UFSM

Trabalho de Graduação

Adicionando suporte a Soquetes Brutos à biblioteca de comunicação MicroVAPI

Juliano Foletto Reckziegel

Curso de Ciência da Computação

Santa Maria, RS, Brasil

2006

Adicionando suporte a Soquetes Brutos à biblioteca de comunicação MicroVAPI

por

Juliano Foletto Reckziegel

Trabalho de Graduação apresentado ao Curso de Ciência da Computação – Bacharelado, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

Curso de Ciência da Computação

Trabalho de Graduação nº 206

Santa Maria, RS, Brasil

2006

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada, aprova o Trabalho de Graduação

Adicionando suporte a Soquetes Brutos à biblioteca de comunicação MicroVAPI

elaborado por

Juliano Foletto Reckziegel

como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Comissão Examinadora:

Prof. Dr. Benhur de Oliveira Stein
(Orientador)

Prof. Antonio Marcos de Oliveira Candia

Prof^a. Dr^a. Roseclea Duarte Medina

Santa Maria, 6 de janeiro de 2006.

Agradecimentos

Agradeço inicialmente aos meus pais, Marino e Ilsa, e a minha mana Michele, por sempre terem me dado muito amor, por me apoiarem e me orientarem nas escolhas que devemos fazer e por me proporcionarem os meios que me levaram a essa conquista.

Agradeço aos meus amigos que foram um apoio, ajudando em horas difíceis, sempre me incentivando a persistir nas tarefas complicadas. Não podendo deixar de citar os meus grandes amigos João, Pídio, Diogo, Brother, e os meus colegas Rafael, Rubens e Cristóvão, que sempre foram os companheiros de noites não dormidas, tanto para fazer trabalhos intermináveis, como para fazer muita festa e muito agito. Sem esquecer este último, o Cristóvão, que infelizmente já se foi. Cara tu foste show, fico muito feliz por tu teres me dado a oportunidade de te conhecer, com certeza tu estarás comigo para sempre, meu grande amigo dos bailinhos.

Agradeço a todos os professores, em especial ao professor Benhur Stein por ter me orientado neste trabalho de uma forma tranqüila, com sua calma e dedicação, sempre disposto a ajudar. Sem esquecer de falar do professor Pasin, que antes de se ausentar para o seu pós-doutorado me auxiliava e orientava. Agradeço também ao Rodrigo Righi que me ajudou nas tarefas difíceis, em horas que pareciam que os programas nunca iriam funcionar, também auxiliando-me na elaboração de artigos.

Agradeço a Fani e a Angela por me auxiliarem na elaboração do texto, em dúvidas de português e, principalmente, por se disponibilizarem e me ajudar.

Agradeço, por fim, a uma pessoa muito especial, minha namorada Roberta, pelos momentos felizes, pelos beijos agradáveis, pelos sorrisos verdadeiros, pelo carinho, por me fazer feliz, por um relacionamento de verdade, e principalmente, por um amor que eu jamais vivi.

Conteúdo

Lista de Figuras

RESUMO

Trabalho de Graduação
Ciência da Computação
Universidade Federal de Santa Maria

Adicionando suporte a Soquetes Brutos à biblioteca de comunicação MicroVAPI

Autor: Juliano Foletto Reckziegel

Orientador: Prof. Dr. Benhur de Oliveira Stein

Data e Local da Defesa: Santa Maria, 6 de janeiro de 2006.

Os aglomerados de computadores [BUY 99] têm se tornado uma alternativa viável e de baixo custo para aplicações que necessitem de alto poder computacional. Como eles são compostos de nós individuais e interligados por uma ou mais redes [WIL 99], é necessário trocar mensagens entre as aplicações que querem utilizar o seu paralelismo.

Como o tempo de comunicação entre os nós é mais oneroso que o de computação [WIL 99], existem diversas pesquisas em tecnologia de redes para interligação dos nós que compõem o aglomerado, onde se destaca a tecnologia InfiniBand [IBT 2002]. Porém essa arquitetura ainda é muito recente, o que faz com que ela seja muito cara para uma utilização maciça em aglomerados.

Para que se possa utilizar programas escritos para operar sobre InfiniBand em aglomerados que não possuem essa tecnologia, foi desenvolvida a biblioteca de comunicação MicroVAPI. Essa biblioteca utiliza o DECK [BAR 98] como ambiente de comunicação para implementar as chamada InfiniBand.

Este trabalho descreve a implementação de uma segunda versão para a MicroVAPI [REC 2005], juntamente com os resultados dos testes realizados com as duas versões. Essa nova versão utiliza soquetes brutos, ao invés de trabalhar com o DECK. Os testes realizados demonstraram um bom ganho de desempenho da versão que utiliza os soquetes brutos em relação à que utiliza o DECK.

Capítulo 1

Introdução

Os aglomerados de computadores (*clusters of workstations*) são uma alternativa aos supercomputadores. Eles se disseminaram rapidamente devido ao seu baixo custo, alta flexibilidade e escalabilidade [BUY 99]. Os aglomerados são compostos de nós individuais de processamento com módulos privativos de memória inter-conectados por uma ou mais redes, geralmente de alto desempenho [WIL 99].

Pelo fato da memória não ser compartilhada nos aglomerados, para que os programas executem com os dados distribuídos, faz-se necessária a troca de mensagens entre os nós envolvidos. Essa troca de mensagens garantirá o compartilhamento dos dados. Para implementar um sistema com troca de mensagens faz-se uso de bibliotecas de comunicação.

Como o tempo de comunicação entre os nós é mais oneroso que o de computação [WIL 99], existem diversas pesquisas em tecnologia de redes para interligação dos nós que compõem o aglomerado, onde se destaca a tecnologia InfiniBand [IBT 2002]. Para se utilizar essa rede, deve-se fazer uso da biblioteca VAPI [MEL 2000]. Essa biblioteca implementa todas as funcionalidades e estrutura de dados apresentadas pela especificação InfiniBand. Porém, essa arquitetura ainda é muito recente, o que faz com que ela seja muito cara para uma utilização maciça em aglomerados.

Para que os programas escritos com a biblioteca VAPI possam ser usados em aglomerados, onde a tecnologia InfiniBand não está disponível, foi desenvolvida a MicroVAPI [REC 2005], que é uma biblioteca de adaptação que porta as chamadas VAPI [MEL 2000] para DECK [BAR 98]. Entretanto, quando se está utilizando a

MicroVAPI sobre Ethernet, o desempenho deixa a desejar, pois se acaba fazendo uso do protocolo TCP, juntamente com sua sobrecarga, o que aumenta a latência e diminui a largura de banda. Para evitar essa sobrecarga existem os soquetes brutos (*Raw Sockets*) que fazem uma comunicação de baixo nível, dispensando várias etapas que são necessárias quando se utiliza o protocolo TCP.

Este trabalho propõe-se a integrar à MicroVAPI a capacidade de utilizar os soquetes Brutos, como alternativa de comunicação sobre redes Ethernet, dispensando a utilização do DECK. Assim, é possível diminuir a sobrecarga no sistema e possibilitar um maior desempenho das aplicações escritas com a *interface* VAPI que queiram utilizar a rede Ethernet.

Este trabalho está dividido em capítulos. Primeiramente, é apresentada uma revisão de literatura. Logo após, situa-se o capítulo de implementação, que descreve as questões de projeto e a reimplementação da biblioteca MicroVAPI. Em seguida encontra-se o capítulo referente à análise de resultados e testes da biblioteca. Para finalizar, é apresentado o capítulo de conclusões, que finaliza o trabalho, reunindo as principais idéias e os resultados obtidos.

Capítulo 2

Revisão de Literatura

Na revisão de literatura foram introduzidos conceitos sobre paralelismo, também foi falado sobre algumas tecnologias que foram utilizadas na elaboração do trabalho, juntamente com a descrição de seus funcionamentos.

2.1 Paralelismo

Um programa concorrente [OLI 2001] é aquele que é executado simultaneamente por dois ou mais processos que cooperam entre si. A execução concorrente ou paralela dos sub-programas pode gerar um ganho de desempenho expressivo, quando executados em máquinas com mais de um processador, ou em computadores distintos, fazendo com que o programa paralelo, muitas vezes, seja mais rápido que o seqüencial.

Uma máquina paralela é um conjunto de processadores capazes de trabalhar cooperativamente para resolver um problema computacional. O usuário de um sistema como esse possui uma visão única, ou seja, para ele é como se os vários processadores fossem um só. Existem dois tipos de máquinas paralelas: os multiprocessadores e os multicomputadores.

Nos multiprocessadores (figura 2.1), a memória, normalmente, é compartilhada entre todos os processadores. O acesso é feito através de um barramento que os interliga. Para que um programa seja executado em um multiprocessador, e tire proveito dessa arquitetura, ele deve ser dividido em diversos fluxos de execução

(*threads*), dessa forma, cada fluxo é executado concorrentemente.

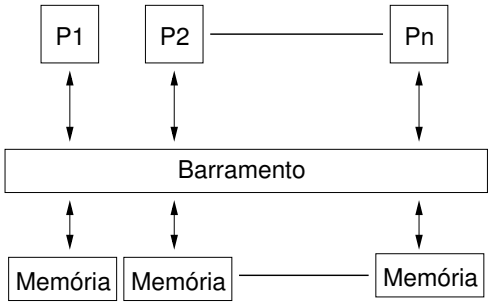

Figura 2.1: Arquitetura de um Multiprocessador

Já um multicomputador (figura 2.2) é composto por dois ou mais computadores, também chamados de nós, que possuem bancos de memórias individuais e são inter-conectados por uma ou mais redes, geralmente de alto desempenho [WIL 99]. Ou seja, como os processos estão em máquinas distintas, para que haja cooperação entre eles é necessária uma troca de mensagem através da rede.

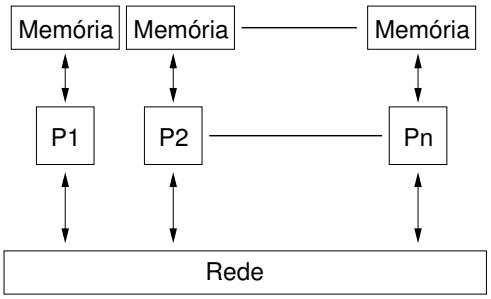

Figura 2.2: Arquitetura de um Multicomputador

A arquitetura que vem sendo mais utilizada é a dos multicomputadores, também chamados de aglomerados de computadores (*clusters of workstations*), pois para um mesmo poder de processamento, os aglomerados possuem um custo inferior aos multiprocessadores, sem falar na escalabilidade, pois se for preciso de mais poder computacional, apenas deve-se adicionar mais computadores e interligá-los. Outro fator importante é que, quando um dos nós estraga o resto do sistema continua trabalhando, o que nem sempre acontece quando isso ocorre com um dos processadores de um multiprocessador.

2.2 Protocolo TCP

O TCP (*Transmission Control Protocol*) é um protocolo de transporte especificamente projetado para oferecer um fluxo de dados ponto a ponto, confiável em uma inter-rede não-confiável [TAN 97]. Uma inter-rede pode ser composta de várias arquiteturas de redes com diferentes topologias, larguras de banda, MTU (*Maximum Transmission Unit*) distintos, entre outros parâmetros.

O TCP foi planejado para funcionar através de diferentes redes de uma forma transparente, assumindo que a comunicação é entre pontos autônomos e, por isso, possui a mínima cooperação possível [ROD 97], enquanto que quando se está trabalhando em um aglomerado, normalmente se possui alguns sistemas de gerenciamento comuns, sistema de arquivos compartilhados, e, normalmente, se tem uma rede confiável, sem diferenças na topologia de rede, o que faz com que o protocolo TCP não seja uma boa escolha para ser utilizado em aglomerados

A implementação padrão do TCP divide o protocolo em uma pilha de camadas (figura 2.3). Essa forma de comunicação permite várias combinações de protocolos. Entretanto, quando se quer algo mais específico, fica-se preso a esta estrutura de comunicação entre camadas, o que gera cópias de memória e consome poder computacional, que poderia estar sendo empregado em outras tarefas.

Figura 2.3: Camadas do protocolo TCP

Quando se está utilizando o TCP para fazer a comunicação entre as máquinas de um aglomerado de computadores, acaba-se fazendo uso de todos esses recursos que o TCP possui e que não seriam necessários em um ambiente como esse. Fazendo com que as aplicações percam desempenho quando estão utilizando a rede.

2.3 InfiniBand

A arquitetura InfiniBand (IBA - *InfiniBand Architecture* [IBT 2002]) especifica uma infra-estrutura para ligação de alto desempenho dentro de estações de trabalho e também entre estações (por exemplo, substituindo as tecnologias PCI (*Peripheral Component Interconnect*) e Ethernet). Esta tecnologia é apoiada pelos fabricantes Intel, IBM, Sun, Hewlett Packard e Microsoft. Sua principal finalidade é a interconexão de servidores de entrada/saída (E/S), especialmente servidores de armazenamento, e estações de trabalho em uma rede de sistema. Ela pode agrupar desde um pequeno servidor com um processador e poucos dispositivos de entrada e saída até um supercomputador com centenas de processadores e dispositivos de E/S.

Em computadores convencionais, tipo PC (*Personal Computers*), um dos principais fatores limitantes no desempenho é o subsistema de entrada/saída. A principal proposta da arquitetura InfiniBand é substituir este barramento local, por uma estrutura de comunicação chaveada (*switched communications fabric*), permitindo que muitos dispositivos realizem comunicação concorrente com alta largura de banda e baixa latência. A rede de sistema IBA fornece para seus clientes de E/S e nós processadores uma transferência de mensagens sem cópias (*zero-copy*) intermediárias nos nós envolvidos na comunicação. Essa transmissão acontece sem o envolvimento do núcleo do sistema operacional e utiliza o *hardware* para fornecer alta confiabilidade e tolerância às falhas na comunicação.

A tecnologia InfiniBand implementa diretamente em *hardware* o protocolo de transporte e possibilita o acesso direto à memória remota. Para a troca de mensagens (*send/receive*), a InfiniBand utiliza a abstração de par de filas (*Queue Pais* - QP). Um par de filas atua como um tratador de comunicação que o programador utiliza para se comunicar com outro nó.

Uma das bibliotecas de comunicação desenvolvidas para trabalhar com InfiniBand é a VAPI (*Verbs Application Program Interface*) [MEL 2000], desenvolvida pela empresa Mellanox que oferece uma interface de baixo nível para a escrita de apli-

cações que utilizam equipamentos com essa tecnologia. A VAPI implementa todas as funcionalidades especificadas pela arquitetura InfiniBand. Ela pode ser utilizada diretamente para a construção de aplicações de usuário ou como uma plataforma de comunicação para interfaces de mais alto nível.

A arquitetura InfiniBand ainda é recente e envolveu muitas pesquisas de diversas empresas para ser desenvolvida completamente. Por este motivo, essa tecnologia possui um custo muito elevado, o que faz com que seja dificultada a sua utilização maciça em aglomerados.

2.4 VAPI

A VAPI [MEL 2000] é uma biblioteca de comunicação que oferece uma interface de baixo nível para a escrita de aplicações que utilizam equipamentos de rede InfiniBand. A VAPI foi desenvolvida pela empresa Mellanox e implementa todo o conjunto de funcionalidades apresentados pela especificação InfiniBand. O guia de referência da VAPI apresenta 88 interfaces de função e todas as estruturas de dados que elas recebem como parâmetros de entrada ou de saída. Ela pode ser utilizada diretamente para a construção de aplicações de usuário ou como uma plataforma de comunicação para interfaces de mais alto nível.

A especificação InfiniBand define uma arquitetura complexa e com muitos detalhes. A título de exemplo, esses detalhes podem compreender o estabelecimento de um domínio de proteção para a aplicação, chaves de memória para possibilitar acesso remoto, registro de memória, criação de descritores para a troca de mensagens, entre outros. Como pode ser observado, a VAPI herda algumas complexidades da tecnologia InfiniBand e implementa um conjunto de funções que trabalham com estruturas de dados com vários campos que referenciam propriedades de comunicação de baixo nível. Ao mesmo tempo, esse detalhismo da VAPI possibilita ao programador ajustar a sua aplicação, visando diretamente as questões de hardware, proporcionando um melhor desempenho para a sua execução. Por exemplo, o programador pode explorar a qualidade de serviço proporcionada diretamente pelos adaptadores de rede

InfiniBand. Isso se faz simplesmente completando um dos campo da estrutura de dados VAPI que define as características de uma conexão entre dois computadores.

InfiniBand utiliza a abstração de par de filas (QP) para realizar a troca de mensagens. Um par de filas atua como um tratador de comunicação que o programador utiliza para se comunicar com outro nó InfiniBand. Para trabalhar com a VAPI é necessário que pelo menos um membro da subrede execute o programa responsável por gerenciá-la. Este programa gera identificadores para cada um dos componentes da rede e espalha a tabela de chaveamento entre eles para que seja possível o tráfego de informações.

A relação das principais funções da VAPI está discriminada na tabela 2.1. Como apresentado anteriormente, a InfiniBand, assim como a VAPI, usa a abstração de par de filas para realizar a comunicação entre dois processos. A primitiva VAPI_create_qp() é responsável por criar um par de filas (QP). Um dos atributos para a sua criação é a fila de conclusão associada às filas de envio e de recepção. Ambas as filas de uma QP podem ser associadas a uma única fila de conclusão, assim como cada uma delas por ser mapeada para uma fila de conclusão particular.

Tabela 2.1: Principais primitivas de comunicação

Primitiva	Funcionalidade
VAPI_create_qp()	Criar um par de filas (QP)
VAPI_create_cq()	Criar uma fila de conclusão de requisições de troca de mensagem
VAPI_modify_qp()	Modificar as propriedades de uma QP
VAPI_register_mr()	Registrar uma região de memória para a troca de mensagem
VAPI_post_sr()	Colocar uma requisição de transferência de dados na fila de envio de uma QP
VAPI_post_rr()	Colocar uma requisição de recebimento de dados na fila de recepção de uma QP
VAPI_poll_cq()	Verificar em uma fila de conclusão o término de uma troca de mensagem
VAPI_poll_cq_block()	Bloquear até a conclusão de uma troca de mensagens

A primitiva VAPI_modify_qp() altera o estado de uma QP. É através dela que duas QPs realizam o processo de conexão. Os principais campos presentes na estrutura que contêm os atributos de conexão são o número da QP remota, o identificador do adaptador de canal remoto, a MTU a ser utilizada para a transferência de mensagens e a qualidade de serviço desejada. Para um processo conhecer a QP e o identificador do adaptador de canal remoto é necessário algum outro tipo de interação explícito para a troca dessas informações. Por exemplo, tais informações podem ser passadas utilizando outra rede, um arquivo compartilhado ou em linha de comando. Após a chamada dessa primitiva, uma QP está pronta para receber e enviar dados de/para uma outra remota.

A região de memória utilizada para a troca de mensagens sobre uma rede InfiniBand deve necessariamente ser registrada. Tal processo é realizado através da primitiva VAPI_register_mr(). Após o registro de memória, pode-se proceder com as operações de troca de mensagens. Para tal, primeiramente deve-se criar uma estrutura de requisição, chamada descritor. Ele possui campos que descrevem a troca de mensagem, como a quantidade de dados envolvida e um ponteiro para a região de memória onde estão ou devem ser colocados os dados. As primitivas VAPI_post_sr() e VAPI_post_rr() efetivam a troca de mensagem. Os descritores VAPI são processados seqüencialmente, ou seja, o primeiro a ser postado na fila de uma QP é o primeiro a ser processado.

As primitivas VAPI_poll_cq() e VAPI_poll_cq_block() verificam a conclusão do processamento de um descritor. Ambas primitivas verificam a primeira entrada de uma fila de conclusão que é passada como um dos seus parâmetros de entrada. Como mencionado anteriormente, cada fila de uma QP está ligada a outra de conclusão e, quando um descritor em uma das filas da QP é completamente processado, é lançada uma entrada na fila de conclusão na qual ela está associada. A primitiva VAPI_poll_cq() é não bloqueante, ou seja, se não houver nenhuma entrada na fila de conclusão dela, automaticamente, retorna para o fluxo de execução do programa, enquanto a VAPI_poll_cq_block() faz a mesma coisa, mas tem um caráter bloqueante, esperando até que uma nova entrada na fila de conclusão seja adicio-

nada. Caso uma dessas primitivas encontre uma entrada numa fila de conclusão, ela completa um descritor de saída que informa a quantidade de bytes transferidos. Após a conclusão da troca de mensagens identificada por um descritor, o processo receptor pode verificar automaticamente os dados na região de memória registrada disponibilizada para a recepção. Da mesma forma, um processo transmissor pode reutilizar ou destruir os dados que foram enviados.

2.5 Biblioteca MicroVAPI

A MicroVAPI é uma biblioteca de adaptação que, através do DECK (*Distributed Execution and Communication Kernel*) [BAR 98], possibilita que os programas escritos para operar sobre InfiniBand, utilizando a biblioteca VAPI sejam capazes de trabalhar sobre redes SCI, VIA, Myrinet e Ethernet. Para que isso funcione, as 17 funções mais utilizadas para fazer o envio e recepção de mensagens são reimplementadas, portando as chamadas VAPI para DECK.

A comunicação entre dois computadores com DECK é realizada através da abstração de caixas de mensagens (*Mail Box - MB*). Uma caixa de mensagens representa um ponto final de comunicação que pode receber ou enviar mensagens. A biblioteca DECK fornece uma interface comum de programação para trabalhar com diferentes protocolos de comunicação, podendo ser utilizada para prover trocas de mensagens sobre equipamentos de rede, voltados para alto desempenho. Essa característica representa uma das suas grandes vantagens, a portabilidade. Ela também pode ser utilizada para a construção de ferramentas para a integração de aglomerados, visto que suporta múltiplos ambientes de comunicação. Por essa razão, DECK é o ambiente de execução escolhido para desenvolver o modelo MultiCluster [BAR 2000], que é um projeto que visa a criação de um ambiente de execução capaz de operar transparentemente sobre aglomerados de computadores que possuam diferentes tecnologias de redes e/ou diferentes arquitetura de computadores.

Como as interfaces de chamada DECK e VAPI são ligeiramente diferentes, a MicroVAPI possui funções que fazem a adaptação de uma para outra. Algumas

dessas funções nem chegam a chamar a biblioteca DECK, como, por exemplo, as de registro de memória. Por outro lado, a inicialização do DECK se baseia em um *script* de lançamento (deckrun), que foi substituído por funções da MicroVAPI.

As bibliotecas VAPI e DECK utilizam paradigmas diferentes para descrever uma troca de mensagem, a MicroVAPI possui algumas adaptações nesse ponto. No DECK é utilizada a abstração de caixa de mensagens (*Mail Box* - MB). Cada caixa, em um dado instante, pode somente receber ou enviar mensagens.

Para uma caixa de mensagens poder receber dados, ela deve ser criada, já para enviar, ela deve clonar a caixa de mensagens que o nó receptor criou. Todavia, InfiniBand utiliza um único ponto final de conexão, chamado par de filas (QP). Através de uma única QP, pode-se enviar e receber mensagens. A MicroVAPI redefine a estrutura de uma QP como sendo composta de duas caixas de mensagens. A figura 2.4 apresenta essa organização, com o encapsulamento das caixas de mensagens dentro de uma QP.

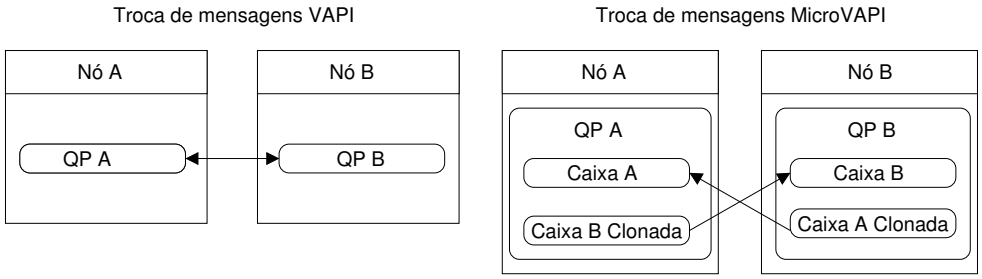

Figura 2.4: Caixa de Mensagens na MicroVAPI

Para enviar mensagens através da VAPI, é necessária a criação de um descritor de requisição. Esse descritor é uma estrutura que possui dois campos principais: um ponteiro para a região onde estão ou devem ser colocados os dados e outro para a quantidade de dados a serem enviados ou recebidos. Com a utilização de descritores VAPI, no momento da chegada dos dados, o receptor sabe a quantidade enviada pelo outro ponto final. Entretanto, o ambiente DECK não possibilita que o receptor saiba a quantidade de dados enviados. Para resolver isso, é empacotado na própria mensagem DECK o tamanho dos dados e logo após, são empacotadas as informações. Da mesma forma, no lado receptor, é desempacotado o tamanho e logo após os dados

efetivamente.

O DECK não possui comunicação assíncrona, já a VAPI através dos pares de filas utiliza isso transparentemente. Isso faz com que não se possa fazer chamadas diretas de uma para a outra, pois se isso fosse feito transformar-se-iam as chamadas que eram assíncronas na VAPI em síncronas, o que faria com que os programadores não pudessem utilizar as facilidades oferecidas pelos pares de filas.

A adaptação realizada para resolver o problema do assincronismo consiste em criar para cada canal de transmissão ou recepção de dados um novo fluxo concorrente de execução, aqui chamado de demônio (*daemon*), para processar as requisições de comunicação. Essa implementação de comunicação assíncrona é descrita através de três componentes básicos: um produtor, outro consumidor e uma estrutura de dados compartilhada (fila de requisições). A organização desses três componentes pode ser visualizada na figura 2.5.

Na figura 2.5, o produtor representa a função da MicroVAPI encarregada de realizar a comunicação assíncrona e atua como o fluxo corrente da aplicação. Ele adiciona no fim da fila de requisição de comunicação mais uma entrada a ser processada e retorna logo após essa tarefa. O componente consumidor é um demônio que representa um fluxo concorrente de execução ao fluxo principal da aplicação. Ele captura a primeira requisição ainda não processada na fila compartilhada e efetua a comunicação pela rede de interconexão segundo a ordem que tem em mãos. Por fim, a fila de requisições é uma estrutura compartilhada por ambos fluxos de execução e representa uma região crítica. Isso porque nessa região podem haver problemas de condição de execução para a sua atualização. Para manter a integridade dessa estrutura e a estabilidade da MicroVAPI são utilizadas estruturas de exclusão mútua e variáveis de condição, de modo a garantir um acesso controlado e confiável a memória compartilhada. As variáveis de exclusão mútua servem para serializar o acesso à região crítica. Já as variáveis de condição são empregadas para notificar o demônio de uma nova requisição na fila compartilhada. Aliada a essa funcionalidade, elas também são usadas para avisar a função da MicroVAPI que espera pelo término das requisições de comunicação que já existe algum descritor que já foi totalmente

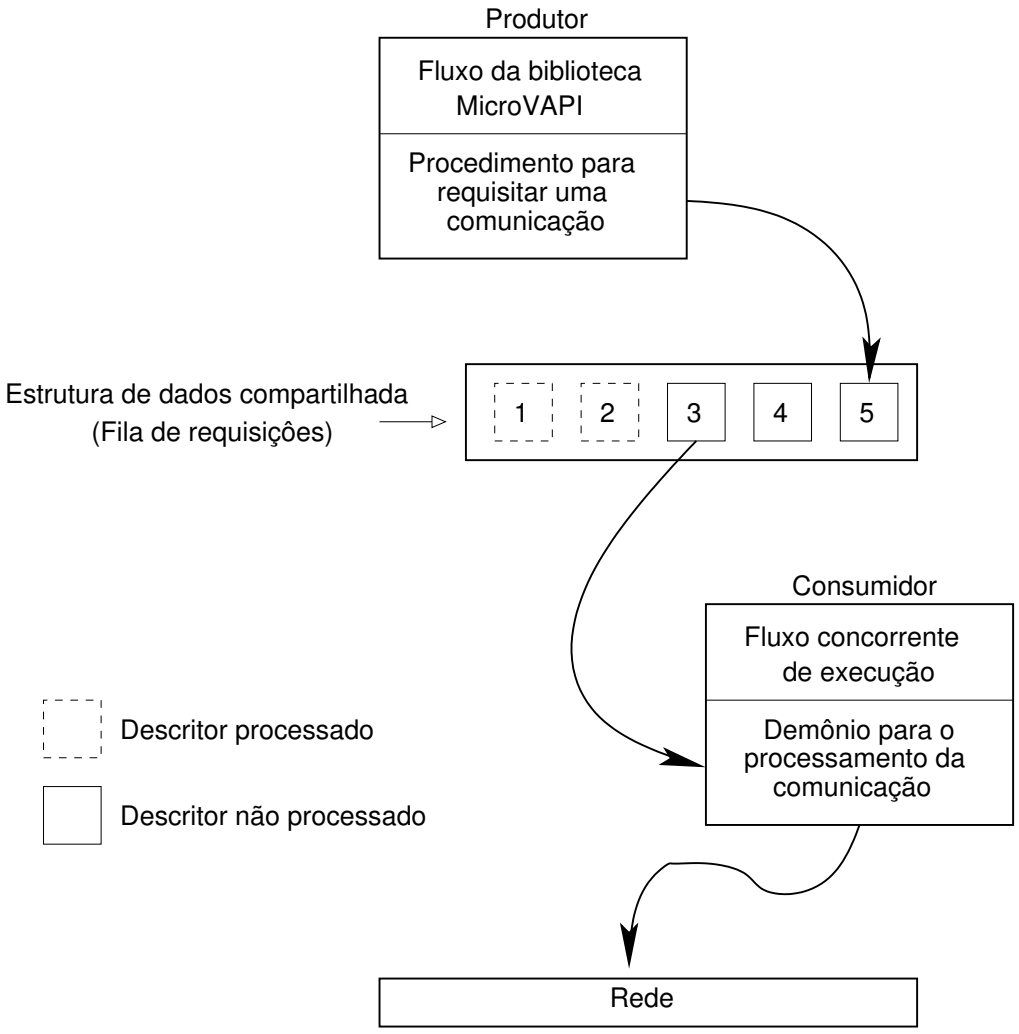

Figura 2.5: Adaptação para comunicação assíncrona

processado.

Quando se está utilizando a MicroVAPI sobre Ethernet através do DECK, acaba-se utilizando o protocolo TCP, que possui uma latência grande, juntamente com a sua sobrecarga, pois envia muitos dados e executa somas de verificação (*check-suns*) para garantir a confiabilidade dos pacotes. Além disso, o protocolo possui tempos limites (*timeouts*) grandes para fazer o controle de recebimento dos pacotes enviados, o que em uma rede local não é necessário. Em função das características do TCP recém citadas, o DECK sobre Ethernet não possui um bom desempenho.

2.6 Soquetes brutos

Os soquetes brutos (*raw sockets*) são capazes de fornecer um caminho rápido entre as aplicações e a rede, pois eles oferecem três recursos não fornecidos pelos soquetes TCP e UDP normais [STE 2005], além de trabalharem quase que diretamente na camada de enlace de dados:

- Permitem ler e gravar pacotes ICMPv4, IGMPv4 e ICMPv6. O programa *ping*, por exemplo, envia solicitações de eco ICMP e recebe respostas de eco ICMP. O *daemon* de roteamento de multicast, *mrouted*, envia e recebe pacotes IGMPv4.

  Essa capacidade também possibilita que aplicações construídas, utilizando ICMP ou IGMP, sejam tratadas inteiramente como processos de usuários, em vez de colocar mais código no *Kernel*. O *daemon* de descoberta de roteador, por exemplo, é construído dessa maneira. Ele processa duas mensagens ICMP (anúncio e solicitação de roteador) sobre as quais o *Kernel* nada sabe.

- Com um soquete bruto, um processo pode ler e gravar datagramas IPv4 com um campo de protocolo IPv4 que não é processado pelo *Kernel*. A maioria dos *Kernels* processa somente datagramas, contendo os valores 1 (ICMP), 2 (IGMP), 6 (TCP) e 17 (UDP). Mas muitos outros valores são definidos para o campo de protocolo: o registro "Número de Protocolo"do IANA lista todos os valores. Por exemplo, o protocolo de roteamento OSPF não usa TCP nem UDP, mas IP diretamente, configurando o campo de protocolo do datagrama IP como 89. O programa *gated*, que implementa OSPF, deve utilizar um soquete bruto para ler e gravar esses datagramas IP, pois eles contêm um campo de protocolo, sobre o qual o *Kernel* nada sabe. Essa capacidade também persiste no IPv6.

- Com um soquete bruto, um processo pode construir seu próprio cabeçalho IPv4, utilizando a opção de soquete *IP_HDRINCL*. Isso pode ser utilizado,

por exemplo, para construir pacotes UDP e TCP, sem ter que fazer chamadas ao *Kernel*.

Quando se está utilizando os soquetes brutos, deve-se possuir permissão de superusuário do sistema para se ter um acesso de tão baixo nível, isso se deve ao fato de que se fosse possível um usuário normal ter um acesso de tão baixo nível, se estaria abrindo uma brecha na segurança do sistema. Entretanto, como se está utilizado um ambiente fechado que muitas vezes possui uma rede dedicada, isso não se torna um fator crítico, mas que deve ser considerado antes de ser utilizado.

Se forem utilizados pacotes brutos maiores que a MTU (*Maximum Transmission Unit*) da NIC (*Network Interface Card*) o *Kernel* os fragmenta e envia automaticamente. Da mesma forma, se um datagrama fragmentado é recebido, nada é passado para um soquete bruto até que todos os pedaços tenham chegado e estejam montados.

Através do uso dos recursos fornecidos pelos soquetes brutos tem-se uma redução na latência e um aumento na banda passante. Isso proporciona um desempenho melhor nos aglomerados comparado a quando se utiliza o DECK sobre Ethernet como alternativa para comunicação.

Capítulo 3

Implementação

Este capítulo descreve a implementação das funções que foram usadas para que a biblioteca MicroVAPI pudesse utilizar soquetes brutos como alternativa de comunicação, juntamente com as modificações feitas na estrutura geral da biblioteca, além das estratégias necessárias que foram utilizadas para que fosse possível sua concretização. A MicroVAPI é uma biblioteca de comunicação entre nós de aglomerados, desenvolvida em linguagem de programação ANSI C [KER 89].

Este trabalho utilizou o sistema operacional LINUX, juntamente com sua interface de soquetes (*sockets*) para realizar a comunicação na MicroVAPI, que, na versão anterior, fazia uso da biblioteca DECK. Essa nova implementação utiliza mais especificamente soquetes brutos que dão ao programador um maior grau de controle das informações que são transferidas pela rede. Como, por exemplo, a possibilidade de o programador não apenas escrever os dados que quer enviar, mas também o cabeçalho IP que o pacote possuirá.

3.1 Visão geral da MicroVAPI

Para a elaboração deste trabalho foi criada uma mini biblioteca, que compreende três funções, uma responsável pela criação do soquete bruto, outra responsável pelo envio dos dados, utilizando soquetes brutos e, a última, responsável por receber os dados com soquetes brutos. Para adicionar suporte a soquetes brutos à MicroVAPI, as funções desta, responsáveis pela transmissão dos dados, passaram a

chamar as funções criadas nessa mini biblioteca.

A organização, que antes compreendia a aplicação do usuário, passando para a biblioteca MicroVAPI, esta passando para o DECK e este utilizando o protocolo TCP para acessar a rede, como pode ser visualizado na figura 3.1, passou a compreender a aplicação do usuário, passando para a biblioteca MicroVAPI e esta passando para a mini biblioteca, que é responsável por acessar a rede através dos soquetes brutos, ou seja, transmitir os dados.

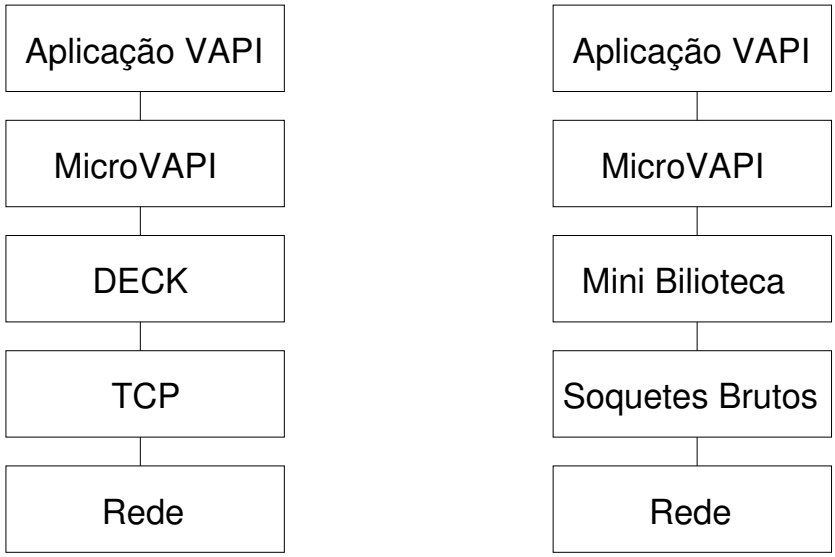

Figura 3.1: Organização da MicroVAPI com DECK e Soquetes Brutos

Para a concretização deste trabalho não foram feitas modificações no mecanismo que cria a comunicação assíncrona na MicroVAPI, que está localizado no bloco denominado MicroVAPI da figura 3.1. Nas funções da mini biblioteca foi implementado um protocolo que é responsável pela transmissão de dados que sejam maiores que 65535 *bytes*, que é o tamanho máximo de um datagrama.

3.2 Protocolo

Para se fazer a transferência de dados o protocolo IP permite a criação de datagramas de até 65535 *bytes* (incluindo os dados do cabeçalho IP) [TOR 2001]. Com isso, foi necessária a criação de um protocolo simples para possibilitar a transmissão

de uma quantidade maior de dados.

O protocolo criado, primeiramente, envia uma mensagem para o receptor, dizendo qual é a quantidade de dados que serão enviados e o tamanho máximo dos pacotes Logo após, são transferidos, em ordem reversa, os pacotes com os dados, e espera-se uma resposta de confirmação do receptor.

Quando são recebidos dados de um soquete bruto, sempre é retornado o cabeçalho IP, junto com os dados recebidos. Por isso, ao se receber mais de um pacote, não se pode apenas receber o primeiro pacote de dados, e logo após, passar o endereço seguinte, pois se criaria uma seção de informações IP, outra de dados recebidos, mais uma de informações IP, mais uma de dados recebidos, e assim sucessivamente, como mostra a figura 3.2, o que obrigaria cópias de memória, para reunir os dados em sua totalidade.

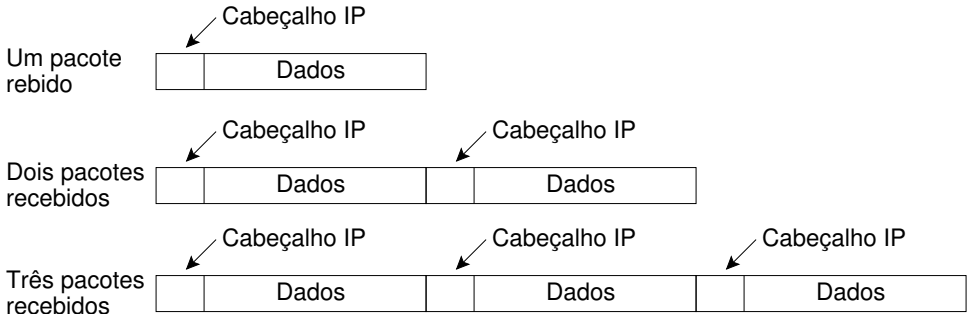

Figura 3.2: Recepção dos pacotes em ordem

Para se evitar cópias de dados na recepção, os pacotes de dados são enviados em ordem inversa. A transmissão é feita dessa forma, pois assim o pacote seguinte sempre sobrescreve os dados do cabeçalho IP do pacote anterior, como demonstrado na figura 3.3, obtendo, no final, apenas um cabeçalho IP e os dados na ordem correta.

Por exemplo, para transferir 12000 *bytes*, com datagramas de tamanho máximo igual a 5020 *bytes* de um nó A para um nó B, inicialmente deve-se deixar o nó B esperando uma mensagem contendo a quantidade de dados e o tamanho máximo dos datagramas, após ele receber essa mensagem ele passaria a esperar os dados. No nó A deve-se enviar para B uma mensagem contendo os números inteiros 12000 e 5020, e logo após deve-se enviar uma mensagem contendo os 2000 *bytes* finais mais

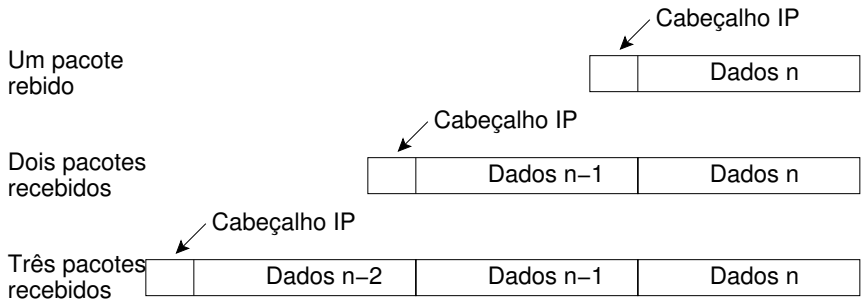

Figura 3.3: Recepção dos pacotes em ordem inversa

os 20 bytes do cabeçalho IP, totalizando 2020 *bytes*, a seguir são enviados os 5000 *bytes* intermediários mais os 20 bytes do cabeçalho IP, totalizando 5020, e, enfim, são enviados os 5000 *bytes* iniciais mais os 20 bytes do cabeçalho IP, totalizando 5020, ficando-se a espera de uma mensagem do nó B de transferência efetuada com sucesso. Após o nó B receber os dados ele retornaria uma mensagem de transferência efetuada com sucesso. Esse exemplo pode ser visualizado na figura 3.4.

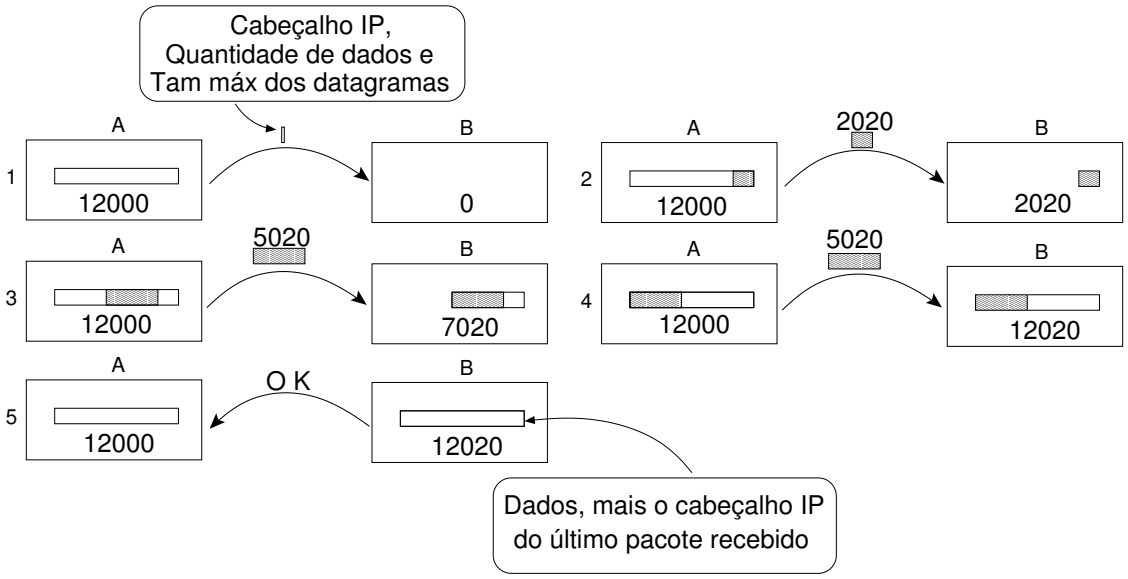

Figura 3.4: Transferência de 12000 *bytes* com datagrama de tamanho máximo igual a 5020 *bytes*

Como se está trabalhando em um aglomerado de computadores que possui uma rede local e confiável, Acredita-se que os pacotes serão entregues em ordem e que a perda de pacotes seja praticamente nula. Isso acontece porque existe apenas um caminho para os dados trafegarem entre os nós que compõem o aglomerado

e, além disso, não existe roteamento entre diferentes redes. Baseado nessas características, o protocolo criado não implementa nenhum mecanismo para verificar a ordem de chegada dos pacotes e não faz nenhuma soma de verificação sobre os dados transferidos.

Em função de se saber a quantidade dos dados e o tamanho dos datagramas a serem enviados, informados pela primeira mensagem recebida, pode-se saber onde devem ser colocados os dados de cada mensagem que chega, e o número de pacotes que se deve receber. Com isso, sabe-se a quantidade de pacotes que se deve esperar. Se a quantidade de dados recebidos coincidir com a que foi informada pela primeira mensagem, retorna-se uma mensagem de transmissão, efetuada com sucesso, da mesma forma, se ficar faltando algum pacote, é transmitida uma mensagem pedindo a retransmissão dos dados, e os dados são retransmitidos integralmente.

3.3 Criação do soquete bruto

Para se criar um soquete bruto, é utilizada a função socket, que retorna um descritor, para um ponto final de comunicação, sendo que é passado para ela PF_INET, indicando que será utilizado IPv4, SOCK_RAW para a criação de um soquete bruto e um número que indica o protocolo a ser utilizado. Se a função retornar o valor -1, isso indica que houve erro na criação do soquete, da mesma forma, a função da mini biblioteca retorna o descritor para o soquete ou o valor -1 se houver algum erro.

Como não se está utilizando nenhum protocolo existente, é passado o número 254, que é um número de protocolo reservado para testes, dessa forma, quando se está recebendo dados no soquete, o núcleo do sistema operacional apenas repassa os pacotes desse mesmo protocolo, o que evita a captura de pacotes de outras aplicações.

3.4 Envio dos dados

Nessa versão da MicroVAPI foi utilizada a opção *IP_HDRINCL*, que torna possível a escrita dos cabeçalhos IP. Isso diminui o trabalho passado ao núcleo do

sistema operacional, deixando a cargo deste apenas a soma de verificação do cabeçalho do pacote IP. Como essa opção foi utilizada, primeiramente, deve-se alocar uma região de memória auxiliar, logo após deve-se escrever o cabeçalho do datagrama IP na região alocada e em seguida inicia-se a escrita dos dados a serem enviados. Após esse processo, é chamada a função sendto para que os dados sejam de fato transferidos, lembrando que, para que esse sistema funcione, é necessário que o número de *bytes* a ser transferido, que é passado para a função, seja igual ao dos dados a serem transferidos mais o tamanho do cabeçalho IP, como mostra a figura 3.5.

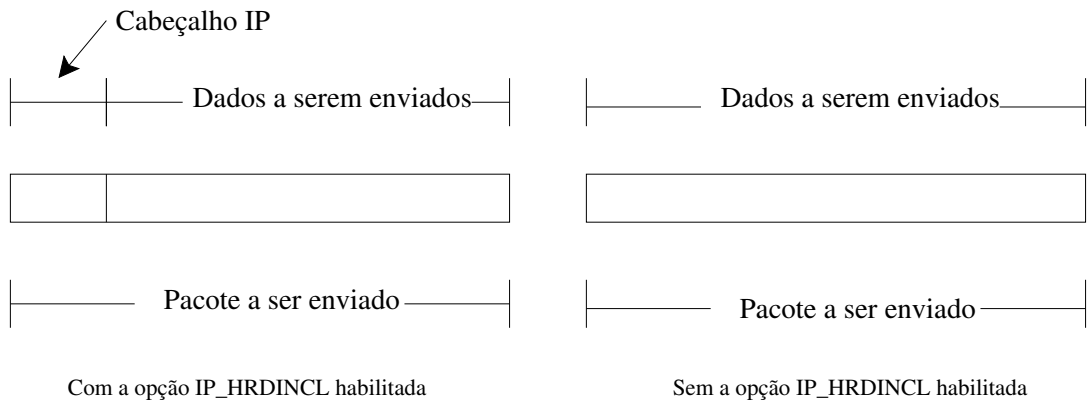

Figura 3.5: Transmissão de dados, com e sem a opção IP_HDRINCL habilitada

Após o envio dos dados, a função de envio da mini biblioteca espera por uma resposta. Essa resposta pode ser tanto de transferência de dados efetuada com sucesso, ou de problemas na transmissão. Caso aconteça a primeira situação, é retornada a quantidade de dados transferidos e o programa segue sua execução normal. Se chegar uma mensagem de problemas na transmissão, então a função reenvia todos os dados, se não chegar nenhuma resposta, é retornado o valor -1, indicando falha no envio dos dados. Para que isso funcione, foi configurado um tempo limite de 3 segundos para esperar a resposta.

3.5 Recepção dos dados

Para a recepção, o usuário deve passar um apontador para uma região de memória, previamente alocada, que possa armazenar os dados recebidos mais os

dados de um cabeçalho IP, que sempre ficam no início da região dos dados (figura 3.6).

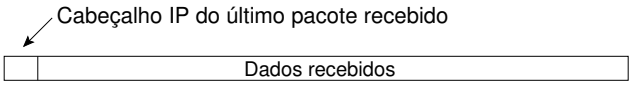

Figura 3.6: Disposição final dos dados após a recepção

Quando a função de recepção dos dados da mini biblioteca é chamada, ela espera por uma mensagem que contenha a quantidade de dados, o tamanho máximo do datagrama e o cabeçalho IP do pacote, o que totaliza 28 *bytes*, se chegar qualquer outra mensagem que contenha um valor diferente de 28 *bytes* a função descarta esse pacote e volta a esperar uma mensagem com 28 *bytes*.

Para detectar que não houve problema na recepção dos dados, foi configurado um tempo-limite de um segundo para esperar por cada pacote de dados. Se não chegar nenhum pacote nesse tempo, ou chegar algum pacote que contenha uma quantidade de dados inferior aos dados esperados para aquele pacote é enviada a mensagem pedindo a retransmissão. A função de recepção da mini biblioteca sempre retorna a quantidade de dados recebidos.

3.6 Adaptações realizadas na MicroVAPI

Para que a MicroVAPI fosse capaz de operar com soquetes brutos foram realizadas modificações em algumas funções que a compõem. Atualmente, existem dois módulos na MicroVAPI, um deles faz a comunicação, utilizando DECK e o outro faz a comunicação, utilizando soquetes brutos.

3.6.1 Inicialização do ambiente

A inicialização do ambiente é feita chamando-se a função EVAPI_get_hca_hnd, que faz a conversão dos nomes dos nós que irão fazer parte do ambiente de execução para o seu respectivo endereço IP e armazenando estes em um arquivo temporário.

Essa inicialização praticamente não sofreu alterações, pois tanto com DECK, como com soquetes brutos, essa conversão de nome para endereço IP é necessária.

O único ponto que foi modificado é que o DECK precisa que seja chamada uma função de inicialização do ambiente DECK, já os soquetes brutos não, então nessa função apenas foi retirada essa chamada de inicialização.

3.6.2 Inicialização do ponto final de comunicação

Para se criar um ponto final de comunicação, a função VAPI_create_qp é chamada. Essa função faz algumas inicializações nas variáveis de condição e de exclusão mútua que são utilizadas para realizar a comunicação assíncrona, nessas variáveis não foi feita nenhuma modificação, pois o esquema de comunicação não foi alterado. O único ponto modificado é que a versão que utiliza DECK como ambiente de comunicação, faz a criação de sua caixa de mensagens aqui, já com os soquetes brutos essa função foi retirada e no lugar foi colocada a função que cria o soquete bruto.

3.6.3 Conexão com o nó remoto

Para se fazer a conexão com o nó remoto é utilizada a função VAPI_modify_qp, que na versão que utiliza DECK como forma de comunicação, chama a função DECK responsável por clonar a caixa de mensagens do nó remoto. Como soquetes brutos, não apresenta o conceito de conexão (*connectionless*), a função DECK responsável pela conexão foi retirada e a função VAPI_modify_qp passou apenas a retornar verdadeiro.

3.6.4 Envio dos dados

Para o envio é utilizada uma função que, na primeira vez que é chamada, lança um processo concorrente, e adiciona um descritor na fila de envio. Esse processo concorrente, que é lançado, é responsável pelo envio das mensagens que estão na fila. Nas outras vezes que essa função é chamada ela apenas adiciona um descritor na fila de envio. Essa função que faz o lançamento do processo e adiciona os descritores na fila de envio não sofreu alterações. Já no processo concorrente que foi denominado de demônio foram necessárias algumas adaptações.

O DECK possui uma abstração para indicar cada nó do ambiente que é representado por um número. Ao se fazer o envio de mensagens, deve-se indicar apenas o número do nó que deve receber os dados e o DECK automaticamente faz a conversão para o endereço IP do mesmo e envia a mensagem. Já os soquetes brutos não utilizam isso, eles apenas trabalham com o endereço IP diretamente, então, no demônio, foi feito um mecanismo de resolução de nomes que busca no arquivo temporário, que foi criado na inicialização, o endereço correspondente ao número passado.

Com o problema da conversão dos nomes resolvido, as funções de empacotamento e envio dos dados foram retiradas, e no lugar delas foi colocada a função responsável por quebrar os dados em pedaços e enviá-los.

3.6.5 Recepção dos dados

Para a recepção dos dados, assim como o envio dos dados, apenas o demônio de recepção foi alterado. Foram retiradas as funções de recepção e desempacotamento do DECK e foi colocada a função que faz a recepção e agrupamento dos dados, utilizando soquetes brutos.

3.6.6 Destruição do ambiente

Para se finalizar uma aplicação que utiliza a biblioteca MicroVAPI deve-se chamar a função VAPI_destroy_qp, que na versão DECK é responsável por chamar a função DECK, que faz a desconexão dos nós e finaliza o ambiente. Já na versão que utiliza soquetes brutos ela é responsável por fechar o descritor do soquete bruto, que foi aberto na inicialização do ponto final de comunicação.

Capítulo 4

Avaliação de Resultados

Este capítulo apresenta a análise de desempenho da MicroVAPI, comparando-se a implementação que utiliza DECK como meio de comunicação e a que utiliza soquetes brutos. Foram implementadas duas aplicações para testar a comunicação síncrona da biblioteca: a primeira delas objetiva o cálculo do tempo de latência e a segunda faz a análise da largura de banda e o tempo de comunicação. Cada aplicação apresenta duas versões: a primeira versão utiliza a biblioteca MicroVAPI, configurada para trabalhar sobre DECK e a segunda configurada para trabalhar com soquetes brutos.

As aplicações foram executadas em duas máquinas do aglomerado de computadores do Laboratório de Sistemas de Computação, sendo que cada um dos nós possui dois processadores Pentium III de 1GHz, com 1024 *megabytes* de memória principal e 256 *quilobytes* de memória *cache* L2 em cada processador. Os dois nós possuem o sistema operacional LINUX, versão do núcleo do sistema 2.6.5. Nos testes foram utilizadas placas de rede *Gigabit Ethernet* 3Com 3C996-T em ambas as máquinas, sendo que a comunicação entre essas utilizou um comutador 3Com 3C17700 *Super Stack Gigabit Ethernet*. Na hora da realização dos testes foi verificado que tanto o comutador, como os nós do aglomerado, estavam sendo utilizados exclusivamente pelas aplicações de testes.

4.1 Latência de Comunicação

A latência de comunicação é o tempo decorrido na transmissão de uma mensagem com tamanho 0 *bytes* [COM 2001, TAN 97, WIL 99] de dados, ou seja, são enviados de um nó para o outro apenas os cabeçalhos dos protocolos e as mensagens de controle. Para se avaliar a latência, é calculada a sobrecarga de *software* e *hardware*, para transmitir e receber a mensagem vazia. Essa sobrecarga inclui: cópias de dados para regiões intermediárias, controle de interrupções, trocas de contexto entre os níveis de usuário e núcleo do sistema operacional, etc [WIL 99].

Para realizar o cálculo da latência foram realizados 5000 iterações de troca de mensagens em cada versão da aplicação. A latência é calculada a partir da divisão do tempo total obtido pelo número de iterações. A tabela 4.1 apresenta os valores obtidos.

Tabela 4.1: Latência de comunicação com DECK e Soquetes Brutos

Meio de comunicação	Latência
DECK	50,5 μs
Soquetes brutos	44,3 μs

Pode-se verificar que o tempo obtido na MicroVAPI que utiliza soquetes brutos é inferior ao da implementação que utiliza DECK como meio de comunicação, para ser mais preciso, o tempo de comunicação da versão que utiliza soquetes brutos é 12% inferior ao da versão que utiliza DECK. Isso se deve ao fato de que quando se utiliza o DECK acaba-se fazendo uso das abstrações de caixa de mensagens que ele proporciona, e as chamadas ao sistema operacional para fazer o envio dos dados, construção de pacotes TCP e IP, controles do protocolo TCP, etc. Já quando se utilizam os soquetes brutos, elimina-se a abstração de caixa de mensagens e diminui-se drasticamente o trabalho passado ao núcleo do sistema operacional, pois o próprio programador deve escrever os pacotes IP, incluindo o cabeçalho IP, e não se utiliza o protocolo TCP.

4.2 Largura de Banda

Esta aplicação tem a função de medir a vazão de dados que podem ser enviados pela rede. Ela é executada em duas máquinas distintas, uma cliente e outra servidora. A cliente envia uma quantidade de dados para a servidora e esta, após receber os dados, os retransmite para a máquina cliente, o que caracteriza uma aplicação PING-PONG.

Para realizar o cálculo da largura de banda foram utilizadas 40 quantidades de dados distintos, que iniciaram em 1 *byte* e terminaram em 1 *megabyte*, cada execução foi repetida 200 vezes e foi feita uma média aritmética entre elas, lembrando que os dados são enviados duas vezes, uma pelo cliente e outra pelo servidor, o que faz com que o tempo obtido tenha que ser dividido por dois.

Quando se utilizam soquetes brutos não se pode transferir datagramas maiores que 65535 *bytes*, o que impede a transmissão de quantias de dados maiores que essa com apenas uma mensagem. Por isso, foi implementado o protocolo que foi explicado no capitulo anterior, com esse protocolo, o usuário deve passar para aplicação o tamanho máximo dos datagramas que ele quer que a aplicação utilize. A fim de encontrar um tamanho máximo de datagrama ideal, foi testada a transferência de 100 *quilobytes* de dados de uma máquina cliente para outra servidora, e capturado os tempos, o tamanho máximo do datagrama inicial foi igual a 100 *bytes*, este sendo incrementado de 100 *bytes* a cada 20 repetições, finalizando os teste com o tamanho máximo de 65000 *bytes*. Após os testes, foram feitas as médias aritméticas para os resultados obtidos e o tamanho que apresentou o menor tempo para a transferência dos 100 *quilobytes* foi 10000 *bytes*, e este foi o valor utilizado na realização dos testes de largura de banda com soquetes brutos.

Cada aplicação gera dois números, o primeiro é o tempo que dura o envio e a recepção dos dados divididos por dois, que é dado em milisegundos, e o segundo é a largura de banda que é calculada pegando-se o número de *bytes* transferidos, dividindo por 1048576 (1 *megabyte*), multiplicando por 1000, pois o tempo está em milisegundos, e dividindo pelo primeiro número, o resultado é dado em MBps

(*megabytes* por segundo).

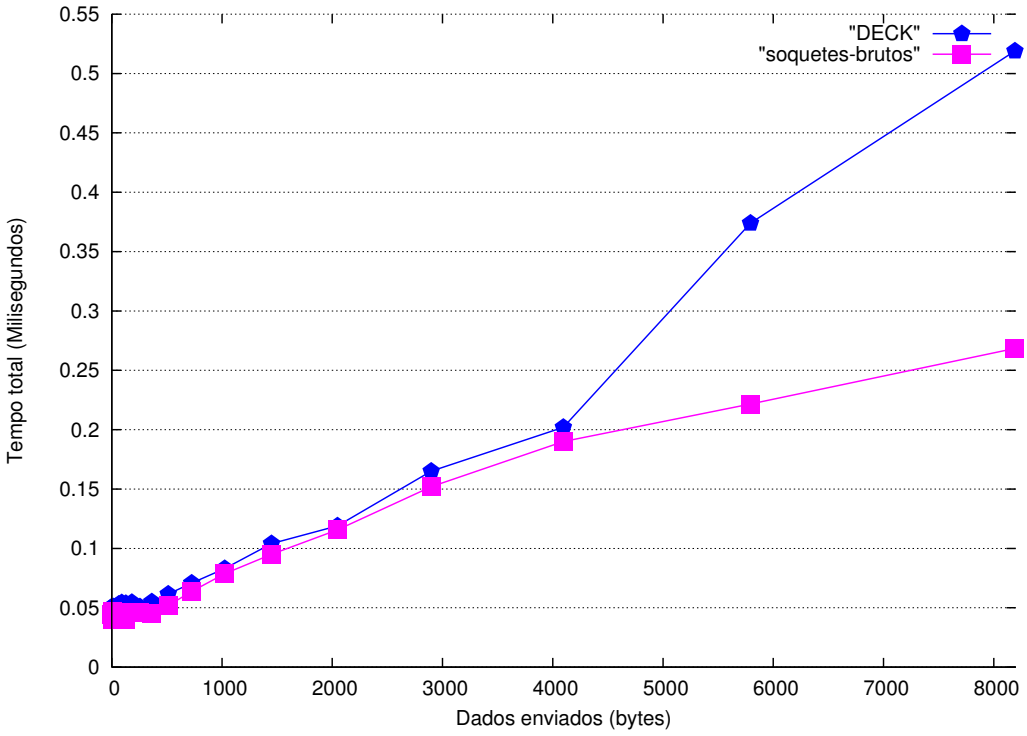

Figura 4.1: Tempo de comunicação para mensagens de até 8200 *bytes*

Através dos gráficos 4.1 e 4.2 pode-se visualizar os tempos de comunicação e a largura de banda na transmissão de mensagens de até 8200 *bytes*. Nesses testes, pode-se observar que a versão que utiliza soquetes brutos apresenta tempos de comunicação inferiores e larguras de banda superior à implementação que utiliza DECK.

As figuras 4.3 e 4.4 mostram uma visão geral dos resultados dos testes, pois nelas são demonstrados os resultados da troca de mensagens de 3 *quilobytes* a 1 *megabyte*. Observando-as, pode-se notar que quando se utilizam os soquetes brutos para transmissão de mensagens grandes, eles apresentam tempos bem inferiores aos atingidos com o DECK, sendo que, com soquetes brutos obtém-se uma largura de banda máxima de 45,76 MBps, ou 366,09 Mbps quando se transmitem 741456 *bytes*. Já quando se utiliza o DECK, consegue-se uma largura de banda máxima de 19,53 MBps, ou 156,25 Mbps quando se transferem 4096 *bytes*, que é inferior à metade da largura de banda obtida com os soquetes brutos, ou seja, em um mesmo intervalo

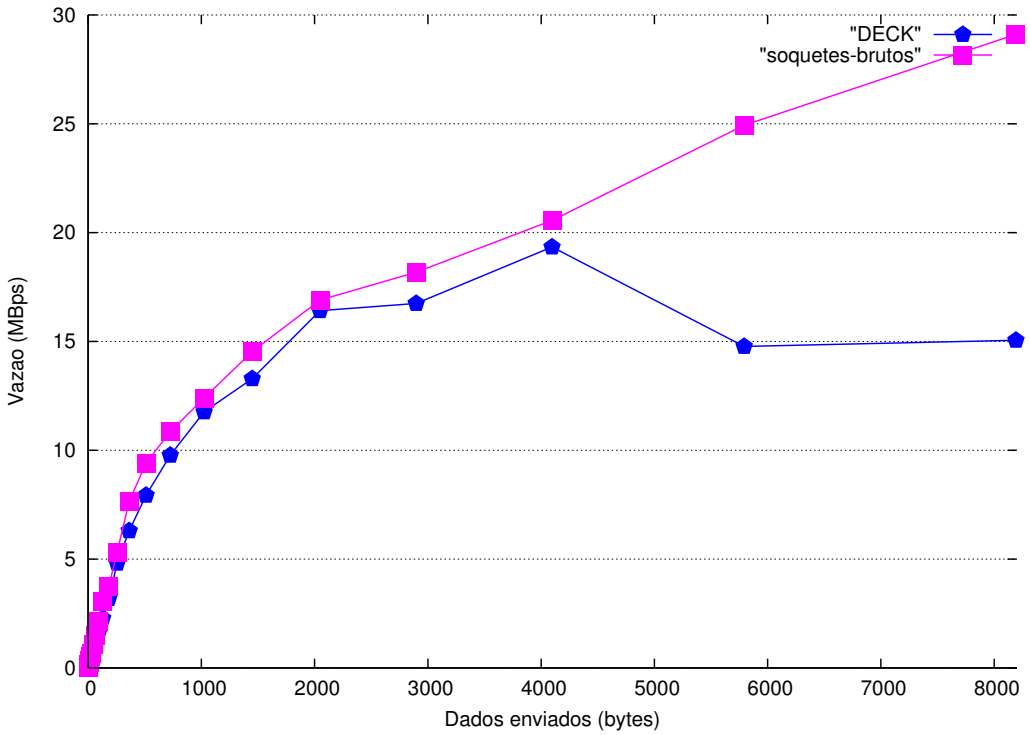

Figura 4.2: Vazão para mensagens de até 8200 *bytes*

de tempo se poderia transmitir mais do que o dobro de dados com soquetes brutos, do que se estive utilizando DECK, como pode ser visualizado na figura 4.4.

Esses resultados demonstram que os soquetes brutos apresentam desempenho superior ao DECK. Sua utilização, porém, pede cuidados, pois para utilizá-los deve-se, obrigatoriamente, possuir permissão de superusuário, o que pode gerar falhas de segurança e tornar o sistema vulnerável.

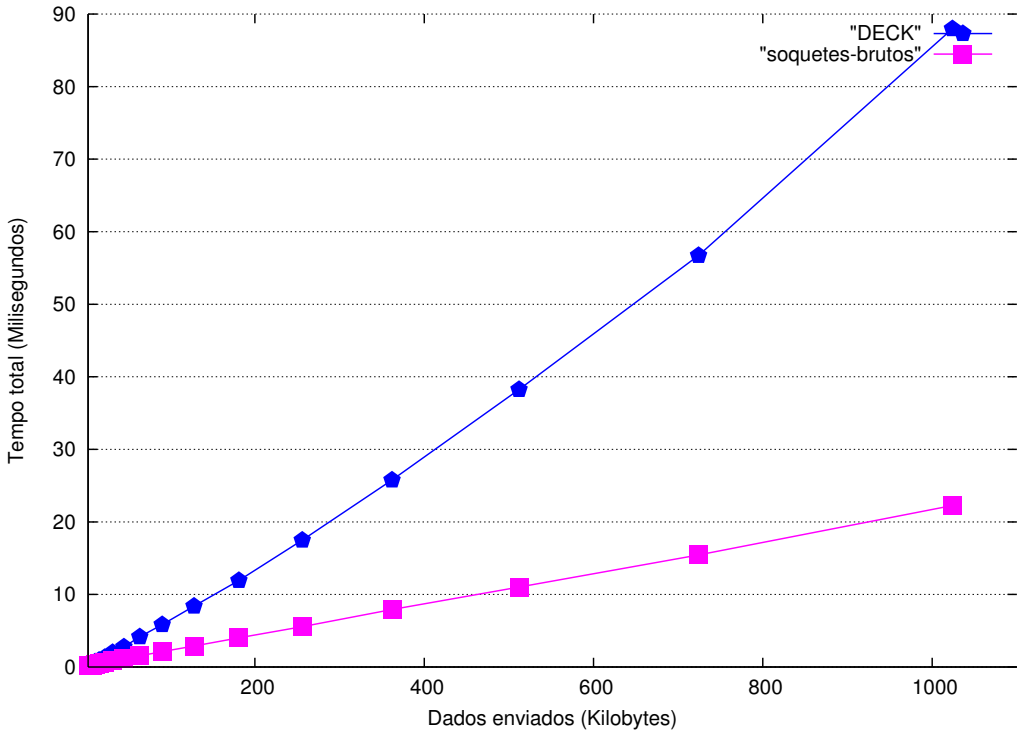

Figura 4.3: Tempo de comunicação para mensagens de 3 *quilobytes* até 1 *megabyte*

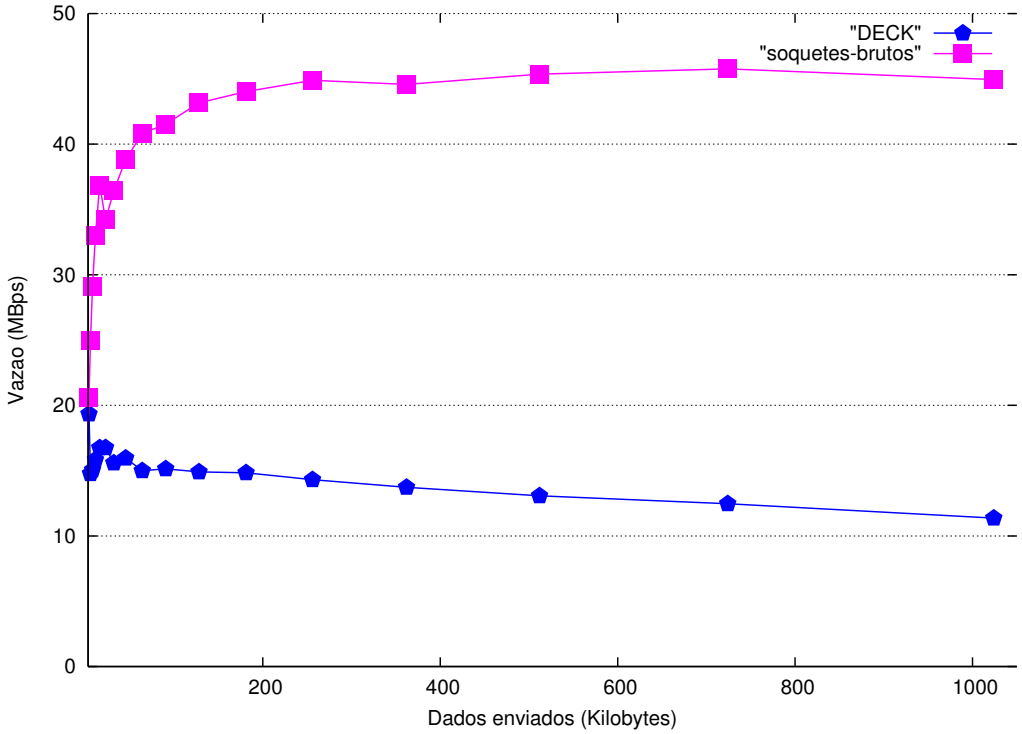

Figura 4.4: Vazão para mensagens de 3 *quilobytes* até 1 *megabyte*

Capítulo 5

Conclusão

Os aglomerados de computadores [BUY 99] se tornaram uma alternativa viável e de baixo custo para aplicações que necessitem de alto poder computacional. Como eles são compostos de nós individuais e interligados por uma ou mais redes [WIL 99], é necessário trocar mensagens entre as aplicações que querem utilizar o seu paralelismo.

Como essa troca de mensagens (tempo de comunicação) é mais onerosa que o tempo de computação [WIL 99], existem diversas pesquisas em tecnologia de redes para interligação dos nós que compõem o aglomerado, onde se destaca a tecnologia InfiniBand que pode transferir até 10 Gbps (*gigabits* por segundo). Porém, essa arquitetura ainda é muito recente, o que faz com que ela seja muito cara para uma utilização maciça em aglomerados.

Para que se possa utilizar programas escritos para operar sobre InfiniBand em aglomerados que não possuem essa tecnologia, foi desenvolvida a biblioteca de comunicação MicroVAPI [REC 2005]. Essa porta as chamadas da biblioteca VAPI (biblioteca que controla a comunicação sobre InfiniBand) [MEL 2000], para o DECK que é capaz de trabalhar sobre as redes SCI, Myrinet, VIA além da tradicional Ethernet. O que faz com que os programas escritos para trabalhar com InfiniBand possam trabalhar sobre essas outras redes.

Quando se está utilizando o DECK [BAR 98] para fazer comunicação sobre Ethernet acaba-se fazendo uso do protocolo TCP, que implementa diversas características para trabalhar sobre diferentes redes e redes não confiáveis, além das abstra-

ções implementadas no DECK como as caixas de mensagens. Essas características não são necessárias em um aglomerado, que, normalmente possui uma rede homogênea e confiável, o que faz com que as aplicações que utilizam DECK não consigam utilizar toda a capacidade oferecida pela tecnologia Ethernet.

Baseado nesse contexto, foi implementada uma segunda versão da biblioteca MicroVAPI que utiliza soquetes brutos em vez de DECK, como alternativa de comunicação. Os soquetes brutos foram escolhidos, pois possibilitam um rápido caminho entre as aplicações e a rede. Eles permitem a manipulação de pacotes IP diretamente em espaço de endereçamento do usuário, o que evita chamadas ao núcleo do sistema operacional, além de evitar cópias de mensagens.

Para testar a nova implementação, foram realizados dois testes com as duas versões da MicroVAPI, o primeiro com o objetivo de medir a latência e o segundo de medir a largura de banda da rede.

No primeiro teste que mediu o tempo de latência, a implementação que utiliza soquetes brutos foi 12% inferior, mostrando que os soquetes brutos são uma interessante alternativa de comunicação. No segundo teste, que mediu a largura de banda, a implementação que utiliza o DECK para fazer a comunicação atingiu a largura máxima de 19,53 MBps, o que é inferior à metade da atingida com os soquetes brutos que foi de 45,76 MBps.

Através dos testes realizados, pode-se verificar que os soquetes brutos são uma excelente alternativa ao DECK, porém esta substituição tem que ser feita com muito cuidado, pois para se utilizar os soquetes brutos, obrigatoriamente, deve-se ter permissão de superusuário do sistema, o que pode gerar falhas de segurança, e deixar o sistema vulnerável a ataques maliciosos.

Como trabalhos futuros para a MicroVAPI, existe a possibilidade de elaborar um protocolo para troca de mensagens mais confiável, onde esse protocolo implemente, por exemplo, somas de verificação (*checksums*) sobre os dados transferidos e verifique a ordem dos pacotes. Além disso, pode-se implementar outras versões da biblioteca MicroVAPI que trabalhem diretamente sobre o protocolo TCP, sem fazer uso da biblioteca DECK, o que eliminaria a sobrecarga imposta pelas abstrações

presentes no ambiente DECK, ou sobre o protocolo UDP (*User Datagram Protocol*), que fornece aos usuários normais acesso ao serviço de entrega de datagramas, porém sem a possibilidade de escrita direta do datagrama, fornecida pelos soquetes brutos.

Bibliografia

[BAR 98] BARRETO, M. E.; NAVAUX, P. O. A.; RIVIÈRE, M. P. DECK: a new model for a distributed executive kernel integrating communication and multithreading for support of distributed object oriented application with fault tolerance support. In: CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACION, 1998, Argentina. **Anais...** [S.l.: s.n.], 1998. v.2, p.623–637.

[BAR 2000] BARRETO, M.; ÁVILA, R.; NAVAUX, P. The multicluster model to the integrated use of multiple workstation clusters. **PC-NOW**, Cancun, Mexico, 2000.

[BUY 99] BUYYA, R. **High performance cluster computing**: architectures and systems. Upper Saddle River, New Jersey: Prentice Hall, 1999.

[COM 2001] COMMER, D. E. **Redes de computadores, transmissão de dados, ligação interredes e web**. [S.l.]: Bookman, 2001. v.2.

[IBT 2002] IBTA. **Inifiniband architecture specification, vol. 1, release 1.1, 2002**. v.1. Disponível em http://www.infinibandta.org/specs.

[KER 89] KERNIGHAN, B. W.; RITCHIE, D. M. **C, a linguagem de programação padrão ANSI**. [S.l.]: Editora Campus LTDA, 1989.

[MEL 2000] MELLANOX. **Introduction to Infiniband**. Santa Clara, California (EUA): Mellanox Technologies Inc., 2000.

[OLI 2001] OLIVEIRA, R. S. de; SILVA CARISSIMI, A. da; TOSCANI, S. S. **Sistemas operacionais**. [S.l.]: Sagra-Luzzato, 2001. v.2.

[REC 2005] RECKZIEGEL, J. F.; RIGHI, R.; PASIN, M. MicroVAPI: utilização da biblioteca DECK em programas java. **ERAD 2005. Escola Regional de Alto Desempenho**, Canoas, RS, Brasil, 2005.

[ROD 97] RODRIGUES, S. H.; ANDERSON, T. E.; CULLER, D. E. **High-performance local-area communication with fast sockets**. 257–274p.

[STE 2005] STEVENS, W. R.; FENNER, B.; RUDOFF, A. M. **Programação de rede unix, API para soquetes de rede**. 3.ed. [S.l.]: Bookman, 2005.

[TAN 95] TANENBAUM, A. S. **Distributed operating systems**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[TAN 97] TANENBAUM, A. S. **Redes de computadores**. Rio de Janeiro - RJ: Ed. Campus, 1997. v.3.

[TOR 2001] TORRES, G. **Redes de computadores, curso completo**. 1.ed. [S.l.]: Axcel Books, 2001.

[WIL 99] WILKINSON, B.; ALLEN, M. **Parallel programming**: Techniques and applications using networked workstations and parallel computers. Upper Saddle River, New Jersey: Prentice Hall, 1999.