IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO DO PROTOCOLO DE SESSÃO

José Ferreira de Rezende Otto Carlos M. B. Duarte
Grupo de Teleinformática e Automação – GTA/UFRJ
Programa de Engenharia Elétrica
COPPE/EE – Universidade Federal do Rio de Janeiro
CP. 68504 – CEP 21945 – Rio de Janeiro, RJ
coe10029@ufrj.bitnet, dirpg@brufu.bitnet

Sumário

Com o desenvolvimento das redes de computadores, principalmente das redes locais, que oferecem uma banda passante cada vez maior, o processamento das aplicações e da comunicação passou a ser o principal obstáculo na obtenção de um alto desempenho.

A partir da implementação das camadas de protocolos definidas pelo Modelo OSI, faz–se necessária uma análise do seu processamento com o objetivo de detectar e corrigir os gargalos que influem no desempenho do sistema. A arquitetura OSI e a especificação dos protocolos não são os principais fatores de limitação do desempenho que é extremamente dependente da implementação.

Este trabalho descreve os serviços fornecidos pela camada Sessão, assim como, algumas decisões de implementação que visam alto desempenho. É apresentada uma análise de desempenho do protocolo de Sessão implementado, e os resultados indicam uma vazão de transferência de dados muito superior (10Mbits/segundo para quadros de 1024 octetos) à que as camadas mais baixas, deste sistema, podem suportar.

Abstract

With the development of computer networks, especially local area networks, that offer a bandwidth larger and larger, the applications and communications processing have been the major obstacle in the obtention of high performance.

From the implementation of the layers of protocols defined by OSI Model, it is necessary an analysis of the processing in order to detect and correct the performance bottlenecks of the system. The OSI architecture and protocol specifications are not the principal factors of performance limitations that is extremely dependent of the implementation.

This work describes the services provided by the Session layer as some implementation decisions that aim at high performance. It is shown a performance evaluation of the Session protocol implemented and the results indicate a throughput in the data transfer much superior (10 Mbits/s for 1024 octets frames) than the lower layers of this system can support.

1 Introdução

Em 1979, a Organização Internacional de Padrões (*International Standards Organization* – ISO) definiu um Modelo de Referência para a

Interconexão de Sistemas Abertos (*Reference Model for Open Systems Interconnection* – RM–OSI) [1–3] com a finalidade de padronizar a interconexão de sistemas heterogêneos. Desde então, os protocolos definidos por este modelo têm sido adotados por vários países e diferentes organizações.

Vários estudos apresentados [4,5] indicam que as decisões de implementação destes protocolos influenciam no desempenho da comunicação, podendo limitar a vazão das redes locais e de longa distância. É importante ressaltar que as normas OSI não estabelecem nenhuma restrição à implementação que fica sobre a responsabilidade do implementador do sistema.

Além do ambiente onde o sistema será implementado (sistema operacional, linguagem de programação, compiladores, ...), existe um conjunto de decisões e técnicas de implementação que afetam o desempenho de execução do protocolo, tais como: interface entre as camadas, forma de adição/retirada de cabeçalhos de cada camada, forma de execução do sistema, gerenciamento de memória.

Este trabalho descreve e analisa o desempenho de uma implementação do protocolo de Sessão que visa um alto desempenho. Inicialmente, os serviços de Sessão são apresentados seguidos de uma discussão das decisões e técnicas de implementação utilizadas. Por fim, apresenta–se a análise do desempenho obtido na transferência de dados que permite estabelecer uma avaliação quantitativa pormenorizada da implementação realizada.

2 Camada Sessão

A camada Sessão é a primeira das chamadas camadas altas e provê um serviço orientado ao usuário. Os principais conceitos envolvidos na oferta destes serviços são: *tokens*, gerenciamento de diálogo, sincronização e resincronização, atividades, liberação ordenada e transferência de dados.

O serviço Sessão [6] fornece meios para organizar e sincronizar a troca de dados entre usuários cooperantes do mesmo. Através do serviço Sessão seus usuários podem:

- estabelecer conexão com outro usuário do serviço Sessão, trocar dados com este usuário de maneira sincronizada e liberar a conexão de forma ordenada;
- negociar o uso de fichas (*tokens*) para troca de dados, sincronização e liberação da conexão, e decidir se a troca de dados será *half–duplex* ou *duplex*;
- estabelecer pontos de sincronização dentro do diálogo e, no caso de ocorrência de erros, recuperá–los ao retransmitir a partir de um ponto de sincronização combinado;
- interromper um diálogo e prosseguí–lo, mais tarde, de um ponto pré–estabelecido.

Conceito de *Token*

Um *token* é um atributo de uma conexão de Sessão concedido dinamicamente a um usuário do serviço Sessão (SS) a cada instante, permitindo que certos serviços sejam invocados de maneira exclusiva.

Existem quatro tipos de *tokens* : *token* de dados, *token* de liberação, *token* de sincronização menor e *token* de sincronização maior/atividade.

O *token* está sempre em um dos seguintes estados :

- disponível. Neste caso, ele está sempre atribuído a um dos usuários, e não atribuído ao outro usuário, que pode adquirí–lo mais tarde; ou
- não disponível. Neste caso nenhum dos usuários tem o uso exclusivo do serviço associado. O serviço torna–se disponível a ambos os usuários

(transferência de dados e liberação), ou não disponível a nenhum dos usuários (sincronização e atividades).

Gerenciamento de Diálogo

Em princípio, todas as conexões do modelo OSI são *full–duplex*, isto é, PDUs podem ser trocadas em ambas as direções simultaneamente através de uma mesma conexão. No entanto, existem muitas situações em que o software das camadas superiores é estruturado de modo que cada usuário tenha a sua vez de transmitir (comunicação *half–duplex*).

O gerenciamento que designa qual dos usuários tem o direito de transmitir a cada instante é chamado de gerenciamento de diálogo. Este é um dos serviços oferecidos pela camada Sessão. O gerenciamento de diálogo é implementado pelo uso do *token* de dados. Durante a fase de estabelecimento de conexão, os usuários do SS podem combinar se a troca de dados será *half–duplex* ou não através da seleção da Unidade Funcional *Half–Duplex*. Se a operação *half–duplex* for escolhida, a negociação inicial também determina qual lado deterá o *token* inicialmente. Somente o usuário que detém o *token* pode transmitir dados.

Ao final da transmissão, o usuário detentor do *token* pode passá–lo ao outro usuário através dos serviços de gerenciamento de *tokens*.

Sincronização e Resincronização

Através de um processo chamado sincronização, os usuários do SS podem inserir pontos de sincronização entre os dados que estão transmitindo. Qualquer semântica associada, pelos usuários do SS, a esses pontos é transparente para o provedor do SS.

Na ocorrência de erro ou de desacordo entre esses usuários, é possível retornar a conexão de Sessão a um desses pontos e prosseguir a comunicação a partir dele. Este processo é chamado de resincronização.

Cada ponto de sincronização é identificado por um número serial mantido pelo provedor do SS. Existem dois tipos de pontos de sincronização: pontos de Sincronização Menor, pontos de Sincronização Maior.

As unidades de dados delimitadas pelos pontos de sincronização maior são chamadas unidades de diálogo que usualmente representam partes distintas de trabalho. Um ponto de sincronização maior indica o fim de uma unidade de diálogo e o início da próxima. Já os pontos de sincronização menor são usados para estruturar a troca de dados dentro de uma unidade de diálogo. A figura 1 ilustra como uma unidade de diálogo pode ser estruturada através do uso de pontos de sincronização menor.

Os pontos de sincronização maior e menor diferem em vários aspectos. Na resincronização não se pode retornar além do mais recente ponto de sincronização maior. Ao contrário do ponto de sincronização menor. Cada ponto de sincronização maior é explicitamente confirmado (reconhecido); já o ponto de sincronização menor pode ou não ser confirmado explicitamente. A escolha do tipo

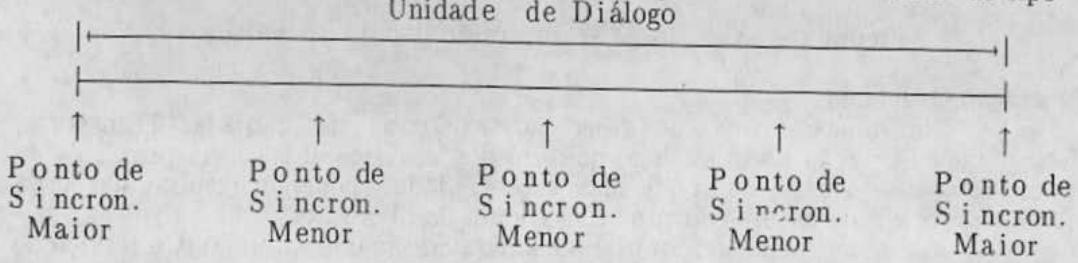

Figura 1 – Exemplo de Estruturação de uma Unidade de Diálogo.

e da localização do ponto de sincronização é uma decisão do usuário do SS.

Para se inserir pontos de sincronização maior ou menor é exigida a posse do *token* correspondente. Na resincronização, todos os *tokens* disponíveis retornam às posições ocupadas no instante em que o ponto de sincronização foi inserido.

Atividades

O conceito de atividade permite aos usuários do SS distinguir diferentes partes lógicas de trabalho que são denominadas atividades. Cada atividade consiste em uma ou mais unidades de diálogo completamente independente de quaisquer outras atividades anteriores ou posteriores, cabendo ao usuário determinar o que ela representa.

As atividades estão intimamente relacionadas aos pontos de sincronização. Quando uma atividade é iniciada, é atribuido o valor 1 ao número serial de sincronização e um ponto de sincronização maior é inserido. É possível inserir pontos de sincronização adicionais – maior ou menor – dentro de uma atividade, como mostrado na figura 2.

O início de cada atividade corresponde a um ponto de sincronização maior, portanto não é possível resincronizar a um ponto anterior ao início da atividade. Em suma, não se pode resincronizar a um ponto de sincronização de uma atividade anterior.

Para prevenir que ambos os usuários tentem iniciar uma atividade simultaneamente, o gerenciamento de atividade é controlado por um *token*, que, na verdade é o mesmo usado para pontos de sincronização maior. Então, para se invocar um serviço de atividade, o usuário deve ter em seu poder o *token* de sincronização maior/atividade.

Mesmo assim, outros problemas ocorriam quando um usuário iniciava uma atividade enquanto o outro inseria um ponto de sincronização menor. Novamente, para se prevenir esse tipo de erro estabeleceu–se que o usuário deve ter em seu poder os *tokens* de sincronização maior/atividade e de sincronização menor, assim como o *token* de dados (se disponível) antes de iniciar uma atividade ou uma operação de sincronização. Esta estratégia eliminou os problemas originais, mas originou a ocorrência de *deadlocks* [7,8]. A única solução para isso seria um projeto bastante cuidadoso das aplicações que se servem desta facilidade.

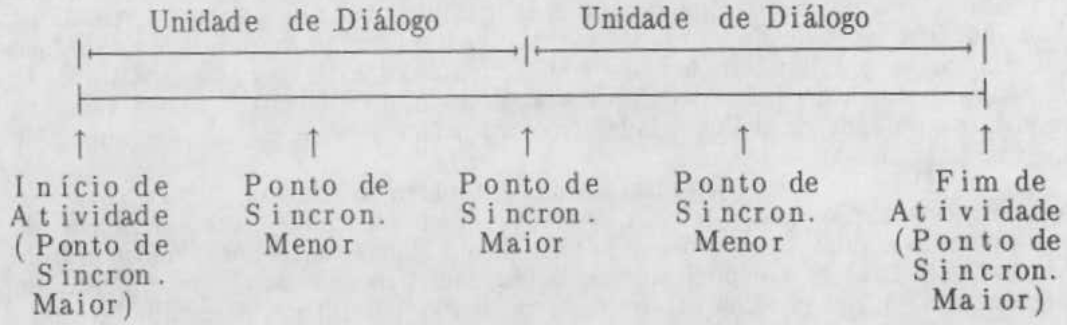

Figura 2 – Exemplo de Estruturação de uma Atividade.

Liberação Ordenada

Na única forma de liberação existente na camada Transporte, denominada liberação abrupta, logo que um dos usuários envia uma primitiva de desconexão, esse usuário não pode mais receber dados, podendo resultar na perda irrecuperável dos dados em trânsito no momento da liberação.

Além da liberação abrupta, a camada Sessão oferece um outro serviço de liberação da conexão de Sessão, denominado liberação ordenada, onde ambos os

usuários cooperantes do SS devem concordar com a liberação. Uma vez iniciado o pedido de liberação por um dos usuários, este ainda pode aceitar mensagens enviadas pelo outro usuário até que o mesmo confirme a liberação.

Transferência de Dados

Além dos dois tipos de dados permitidos pela camada Transporte (Dados Normais e Dados Expressos), a camada Sessão permite a transferência de mais dois tipos de dados: Dados Tipados e Dados Capacitados.

Os Dados Tipados são como os Dados Normais, no entanto eles podem ser enviados mesmo que a entidade não esteja de posse de qualquer um dos *tokens*. Eles podem ser usados pelo usuário do serviço Sessão para o controle de mensagens ou para qualquer outra finalidade. A intenção do comitê da ISO ao criar esse tipo de dados foi de prover uma transferência de dados *out–of–band* para o controle de informação das camadas superiores, manutenção da rede e gerenciamento do sistema. Por isso, a necessidade de que eles possam ser enviados independentemente da disponibilidade e atribuição dos *tokens*.

Os Dados Capacitados é usado para o controle da própria camada Sessão, permitindo a troca de parâmetros e opções durante a conexão de Sessão. Esses dados somente podem ser enviados entre as atividades e por isso exigem que o usuário esteja de posse dos *tokens* de dados, de sincronizacão menor e de sincronização maior/atividade.

3 Implementação do Protocolo de Sessão

Toda a implementação da camada Sessão foi realizada em computadores pessoais IBM–PC/XT/AT, utilizando–se a linguagem de programação C (Turbo C 2.0 Borland) [9,10] em conjunto com o desenvolvimento das camadas inferiores realizado pelo Grupo de Teleinformática e Automação (GTA/UFRJ).

O sistema até então desenvolvido consiste em: camada Aplicação (Elementos Comuns), camada Apresentação (Kernel sem ASN.1), camada Sessão, camada Transporte (Classe 2 e Classe 4) [11], camada Rede, camada Enlace [12] e camada Física.

Na primeira experiência de transferência de arquivos realizada pelo GTA [13], somente 1,72% da banda passante do canal era utilizada e por conseguinte o grande fator limitante do desempenho era o processamento dos protocolos. Observou–se que uma grande parte do tempo de processamento era utilizada na passagem das informações [4] que era feita por recópias. A partir desta experiência, decidiu–se por uma implementação de um sistema OSI considerando alguns fatores que afetam o desempenho. Neste sentido, propôs–se um Escalonador, um Gerenciador de Memória, um Gerenciador de Temporizações e estruturas de dados específicas que pudessem tirar proveito das particularidades de um sistema de comunicação OSI. O modelo e as estruturas utilizadas, assim como estes módulos serão descritos a seguir.

Estruturas de Dados Utilizadas

Os serviços oferecidos pelas camadas superiores (Sessão, Apresentação e Aplicação) do Modelo de Referência OSI se caracterizam por um grande número de primitivas de serviço. Onde a maioria dos parâmetros destas primitivas são opcionais e de tamanhos variáveis. Levando–se em conta esta característica, a representação dos dados trocados entre entidades adjacentes devem apresentar características que permitam o uso eficiente da memória e que facilitem o acesso dos parâmetros recebidos. Ainda, uma outra característica marcante desse modelo, consiste na adição/retirada de Informação de Controle do Protocolo (*Protocol Control Information* – PCI) às/das SDUs relativo às informações entre entidades

pares. Com a finalidade de evitar a recópia, melhorando o desempenho do sistema, fez–se necessária a utilização de técnicas de encadeamento que consistem em associar, a cada região de dados, uma estrutura que permita o seu encadeamento.

Após estudos das várias representações para primitivas e parâmetros, observando as características de cada camada do modelo de referência OSI, definiu–se as estruturas de dados básicas utilizadas na implementação. As linhas básicas do modelo geral de uma camada e as estruturas de dados utilizadas foi fruto de discussão e proposição coletiva de alguns componentes do Grupo de Teleinformática e Automação (GTA/UFRJ).

Duas estruturas de dados foram utilizadas na interface entre as camadas (figura 3): a Estrutura_Primitiva e a Estrutura_Parametros.

A Estrutura_Primitiva é uma estrutura passada de uma camada a outra, como um evento de interface do protocolo. Os campos desta estrutura descrevem a primitiva de serviço e referenciam os parâmetros da mesma. São eles:

- Nome_Primitiva – é um campo de tipo enumerado que indica o nome da primitiva de serviço;
- Ptr_Parametros_Primitiva – é um ponteiro que referencia uma outra estrutura de dados denominada Estrutura_Parametros, descrita a seguir;
- Ptr_Proximo – é um ponteiro utilizado no encadeamento dessa estrutura na fila FIFO, utilizada como comunicação entre as camadas. Ele referencia uma outra Estrutura_Primitiva, caso exista, presente nesta fila.

A Estrutura_Parametros identifica as informações trocadas entre entidades adjacentes (parâmetros) e entidades pares (Dados do Usuário). Os campos desta estrutura referenciam os parâmetros da primitiva de serviço. São eles:

- Ptr_Proximo – é um ponteiro usado para encadear Estrutura_Parametros;
- Ptr_Parametros – é um ponteiro que referencia uma estrutura de dados, diferente para cada primitiva de serviço, definida na interface entre as camadas que contêm os parâmetros desta primitiva de serviço, em particular;
- Ptr_Dados_Usuario – é um ponteiro que referencia o parâmetro Dados do Usuário do Serviço, presente na maioria das primitivas de serviço. Este parâmetro pode estar segmentado, neste caso, cada segmento deve ser referenciado por uma Estrutura_Parametros encadeada.
- Tam_Dados_Usuario – este campo indica o tamanho de cada segmento dos Dados do Usuário presentes nesta primitiva de serviço;
- Num_Alocacao – este campo indica o número de alocações dos dados desta estrutura, com a finalidade de evitar a recópia de dados para filas

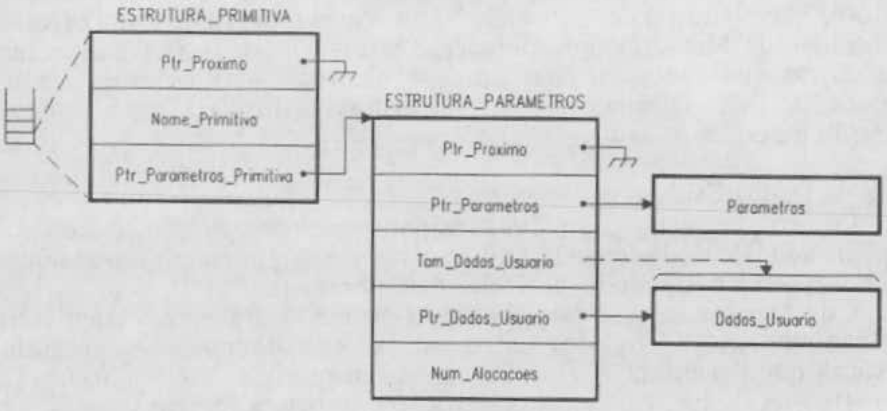

Figura 3 – Estruturas de dados utilizadas na interface entre as camadas.

de retransmissão utilizada em algumas camadas.

Para cada primitiva de serviço, que contenha parâmetros, é definida uma estrutura de dados que contém ou referencia esses parâmetros. Foi adotada uma estratégia para a definição dessas estruturas:

- Parâmetros mandatórios de tamanho fixo – é definido por um campo na estrutura de dados com o tamanho desse parâmetro;
- Parâmetros opcionais de tamanho fixo – é definido por um campo na estrutura de dados com o tamanho desse parâmetro e por uma variável booleana associada a ele que indica a sua presença ou não;
- Parâmetros mandatórios ou opcionais de tamanho variável – é definido por um ponteiro que o referencia e por uma variável que indica o seu tamanho. No caso de parâmetros opcionais, o tamanho igual a zero indica que o parâmetro não está presente.

Gerenciador de Memória

Ainda, com a finalidade de otimizar a utilização da memória disponível e evitar a recópia, garantindo um melhor desempenho, foi implementado um Gerenciador de Memória dedicado. Este módulo permite a retirada de PCIs através da liberação parcial de uma área de memória alocada anteriormente, o que não era possível através de gerenciadores convencionais.

O gerenciador de memória, comum à todas as camadas, controla três áreas diferentes: duas áreas subdivididas em blocos de tamanho fixo, e uma outra área livre onde objetos de memória de tamanho arbitrário podem ser alocados.

A primeira área de memória é reservada para alocação das estruturas de dados Estrutura_Primitiva, cujo tamanho fixo é de 9 octetos. A segunda área, controlada pelo gerenciador, é usada para a alocação das estruturas de dados Estrutura_Parametros de tamanho fixo de 15 octetos.

A subdivisão dessas áreas de memória em blocos de tamanho fixo elimina os problemas de fragmentação externa e interna. E ainda diminui os tempos de alocação e liberação dessas estruturas, contrastando com os gerenciadores de memória de sistemas operacionais convencionais que não são otimizados para velocidade de execução.

A terceira, e última área, é reservada para alocação de objetos de memória de tamanho variável de uso geral.

Como o fluxo de SDUs transmitidas é independente do fluxo das SDUs recebidas, a memória foi dividida em duas regiões: de transmissão e de recepção. Desta forma, a utilização da memória tornou-se mais ordenada, agilizando, assim, o seu gerenciamento.

Interface entre as Camadas

Duas formas de interfaceamento entre as camadas foram estudadas. São elas:

- Chamadas de Rotinas – Neste método, a ativação de uma primitiva de serviço da camada (N) é implementada pela chamada de uma rotina desta camada, realizada pela camada (N+1). Este método possivelmente apresenta um melhor desempenho, uma vez que um pacote de dados é processado pelas camadas de protocolo sem ser colocado em uma fila, o que minimiza o número de instruções para o seu processamento. No entanto, para garantir o funcionamento deste método seria necessário que as camadas de protocolos fossem reentrantes, o que dificultaria muito o trabalho de implementação.
- Envio de Mensagens – Neste método, as camadas trocam mensagens em forma de primitivas de serviço. Essa troca de mensagens pode ser realizada através de mecanismos de sistemas operacionais ou através de memória compartilhada.

A forma adotada neste sistema foi por envio de mensagens (primitivas de serviço) colocadas em filas FIFO, o que permitiu manter a independência entre as camadas e, ainda, facilita a sua migração para um sistema operacional multi–tarefa e para uma implementação multiprocessada.

Na camada Sessão, quatro filas FIFO estão associadas à cada instância: duas de primitivas de serviço Sessão, sendo uma de chegada e outra de saída e; duas de primitivas de serviço Transporte, sendo uma de chegada e outra de saída, como mostrado na figura 4. Estas filas estão definidas nos módulos de interface entre as camadas onde também estão definidos os procedimentos e funções que as manipulam.

Escalonador de Tarefas

Para a execução de todo o sistema foi necessária a implementação de um Escalonador de Tarefas, onde as tarefas representam funções das camadas do modelo OSI.

Inicialmente, partiu–se de um escalonador bem simples que consistia em realizar chamadas a todas as camadas em um *loop* infinito. Cabendo a cada entidade de uma camada realizar uma busca nas filas de interface às camadas adjacentes para verificar a chegada ou não de um evento. Quando da sua chegada, o evento era tratado e a execução retornava ao escalonador que chamava a próxima camada, e assim por diante.

Para evitar que todas as entidades de uma camada fossem obrigadas a verificar suas filas de eventos a cada chamada realizada, foi criada uma variável para cada instância. Esta variável determinava se ela estava ativa ou não, interpretando como ativas aquelas instâncias cujo estado da máquina do protocolo não estivesse mais no estado de repouso (*idle*). Com isso, apenas as instâncias ativas precisavam verificar suas filas de eventos.

Mesmo assim, não é difícil perceber que, em questão de desempenho, essa forma de escalonamento é bastante ineficiente, uma vez que há um grande desperdício de tempo em verificações das situações das filas (vazias ou não vazias) e das instâncias (ativas ou não ativas).

Então, buscando–se um melhor desempenho, partiu–se para a

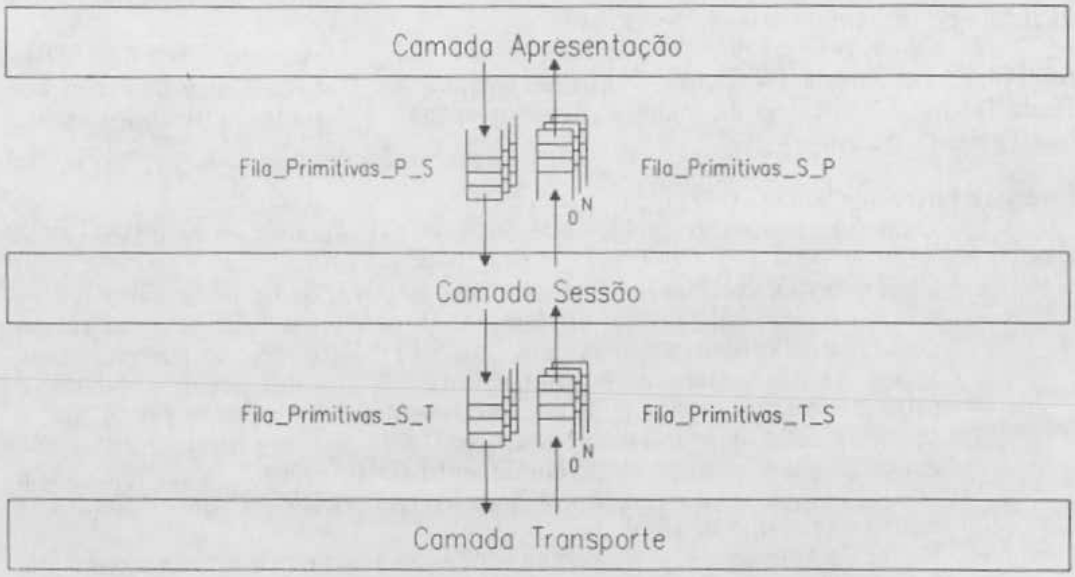

Figura 4 – Interface entre as camadas.

elaboração de um novo escalonador, num trabalho realizado por Baginski [14], que executa as tarefas a partir de uma fila de tarefas prontas para execução. Sendo tarefas prontas aquelas que receberam algum evento.

O processo de colocação de uma tarefa na fila de prontas é realizado pela tarefa que lhe enviou uma primitiva de serviço, ou, no caso da expiração de uma temporização, pelo Gerenciador de Temporizações.

Sendo assim, o escalonador consiste em uma tarefa principal que fica em *loop* infinito, verificando o usuário e executando a primeira tarefa da fila de prontas.

É preciso destacar que o escalonador também informa qual entidade da camada deverá ser executada, uma vez que ele passa essa informação para a camada indicando em qual das filas está o evento.

Gerenciador de Temporizações

Como algumas camadas do Modelo de Referência OSI (Aplicação, Sessão, Transporte e Enlace) prevêem a utilização de um mecanismo de temporização, um Gerenciador de Temporizações foi implementado. Este se baseia em interrupções periódicas (*Timer Tick* do IBM–PC) a fim de incrementar um índice de um *buffer* circular de tamanho fixo, como proposto por Varghese & Lauck [15]. A cada temporizador é associado um descritor, colocado em uma posição específica desse *buffer* circular, que, quando apontado pelo índice, ocorre o estouro da temporização.

Os valores máximos e mínimos permitidos para a temporização são dependentes do tempo decorrido entre duas interrupções periódicas e do tamanho do *buffer* circular.

Quando uma entidade deseja iniciar uma temporização, ela realiza uma chamada a uma função do gerenciador, indicando o tempo (em segundos) da temporização e a instância que deseja iniciá–la. Na ocorrência de um estouro da temporização, o Gerenciador coloca a tarefa de tratamento de temporizações da camada em questão na fila de prontas do Escalonador de Tarefas, passando–lhe a informação de que instância dessa camada pertence a temporização.

3.1 Módulos da Camada Sessão

Gerenciador do Protocolo de Sessão (GPS)

Este módulo é responsável pelo reconhecimento dos eventos da camada Sessão (primitivas do serviço Sessão, primitivas do serviço Transporte e temporizações) e pela chamada das funções apropriadas da máquina de estados, de acordo com o evento e o estado atual do protocolo de Sessão.

No caso do evento ser a chegada de uma SPDU, o GPS faz uma chamada ao módulo de decodificação de SPDUs que retorna uma informação que diz se a SPDU recebida é início, meio ou fim de uma SSDU. Cabe ao Gerenciador do Protocolo Sessão, o processo de remontagem dessa SSDU e, somente quando ela estiver completa, a invocação da função da máquina de estados correspondente.

Cabe ainda a esse módulo, gerenciar o estabelecimento de conexões, instanciando novas SPMs associadas às respectivas filas de eventos, a cada novo pedido de conexão iniciado pelo usuário.

Módulo de Funções da Máquina de Estados

Esse módulo contém todas as funções da máquina de estados, cujos endereços estão relacionados à determinadas intersecções evento–estado, de acordo com a Tabela de Transição de Estados.

Para permitir uma melhor estruturação do código, as funções foram divididas em três tipos: funções relativas aos eventos gerados pelo provedor

(Transporte), pelo usuário (Apresentação) ou pelo estouro de uma temporização.

Módulo de Codificação de SPDUs

Esse módulo é responsável pela montagem das SPDUs, realizando a segmentação e a concatenação básica, de acordo com a especificação do protocolo de Sessão.

Módulo de Decodificação de SPDUs

Esse módulo é responsável pela decodificação das SPDUS, observando a validade de cada uma delas, de acordo com a especificação do protocolo de Sessão.

Num estudo mais detalhado do protocolo de Sessão, observa–se que, em geral, quando uma SPDU é recebida e certas condições são satisfeitas ocorre o envio de uma primitiva de serviço Sessão. Como estas condições, na maioria das vezes, dependem de informações contidas na própria SPDU, decidiu–se realizar a montagem dessa primitiva concomitantemente com a decodificação, mesmo sem saber se ela será realmente enviada. Caso os predicados para o envio dessa primitiva, previamente montada, não sejam satisfeitos, ela é liberada e as ações especificadas no protocolo são realizadas.

Como o protocolo de Sessão permite a segmentação de SSDUs em várias SPDUs, este módulo guarda todas as informações necessárias à execução da máquina de estados no Descritor_do_Protocolo_Sessao.

Instâncias de SPMs

O estado total da máquina de protocolo de Sessão (SPM) é definido pelo estado do protocolo, assim como pelas variáveis que controlam a sua execução. Para guardar todas essas informações foi criada uma estrutura de dados chamada Descritor_do_Protocolo_Sessao, para cada SPM. Além das informações que ditam o estado total da SPM, essa estrutura contém outras que descrevem a conexão, tais como: situação da conexão (ativa ou não–ativa), unidades funcionais selecionadas, atribuição dos *tokens* disponíveis, versão do protocolo, etc.

A cada SPM estão associadas as filas FIFOs de comunicação entre as camadas e um Descritor_do_Protocolo_Sessao.

Execução da Máquina de Estados

Na recomendação X.225 do CCITT [16], o protocolo de Sessão é especificado de duas maneiras. A primeira apresenta uma descrição informal do protocolo de Sessão em forma de texto, e a segunda descreve o seu comportamento através de tabelas evento–estado.

A intersecção de qualquer evento de Sessão com um estado válido do protocolo de Sessão indica um conjunto de ações específicas e o próximo estado do protocolo.

Uma implementação completa do serviço Sessão é responsável pela coordenação das intersecções de aproximadamente 80 diferentes eventos com 32 possíveis estados do protocolo. Isso nos fornece 2560 possíveis transições da tabela de estados. No entanto, um estudo mais aprofundado revela que somente 600 das 2560 transições possíveis são consideradas válidas. Ainda, muitas das intersecções válidas resultam em mesmas ações e mesmos próximos estados.

A maneira mais comum e direta para implementar o comportamento dessas tabelas de estados é a criação de uma série massiva de sentenças *if–then–else* e/ou *switch* para cada intersecção evento–estado válida. Levando–se em conta que 600 intersecções são válidas, a complexidade do código se torna alta e apresenta uma baixa manutenção.

Com o objetivo de explorar a estrutura tabular do protocolo de Sessão do modelo OSI, a implementação das tabelas de estados baseia–se na criação de uma estrutura de matriz bidimensional, mantendo–se um direto relacionamento

entre o padrão OSI e a implementação (figura 5). O esquema adotado foi:

- valores enumerados do estado atual da máquina do protocolo de Sessão e dos eventos de Sessão são usados como índices em uma matriz bidimensional. Os elementos dessa matriz são valores inteiros que representam um índice para uma única função, responsável pelo processamento de ações específicas de uma intersecção evento–estado;
- esse índice é usado para selecionar um ponteiro específico para uma função de um vetor de ponteiros para funções. A função selecionada é, então, invocada para realizar os requisitos impostos pelo evento Sessão e o estado da máquina do protocolo de Sessão.

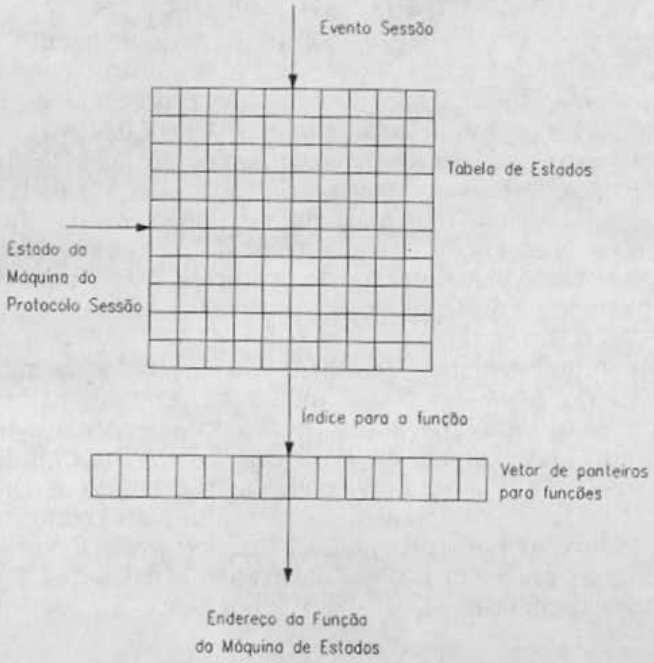

Figura 5 – Implementação da máquina de estados do protocolo de Sessão.

4 Análise do Desempenho

Com a finalidade de se efetuar uma análise quantitativa do desempenho de execução do protocolo de Sessão, foram tiradas medidas do serviço de Transferência de Dados Normais. Para o estabelecimento e liberação da conexão de Sessão os tempos obtidos foram de 11 ms e 6,8 ms, respectivamente.

Com o objetivo de identificar os pontos críticos na realização deste serviço, pelo protocolo de Sessão, foram tiradas medidas de cada uma das funções necessárias à obtenção deste serviço, tanto no processo de transmissão quanto no de recepção de dados. Estas medidas dizem respeito, apenas, aos tempos gastos pelo protocolo de Sessão, sem levar em conta o tempo gasto pelas camadas inferiores.

De acordo com a implementação realizada, a chegada de um evento do protocolo de Sessão, vindo do usuário do SS (vindo do provedor do SS), implica na execução dos seguintes passos:

– Receber evento da fila de interface;

- Reconhecer evento;
- Decodificar SPDU (esta função se aplica apenas ao processo de recepção);
- Executar a máquina de estados:
  - Avaliar predicado;
  - Codificar SPDU (esta função se aplica apenas ao processo de transmissão);
  - Montar primitiva de Transporte (Sessão);
  - Enviar primitiva de Transporte (Sessão);
- Retirar evento da fila de interface.

Para a realização destas medidas foi utilizado um Analisador Lógico Tektronix Modelo 1230, conectado à saída paralela de um microcomputador IBM–PC (NOVADATA MODELO ND 4000 AT, 8 MHz de Relógio). O Analisador Lógico tem como função monitorar periodicamente os sinais da saída paralela e, a partir de uma determinada condição (condição de disparo), armazená–los até que toda a sua memória seja preenchida. Então, através de instruções de saídas na porta paralela (*outportb* (0x378,0x##)) inseridas em pontos específicos do *software*, pôde–se medir o tempo decorrido entre eles.

Os tempos obtidos no processo de transmissão (tabela 1.a) e recepção de dados (tabelas 1.b) revelam que mais da metade do tempo de processamento é gasto pelas funções de suporte (gerenciamento de memória e manipulação das filas de interface) e o restante pelas funções do protocolo. Isto evidencia o fato de que as decisões de implementação são os fatores que mais influenciam no desempenho do sistema.

Fazendo uma análise detalhada da execução da máquina de estados (tabela 2.a e 2.b), pode–se notar que uma grande parcela do tempo de processamento é gasto na avaliação do predicado que consiste em testar a seleção das Unidades Funcionais Duplex ou Half–Duplex e a disponibilidade e atribuição do *token* de dados. Este procedimento pode ser melhorado através da definição de váriaveis booleanas, uma para transmissão e outra para recepção, que indiquem se dados podem ser enviados ou recebidos. Estas variáveis deverão ser inicializadas durante a fase de estabelecimento de conexão e atualizadas a cada mudança na atribuição do *token* de dados.

Tabela 1 – Medidas de Transferência de Dados Normais.

	Tempo	%
Reconhecimento do Evento	31	3,7%
Execução da Máquina de Estados	283	33,9%
Consulta e Manipulação das Filas de Interface	214	25,7%
Gerenciamento de Memória	307	36,7%
Total	835	100%

Tempo em microsegundos.

(a) Transmissão.

	Tempo	%
Reconhecimento e Decodificação do SPDU	288	24,3%
Execução da Máquina de Estados	205	17,2%
Consulta e Manipulação das Filas de Interface	219	18,5%
Gerenciamento de Memória	475	40%
Total	1187	100%

Tempo em microsegundos.

(b) Recepção.

Tabela 2 – Medidas da execução da máquina de estados.

	Tempo	%
Entrada	34	12,2%
Predicado	93	33%
Codificar SPDU DT	85	30%
Montar primitiva	65	22,9%
Saída	6	1,9%
Total	283	100%

Tempo em microsegundos.

(a) Transmissão.

	Tempo	%
Entrada	40	19,3%
Predicado	95	46,4%
Montar primitiva	64	31,1%
Saída	6	3,2%
Total	205	100%

Tempo em microsegundos.

(b) Recepção.

5 Conclusão

A camada Sessão é a primeira das chamadas camadas altas e, contrariamente às camadas in. riores, provê um serviço orientado ao usuário. Várias facilidades são oferecidas, na maioria opcionais, que podem ou não ser selecionadas durante a fase de estabelecimento de conexão.

Estes serviços repercutem diretamente na camada Aplicação, cabendo ao seu implementador o conhecimento e o uso correto dos serviços de Sessão. O mau uso destes serviços pode inclusive provocar *deadlocks*. Para a realização de testes foi implementado um usuário de Sessão que utiliza todos os serviços implementados.

Muitas das facilidades especificadas no serviço Sessão permanecem ainda inexploradas pelas aplicações normalizadas. As aplicações de Correio Eletrônico através da opção de contexto de Aplicação com Transferência Confiável (*Reliable Transfer Service Element* – RTSE) e de Transferência de Arquivos (*File Transfer Access and Manipulation* – FTAM) utilizam, de forma limitada, apenas um subconjunto das Unidades Funcionais de Sessão.

A implementação, realizada em linguagem C, conta com 15000 linhas de código fonte e se serve de várias funções de suporte, implementadas em módulos independentes comuns à todas as camadas, que visam um alto desempenho.

Os resultados obtidos revelam um tempo de 11 ms e 6,8 ms para o estabelecimento e liberação da conexão de Sessão, respectivamente. Na fase de transferência de dados obteve-se 1200 Kbits/s para quadros de 128 octetos e 10Mbits/s para quadros de 1024 octetos.

Bibliografia

[1] – ISO, "Basic Reference Model for Open Systems Interconnection", ISO 7498, 1983.

[2] – DAY, JOHN D. E ZIMMERMANN, HUBERT , "OSI Reference Model", *Proceedings of the IEEE*, vol. 71, no. 12 pp. 1334–1340, Dezembro de 1983.

[3] – CCITT X.200, "Reference Model of Open Systems Interconnection for CCITT Applications", *Blue Book*, vol.VIII, fasc. VIII.4, pp. 1–56, Novembro de 1988.

[4] – WOODSIDE,C. M. e MONTEALEGRE, J. R. "The Effect of Buffering Strategies on Protocol Execution Performance", *IEEE Transactions on Communications*, vol. 37, no. 6, pp. 545–553, Junho de 1989.

[5] – CLARK, D. D. e JACOBSON, V. "An Analysis of TCP Processing Overhead", *IEEE Communications Magazine*, pp. 23–29, Junho de 1989.

[6] – CCITT X.215, "Session Service Definition for Open Systems Interconnection for CCITT Applications", *Blue Book*, vol.VIII, fasc. VIII.4, pp. 303–385, Novembro de 1988.

[7] – TANEMBAUM, A. S., "Computer Networks – 2nd ed.", Prentice–Hall International, 1988.

[8] – WEST, C. H., "A Validation of the OSI Session Layer Protocol", *Computer Networks and ISDN Systems*, Vol. 11 pp. 173–182, November 1986.

[9] – "Turbo C – User's Guide", Versão 2.0, Borland International Inc.,1988.

[10] – "Turbo C – Reference Guide", Versão 2.0, Borland International Inc., 1988.

[11] – SCHATZMAYR, R., "Implementação do Protocolo de Transporte Classe 4", Projeto de Fim de Curso, DEL/UFRJ, Janeiro de 1991.

[12] – LANZA, Marcelo L. D., "Especificação Formal e Implementação da Camada Enlace", Tese de Mestrado, COPPE Elétrica/UFRJ, Novembro de 1991.

[13] – DUARTE, O. C., SCHATZMAYR, R. , BAGINSKI, L. F. e MASCARENHAS, F. "" 8 Simpósio Brasileiro de Redes de Computadores, pp. 19.4.1–19.4.5, São Paulo, Março de 1990.

[14] – BAGINSKI, L. F. e DUARTE, O. C., "Um Modelo de Implementação de Alto Desempenho para Sistemas Abertos", 9 Simpósio Brasileiro de Telecomunicações, pp. 19.4.1–19.4.5, São Paulo, Setembro de 1991.

[15] – VARGHESE, G. e LAUCK, T., "Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility", *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, ACM Operating Systems Review, pp. 25–38, Austin TX, Novembro de 1987.

[16] – CCITT X.225, "Session Protocol Specification for Open Systems Interconnection for CCITT Applications", *Blue Book*, vol.VIII, fasc. VIII.5, pp. 129–271, Novembro de 1988.