3

Tecnologias de Rede para Processamento de Alto Desempenho

Marinho Pilla Barcellos[1] *(PIPCA/UNISINOS, marinho@exatas.unisinos.br)*
Luciano Paschoal Gaspary [2] *(PIPCA/UNISINOS, paschoal@exatas.unisinos.br)*

Resumo:

Em ambientes de agregados, a comunicação entre máquinas tem historicamente representado um gargalo no desempenho. Em função disso, tecnologias de rede específicas têm sido desenvolvidas buscando aumentar a vazão e diminuir o tempo de latência. Este curso abordará as principais tecnologias de rede tipicamente utilizadas em ambientes de programação paralela e distribuída.

[1]Doutor em Ciência da Computação pela University of Newcastle upon Tyne, Inglaterra, e Mestre em Ciência da Computação junto ao Curso de Pós-graduação em Ciência da Computação da UFRGS. Professor e Pesquisador junto ao PIPCA - Programa Interdisciplinar de Pós-graduação em Computação Aplicada, UNISINOS. Áreas de Interesse: Redes de Computadores, Processamento de Alto Desempenho e Sistemas Distribuídos.

[2]Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul, e Mestre em Ciência da Computação pela mesma Universidade. Professor e Pesquisador junto ao Programa Interdisciplinar de Pós-graduação em Computação Aplicada, UNISINOS. Áreas de Interesse: Redes de Computadores, Gerência de Redes e Segurança.

3.1 Introdução

Em computação de alto desempenho, software de comunicação entre estações tem sido historicamente um "gargalo tecnológico", restringindo o desempenho global do sistema ([9]). Por isso é reconhecida na comunidade em geral a necessidade de se pesquisar novas soluções em termos de interligação de computadores, refletindo no surgimento de eventos específicos para tratar do tema (CAC - *Workshop on Communication Architecture for Clusters*). Entre outros tópicos, são investigados atualmente sistemas de comunicação de alto desempenho e suporte à programação de baixa sobrecarga (*overhead*), suporte de rede (NICs - *Network Interface Cards*, *switches* e roteadores) para comunicação ponto-a-ponto e coletiva em intra e inter-agregados, e soluções baseadas em protocolos de comunicação em nível de usuário ([47]).

O presente estudo é de caráter avançado, à medida em que pressupõe uma série de conhecimentos do leitor. Assume-se que este possua conhecimentos básicos sobre sistemas operacionais, entendendo os conceitos de processo, gerência de memória, gerência de E/S, *drivers* de dispositivo, particularmente os associados ao sistema Linux; assume-se igualmente conhecimentos básicos sobre elementos de rede, tais como estação *(host*), roteador, *switch*, *hub*, *link*, e tecnologias básicas como Ethernet. Por fim, são desejáveis noções básicas de processamento de alto desempenho, como agregados, grades e aplicações computacionalmente intensivas.

Este trabalho aborda tecnologias de rede, em suas diversas camadas, e como elas podem ser utilizadas para o processamento de alto desempenho. Por uma questão de espaço, ele não aborda ambientes/interfaces de programação, como PVM e MPI. O texto está estruturado em seis seções, tratando o tema segundo uma abordagem "top-down". A Seção 3.2 trata de **aplicações** que demandam redes de alto desempenho, motivando o estudo de tecnologias que podem oferecer isso. A Seção 3.3 revisa brevemente os dois **modelos básicos de comunicação** usados em aplicações de PAD (mensagens e memória compartilhada). Pode-se considerar que tais modelos são implementados por mecanismos que residem logo acima do **sistema operacional** e da **pilha de protocolos** que ele implementa (por uma questão de compatibilidade, quase sempre a pilha TCP/IP). Estes mecanismos, além dos aspectos de comunicação pertinentes ao sistema, são tratados na Seção 3.4. Estendendo esses conceitos, a Seção 3.5 apresenta **tecnologias de rede para interligação**, abordando soluções genéricas, para processamento de alto desempenho em redes locais (agregados) e de longa distância (grades) e para redes de armazenamento. A seguir, na Seção 3.6, são apresentadas **soluções baseadas em protocolos em nível de usuário**, que atingem significativos ganhos em termos de desempenho sacrificando segurança, generalidade e compatibilidade. O texto é concluído na Seção 3.7 com comentários finais e pontos abertos de pesquisa.

3.2 Aplicações de Alto Desempenho

Existe uma série de aplicações cujo "alto desempenho" refere-se à capacidade de rede, em termos de abundância de largura de banda e de pequena latência. Muitas delas exigem também imensa capacidade de processamento. A seguir, são comentados alguns exemplos de aplicações que demonstram a grande demanda por redes de alto desempenho.

Uma classe de aplicações que usualmente exige alto desempenho da rede é conhe-

cida como *transmissão de mídia contínua*. Nestas aplicações, um servidor envia continuamente um ou mais fluxos de aúdio, vídeo, etc. codificados para um ou mais receptores "ouvintes". Por exemplo, em novembro de 2001, um show de música teve amplificação remota: em um ambiente, o som de um músico tocando um instrumento foi captado, codificado e enviado para que fosse tocado em um anfiteatro a muitos quilômetros dali; neste anfiteatro, o som reproduzido era captado, codificado e enviado de volta ao ambiente de origem, para reprodução "amplificada" ([25]).

A *colaboração em tempo real* também representa uma classe importante de aplicações que requerem infraestrutura de comunicação de alto desempenho. Em [25] são descritos dois exemplos desse tipo de aplicação, que foram reproduzidos em um *testbed* envolvendo diversas empresas e universidades. A primeira explorou a colaboração entre artistas, músicos, cinegrafistas e coreógrafos em uma apresentação onde os dançarinos encontravam-se em Denver, Mineápolis e Flórida, e foram acompanhados por músicos no Brasil. Para realizar essa apresentação foram utilizados vários nós com suporte à conferência e um conjunto de fluxos de áudio e vídeo de alta qualidade. A segunda aplicação compreendeu a manipulação remota de um microscópio localizado em São Diego, operado a partir da visualização ao vivo de um vídeo.

Existem aplicativos que necessitam acessar enormes quantidades de dados. Exemplo disso são as aplicações de *visualização remota*. Visapult ([43]) é um protótipo desenvolvido pelo *Lawrence Berkeley National Laboratory* para a visualização remota e distribuída de dados científicos da ordem de Terabytes a uma taxa sustentada de 1 Gbps. Para tal, são utilizadas *caches* de dados, infraestrutura de rede de alta velocidade e mecanismos de "renderização" paralela.

Aplicações de *mineração de dados* também manipulam grandes volumes de dados. TWDM - *Terra Wide Data Mining Testebed* ([31]) é uma infraestrutura criada para análise remota, mineração e interação em tempo real de dados complexos (científicos, engenharia, negócios, etc.). Aplicações de mineração, nesse *testbed*, são projetadas para explorar as capacidades oferecidas por tecnologias de rede locais e de longa distância de modo que conjuntos de dados da ordem de Gigabytes e Terabytes possam ser explorados remotamente em tempo real.

Certas aplicações exigem tanto capacidade de rede quanto de processamento. Um exemplo de aplicação é o Cactus ([2]), que visa a *modelagem* e *simulação* de fenômenos astronômicos complexos. Na realidade Cactus é um *toolkit* de componentes de software, incluindo mecanismos para a realização de entrada e saída em paralelo, visualização, além de ferramentas computacionais para investigar campos gravitacionais dinâmicos.

Um nova abordagem para atender aos requisitos por infraestrutura de alto desempenho imposta pelas aplicações científicas de larga escala são as *grades computacionais*. Os sistemas para computação em grade procuram simplificar o acesso a e o uso coordenado de recursos como supercomputadores, sistemas de armazenamento que operam na escala de Terabytes e outros dispositivos de experimentação. É importante salientar que esses recursos, em geral, encontram-se espalhados geograficamente. As grades têm sido usadas para executar um conjunto variado de aplicações como as relacionadas à supercomputação distribuída e teleimersão.

Por fim, uma "aplicação" clássica que demanda redes de alto desempenho são os agregados de computadores, que almejam latência zero e grande quantidade de banda passante na comunicação. O tópico comunicação em agregados é enfatizado ao longo do texto.

3.3 Mecanismos de Comunicação

Esta seção discute brevemente os principais modelos de comunicação entre processos: troca de mensagens, chamada remota de procedimento, invocação remota de método e memória compartilhada distribuída.

3.3.1 Troca de Mensagens

Abordagem básica, a troca de mensagens pode ser comparada ao envio de uma área binária de memória para um destinatário remoto. A comunicação se dá através de um par de primitivas e variantes: send() para enviar e receive() para receber. Existem variações dessas primitivas, dependendo do esquema de endereçamento usado, da existência ou não de conexão, etc. Mais precisamente, as principais variações em mecanismos de troca de mensagens se referem à existência ou não de conexão, à forma com que as mensagens são endereçadas, ao modelo (1-N ou $N - N$), à existência de controle de fluxo, às garantias de confiabilidade mediante falhas, às garantias de ordenamento em relação à entrega, e ao sincronismo entre remetente e receptor (em que momento após enviar o controle retorna ao remetente).

A primitiva send() recebe como argumentos, usualmente, o endereço destino ou identificador de conexão, e o endereço de um *buffer* contendo a mensagem a ser enviada. Outro argumento pode ser o tipo de mensagem, conforme definido pela aplicação; se mensagens são "tipadas", com suporte do sistema pode ser possível a uma aplicação selecionar um determinado tipo de mensagem a ser recebida. A mensagem, contida na área informada, corresponde a uma estrutura de dados contendo informações de controle e dados de aplicação. O processo destinatário deve conhecer os tipos de mensagens e entender o formato de cada uma; o destinatário tipicamente realiza a consistência sobre os valores recebidos.

Em termos de implementação, o modelo de mensagens pode se manifestar em diferentes camadas, como p.ex. uma aplicação que usa *Sockets* ou MOM - *Message-Oriented Middleware*. A API - *Application Programming Interface* ou interface de programação de *Sockets* é o exemplo mais popular do modelo de troca de mensagens. *Sockets* permitem diferentes famílias de protocolos, de endereçamento e diferentes tipos de serviço. O modelo de mensagens é melhor expresso pelo protocolo UDP, que implementa a transmissão de datagramas (de até 64 Kbytes). Outro protocolo sempre presente é o TCP, que oferece a transmissão confiável de uma seqüência de bytes (uma *stream*) e controle de fluxo. Um *socket* é um descritor que representa um *endpoint* e está vinculado a um endereço IP (usualmente o endereço associado à única NIC na estação) e uma porta de comunicação (um valor inteiro). A API de *Sockets* será discutida em maior detalhe na Seção 3.4, no âmbito da implementação da pilha TCP/IP. Em contraste, MOM é um esquema de *middleware* onde processos criam caixas postais para recebimento de mensagens. Uma das principais características (ou vantagens) de MOM é que transmissor e receptor não precisam se comunicar sincronamente: quando o rementente envia, o receptor pode estar desligado, *off-line*; similarmente, quando o receptor recebe, o remetente não precisa estar *on-line*.

O modelo de troca de mensagens é extensivamente utilizado em processamento de alto de desempenho. Os dois exemplos mais contundentes são PVM - *Parallel Virtual Machine* ([38]) e MPI - *Message Passing Interface* ([29]). PVM é um ambiente cujo ob-

jetivo é oferecer uma máquina virtual que congrega os recursos de processamento de nós de um agregado. MPI é a especificação de uma interface padrão para troca de mensagens; existem tanto implementações livres como proprietárias de MPI. Em relação a *Sockets*, PVM e MPI acrescentam esquemas de endereçamento mais adequados a um agregado do que *endpoints* TCP/IP, além de primitivas de sincronização entre processos e outras facilidades.

Mensagens Ativas ([9, 13]) é um protocolo simples, peso-leve, flexível, que pode ser usado para implementar abstrações de passagem de mensagens de mais alto nível, como MPI. Uma mensagem ativa contém o **endereço** de uma rotina de tratamento a ser executada quando a mesma chegar no receptor. A rotina de tratamento de mensagem deve ser pequena e simples. Mensagens podem ser do tipo *request* ou de *reply;* a diferença entre elas é que apenas mensagens de *request* podem originar uma nova mensagem. O mecanismo é em torno de 10 vezes mais rápido que mecanismos normais de comunicação, do tipo *Sockets*.

3.3.2 Chamada Remota de Procedimento

Apesar de representar hoje um padrão para o desenvolvimento de aplicações de rede ou distribuídas, o modelo de troca de mensagens é considerado por muitos como um paradigma de baixo nível de abstração. Com troca de mensagens, é preciso lidar com detalhes de endereçamento, formato de mensagens e conversões de valores quando a comunicação envolve arquiteturas diferentes. O modelo de comunicação baseado em chamada remota de procedimento, ou RPC ([7]), resolve parte das limitações do modelo de mensagens ao estender o modelo tradicional de chamadas de procedimento, locais, para operação em ambientes distribuídos. Um processo de aplicação pode solicitar a execução de um procedimento localizado em um processo (servidor) em outra máquina, desde que este tenha sido devidamente "registrado". Com RPC, há o desvio do fluxo de controle da máquina do cliente para a do servidor, e de volta; tipicamente, o cliente fica bloqueado esperando a resposta do servidor.

RPC é popular porque oferece a desenvolvedores um *framework* bem conhecido, a chamada de procedimento. Serviços remotos são virtualizados através de procedimentos. Este modelo, onde o cliente espera sincronamente pelo servidor, é bem próximo do esquema seqüencial que programadores conhecem. Programação paralela e distribuída pode ser obtida através da execução assíncrona de chamadas RPC (controle retorna imediatamente ao cliente, enquanto servidor continua a executar a chamada).

Estruturas de dados passadas como parâmetro dos procedimentos remotos devem ser descritas através de uma linguagem de representação de dados com sintaxe similar a C, denominada XDR - *eXternal Data Representation* ([48]). Esta descrição é tomada como parâmetro de uma ferramenta (rpcgen), que gera código *stub* para o cliente e para o servidor. O código *stub* é compilado e ligado junto ao código desenvolvido para cliente e servidor; ele é responsável por esconder do cliente a existência do servidor, e do servidor, a do cliente. Para tal, faz a preparação dos dados, serializando os mesmos e colocando o resultado em pacotes.

Embora RPC seja um modelo de comunicação bastante popular (através da implementação Sun RPC), apresenta uma série de limitações, e não consegue oferecer completa transparência. Sua semântica é diferente da chamada local por uma série de razões, tal como a ocorrência de falhas na rede ou no servidor, a exigência de uma fase de amarração *(binding*) e a impossibilidade de passar apropriadamente parâmetros por

referência (estruturas encadeadas). As abstração oferecidas pelo mecanismo possuem em contrapartida um custo em desempenho, muitas vezes mais lento que *Sockets*. Existem otimizações visando aumentar o desempenho de RPC, como [6], e recentemente o modelo foi retomado como alternativa de comunicação em PAD, visando especificamente o domínio de aplicações em grade; um exemplo é o GridRPC ([42]).

3.3.3 Invocação Remota de Método

Enquanto RPC segue o paradigma imperativo, RMI se aplica ao orientado a objetos. Em um ambiente onde os objetos se encontram distribuídos, um dado objeto pode "invocar" a execução de um método sobre um objeto remoto em relação a ele. Como argumento passado ao método, é possível fornecer valores por cópia ou referências para objetos que podem ser remotamente acessados (ditos "objetos remotos"). A implementação mais popular de invocação remota de método é o Sun Java RMI - *Remote Method Invocation* ([23]).

Existem diversos projetos investigando o uso de sistemas de objetos distribuídos em Java e com RMI de paradigma de comunicação para aplicações de PAD. O tópico é extenso e foge do escopo deste trabalho; o leitor interessado pode recorrer, por exemplo, a [26] e [5].

3.3.4 Memória Compartilhada Distribuída

Os esquemas de comunicação anteriores se baseiam ou na cópia de áreas de memória (troca de mensagens), ou na passagem de argumentos e desvio do fluxo de execução (ambos RPC e RMI). Um esquema de (mais) alto nível é a memória compartilhada distribuída: o modelo de compartilhamento de dados em memória física freqüentemente encontrado em multiprocessadores, dito "fortemente acoplado", é estendido para multicomputadores de forma que processos de aplicação tenham a ilusão de uma memória física global.

Existem diversos tipos de sistema de memória compartilhada distribuída. Uma possibilidade é processos em diferentes máquinas compartilharem um segmento de memória, cada um mapeando tal área em seu espaço de endereçamento virtual. Modificações na memória compartilhada refletirão nas áreas mapeadas nos demais processos, mais cedo ou mais tarde. Por uma questão de desempenho, foram propostas várias semânticas diferenciadas, "relaxadas" em relação à semântica oferecida por uma memória física compartilhada; elas variam em relação às garantias de consistência nos dados, ou quando modificações são garantidamente vistas por outros processos e se conflitos podem surgir, como eles são tratados.

Um exemplo é o Split-C ([20]), que é extensão paralela SPMD simples à linguagem C. Split-C, de paralelismo e dados explícitos, oferece memória global, com *arrays* distribuídos e apontadores globais.

3.4 Pilha de Protocolos e Sistema Operacional

Os modelos apresentados na seção anterior são utilizados na comunicação entre os múltiplos processos distribuídos que compõem uma aplicação, e que residem em

máquinas diferentes. Em ambos os casos, haverá transmissão de dados via rede e a mesma será governada por uma pilha de protocolos, parte dos sistemas operacionais nas estações fim. Nas décadas passadas, as tecnologias de ligação em rede evoluíram muito, incrementando larguras de banda de Kbits para dezenas de Gigabits por segundo e barateando o hardware de rede necessário. Este fenômeno levou à popularização de redes de computadores com "alta capacidade", similarmente ao ocorrido com microprocessadores. No entanto, o crescimento da capacidade de rede suplantou àquele visto com microprocessadores, e colocou novas demandas de desempenho sobre os sistemas operacionais e a implementação de protocolos.

Esta seção revisa rapidamente a pilha de protocolos TCP/IP através da descrição de uma implementação típica, e discute os principais aspectos de desempenho relacionados ao sistema operacional.

3.4.1 Implementação de TCP/IP em 3 Camadas

A forma tradicional de se tratar com a complexidade inerente de redes de computadores é organizar os protocolos em camadas, tal como no modelo OSI. Entretanto, quanto maior o número de camadas, menos eficiente será a comunicação, particularmente devido a cópias de memória e decisões inapropriadas sobre *bufferização*. A pilha TCP/IP, o padrão da Internet, é pragmaticamente dividida em 4 camadas, conforme ilustrado na Figura 3.1.

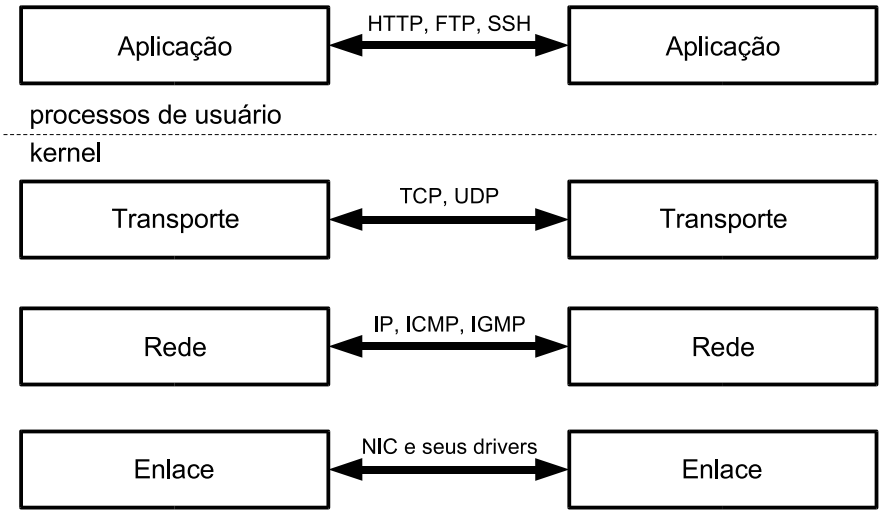

Figura 3.1: Pilha de protocolos TCP/IP.

Como parte do sistema operacional, a pilha de protocolos deve oferecer aos processos de usuário uma interface de acesso ao hardware de rede uniforme e de alto nível. Deve também garantir a proteção do sistema, evitando que um processo envie ou receba pacotes arbitrariamente na rede, ou acesse um recurso de forma injusta com os demais processos. No caso da pilha TCP/IP, estas garantias se encontram disponíveis nas diversas implementações existentes, variando-se minimamente com a família de sistema operacional considerada. Muitas das implementações atuais de TCP/IP foram derivadas

da versão BSD, criada originalmente para o Unix BSD, incluindo a do Linux. A mesma será tomada como base para descrição da implementação da pilha, a seguir.

Conforme ilustrado na Figura 3.2, o código de rede da pilha está organizado em três camadas: *sockets*, *protocolos* e *interface*. A camada de *sockets* é utilizada para oferecer uma API de programação, facilitando o desenvolvimento de aplicações de rede e novos protocolos em nível de usuário. Originalmente, a API de *sockets* foi projetada para ser flexível, possibilitando a utilização de diferentes protocolos de transporte e arquiteturas de rede; no entanto, com a predominância de TCP/IP, a imensa maioria das aplicações utiliza *sockets* com a família IP de protocolos e tendo TCP ou UDP como protocolos de transporte, sendo a escolha entre estes últimos guiada pelo tipo de serviço desejado (*stream* confiável ou datagramas não confiável). Para iniciar uma comunicação, o processo de aplicação cria um *socket*, o que significa alocar um descritor (similarmente a arquivos), e então pode conectar esse *socket* a um *endpoint* remoto (necessário apenas no caso de TCP). Dados podem ser enviados através deste descritor utilizando-se chamadas de sistema tradicionais para manipulação de arquivos, como write(), ou uma das chamadas específicas para isso, como sendto(). Essas chamadas tomam como argumento, entre outros, o endereço do *buffer* que contém os dados, residente no espaço do processo. Outro possível argumento é o endereço destino: no caso de UDP, cada datagrama enviado pelo *socket* poderá ter um destino diferente, informando-se o endereço IP no seu cabeçalho; já no caso de TCP, o endereço remoto é fornecido apenas uma vez, quando do estabelecimento da conexão, posto que todos os dados fluem através da mesma.

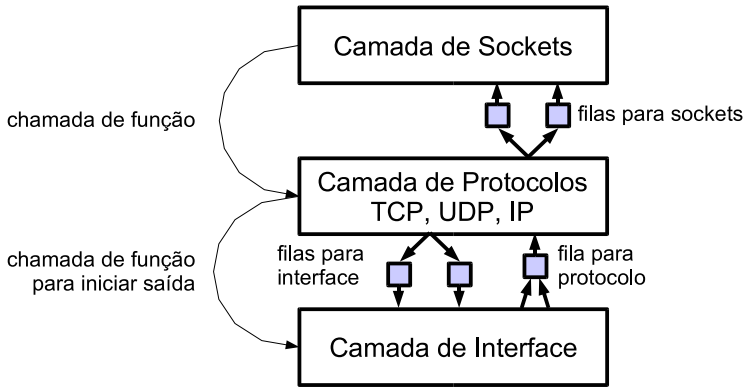

Figura 3.2: Arquitetura da implementação BSD de TCP/IP.

Quando uma chamada write() ou similar é executada sobre um *socket*, a camada de *sockets* chama uma função de transmissão específica ao protocolo, que copia dados do espaço do usuário para estruturas de dados utilizadas pelo código de rede para armazenar pacotes, denominadas *mbufs*.

A camada de protocolos é responsável por acrescentar cabeçalhos específicos do protocolo na frente (antes) dos dados do usuário e inserir o pacote resultante na fila de transmissão da interface correspondente. Após a inserção, a camada sinaliza ao *driver* de dispositivo (ou apenas *dispositivo*) que ele deve iniciar o envio do pacote ou pacotes na sua fila de transmissão, que segue de forma assíncrona, liberando a CPU. Quando a operação de envio do pacote termina, a NIC gera uma interrupção para avisar o sistema que os recursos atualmente ocupados (particularmente, *buffers*) podem ser liberados. Portanto, no caso do envio, os dois principais impedimentos em termos de desempenho são a cópia

de memória e a interrupção gerada.

Na chegada de um pacote, primeiramente o quadro correspondente é armazenado nos *buffers* internos da NIC. Quando o último byte do quadro chega, o quadro é processado e o pacote resultante pode ser passado ao nível de cima. Para tal, o dispositivo gera uma interrupção que é processada pelo tratador de interrupções associado ao mesmo no sistema operacional. No tratamento, uma rotina dependente do meio copia o pacote dos *buffers* da NIC para uma área de memória dentro do *kernel*, insere o pacote em uma fila de protocolo e gera um sinal (interrupção de software) para que a fila seja processada. No caso de pacotes IP, a rotina de entrada processa todos os pacotes na fila, verificando a validade do *checksum*, tratando *flags*, e averiguando se o endereço destino é o mesmo dessa máquina ou se a mesma está configurada para realizar o encaminhamento de pacotes. Por fim, determina o *socket* para o qual o pacote é destinado e "coloca" a carga de dados do pacote no *buffer* de recebimento associado ao *socket*. A aplicação utiliza chamadas do estilo read(), recv() ou recvfrom() para obter os dados que se encontram neste *buffer*; estas chamadas tomam como um dos parâmetros a posição de memória para onde os dados devem ser copiados.

3.4.2 Fontes de Sobrecarga

O desempenho da rede é afetado por diversos fatores. O custo de processamento ao se transferir pacotes pode ser dividido em *custos por pacote* e *custos por byte*. No primeiro caso, as duas principais fontes são o tratamento de interrupções e o processamento realizado pelo protocolo. Uma forma simples de se reduzir o impacto do custo por pacote é a utilização de pacotes maiores; entretanto, é usualmente inadequada devido a limitações impostas pelas principais tecnologias de rede quanto ao tamanho de quadro utilizado. Os custos por byte estão geralmente associados à necessidade de acessar na memória os dados do pacote, seja para copiá-los, seja para calcular um *checksum* a ser incluído no cabeçalho. Os custos por pacote e por byte acima podem ser reduzidos através de técnicas de otimização desenvolvidas e implementadas em protocolos ao longo dos anos. A seguir, são comentados as principais fontes de sobrecarga e, após, como elas tem sido abordadas.

Tratamento de interrupções. Tradicionalmente, tanto o envio como a recepção de pacotes causa uma interrupção. Em uma transferência, o número de pacotes trocados dependerá da largura de banda implementada e do tamanho de pacote utilizado; como o tamanho de pacote é limitado em função das tecnologias de enlace subjacentes, para se obter uma largura de banda da ordem de Gigabits será necessário tratar uma interrupção a cada 10 micro-segundos nas estações fim, caso o tamanho do quadro esteja limitado a 1500 bytes[3]. O tratamento de interrupções incorre em substancial sobrecarga em função das trocas de contexto necessárias, do processo atual para o tratador de interrupções, e então para o mesmo processo ou outro; ao realizar uma troca de contexto, uma cópia completa dos registradores (de CPU, memória, etc.) é salva em um descritor de processo (PCB) na memória, e novos valores carregados de outro PCB. Além disso, uma troca de contexto implica freqüentemente limpar a *cache* de tradução de endereços virtuais em físicos (chamada TLB, de *Translation Look-a-side Buffer*) existente em sistemas paginados, afetando sensivelmente o desempenho. O excesso de interrupções pode levar uma estação a uma situação conhecida como *livelock*: a taxa de interrupções é tão alta que

[3] Valores aproximados.

praticamente não sobra tempo para efetuar qualquer processamento útil. Este é o caso particularmente de um receptor que é "atropelado" pela chegada de uma rajada de pacotes, que é tão rápida que não sobra tempo de CPU para a aplicação processar os dados recebidos, levando ao enchimento dos *buffers* e ao descarte de pacotes *(receive livelock*, [28]).

Processamento do protocolo. A sobrecarga gerada dependerá do protocolo em questão, das garantias que ele oferece, de sua complexidade e também das premissas assumidas sobre a camada subjacente (tipos de erros, etc). Por exemplo, um protocolo que não garante ordenamento nem detecção e correção de erros, como UDP, é muito mais "leve" do que um protocolo que o faz, como TCP.

Cópias (acesso à memória). Salvo raras exceções, todos os computadores modernos seguem a arquitetura de Von Neuman. Por que a velocidade de processadores proporcionalmente cresceu muito mais do que a velocidade da memória principal, nessa arquitetura a memória hoje representa um gargalo. Em protocolos de comunicação, o problema se agrava pois cópias de áreas de memória são necessárias. Elas são, entretanto, necessárias à medida em que exige-se uma separação clara entre as camadas de protocolos que compõem a pilha e a aplicação. Tipicamente, nas implementações atuais de TCP/IP dados são copiados pelo menos duas vezes, tanto no envio como na recepção, para transferir bytes da aplicação para o *kernel*, e então do *kernel* para a NIC; na recepção, o caminho inverso é feito.

Cálculo de *checksum*. Como a maioria dos protocolos, TCP/IP se baseia no uso de *checksums* para preservar a integridade dos cabeçalhos e dos dados transferidos. Antes de enviar um segmento ou datagrama, TCP ou UDP calculam (com XOR) um valor de 16 bits em função dos bytes de dados sendo enviados, e incorporam ao cabeçalho em questão. Na chegada, o receptor processa um novo *checksum* sobre o conteúdo recebido e confere com o original (na realidade, como o *checksum* faz parte do cabeçalho, o resultado deve ser 0). O *checksum* permite que a maior parte[4] dos embaralhamentos de bits seja detectada. O protocolo IP emprega *checksum* para proteger apenas seu cabeçalho, enquanto TCP e UDP usam para proteger ambos dados e cabeçalho. O cálculo de *checksum* exige o acesso sequencial a cada um dos bytes, e por isso representa substancial sobrecarga.

3.4.3 Otimizações

Transferência entre placa de rede e memória principal via DMA. As operações de cópia de memória podem ser realizadas de diferentes maneiras pelo hardware; particularmente, uma transferência entre os *buffers* da NIC e a memória principal pode ser efetuada através de instruções de E/S, mapeamento de memória (*buffer* da NIC mapeado em uma região da memória principal) ou cópia em bloco através de DMA, ou *Direct Memory Access*. Esta última opção é amplamente utilizada, por permitir que uma região de memória seja copiada sem ocupar a CPU em uma cópia byte a byte, bastando programar a cópia via endereço origem, endereço destino e tamanho do bloco. Uma variação dessa técnica, chamada *scatter-gather DMA*, é aplicada na maioria das NICs novas hoje no mercado. Ela permite que uma lista de blocos seja fornecida quando da programação da cópia, de forma que cabeçalhos possam ser facilmente acrescentados quando um pacote passa de uma camada para a subjacente, seja da aplicação para o *kernel* ou do *kernel* para

[4]conforme demonstrado em [34], com um *checksum* de 16 bits, 1 em cada 300 milhões de pacotes é entregue corrompido, ou seja, sem que o protocolo TCP tenha detectado o problema.

a NIC.

Otimização do cálculo de *checksum*. A velocidade de cálculo do *checksum* é limitada realmente pela memória. Uma técnica proposta por Clark (RFC817) e implementada no TCP por Clark and Jacobson ([10]) é denominada *copy-and-checksum*: reorganizar as operações do protocolo de forma a calcular o *checksum* junto ao processo de cópia, evitando acessos adicionais à memória. Outra técnica utilizada é passar o cálculo do *checksum* para o hardware da NIC; segundo [35], a maioria das NICs novas hoje já permite isso (embora de maneira não robusta), porém nem todos os sistemas operacionais fazem uso. Por fim, o cálculo de *checksum* na NIC não permite a detecção de corrupção de dados durante a cópia para os *buffers* da NIC, e vice-versa.

Otimizações no processamento do protocolo. Ainda segundo [10], para se otimizar protocolos deve-se atacar o caso "mais comum" de processamento. Considerando as tecnologias de rede atuais, quando um receptor é acionado pela chegada de um pacote, a tendência é que pacotes cheguem sem erros e em ordem. No caso específico do TCP, existe uma técnica conhecida como previsão de cabeçalho (*header prediction*), que separa o processamento do protocolo em um caminho rápido (*fast path*) e um caminho lento (*slow path*). Na chegada de segmentos, o protocolo procura por aqueles que obedeçam uma série de condições (como p.ex., relativos a conexões já estabelecidas, que sejam o próximo segmento na ordem esperada, e não sejam relativos a retransmissões), caracterizando um "segmento padrão". No caso de satisfazer as condições, o segmento é processado eficientemente com apenas algumas poucas instruções, que removem o cabeçalho e passam os dados ao nível de cima. Caso contrário, o segmento deve ser tratado pelo código tradicional, lento, mas que lida com uma gama de situações (boa parte bastante incomum) conforme especificado na RFC793.

Aumento no tamanho dos quadros. Esta técnica permite diminuir o número total de pacotes, e portanto reduzir a carga de processamento imposta sobre a CPU (com tratamento de interrupções e processamento do protocolo por pacote). Entretanto, o tamanho máximo de quadro é uma propriedade de cada tecnologia de enlace, e assim pode não ser possível aumentar o tamanho do quadro arbitrariamente. A família Ethernet como um todo limita o tamanho da carga do quadro a 1500 bytes, enquanto Myrinet oferece quadros de até 4 Gigabytes (vide Seção 3.5). Esta solução se aplica melhor a redes dedicadas, como por exemplo *Storage Area Networks*.

Fragmentação assistida por hardware. Uma alternativa ao uso de quadros grandes é implementar na NIC funcionalidade para fragmentação e defragmentação de pacotes. Com isso, o sistema operacional pode operar com pacotes bem maiores do que o limite imposto pela tecnologia de enlace subjacente, enquanto o hardware se encarrega de, transparentemente, fragmentar pacotes e transmiti-los, mantendo a compatibilidade com o padrão da rede. No receptor, a NIC com suporte à fragmentação, quando presente, se encarrega de receber os quadros e juntá-los, entregando ao sistema operacional menos quadros, porém maiores. Tal esquema já foi implementado em protótipo para uma placa Gigabit Ethernet no Linux, e os resultados demonstraram que as mesmas velocidades foram atingidas, mas com uma sensível redução na ocupação da CPU. Suporte à fragmentação em hardware também foi implementado para divisão de segmentos TCP. A principal dificuldade com esse tipo de solução é lidar com a complexidade desses protocolos e tratar adequadamente, em hardware, todas as situações que podem surgir.

Combinação de interrupções. é um conjunto de técnicas que visa reduzir o número total de interrupções geradas; a nomenclatura varia de acordo com o autor, como por exemplo *interrupt hold-off* ([36]) ou *interrupt mitigation* ([24]). Em geral, a NIC

gera uma interrupção na recepção de um pacote para indicar ao sistema que um pacote deve ser lido das filas. O número de interrupções pode ser reduzido fazendo com que a NIC atrase levemente a interrupção quando da chegada de um pacote, na expectativa de que outro(s) cheguem entrementes, e a notificação seja relativa a vários pacotes. Para tal, assume-se que após uma interrupção o sistema verifique a disponibilidade de não apenas um, mas de vários pacotes na fila e os trate em seqüência. Em [24], descreve-se uma implementação da técnica *(timer mode driver)* em sistemas Linux para uma NIC baseada em um *chip* Tulip. As modificações são isoladas no *driver* de rede. A implementação explora o fato que o *chip* Tulip permite gerar continuamente interrupções em intervalos múltiplos de 81,92 μs; com isso, interrupções de recepção (Rx) são desabilitadas, e quadros recebidos são tratados periodicamente (quando da interrupção do temporizador da NIC). O lado negativo da abordagem de combinação de interrupções é que ela em geral aumenta a latência, particularmente em situações de carga baixa. Além disso, a NIC não faz distinção entre diferentes fluxos de aplicação, de forma que uma aplicação interativa, com pouco tráfego, sofrerá os mesmos atrasos que uma aplicação de transferência de dados. Uma outra solução é o sistema monitorar a chegada de pacotes e, caso a carga exceda um limite, passar a operar em modo de *polling* - o *driver* verifica com a NIC (p.ex., consultando seus registradores) sobre a chegada de novos pacotes, assim evitando interrupções. A junção de interrupções exige suporte de hardware e é particularmente útil quando boa parte dos pacotes trocados são pequenos.

Implementação de TCP em Hardware. Uma das técnicas para aumentar o desempenho é realizar o cálculo do *checksum* do IP através da NIC. Uma extensão, bem mais ambiciosa, consiste em implementar em hardware (ou mais provavelmente, *firmware*, dada a complexidade e o custo da tarefa) uma parte da pilha TCP/IP, ou sua totalidade. No primeiro caso, apenas o *data path* é passado para a NIC, mantendo a parte de estabelecimento e término de conexão com o sistema operacional. Esta alternativa segue a filosofia básica proposta por Van Jacobson, mencionada anteriormente, de se otimizar o caso típico. A segunda opção corresponde a uma implementação completa da pilha na NIC, cujos benefícios serão maiores quando a freqüência de novas conexões for bastante alta. A técnica conhecida como TOE, ou *TCP Offload Engine*, está sendo desenvolvida para o mercado de *Storage Area Networks*, buscando criar uma placa de rede de 10 Gigabits que propicie alta vazão e baixa latência ao mesmo tempo que sobrecarregue pouco a CPU ([49]).

3.4.4 Zero-copy

Como o próprio nome já indica, em implementações "zero-copy" não há cópias de áreas de memória. Em termos gerais, o esquema funciona da seguinte maneira: no lado transmissor, os dados a serem enviados são acessados pela NIC diretamente da área do processo usuário; no lado receptor, os dados que chegam são copiados para um *buffer* de memória que é posteriormente re-mapeado para fazer parte da aplicação.

Um impedimento para implementação de TCP ou UDP sem cópias é a semântica das chamadas de envio de dados, como write(): o controle retorna ao usuário logo após a cópia dos dados para o *kernel*, quando a aplicação está autorizada a reusar o *buffer*. No caso do TCP, em particular, uma cópia dos dados precisa ser mantida intacta pelo *kernel* para que os mesmos sejam re-enviados caso uma perda ocorra.

Uma técnica visando garantir a imutabilidade dos dados consiste em marcar as páginas contendo os dados como COW (*copy-on-write*), o que faz com que a página seja

duplicada caso um acesso de escrita seja realizado. Caso a aplicação tente escrever nos *buffers* após o write() (e antes do *ack*), o hardware MMU - *Memory Management Unit* detecta o acesso, causando uma interrupção a ser tratada pelo sistema operacional, que por sua vez faz a cópia da página e transparentemente mapeia os *buffers* residentes na página para a área do usuário. Quando o *buffer* pode ser liberado, então a página é liberada. Esta técnica só é efetiva quando os dados não são modificados antes da chegada do *ack* do TCP; caso contrário, o desempenho resultante pode ser pior do que sem essa otimização.

Uma outra alternativa é a inclusão de uma nova primitiva, denominada sendfile(), que toma como parâmetros um descritor de arquivo e um de *sockets*. Sendfile() envia um arquivo inteiro; como o *kernel* é inteiramente responsável pela transmissão, é possível garantir que os dados não serão modificados (a não ser que o arquivo seja modificado concorrentemente por outro processo). Esta primitiva pode representar substancial ganho de desempenho em aplicações onde a parte principal seja a transferência de arquivos, tal como um servidor *web*, ao evitar a cópia de dados entre espaço de usuário e *kernel*. A versão disponível no Linux implementa transferência com *zero-copy*; sua definição indica também que qualquer um dos dois descritores tomados como argumento pode ser um *socket*, embora a implementação atual (*kernel* 2.4.19) aceite um *socket* apenas no descritor destino. Não existe uma padronização referente a essa primitiva, variando o nome e a semântica exata entre sistemas operacionais.

Evitar a cópia de dados no receptor é bem mais difícil do que no remetente, pois não existe estado de protocolo suficiente para prever no receptor, antes de chegada dos dados, onde exatamente os mesmos devem ser colocados. Caso o tamanho dos dados recebidos seja múltiplo do tamanho da página de memória, o sistema pode usar a MMU para mapear o *buffer* diretamente no espaço de endereçamento virtual da aplicação. Esta técnica, denominada *page flipping*, requer o alinhamento da carga do pacote e do *buffer* de usuário com a separação entre páginas; ou seja, os cabeçalhos devem ficar antes do início da página.

Uma técnica que permite evitar essa cópia adicional no receptor se baseia no acréscimo de estado no protocolo para isso. Antes de enviar dados, o remetente deve reservar uma área de memória para onde os dados serão copiados ao chegarem. O protocolo STP - *Scheduled Transfer Protocol* ([3, 4]), por exemplo, se baseia nesse estratagema; o mesmo se aplica aos protocolos utilizados no Projeto Avalanche ([11]).

3.4.5 Sobrepasso (*bypass*) do Sistema Operacional

Apesar das otimizações nas seções anteriores, ainda o sistema operacional naturalmente restringe o desempenho e é fonte de sobrecarga. Em função disso, uma abordagem na busca por eficiência consiste em sobrepassar o sistema operacional, permitindo que um processo de aplicação se comunique diretamente com a NIC.

Uma opção é permitir que a aplicação escreva diretamente nos registradores da NIC, embora não seja interessante por ser dependente de hardware e não permitir que mais de uma aplicação use a rede. Uma forma mais adequada é permitir acesso **controlado** de processos de usuário à NIC, que deve oferecer certo suporte: a NIC implementa múltiplas filas, que são alocadas às diferentes aplicações, e é responsável por reconhecer os pacotes na chegada e colocá-los na fila correta.

A NIC CLAN ([39]) é uma interface Gigabit desenvolvida pela AT&T segundo o modelo de comunicação baseado em memória compartilhada distribuída sem coerência, onde uma porção da região de memória virtual da aplicação é mapeada na memória

física de uma estação remota. A NIC implementa um esquema de RDMA, ou *Remote* DMA, que permite a transferência de blocos da memória local para uma estação remota sem a intervenção direta da CPU. Para diminuir o número de chamadas do sistema, a comunicação entre a aplicação e o *driver* de rede local ocorre através de uma *fila assíncrona*, estrutura mantida em uma região de memória compartilhada entre aplicação e *driver*.

A desvantagem associada a abordagem de sobrepasso do sistema é tornar as aplicações mais complexas, pois as mesmas precisam lidar com detalhes da rede usualmente ocultos pelo sistema, como colocar dados em pacotes, gerenciar descritores de transmissão e recepção e esperar por novos dados. Para simplificar a tarefa, comprometendo-se minimamente o desempenho, pode-se usar uma biblioteca do estilo MPI ou PVM sobre um sistema operacional que permita o sobrepasso.

Muitos são os sistemas de comunicação que exploram a abordagem de sobrepasso do sistema operacional. Exemplos tratados neste trabalho incluem Myrinet (Seção 3.5.3), InfiniBand (3.5.7), U-Net e VIA (ambas na Seção 3.6).

3.5 Tecnologias de Enlace e Interligação

A seção anterior apresentou protocolos e técnicas de otimização no sistema operacional (com o auxílio do *hardware* de rede) para aumentar o desempenho na comunicação. Esta seção trata de tecnologias de interligação entre computadores, abordando tanto aquelas que são de propósito geral como as que são dedicadas ao processamento de alto desempenho. A discussão coloca ênfase nos níveis físicos e de enlace.

Porém, antes de tratar da comunicação inter-máquinas, faz-se necessário introduzir diferentes ambientes de interligação e apresentar alguns conceitos relacionados, incluindo componentes básicos das tecnologias de interligação, métricas de desempenho e esquemas de encaminhamento de pacotes comumente usados.

3.5.1 Conceitos relacionados à interligação

Dependendo das aplicações que demandam rede de alto desempenho, diferentes ambientes de interligação podem ser necessários: sistema (locais), SANs - *System Area Networks*, LANs - *Local Area Networks* (redes locais), MANs - *Metropolitan Area Networks* (metropolitanas) e WANs - *Wide Area Networks* (de longa distância). Sistemas locais ou barramentos de E/S ou *switches crossbar*[5] multinível[6] são interligações encontradas (a) na placa mãe de um computador ou (b) entre computadores localizados em uma mesma sala. As SANs, por sua vez, conectam processadores, memória e outros dispositivos a curtas distâncias. As LANs, MANs e WANs são caracterizadas, respectivamente, (a) por estarem restritas a um prédio ou Campus, (b) por cobrirem uma área como um bairro ou uma cidade e (c) por interligarem um conjunto de redes de menor porte disperso geograficamente em um país ou continente ([19]).

Um conjunto significativo de tecnologias se enquadra em um ou mais dos ambientes mencionados. A Figura 3.3, originalmente apresentada em [19], lista algumas delas

[5]Oferecem conexão direta, implementada em *hardware* através de uma matriz de chaveamento, entre quaisquer portas.

[6]O chaveamento é realizado em etapas usando mais de um dispositivo de comutação.

e as posiciona quanto ao ambiente e às taxas de transmissão suportadas.

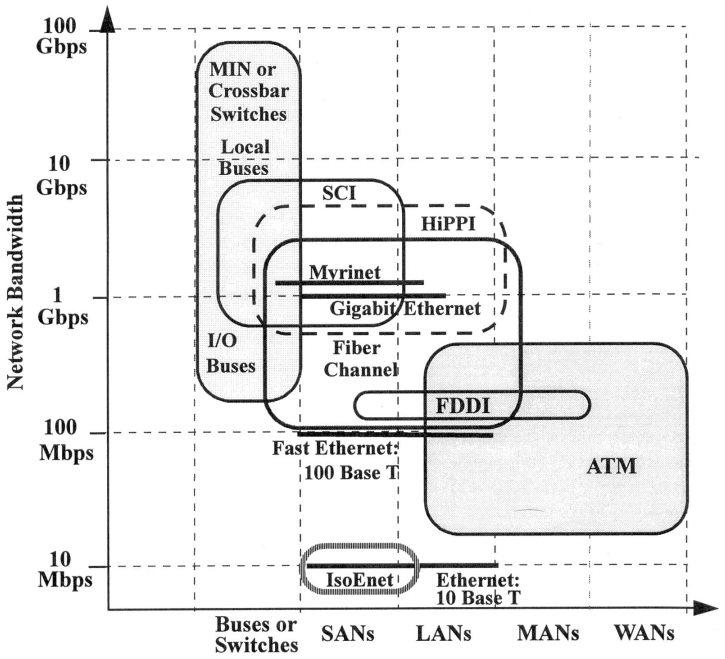

Figura 3.3: Ambientes de interligação e tecnologias de rede para processamento de alto desempenho ([19]).

Embora este capítulo enfatize tecnologias de interligação entre computadores, a Figura 3.4 ilustra um conjunto de termos relacionados a ambientes locais e sua relação com *System Area Networks* e redes locais. Os pinos de um processador constituem o seu barramento (*processor bus*). O barramento que conecta o processador e os módulos de memória é denominado barramento local ou de memória (*local bus* ou *memory bus*). O barramento do sistema ou *system bus*, por sua vez, provê *slots* para interligar dispositivos de entrada e saída tais como unidades de disco, fita e interfaces de rede. Este barramento se conecta ao barramento local através de um circuito que faz essa ponte (I/O *bridge*). ISA, EISA, PCI, Sbus, SCSI e FireWire são alguns padrões estabelecidos de barramentos do sistema. Extrapolando a arquitetura intra-máquina, muitos sistemas, os chamados aglomerados de computadores, possuem seu *hardware* distribuído em um conjunto de máquinas. Os dispositivos usados para interligar essas máquinas para dar forma a um único sistema são as já mencionadas *System Area Networks*. Por outro lado, as redes locais ou LANs são usadas para interligar dois ou mais sistemas. é importante ressaltar, contudo, que algumas tecnologias como Ethernet podem ser usadas para implementar tanto SANs quanto LANs.

Componentes básicos das tecnologias de interligação

A maioria dos aglomerados de computadores, indepentende da tecnologia de interligação usada, é composta por três elementos essenciais: canais (*links*), *switches* e interfaces de rede. O termo canal refere-se a conexões físicas entre dois dispositivos. Essas conexões podem ser realizadas através de fios de cobre, fibras ópticas e, mais recentemente, através do próprio ar (*wireless*). A utilização de fios de cobre como UTP

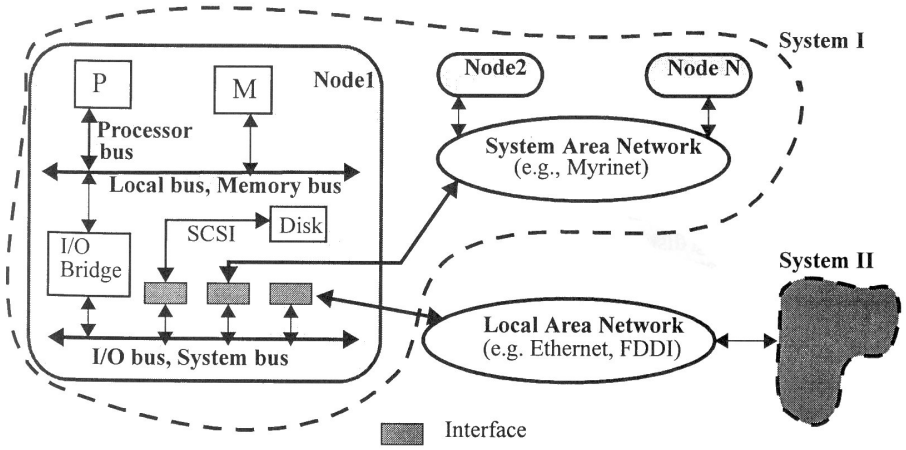

Figura 3.4: Ambientes locais, *System Area Networks* e redes locais ([19]).

(*Unshielded Twisted Pair*) e STP (*Shielded Twisted Pair*) apresenta como vantagem o baixo custo, mas em contrapartida é limitada a distâncias curtas (metros). Canais de fibras ópticas são mais caros, mas oferecem maior largura de banda e permitem ligações a distâncias mais longas (quilômetros).

Os *switches* são peça-chave de boa parte das tecnologias de interligação atualmente disponíveis. Possuem diversas portas de entrada e saída. Cada porta de entrada tem um dispositivo de recepção e um *buffer* para manipular os pacotes que chegam. Cada porta de saída tem associada a si um dispositivo transmissor para encaminhar esses pacotes para outro *switch* ou para uma interface de rede. A Figura 3.5 ilustra um *switch* que possui quatro portas de entrada e quatro de saída. Uma matriz de chaveamento é usada para estabelecer n conexões entre n entradas e n saídas simultaneamente. Cada ponto de cruzamento pode estar em uso ou não.

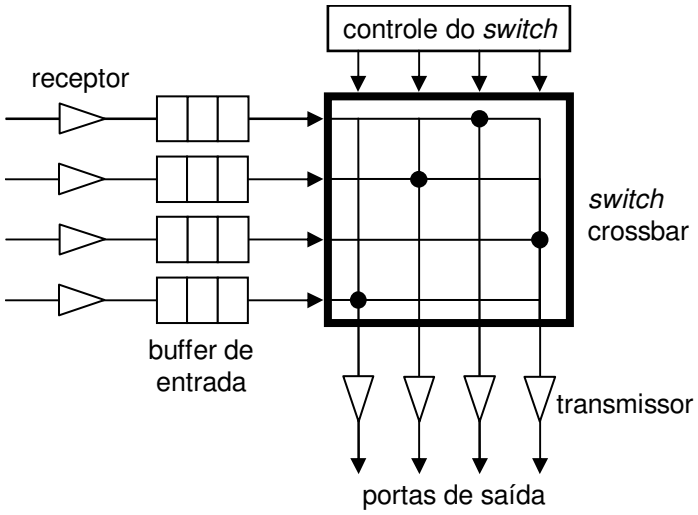

Figura 3.5: *Switch crossbar* de quatro portas.

Já mencionadas anteriormente, as interfaces de rede ou NICs - *Network Interface Cards* são usadas para interligar um computador a uma LAN ou a outro tipo de rede.

As NICs devem ser capazes de manipular tráfego de entrada e saída entre o computador e a rede. Portanto, é natural que a arquitetura de uma interface dependa desses dois componentes. Comum a todas elas é a presença de um processador, *buffers* de entrada e saída e uma lógica de controle bem definida. Entre as funções desempenhadas por uma interface de rede, pode-se citar formatação dos pacotes, seleção de rota, controle de fluxo e de erros.

Métricas de desempenho

Latência é uma das métricas fundamentais para medir o desempenho de tecnologias de interligação. A *latência de comunicação* é dada pelo tempo total necessário para transmitir uma mensagem de uma máquina origem a uma destino, e consiste de quatro componentes: (a) o *overhead* do software associado ao envio e à recepção das mensagens, (b) o atraso do canal, dado pela razão entre o tamanho total da mensagem e a largura de banda do canal, (c) o atraso de roteamento, causado pelo tempo gasto para encaminhar os pacotes em cada *switch* ao longo da rota e (d) o atraso de contenção, causado pelo tráfego que compete pela banda da rede. A partir do conceito de latência de comunicação é possível definir *latência de rede*, que corresponde à soma do atraso do canal e do atraso de roteamento.

Largura de banda por porta corresponde ao número máximo de bits que podem ser transmitidos por segundo a partir de uma porta para qualquer outra na rede. Em uma *rede simétrica* essa largura de banda é independente da localização da porta. Quando essa condição não é satisfeita, a rede é considerada *assimétrica*. **Largura de banda agregada**, por sua vez, é definida como o número máximo de bits por segundo que podem ser transmitidos a partir de metade dos nós da rede para a outra metade.

Esquemas de encaminhamento de pacotes

Dois esquemas de encaminhamento de pacotes são implementados nos equipamentos de interligação como os *switches*: *store and forward* e *cut-through*. No esquema *store and forward* o pacote completo precisa ser armazenado no *buffer* do equipamento antes de ser encaminhado a um canal de saída. Dessa forma, pacotes sucessivos são transmitidos seqüencialmente sem sobreposição no tempo. Já no esquema de encaminhamento *cut-through*, cada nó usa um *buffer* especial que armazena o início do pacote, onde se encontra o cabeçalho. O conteúdo desse *buffer*, após ser decodificado, é encaminhado para o canal de saída. O restante do pacote, à medida em que continua sendo recebido, vai sendo encaminhado pelo mesmo caminho. A Figura 3.6 ilustra a transmissão de um pacote do nó 1 ao 4, passando por dois nós intermediários, considerando os dois tipos de encaminhamento. Como é possível observar, usando o esquema *cut-through* foram necessárias sete unidade de tempo para o pacote ser recebido por completo no nó 4, ao passo que dezesseis unidades de tempo foram requeridas usando o esquema *store and forward*. Boa parte das tecnologias de interligação usadas para processamento de alto desempenho utiliza o esquema *cut-through* em razão dessa maior eficiência.

3.5.2 Gigabit Ethernet

Gigabit Ethernet é uma extensão dos padrões de rede Ethernet (10 Mbps) e Fast Ethernet (100 Mbps), tendo sido padronizado em junho de 1998 pelo IEEE (IEEE 802.3z).

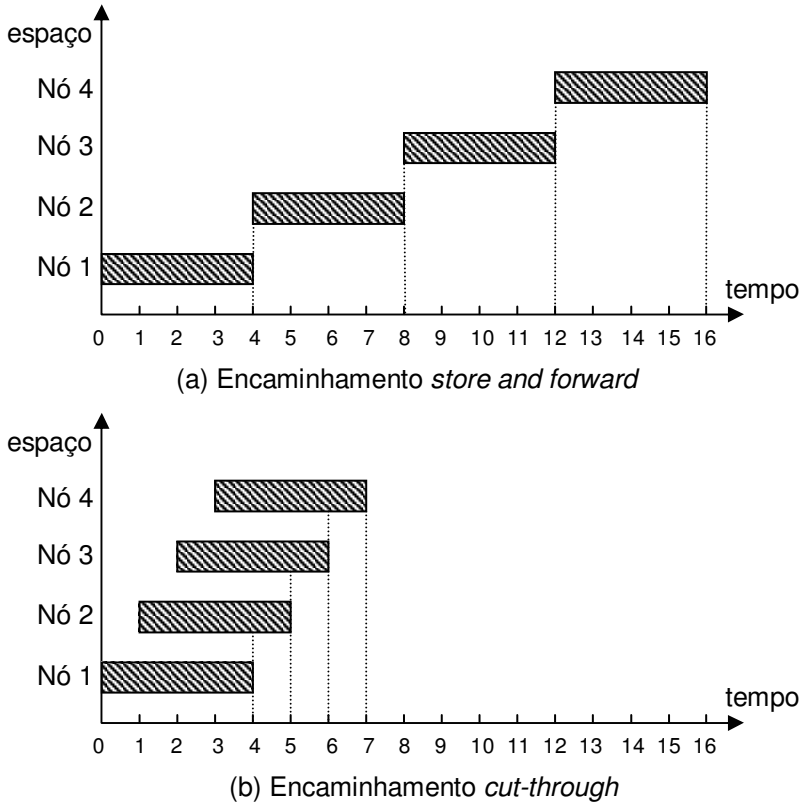

Figura 3.6: Encaminhamento *store and forward* x *cut-through*.

Ao mesmo tempo em que propõe novas configurações relacionadas aos meios de transmissão, o padrão mantém o protocolo CSMA/CD[7] para controlar o acesso ao meio e o formato dos quadros Ethernet usados pelos padrões predecessores. Esta compatibilidade permite uma transição gradual de redes Ethernet e Fast Ethernet para Gigabit Ethernet, o que representa um importante atrativo para grande parte das organizações que há muito utilizam essa tecnologia ([40]). A seguir são abordadas as principais características do padrão.

Acesso ao meio. Conforme definido no padrão IEEE 802.3x, dois nós conectados por um caminho *full-duplex* podem simultaneamente enviar e receber pacotes. O padrão Gigabit segue essa especificação na comunicação *full-duplex*. é importante destacar que, quando operando nesse modo, não ocorrem colisões na rede. Por outro lado, para operar no modo *half-duplex*, o padrão utiliza uma versão aprimorada do protocolo CSMA/CD com o propósito de controlar o acesso ao meio.

Quando o padrão opera no modo *full-duplex*, ele utiliza *buffers* para armazenar quadros que chegam à interface ou que precisam ser enviados, até que a sub-camada MAC tenha condições de repassá-los aos níveis superiores ou ao nível físico. Durante

[7]Com CSMA/CD (*Carrier Sense Medium Access with Collision Detection*) apenas um computador pode transmitir por vez. Uma colisão ocorre sempre que dois computadores escutarem o barramento, não identificarem tráfego, e iniciarem a transmitir. Assim que os computadores envolvidos detectarem a colisão, eles interrompem a transmissão, aguardam um certo tempo e procuram realizar novamente a transmissão do quadro que colidiu. Para serem capazes de detectar essa colisão os computadores precisam estar ainda transmitindo o quadro. Para garantir esse requisito, o quadro sendo transmitido deve ser suficientemente grande para que ele se propague até o final do barramento e uma possível informação de colisão chegue até ele (*slot* de transmissão).

Tabela 3.1: Meios de transmissão e distâncias máximas suportadas.

	10 BASE-T	100 BASE-X	Gigabit Ethernet
Taxa de transmissão(Mbps)	10	100	1000
UTP-5 (m)	100	100	25-100
Cabo coaxial	500	100	25-100
Fibra multimodo	2 km	400m (half-duplex) 2 k (full-duplex)	500m
Fibra monomodo	25km	20km	2km

transmissões que envolvem tráfego elevado é possível que o *buffer* de saída fique cheio, uma vez que a sub-camada MAC não consegue processar todos os quadros com a agilidade necessária. Quando isto ocorre, ela impede que as camadas superiores enviem dados até que exista espaço disponível no *buffer* de saída para armazenar mais quadros, evitando, assim, perdê-los. De forma similar, quando o *buffer* de recepção está próximo de ficar cheio, a sub-camada MAC do nó receptor envia um sinal para o nó transmissor solicitando que o mesmo interrompa a transmissão por um período de tempo determinado. O nó transmissor pára de enviar quadros até que esse período encerre ou até receber um novo pacote do nó receptor solicitando a retomada da transmissão.

Operando no modo *half-duplex* o desempenho das redes Gigabit Ethernet é bastante comprometido devido à utilização do protocolo CSMA/CD, que é sensível ao comprimento do quadro. O *slot* de transmissão padrão para quadros Ethernet (64 bytes) não é grande o suficiente para percorrer 200 metros de cabo à velocidade de 1 Gbps. Para solucionar esse problema, a duração do *slot* foi modificada para 512 bytes usando uma técnica denominada *carrier extension*. Na realidade, o tamanho mínimo do quadro não foi modificado. Sempre que um quadro menor que 512 bytes tiver que ser transmitido, ele é completado com símbolos especiais. Como é possível observar, esta técnica provoca desperdício de banda. Por exemplo, um quadro de 64 bytes será preenchido com 448 bytes antes de ser transmitido.

Para a realização de processamento de alto desempenho é ideal que se utilize transmissão *full-duplex*, posto que é mais eficiente (conexões ponto-a-ponto com largura de banda disponível dedicada entre os dois nós interligados).

Meios de transmissão. O meio de transmissão preferencial do padrão Gigabit Ethernet é a fibra óptica, mas par trançado (UTP e STP) e coaxial também são suportados. A Tabela 3.1 apresenta os meios de transmissão admitidos pelos padrões Ethernet, Fast e Gigabit Ethernet, informando as taxas de transmissão e as distâncias máximas que devem ser respeitadas.

Topologias. Gigabit Ethernet suporta configurações de rede local compartilhada e segmentada. As configurações de rede possíveis em ambiente compartilhado são restritas a (a) um canal ponto-a-ponto ligando exatamente duas estações ou (b) um repetidor (*hub*) com ligações de par trançado, cabo coaxial ou fibra óptica usando um método único de codificação. Esta limitação implica a observação de poucos requisitos no projeto: garantir (a) que o comprimento dos canais seja menor que o tamanho máximo admitido e (b) que repetidores não sejam interligados dentro de um mesmo domínio de colisão. Estações conectadas em um ambiente compartilhado operam no modo *half-duplex*.

Por outro lado, a transmissão *full-duplex* pode ser realizada (a) entre dois *switches*, (b) entre um computador e um *switch* ou (c) entre dois computadores, possibilitando a configuração de um conjunto variado de topologias, embora a em estrela seja mais usual. A única restrição para o projeto de redes Gigabit Ethernet segmentadas está relacionada à observação dos comprimentos máximos para os meios de transmissão. A Figura 3.7 ilustra um ambiente típico de interligação baseado na tecnologia Gigabit Ethernet.

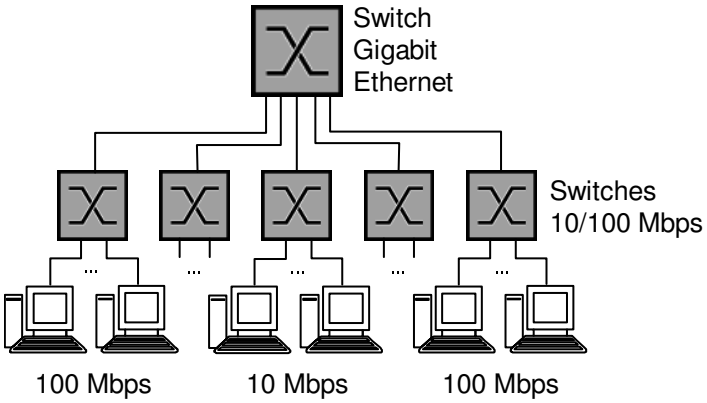

Figura 3.7: Configuração típica de uma rede Gigabit Ethernet.

10 Gigabit Ethernet. Em 1999 foi criada a *10 Gigabit Ethernet Alliance*, formada por um conjunto de empresas, com o objetivo de propor uma tecnologia, baseada em Ethernet, para operar à taxa de 10 Gbps e ser utilizada para implantar MANs e WANs, além de redes locais. A tecnologia, padronizada em 2002 (IEEE 802.3ae), é compatível com as antecessoras (Ethernet, Fast e Gigabit), opera exclusivamente no modo *full-duplex* e requer a utilização de fibra óptica como meio de transmissão ([1]).

3.5.3 Myrinet

Myrinet é uma tecnologia de rede comutada, de alta velocidade, de propriedade da empresa Myricom ([8]). Segundo Hwang et al. ([19]), um dos objetivos da empresa é oferecer produtos para permitir a implantação, de forma confortável e rápida, de aglomerados de computadores.

Myrinet é definida no nível de enlace e é caracterizada por (a) pacotes de tamanho variável, (b) controle de erro e de fluxo em cada canal, (c) uso de *switches cut-through* para encaminhar os pacotes e (d) estações com interfaces programáveis. No nível físico, quando utilizada como uma SAN, Myrinet utiliza canais *full-duplex* de até 3 metros, atingindo uma taxa de transmissão de 2 + 2 Gbps. Ao utilizá-la como uma tecnologia de rede local, 2 + 2 Gbps podem ser atingidos em cabos seriais de até 10 metros ou em fibras ópticas de até 200 metros ([30]).

***Switches* Myrinet.** A rota dos pacotes em uma rede Myrinet é fixa e definida na origem (*source-routed*). Usualmente os computadores usam o algoritmo de roteamento *up*/down** para calcular rotas livres de *deadlock* para qualquer outro computador na rede. Dado o destino de um pacote, o computador origem adiciona ao cabeçalho do pacote informações sobre cada porta a ser usada ao longo do caminho até o computador destino. Os *switches* Myrinet lêem o primeiro byte do cabeçalho de cada pacote que chega para

determinar o canal de saída sugerido. Em seguida, este byte é removido e o CRC da mensagem é recalculdado. O próximo *switch* lê e remove o "novo" primeiro byte, encaminha o pacote para o *switch* seguinte, e assim por diante. Para implementar esse mecanismo o cabeçalho inicial do pacote deve possuir um byte para cada *switch* ao longo da rota e, por essa razão, o seu tamanho vai diminuindo em um byte a cada *switch* por que passa. Como as rotas têm diferentes comprimentos, o tamanho inicial do cabeçalho é variável.

O esquema de encaminhamento usado pelos *switches* Myrinet é o *cut-through* bloqueante[8]. Dentro de cada *switch* existe uma matriz de chavamento com controle de fluxo e *buffers* de entrada similares ao ilustrado na Figura 3.5. Um pacote avança para um canal de saída tão logo o cabeçalho seja recebido e decodificado. Seguindo esse esquema de encaminhamento múltiplos pacotes podem trafegar simultaneamente. O *payload* de um pacote Myrinet possui tamanho arbitrário, podendo carregar qualquer tipo de pacote (como IP) sem um nível de adaptação.

Interfaces de rede. As interfaces de rede utilizadas possuem barramento de 64 bits, processador RISC (*chip* LaNai) dotado de interface Myrinet, interface de pacotes, DMA e memória RAM *fast static*. Esta memória é usada para armazenar pacotes e o *Myrinet Control Program* (MCP), responsável pelas transferências de mensagens entre o computador e a rede.

Topologias. As redes Myrinet podem ter qualquer topologia, não ficando restritas a *mesh* de *switches* ou outra topologia convencional. *Switches* de múltiplas portas são conectados por canais a outros *switches* ou a interfaces de computadores. A Figura 3.8 ilustra a utilização de três *switches* para construir uma rede local Myrinet.

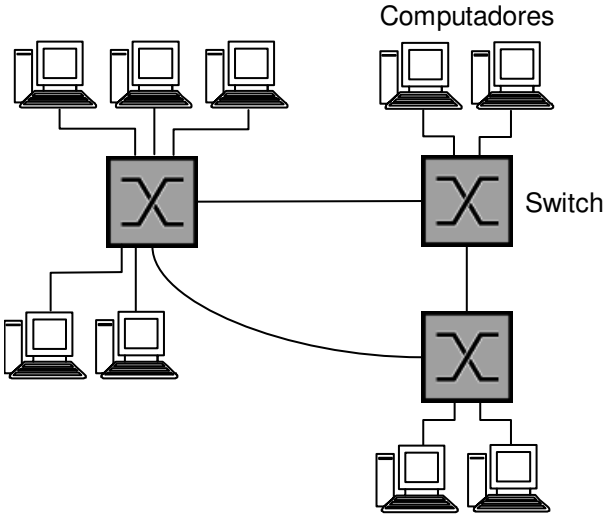

Figura 3.8: Configuração possível de rede local Myrinet.

APIs. Os mecanismos de comunicação disponíveis em tecnologias de interligação convencionais são, em geral, baseados na pilha TCP/IP. Isto não chega a ser um problema para redes que operam em taxas de transmissão baixas (p.ex., 10 Mbps). Conforme já mencionado na Seção 3.4.2, a comunicação nesse cenário requer a intervenção direta do

[8]Se o canal de saída requisitado no pacote não estiver disponível o pacote é bloqueado e armazenado em um *buffer* até que o canal seja liberado ou que o tempo de espera se esgote. Nesse último caso o pacote é descartado.

sistema operacional e a realização de cópias de memória à medida em que os pacotes transitam pela pilha de protocolos. Para maximizar o aproveitamento da capacidade de transmissão da tecnologia Myrinet, o fabricante oferece uma API para acessar diretamente o hardware de rede, sobrepassando o sistema operacional (chamadas de sistema e cópias de memória desnecessárias) ([33]). A Figura 3.9 ilustra as opções de comunicação disponíveis em interfaces de rede Myrinet. Como o *Myrinet Control Program* é aberto, alguns grupos de pesquisa desenvolveram seus próprios programas de controle, alcançando baixa latência e alta vazão ([37, 33]).

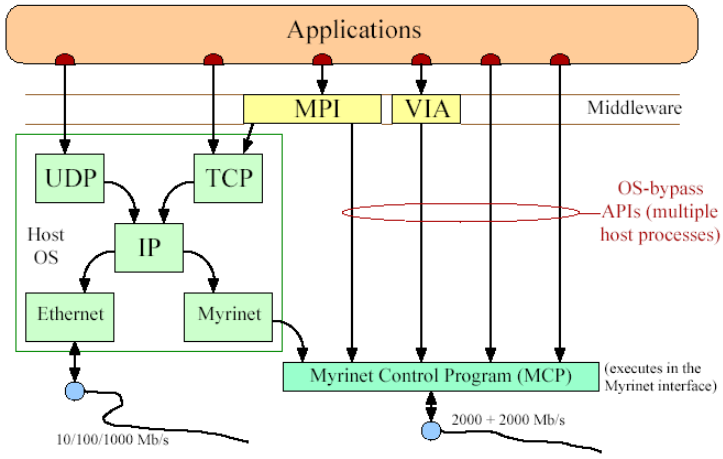

Figura 3.9: Estrutura básica de sobrepasso do sistema para redes Myrinet ([41]).

3.5.4 ATM (Assynchronous Transfer Mode)

ATM é uma tecnologia definida pelo ATM Forum (fundado em 1991) e pelo ITU (*International Telecommunication Union*). Na realidade trata-se de um protocolo de transmissão independente do meio que segmenta o tráfego em células de 53 bytes para serem transmitidas em um ambiente baseado em comutação de células[9]. Um dos objetivos do ATM é oferecer, em uma única tecnologia, mecanismos para transmissão eficiente de dados sensíveis e não sensíveis ao atraso. Para tal, a tecnologia provê alocação dinâmica de banda e comutação em alta velocidade.

Estabelecimento de caminhos e canais virtuais. ATM é uma tecnologia orientada à conexão, permitindo ao usuário especificar os recursos necessários e à rede alocar os recursos baseada nessas necessidades. Dessa forma, os diferentes fluxos de dados podem ser tratados de forma individualizada. Dois tipos de conexões estão presentes no padrão: *Virtual Path Connections* (VPCs), que contêm *Virtual Channel Connections* (VCCs), conforme ilustra a Figura 3.10. Uma VCC ou circuito virtual é uma unidade básica de transmissão e transporta um único fluxo de células, em ordem, de um usuário para outro. Um conjunto de circuitos virtuais pode ser agrupado em uma VPC. Essas

[9]A comutação de células difere da comutação de pacotes por manipular e encaminhar pacotes de tamanho fixo e pequeno (células). As vantagens dessa comutação são a possibilidade de ter um atraso de transmissão constante, o que é desejável para transmissões de tráfego sensível como voz, e um custo reduzido na produção do hardware que faz a comutação.

conexões também podem ser criadas fim-a-fim. Uma das vantagens desse tipo de agrupamento é proporcionar um tempo de recuperação mais rápido em caso de falha na rede.

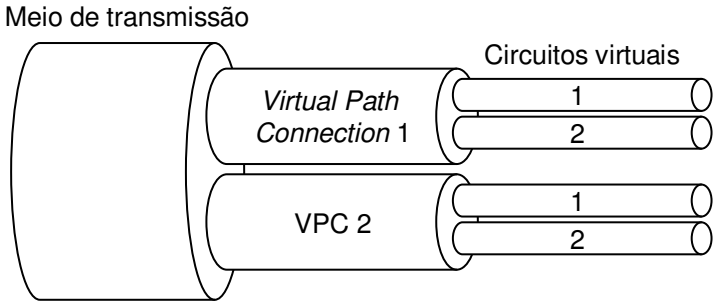

Figura 3.10: *Virtual Path Connections* e *Virtual Channel Connections*.

Encaminhamento de células. As células ATM podem ser roteadas através do esquema *source-routing* ou do esquema *hop-by-hop*. Conforme já foi apresentado na Seção 3.5.3, no esquema *source-routing* todas as informações sobre a rota a ser seguida pela célula são incluídas em seu cabeçalho. Já no esquema *hop-by-hop* apenas o identificador de salto deve estar presente no cabeçalho, proporcionando uma forma mais flexível de escolha das rotas. Boa parte dos *switches* ATM implementa o esquema *hop-by-hop*. Na Figura 3.11 é apresentado o conceito de comutação de células baseado nesse esquema. Em cada canal de entrada de um *switch* ATM o *virtual path identifier* (VPI) e o *virtual channel identifier* (VCI) de cada célula que chega identificam, de forma unívoca, o caminho e o canal pelos quais a célula percorerrá. Um novo identificador virtual será armazenado no cabeçalho dessa célula no canal de saída. As tabelas de roteamento são iniciadas por um procedimento de estabelecimento de caminho virtual.

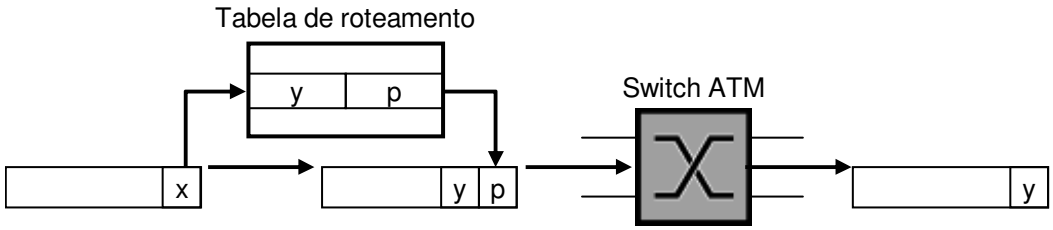

Figura 3.11: Comutação de células baseada no esquema *hop-by-hop*.

Ao entrar em um *switch* ATM, o campo VPI da célula (x) é usado para selecionar uma entrada na tabela de roteamento. Esta entrada contém o número da porta de saída (p), pela qual a célula deve ser encaminhada. Um novo valor para o VPI (y) será armazenado na célula antes dela ser encaminhada ao próximo *switch*. Sobre o mesmo caminho virtual dois computadores podem multiplexar fluxos de várias aplicações, usando o campo VCI para distinguir um fluxo de outro.

Classes de serviço. O padrão ATM oferece cinco classes de serviço, conforme ilustra a Tabela 3.2. Cada nova requisição de conexão precisa indicar uma dessas classes - aquela que for mais apropriada para atender aos requisitos da aplicação.

Tabela 3.2: Classes de serviço suportadas pelo padrão ATM.

Classe	Parâmetros de qualidade
Constant bit rate (CBR)	Esta classe é usada para emular comutação de circuitos. A taxa de transmissão de células é constante no tempo. Exemplos de aplicações que requerem tráfego CBR são telefonia e videoconferência.
Variable bit rate-non real time (VBR-NRT)	A classe VBR-NRT permite aos usuários enviar tráfego a uma taxa que varia com o tempo de acordo com a existência de informações a serem transmitidas. Multiplexação estatística é utilizada para proporcionar o uso eficiente dos recursos da rede. Um exemplo de aplicação que se encaixa a essa classe de serviço é e-mail multimídia.
Variable bit rate-real time (VBR-RT)	Esta classe é similar à anterior, mas foi projetada para aplicações sensíveis a variações no atraso das células. Vídeo comprimido interativo é um exemplo de aplicação que requer um serviço de rede como o VBR-RT.
Available bit rate (ABR)	Esta classe de serviço provê controle de fluxo baseado na taxa e é destinado à transmissão de dados como arquivos e e-mail.
Unspecified bit rate (UBR)	A classe UBR não oferece nenhum tipo de garantia, sendo usada atualmente para trafegar TCP/IP.

Velocidade. As redes ATM podem operar em diversas velocidades, variando de 25 a 51 Mbps, 155 e 622 Mbps. Naturalmente, quanto mais baixa a velocidade menor será o custo dos *switches* e dos canais de comunicação envolvidos.

3.5.5 *Fiber Channel*

O padrão ANSI X3T11 especifica FC - *Fiber Channel* como um conjunto integrado de padrões de rede, de armazenamento e de transferência de dados entre estações, *mainframes,* supercomputadores, dispositivos de armazenamento e *displays*. FC é uma tecnologia que busca oferecer mecanismos para transferência rápida de grandes volumes de dados. Tem sido utilizada para comunicação *servidor-dispositivo de armazenamento* e *servidor-servidor*, na implantação das chamadas *Storage Area Networks*.

Meios de transmissão. *Fiber Channel* pode ser implementada tanto em meio compartilhado quanto em ambiente segmentado, operando a uma velocidade que varia de 100 a 133, 200, 400, 800 Mbps e 2 Gbps. Com relação aos meios de transmissão, FC opera sobre par trançado (STP) a 100 Mbps, se respeitada a distância de 50 m, e sobre fibra óptica monomodo de até 10 quilômetros. Com fibra multimodo de até 2 quilômetros a tecnologia atinge a taxa de 200 Mbps.

Topologias. *Fiber Channel* suporta três topologias: ponto-a-ponto, *loop* arbitrário e estrela (*switched fabric*):

- *ponto-a-ponto*: permite conectar um computador a outro computador ou a um disco, oferecendo a maior largura de banda das três topologias;

- *loop* arbitrário: permite conectar até 126 dispositivos. É adequada para a interligação de um número grande de dispositivos de armazenamento. A largura de banda

disponível é compartilhada entre todos os dispositivos. A vantagem da topologia em anel é o seu baixo custo, uma vez que não requer a utilização de *switch*;

- estrela: esta topologia é a que oferece maior vazão. Diversos dispositivos operando em diferentes velocidades podem ser conectados ao *switch* central (*fabric switch*).

A Figura 3.12 ilustra a estrutura típica de uma *Storage Area Network* com a utilização da tecnologia *Fiber Channel*. Esta rede fica dedicada para a conexão entre servidores e dispositivos de armazenamento. A *Storage Area Network* é usada por poucos servidores, que atendem a um grande número de usuários ligados a eles por uma rede tradicional (por exemplo, IP trafegando sobre Ethernet). A rede *Fiber Channel* é otimizada para grandes transferências de dados com baixa sobrecarga, baixa latência, e interrupções mínimas nos fluxos de dados. Um usuário típico não tem acesso direto à *Storage Area Network*.

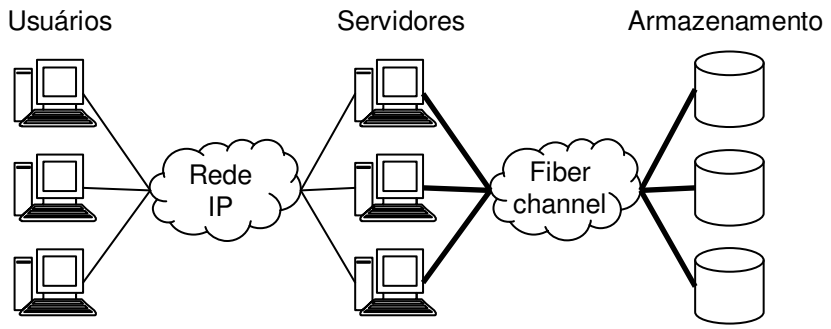

Figura 3.12: Configuração típica de uma *storage area network* usando *Fiber Channel*.

3.5.6 SCI - *Scalable Coherent Interface*

Todas as tecnologias de interligação apresentadas, como Gigabit Ethernet, Myrinet, Fiber Channel e ATM, possuem um aspecto em comum: dependem da conexão dos nós ao barramento de entrada e saída, e a comunicação ocorre pela troca de mensagens entre os nós. Segundo Gustavson e Li ([17] *apud* [19]), este tipo de comunicação possui desempenho inferior à comunicação via memória compartilhada.

A comunicação através de memória compartilhada possui baixa latência, uma vez que a comunicação se restringe a um processador executando uma intrução store e um outro processador executando uma instrução load. Por outro lado, quando se trata da comunicação em que o barramento de entrada e saída é envolvido, centenas de instruções precisam ser executadas. Além disso, nesse tipo de comunicação é difícil aproveitar as informações armazenadas em *cache* que são geradas pelo hardware em um agregado de computadores baseado em barramento.

O padrão IEEE/ANSI 1596-1992, que define SCI - *Scalable Coherence Interface*, levou os aspectos supracitados em consideração ao propor uma extensão do barramento local (ou de memória) para suportar uma estrutura de interligação ponto-a-ponto, *full-duplex* que provê uma imagem consistente de *cache* de memória compartilhada distribuída ([18, 12]).

Interligações SCI. A tecnologia foi projetada para prover interligações ponto-a-ponto de baixa latência e alta largura de banda. A versão mais recente do padrão (IEEE

1596-1996) especifica larguras de banda que variam de 250 MB/s a 8 GB/s. As ligações podem ser realizadas com cabos metálicos ou fibra óptica.

A consistência de *cache* é implementada a partir de uma lista ligada de módulos SCI. O processador de cada nó é associado a um módulo SCI. Quando um desses processadores atualiza sua *cache*, o estado da *cache* é propagado aos demais módulos SCI. Esta lista ligada é escalável, uma vez que um sistema maior pode ser implantado adicionando mais módulos SCI a ela.

Interface do nó. A interface do nó é ilustrada na Figura 3.13. Um canal SCI possui dezoito bits (um para o *clock*, um bit *flag* e dezesseis bits de dados), ou seja, transmite dezoito bits simultaneamente (*símbolo*). O pacote possui um cabeçalho de quatorze bytes, dois bytes para armazenar o CRC e 0, 16, 64 ou 256 bytes de dados. A cada ciclo do *clock*, um símbolo (dois bytes do pacote) pode ser transmitido. O *clock* e o bit de *flag* são gerados pelo hardware.

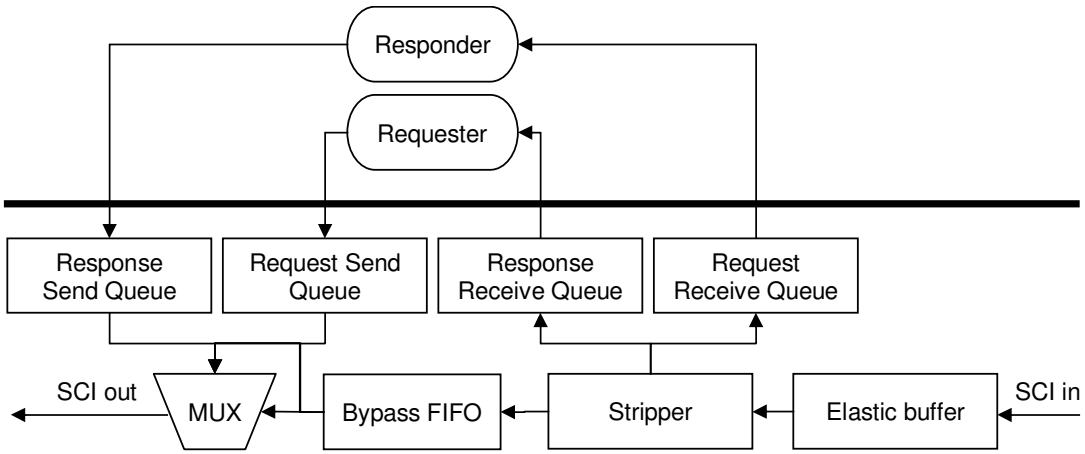

Figura 3.13: Estrutura da interface de um nó SCI ([19]) .

Suponha que um nó origem deseje ler dados de um nó destino. O nó origem, denominado *requester,* injeta um pacote de requisição (*request*) no anel SCI através da fila *resquest send queue*. O cabeçalho da requisição contém, entre outras informações, o endereço da origem, do destinatário e o comando. O pacote transita pelo anel, de nó a nó, até alcançar o nó destino.

Em cada nó, o *stripper* retira do anel os pacotes cujo endereço coincida com o seu e os armazena em uma das filas de recepção. O nó destino (ou *responder*), após interpretar a requisição, busca o dado, prepara um pacote de resposta e o injeta no anel SCI.

Quando um pacote chega a um nó que não é o destino ele é diretamente encaminhado para o canal de saída. Se o canal já estiver sendo ocupado pelo próprio nó, i.e. se ele próprio estiver enviando dados, o pacote é armazenado temporariamente em um *buffer* (*bypass* FIFO) até que o canal fique disponível.

Temporização. SCI é uma tecnologia *síncrona*, ou seja, todos os nós e circuitos operam com base no mesmo relógio. Para lidar com variações do relógio, um *buffer* elástico é usado em cada nó para adicionar e remover símbolos indicando ociosidade (*idle*) em pacotes que chegam à interface.

Quando um nó transmissor não possui pacotes a transmitir ele envia símbolos *idle* para manter o receptor sincronizado. Mais tarde, quando o transmissor tiver paco-

tes a transmitir ele pode fazê-lo imediatamente, sem precisar enviar um preâmbulo de sincronização, como ocorre em tecnologias como Ethernet.

O protocolo de comunicação SCI foi projetado para garantir o uso justo da rede e para evitar a ocorrência de *deadlocks* e *livelocks*. Um exemplo disso é o uso de filas distintas de entrada e saída. Sem essa separação seria possível que o recebimento de um número elevado de requisições (próximas umas das outras) impedisse o envio de uma resposta, provocando *deadlock*.

Protocolo de consistência das *caches*. O esquema usado por SCI para manter a consistência das *caches* é baseado em diretórios ([18]). Este diretório possui informações sobre que dados estão sendo utilizados por que *caches* de modo que elas possam ser atualizadas sempre que necessário. A implementação do diretório se dá através de uma lista duplamente encadeada distribuída, onde o ponteiro para o início da lista é armazenado em um controlador de memória e os ponteiros responsáveis pela ligação da lista são armazenados nos próprios controladores das *caches* (vide Figura 3.14).

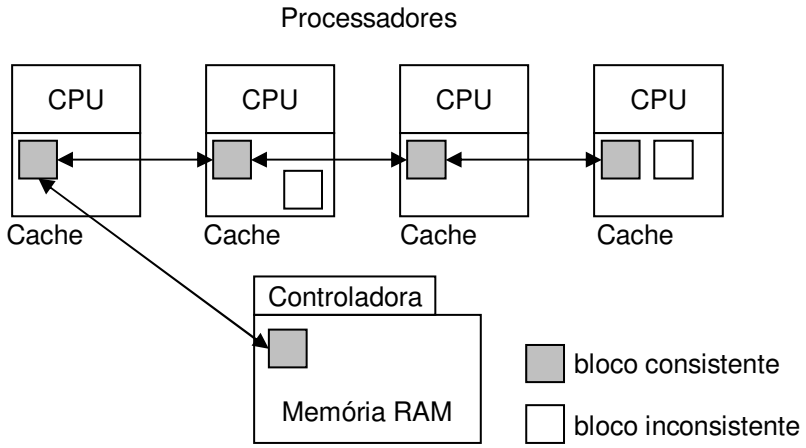

Figura 3.14: Protocolo SCI para consistência das *caches*.

O mecanismo de consistência de *cache* opera da seguinte forma: o controlador de memória/*cache* de qualquer processador que não tiver a instrução a ser executada armazenada em *cache* (*cache miss*) realiza uma requisição SCI para a memória que foi endereçada. Quando a requisição chega à memória, o controlador de memória substitui incondicionalmente o campo de 16 bits nodeID (ponteiro para quem está fazendo a requisição) pelo identificador de quem requisitou o mesmo dado antes.

Essa troca informa a quem está fazendo a requisição o endereço de seu predecessor, formando uma lista simplesmente encadeada. Se a memória possui um dado válido ela retorna esse dado e o ponteiro; caso contrário ela retorna o ponteiro e um indicador de *status*. Em qualquer um dos casos o requisitante atual contata o requisitante anterior, se houver, e solicita duplo encadeamento com ele.

Após essa ligação e de ter recebido o dado de seu predecessor ou da memória, o nó requisitante se torna o primeiro da lista e permanece como tal até que receba a ordem para se ligar ao sucessor. Apenas o primeiro nó da lista pode modificar dados (e apenas se a requisição original notificou a memória dessa intenção). Se dados forem modificados, o nó precisa percorrer toda a lista duplamente encadeada e invalidar as outras cópias, eliminando a lista.

Topologias. A topologia de uma rede SCI é bastante flexível. Cada nó pode ser um processador com memória e dispositivos de entrada e saída. Independente dessa estrutura, o nó possui um canal de entrada e outro de saída, que são conectados a um anel SCI ou a um *crossbar*.

3.5.7 InfiniBand

InfiniBand ([21, 22]) é uma tecnologia de interligação apoiada por fabricantes como Intel, IBM, Sun, HP e Microsoft, e que especifica uma arquitetura de hardware para interligação de alto desempenho intra e inter-estações (por exemplo, substituindo PCI e Gigabit Ethernet, respectivamente). Em teoria, a arquitetura InfiniBand (**IBA**, de *InfiniBand Architecture*) é independente do sistema operacional da estação e da CPU. Em estações convencionais, um dos principais fatores limitantes no desempenho é o subsistema de E/S, particularmente o barramento compartilhado. A proposta da arquitetura InfiniBand é substituir este barramento local, compartilhado, por uma estrutura de interligação baseada em uma malha de comutação (*switching fabric*). A principal aplicação (alvo) de InfiniBand é a interligação de servidores (de dados) em um mesmo agregado (como a seguir descrito).

A Figura 3.15 ilustra a arquitetura InfiniBand através de um exemplo. Existem três nós processadores, cada um contendo múltiplas CPUs interligadas através de um barramento comum com a memória e um ou mais controladores HCA. Estes controladores são conectados ponto a ponto com uma malha de *switches*. Além de nós processadores, a rede conta com subsistemas de E/S (na figura, *RAID Subsystem* e *Storage Subsystem*), cada um contendo várias unidades de disco, e *chassis* de E/S que servem como conversores entre InfiniBand e outros formatos (como Ethernet e SCSI), via módulos de E/S.

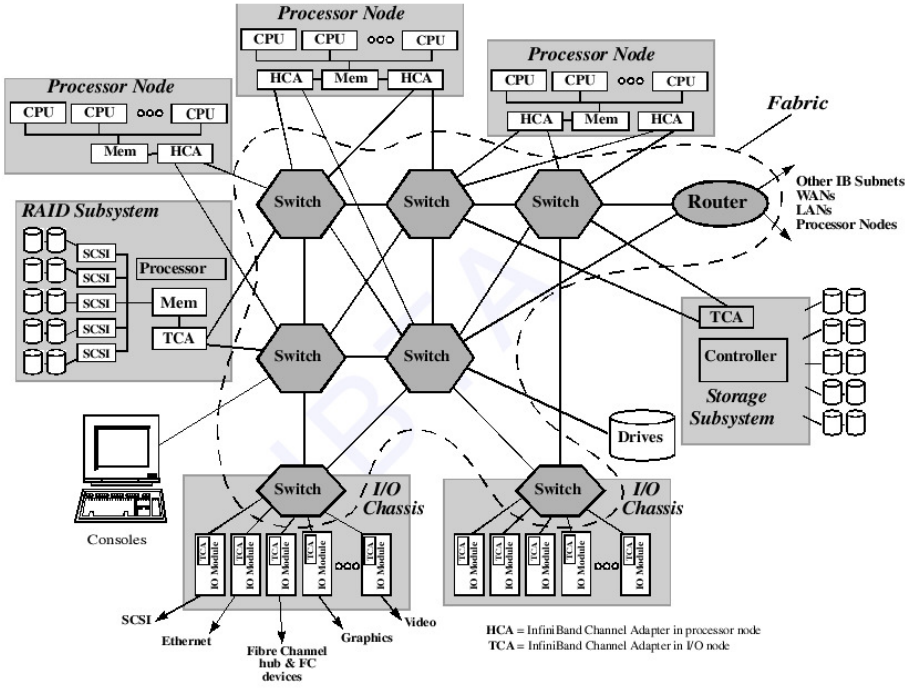

Figura 3.15: Visão global da arquitetura InfiniBand ([22]).

A malha comutada provê um mecanismo de transporte confiável onde mensagens são enfileiradas para entrega entre nós fim. O formato ou significado de mensagens não é especificado pela arquitetura. IBA define protocolos de transporte confiáveis implementados **em hardware** que oferecem transmissão confiável (com *send*/*receive*) e semântica de compartilhamento de memória (p.ex., DMA remoto) sem intervenção do software para movimentação dos dados ([22]). A IBA define mecanismos de proteção e detecção de erro que permitem que transações iniciem e terminem ou do modo de *kernel* privilegiado ou do espaço de usuário.

3.5.8 ServerNet

ServerNet ([32]) é uma tecnologia similar à InfiniBand, promovida pela Compaq. Um agregado ServerNet (*NonStop ServerNet Cluster*) é uma rede de servidores *NonStop Himalaya S-series* interligados através do protocolo ServerNet e compartilhando um transporte comum, para comunicação intra e inter-estação. O agregado oferece comunicação fim-a-fim rápida sem pilhas de protocolos intermediárias, atingindo velocidades comparáveis para comunicação na fibra ServerNet interna e externa.

3.5.9 Comparação das Tecnologias

Após terem sido abordadas cinco tecnologias de interligação que têm sido utilizadas para suportar ambientes de processamento de alto desempenho, esta seção apresenta uma breve reflexão sobre as mesmas, apontando vantagens e desvantagens, cenários de aplicação e alguns aspectos relacionados ao desempenho.

A tecnologia Gigabit Ethernet é a que oferece maior facilidade de implantação. Por ser totalmente compatível com seus padrões predecessores, pode ser implantada de forma gradual, a um custo bastante baixo. No que se refere aos mecanismos de transmissão oferecidos, a tecnologia não garante tempo mínimo de entrega de quadros, não permite prever latência e não garante atraso constante. Essas métricas são influenciadas diretamente pela topologia da rede e pela natureza do tráfego gerado pelas aplicações executadas ([15, 27]). Nesse contexto, Gigabit Ethernet mostra-se adequada para transportar dados relacionados à Intranet, arquivos, aplicações cliente-servidor, aplicações de bancos de dados e *datawarehouse*.

Myrinet é uma tecnologia proprietária, de difícil integração com outras tecnologias de interligação disponíveis e de custo elevado (se comparada com Gigabit Ethernet). Um ponto a favor da tecnologia reside nas interfaces de rede para onde algumas tarefas, anteriormente realizadas pelo sistema operacional, foi transferida. Além disso, a possibilidade de desenvolver aplicações que sobrepassam o sistema operacional, graças a API oferecida pelo fabricante, proporciona baixa latência e alta vazão. No que se refere aos cenários de aplicação da tecnologia, Myrinet mostra-se particularmente adequada para a implantação de *System Area Networks*.

ATM foi originalmente concebida para ser uma tecnologia convergente, capaz de integrar diferentes tipos de serviços (ex: vídeo, voz e dados), e para ser amplamente utilizada tanto em ambientes locais quanto em interligações de longa distância. Em função do alto custo dos equipamentos e da incompatibilidade com outras tecnologias em uso há anos nas instituições, ATM não teve boa aceitação em ambientes locais. Em ambiente WAN, em contrapartida, a tecnologia ainda tem sido bastante utilizada. Um aspecto importante a favor de ATM é o suporte a diferentes classes de serviço, possibilitando a

transmissão adequada de diversos tipos de informações, sejam elas sensíveis ou não ao tempo; a classe de serviço é contratada no estabelecimento de cada conexão, onde são negociados parâmetros como vazão desejada, além de atraso e *jitter* aceitáveis.

Embora possa ser usada para implantar vários tipos de ambientes, *Fiber Channel* é uma tecnologia adequada para a implantação de *Storage Area Networks*. Entre os aspectos positivos de FC vale citar confiabilidade, interface de comunicação com diversos padrões como SCSI e ESCON e escalabilidade.

SCI utiliza uma abordagem de comunicação baseada em memória compartilhada, enquanto boa parte das aplicações relacionadas a processamento de alto desempenho é construída sobre MPI e PVM, que exploram troca de mensagens. Por se tratar de uma tecnologia proprietária, apresenta dificuldades de interoperar com outras tecnologias de rede. Em [19], SCI é apresentada como uma alternativa adequada para interligar ambientes multiprocessados.

3.6 Comunicação de Rede em Nível de Usuário

Conforme visto na Seção 3.4, em um computador o software responsável pela interligação em rede reside no sistema operacional, pois interage com dispositivos de hardware e esconde a complexidade do usuário, gerenciando de maneira robusta e justa os recursos físicos do computador. Sua complexidade faz com que implementações sigam uma arquitetura modular em camadas, mais precisamente em uma pilha de protocolos. Embora seja inquestionável a necessidade da separação de interesses (em camadas), essa organização modular, e o próprio arbítrio exercido pelo sistema operacional, são sérios impedimentos à obtenção do desempenho máximo[10].

Esta seção descreve soluções que permitem que aplicações sobrepassem os mecanismos de um sistema operacional convencional (ou seja, o sistema operacional é removido do caminho crítico de transferência de dados). Aplicações acessam a rede de maneira "nativa", com menor latência e sobrecarga de processamento por pacote. De forma geral, alocam uma *interface virtual* própria e a manipulam para enfileirar pacotes para transmissão ou obter pacotes recebidos. Vantagens usualmente associadas a essa abordagem são flexibilidade e capacidade de realizar otimizações específicas da aplicação.

Parte das abordagens de ligação de rede em nível de usuário foi desenvolvida para redes dedicadas (*system* ou *Storage Area Networks*, visando PAD ou armazenamento em rede, respectivamente). Em sistemas de propósito geral, há aspectos de segurança que devem ser tratados com cuidado, como por exemplo evitar que aplicações espiem mensagens de outras aplicações *(snooping)* ou gerem mensagens se passando por outra *(spoofing)*. Para lidar com essa questão, uma possibilidade é o estabelecimento de forma segura, pelo sistema operacional, de um "canal" entre duas aplicações remotas; a partir daí, o sistema precisa apenas garantir que a comunicação entre as duas ocorra através do canal estabelecido. Em redes orientadas a conexão, como ATM, canais podem ser mapeados em circuitos virtuais; já em redes com tecnologia Ethernet, por exemplo, é mais difícil. A seguir, são descritos diversos exemplos representativos de sistemas de comunicação que empregam sobrepasso do sistema operacional e suporte da NIC.

U-Net ([14, 45, 46]) foi a precursora em termos de arquitetura de comunicação em nível de usuário, oferecendo uma visão virtual da NIC para habilitar acesso de usuário

[10]Taxas de transmissão efetivas próximas à taxa bruta anunciada para o dispositivo.

a dispositivos de comunicação de alto desempenho. A aplicação cria um ou mais *end-points* virtuais, e então associa a cada um deles um segmento de comunicação assim como um conjunto de filas de envio e recepção. O envio de uma mensagem é feito da seguinte maneira: primeiro, a aplicação insere um descritor para uma mensagem na fila de transmissão; o *driver* continuamente faz um *poll* das filas e transmite as mensagens que encontra; após o envio, um flag no descritor é ligado de forma a indicar que o descritor pode ser reusado. A última versão do U-Net é a 2.1, de 1998, e suporta Myrinet, ATM e Fast Ethernet.

VIA - Virtual Interface Architecture ([16]), desenvolvida conjuntamente pela Compaq, Intel e Microsoft, foi a primeira iniciativa da indústria de especificar uma API de comunicação em nível de usuário para uso em agregados (infelizmente, a documentação sobre VIA é escassa, pois o projeto foi abandonado pelas companhias que o promoviam). O objetivo da arquitetura é obter uma API de baixo nível porém genérica (adequada ao uso por camadas de *Sockets* ou MPI), sem a sobrecarga advinda da gerência do sistema operacional, reduzindo a diferença de desempenho entre a vazão máxima (teoricamente alcançável pelo hardware) e aquelas que podem ser obtidas com software. [16] descreve uma implementação de VIA sobre SCI não coerente, e compara por simulação duas implementações de VIA. O projeto foi recentemente abandonado, não sendo mais promovido pelo consórcio de empresas; o *site web* que continha documentação[11] foi desativado e hoje[12] é ocupado por um *site* de material pornográfico.

Avalanche ([11]) é um conjunto de protocolos baseados em troca de mensagens e uma API de acesso a esses protocolos. Estes são auxiliados com processamento realizado por uma NIC não padrão; na comunicação, o sistema operacional é sobrepassado e mantém apenas as funções relativas à gerência de conexão. Os protocolos são baseados na política de reserva de *buffers* no receptor, que ocorre de antemão e permite que os dados sejam diretamente recebidos na área de memória do usuário (poupando uma cópia). Denominada *Widget*, a NIC é conectada no barramento de memória de cada estação, que são multiprocessadores HP. A NIC segue a tecnologia Myrinet, substituindo a tradicional NIC LaNai (conectada via barramento de E/S). O desempenho obtido em *benchmarks* indica latências de comunicação fim-a-fim (nível de usuário) inferiores a 4 microssegundos.

Arsenic ([36]) é uma NIC Gigabit Ethernet que exporta uma interface estendida para o sistema operacional e para aplicações de usuário. Parte do processamento tipicamente encontrado no sistema operacional é realizado pela NIC, diminuindo a demanda de CPU. Cada aplicação acessa a rede através de uma interface virtual, com suas próprias filas de envio e recepção. Arsenic permite a carga dinâmica de filtros de pacotes na NIC pelo sistema operacional; tais filtros são responsáveis na transmissão pela multiplexação de pacotes, preparação de cabeçalhos e verificações de segurança, e pela demultiplexação baseada em fluxos de aplicação e verificações de segurança na recepção. A carga dinâmica de filtros na NIC permite a demultiplexação por fluxo de pacotes na recepção, possibilitando utilizar estratégias específicas de interrupção de acordo com as características apresentadas por fluxos individuais. Arsenic foi implementado em uma NIC de prateleira, porém programável; um *driver* altamente modificado foi escrito para o *kernel* 2.3 do Linux.

EMP - Ethernet Message Passing ([44]) é uma camada de troca de mensagens com sobrepasso do sistema operacional e que propicia comunicação com zero cópias. Desen-

[11] Em `http://www.viarch.org`.
[12] Dezembro de 2002.

volvida para a família Ethernet, um protótipo foi implementado com base na programação de NICs Alteon Gigabit Ethernet. EMP oferece confiabilidade na transmissão de dados.

A abordagem em nível de usuário possui também limitações. Todo o processamento do protocolo é executado em nível de aplicação, sendo o tempo de processamento disponível ao protocolo, portanto, limitado pelo escalonador do sistema operacional ([46]). A migração de processamento para a NIC traz benefícios em termos de desempenho, porém obriga a aplicação a lidar com certa complexidade.

3.7 Comentários Finais

Este trabalho abordou um conjunto de mecanismos e tecnologias utilizados para promover a comunicação em ambientes que requerem alto desempenho. A Seção 3.2 listou um conjunto de aplicações que exigem, de alguma forma, alto desempenho, seja nas estações finais ou na própria rede. Na Seção 3.3 foram discutidos os principais modelos de comunicação entre processos: troca de mensagens, chamada remota de procedimento, invocação remota de método e memória compartilhada distribuída. Os principais aspectos de desempenho relacionados ao sistema operacional foram tratados na Seção 3.4. Na Seção 3.5 foram apresentadas tecnologias de interligação entre computadores, abordando tanto aquelas que são de propósito geral como as que são dedicadas ao processamento de alto desempenho. Por fim, na Seção 3.6, soluções que permitem que aplicações sobrepassem os mecanismos de um sistema operacional convencional foram abordadas.

Por uma questão de espaço (e conhecimento), este trabalho está longe de ser completo. Ao contrário, ele oferece uma visão geral do assunto redes (de alto desempenho) para processamento de alto desempenho. A busca por desempenho deve ocorrer em várias frentes; não poderia deixar de ser o caso com a parte de comunicação, que tem representado tanto em agregados como em grades um grande entrave na obtenção de sistemas paralelos com alto poder computacional.

A bibliografia apresentada, mesmo que discutida superficialmente, serve como base de referência para leitores interessados. Este é, sem dúvida, um campo de pesquisa que apresenta, ao mesmo tempo, grandes desafios científicos (na busca incessante por maior desempenho) e forte aplicabilidade no mercado (servidores, centros de dados, PAD), com claros benefícios para a sociedade.

3.8 Bibliografia

[1] 10 Gigabit Ethernet Alliance, “10 Gigabit Ethernet Technology Overview, 2002”, Disponível em `http://www.10gea.org/10GEA%20White%20Paper_0502.pdf`.

[2] G. Allen et alli, “Cactus Tools for Grid Applications”, Cluster Computing, v.4, n.3, pp.179-188, 2001.

[3] ANSI INCITS 337-2000, “Information Technology - Scheduled Transfer Protocol (ST)”, 2000.

[4] ANSI INCITS 343-2001, “Information Technology - Scheduled Transfer - Reliable Transport Profile (ST-RTP)”, 2001.

[5] Marinho P. Barcellos, "Programação Paralela e Distribuída em Java", ERAD 2002 - Escola Regional de Alto Desempenho, SBC, 2002. Cap. 6, pp.179-192, ISBN 85-88442-16-7.

[6] B. Bershad, T. Anderson, E. Lazowska and H. Levy, "Lightweight Remote Procedure Call", ACM Transactions on Computer Systems 8, 1 (February 1990), pp. 37–55.

[7] A. Birrel and G. Nelson, "Implementing Remote Procedure Calls", ACM Transactions on Computer Systems (TOCS), v.2, n.1, pp.39-59, 1984.

[8] N. Boden, D. Cohen, and R. Felderman, "Myrinet - a Gigabit per Second Local Area Network", IEEE Micro, p. 29-36, Feb. 1995.

[9] G. Ciaccio, "A Communication System for Efficient Parallel Processing on Clusters of Personal Computers", PhD Thesis, Università di Genova, June 1999. Disponível em `ftp://ftp.disi.unige.it/pub/project/gamma/gamma_thesis.ps.gz`.

[10] D.D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead", IEEE Communications Magazine, v.27, n.6, June 1989.

[11] A. Davis, M. Swanson, and M. Parker, "Efficient Communication Mechanisms for Cluster Based Parallel Computing", Proc. of the 1st International Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC'97), Feb. 1997.

[12] C. F. De Rose et alli, "The Scalable Coherent Interface (SCI) as an Alternative for Cluster Interconnection", Proc. of 13th Symposium in Computer Architecture and High Performance Computing (SBAC-PAD'2001). Pirenópolis, Sept. 2001. SBC, pp. 156-163.

[13] T. von Eicken et alli, "Active Messages: A Mechanism for Integrated Communication and Computation", Proc. of 19th Annual ISCA, 1992, pp. 256-266.

[14] T. von Eicken et alli, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing", Proc. of 15th ACM SOSP, Dec. 1995, pp. 40-53.

[15] P. Farrel, H. Ong, "Communication Performance over a Gigabit Ethernet Network", Proc. of 19th IEEE International Performance, Computing, and Communications Conference, pp. 181-189, 2000.

[16] K. Ghouas, K. Omang, and H. Bugge, "VIA over SCI - Consequences of a Zero Copy Implementation, and Comparison with VIA over Myrinet", Proc. of Communication Architectures for Clusters, CAC'01, San Fransisco, Apr 2001.

[17] D. B. Gustavson and Q. Li, "Low Latency, High-Bandwidth, and Low Cost for Local-Area Multi-Processor", Dept. of Computer Engineering, Santa Clara University, 1996.

[18] D. B. Gustavson and Q. Li, "The Scalable Coherent Interface (SCI)", IEEE Communications, p. 52-63, Aug. 96.

[19] K. Hwang, Z. Xu, "Scalable Parallel Computing - Technology, Architecture, Programming", USA: WCB/McGraw-Hill, 1998.

[20] M. Ibel, M. Schmitt, K. Schauser, A. Acharya, "Shared Memory vs Message Passing on SCI: A Case Study Using Split-C", Proc. of SCI: Scalable Coherent Interface - Architecture and Software for High-Performance Compute Clusters, Springer Verlag, LNCS, v. 1734, pp. 267-280, Berlin, 1999.

[21] InfiniBand Trade Association, `http://www.infinibandta.org`.

[22] InfiniBand Trade Association, "InfiniBand Architecture Relase 1.1, Vol. 1 (General Specifications)", 6 Nov. 2002. Disponível em `http://www.infinibandta.org`.

[23] Java Sun RMI Specification. Disponível em `ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf`.

[24] I. Kim, J. Moon, and H. Y. Yeom, "Timer-Based Interrupt Mitigation for High Performance Packet Processing", Proc. of 5th International Conference on High-Performance Computing in the Asia-Pacific Region, Sept. 2001.

[25] W. Kramer, "SCinet: Testbed for High-Performance Networked Applications", Computer, IEEE, June 2002, pp.47-55.

[26] M. Lobosco, C. Amorim and O. Loques, "Java for High-Performance Network-Based Computing: A Survey", Concurrency and Computation: Practice and Experience, Wiley, 36pp., v.114, n.1, 2002.

[27] J. Mache. "An Assessment of Gigabit Ethernet as Cluster Interconnect", Proc. of 1st IEEE International Workshop on Cluster Computing, p. 36-42, 1999.

[28] J. C. Mogul and K. K. Ramakrishnan, "Eliminating Receive Livelock in an Interrupt-driven Kernel", ACM Transactions on Computer Systems, v.15, n.3, pp.217-252, 1997.

[29] MPI Web Site. Disponível em `http://www-unix.mcs.anl.gov/mpi/`.

[30] Myricom, Inc, "Guide to Myrinet-2000 Switches and Switch Networks", 2001. Disponível em `http://www.myricom.com/myrinet/m3switch/guide/myrinet-2000_switch_guide.pdf`.

[31] National Center for Datamining, "The Terra Wide Data Mining Testbed (TWDM)". Disponível em `http://www.ncdm.uic.edu/testbeds.htm`.

[32] Nonstop Compaq Servers Web Site. Disponível em `http://nonstop.compaq.com`.

[33] F. A. D. Oliveira et alli, "A Comparative Study on Low-level APIs for Myrinet and SCI-based Clusters", 4th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, v.4, p.1-6, 2000.

[34] V. Paxson, "End-to-End Internet Packet Dynamics", IEEE/ACM Transactions on Networking v.7, n.3, 277-292, 1999.

[35] P. Pietikäinen, "Hardware Assisted Networking Using Scheduled Transfer Protocol on Linux", Master Thesis, University of Oulo, Finland, 2001, 78 p.

[36] I. Pratt and K. Fraser, "Arsenic: A User-Accessible Gigabit Ethernet Interface". Proc. of IEEE INFOCOM, April 2001.

[37] L. Prylli, B. Tourancheau, "Bip: A New Protocol designed for High Performance Networking on Myrinet", Lecture Notes in Computer Science, 1388:472-488, 1998.

[38] PVM Web Site. Disponível em `http://www.csm.ornl.gov/pvm/pvm_home.html`.

[39] D. Riddoch and S. Pope, "A Low Overhead Application/Device-driver Interface for User-level Networking", Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2001.

[40] R. Seifert, Gigabit Ethernet: Technology and Applications for High-speed LANs. Reading, Massachussets: Addison Wesley Longman, 1998.

[41] C. Seitz. Recent Advances in Cluster Networks. Disponível em http://www.cacr.caltech.edu/cluster2001/program/talks/seitz.pdf.

[42] K. Seymour et alli, "Overview of GridRPC: A Remote Procedure Call API for Grid Computing". GRID2002 - Workshop on Grid Computing. Disponível em `http://icl.cs.utk.edu/publications/`.

[43] J. Shalf and W. Bethel, "How the Grid Will Affect the Architecture of Future Visualization Systems," to appear in the Visualization Viewpoints column of the May/June 2003 issue of IEEE Computer Graphics and Applications.

[44] P. Shivam, P. Wyckoff, and D. Panda, "EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing". In Proceedings of SC2001.

[45] M. Welsh, A. Basu, T. von Eicken, "ATM and Fast Ethernet Network Interfaces for User-level Communication". Proc. of the Third International Symposium on High-Performance Computer Architecture, pp. 332-342, Feb. 1997.

[46] M. Welsh, D. Oppenheimer, and D. Culler, "U-Net/SLE: A Java-based User-Customizable Virtual Network Interface". Proc. of Java for High-Performance Network Computing workshop at EuroPar '98, Southampton, England, September 4, 1998.

[47] Workshop on Communication Architecture for Clusters (CAC) Web Site, disponível em `http://www.cis.ohio-state.edu/ panda/cac/`, evento em con-

junto com o International Parallel and Distributed Processing Symposium (IPDPS).

[48] RFC1832, "XDR: External Data Representation Standard". Disponível em `http://www.ietf.org/rfcs/rfc1832.html`.

[49] E. Yeh, H. Chao, V. Mannen, J. Gervais, B. Booth, "Introduction to TCP Offloading Engine", White Paper HP/QLogic/Adaptec/Alacritech/Intel, Version 1.0, April 2002.

Sumário