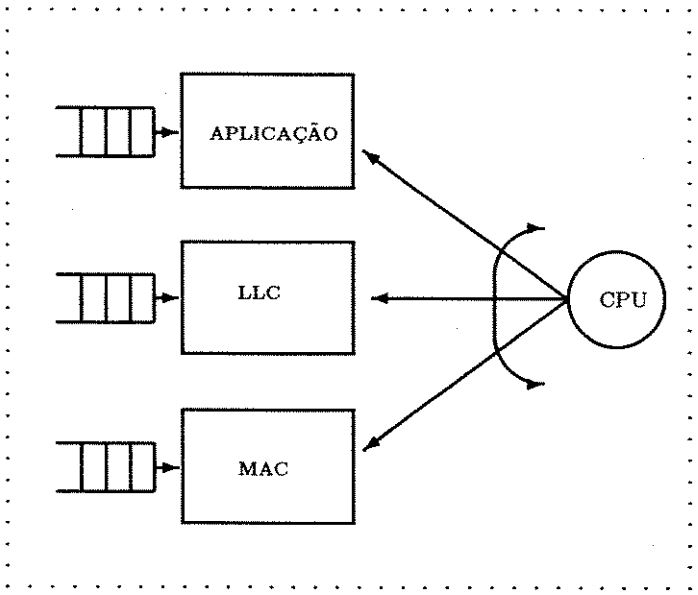


Figura 6.4: Sistema analisado

o PP/SOP é um sistema multitarefa que suporta processamento em tempo real e "time-sharing", conforme a classe dos processos em execução. Na implementação feita os três processos pertencem a classe C [SOP 44], portanto o processador é compartilhado de maneira "time-sharing". Como a comunicação de dados entre os processos é feita através da troca de sinais (da linguagem CHILL), cada processo tem uma fila de sinais associada, onde ficam armazenadas as mensagens recebidas. Pode-se imaginar o processador como mostra a Fig. 6.4.

A política de escalonamento dos processos pode ser descrita pelo diagrama de estados e transições da Fig. 6.5 ([SOP 44], [Sobrinho 43]). Um processo pode estar em um dos seguintes estados:

- NI (Não iniciado): O processo não foi iniciado.
- AT (Ativo): O processo está esperando ser colocado em execução.
- EX (Execução): O processo está sendo executado pelo processador ou momentaneamente interrompido por pseudo-processo.
- SU (Suspenso): O processo está suspenso.

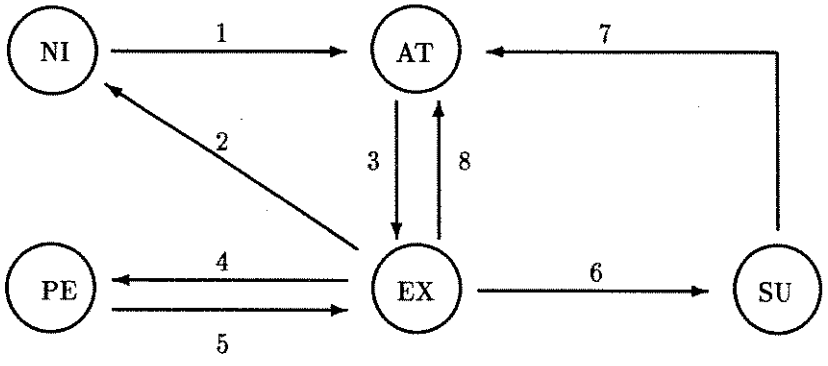


Figura 6.5: Estado e transições dos processos

- PE (Pendente): A execução do processo foi interrompida para permitir a execução de processos de classes mais prioritárias. Somente processos de classe B ou C podem estar neste estado.

As transições do estado de um processo ocorrem conforme os seguintes eventos:

1. Iniciação do processo.
2. Terminação do processo.
3. O processo é colocado em execução.
4. Um processo de mais alta prioridade é colocado em execução.
5. Volta um processo interrompido pela ação 4.
6. Um processo é suspenso.
7. Termina a causa de suspensão de um processo.
8. Execução interrompida por divisão no tempo. Válida apenas para processos da classe C.

O SO/P **interrompe** um processo da classe B ou C quando executar a função de escalonamento e detectar a presença de pelo menos um processo de classe mais prioritária em estado ativo. Este controle é feito unicamente nos seguintes casos:

- interrupção de relógio.
- fim de um pseudo-processo (driver).

O processo é **suspenso** nos seguintes casos:

- execução de uma primitiva de espera de sinais quando não há sinais na fila do processo, nem tratamento de exceção.
- execução de uma primitiva que provoque a espera de um evento.
- execução de uma ação de E/S em casos onde a operação não pode ser executada direta e imediatamente.
- acesso a uma região ocupada.
- auto suspensão temporizada.
- carregamento e execução de um programa com espera de fim de execução.

Um processo é **ativado**, colocado em uma das filas de processos ativos, nos seguintes casos:

- iniciação do processo;
- chegada de um sinal para um processo que estava esperando por ele;
- execução de uma primitiva (CONTINUE[1]), para um processo suspenso pela primitiva DELAY;
- liberação de uma região é para um processo aguardando por isso;
- término da execução de uma operação de E/S de um processo;
- vencimento de uma temporização de um processo que a solicitou.

[1]CONTINUE, DELAY são primitivas da linguagem CHILL [CHILL 9]

Tratamento de interrupção

O seguinte algoritmo simplificado é executado na análise de reescalonamento provocada por interrupção:

```
INT:

    IF processo classe A
      continua execucao
    ENDIF

    IF processo classe B
      procura processo ativo classe A
    ELSE
      IF processo classe C
        procura processo ativo classe A ou B
      ENDIF
    ENDIF

    IF achou processo
      muda estado do processo EX  para PE
      muda estado do processo achado de AT para EX
      executa novo processo
    ELSE
      continua execucao
    ENDIF
```

Tratamento de suspensão

O seguinte algoritmo simplificado é executado na análise de reescalonamento provocada por suspensão:

```
REG:

     procura processo ativo classe A, B, C
```

```
 IF achou
  muda estado do processo AT para EX
  executa novo processo
ENDIF

  carrega contexto completo do SO/P

GOTO SEG
```

Divisão de tempo

O SO/P executa os processo classe C por divisão de tempo, isto é, um processo que estiver executando por um tempo pré-determinado na iniciação do sistema operacinal, é interrompido o seu estado muda de EX para AT para permitir a execução de outros processos ativos com prioridade maior ou igual.

A frequência aconselhada para esta operação é divisão no tempo a cada 160-240 milisegundos, ou seja, o "quantum" de CPU dado a um processo é de 240 milisegundos. Com este valor a carga da CPU para implementar este tipo de escalonamento é inferior a 0.3%.

Portanto, para efeitos de análise, pode-se considerar que o SOP é um sistema operacional "time-sharing", onde um "quantum" é o tempo necessário para o atendimento de uma mensagem.

6.3.2 Escalonamento "Time-Shared"

O método de escalonamento de tarefas "time-shared" foi proposto por Fernando Corbató no final dos anos 50. Desde que tornou-se comprovada sua praticabilidade, pelos resultados obtidos pelo projeto MAC (MIT), o assunto foi muito explorado e estudado.

Existem muitas pesquisas baseadas em análises de modelos analíticos para estes sistemas. J. M. McKinney em [McKinney 38] apresenta os principais resultados encontrados até junho de 1969, procurando diferenciar os parâmetros que distinguem os diversos modelos. Uma extensa e comentada bibliografia é oferecida neste artigo.

O objetivo dos sistemas "time-shared" é oferecer um ambiente concorrente onde vários usuários podem compartilhar os recursos de um equipamento, mas ter a

impressão de estar usando o equipamento de maneira exclusiva.

Internamente o sistema operacional funciona da seguinte maneira: as tarefas a serem executadas (processos dos usuários) são colocados em uma fila organizada por ordem de chegada de seus elementos. O processador executa a primeira tarefa durante um período de tempo denominado "quantum". Se durante este período a tarefa termina, o processador passa a servir a próxima tarefa da fila. Caso contrário, a tarefa é suspensa e volta para o final da fila aguardando por um novo atendimento.

O tempo da CPU gasto em escalonar os contextos das tarefas em execução é denominado "swapping-time" e constitui uma sobrecarga no processamento das tarefas, prejudicando mais aquelas com curta duração de tempo.

Os modelos matemáticos para análise de sistemas "time-shared" são estocásticos e baseiam-se nos resultados da Teoria de Filas. Os parâmetros que diferem os modelos já estudados são:

- Número de canais de entrada
- Número de CPUs
- Tipo do processo de chegada (distribuição de probabilidade: Poisson, Bernoulli,etc)
- Tipo do processo de atendimento (distribuição de probabilidade: Exponencial, Geométrica, etc)
- Consideração dos tempos de "overhead" ("swapping")
- Valor do "quantum" (sua função de distribuição)
- Disciplina de atendimento ("round-robin", prioridades, preemptivo, não-preemptivo)

Modelo "Round-Robin Preemptivo "

Neste modelo existe uma única fila para o processador. O processo de chegada de tarefas na fila é um processo de entrada estocástico descrito pela função de distribuição dos intervalos de chegada, denominada A(t). A medida que as tarefas vão chegando, elas são colocadas no final da fila. O processo de atendimento é descrito pela função de distribuição de probabilidade B(s), do tempo de serviço necessário para executar um job.

A disciplina de atendimento estabelecida é que a primeira tarefa que chega ao sistema é a primeira a ser servida (FIFO). À tarefa a ser servida é fornecido o serviço por um determinado intervalo de tempo, "quantum". Se a tarefa se completa durante este intervalo de tempo, o processador passa a atender a próxima tarefa da fila. Caso contrário, a tarefa é ou interrompida (preemptivo) e volta ao final da fila.

Em um dos seus artigos [Kleinrock 34], Kleinrock e Coffman expõem seus resultados (cálculo do tempo médio de espera de uma tarefa que necessite de k "quantum") para duas situações particulares:

- processo de chegada com distribuição geométrica e tempo de serviço com distribuição geométrica .
- processo de chegada Poissoniano e tempo de atendimento exponencialmente distribuído.

Aplicação no caso de protocolos

A avaliação da performance de um protocolo implica em dimensionar o tamanho da fila de mensagens de cada camada, bem como o tempo de espera de cada mensagem. Para aplicar os resultados até agora descritos, no caso de filas de protocolos, primeiramente existe uma diferença fundamental: o número de tarefas no processador é fixo e interessa o tempo médio de espera de uma mensagem e não de uma tarefa. Abstraindo idéias, poderia ser considerando que cada mensagem é uma tarefa com necessidades de K "quantum". Ainda assim, existe um conflito no atendimento das mensagens. Suponha que existam três camadas A, B e C e que todas as camadas estejam inicialmente com suas filas de mensagens vazias. Imagine que cheguem trêm mensagens, primeiro em A, depois em B e por último em C. O atendimento das mensagens, uma vez que depende do escalonamento, é aleatório, ou seja, não pertence a nenhuma disciplina conhecida: FIFO ("First in, first out"), LIFO("last in, first out), Prioridades, Menor tempo de execução, etc..

Logo, não é possível aplicar os resultados do escalonamento "time-sharing" para escalonamento de filas de protocolos executadas em ambiente "time-sharing". Na realidade o processo de atendimento das mensagens, dentro de cada fila, é FIFO, porém o atendimento das filas é "time-sharing".

Para a elaboração de um modelo para a RALFO, inicialmente foram feitas simplificações no sistema analisado que permitiram a definição de um modelo com solução analítica. Posteriormente foi elaborado um modelo de simulação, uma vez que não foi encontrado um modelo analítico que se adequasse ao sistema real.

6.3.3 Modelo analítico

Para este modelo, foi considerado que um sistema "time-sharing"com N tarefas pode ser analisado como se cada processo possuísse seu próprio processador, porém com uma capacidade de atendimento $\mu = \frac{\mu_r}{N}$, onde μ_r é a capacidade total do processador real. Foram feitas as seguintes suposições:

- Cada processo possui um processador imaginário com uma taxa de serviço igual a 1/3 da taxa de serviço do processador real;
- Só será considerada a comunicação em um sentido ("Simplex");
- A taxa de chegada das mensagens nas filas LLC e MAC são dependentes da taxa de entrada da fila APLICAÇÃO.
- O fato da camada LLC receber uma mensagem e ao processá-la poder, dependendo do caso, originar 0,1 ou 2 mensagens foi simplificado por um procedimento de realimentação com probabilidades associadas.

O modelo, ilustrado na Fig. 6.6, representa as três camadas do protocolo de comunicação de dados, cada uma com um processador independente.

Em um protocolo de comunicação que se baseia em troca de primitivas, a ocorrência de um dado evento (temporização, recebimento de mensagem) pode dar origem ao envio de outras mensagens para as camadas vizinhas. Isto está representado como uma realimentação de uma camada para outra. Como o enfoque está na camada LLC, as realimentações das demais camadas foram ignoradas.

Como está sendo analisando a comunicação em uma única direção, pode-se dizer que:

- λ é a taxa de chegada das mensagens que desejam ser transmitidas;
- λ_2 é a taxa de chegada das mensagens oriundas do gerenciamento da rede (temporizações) e pode ser estimada como $t * \lambda$, onde t é uma porcentagem da taxa de chegada;
- λ_3 é a taxa de chegada de mensagens originadas na camada MAC remota devido ao protocolo (confirmações ou controles de fluxo da rede, pedidos de retransmissão, etc..) e pode ser estimada como $r * \lambda$, onde r é uma porcentagem da taxa de chegada;

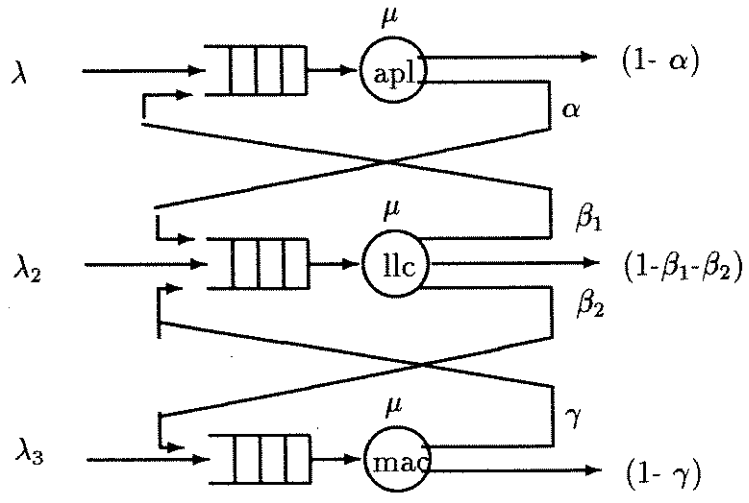


Figura 6.6: Modelo analítico

- α é a porcentagem de realimentação da camada APLICAÇÃO para a LLC e pode ser estimada como sendo $\frac{\lambda}{\lambda+\lambda_3}$;
- β_1 é a porcentagem de realimentação da camada LLC para a APLICAÇÃO e pode ser estimada como sendo $\frac{\lambda_3}{\lambda+\lambda_2+\lambda_3}$;
- β_2 é a porcentagem de realimentação da camada LLC para a MAC e pode ser estimada como sendo $\frac{\lambda}{\lambda+\lambda_2+\lambda_3}$;
- γ é a porcentagem de realimentação da camada MAC para a LLC e pode ser estimada como sendo $\frac{\lambda_3}{\lambda+\lambda_3}$;

Portanto:

$\alpha = \frac{1}{(1+r)}$, $\beta_1 = \frac{r}{(1+r+t)}$, $\beta_2 = \frac{1}{(1+r+t)}$, $\gamma = \frac{r}{(1+r)}$

Aplicando o teorema de Jackson [Stuck 45] para esta rede de filas obtem-se o seguinte resultado:

$E\{N_{llc}\} = \frac{\rho_{llc}*(1+r+t)}{1-\rho_{llc}*(1+r+t)}$, onde $\rho_{llc} = \frac{\lambda}{\mu}$

6.3.4 Modelo de simulação

Existem algumas considerações importantes, simplificadas no modelo analítico, que podem ser analisadas através da definição de um modelo de simulação:

- O processador nem sempre estará nas condições impostas para a análise através de N processadores independentes.
- O tratamento de uma determinada mensagem, pode causar uma realimentação simultânea a duas camadas.

Para considerar estas duas questões foi elaborado um modelo de simulação conforme mostrado na Fig. 6.8. O software de simulação RESQ [Loewner 36] permite a definição de nós, que representam os estágios passados por um processo para tratar uma mensagem. As conexões entre os nós refletem os possíveis fluxo das mensagens. A simbologia dos ícones utilizados pelo software na representação do modelo está descrita na Fig. 6.7.

No modelo, são definidos três nós geradores de mensagens para simular a chegada de mensagens nas camadas APLICAÇÃO, LLC, MAC denominados apl_src, llc_src e mac_src, respectivamente. Os processos que implementam as camadas ([Ussami 46]) estão representados pelas filas apl, llc e mac. O processador é representado pelo nó cpu_queue. Quando um processo recebe uma mensagem, esta fica aguardando a disponibilidade do processador. Os nós sched, cpu_0_3, cpu_1, cpu_2, cpu_a_1, cpu_l_m, cpu_a_m são utilizados para simular o escalonamento dos processos realizado pelo processador.

Após o tratamento de uma mensagem, o processador decide a próxima camada que deve ser atendida. Na simulação isto é feito através da atribuição de um "token" a uma das filas cujas mensagens aguardam por atendimento. Quem estiver com o "token" tem direito de uso da CPU. A escolha é feita da seguinte maneira: se as três filas estão vazias ou cheias, o "token" é alocado para a fila apl, uma vez que a taxa de chegada de mensagens para esta fila é muito maior que as demais. Caso somente uma das filas tenha mensagens, o "token" é dado a esta fila. Caso duas das filas tenha mensagens aguardando tratamento, a escolha é feita aleatoriamente com iguais probabilidades de escolha.

Decidido o destino do processador, é preciso cuidar do efeito do tratamento de uma mensagem. Os nós dest_apl, dest_llc, dest_mac decidem se uma mensagem vinda de uma das camadas (apl, llc, mac) ocasionará o envio de uma ou mais mensagens

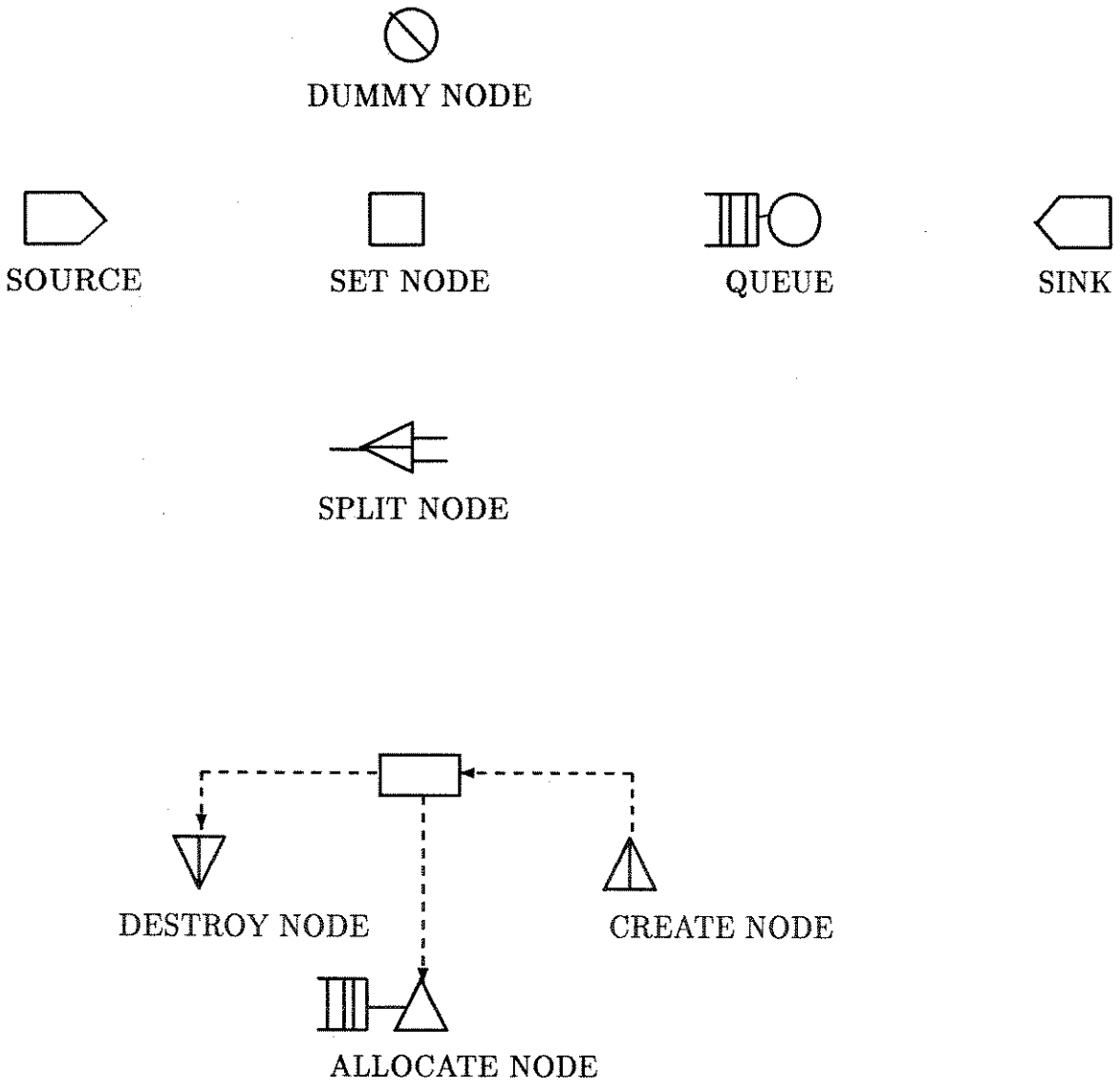


Figura 6.7: Simbologia dos elementos do software de simulação RESQ

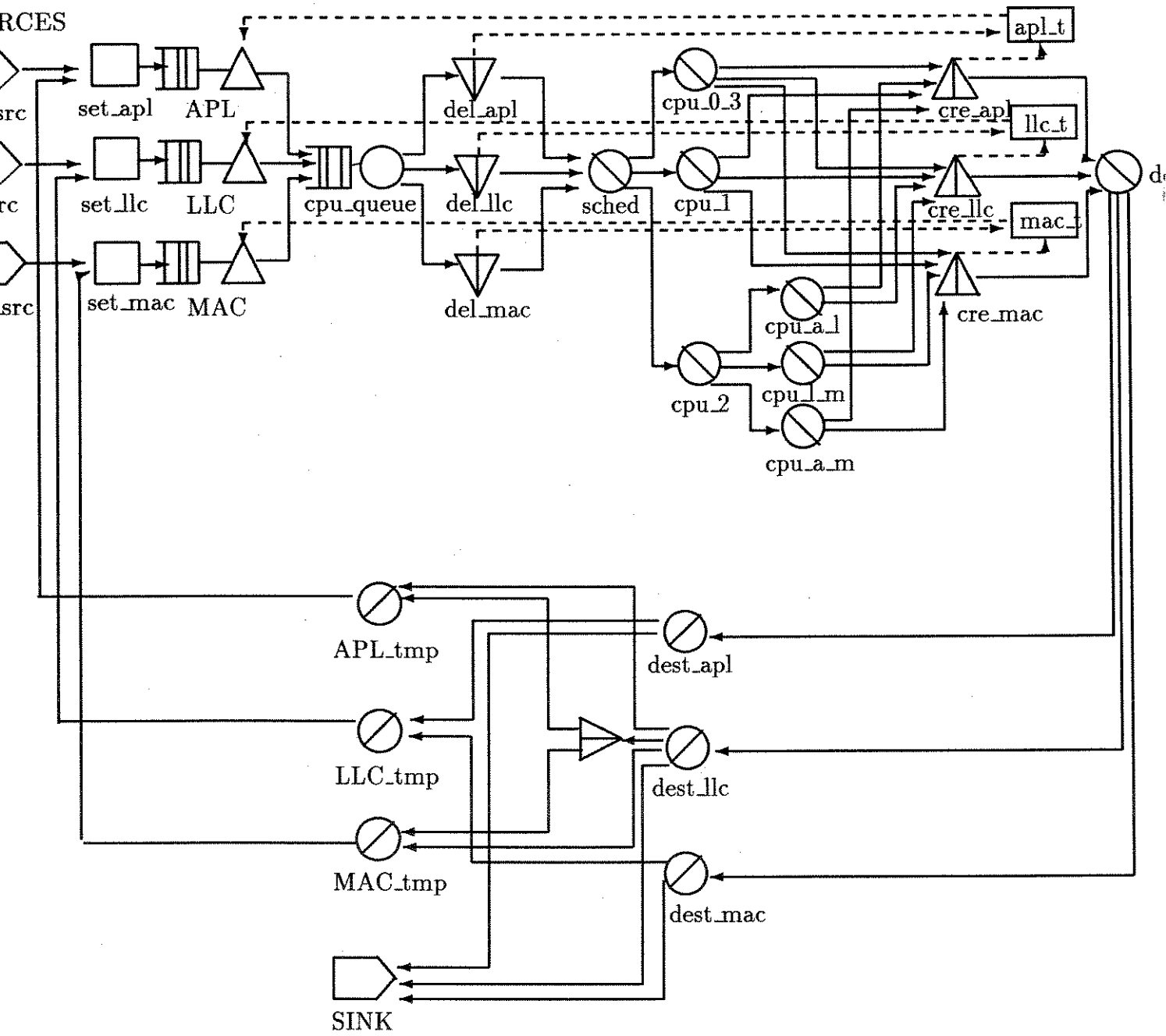


Figura 6.8: Modelo real

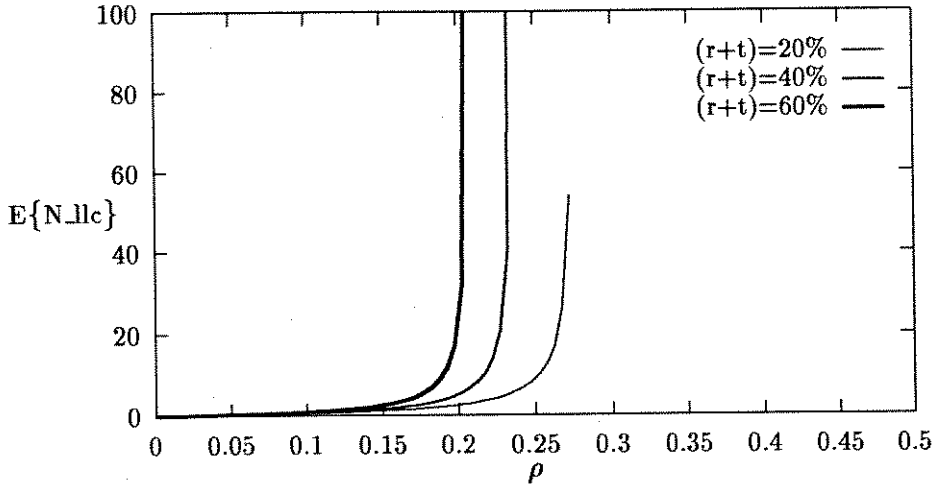


Figura 6.9: Modelo analítico da camada LLC (E{N_llc})

às outras camadas: realimentação única (nós apl_tmp, llc_tmp, mac_tmp) ou realimentação simultânea (nó spli_no). O nó SINK representa a liberação da mensagem do sistema, ou seja, a mensagem é consumida pelo processo.

O software usado na simulação permite um alto grau de parametrização, possibilitando muitos estudos de caso.

6.4 Avaliação do desempenho do protocolo LLC

Para obter as curvas de desempenhos dos dois modelos descritos na seção anterior foi calculado o número médio de mensagens na camada LLC (E{N_llc}) em função de ρ_{llc}, para diversos valores de (r+t). Estes dados foram obtidos para diversos valores de ρ_{llc}, hipotéticos, uma vez que na ocasião da avaliação de desempenho, o equipamento utilizado na implementação (PP) estava danificado.

Para o modelo analítico, onde $\rho_{llc} = \frac{\lambda}{\mu}$, e $\mu = \frac{\mu_r}{3}$, são apresentados os resultados para valores de sobrecarga (r+t) fixados em 20% ($3.6*\rho/(1-3.6*\rho)$), 40% ($4.2*\rho/(1-4.2*\rho)$) e 60% ($4.8*\rho/(1-4.8*\rho)$ na Fig. 6.9.

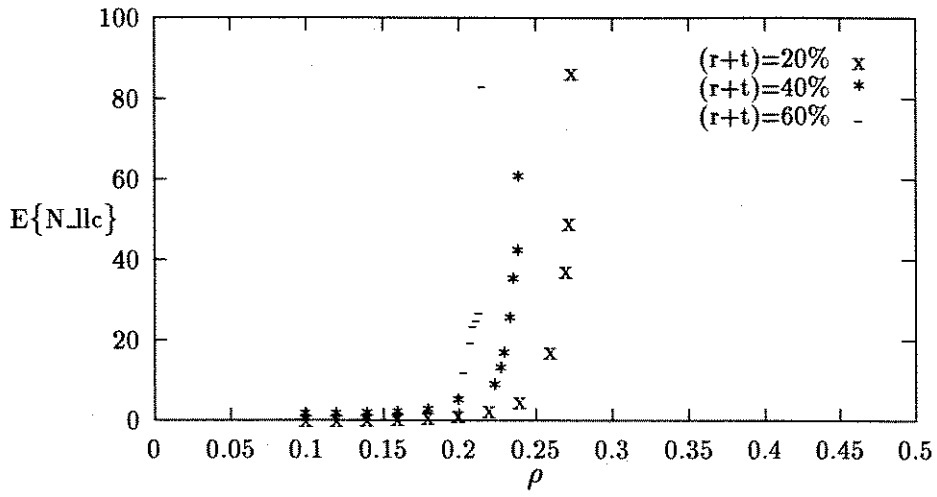


Figura 6.10: Modelo simulado da camada LLC (E{N_llc})) - Realimentação simultânea de 5%

Para obter dados do modelo simulado, a taxa de entrada na camada APLICAÇÃO foi estabelecida como sendo 0.2 mensagens/segundo. Para obter os valores de ρ_{llc} variou-se o valor da taxa de atendimento μ_{llc}. Considerou-se as mesmas taxas de sobrecarga: 20% (t=5, r=15), 40% (t=10, r=30) e 60% (t=15, r=30) avaliadas no modelo analítico. Aplicando as fórmulas para o cálculo das probabilidades de realimentação, foram calculadas as taxas α, β_1, β_2 e γ para cada caso.

A probabilidade de realimentação simultânea foi introduzida como sendo 5%. Consequentemente, β_1, β_2 foram diminuídos de 2.5% cada. Foi assumido que a probabilidade de um processo ser escalonado é igual para qualquer processo (não há prioridade nos processos).

Para cada situação, obtive-se o valor do número médio de mensagens na camada LLC em função de ρ conforme ilustrado na Fig. 6.10. Diminuindo a probabilidade de realimentação simultânea de mensagens para 2% (consequentemente β_1, β_2 são diminuídos de 1% cada), foi obtida uma nova situação representada pelo gráfico da Fig. 6.11.

Observando as Figs. 6.9, 6.10 e 6.11 pode-se concluir que para valores peque-

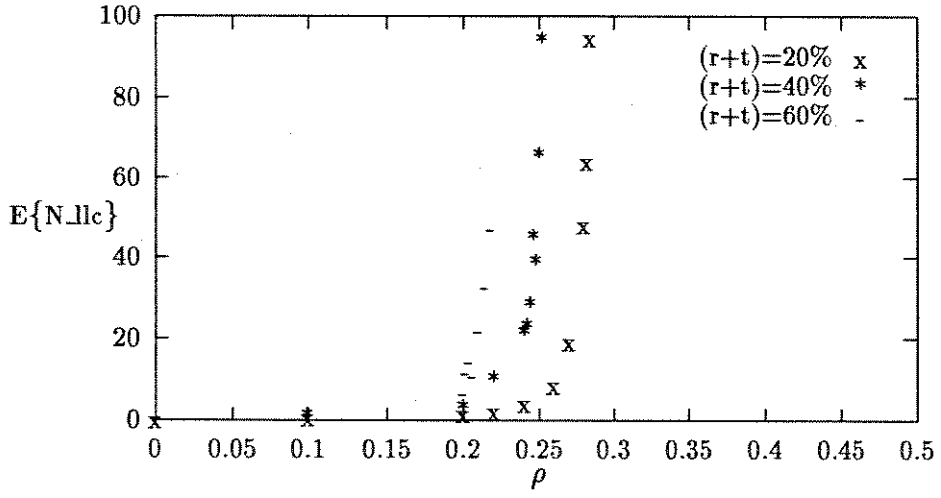


Figura 6.11: Modelo simulado da camada LLC (E{N_llc})) - Realimentação simultânea de 2%

nos da sobrecarga causada pela retransmissão (r+t) e realimentação simultânea, o modelo analítico é sub estimado. No entanto, para valores maiores, os dois modelos apresentam resultados muito próximos. Isto se deve a otimização do uso da CPU que, no modelo simulado, favorece o atendimento de uma fila, uma vez que pelo menos uma das outras filas estão vazias, ou com taxas de chegadas menores. Nos casos onde as taxas de chegada das filas LLC e MAC são iguais que a APL, o processador passará a atender as filas de modo mais "igualitário", aproximando-se ao modelo analítico, onde a CPU tem sua capacidade limitada a um terço, independentemente das taxas de chegada nas demais filas.

6.4.1 Tempo de escalonamento dos processos

O modelo de simulação apresentado, não leva em consideração o tempo gasto pelo processador para substituir o contexto do processo em execução. Este tempo é um fator muito importante na avaliação. Para considerá-lo, foi adicionado ao modelo um elemento que causa um retardo no atendimento de mensagens na fila LLC. A melhor maneira encontrada para representar este atraso foi introduzir uma fila com servidores, cujo tempo de atendimento é constante e parametrizável. Assim o modelo serve para avaliar outros processadores diferentes do PP. O novo modelo pode ser observado na Fig. 6.12.

Para obter os dados deste novo modelo, foram consideradas as mesmas taxas de sobrecarga: 20%(t=5, r=15), 40% (t=10, r=30) e 60% (t=15, r=30), a probabilidade de realimentação simultânea de mensagens como 2% e o tempo gasto no reescalonamento foi considerado constante e igual a 50ms. A nova situação obtida está representada pelo gráfico da Fig. 6.13. Comparando este gráfico com o obtido no modelo analítico (Fig. 6.9), pode-se observar que o comportamento é idêntico.

6.5 Conclusões

O modelo de camadas do protocolos de comunicação sugere um modelo de análise onde cada uma camada representa uma fila que recebe elementos das filas das camadas vizinhas, e que realimenta estas mesmas filas. Cada fila, de acordo com o protocolo de comunicação da camada que representa, estabelece as porcentagens de realimentação para as filas vizinhas. Dependendo da implementação das camadas: cada uma com seu processador, ou várias camadas em um processador e a política

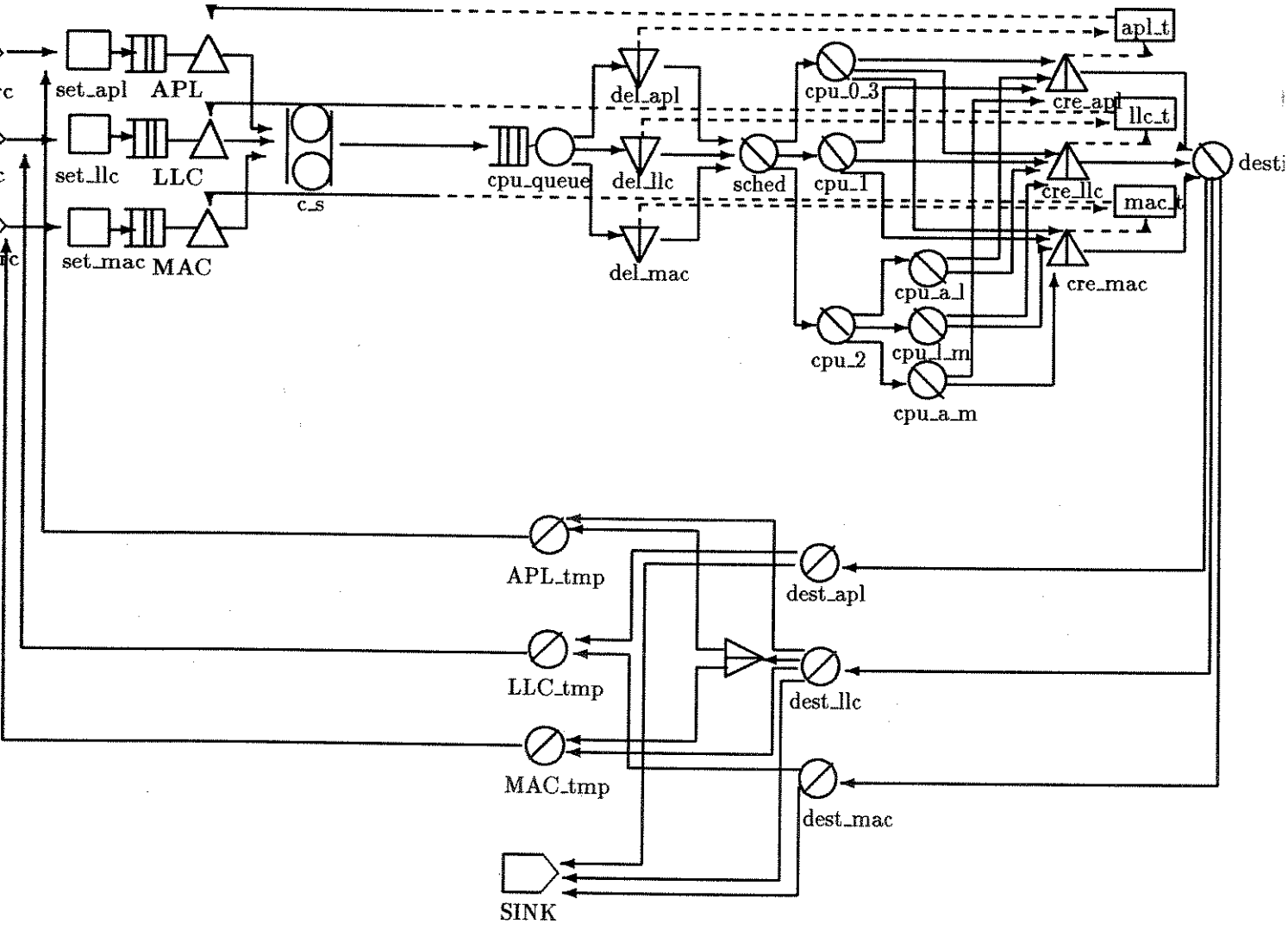


Figura 6.12: Modelo simulado considerando tempo de escalonamento

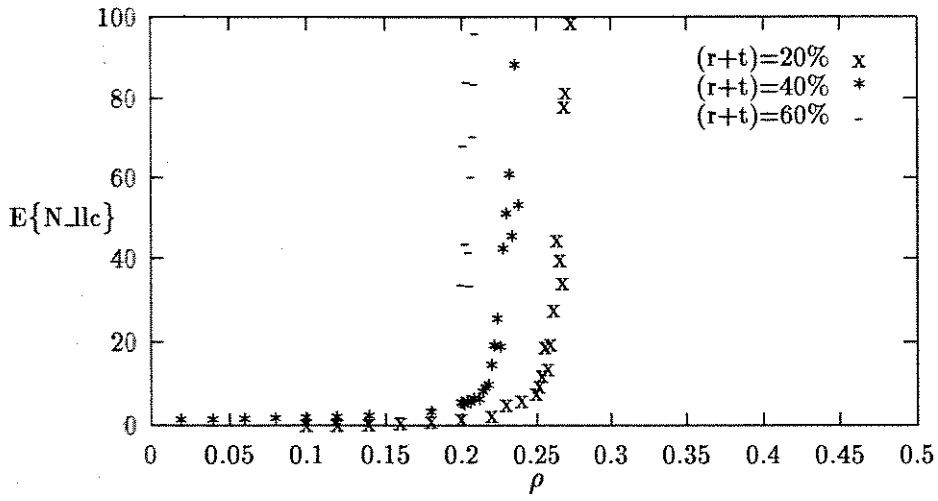


Figura 6.13: Modelo simulado da camada LLC (E{N_llc}) - Realimentação simultânea de 2%, tempo de escalonamento de 0.05s

de escalonamento do processador; se estabelece uma disciplina de atendimento da fila.

Para a elaboração de um modelo para a RALFO, inicialmente foi elaborado um modelo analítico com simplificações no modelo analisado. Posteriormente o modelo foi refinado, na tentativa de espelhar a realidade da implementação, através de um software de simulação (RESQ).

Uma característica muito interessante deste software de simulação é a facilidade de parametrizar dados (taxas de chegada, taxa de serviço, probabilidades no roteamento das mensagens) permitindo ampla exploração do modelo. A possibilidade de implementar a necessidade de uma mensagem alocar um recurso (uso de "tokens") permitiu a simulação do escalonamento dos processos.

Os resultados obtidos pelo modelo simulado validam o modelo analítico, uma vez que os gráfico obtidos têm comportamento semelhante. Portanto pode-se concluir que o modelo analítico pode ser utilizado para obter resultados em primeira aproximação. Caso se queira obter dados mais próximos a realidade, basta utilizar o modelo de simulação que está parametrizado suficientemente para suportar diversas situações.

Infelizmente não foi possível confrontar dados reais obtidos da implementação da camada LLC no protótipo da RALFO, com os resultados obtidos nos modelos aqui apresentados, uma vez que o equipamento foi danificado na ocasião da análise.

Capítulo 7

Conclusões

Os objetivos deste trabalho foram: definir um ambiente de desenvolvimento para a implementação de protocolos de comunicação, na rede RALFO; e nesta rede, implementar e avaliar o desempenho do protocolo LLC (IEEE 802.2).

O projeto RALFO propõe uma rede de serviços integrados de voz e dados. Ela utiliza como meio de transmissão, fibras ópticas em uma topologia de anel duplo, com o método de acesso próprio, baseado no "slotted ring" do Anel de Cambridge. Para implementar esta rede foi definido um modelo de camadas baseado no modelo de referência OSI/ISO, onde duas camadas são definições próprias do projeto RALFO: camada MAC, implementa o método de acesso da RALFO; e a camada LLC/VOZ, responsável pelo tratamento da sinalização do transporte de voz. Para testar a viabilidade desta rede está sendo implementado um protótipo experimental que utiliza, como nó da rede, o processador PP. A ele são acopladas mais duas placas processadoras: uma placa para tratamento dos terminais de voz (telefones); e outra placa para acesso ao meio físico (fibras ópticas). Serão implementadas as camadas PHY, MAC, LLC, LLC/VOZ, APLICAÇÃO/VOZ e a APLICAÇÃO/DADOS.

As primeiras decisões que devem ser tomadas para iniciar a implementação de protocolos é a escolha do sistema operacional, que gerenciará a execução de todas as camadas no equipamento escolhido, e a escolha da linguagem de programação, que será utilizada na programação dos protocolos. O sistema operacional deve ter características de tempo-real, oferecer um ambiente multi-tarefa, fornecer mecanismos de comunicação de dados entre processos e mecanismos que permitam a implementação de temporizadores. A linguagem de programação deve oferecer recursos para a implementação de tarefas (processos) de maneira modular e independente. Assim, cada camada poderá ser implementada como um processo. Para que as camadas

troquem primitivas é preciso algum mecanismo de troca de dados entre os processos, por exemplo, envio de mensagens, compartilhamento de memória, etc.. Finalmente, para temporizar primitivas enviadas às camadas remotas, é desejável que a linguagem ofereça facilidades na supervisão de tempo.

No projeto RALFO, optou-se pelo sistema operacional PP/SOP e a linguagem de programação CHILL. Por ter sido projetada pelo CCITT, com o objetivo de desenvolver sistema de comunicação, CHILL oferece recursos muito poderosos e parece ser uma ótima opção para a implementação de protocolos. Para implementar os protocolos da RALFO, foi utilizado o conjunto de ferramentas APCC, desenvolvido eficientemente pelo CPqD/Telebrás. Uma vez tendo o protocolo especificado através de diagramas de transições para cada estado do protocolo, a implementação usando CHILL fica imediata:

1. Define-se um processo para cada camada.

2. Para cada camada define-se um sinal para cada primitiva, e os parâmetros da primitiva formam o conjunto de dados associado ao sinal.

3. Para cada estado de uma camada define-se um procedimento que fará o tratamento dos sinais recebidos através de instruções do tipo RECEIVE CASE.

4. Todo o envio de sinais é feito através de instruções SEND.

5. Implementa-se as temporizações através de sinais temporizados.

Todos os aspectos da implementação da comunicação e sincronização dos processos ficam transparentes ao programador, tendo este que se preocupar exclusivamente com o protocolo de comunicação. Os recursos da linguagem CHILL possibilitam a implementação de aplicações em tempo real, que necessitam de mecanismos de concorrência e supervisão de tempo. A capacidade de modularização e os tipos de dados oferecidos são pontos fortes da linguagem.

O uso da linguagem CHILL mostrou-se muito adequado à implementação de protocolos e vem sendo objeto de pesquisa na área de implementação de protocolos. Sua utilização implica na disponibilidade de um compilador que gere código executável para um equipamento que suporte o ambiente concorrente da linguagem, que neste trabalho foi o PP com o sistema operacional SOP. Atualmente esta disponibilidade é restrita, o que torna o uso da linguagem dependente da arquitetura de hardware escolhida. Embora existam outros sistemas de execução para a linguagem, nenhum é suficientemente disponível como os sistema operacionais DOS e UNIX que atingem

uma grande plataforma de hardware. Entretanto, faz parte dos planos da equipe do projeto CHILL do CPqD da Telebrás, elaborar um compilador capaz de gerar código para processadores INTEL 486, o que aumentaria a possibilidade de utilização da linguagem CHILL para uma plataforma de equipamentos bem maior que o PP.

Uma vez implementado o protocolo, passou-se a tarefa de avaliação de desempenho do protocolo implementado. O desempenho de um protocolo deve ser avaliado considerando-se os recursos utilizados para a implementação da comunicação de dados entre as camadas e o mecanismo de escalonamento destas, realizado pelo processador onde elas serão implementadas.

O modelo de camadas do protocolos de comunicação sugere um modelo de análise onde cada uma camada representa uma fila que recebe elementos das filas das camadas vizinhas, e que realimenta estas mesmas filas. Cada fila, de acordo com o protocolo de comunicação da camada que representa, estabelece as porcentagens de realimentação para as filas vizinhas. Dependendo da implementação das camadas: cada uma com seu processador, ou várias camadas em um processador e a política de escalonamento do processador; se estabelece uma disciplina de atendimento da fila.

Para a elaboração de um modelo para a RALFO, foi analisado isoladamente o protocolo de comunicação de dados, implementado no processador principal do nó da rede (PP). Inicialmente foi elaborado um modelo analítico (teoria de filas) e posteriormente o modelo foi refinado (modelo simulado), refletindo melhor a implementação, através de um software de simulação (RESQ). Observa-se que os resultados do modelo teórico são sub estimados, mas podem ser utilizados para se ter uma avaliação aproximada do protocolo. Estes modelos permitem prever o tamanho da fila de mensagem da camada LLC, para diversos valores de ρ_{llc} $(\frac{\lambda_{llc}}{\mu_{llc}})$. Uma vez conhecida a taxa de chegada da camada APLICAÇÃO, estabelecidas as porcentagens das taxas de retransmissão e temporização, e conhecida a taxa de atendimento do processador, é possível estimar o tamanho da fila de mensagens da camada LLC.

Trabalhos futuros

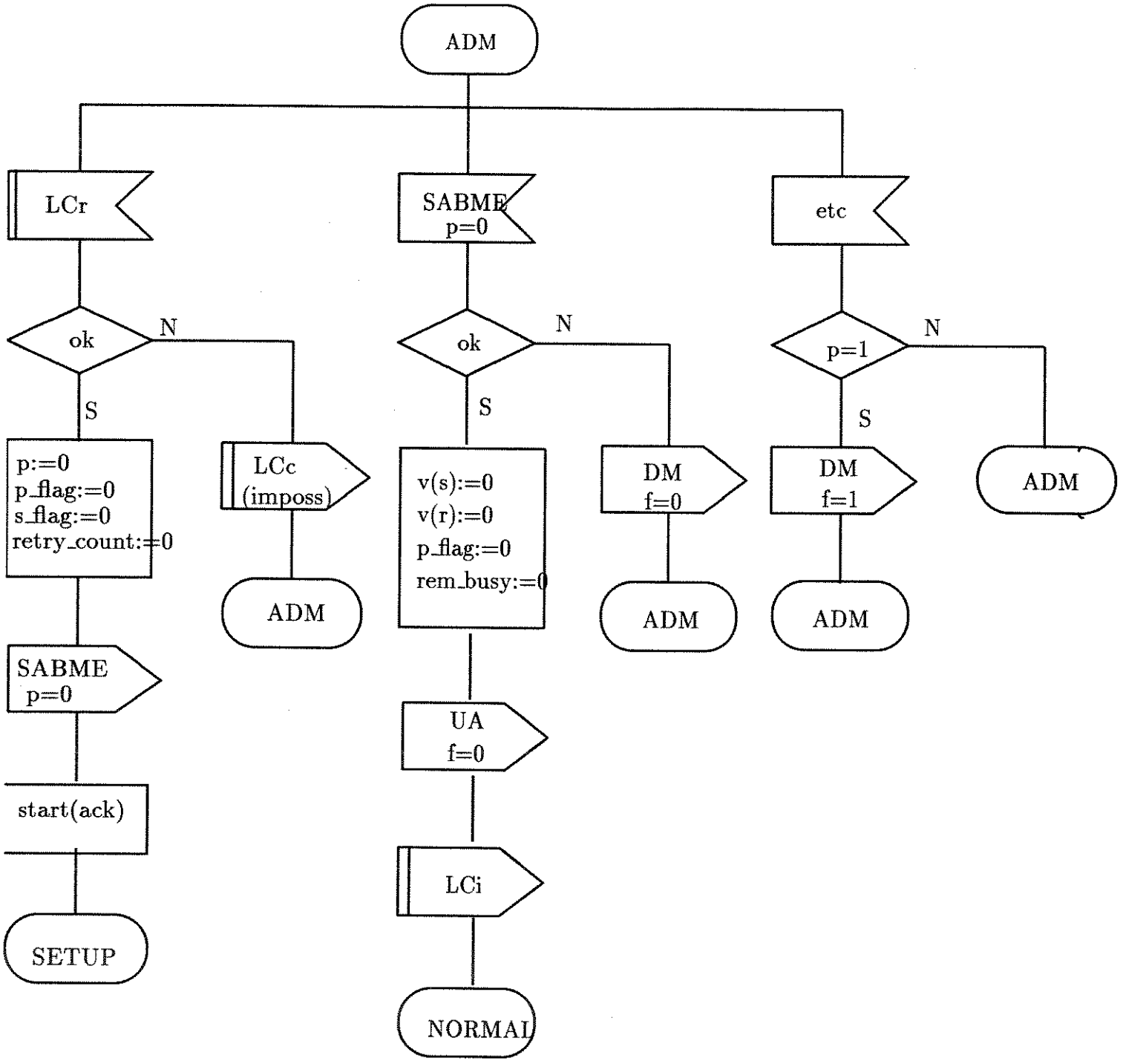
Esta implementação da camada LLC é suficientemente genérica para ser utilizada em outras redes que implementem esta camada segundo a recomendação IEEE 802.2 e utilizem a linguagem CHILL. Uma vez que a comunicação com as camadas vizinhas (MAC e APLICAÇÃO) é feita através de trocas de primitivas, basta que este conceito seja mantido para que o protocolo possa ser utilizado.

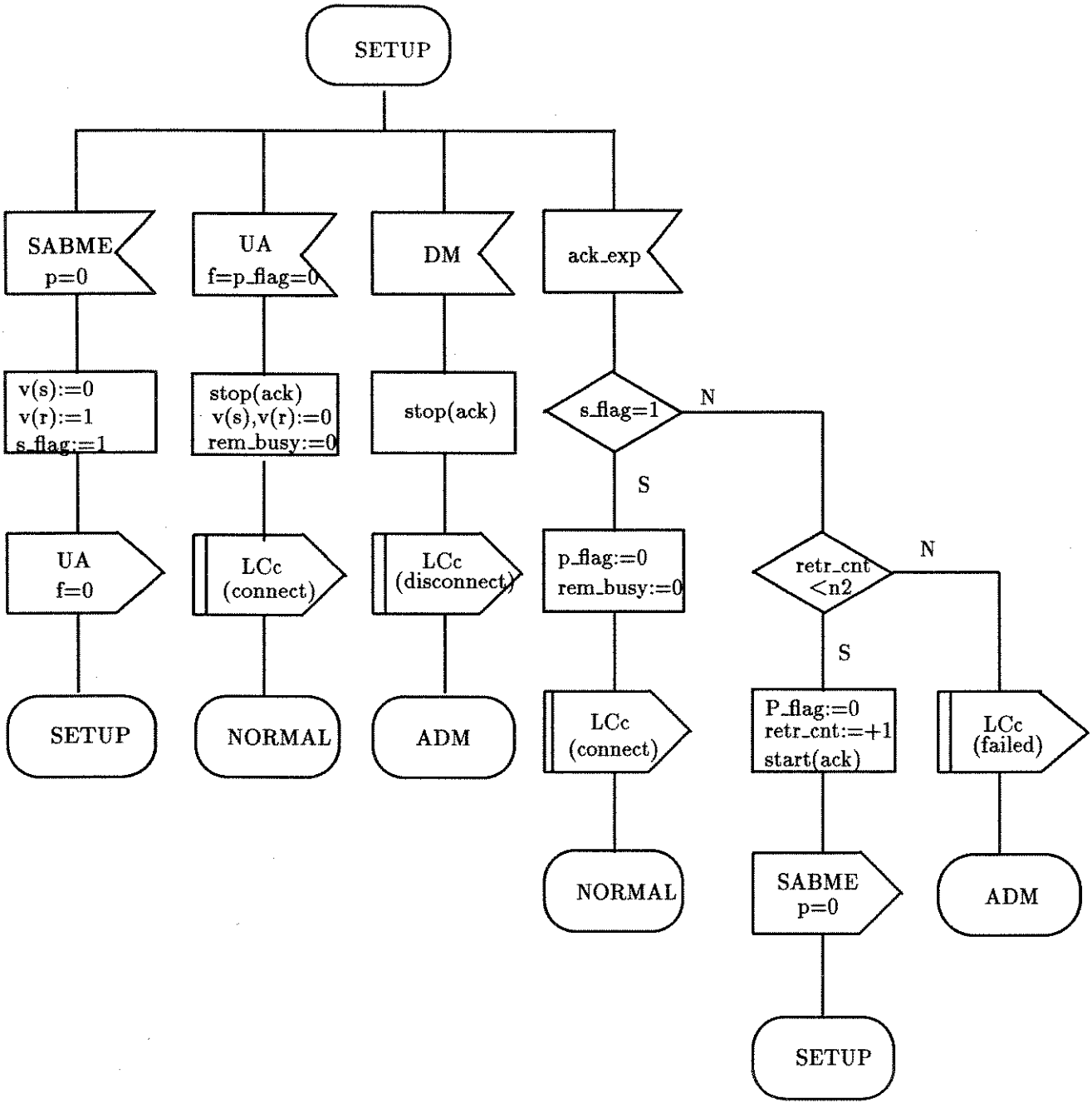
Infelizmente, o PP utilizado para a implementação dos protocolos da RALFO,

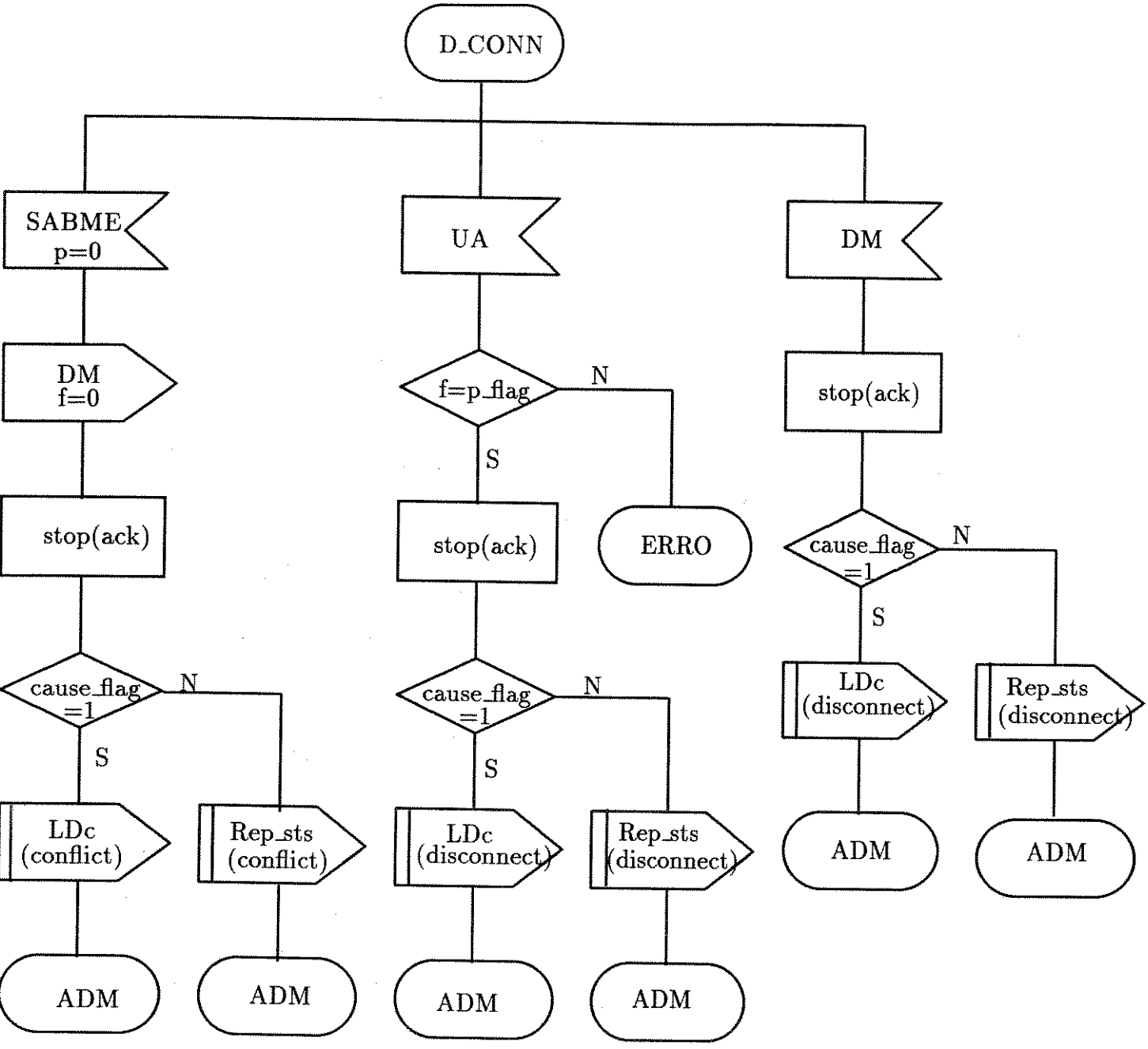
foi danificado no final deste trabalho, impossibilitando os testes de integração do módulo que implementa o protocolo LLC, propriamente dito, e o módulo que implementa os temporizadores da camada LLC. Uma sugestão para a continuidade deste trabalho é justamente finalizar estes testes de integração e uma vez tendo o protocolo operacional, seria possível fazer uma avaliação da performance do protocolo como um todo, simulando vários casos de temporização, perda de informação, quantidade de mensagens pendentes em cada camada, tempo médio de atendimento de uma mensagem, etc.. Assim, estes dados poderiam ser confrontados com os resultados obtidos pelos modelos propostos neste trabalho.

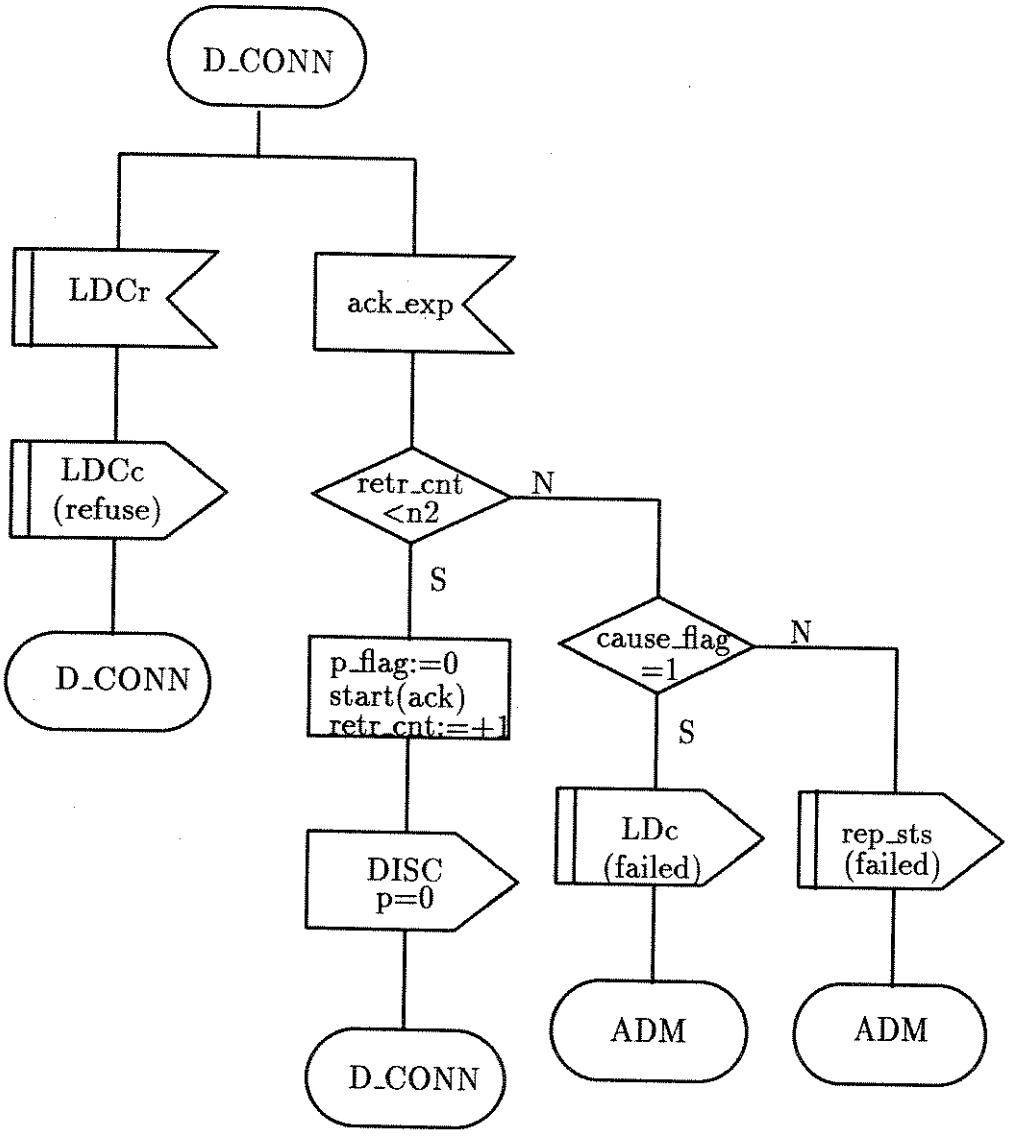
Apêndice A

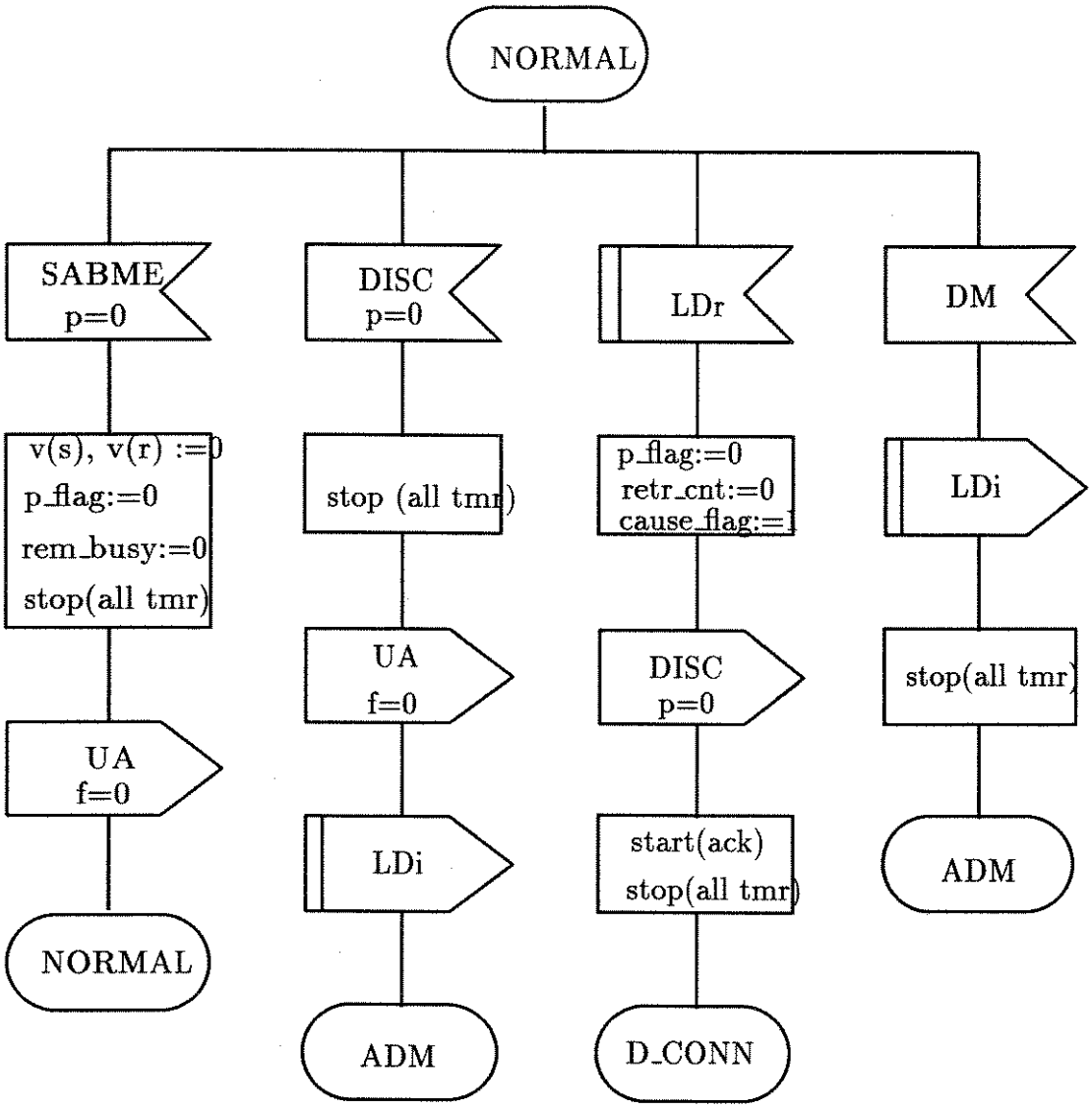
Especificação SDL do protocolo LLC implementado

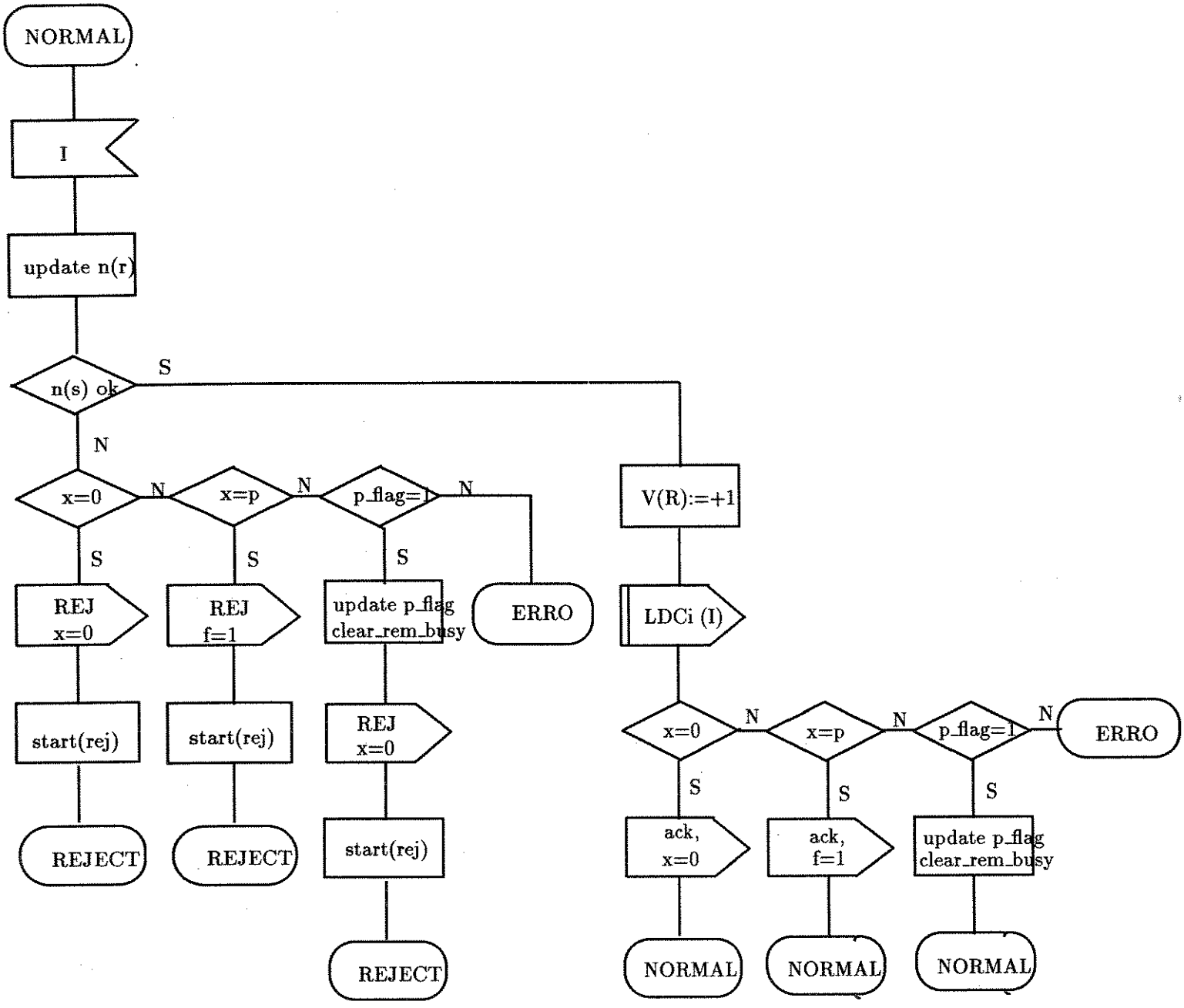

ADM
LCr
SABME
p=0
etc
ok
N
S
ok
N
S
p=1
N
S
p:=0
p_flag:=0
s_flag:=0
retry_count:=0
LCc
(imposs)
v(s):=0
v(r):=0
p_flag:=0
rem_busy:=0
DM
f=0
DM
f=1
ADM
ADM
ADM
ADM
SABME
p=0
UA
f=0
start(ack)
LCi
SETUP
NORMAL

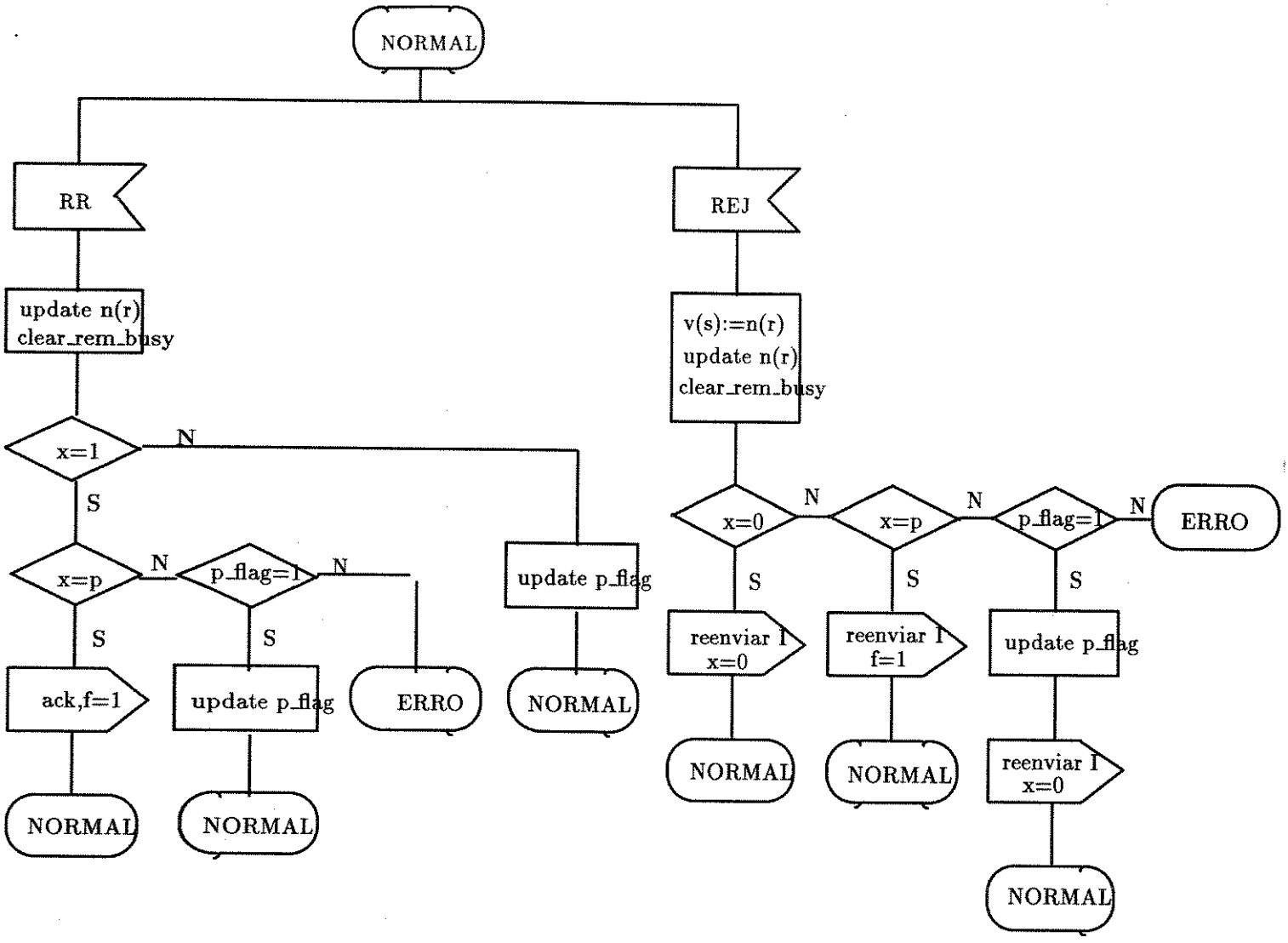

SETUP
SABME
p=0
UA
f=p_flag=0
DM
ack_exp
v(s):=0
v(r):=1
s_flag:=1
stop(ack)
v(s),v(r):=0
rem_busy:=0
stop(ack)
s_flag=1
N
S
UA
f=0
LCc
(connect)
LCc
(disconnect)
p_flag:=0
rem_busy:=0
retr_cnt
<n2
N
S
SETUP
NORMAL
ADM
LCc
(connect)
P_flag:=0
retr_cnt:=+1
start(ack)
LCc
(failed)
NORMAL
SABME
p=0
ADM
SETUP

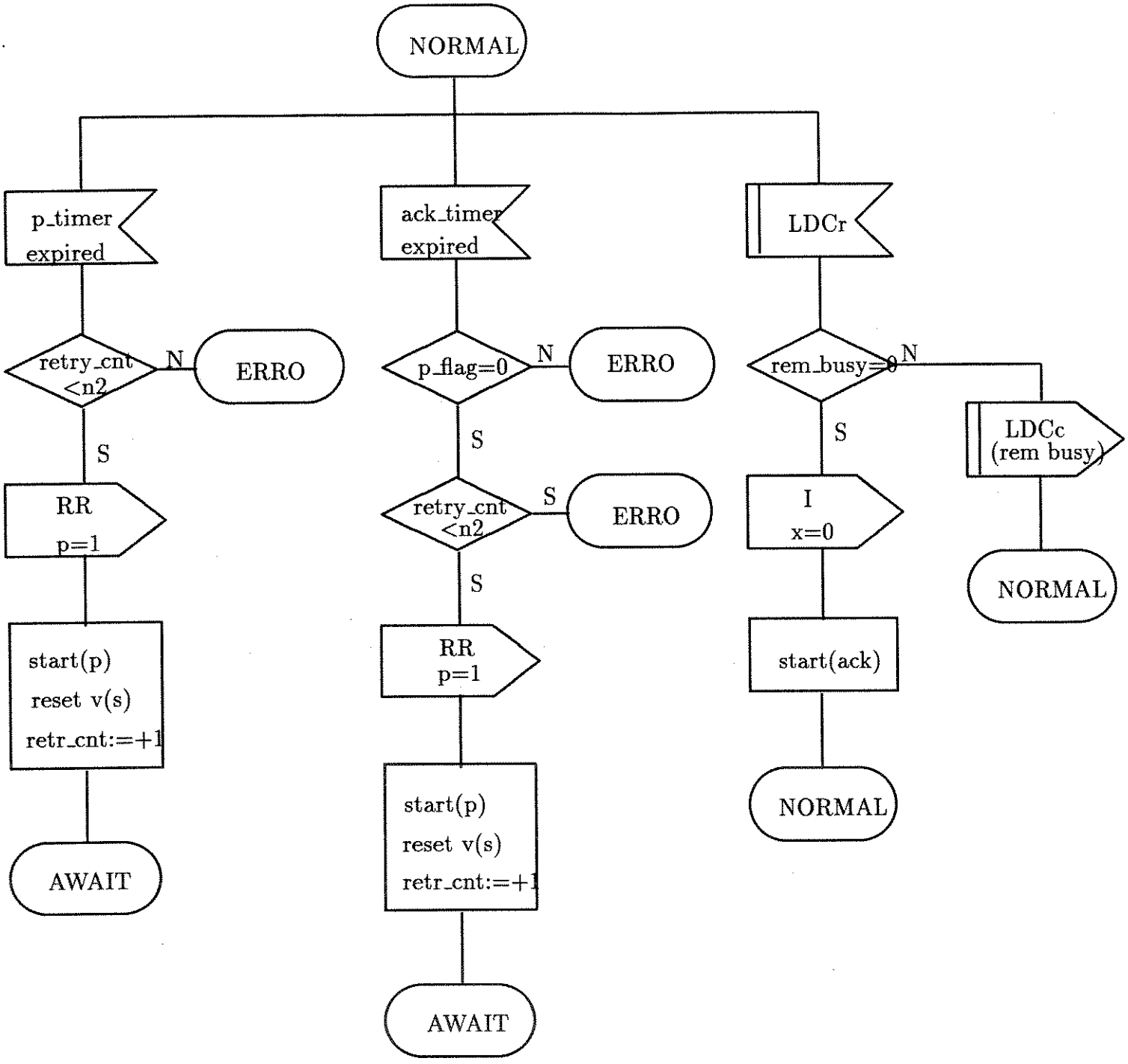

D_CONN
SABME
p=0
UA
DM
DM
f=0
f=p_flag
N
S
stop(ack)
stop(ack)
ERRO
stop(ack)
cause_flag
=1
N
S
cause_flag
=1
N
S
cause_flag
=1
N
S
LDc
(conflict)
Rep_sts
(conflict)
LDc
(disconnect)
Rep_sts
(disconnect)
LDc
(disconnect)
Rep_sts
(disconnect)
ADM
ADM
ADM
ADM
ADM
ADM

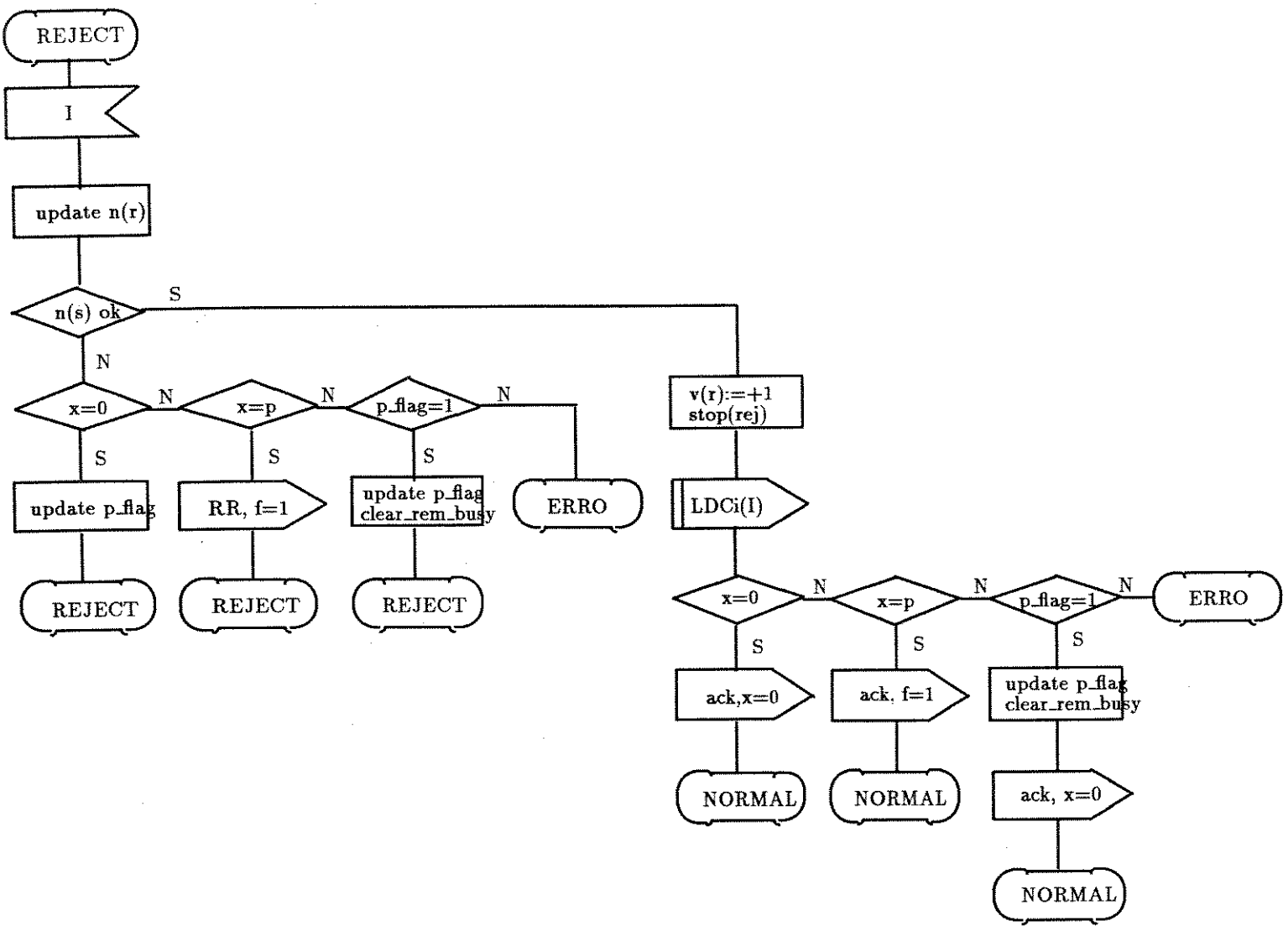

D_CONN
LDCr
ack_exp
LDCc
(refuse)
retr_cnt
<n2
N
S
D_CONN
p_flag:=0
start(ack)
retr_cnt:=+1
cause_flag
=1
N
S
DISC
p=0
LDc
(failed)
rep_sts
(failed)
D_CONN
ADM
ADM

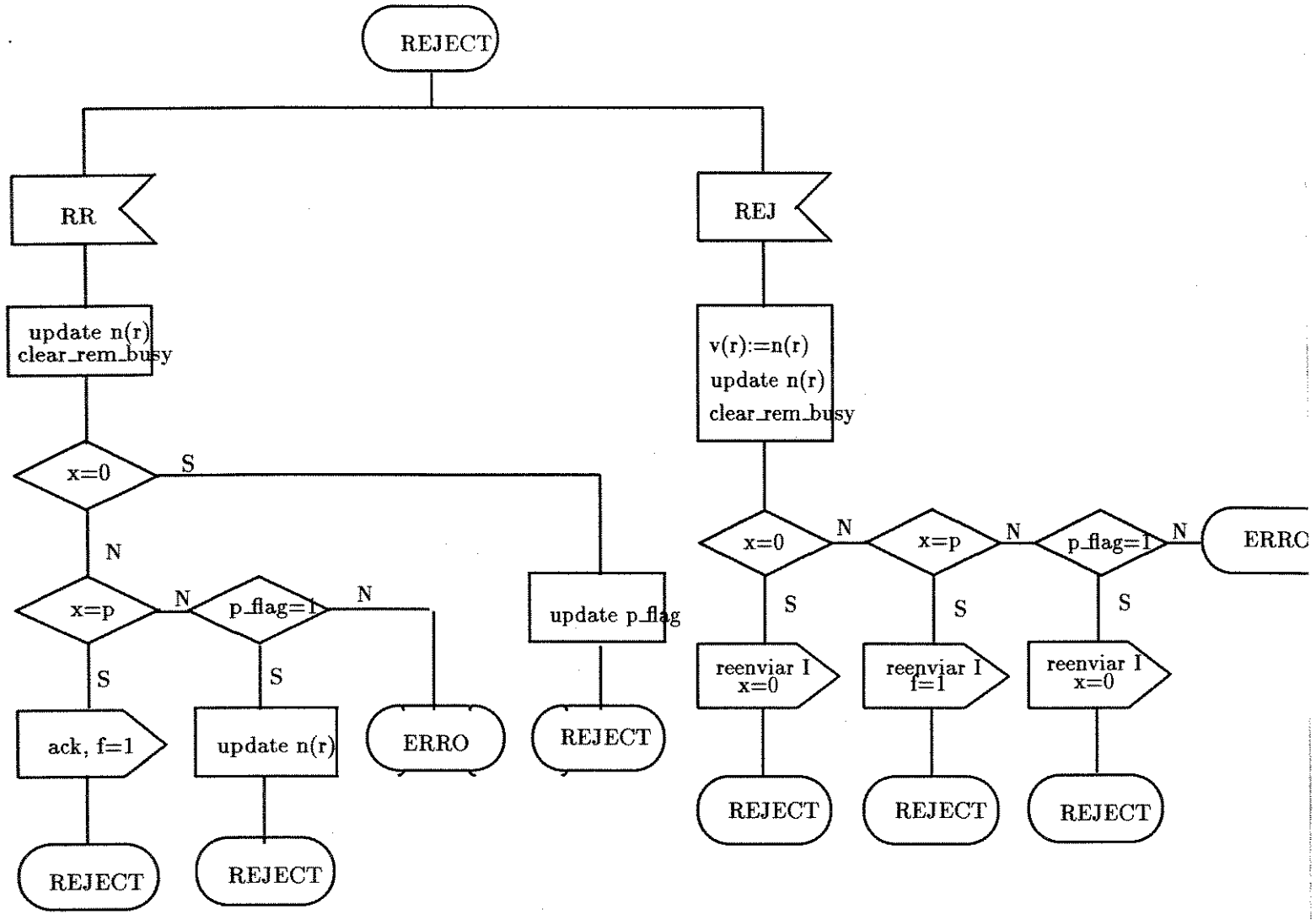

NORMAL
SABME
p=0
DISC
p=0
LDr
DM
v(s), v(r) :=0
p_flag:=0
rem_busy:=0
stop(all tmr)
stop (all tmr)
p_flag:=0
retr_cnt:=0
cause_flag:=
LDi
UA
f=0
DISC
p=0
stop(all tmr)
UA
f=0
LDi
start(ack)
stop(all tmr)
ADM
NORMAL
ADM
D_CONN

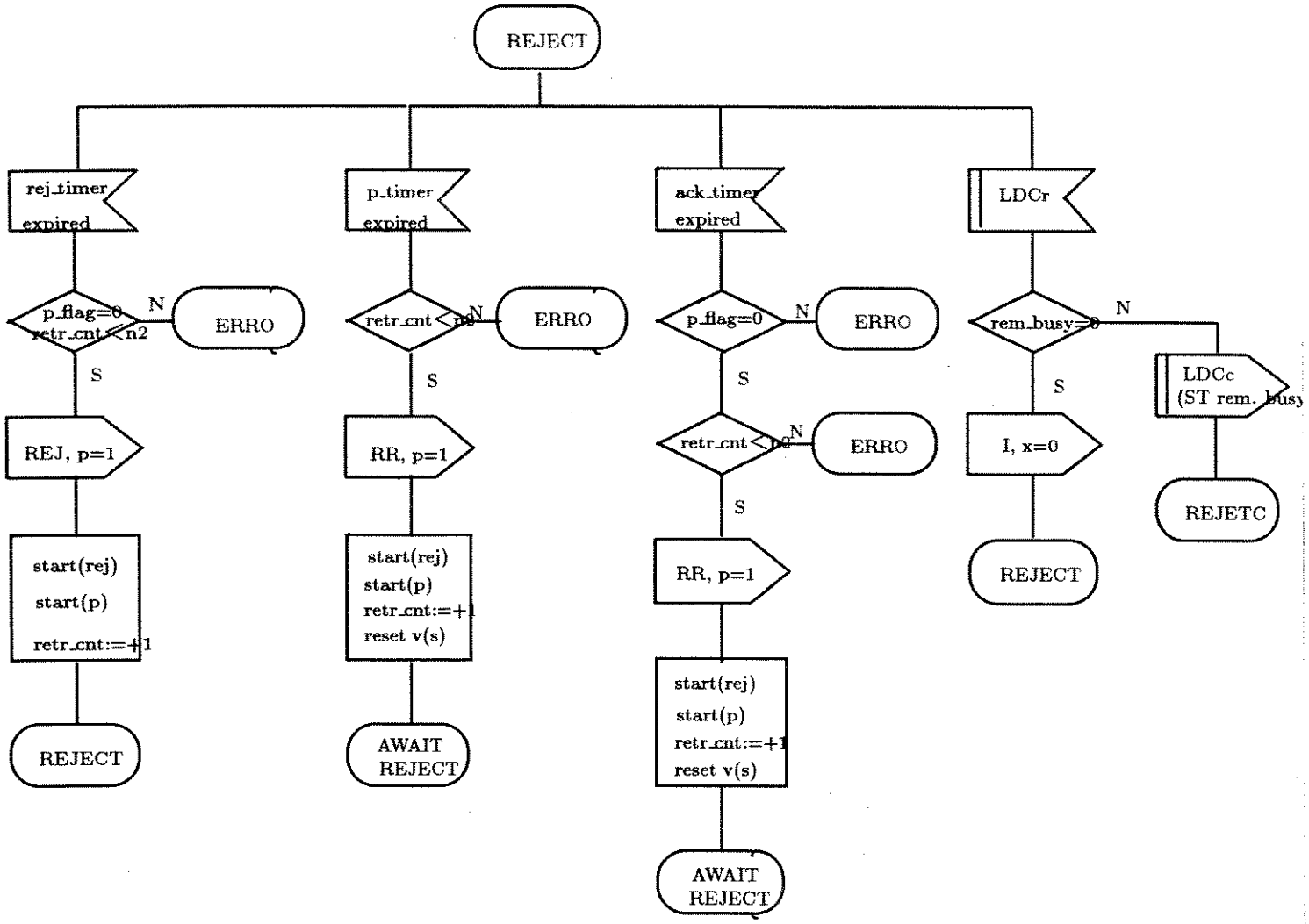

NORMAL
I
update n(r)
n(s) ok
S
N
x=0
N
x=p
N
p_flag=1
N
S
S
S
REJ
x=0
REJ
f=1
update p_flag
clear_rem_busy
ERRO
start(rej)
start(rej)
REJ
x=0
REJECT
REJECT
start(rej)
REJECT
V(R):=+1
LDCi (I)
x=0
N
x=p
N
p_flag=1
N
ERRO
S
S
S
ack,
x=0
ack,
f=1
update p_flag
clear_rem_busy
NORMAL
NORMAL
NORMAL

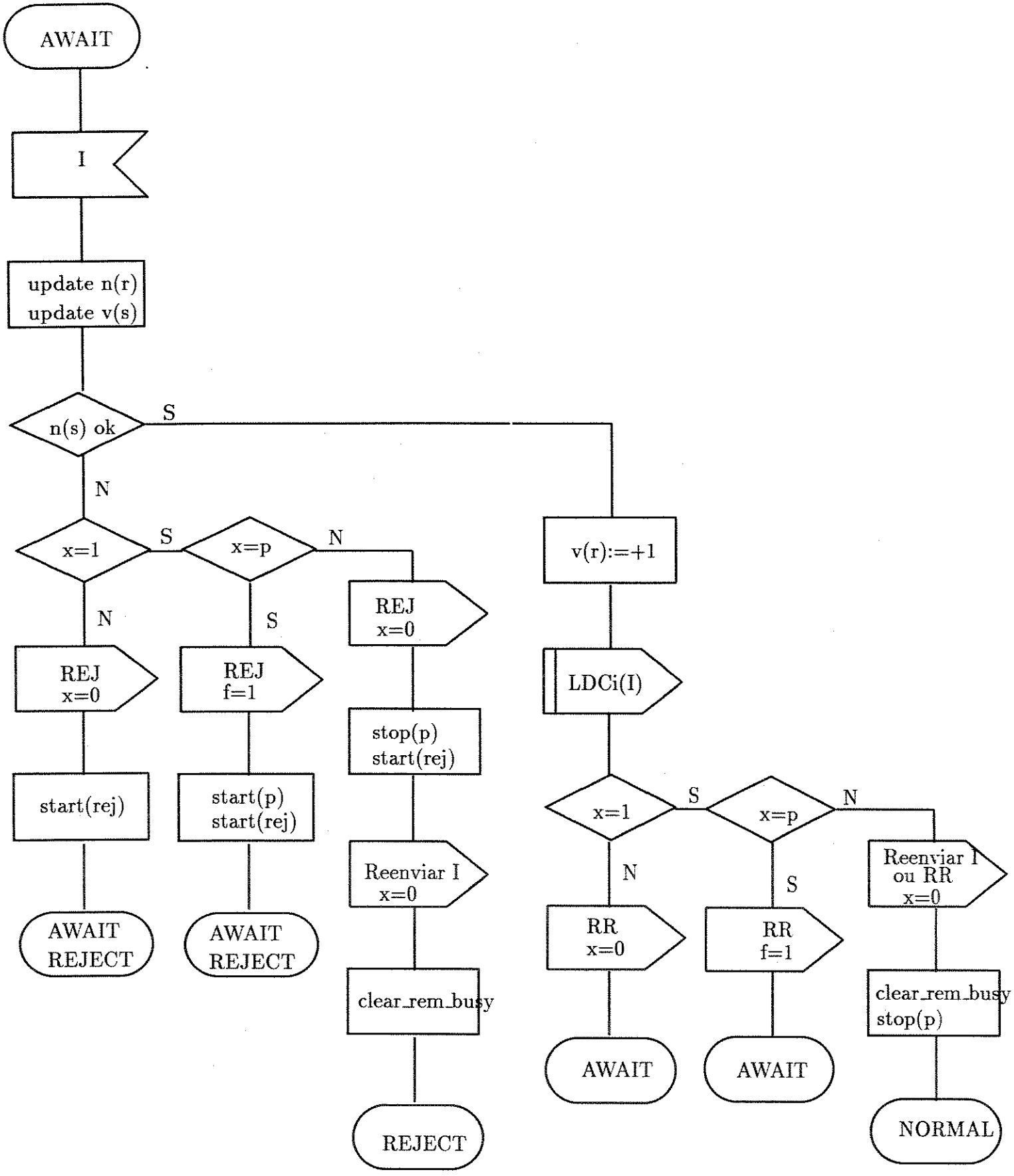

NORMAL
RR
REJ
update n(r)
clear_rem_busy
v(s):=n(r)
update n(r)
clear_rem_busy
x=1
N
S
x=p
N
p_flag=1
N
update p_flag
S
S
ack,f=1
update p_flag
ERRO
NORMAL
NORMAL
NORMAL
x=0
N
x=p
N
p_flag=1
N
ERRO
S
S
S
reenviar I
x=0
reenviar I
f=1
update p_flag
NORMAL
NORMAL
reenviar I
x=0
NORMAL

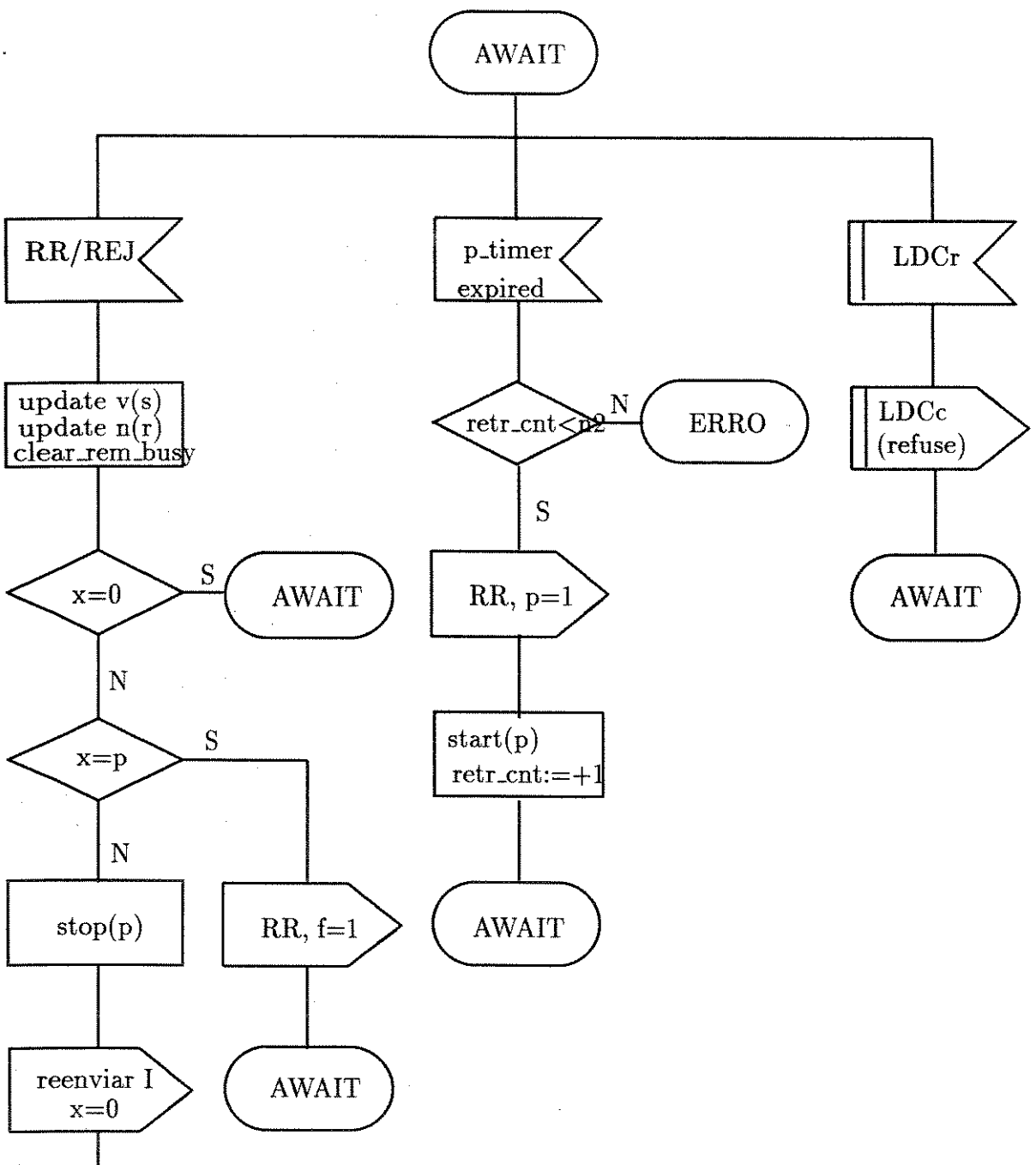

NORMAL
p_timer expired
ack_timer expired
LDCr
retry_cnt <n2
N
ERRO
p_flag=0
N
ERRO
rem_busy=0
N
S
S
S
LDCc (rem busy)
RR p=1
retry_cnt <n2
S
ERRO
I x=0
NORMAL
S
start(p) reset v(s) retr_cnt:=+1
RR p=1
start(ack)
start(p) reset v(s) retr_cnt:=+1
AWAIT
NORMAL
AWAIT

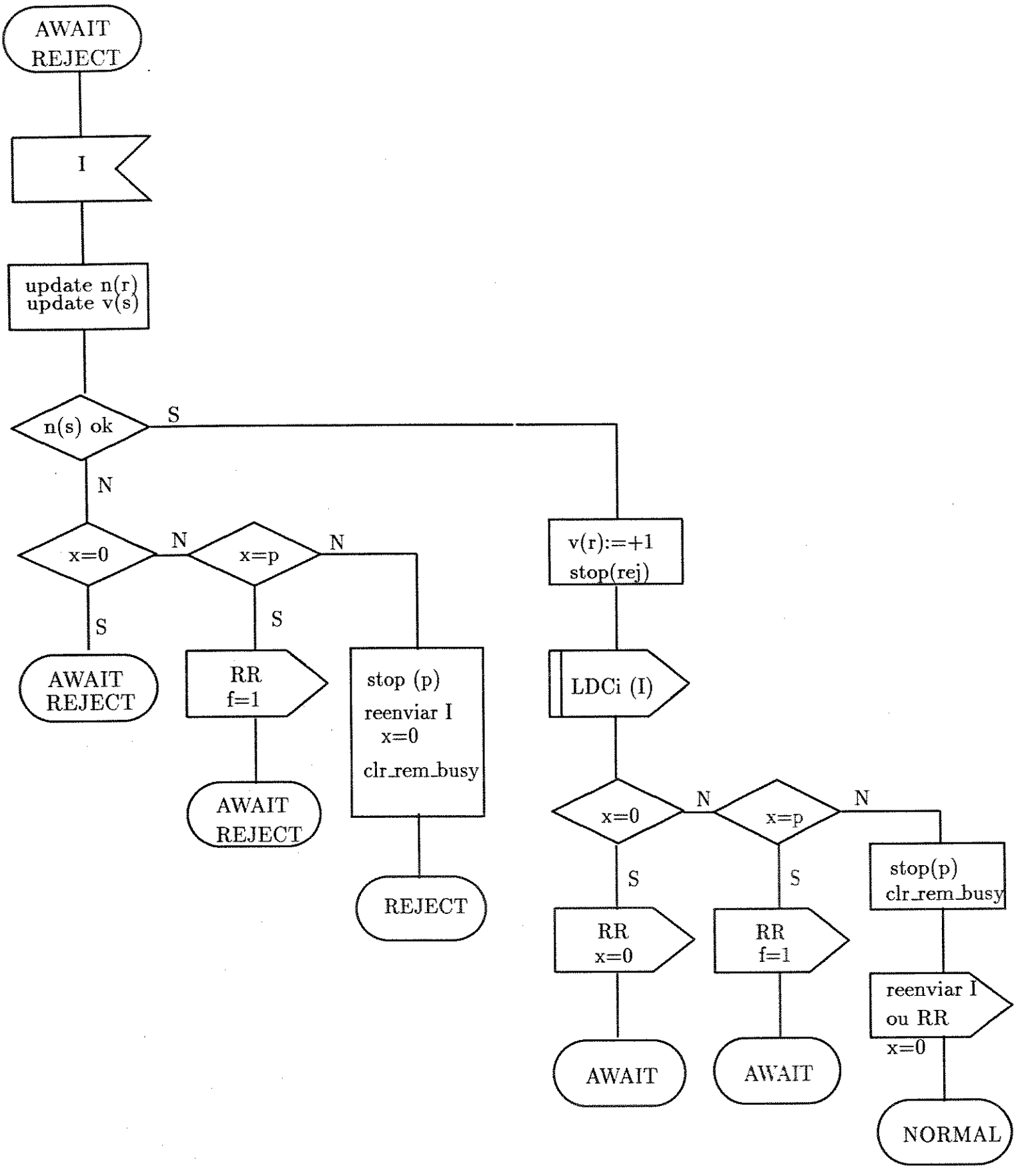

REJECT
I
update n(r)
n(s) ok
S
N
x=0
N
x=p
N
p_flag=1
N
v(r):=+1
stop(rej)
S
S
S
update p_flag
RR, f=1
update p_flag
clear_rem_busy
ERRO
LDCi(I)
REJECT
REJECT
REJECT
x=0
N
x=p
N
p_flag=1
N
ERRO
S
S
S
ack,x=0
ack, f=1
update p_flag
clear_rem_busy
NORMAL
NORMAL
ack, x=0
NORMAL

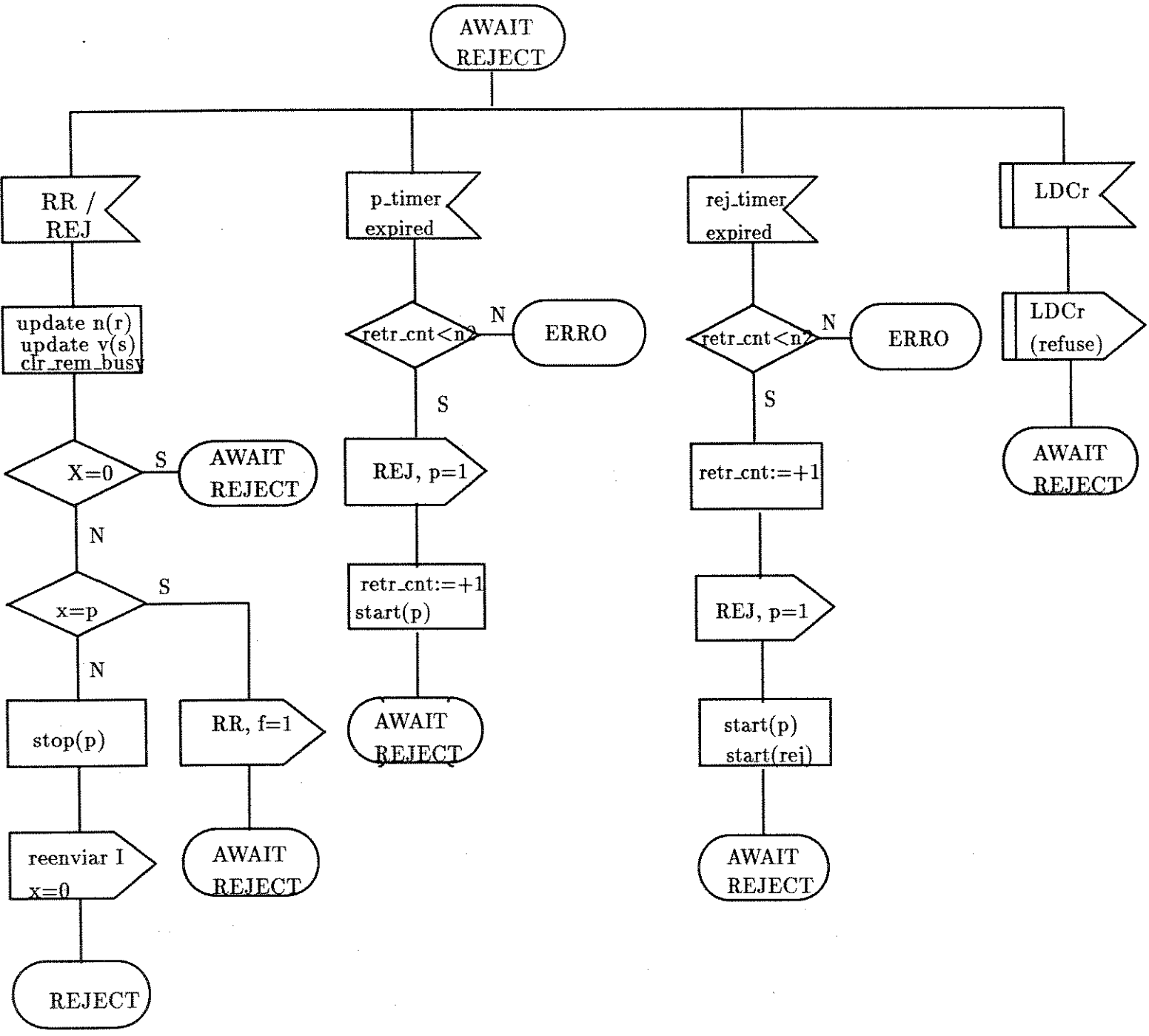

REJECT
RR
REJ
update n(r)
clear_rem_busy
v(r):=n(r)
update n(r)
clear_rem_busy
x=0
S
N
x=p
N
p_flag=1
N
update p_flag
x=0
N
x=p
N
p_flag=1
N
ERRO
S
S
S
S
S
ack, f=1
update n(r)
ERRO
REJECT
reenviar I
x=0
reenviar I
f=1
reenviar I
x=0
REJECT
REJECT
REJECT
REJECT
REJECT
REJECT

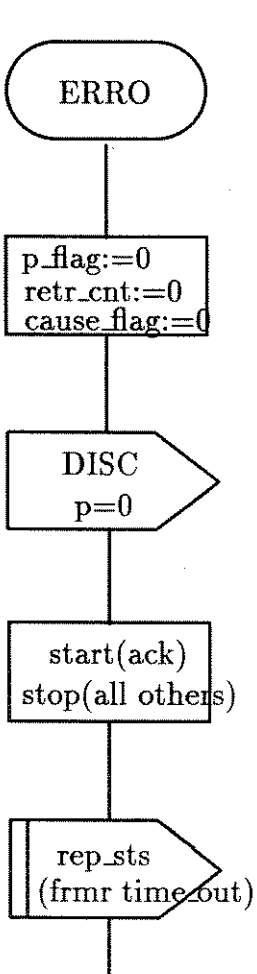

REJECT
rej_timer expired
p_flag=0 retr_cnt<n2
N
ERRO
S
REJ, p=1
start(rej)
start(p)
retr_cnt:=+1
REJECT
p_timer expired
retr_cnt<n2
N
ERRO
S
RR, p=1
start(rej)
start(p)
retr_cnt:=+1
reset v(s)
AWAIT REJECT
ack_timer expired
p_flag=0
N
ERRO
S
retr_cnt<n2
N
ERRO
S
RR, p=1
start(rej)
start(p)
retr_cnt:=+1
reset v(s)
AWAIT REJECT
LDCr
rem_busy=0
N
S
I, x=0
REJECT
LDCc
(ST rem. busy
REJETC


AWAIT
I
update n(r)
update v(s)
n(s) ok
S
N
x=1
S
x=p
N
N
S
REJ
x=0
REJ
f=1
REJ
x=0
start(rej)
start(p)
start(rej)
stop(p)
start(rej)
AWAIT
REJECT
AWAIT
REJECT
Reenviar I
x=0
clear_rem_busy
REJECT
v(r):=+1
LDCi(I)
x=1
S
x=p
N
N
S
RR
x=0
RR
f=1
Reenviar I
ou RR
x=0
clear_rem_busy
stop(p)
AWAIT
AWAIT
NORMAL


AWAIT
RR/REJ
p_timer
expired
LDCr
update v(s)
update n(r)
clear_rem_busy
retr_cnt<n2
N
ERRO
LDCc
(refuse)
S
x=0
S
AWAIT
RR, p=1
AWAIT
N
start(p)
retr_cnt:=+1
x=p
S
N
stop(p)
RR, f=1
AWAIT
reenviar I
x=0
AWAIT
NORMAL


AWAIT
REJECT
I
update n(r)
update v(s)
n(s) ok
S
N
x=0
N
x=p
N
S
S
AWAIT
REJECT
RR
f=1
AWAIT
REJECT
stop (p)
reenviar I
x=0
clr_rem_busy
REJECT
v(r):=+1
stop(rej)
LDCi (I)
x=0
N
x=p
N
S
S
RR
x=0
RR
f=1
AWAIT
AWAIT
stop(p)
clr_rem_busy
reenviar I
ou RR
x=0
NORMAL


AWAIT REJECT
RR / REJ
p_timer expired
rej_timer expired
LDCr
update n(r)
update v(s)
clr_rem_busy
retr_cnt<n?
N
ERRO
S
LDCr (refuse)
X=0
S
AWAIT REJECT
REJ, p=1
retr_cnt:=+1
AWAIT REJECT
N
x=p
S
retr_cnt:=+1
start(p)
REJ, p=1
N
stop(p)
RR, f=1
AWAIT REJECT
start(p)
start(rej)
reenviar I
x=0
AWAIT REJECT
AWAIT REJECT
REJECT


ERRO
p_flag:=0
retr_cnt:=0
cause_flag:=0
DISC
p=0
start(ack)
stop(all others)
rep_sts
(frmr time-out)
D_CONN)

Apêndice B

Especificações dos modelos de simulação

B.1 Modelo de Simulação - Primeira aproximação

```
RESQ3 Translator V03.1992.01.15 Date and Time: Thu Mar 18 18:35:41 1993


*     1* 0* /* ********************************************************** */
*     2* 0* MODEL:llc_sim
*     3* 0*     METHOD:SIMULATION
*     4* 0*     NUMERIC PARAMETER:apl_arr
*     5* 0*     NUMERIC PARAMETER:pr pt pb3
*     6* 0*     NUMERIC PARAMETER:cpu_serv
*     7* 0*     NUMERIC IDENTIFIER:typ apl_typ llc_typ mac_typ
*     8* 0*     TYP: 0
*     9* 0*     APL_TYP:1
*    10* 0*     LLC_TYP:2
*    11* 0*     MAC_TYP:3
*    12* 0*     NUMERIC IDENTIFIER:llc_arr mac_arr
```

```
*    13* 0*     llc_arr:apl_arr/pt
*    14* 0*     mac_arr:apl_arr/pr
*    15* 0*     NUMERIC IDENTIFIER:a b1 b2 c
*    16* 0*     a:1/(1+pr)
*    17* 0*     b1:pr/(1+pr+pt)
*    18* 0*     b2:1/(1+pr+pt)
*    19* 0*     c:pr/(1+pr)
*    20* 0* QUEUE:cpu
*    21* 0* TYPE:FCFS
*    22* 0* CLASS LIST:cpu_queue
*    23* 0*     WORK DEMANDS:cpu_serv
*    24* 0* QUEUE:apl_t
*    25* 0*     TYPE: PASSIVE
*    26* 0*     TOKENS:1
*    27* 0*     DSPL:FCFS
*    28* 0*     ALLOCATE NODE LIST:apl
*    29* 0*        NUMBER OF TOKENS TO ALLOCATE: 1
*    30* 0*     DESTROY NODE LIST:del_apl
*    31* 0*     CREATE NODE LIST:cre_apl
*    32* 0*        NUMBER OF TOKENS TO CREATE:1
*    33* 0* QUEUE:llc_t
*    34* 0*     TYPE: PASSIVE
*    35* 0*     TOKENS:0
*    36* 0*     DSPL:FCFS
*    37* 0*     ALLOCATE NODE LIST:llc
*    38* 0*        NUMBER OF TOKENS TO ALLOCATE: 1
*    39* 0*     DESTROY NODE LIST:del_llc
*    40* 0*     CREATE NODE LIST:cre_llc
*    41* 0*        NUMBER OF TOKENS TO CREATE: 1
*    42* 0* QUEUE:mac_t
*    43* 0*     TYPE: PASSIVE
*    44* 0*     TOKENS:0
*    45* 0*     DSPL:FCFS
*    46* 0*     ALLOCATE NODE LIST:mac
*    47* 0*        NUMBER OF TOKENS TO ALLOCATE: 1
*    48* 0*     DESTROY NODE LIST:del_mac
*    49* 0*     CREATE NODE LIST:cre_mac
```

```
*    50* 0*       NUMBER OF TOKENS TO CREATE: 1
*    51* 0* SET NODE:set_apl
*    52* 0*    ASSIGNMENT LIST:jv(typ)=apl_typ
*    53* 0* SET NODE:set_llc
*    54* 0*    ASSIGNMENT LIST:jv(typ)=llc_typ
*    55* 0* SET NODE:set_mac
*    56* 0*    ASSIGNMENT LIST:jv(typ)=mac_typ
*    57* 0* SPLIT NODE:spli_no
*    58* 0* DUMMY NODE:cpu_0_3
*    59* 0* DUMMY NODE:cpu_1
*    60* 0* DUMMY NODE:sched
*    61* 0* DUMMY NODE:cpu_2
*    62* 0* DUMMY NODE:cpu_a_l
*    63* 0* DUMMY NODE:cpu_l_m
*    64* 0* DUMMY NODE:cpu_a_m
*    65* 0* DUMMY NODE:apl_tmp
*    66* 0* DUMMY NODE:llc_tmp
*    67* 0* DUMMY NODE:mac_tmp
*    68* 0* DUMMY NODE:dest_apl
*    69* 0* DUMMY NODE:dest_llc
*    70* 0* DUMMY NODE:dest_mac
*    71* 0* DUMMY NODE:destino
*    72* 0* CHAIN:cadeia
*    73* 0*    TYPE:OPEN
*    74* 0*    SOURCE LIST:apl_src
*    75* 0*       ARRIVAL TIMES:apl_arr
*    76* 0*    SOURCE LIST:llc_src
*    77* 0*       ARRIVAL TIMES:llc_arr
*    78* 0*    SOURCE LIST:mac_src
*    79* 0*       ARRIVAL TIMES:mac_arr
*    80* 0*    :apl -> cpu_queue ;1.0
*    81* 0*    :llc -> cpu_queue ;1.0
*    82* 0*    :mac -> cpu_queue ;1.0
*    83* 0*    :cpu_queue -> del_llc ;if (jv(typ)=llc_typ)
*    84* 0*    :apl_src -> set_apl ;1.0
*    85* 0*    :set_apl -> apl ;1.0
*    86* 0*    :llc_src -> set_llc ;1.0
```

```
*    87* 0*     :set_llc -> llc ;1.0
*    88* 0*     :mac_src -> set_mac ;1.0
*    89* 0*     :set_mac -> mac ;1.0
*    90* 0*     :cpu_queue -> del_apl ;if (jv(typ)=apl_typ)
*    91* 0*     :cpu_queue -> del_mac ;if (jv(typ)=mac_typ)
*    92* 0*     :del_llc -> sched ;1.0
*    93* 0*     :sched -> cpu_0_3 ; if ((ql(apl)>0 and ql(llc)>0 and ql(mac)>0) or +
                                        (ql(apl)=0 and ql(llc)=0 and ql(mac)=0) )
*    94* 0*     :sched -> cpu_1 ;if ((ql(apl)>0 and ql(llc)=0 and ql(mac)=0) or    +
                                     (ql(apl)=0 and ql(llc)>0 and ql(mac)=0) or    +
                                     (ql(apl)=0 and ql(llc)=0 and ql(mac)>0))
*    95* 0*     :sched -> cpu_2 ;if ((ql(apl)>0 and ql(llc)>0 and ql(mac)=0) or    +
                                     (ql(apl)>0 and ql(llc)=0 and ql(mac)>0) or    +
                                     (ql(apl)=0 and ql(llc)>0 and ql(mac)>0) )
*    96* 0*     :cpu_2 -> cpu_a_l ;if (ql(mac)=0)
*    97* 0*     :cpu_2 -> cpu_l_m ;if (ql(apl)=0)
*    98* 0*     :cpu_2 -> cpu_a_m ;if (ql(llc)=0)
*    99* 0*     :cpu_0_3 -> cre_apl ;1
*   100* 0*     :cpu_0_3 -> cre_llc ;0
*   101* 0*     :cpu_0_3 -> cre_mac ;0
*   102* 0*     :cpu_1 -> cre_apl ;if (ql(apl)>0)
*   103* 0*     :cpu_1 -> cre_llc ;if (ql(llc)>0)
*   104* 0*     :cpu_1 -> cre_mac ;if (ql(mac)>0)
*   105* 0*     :cpu_a_l -> cre_apl ;0.5
*   106* 0*     :cpu_a_l -> cre_llc ;0.5
*   107* 0*     :cpu_l_m -> cre_llc ;0.5
*   108* 0*     :cpu_l_m -> cre_mac ;0.5
*   109* 0*     :cpu_a_m -> cre_apl ;0.5
*   110* 0*     :cpu_a_m -> cre_mac ;0.5
*   111* 0*     :del_apl -> sched ;1.0
*   112* 0*     :del_mac -> sched ;1.0
*   113* 0*     :dest_apl -> llc_tmp ;a
*   114* 0*     :dest_apl -> SINK ;1-a
*   115* 0*     :dest_llc -> apl_tmp ;(b1-pb3/2)
*   116* 0*     :dest_llc -> spli_no ;pb3
*   117* 0*     :dest_llc -> mac_tmp ;(b2-pb3/2)
*   118* 0*     :dest_llc -> SINK ;(1-b1-b2)
```

```
*  119* 0*     :dest_mac -> llc_tmp ;c
*  120* 0*     :dest_mac -> SINK ;1-c
*  121* 0*     :spli_no -> apl_tmp ;SPLIT
*  122* 0*     :spli_no -> mac_tmp ;SPLIT
*  123* 0*     :cre_apl -> destino ;1.0
*  124* 0*     :cre_llc -> destino ;1.0
*  125* 0*     :cre_mac -> destino ;1.0
*  126* 0*     :destino -> dest_apl ;if (jv(typ)=apl_typ)
*  127* 0*     :destino -> dest_llc ;if (jv(typ)=llc_typ)
*  128* 0*     :destino -> dest_mac ;if (jv(typ)=mac_typ)
*  129* 0*     :apl_tmp -> set_apl ;1.0
*  130* 0*     :llc_tmp -> set_llc ;1.0
*  131* 0*     :mac_tmp -> set_mac ;1.0
*  132* 0* CONFIDENCE INTERVAL METHOD:NONE
*  133* 0* INITIAL STATE DEFINITION-
*  134* 0* RUN LIMITS-
*  135* 0*    EVENTS:30000
*  136* 0* LIMIT - CP SECONDS: 100
*  137* 0* TRACE:NO
*  138* 0* END

 NO FATAL ERRORS DETECTED DURING COMPILATION.
```

SETUP CROSS REFERENCE AND IDENTIFIER INFORMATION

NAME	LINE	ELEMENT TYPE	AFFILIATION	CONTAINING SUBMODEL	DIM
APL_ARR	4	NUMERIC PARAMETER (S)		llc_sim	0
PR	5	NUMERIC PARAMETER (S)		llc_sim	0
PT	5	NUMERIC PARAMETER (S)		llc_sim	0
PB3	5	NUMERIC PARAMETER (S)		llc_sim	0
CPU_SERV	6	NUMERIC PARAMETER (S)		llc_sim	0
TYP	7	NUMERIC IDENTIFIER		llc_sim	0
APL_TYP	7	NUMERIC IDENTIFIER		llc_sim	0
LLC_TYP	7	NUMERIC IDENTIFIER		llc_sim	0

MAC_TYP	7	NUMERIC IDENTIFIER		llc_sim	0
LLC_ARR	12	NUMERIC IDENTIFIER		llc_sim	0
MAC_ARR	12	NUMERIC IDENTIFIER		llc_sim	0
A	15	NUMERIC IDENTIFIER		llc_sim	0
B1	15	NUMERIC IDENTIFIER		llc_sim	0
B2	15	NUMERIC IDENTIFIER		llc_sim	0
C	15	NUMERIC IDENTIFIER		llc_sim	0
CPU	20	ACTIVE QUEUE		llc_sim	0
CPU_QUEUE	22	CLASS	CPU	llc_sim	0
APL_T	24	PASSIVE QUEUE		llc_sim	0
APL	28	ALLOCATE NODE	APL_T	llc_sim	0
DEL_APL	30	DESTROY NODE	APL_T	llc_sim	0
CRE_APL	31	CREATE NODE	APL_T	llc_sim	0
LLC_T	33	PASSIVE QUEUE		llc_sim	0
LLC	37	ALLOCATE NODE	LLC_T	llc_sim	0
DEL_LLC	39	DESTROY NODE	LLC_T	llc_sim	0
CRE_LLC	40	CREATE NODE	LLC_T	llc_sim	0
MAC_T	42	PASSIVE QUEUE		llc_sim	0
MAC	46	ALLOCATE NODE	MAC_T	llc_sim	0
DEL_MAC	48	DESTROY NODE	MAC_T	llc_sim	0
CRE_MAC	49	CREATE NODE	MAC_T	llc_sim	0
SET_APL	51	SET NODE		llc_sim	0
SET_LLC	53	SET NODE		llc_sim	0
SET_MAC	55	SET NODE		llc_sim	0
SPLI_NO	57	SPLIT NODE		llc_sim	0
CPU_0_3	58	DUMMY NODE		llc_sim	0
CPU_1	59	DUMMY NODE		llc_sim	0
SCHED	60	DUMMY NODE		llc_sim	0
CPU_2	61	DUMMY NODE		llc_sim	0
CPU_A_L	62	DUMMY NODE		llc_sim	0
CPU_L_M	63	DUMMY NODE		llc_sim	0
CPU_A_M	64	DUMMY NODE		llc_sim	0
APL_TMP	65	DUMMY NODE		llc_sim	0
LLC_TMP	66	DUMMY NODE		llc_sim	0
MAC_TMP	67	DUMMY NODE		llc_sim	0
DEST_APL	68	DUMMY NODE		llc_sim	0
DEST_LLC	69	DUMMY NODE		llc_sim	0

```
DEST_MAC      70   DUMMY NODE                          llc_sim    0
DESTINO       71   DUMMY NODE                          llc_sim    0
CADEIA        72   OPEN CHAIN                          llc_sim    0
APL_SRC       74   SOURCE NODE                         llc_sim    0
LLC_SRC       76   SOURCE NODE                         llc_sim    0
MAC_SRC       78   SOURCE NODE                         llc_sim    0
```

```
ACTUAL TABLE SIZES USED        SIZE
---------------------------- -----
Symbol table          SYMSIZ=    52
Routing table         RTBSIZ=    52
Expression table      EXPSIZ=   204
Element vector table  ELVSIZ=   594
Longest line size     LINSIZ=   222
```

B.2 Modelo de Simulação - Considerando a troca de contexto entre processos

```
RESQ3 Translator V03.1992.01.15 Date and Time: Tue Apr 20 11:21:52 1993
```

```
*     1* 0* /* ************************************************** */
*     2* 0* MODEL:llc_c_s
*     3* 0*     METHOD:SIMULATION
*     4* 0*     NUMERIC PARAMETER:apl_arr
*     5* 0*     NUMERIC PARAMETER:pr pt pb3
*     6* 0*     NUMERIC PARAMETER:cpu_serv
*     7* 0*     NUMERIC PARAMETER:c_s_time
*     8* 0*     NUMERIC IDENTIFIER:typ apl_typ llc_typ mac_typ
*     9* 0*     TYP: 0
*    10* 0*     APL_TYP:1
```

```
*    11* 0*     LLC_TYP:2
*    12* 0*     MAC_TYP:3
*    13* 0*     NUMERIC IDENTIFIER:llc_arr mac_arr
*    14* 0*     llc_arr:apl_arr/pt
*    15* 0*     mac_arr:apl_arr/pr
*    16* 0*     NUMERIC IDENTIFIER:a b1 b2 c
*    17* 0*     a:1/(1+pr)
*    18* 0*     b1:pr/(1+pr+pt)
*    19* 0*     b2:1/(1+pr+pt)
*    20* 0*     c:pr/(1+pr)
*    21* 0* QUEUE:cpu
*    22* 0* TYPE:FCFS
*    23* 0* CLASS LIST:cpu_queue
*    24* 0*     WORK DEMANDS:cpu_serv
*    25* 0* QUEUE:C_S_queue
*    26* 0* TYPE: IS
*    27* 0* CLASS LIST:C_S
*    28* 0*     WORK DEMANDS:constant(c_s_time)
*    29* 0* QUEUE:apl_t
*    30* 0*     TYPE: PASSIVE
*    31* 0*     TOKENS:1
*    32* 0*     DSPL:FCFS
*    33* 0*     ALLOCATE NODE LIST:apl
*    34* 0*        NUMBER OF TOKENS TO ALLOCATE: 1
*    35* 0*     DESTROY NODE LIST:del_apl
*    36* 0*     CREATE NODE LIST:cre_apl
*    37* 0*        NUMBER OF TOKENS TO CREATE:1
*    38* 0* QUEUE:llc_t
*    39* 0*     TYPE: PASSIVE
*    40* 0*     TOKENS:0
*    41* 0*     DSPL:FCFS
*    42* 0*     ALLOCATE NODE LIST:llc
*    43* 0*        NUMBER OF TOKENS TO ALLOCATE: 1
*    44* 0*     DESTROY NODE LIST:del_llc
*    45* 0*     CREATE NODE LIST:cre_llc
*    46* 0*        NUMBER OF TOKENS TO CREATE: 1
*    47* 0* QUEUE:mac_t
```

```
*   48* 0*     TYPE: PASSIVE
*   49* 0*     TOKENS:0
*   50* 0*     DSPL:FCFS
*   51* 0*     ALLOCATE NODE LIST:mac
*   52* 0*        NUMBER OF TOKENS TO ALLOCATE: 1
*   53* 0*     DESTROY NODE LIST:del_mac
*   54* 0*     CREATE NODE LIST:cre_mac
*   55* 0*        NUMBER OF TOKENS TO CREATE: 1
*   56* 0* SET NODE:set_apl
*   57* 0*     ASSIGNMENT LIST:jv(typ)=apl_typ
*   58* 0* SET NODE:set_llc
*   59* 0*     ASSIGNMENT LIST:jv(typ)=llc_typ
*   60* 0* SET NODE:set_mac
*   61* 0*     ASSIGNMENT LIST:jv(typ)=mac_typ
*   62* 0* SPLIT NODE:spli_no
*   63* 0* DUMMY NODE:cpu_0_3
*   64* 0* DUMMY NODE:cpu_1
*   65* 0* DUMMY NODE:sched
*   66* 0* DUMMY NODE:cpu_2
*   67* 0* DUMMY NODE:cpu_a_l
*   68* 0* DUMMY NODE:cpu_l_m
*   69* 0* DUMMY NODE:cpu_a_m
*   70* 0* DUMMY NODE:apl_tmp
*   71* 0* DUMMY NODE:llc_tmp
*   72* 0* DUMMY NODE:mac_tmp
*   73* 0* DUMMY NODE:dest_apl
*   74* 0* DUMMY NODE:dest_llc
*   75* 0* DUMMY NODE:dest_mac
*   76* 0* DUMMY NODE:destino
*   77* 0* CHAIN:cadeia
*   78* 0*     TYPE:OPEN
*   79* 0*     SOURCE LIST:apl_src
*   80* 0*        ARRIVAL TIMES:apl_arr
*   81* 0*     SOURCE LIST:llc_src
*   82* 0*        ARRIVAL TIMES:llc_arr
*   83* 0*     SOURCE LIST:mac_src
*   84* 0*        ARRIVAL TIMES:mac_arr
```

```
*    85* 0*     :cpu_queue -> del_llc ;if (jv(typ)=llc_typ)
*    86* 0*     :apl_src -> set_apl ;1.0
*    87* 0*     :set_apl -> apl ;1.0
*    88* 0*     :llc_src -> set_llc ;1.0
*    89* 0*     :set_llc -> llc ;1.0
*    90* 0*     :mac_src -> set_mac ;1.0
*    91* 0*     :set_mac -> mac ;1.0
*    92* 0*     :cpu_queue -> del_apl ;if (jv(typ)=apl_typ)
*    93* 0*     :cpu_queue -> del_mac ;if (jv(typ)=mac_typ)
*    94* 0*     :del_llc -> sched ;1.0
*    95* 0*     :sched -> cpu_0_3 ; if ((ql(apl)>0 and ql(llc)>0 and ql(mac)>0) or +
                                        (ql(apl)=0 and ql(llc)=0 and ql(mac)=0) )
*    96* 0*     :sched -> cpu_1 ;if ((ql(apl)>0 and ql(llc)=0 and ql(mac)=0) or    +
                                    (ql(apl)=0 and ql(llc)>0 and ql(mac)=0) or    +
                                    (ql(apl)=0 and ql(llc)=0 and ql(mac)>0))
*    97* 0*     :sched -> cpu_2 ;if ((ql(apl)>0 and ql(llc)>0 and ql(mac)=0) or    +
                                    (ql(apl)>0 and ql(llc)=0 and ql(mac)>0) or    +
                                    (ql(apl)=0 and ql(llc)>0 and ql(mac)>0) )
*    98* 0*     :cpu_2 -> cpu_a_l ;if (ql(mac)=0)
*    99* 0*     :cpu_2 -> cpu_l_m ;if (ql(apl)=0)
*   100* 0*     :cpu_2 -> cpu_a_m ;if (ql(llc)=0)
*   101* 0*     :cpu_0_3 -> cre_apl ;1
*   102* 0*     :cpu_0_3 -> cre_llc ;0
*   103* 0*     :cpu_0_3 -> cre_mac ;0
*   104* 0*     :cpu_1 -> cre_apl ;if (ql(apl)>0)
*   105* 0*     :cpu_1 -> cre_llc ;if (ql(llc)>0)
*   106* 0*     :cpu_1 -> cre_mac ;if (ql(mac)>0)
*   107* 0*     :cpu_a_l -> cre_apl ;0.5
*   108* 0*     :cpu_a_l -> cre_llc ;0.5
*   109* 0*     :cpu_l_m -> cre_llc ;0.5
*   110* 0*     :cpu_l_m -> cre_mac ;0.5
*   111* 0*     :cpu_a_m -> cre_apl ;0.5
*   112* 0*     :cpu_a_m -> cre_mac ;0.5
*   113* 0*     :del_apl -> sched ;1.0
*   114* 0*     :del_mac -> sched ;1.0
*   115* 0*     :dest_apl -> llc_tmp ;a
*   116* 0*     :dest_apl -> SINK ;1-a
```

```
*   117* 0*     :dest_llc -> apl_tmp ;(b1-pb3/2)
*   118* 0*     :dest_llc -> spli_no ;pb3
*   119* 0*     :dest_llc -> mac_tmp ;(b2-pb3/2)
*   120* 0*     :dest_llc -> SINK ;(1-b1-b2)
*   121* 0*     :dest_mac -> llc_tmp ;c
*   122* 0*     :dest_mac -> SINK ;1-c
*   123* 0*     :spli_no -> apl_tmp ;SPLIT
*   124* 0*     :spli_no -> mac_tmp ;SPLIT
*   125* 0*     :cre_apl -> destino ;1.0
*   126* 0*     :cre_llc -> destino ;1.0
*   127* 0*     :cre_mac -> destino ;1.0
*   128* 0*     :destino -> dest_apl ;if (jv(typ)=apl_typ)
*   129* 0*     :destino -> dest_llc ;if (jv(typ)=llc_typ)
*   130* 0*     :destino -> dest_mac ;if (jv(typ)=mac_typ)
*   131* 0*     :apl_tmp -> set_apl ;1.0
*   132* 0*     :llc_tmp -> set_llc ;1.0
*   133* 0*     :mac_tmp -> set_mac ;1.0
*   134* 0*     :apl -> C_S ;1.0
*   135* 0*     :llc -> C_S ;1.0
*   136* 0*     :mac -> C_S ;1.0
*   137* 0*     :C_S -> cpu_queue ;1.0
*   138* 0* CONFIDENCE INTERVAL METHOD:NONE
*   139* 0* INITIAL STATE DEFINITION-
*   140* 0* RUN LIMITS-
*   141* 0*     EVENTS:30000
*   142* 0* LIMIT - CP SECONDS: 100
*   143* 0* TRACE:NO
*   144* 0* END

 NO FATAL ERRORS DETECTED DURING COMPILATION.


SETUP CROSS REFERENCE AND IDENTIFIER INFORMATION

                    ELEMENT                                CONTAINING
NAME         LINE   TYPE                  AFFILIATION     SUBMODEL    DIM
----------   ----   ----------            -----------     --------    ---
```

APL_ARR	4	NUMERIC PARAMETER (S)		llc_c_s	0
PR	5	NUMERIC PARAMETER (S)		llc_c_s	0
PT	5	NUMERIC PARAMETER (S)		llc_c_s	0
PB3	5	NUMERIC PARAMETER (S)		llc_c_s	0
CPU_SERV	6	NUMERIC PARAMETER (S)		llc_c_s	0
C_S_TIME	7	NUMERIC PARAMETER (S)		llc_c_s	0
TYP	8	NUMERIC IDENTIFIER		llc_c_s	0
APL_TYP	8	NUMERIC IDENTIFIER		llc_c_s	0
LLC_TYP	8	NUMERIC IDENTIFIER		llc_c_s	0
MAC_TYP	8	NUMERIC IDENTIFIER		llc_c_s	0
LLC_ARR	13	NUMERIC IDENTIFIER		llc_c_s	0
MAC_ARR	13	NUMERIC IDENTIFIER		llc_c_s	0
A	16	NUMERIC IDENTIFIER		llc_c_s	0
B1	16	NUMERIC IDENTIFIER		llc_c_s	0
B2	16	NUMERIC IDENTIFIER		llc_c_s	0
C	16	NUMERIC IDENTIFIER		llc_c_s	0
CPU	21	ACTIVE QUEUE		llc_c_s	0
CPU_QUEUE	23	CLASS	CPU	llc_c_s	0
C_S_QUEUE	25	ACTIVE QUEUE		llc_c_s	0
C_S	27	CLASS	C_S_QUEUE	llc_c_s	0
APL_T	29	PASSIVE QUEUE		llc_c_s	0
APL	33	ALLOCATE NODE	APL_T	llc_c_s	0
DEL_APL	35	DESTROY NODE	APL_T	llc_c_s	0
CRE_APL	36	CREATE NODE	APL_T	llc_c_s	0
LLC_T	38	PASSIVE QUEUE		llc_c_s	0
LLC	42	ALLOCATE NODE	LLC_T	llc_c_s	0
DEL_LLC	44	DESTROY NODE	LLC_T	llc_c_s	0
CRE_LLC	45	CREATE NODE	LLC_T	llc_c_s	0
MAC_T	47	PASSIVE QUEUE		llc_c_s	0
MAC	51	ALLOCATE NODE	MAC_T	llc_c_s	0
DEL_MAC	53	DESTROY NODE	MAC_T	llc_c_s	0
CRE_MAC	54	CREATE NODE	MAC_T	llc_c_s	0
SET_APL	56	SET NODE		llc_c_s	0
SET_LLC	58	SET NODE		llc_c_s	0
SET_MAC	60	SET NODE		llc_c_s	0
SPLI_NO	62	SPLIT NODE		llc_c_s	0
CPU_0_3	63	DUMMY NODE		llc_c_s	0

CPU_1	64	DUMMY NODE	llc_c_s	0
SCHED	65	DUMMY NODE	llc_c_s	0
CPU_2	66	DUMMY NODE	llc_c_s	0
CPU_A_L	67	DUMMY NODE	llc_c_s	0
CPU_L_M	68	DUMMY NODE	llc_c_s	0
CPU_A_M	69	DUMMY NODE	llc_c_s	0
APL_TMP	70	DUMMY NODE	llc_c_s	0
LLC_TMP	71	DUMMY NODE	llc_c_s	0
MAC_TMP	72	DUMMY NODE	llc_c_s	0
DEST_APL	73	DUMMY NODE	llc_c_s	0
DEST_LLC	74	DUMMY NODE	llc_c_s	0
DEST_MAC	75	DUMMY NODE	llc_c_s	0
DESTINO	76	DUMMY NODE	llc_c_s	0
CADEIA	77	OPEN CHAIN	llc_c_s	0
APL_SRC	79	SOURCE NODE	llc_c_s	0
LLC_SRC	81	SOURCE NODE	llc_c_s	0
MAC_SRC	83	SOURCE NODE	llc_c_s	0

ACTUAL TABLE SIZES USED		SIZE
Symbol table	SYMSIZ=	55
Routing table	RTBSIZ=	53
Expression table	EXPSIZ=	208
Element vector table	ELVSIZ=	606
Longest line size	LINSIZ=	222

Bibliografia

[ADA 1] Ichbiah, J.D. "Preliminary ADA Reference Manual", Sigplan Notices, jun 1979.

[APCC 2] "APCC 4.0 - Documentação do Usuário", Volumes I e II, CPqD - Telebrás, 1991.

[Araujo 3] Araujo, G.B., A. Yamakami, S. Motoyama: "Implementação da sub-camada MAC em uma rede com integração de serviços utilizando circuitos integrados dedicados", XI Simpósio Brasileiro de Redes de Computadores, Campinas, SP, 1993.

[Brinch 4] Brinch Hansen, P. "Structured multiprogramming", Communications ACM, jul. 1972.

[Brinch 5] Brinch Hansen, P. "Operating System Principles", Prentice-Hall, 1973

[Carneiro 6] Carneiro M.C.C.: "Rede local de computadores multiserviços: proposição de uma arquitetura e implementação da camada sinalização", Tese de mestrado apresentada na Universidade Federal de São Carlos, SP, 1991.

[CCITT 7] IEEE 802.2 - 1985 (ISO/DIS 8802/2) Local Area Network - 802.2 Logical Link Control, 1985.

[CCITT 8] "Reference model of open systems interconnection for CCITT applications" Provisional Recomendation X.200, set 1982.

[CHILL 9] "Manual de Programação CHILL para o PP-SO/P" CPqD - Telebrás, 1990.

[CHILL 10] "CCITT High Level Language (CHILL) - Recommendation Z.200 - ISO/IEC 9496", CCITT, 1988.

[Davies 11] Davies D.W., Barber D.L.A.: "Communication Networks for Computers", John Wiley, 1973.

[Dijkstra 12] Dijkstra E.W. "Cooperating sequential process, THE, The Nederlands, 1965.

[Eglowstein 13] Eglowstein H., Diehl S.: "The Multiuser Solution", BYTE, sep. 1989.

[Ferreira 14] Ferreira A.R.Q, Paro E., Ferreira J., Corradi N., Gurtler P.V., Cotrim Filho R.F.:"The Experience of Using CHILL in the Implementation of a Text Communication Exchange", Proceedings of 5th CHILL Conference, RJ, Brasil, 1990.

[Fiedler 15] Fiedler D.:"Unix on Personal Computers: Why and How", BYTE, Setembro 1989.

[Franco 16] Franco, J.H.A., Melo E.B.:"The Use of CHILL in the Tropico-RA Switchig System", Proceedings of 5th CHILL Conference, RJ, Brasil, 1990.

[Giozza 17] Moura J.A.B., Sauvé J.P.,Giozza W. F. , Araújo J.F.M. : "Redes Locais de Computadores - Protocolos de Alto Nível e Avaliação de Desempenho", McGraw-Hill, 1986.

[Guardiero 18] Guardiero P.R.: "Um método de controle de acesso para rede local com fibras ópticas e integração de voz e dados", Tese de doutorado apresentada na FEE, UNICAMP, dezembro 1991.

[Hallsteinsen 19] Venstad A., Hallsteinsen S., Martinsen H., Nyeng A.:"Transformational program development with SDL and CHILL", Proceedings of 5th CHILL Conference, RJ, Brasil, 1990.

[Hallsteinsen 20] Venstad A., Hallsteinsen S., Martinsen H., Nyeng A.:"Transformational program development - An approach for translating SDL to CHILL", SDL'89 The Language at Work, North Holland, 1989.

[Halsall 21] Halsall, F.:"Data Communications, computer networks and OSI", Addison-Wesley Publishing Company, 1988.

[Hammond 22] Hammond J.L, O'Rellly P.J.P.:"Performance Analysis of Local Computer Networks".

[Hari 23] Hari G., Umashankar V., Sivasubramanian P., Rajagopalan S.:" Run Time System for CCITT High Level Language", Proceedings of 5th CHILL Conference, RJ, Brasil, 1990.

[Hao 24] Hao W., Guo Z., Kou H., Xi D.:"The Design and Implementation of the Operating System Kernel OSK of a CHILL Supporting System", Proceedings of 5th CHILL Conference, RJ, Brasil, 1990.

[Harland 25] Harland D.M. "On facilities for Interprocess Communication", Information Processing Communication, oct. 1981.

[Hoare 26] Hoare C.A.R. "Towards a theory of parallel programming", Operating System Techniques, Academic Press, N.Y., 1972.

[Hoare 27] Hoare C.A.R." Communicating Sequential Processes", Communications ACM, aug. 1978.

[Hopper 28] Hopper, A., Williamson, R. C.: " Design and Use of a Integrate Cambridge Ring", IEEE Journal on Selected Areas in Communication, vol SAC-1 november, 1983, p775-784.

[ISO 29] ISO/TC97/SC16 - Documento N34. "Provisional model of open systems architecture", mar. 1978.

[ISO 30] ISO/TC97/SC16 - Documento N227. "Reference model of open systems architecture", jun. 1979.

[ISO 31] "Basic reference model for open systems interconnection" ISO 7498, 1983.

[Johansen 32] Johansen U., Vefsnmo: "Automatic Program Generation of SDL Specifications. Principles and Solutions", Proceedings of the 3rd SDL Forum, North Holland, 1987.

[KERMIT 33] "Manual do usuário do utilitário KERMIT", Relatório Interno do CPqD/Telebrás, abr. 1990.

[Kleinrock 34] Kleinrock L., Coffman E.G.:"Feedback Queuing Models for Time-Shared Systems", Journal of the Association for Computes Machinery, Vol 15, No4, Outubro 1968.

[Kong 35] Kong E.C.: "An Extensible Approach to Automatic Program Generation in Telecommunications Applications" Tese de Mestrado, RMIT Department of Computing, jan. 1987.

[Loewner 36] Gordon R.F., Loewner P.G., MacNair E.A. "RA 210 The Research Queueing Package Version 3 - Language Reference Manual", IBM Thomas J. Watson Research Center, mar. 1992.

[Madeira 37] Madeira, E.M.:"Implementação do protocolo X-25 num concentrador de comunicações baseado no 8080", Relatório Interno 323, IMECC, UNICAMP, jul. 1985.

[McKinney 38] McKinney J.M.:"A Survey of Analytical Time-Shared Models", Computing Surveys, Vol1, No 2, Junho 1969.

[Pessoa 39] Pessoa P.M.C.: "Implementação de uma interface de voz para rede local com fibras ópticas e integração de voz e dados", Tese de mestrado apresentada na FEE, UNICAMP, dez. 1991.

[Peterson 40] Peterson J.L., Silberschatz A.:"Operating System Concepts", 2nd Edition, Addison-Wesley Publishing Co, 1987.

[PP 41] "Processador Preferencial - Relatório Interno, CPqD - Telebrás, 1984.

[Richer 42] Richer I., Steiner M., Seigoku M.: "Office Communications and the Digital PBX", Computer Networks, December 1980, pages 411-422.

[Sobrinho 43] Sobrinho S.C., Araújo C.C., Luz P.R., Cavalli E., Rolim L.A.: "A Real-Time Operating System for Distributed Applications with CHILL Execution Environment Support", "Proceedings of the 5th CHILL Conference - Rio de Janeiro, Brazil, 19-22 March, North-Holland, 1991".

[SOP 44] " Especificação do Sistema Operacional PP-SO/P - Relatório Interno, CPqD - Telebrás, 1990.

[Stuck 45] Stuck B.W., Arturs E.: "A Computer and Communications Network Performance Analysis Primer", Prentice-Hall Inc, 1985.

[Ussami 46] Ussami M.C.E., Motoyama S. :" Implementação de um protocolo de comunicação em uma rede com integração de serviços utilizando a linguagem CHILL", XI Simpósio Brasileiro de Redes de Computadores, Campinas, SP, 1993.

[Ussami 47] Ussami M.C.E., Motoyama S. :" Modelagem e Análise de Desempenho de um Protocolo de Comunicação Implementado em CHILL", submetido ao XI Simpósio Brasileiro de Telecomunicações, Natal, RN, 1993.

[Wirth 48] Wirth N. "Modula: a language for modular programming", Software practice and Experience, jan. 1977.

[Zabeu 49] Zabeu, M.C.A.:"Um modelo para configurador de núcleos para tempo real", Tese de mestrado apresentada na FEE, UNICAMP, nov. 1989.