LESLIE HARLLEY WATTER

AVALIAÇÃO DE DESEMPENHO DO PROTOCOLO IEEE 802.2-LLC NO KERNEL DO LINUX

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador Prof. Dr.: Roberto A. Hexsel

CURITIBA

2006

LESLIE HARLLEY WATTER

AVALIAÇÃO DE DESEMPENHO DO PROTOCOLO IEEE 802.2-LLC NO KERNEL DO LINUX

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador Prof. Dr.: Roberto A. Hexsel

CURITIBA

2006

AGRADECIMENTOS

À Deus.

À minha família, por tudo, pelo apoio dado, sem o qual eu não teria conseguido.

Ao professor Roberto André Hexsel, por ter acreditado em mim.

Aos Amigos Aristeu Sergio Rozanski Filho, pelas intermináveis horas de IRC e a paciência de um monge; Arnaldo Carvalho de Mello, pela força com o kernel, longas conversas e ajuda no processo todo; Jorge Godoy Filho, pelas inúmeras conversas e incentivos; Silvio Luis Nisgoski, por me ajudar a recuperar os dados do HD num momento crítico; Felipe Augusto van de Wiel, pela dica da utilização do R.

Aos professores Alexandre Ibrahim Direne, Antônio Edison Urban e Bruno Müller Júnior, pelas conversas, pelo grande suporte psicológico e burocrático. À professora Cristina Duarte Murta, pela paciência na entrega das máquinas na primeira fase do mestrado.

Aos professores e funcionários do departamento de informática da UFPR.

Ao prof. Vóldi Costa Zambenedetti, pelo apoio dado no tempo em que estive no Lactec.

O apoio dado pelo CCE nas pessoas de Roberto Haruo Takata e Sandra Chiocca.

E finalmente, mas não menos importantes, a todos que, de alguma maneira, me ajudaram nessa fase da minha vida.

SUMÁRIO

LISTA DE FIGURAS v

LISTA DE TABELAS vi

RESUMO vii

ABSTRACT viii

1 INTRODUÇÃO 1

2 COMUNICAÇÃO DE DADOS 4

2.1 Troca de Mensagens 4

2.1.1 Caminho da mensagem entre duas aplicações 4

2.2 Transporte de Dados 10

2.3 A Camada de Aplicação 11

2.3.1 Soquetes 11

2.4 Protocolo TCP 13

2.5 Protocolo IP 15

2.6 A Camada de Enlace 17

2.6.1 A Sub-Camada LLC 18

2.6.2 O protocolo LLC 19

2.6.3 A Sub-Camada MAC 23

2.6.4 Comparação dos protocolos LLC e TCP/IP 24

2.7 Implementação do Protocolo LLC no Kernel do Linux 26

2.7.1 Acesso às estruturas do soquete LLC no Kernel do Linux 28

3 TRABALHOS RELACIONADOS 30

4 BIBLIOTECAS PARALELAS 34

4.1 Bibliotecas Paralelas MPI e PVM 34

4.1.1 Diferenças entre MPI e PVM 34

4.1.2 Características Comuns 35
4.2 Biblioteca MPI 36
4.2.1 OPENMPI 1.0 37
4.2.2 Estrutura da Biblioteca OPENMPI 38
4.2.3 Implementação do Componente BTL LLC 40

5 AVALIAÇÃO DE DESEMPENHO 44

5.1 Ambiente de Execução dos Testes 45
5.2 Análise Estatística 46
5.3 NETPIPE 47
5.3.1 O Projeto e Metodologia de Testes do NETPIPE 47
5.3.2 Resultados 48
5.4 MPPTEST – *Measuring MPI Performance* 51
5.4.1 Tamanho das mensagens 52
5.4.2 Escalonamento dos Testes 52
5.4.3 Execução dos Testes 53
5.4.4 Análise dos Resultados 54
5.5 A Transformada de Fourier 59
5.5.1 O Algoritmo Paralelo da Transformada Rápida de Fourier 60
5.5.2 Testes e Execução do Algoritmo FFT 62
5.5.3 Análise dos Resultados Obtidos 62
5.6 O Algoritmo de Ordenação Radix 64
5.6.1 O Algoritmo de Ordenação Radix Paralelo 65
5.6.2 Análise dos Resultados Obtidos 66

6 CONCLUSÃO 69

A APÊNDICE 72

A.1 Teste t de Student 72
A.1.1 NetPIPE 72
A.1.2 FFT – Fast Fourier Transform 73
A.1.3 Algoritmo de Ordenação Radix 74

A.2 Componente BTL LLC 74

Referências Bibliográficas **78**

LISTA DE FIGURAS

2.1 Caminho de uma mensagem através do Kernel 9

2.2 Modelo Estratificado ou Modelo TCP/IP 10

2.3 Pacote TCP 14

2.4 Datagrama IP 16

2.5 Subdivisões da Camada de Enlace 17

2.6 Padrão de Quadro Ethernet 23

2.7 Comparação Entre o Modelo Estratificado e a Implementação no Kernel 25

2.8 *Information transfer command/response (I-format PDU)* 27

2.9 Estrutura utilizada para definir o soquete LLC 29

5.1 Tempo de Execução 49

5.2 Vazão Atingida 50

5.3 Laço de Medição do MPPTEST 53

5.4 Execução do MPPTEST em 2 computadores 55

5.5 Execução do MPPTEST em 4 computadores 56

5.6 Execução do MPPTEST em 8 computadores 56

5.7 Execução do MPPTEST em 16 computadores 57

5.8 Porcentagens de Ganho para os Testes com 2,4,8 e 16 Computadores 58

5.9 Eliminação os Efeitos da Cache 59

5.10 Tempo de Execução do Algoritmo FFT 63

5.11 Tempo de Execução do Kernel Radix 67

LISTA DE TABELAS

2.1 Carga na troca de mensagens dentro do Linux 7
2.2 Localização da Camada LLC 19
2.3 Divisão de Classes e Tipos do Protocolo LLC 20
2.4 Formato do PDU do protocolo LLC 21
2.5 Comparação Funcionalidades TCP/IP e LLC 25

4.1 Classificação de Exclusividade de Seleção do Componente BTL 42

5.1 Ganhos em Porcentagem do LLC com Relação ao TCP 63

A.1 Teste *t de Student* para o benchmark NetPIPE - Tempo 72
A.2 Teste *t de Student* para o benchmark NetPIPE - Megabytes 73
A.3 Resultados do Teste *t de Student* para o algoritmo FFT 73
A.4 Resultados do Teste *t de Student* para o algoritmo Radix 74

RESUMO

Para suprir a necessidade de poder computacional de aplicações científicas utilizam-se aglomerados de computadores. A comunicação das aplicações nesses aglomerados é feita usando bibliotecas de troca de mensagens que utilizam normalmente os protocolos TCP/IP como meio de transporte. Restringindo a rede do aglomerado de computadores a uma rede local é possível substituir os protocolos TCP/IP pelo protocolo LLC, com ganho de desempenho.

Este trabalho apresenta uma avaliação de desempenho de uma modificação da biblioteca OPENMPI para trabalhar com o LLC, comparando-a com a implementação TCP/IP. Para a avaliação de desempenho foram utilizados os aplicativos NETPIPE, MPPTEST, a *Transformada Rápida de Fourier* e a *ordenação Radix*.

Os resultados obtidos para o NETPIPE mostram que o LLC tem um desempenho de 16 a 21% superior ao TCP/IP; para o MPPTEST esse resultado varia na faixa de 3 a 12%, sendo que os maiores ganhos ocorrem para mensagens pequenas. Os resultados da *Transformada Rápida de Fourier (FFT)* mostram que o LLC é de 2.8 a 6.8% mais rápido que o TCP/IP variando o número de processadores de 2 a 16. O resultado da *ordenação Radix* não aponta ganho real para o LLC porque esse programa não gera demanda significativa sobre o subsistema de comunicação.

ABSTRACT

Clusters of PCs have been deployed to fulfill the computing demands of scientific applications. Applications are written so that communication is implemented with message passing libraries, and in low cost installations these are based on the TCP/IP protocol suite. If the cluster is physically constrained to the scope of an IEEE 802 local area network, the transport protocol can be replaced by the Logical Link Control (LLC) protocol. Since LLC is far simpler than TCP/IP performance gains can be archieved.

This work presents the performance evaluation of the OPENMPI library that has been adapted to run on top of LLC, and a comparison of this version to the original TCP/IP based library. In assessing the performance were employed the microbenchmarks NETPIPE and MPPTEST, and two complete applications, *Fast Fourier Transform* and *Radix sort*.

The NETPIPE results have shown LLC's performance to be 16 to 21% above TCP/IP's. For MPPTEST, the gains are from 3 to 12%, with short messages yielding the largest gains. The results for *Fast Fourier Transform* have shown that LLC outperforms TCP/IP by 2.8 to 6.8%, as the number of processors varies from 2 to 16. The results for *Radix sort* are very similar for the two versions because this application does not put significant demands on the communication subsystem.

CAPÍTULO 1

INTRODUÇÃO

A demanda por poder computacional em aplicações científicas tem sido crescentemente suprida por aglomerados (*clusters*) de PCs, interligados por uma rede local. Bibliotecas de troca de mensagens como MPI [1] e PVM [2] são usadas para a programação das aplicações segundo o modelo de Troca de Mensagens em Multicomputadores [3].

As bibliotecas de comunicação são implementadas para executarem sobre os protocolos TCP/IP [4]. O protocolo TCP [5] provê comunicação confiável entre computadores localizados em quaisquer locais servidos por uma rede com protocolo similar ao IP. Por conta da incerteza na localização dos computadores, o TCP incorpora uma série de mecanismos para garantir a entrega confiável das mensagens, bem como para manter a rede operando em condições razoáveis de tráfego e sem congestionamento.

Se um aglomerado é instalado num único prédio ou numa sala, os computadores possivelmente estarão interligados por uma rede local, que na grande maioria dos casos é uma rede Ethernet (ou IEEE 802.3*). O protocolo de enlace da Ethernet é o padrão IEEE 802.2 (*Logical Link Control - LLC*), e este incorpora uma série de funcionalidades que são similares ou equivalentes a um sub-conjunto das funcionalidades do TCP. O escopo geográfico do LLC é, por projeto, menor que o do TCP, e o meio no qual o LLC transmite seus dados (sub-camada MAC) é implementado em hardware nos adaptadores de interface de rede. Em PCs de baixo custo, a implementação do TCP (e do IP) é inteiramente em software. Por conta destas duas características, é de se supor que um aglomerado de PCs construído sobre uma biblioteca que utilize LLC como seu protocolo de transporte seja mais eficiente do que um aglomerado baseado na mesma biblioteca mas com TCP como seu protocolo de transporte. Isso decorre das diferentes implementações: os protocolos TCP/IP são implementados com funcionalidades para garantir

entrega confiável através de redes diversificadas, enquanto que o LLC é restrito a uma rede local e o controle da integridade dos dados é responsabilidade do adaptador de rede, que tem o algoritmo de detecção de erros (CRC) implementado em hardware.

Independente da arquitetura empregada na construção de um aglomerado de computadores, e da técnica utilizada na aplicação, todas tem um ponto de convergência: a necessidade de troca de mensagens entre as diferentes instâncias que estão executando a aplicação. Estas instâncias podem ser computadores completos ou apenas processadores com memória compartilhada, aonde todos tem que, em algum momento, se comunicar para trocar resultados e/ou sincronizar o trabalho em algum ponto.

Tamanha é a necessidade de troca de informações que ela pode facilmente tornar-se o gargalo na computação do resultado. Têm-se procurado incessantemente reduzir a carga adicionada à computação pelo número de troca de mensagens, ora procurando minimizar as trocas, ora buscando reduzir o tempo agregado à passagem da mensagem de um ponto a outro do sistema.

Dessa maneira, os estudos voltaram-se para a tentativa de eliminar e/ou reduzir ao máximo o tempo gasto na troca de mensagens. Alguns apontam para os protocolos intermediários como sendo os grandes vilões na utilização do tempo necessário para a troca de mensagens [6–8], enquanto outros apontam para o hardware, e partem em busca de melhorá-lo de maneira a deixar o processador mais tempo livre para executar o processamento [9].

Porém em um ponto há convergência nas opiniões: reduzindo a sobrecarga imposta na troca de mensagens e melhorando o sistema de comunicação haverá uma melhora significativa no desempenho da aplicação.

Como melhorar o sistema de comunicação? Quais os fatores envolvidos na troca de mensagens entre uma máquina e outra? As respostas a essas perguntas dependerão da abordagem a ser utilizada: hardware, melhorando o hardware utilizado na comunicação dos nodos, ou software, tornando-o mais eficiente.

Uma abordagem depende diretamente da outra e não é possível simplesmente melhorar o software sem levar em conta o hardware sob o qual o software executa. Da mesma maneira

ocorre com hardware, aonde uma abordagem possível de ser utilizada é especializar o hardware para atuar em determinado tipo de aplicação [10] como, por exemplo, embutir na interface de rede o processamento dos protocolos TCP/IP [11].

A abordagem utilizando o software tem se mostrado mais flexível uma vez que procura tirar proveito do hardware existente, não sendo necessariamente específica a uma aplicação.

Para reduzir a sobrecarga imposta pelo software, é necessário verificar os protocolos que transportam as mensagens. Aqueles que apontam para os protocolos como sendo os vilões no consumo do tempo de processamento na troca de mensagens normalmente não o fazem apontando para o protocolo como um todo, mas sim para a cópia de dados necessária no processamento desses protocolos [11, 12].

O trabalho desenvolvido em [13] serviu de base para o presente estudo. As modificações efetuadas no kernel do Linux, mais especificamente no protocolo LLC permitem que uma interface de soquete que utilize o protocolo LLC seja criada no espaço de usuário. Antes desse trabalho, somente protocolos de comunicação presentes no kernel do Linux, por exemplo X.25, podiam utilizar-se do protocolo LLC como meio de transporte na rede local.

Para o presente estudo, foi necessário modificar a biblioteca paralela OPENMPI para que esta tivesse suporte ao protocolo LLC. Um novo componente responsável pela comunicação de dados, foi desenvolvido. O resultado desse desenvolvimento é um patch[1] de 9464 linhas que acrescenta o componente BTL LLC à lista de componentes da biblioteca OPENMPI.

Após o trabalho de desenvolvimento na biblioteca OPENMPI, foi efetuada a análise de desempenho dos aplicativos NETPIPE, MPPTEST, FFT e Ordenação Radix variando-se apenas o protocolo de comunicação subjacente (LLC e TCP/IP). Dessa avaliação obtemos resultados que apontam ganhos de desempenho do protocolo LLC com relação ao TCP/IP da ordem de 3% para mensagens grandes e 15% para mensagens pequenas.

[1] Arquivo que contém as diferenças entre uma versão e outra, gerado pelo comando *diff*.

CAPÍTULO 2

COMUNICAÇÃO DE DADOS

2.1 Troca de Mensagens

Os estudos efetuados por Jiuxing Liu et. al [14] mostram que, para se obter um maior nível de detalhe das características de desempenho de interconexão de aglomerados de alta velocidade é importante ir além de simples testes como os utilizados para medir a latência e largura de banda, especificamente considerando certas características como acesso à memória remota e mecanismos de tradução de endereços na interface de rede.

A utilização do protocolo LLC pode ser entendida como um agente facilitador na comunicação dentro de aglomerados, independentemente da arquitetura de rede utilizada. O objetivo dos modelos existentes tem sido o de melhorar o desempenho especificamente reduzindo a latência, a sobrecarga na comunicação, além de aumentar a largura de banda [14].

2.1.1 Caminho da mensagem entre duas aplicações

Na execução de diversas aplicações nota-se que a troca de mensagens entre os nodos do aglomerado é um ponto muito importante e imprescindível em aplicações que executam em paralelo. Comparada à velocidade do processador, a troca de mensagens através da rede torna-se um dos pontos de gargalo na execução das aplicações [3].

Se o processador do sistema for muito lento, esse gargalo pode ser transferido para o processador. Como a velocidade de processamento cresce muito rapidamente, aproximadamente a uma taxa de 35% a 55% ao ano [15], e embora a banda de rede cresça a uma taxa aproximadamente igual, ainda há degraus na banda de rede mais acentuados que no desenvolvimento

do hardware; e, no intervalo de transição de um degrau para o outro na tecnologia, a rede torna-se novamente o gargalo na comunicação. Em sistemas com processadores rápidos, é necessário melhorar o acesso/reduzir o tempo embutido na troca de mensagens de maneira a utilizar efetivamente os processadores reduzindo o tempo de execução da aplicação. A partir da observação do crescimento da velocidade dos processadores, conclui-se que é necessário despender esforços no sentido de minimizar o tempo decorrido na troca de mensagens e o caminho crítico da aplicação ao adaptador de rede. A seguir, descrevem-se as camadas por onde passa uma mensagem e quais são algumas das melhorias possíveis de serem executadas nessas camadas.

Partindo da aplicação, a primeira interface por onde a mensagem passa é a interface de soquete, aonde é feita a cópia dos dados do espaço de memória do usuário para o espaço de memória do kernel quando enviando dados e, do espaço de memória do kernel para o espaço de memória do usuário quando recebendo dados. Nesta cópia de dados de um espaço de memória para o outro, ocorre uma troca de contexto (uma vez que é necessário mudar de um modo não protegido, usuário, para um modo protegido, kernel), fato que é custoso em termos de tempo e processamento porque perde-se todo o ganho obtido através das técnicas de localidade espacial e localidade temporal utilizadas no preenchimento das memórias cache [15]. Um ponto agravante nesse processo de troca de contexto é que o acesso aos dados localizados em dispositivos de entrada/saída têm baixa localidade [16].

No momento em que acontece a troca de contexto o sistema operacional precisa guardar os valores dos registradores, efetivar na memória e/ou disco as alterações efetuadas nas hierarquias de memória (caches e RAM) e, possivelmente, se o sistema estiver com uma carga alta, movimentar a página de memória da aplicação da memória RAM para o disco rígido na área de swap. Toda essa troca de contexto é gerenciada pelo sistema operacional e não haveria como evitá-la sem reescrever partes do próprio sistema operacional.

A segunda camada onde a mensagem deve passar é a camada de processamento do protocolo. Nessa camada, que pode ser subdividida em diversas camadas menores, ocorre o processamento dos diferentes protocolos. No caso específico dos protocolos da família TCP/IP essa

camada é subdividida em duas, uma para os protocolos TCP e/ou UDP, e outra para o protocolo IP. No processamento dos protocolos TCP/UDP ocorre a montagem do segmento TCP e do datagrama UDP respectivamente, além de ser efetuado o cálculo da paridade do cabeçalho dos segmentos/datagramas. Esse cálculo da paridade incorre em mais uma troca de contexto. Independentemente dos protocolos TCP/UDP ocorre também a montagem dos pacotes IP e, se necessária a fragmentação dos dados provindos da camada superior (TCP/UDP) e, o cálculo da paridade do pacote IP.

Depois de percorrida a camada de processamento dos protocolos TCP/IP, a mensagem é enviada diretamente ao controlador do dispositivo (*device driver*) para ser enviada à interface de rede e ao meio físico.

Usando o protocolo LLC, ao invés de montar um pacote como no caso do TCP/IP, monta-se o quadro do protocolo LLC. Após montado o quadro LLC a mensagem é enviada para o controlador do dispositivo, que irá enviá-la para a interface de rede a qual é responsável por, em hardware, montar o quadro Ethernet [17], e calcular o CRC do quadro antes de enviá-lo. O cálculo do CRC é feito diretamente no hardware do adaptador de rede. Por fim a mensagem que chegou ao adaptador deve ser enviada ao meio físico, e no caso da Ethernet ao cabo par trançado que a conduz até a outra estação (ligação ponto a ponto) ou ao hub/switch que rotea-la até a estação destino. A tabela 2.1 ilustra as camadas onde ocorrem os aumentos de carga do sistema [18].

Camada	Sobrecarga	Descrição	Dispositivo
Soquete	$M_{uk}(m)$	cópia de dados do espaço do usuário para o espaço do kernel	CPU
	$M_{ku}(m)$	cópia de dados do espaço do kernel para o espaço do usuário	CPU
Protocolo	CS(m)	cálculo do checksum	CPU
	TCP_{in}/TCP_{out}	processamento específico ao protocolo TCP	CPU
	IP_{in}/IP_{out}	processamento específico ao protocolo IP	CPU
Interface de Rede	$M_{ka}(m)$	cópia de dados do espaço de kernel para o espaço de E/S (adaptador)	CPU
	$M_{ak}(m)$	cópia de dados do espaço de E/S (adaptador) para o espaço de kernel	DMA
	ETH_{out}	processamento do enlace	CPU
	I_s	interrupção de transmissão completa	CPU
	I_r	interrupção de recebimento e processamento do enlace	CPU
Adaptador de Rede	Tx(m)	tempo de transmissão do pacote	Adaptador
	Rx(m)	tempo de recepção do pacote	Adaptador

Tabela 2.1: *Carga na troca de mensagens dentro do Linux*

Na camada física ainda devem ser considerados alguns pontos que interferem no tempo de transmissão da mensagem entre uma estação e outra [19].

Tempo no Fio o tempo de transmissão do quadro no meio físico.

Latência do Controlador o tempo gasto pode ser atribuído a dois fatores:

1. O tempo que o emissor demora para iniciar a transferência de dados na rede uma vez que a estação disponibilizou os dados para o controlador, e
2. O intervalo de tempo entre a chegada dos dados no receptor e o tempo que ele leva para disponibilizar esses dados para a estação/destino.

Tempo de Controle/Transferência de Dados Os dados são movidos ao menos uma vez entre o barramento de rede e o barramento de memória.

Embora esse tempo possa ser reduzido por controladores que utilizem DMA, existe uma carga extra no controle da CPU porque devem ser alocados descritores de memória especiais para a transferência de dados. Utilizando essa técnica o tempo de transferência de dados é absorvido pelo tempo de Latência do Controlador.

Vetoração de Interrupções No receptor o software deve organizar as interrupções de chegada de pacote para o tratador de interrupções do *device driver*, permitindo que elas sejam tratadas em seqüencia, ou respeitando o nível de prioridade da interrupção.

Serviço de Interrupção Quando ocorre uma interrupção, o sistema operacional deve executar algumas ações importantes antes de atendê-la: salvar o conteúdo dos registradores, apontadores, efetivar as modificações efetuadas na hierarquia de memória cache e tabela de páginas de maneira a permitir que o estado atual da aplicação possa ser restaurado ao retornar do tratamento da interrupção. Nesse momento ocorre outra troca de contexto, tornando ainda mais custosa a operação de troca de mensagens.

O tempo que leva para a mensagem ir de uma estação a outra pode ser chamado de latência. É, dentre outros, esse tempo que se deseja reduzir. Quanto menor a latência na troca de mensagens, maior a possibilidade de aumentar a largura de banda, e aumentar a quantidade de informações trocadas em determinado período de tempo entre dois pontos.

O objetivo final é reduzir o tempo gasto na troca de mensagens entre as estações. Analisando o caminho percorrido pela mensagem durante a sua recepção, observa-se que a camada/protocolo LLC é a primeira camada atingível via software que fornece os serviços mínimos necessários para a comunicação na rede local. As demais camadas, IP, TCP/UDP, acrescentam funcionalidades imprescindíveis quando é necessária a comunicação inter-redes, porém para uma rede local, o protocolo LLC pode ser utilizado como alternativa menos custosa/mais eficiente que os demais protocolos.

A figura 2.1 ilustra o caminho que a mensagem percorre quando é transmitida de um ponto a outro da rede.

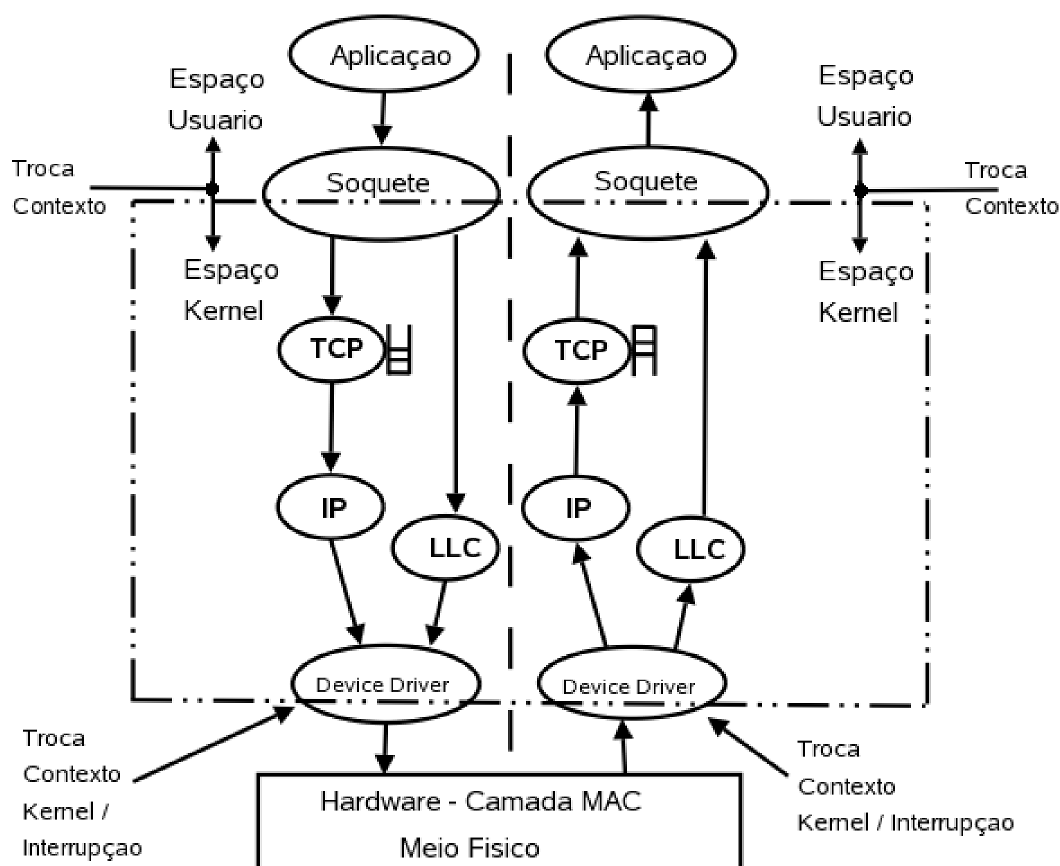

Figura 2.1: *Caminho de uma mensagem através do Kernel*

O tempo que a mensagem leva para percorrer o meio físico só pode ser reduzido com melhorias no hardware do adaptador, e mesmo assim até certo limite.

Para avaliar o desempenho dos modelos existentes e simular variações nas propostas de modelos possíveis encontrou-se um modelo de avaliação. O modelo de avaliação LogP [14] avalia os seguintes parâmetros da rede/do modelo:

L o limite máximo para a LATÊNCIA ou atraso na transmissão de uma ou mais palavras da mensagem da origem até o destino.

o a sobrecarga ou OVERHEAD que define a quantidade de tempo que o processador trabalha na transmissão ou recepção de uma mensagem. Tempo no qual o processador não faz nada além de tratar a mensagem.

g o intervalo ou GAP; definido como sendo o menor intervalo de tempo entre a transmissão/re-

cepção de duas mensagens consecutivas. A recíproca do gap é a largura de banda disponível ao processador.

P o número de processadores/módulos de memória do sistema.

Dos parâmetros utilizados pelo modelo LogP , o parâmetro a ser utilizado/mensurado nesse trabalho é a *sobrecarga (o)*. Pretende-se reduzir a sobrecarga imposta pelo sistema de camadas e protocolos à transmissão de mensagens entre as estações da rede local através da eliminação do tratamento das mensagens pelos protocolos TCP/IP.

Um dos desafios a serem resolvidos para que a troca de mensagens seja mais eficiente é: reduzir a sobrecarga imposta pelos protocolos para a transmissão de dados, uma vez que a grande parte dos dados trafegados na rede são de controle dos protocolos das camadas intermediárias, criando carga adicional para a transmissão de dados.

2.2 Transporte de Dados

Quando uma aplicação necessita comunicar-se com outra aplicação remota, que reside em uma outra máquina de uma rede local, a primeira deve enviar uma mensagem para a segunda. Essa mensagem terá que percorrer algumas camadas intermediárias para que consiga sair do computador no qual está executando, trafegar na rede e chegar na outra máquina onde a segunda aplicação está executando. Nesse texto é empregado o modelo estratificado (de 5 camadas), também conhecido como TCP/IP usado na Internet. Esse modelo é ilustrado pela figura 2.2

<table>
<tr><th>Camada</th><th>Protocolo</th></tr>
<tr><td colspan="2">Aplicação</td></tr>
<tr><td>Transporte</td><td>TCP</td></tr>
<tr><td>Rede</td><td>IP</td></tr>
<tr><td colspan="2">Enlace</td></tr>
<tr><td colspan="2">Física</td></tr>
</table>

Figura 2.2: *Modelo Estratificado ou Modelo TCP/IP*

2.3 A Camada de Aplicação

A camada de aplicação é onde as aplicações estabelecem a comunicação, sem se preocupar com a implementação das funcionalidades fornecidas pelas camadas inferiores.

Essa camada permite isolar a aplicação dos padrões físicos, mecânicos e referentes ao transporte de dados, permitindo assim que uma aplicação possa ser transportada de uma rede para a outra rede com diferentes protocolos e meios físicos sem perder suas funcionalidades.

Nessa camada encontramos várias áreas, dentre elas: a área de segurança, tratada por uma gama de protocolos que podem ser utilizados para garantir a privacidade do usuário; a área de manipulação de nomes, representada pelo DNS (*Domain Name Service*) [20, 21], e a área de apoio ao gerenciamento de redes, representada pelo protocolo SNMP (*Simple Network Management Protocol*) [22].

A comunicação da aplicação com as camadas inferiores é feita através de um mecanismo chamado de soquetes, descritos a seguir.

2.3.1 Soquetes

Os soquetes são canais de comunicação entre processos e também os meios primários de comunicação com outras máquinas [23]. É através de soquetes que a aplicação transmite os dados que quer enviar através da rede.

No Linux é possível utilizar os soquetes de diferentes maneiras, porém a padronização definida pela interface de programação (API) definida pela Berkeley Software Distribution (BSD) é aceita como sendo a melhor maneira de se utilizar os soquetes. A forma como são definidas as funções dessa padronização facilita a interoperabilidade da aplicação com o sistema. A seguir são descritos alguns dos aspectos principais dos soquetes no contexto do sistema operacional.

A estrutura de comunicação do kernel consiste de três partes: a camada de soquete, a camada do protocolo, e a camada de dispositivo. A *camada de soquete* fornece uma interface entre as

chamadas dos sistema e as camadas inferiores com a aplicação, a *camada de protocolo* contém os módulos de protocolo utilizados para efetuar a comunicação (por ex. TCP e IP), e a *camada de dispositivo* contém os *device drivers* que controlam os dispositivos de rede [23].

Os processos se comunicam utilizando a arquitetura cliente-servidor: um processo servidor escuta em um soquete, e o processo cliente que se comunica com o servidor envia no soquete. O kernel do sistema mantém internamente as conexões e rotas do cliente para o servidor.

Soquetes que compartilham propriedades de comunicação, como por exemplo convenções de nomes e formatos de protocolos de endereço, são agrupados em domínios, como por exemplo o domínio *INET* que agrupa os soquetes de protocolos Internet. Cada soquete tem um tipo distinto, usa um circuito virtual (*stream socket*) ou um datagrama.

Um *circuito virtual* permite a entrega seqüencial e confiável dos dados. Um circuito virtual cria a ilusão de que existe um meio físico dedicado para a comunicação entre as duas aplicações.

Os *datagramas* não garantem seqüencia, entrega ou mesmo entrega única dos pacotes, porém são menos onerosos que os circuitos virtuais porque não precisam executar operações complexas de início e fim de conexão.

O sistema operacional contém um protocolo padrão para cada tipo de combinação <domínio,soquete> permitida. Por exemplo, o protocolo *Transport Connect Protocol* (TCP) fornece um serviço com circuito virtual e o protocolo *User Datagram Protocol* (UDP) fornece um serviço de datagrama, ambos no domínio Internet.

A interface de soquete se utiliza de diversas chamadas ao sistema e dentre elas se destacam:

socket Utilizada para criar o descritor do soquete.

connect Utilizada para estabelecer a conexão entre o soquete cliente e o soquete servidor.

bind Responsável por associar um nome a um descritor de soquete.

listen Utilizada para estabelecer o tamanho da fila de requisições de conexão ao servidor quando o servidor aceita conexões do tipo circuito virtual.

accept Utilizada para receber as requisições de conexão ao processo do servidor e cria um novo descritor de soquete para estabelecer a conexão.

send Envia dados através do soquete.

recv Responsável por receber os dados provenientes do soquete.

Informações mais detalhadas a respeito dos soquetes podem ser encontradas em [23, 24].

2.4 Protocolo TCP

A camada de transporte é responsável por organizar toda a hierarquia de protocolos, tendo como função prover a transferência de dados entre a origem e o destino, independente do meio físico utilizado.

Na camada de transporte são oferecidos dois tipos de serviço à aplicação: não orientado à conexão e orientado à conexão.

Um serviço orientado à conexão fornece um "caminho virtual" de uma máquina a outra no qual as mensagens trafegam dados. A troca de mensagens envolve um procedimento de abertura de conexão, transmissão de dados e fechamento de conexão. Usualmente um serviço orientado à conexão é mais confiável que um serviço não-orientado à conexão pelo fato de ter mecanismos de controle de entrega e seqüenciamento de pacotes.

Em um serviço não-orientado à conexão não existe um circuito pré-definido para as mensagens. A mensagem é simplesmente enviada na rede e não existe garantia de entrega de mensagens, dessa maneira este serviço tem custo menor que o orientado à conexão que garante a entrega das mensagens.

Outra característica inerente à camada de transporte e conseqüentemente aos protocolos de transporte é a multiplexação de serviços, que cria *circuitos virtuais* de modo que várias conexões de transporte utilizem uma única conexão de rede permitindo um melhor balanceamento de custo/utilização dos circuitos.

A seguir é discutido o protocolo TCP que é um exemplo de protocolo orientado à conexão amplamente utilizado.

O TCP garante a confiabilidade e a entrega ordenada dos fluxos de dados. O TCP é um protocolo duplex que emprega um mecanismo de controle de fluxo de dados que permite ao receptor limitar a quantidade de dados que o emissor pode transmitir. Outra funcionalidade importante é que pacotes corrompidos ou perdidos são retransmitidos transparentemente.

O TCP suporta o mecanismo de endereçamento utilizando portas lógicas. Para identificar unicamente uma aplicação, o protocolo TCP utiliza o par <porta,estação> (onde estação se refere ao endereço IP) como chave de multiplexação. Dessa maneira, os processos são endereçados/identificados localmente à cada estação. Como o tamanho dos campos *Source Port* e *Destination Port* é de 16 bits, o número de portas fica limitado a um total de 65536.

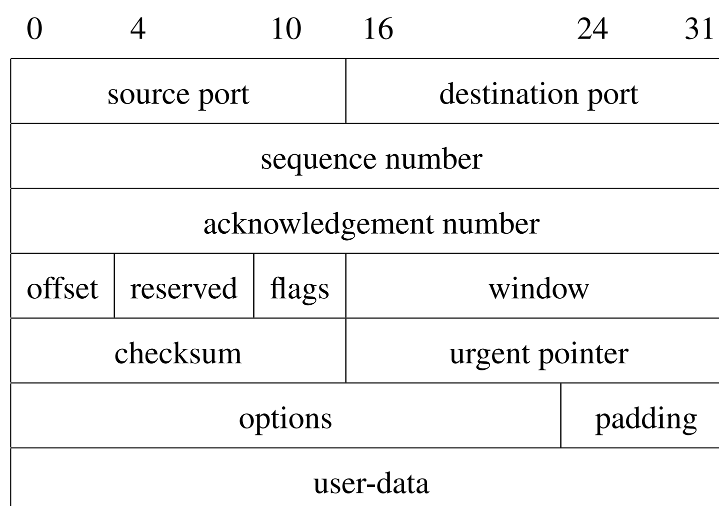

Figura 2.3: *Pacote TCP*

Além dessas características, o TCP implementa um mecanismo eficiente de controle de congestionamento, que limita a velocidade de transmissão do TCP com o objetivo de não sobrecarregar a rede no caso de possivel congestionamento.

2.5 Protocolo IP

A camada de rede provê os recursos para a transferência de pacotes de uma origem para um destino. Para percorrer esse caminho, que pode ter tamanhos máximos de pacotes (MTU[1]) de diferentes tamanhos, passando por roteadores intermediários ou não, a camada de rede utiliza e extende os serviços da camada de enlace. Enquanto a camada de enlace se limita a uma única rede local, a camada de rede provê o transporte ponta a ponta, passando por diferentes redes [25].

O protocolo de rede trata os recursos da camada de rede, criando a abstração necessária para a utilização desta pela camada de transporte. Esse protocolo deve possuir características tais como:

- mapeamento entre endereços de rede e endereços de enlace,
- endereçamento e roteamento de pacotes,
- transmissão dos dados do serviço de rede.

O protocolo de rede mais comummente empregado é o *Internet Protocol* (IP), que é descrito a seguir.

O elemento de interligação das diferentes redes que compõe a Internet é o protocolo IP, através do qual todos os dados são transmitidos [26]. O protocolo IP foi projetado tendo em vista a interligação de redes e tem como principal tarefa o fornecimento da melhor forma de transporte de datagramas da origem para o destino, independente de onde essas máquinas estiverem; seja na mesma rede ou em redes distintas [27].

Como a arquitetura da Internet foi projetada para interligar redes das mais diferentes tecnologias, inevitavelmente deve lidar com diferentes tamanhos de quadros no nível físico. Como exemplo observa-se que, enquanto na Ethernet os quadros podem transportar 1500 octetos de dados, existem outros meios como a rede FDDI (fibra ótica), nos quais os quadros transportam

[1]Maximum Transfer Unit

4470 octetos de dados. Essa diferença no tamanho dos quadros, faz com que seja necessária a fragmentação de quadros [28] para que os pacotes IP sejam transportados em quadros com MTUs diferentes, em redes distintas.

Devido às diferentes implementações de meios físicos existentes, a fragmentação se faz necessária quando o pacote está passando de uma rede com um tamanho de quadro maior para uma rede que tenha um tamanho de quadro menor. O protocolo IP tem um campo no seu cabeçalho, *DF (don't fragment)* e *MF (more fragments)* que indica se um pacote não pode ser fragmentado e se existem mais fragmentos daquele pacote. Caso o pacote não possa ser fragmentado, e sua rota passe por uma rede onde o tamanho do quadro é menor que o atual, ele deve ser roteado de maneira a contornar a rede, ou então ser descartado.

Para atender organizações de tamanhos diferentes, é necessário que o protocolo IP seja capaz de endereçar as redes e máquinas de diferentes maneiras. Tendo em vista essa necessidade, os endereços IP foram divididos em classes que abrigam diferentes tamanhos de redes.

A figura 2.4 mostra os campos de um datagrama IP.

<table>
<tr><td>0</td><td>4</td><td>8</td><td>16</td><td>19</td><td>24 31</td></tr>
<tr><td>vers</td><td>hlen</td><td>service type</td><td colspan="3">total length</td></tr>
<tr><td colspan="3">identification</td><td>flags</td><td colspan="2">fragment offset</td></tr>
<tr><td colspan="2">time to live</td><td>protocol</td><td colspan="3">header checksum</td></tr>
<tr><td colspan="6">source ip address</td></tr>
<tr><td colspan="6">destination ip address</td></tr>
<tr><td colspan="5">options (if any)</td><td>padding</td></tr>
<tr><td colspan="6">data</td></tr>
<tr><td colspan="6">...</td></tr>
</table>

Figura 2.4: *Datagrama IP*

Mais detalhes a respeito do protocolo IP podem ser encontrados em livros didáticos tais como [27, 29]. Os detalhes da fragmentação são importantes aqui uma vez que essa funciona-

lidade está presente tanto no IP quando no LLC e a fragmentação tem uma influencia direta na transmissão de mensagens. A fragmentação é a utilizada na explicação do comportamento observado nos resultados do teste MPPTEST que será visto no capítulo 5.

2.6 A Camada de Enlace

A camada de enlace pode ser dividida em duas sub-camadas, controle lógico de enlace (LLC) e acesso ao meio (MAC) [17]. Essa divisão é ilustrada na figura 2.5, sendo o "meio físico" representado pelo padrão Ethernet IEEE 802.3 [30].

A sub-camada LLC é responsável pela parte de gerenciamento do enlace de dados, sendo a sub-camada MAC responsável pelo acesso ao nível físico.

Camada de Enlace	IEEE 802.2 Camada de Controle do Enlace Lógico (LLC)
	Camada de Controle de Acesso ao Meio (MAC) IEEE 802.3 CSMA/CD

Figura 2.5: *Subdivisões da Camada de Enlace*

A camada de enlace fornece serviços à camada de rede entre duas máquinas ligadas por um enlace físico. Para oferecer serviços à camada de rede, a camada de enlace deve usar os serviços fornecidos pela camada física. A camada física aceita um fluxo contínuo de bits e tenta entregá-lo no destino, sem contudo garantir a entrega e/ou corretude desses bits. A camada de enlace é responsável por detectar e, se necessário, corrigir os erros que porventura tenham ocorrido na camada física. A estratégia adotada pela camada de enlace para verificar a corretude da transmissão dos dados é dividir o fluxo de bits em quadros e verificar se os dados contidos nesse quadro estão corretos usando um dígito de verificação de corretude (também chamado de checksum ou *Cyclic Redundancy Check*[2]) [31].

Quando um quadro chega no destino, o checksum dos dados presentes no quadro é calculado e comparado com o checksum contido no quadro. Caso o checksum calculado no destino seja

[2] CRC

diferente do checksum enviado pela origem, ocorreu um erro na transmissão. Dependendo do tipo do erro, a camada de enlace pede a retransmissão do quadro.

2.6.1 A Sub-Camada LLC

A sub-camada LLC encontra-se na parte superior da camada de enlace de dados e executa as seguintes funções:

- Gerenciamento do enlace de dados
- Endereçamento do enlace
- Definição dos pontos de acesso ao serviço (*Service Access Points (SAPs)*)
- Seqüenciamento dos quadros
- Enquadramento
- Detecção de Erros
- Controle de Fluxo

A sub-camada LLC esconde os detalhes do meio físico, que são do escopo da sub-camada MAC, para as camadas superiores tratarem com qualquer tipo de camada MAC (por exemplo, padrão Ethernet - IEEE 802.3 CSMA/CD ou padrão Token Ring IEEE 802.5). É possível observar na tabela 2.2 onde se encontra a sub-camada LLC no contexto das camadas de abstração mais próximas.

<table>
<tr><td rowspan="2">Rede</td><td>TCP/IP
SAP:80</td><td>NetBIOS
SAP:F0</td><td>IPX
SAP:E0</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td rowspan="2">Enlace</td><td colspan="3">IEEE 802.2 Camada de Controle de Enlace Lógico (LLC)</td></tr>
<tr><td colspan="3">Camada de Controle de Acesso ao Meio (MAC)
IEEE 802.3 CSMA/CD</td></tr>
<tr><td>Física</td><td>802.3 - 10Base5</td><td>802.3a - 10Base2</td><td>802.3i - 10BaseT</td></tr>
</table>

Tabela 2.2: *Localização da Camada LLC*

2.6.2 O protocolo LLC

O protocolo de nível mais baixo que pode ser atingido a partir do espaço de usuário em uma máquina com sistema operacional Linux é o protocolo de controle da camada de enlace ou simplesmente LLC[3].

O protocolo LLC suporta três tipo de serviços para o envio de dados:

LLC tipo 1 não-orientado à conexão e sem aviso de recebimento ou aceitação[4],

LLC tipo 2 orientado à conexão e com aviso de recebimento,

LLC tipo 3 não-orientado à conexão e com aviso de recebimento.

Os tipos 2 e 3 de LLC são confiáveis e garantem que a mensagem chegue ao destino. Esses dois tipos de transmissão utilizam a técnica de janelas deslizantes [27] para garantir que o pacote chegue ao destino. Um fator limitante do LLC é que, por estar perto do hardware, o LLC tipo 2 consegue gerenciar apenas uma única conexão ao mesmo tempo dependendo do hardware utilizado.

Além disso, os serviços fornecidos pelo protocolo LLC são divididos em quatro classes que suportam esses três tipos de operação.

[3]Logical Link Control Protocol
[4]*Acknowledgement*

Tipos de Operação Suportados	Classes de LLC			
	I	II	III	IV
1	x	x	x	x
2		x		x
3			x	x

Tabela 2.3: *Divisão de Classes e Tipos do Protocolo LLC*

O padrão IEEE 802.2 [17] identifica quatro classes distintas de operação do protocolo LLC. A classe I fornece um serviço não orientado à conexão. A classe II fornece os serviços orientado e não-orientado à conexão. A classe III fornece os serviços orientado à conexão com aviso de recebimento e não-orientado à conexão. A classe IV fornece os serviços não-orientado à conexão com aviso de recebimento, orientado à conexão e não-orientado à conexão.

Todas as classes de operação devem suportar pelo menos o tipo 1. Além disso, o suporte ao tipo 1 em um protocolo LLC de classe II, III ou IV deve ser totalmente independente dos modos ou da mudança dos modos de operação. Uma classe II, III ou IV deve ser capaz de alternar entre os tipos de operação 1,2 e 3 tratando, se necessário, individualmente pacote a pacote.

Formato do Pacote LLC

Diferentemente do protocolo TCP, o protocolo LLC não identifica seus processos pelo uso de portas lógicas, ao invés disso, são identificados através das portas de acesso a serviço (*Service Access Point* – SAP). Uma porta de acesso ao serviço é um endereço de 8 bits que pode indicar um único processo ou um grupo de processos.

A unidade de dados do protocolo (*Protocol Data Unit* – PDU) LLC consiste de quatro campos, é mostrada na tabela 2.4.

DSAP	SSAP	Controle	Informação
8 bits	8 bits	8 ou 16 bits	M*8 bits

Tabela 2.4: *Formato do PDU do protocolo LLC*

DSAP O primeiro campo, DSAP contém um endereço simples no formato SAP. Esse campo identifica o *destino* ou *grupo de destino* da unidade de dados do protocolo. Para que um PDU tenha mais de um destino, o campo DSAP deve especificar um grupo SAP que indica todos os membros do grupo. Para identificar se um endereço DSAP especificado nesse campo é um endereço individual ou o endereço de um grupo, é necessário verificar o bit menos significativo do campo DSAP. Caso esse bit seja 1 no endereço, tem-se um endereço de grupo SAP, caso contrário um endereço individual.

SSAP O segundo campo, SSAP também é um endereço de 8 bits no formato SAP. Esse endereço identifica a porta de serviço de *origem* da PDU. O bit menos significativo desse campo pode ser utilizado para enviar PDU's de comandos e respostas para o receptor. Esses comandos e respostas podem ser utilizados para negociar estado, desconexão e teste de conexões entre outras coisas.

Controle Esse campo consiste de um ou dois octetos usados para designar comandos e funções de resposta. O campo de controle contém os números de seqüência do quadro quando o protocolo LLC emprega aviso de recebimento[5].

Informação O último campo da PDU LLC é o campo de informação. Esse campo contém um número variável de octetos (inclusive zero) utilizados para transportar os dados das camadas superiores.

Uma porta de acesso a serviço é equivalente a uma porta para o protocolo da camada de transporte. Se a rede usa múltiplos protocolos, cada protocolo da camada de rede terá a sua própria SAP. Esse é o método utilizado pelo LLC para identificar qual protocolo está conver-

[5] *Acknowledgements*

sando com qual outro. Por exemplo, TCP/IP, NetBIOS e IPX tem cada um a sua porta de acesso (tabela 2.2).

As portas de acesso de serviço garantem que o protocolo da camada de rede na origem converse com o mesmo protocolo na camada de rede no destino.A seguir veremos os tipos de operação da camada LLC e uma breve descrição de cada um deles.

Tipos de operação da camada LLC

A camada LLC define dois tipos básicos de operação para comunicação de dados com relação à conexão:

- Não orientado à conexão – tipos 1 e 3
- Orientado à conexão – tipo 2

Não Orientado à Conexão O serviço não orientado à conexão fornecido pelo protocolo LLC é similar ao serviço fornecido pelo protocolo UDP, não havendo garantias de entrega dos pacotes. Os pacotes simplesmente são enviados sem que haja nenhuma verificação e/ou aviso de recebimento no destino ou de parte do destinatário.

Orientado à Conexão O serviço orientado à conexão é similar a um circuito virtual, com seu ciclo de vida composto por três fases distintas, estabelecimento da conexão, transporte de dados, encerramento da conexão. Os dados são transmitidos com garantias de entrega e corretude.

A camada LLC provê quatro serviços quando opera em modo orientado à conexão:

1. Estabelecimento da Conexão
2. Confirmação e Aceitação do dado que foi recebido.
3. Recuperação de erros através de requisição de re-envio de dados recebidos com erro.

A utilização da técnica de janelas deslizantes no protocolo LLC permite um aumento da taxa de transferência de dados. Quando operando no modo orientado à conexão, todos os quadros LLC (*PDU's*) devem receber um aviso de recebimento (ACK), podendo este ser um aviso de aceitação múltipla uma vez que o protocolo LLC implementado utiliza-se da técnica de janelas deslizantes.

A camada LLC da estação receptora mantém um controle dos quadros que está recebendo, se um quadro for perdido durante a transmissão, ela requisita à estação emissora que reinicie a retransmissão a partir do quadro perdido, descartando todos os quadros subseqüentes anteriormente recebidos.

O número de quadros com recebimento não confirmado é determinado pelo tempo que eles levam para chegar ao destino, adicionando-se o tempo que o destino leva para enviar o quadro de recebimento. O tempo de ida e volta do quadro é dependente tanto da taxa de transmissão quanto da distância que o quadro deve percorrer, e é atribuído automaticamente pelo protocolo.

2.6.3 A Sub-Camada MAC

Ethernet

A Ethernet [30] foi definida como um padrão para redes locais que utiliza o protocolo de acesso ao meio *Carrier Sense Multiple Access/Colision Detect* (CSMA/CD), protocolo este que permite múltiplos acessos de estações ao meio físico compartilhado e detecção de colisão entre tentativas de acesso. Essas características influenciam na definição do quadro Ethernet, fazendo com que determinados campos sejam fundamentais. O quadro Ethernet padrão é mostrado na figura 2.6.

8 octetos	6 octetos	6 octetos	2 octetos	46–1500 octetos	4 octetos
Preâmbulo	End. Destino	End. Origem	Tipo	Dados	CRC

Figura 2.6: *Padrão de Quadro Ethernet*

O *preâmbulo* consiste de 64 bits que auxiliam na sincronização dos nodos. Os endereços de *origem e destino* tem 48 bits que é o tamanho do endereço utilizado pelas interfaces de rede. O *MAC Address* é um endereço único para cada interface, atribuído pelo fabricante da placa de rede e tem o tamanho de 48 bits [27].

Os endereços de origem e destino no meio físico são únicos. Isso se deve ao fato de existir um órgão regulador, o IEEE, que controla a distribuição e venda de faixas de endereços. Para atribuir um endereço Ethernet a uma placa de rede, os fabricantes compram blocos de endereços e atribuem-nos seqüencialmente aos seus produtos. Dessa maneira, não deveria haver duas interfaces Ethernet com o mesmo endereço físico[6].

O campo *tipo* do quadro contém um inteiro de 16 bits que identifica o tipo de dados que estão sendo transportados no quadro. Por exemplo, o tipo $0x0800$ indica que um datagrama IP está sendo transportado pelo quadro Ethernet. Quando o quadro chega a uma estação, o sistema operacional utiliza o campo tipo para identificar qual protocolo deve processá-lo. Esse campo identifica o quadro Ethernet e o conteúdo por ele transportado. Por fim, o campo *CRC* de 32 bits é usado na verificação de integridade do quadro recebido.

2.6.4 Comparação dos protocolos LLC e TCP/IP

A seguir é estabelecida uma correlação entre as características do TCP/IP e do LLC/MAC, de maneira a esclarecer como é possível, dentro dos limites de uma rede local, substituir os protocolos TCP/IP pelo protocolo LLC/MAC. A tabela 2.7 mostra uma comparação entre o modelo estratificado dos protocolos empregados na Internet e a implementação de ambos os protocolos (LLC e TCP/IP) no kernel do Linux.

[6]Embora saiba-se de casos onde o mesmo endereço MAC foi atribuído a duas placas diferentes.

<table>
<tr><th colspan="2">Modelo TCP/IP
Estratificado</th><th colspan="2">Implementação no
Kernel do Linux</th></tr>
<tr><th>Camada</th><th>Protocolo</th><th>TCP/IP</th><th>LLC</th></tr>
<tr><td colspan="2">Aplicação</td><td colspan="2">Aplicação</td></tr>
<tr><td>Transporte</td><td>TCP</td><td>Soquete TCP</td><td></td></tr>
<tr><td>Rede</td><td>IP</td><td>Soquete IP</td><td></td></tr>
<tr><td rowspan="3">Enlace
MAC</td><td>LLC</td><td></td><td>Soquete LLC</td></tr>
<tr><td rowspan="2"></td><td colspan="2">Device Driver</td></tr>
<tr><td colspan="2" rowspan="2">Dispositivo de Rede</td></tr>
<tr><td>Física</td><td></td></tr>
</table>

Figura 2.7: *Comparação Entre o Modelo Estratificado e a Implementação no Kernel*

No tocante ao endereçamento de serviços através da interface de soquetes, o TCP/IP emprega pares <endIP, porta> para endereçar as duas pontas de um enlace lógico, enquanto que LLC/MAC emprega pares <endMAC, SAP>. Para cada *endereço IP* de uma estação, utiliza-se o *endereço MAC* da interface de rede associada àquela estação. Para cada *porta* de serviço TCP, é utilizada uma *porta de acesso a serviço* do protocolo LLC, mantidas as devidas restrições apontadas na tabela 2.5).

Funcionalidades	**TCP/IP**	**LLC**
Endereçamento de pacotes	inter-redes	rede local
Garantia de integridade	checksum dos cabeçalhos TCP/IP	CRC do quadro Ethernet
Largura das janelas deslizantes	2^{32}	2^{8}
Número de portas	65536	128
Tamanho máx. do pacote de dados	65.535 bytes	depende do MTU

Tabela 2.5: *Comparação Funcionalidades TCP/IP e LLC*

Embora o protocolo TCP/IP possa ser usado em praticamente qualquer tipo de rede, este estudo limita-se à rede local, uma vez que a estrutura predominante nos *clusters* de computadores é a rede local. A garantia de transferência íntegra dos dados de ambos os protocolos de uma estação à outra decorre do cálculo do CRC do quadro Ethernet. Assim, o cálculo do *checksum* dos cabeçalhos TCP/IP é redundante (em se tratando de redes locais) porque a taxa de erros em redes Ethernet é extremamente baixa.

Para aumentar a vazão obtida na transmissão de dados, ambos os protocolos implementam a técnica de janelas deslizantes, embora os tamanhos de janela sejam muito diferentes.

Enquanto que no TCP/IP uma *porta* se refere a uma *porta lógica*, no LLC essa definição é utilizada para *porta de acesso a serviço*. Independente da nomenclatura utilizada, ambas as portas são utilizadas para multiplexação de serviços nas camadas superiores.

No TCP/IP é possível enviar um pacote de tamanho maior que o MTU da rede, porque a fragmentação dos pacotes é feita transparentemente pelo protocolo IP. Usando o LLC, o tamanho máximo de um pacote depende do MTU da rede utilizada, descontados os tamanhos do cabeçalho do protocolo físico e do LLC. Na implementação do LLC no kernel do Linux, o próprio protocolo LLC implementa a fragmentação e controle de envio dos dados.

2.7 Implementação do Protocolo LLC no Kernel do Linux

O protocolo LLC está implementado no kernel do Linux desde 2001. Naquela versão não existia a possibilidade de criar um soquete do protocolo LLC em espaço de usuário. Por isso, o protocolo LLC era utilizado somente por outros protocolos presentes no kernel do Linux, como por exemplo o protocolo X25.

No período de 2002 a 2003, Arnaldo Carvalho de Melo ficou responsável pelo desenvolvimento do código do protocolo LLC no kernel do Linux quando escreveu o código necessário para utilizar o protocolo LLC nível 2 a partir de um socket em espaço de usuário, juntamente com outras propostas de mudanças na estrutura de rede [13].

Embora essas propostas de mudanças não tenham sido integradas totalmente na árvore principal do kernel, seu código está disponível na árvore de código de Arnaldo C de Melo, em `http://www.kernel.org/git/?p=linux/kernel/git/acme/net-2.6.17.git;a=summary`.

Neste trabalho utilizou-se a versão daquela árvore para o kernel *2.6.14-rc2*. O protocolo LLC implementado no kernel é o protocolo LLC tipo 2, cujas funcionalidades se assemelham àquelas dos protocolos TCP/IP. Por conta da sua definição, o protocolo LLC tipo 2 é implementado no *kernel* através de um *stream socket*, que transmite dados através de um circuito virtual.

A implementação do LLC contém algumas funcionalidades adicionais, tal como a fragmentação de dados. Por outro lado, devido à definição no protocolo da ordem dos bits do campo de controle do PDU, mostrada na figura 2.8, mais especificamente devido ao bit *P/F*, na implementação atual do protocolo somente as portas de acesso a serviço (*SAP*) com número par podem ser utilizadas. Na figura 2.8 *N(S)* representa o número da sequência de envio e *N(R)* o número da sequência de recebimento.

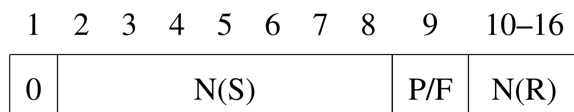

Figura 2.8: *Information transfer command/response (I-format PDU)*

Durante a fase de testes dos aplicativos que empregam a biblioteca OPENMPI, foi descoberta uma falha na implementação do protocolo. Essa falha não permitia que fossem abertas mais de 16 conexões com a mesma máquina. O código com a correção desta falha foi agregado ao código do kernel em `http://www.kernel.org/git/?p=linux/kernel/git/acme/net-2.6.17.git;a=summary`.

2.7.1 Acesso às estruturas do soquete LLC no Kernel do Linux

Como a estrutura do soquete LLC presente no *kernel* modificado não é acessível diretamente através da biblioteca padrão do sistema (glibc), é necessário utilizar arquivos de cabeçalho específicos, que incluem as informações necessárias ao uso do protocolo LLC.

Em conseqüência do fato de o protocolo LLC não ser suportado na glibc, algumas funções de auxílio ao soquete tiveram que ser desenvolvidas à parte.

As seguintes funções foram desenvolvidas por Arnaldo Carvalho de Melo para servir de suporte à utilização do protocolo LLC.

- *int mygetaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res)*

  Implementa uma interface para chamada da função *llcgetaddrinfo* somente quando o protocolo LLC estiver sendo utilizado.

- *int llcgetaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res)*

  Retorna o endereço MAC do destino com base nas informações fornecidas.

- *int mygetnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags)*

  Implementa uma interface para a chamada da função *llcgetnameinfo* somente quando o protocolo LLC estiver sendo utilizado.

- *int llcgetnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, size_t hostlen, char *serv, size_t servlen, int flags)*

  Retorna o nome do *host* associado ao endereço fornecido.

- *static void get_hwaddr (char *name, struct ether_addr *addr, int *type)*

  Retorna o endereço MAC da interface de rede passada como parâmetro.

A estrutura de soquete utilizada na codificação da versão LLC dos programas desenvolvidos é mostrada na figura 2.9:

```
struct sockaddr_llc {
 sa_family_t     sllc_family; /* AF_LLC */
 sa_family_t     sllc_arphrd; /* ARPHRD_ETHER */
 unsigned char   sllc_test;
 unsigned char   sllc_xid;
 unsigned char   sllc_ua; /* UA data, only for SOCK_STREAM. */
 unsigned char   sllc_sap;
 unsigned char   sllc_mac[IFHWADDRLEN];
 unsigned char   __pad[__LLC_SOCK_SIZE__ -
                    sizeof(sa_family_t) * 2 -
                    sizeof(unsigned char) * 4 - IFHWADDRLEN];
};
```

Figura 2.9: *Estrutura utilizada para definir o soquete LLC*

Devido ao projeto do protocolo LLC o número de portas de acesso a serviço (SAP) é limitado à 127 portas, entretanto, devido à ordem dos bits do quadro LLC e também devido à implementação utilizada, esse número de portas é reduzido para 64 portas, onde somente as portas pares podem ser utilizadas.

Uma característica da implementação do protocolo LLC é que a interface de *loopback* usando o protocolo LLC não foi implementada. Sendo assim, não existe nesta implementação, a possibilidade de estabelecer uma conexão local utilizando o protocolo LLC.

CAPÍTULO 3

TRABALHOS RELACIONADOS

Em um sistema operacional com uma arquitetura eficiente, o sistema de mensagens é fornecido pelo núcleo do sistema operacional, que garante acesso protegido à interface de rede, transpondo os limites do espaço de usuário e espaço de *kernel* utilizando chamada de sistema para efetivar a comunicação. Freqüentemente os serviços de comunicação são acessados através de um pequeno conjunto de chamadas de sistema leves.

Essas chamadas de sistema em conjunto com a definição de um protocolo de comunicação, servem de base para os trabalhos discutidos no que se segue.

GAMMA O GAMMA (*Genoa Active Message MAchine* [32–34] é um conjunto de chamadas de sistema leves associado a uma biblioteca de programação em nível de usuário. O GAMMA foi desenvolvido substituindo toda a camada de comunicação de dados desde o espaço de usuário até inclusive o *device driver* do *kernel* do Linux. Dessa maneira, todas as chamadas de sistema para comunicação invocadas por programas que utilizem o GAMMA são desviadas do fluxo normal do núcleo do sistema operacional e passam pelo código do GAMMA que copia os dados diretamente do espaço de usuário para o buffer de transmissão da interface de rede, enfileirando e tratando as requisições na ordem de chegada.

Algumas características do GAMMA:

- Mensagens maiores que 110 bytes são fragmentadas em uma seqüência de quadros Ethernet com tamanhos variando de 60 a 1536 bytes [34].
- A camada de comunicação GAMMA é inserida no *kernel* do Linux na forma de chamadas de sistema e estruturas adicionais. Essas chamadas de sistema são encaminhadas para um

endereço específico do vetor de interrupções que desvia o fluxo do código para o código GAMMA. O uso de um endereço específico limita a portabilidade do sistema para outras arquiteturas que porventura utilizem esse endereço para alguma outra finalidade.

- O protocolo GAMMA não fornece nenhum serviço de aviso de recebimento de quadros porque o custo em termos de perda na largura de banda é alto e a probabilidade de corrupção de um quadro Ethernet é muito pequena. Por outro lado, emprega um CRC nos dados trafegados, além do CRC empregado pela Ethernet.

- O programa que faz a recepção dos dados é um *device driver* especialmente customizado para utilizar o GAMMA. Essa abordagem implica necessariamente em se ter um *device driver* específico para um determinado adaptador de rede, limitando assim a utilização de vários tipos de interfaces de rede.

- O GAMMA utiliza um sistema de comunicação que permite um máximo de 256 processos paralelos diferentes ativos em um único nodo. Essa limitação, imposta no dispositivo impede uma escalabilidade maior do sistema.

- Cada quadro transmitido tem um cabeçalho de 20 bytes, e o tamanho mínimo de um quadro transmitido é de 60 bytes, sendo que, no caso da transmissão de uma quantidade de dados que seja menor que 40 bytes os bytes restantes do quadro serão preenchidos com zeros até o tamanho mínimo do quadro.

- O Gamma é somente uma camada de *Active Messages* e não um ambiente MPI completo. Em [33] é apresentada uma implementação da biblioteca MPICH sobre o GAMMA, conferindo assim uma versão com um ambiente MPI completo sobre o GAMMA.

CLIC O CLIC (*Communication for parallel programming on LInux Cluster*) [35–37] se utiliza de chamadas leves do sistema para substituir a camada de comunicação de dados (soquetes e protocolos TCP/IP), visando fornecer uma alternativa eficiente na comunicação de dados do sistema operacional.

O CLIC implementa uma camada de comunicação de dados que permite uma comunicação eficiente dos computadores em um cluster, tendo como restrição a portabilidade do sistema entre as diferentes plataformas suportadas pelo Linux [35]. Essa restrição de portabilidade permite que seja utilizado o *device driver* presente no *kernel* do Linux.

As principais características do projeto CLIC são:

- É embutido no kernel do Linux fornecendo uma interface de comunicação alternativa às aplicações dos usuários.
- A melhoria na comunicação decorre da redução do número de camadas de protocolo, o que reduz a sobrecarga de software e o número de cópias de dados efetuadas.
- O CLIC consiste de um protocolo de transporte confiável que interage com a interface de rede utilizando o *device driver* presente no *kernel*.

Como pode-se observar, tanto no GAMMA quanto no CLIC utilizam a abordagem de substituir inteiramente a camada de comunicação do kernel do Linux, removendo a interface de soquetes e os protocolos TCP/IP do caminho do fluxo de dados. O CLIC, por ter uma premissa de portabilidade, não modifica o *device driver* da interface de rede, enquanto que o GAMMA, na tentativa de reduzir ainda mais o número de cópias de dados modifica essa camada. A inserção destes componentes no *kernel* é trabalhosa e demanda cuidado e atenção. A cada versão nova do kernel esse trabalho deve ser re-feito aplicando-se as atualizações do sistema no kernel e vice-versa. Por se tratar de uma abordagem bastante intrusiva, fica confinada especificamente às aplicações de clusters de computadores aonde a especialização tem seu custo reduzido devido aos ganhos em desempenho.

“Em um cluster de computadores aonde as máquinas estão na mesma rede, não é necessário utilizar o protocolo IP para rotear os pacotes” [35].

Este trabalho emprega uma abordagem diferente daquelas do GAMMA e CLIC. O presente trabalho mantém intacta a camada de soquetes e funções do sistema operacional para acesso aos dispositivos, e substitui os protocolos TCP/IP pelo protocolo LLC. Dessa maneira, utilizando

um protocolo mais simples, porém eficiente para redes locais, é possível obter um ganho de desempenho na transmissão de dados. Cria-se assim uma alternativa de baixo custo, eficiente e portável para transmissão de dados em uma rede local. Note-se que o escopo de aplicação deste trabalho não é limitado a uma rede local, mas pode ser usado em qualquer aplicação cujo escopo seja um aglomerado de computadores que empregue uma rede local baseada nos padrões IEEE *802.2* e *802.3**.

CAPÍTULO 4

BIBLIOTECAS PARALELAS

4.1 Bibliotecas Paralelas MPI e PVM

A troca de mensagens é o método de comunicação baseado no envio e recebimento de mensagens através de uma rede de computadores seguindo as regras de um protocolo de comunicação.

Os padrões de troca de mensagens MPI [1] e PVM [2, 38], são os padrões mais utilizados na atualidade como *frameworks* para a criação e execução de aplicações paralelas.

A especificação PVM foi desenvolvida inicialmente por um único grupo de pesquisas, o que permitiu uma grande flexibilidade no projeto e também uma resposta incremental às experiências da comunidade de usuários. Além disso, o grupo que implementa a biblioteca PVM é o mesmo grupo que define o projeto, dessa maneira ocorre uma rápida interação entre o projeto e implementação do PVM.

Em contrapartida, a especificação MPI tem seu projeto definido pelo *MPI Forum* (que agrega programadores e usuários finais), praticamente independente de qualquer implementação específica.

4.1.1 Diferenças entre MPI e PVM

No PVM quem define as características de cada implementação é o implementador, sendo este livre pra incluir funcionalidades extras; sendo assim uma implementação PVM pode fornecer determinadas características em apenas algumas arquiteturas. Ao contrário do PVM, no MPI, uma vez definido que determinada característica está no padrão, todas as implementações deverão fornecê-la.

Provavelmente a maior diferença entre os padrões PVM e MPI se encontra no tratamento de informações de recursos que são utilizados pra determinar aonde será criado um novo processo. Essa diferença será explicada a seguir.

O PVM, através da sua máquina virtual implementada a partir de *daemons* PVM, fornece um sistema operacional distribuído. Em contrapartida, o MPI não fornece uma máquina virtual, mas sim uma maneira (através do objeto MPI *MPI_Info*) de se comunicar com qualquer mecanismo que esteja fornecendo os serviços do sistema operacional distribuído.

Ambas as bibliotecas fornecem meios de comunicação entre os processos, porém utilizando-se de diferentes abordagens. Uma situação na qual essa comunicação entre processos se faz necessária é na hora de definir, num sistema distribuído, quais recursos serão utilizados para uma determinada aplicação, dadas as necessidades de execução desta aplicação. Como informar aos nodos das necessidades que a aplicação precisa e como gerenciar esses recursos? Algumas escolhas se mostram possíveis: (a) separar um pequeno conjunto de recursos que todos os sistemas possam suportar, (b) definir um sistema genérico, ou (c) fornecer uma interface que permita que as informações dos recursos que a aplicação necessita sejam enviadas de uma maneira específica para o sistema.

O PVM optou por utilizar a abordagem (a) por entender ser a maneira mais conveniente para a maioria dos usuários. Embora (b) seja a opção que forneça a maior portabilidade sem sacrificar a expressividade, não é extensível e necessita de uma interface bem definida e acordada entre os usuários. Essas características de (b) levaram o *MPI Forum* a optar por utilizar a opção (c).

4.1.2 Características Comuns

Ambos, MPI e PVM são portáveis, sendo a especificação de cada máquina independente, e podem ser encontradas implementações para uma grande variedade de máquinas, em particular aquelas que mais são utilizadas em clusters de computadores.

Uma vez que um sistema é portável, a característica de homogeneidade pode ser associada a ele. Essa característica indica que dois processos em máquinas de diferentes arquiteturas, independente da diferença de *byte order* podem se comunicar. O PVM utiliza as funções *pvm_pack/unpack* e argumentos de tipos de dados, enquanto que o MPI utiliza-se de um argumento genérico *MPI_Datatype* para tratar dessa comunicação.

Uma última característica [39] é a interoperabilidade que se refere à possibilidade de comunicação entre processos ligados com duas implementações completamente diferentes da biblioteca.

4.2 Biblioteca MPI

A MPI é uma biblioteca que dá suporte à execução paralela (utilizando o modelo de troca de mensagens) em ambientes heterogêneos.

Para este estudo, foi escolhido utilizar uma biblioteca de troca de mensagens (MPI) que fornecesse uma abstração da realidade de hardware para que o software interagisse com todo o conjunto de computadores (*cluster*).

Devido à limitação da implementação atual do protocolo LLC não fornecer a interface de *loopback*, a escolha da biblioteca de troca de mensagens obedeceu à essa restrição.

Várias implementações do padrão MPI estão disponíveis na Internet. Dentre as opções, foram avaliadas as seguintes:

MPICH 1.2.7 A implementação do grupo MPICH na versão 1.2.7 pode ser encontrada no endereço `http://www-unix.mcs.anl.gov/mpi/mpich/` foi a primeira versão escolhida para servir de base para a implementação e testes com o protocolo LLC. Como sua implementação usa de maneira intrínseca e necessária um aspecto em que o protocolo LLC é deficitário no presente momento (a conexão *loopback*) não foi possível efetuar o porte dessa versão da biblioteca para LLC.

MPICH2 A versão 2 da implementação da biblioteca MPICH, que inclui o padrão MPI-2 (http://www-unix.mcs.anl.gov/mpi/mpich2/). Essa versão da biblioteca paralela foi descartada porque o código é escrito em várias linguagens, sendo que uma delas não é de domínio deste autor e por utilizar o mesmo princípio de utilização da conexão *loopback* presente na versão 1.2.7 não disponível para o protocolo LLC.

LAM/MPI 7.1.1 Uma versão bem organizada da implementação do padrão MPI, encontrada em http://www.lam-mpi.org/, com farta documentação. Enquanto ainda em fase de estudo dessa biblioteca, um dos desenvolvedores (Brian Barret) sugeriu que fosse utilizada a versão OPENMPI, por ser uma versão mais nova e com mais facilidades de uso que a biblioteca LAM/MPI.

OPENMPI 1.0 Essa implementação é produzida por um consórcio de vários grupos que implementam padrões MPI (FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI), que se reuniram de maneira a agregar suas tecnologias e recursos visando criar a melhor implementação do padrão MPI. Pode ser encontrada em http://www.open-mpi.org/. Como essa biblioteca utiliza um protocolo diferenciado para a conexão *loopback*(que implementa o envio de mensagens através de referências à memória), essa implementação foi escolhida para ser utilizada nesse trabalho.

4.2.1 OPENMPI 1.0

A implementação OPENMPI 1.0 [40, 41] foi o padrão MPI escolhido para ser portado para utilização do protocolo LLC nesse estudo. Essa implementação foi escolhida por diferentes fatores, como por exemplo, modularidade, flexibilidade, versão em constante desenvolvimento e implementação que suporta os padrões MPI versões 1 e 2.

Porém, o fator decisivo para a escolha dessa implementação foi o fato de ela usar um protocolo diferenciado para conexões locais na máquina (especialmente desenvolvido para esse fim e implementado no componente BTL *self*), evitando a utilização de uma conexão *loopback* do protocolo de rede.

4.2.2 Estrutura da Biblioteca OPENMPI

A biblioteca OPENMPI é estruturada da seguinte forma:

OMPI Contém as funções da interface de programação MPI e a sua lógica de suporte.

ORTE *Open Runtime Environment*, é responsável pelo suporte a diferentes sistemas de execução.

OPAL *Open Portable Access Layer*, é parte do código que faz a ligação entre OMPI e ORTE, além de prover funcionalidades adicionais.

As seguintes definições são utilizadas no tocante à estrutura da biblioteca [42]:

MCA A arquitetura modular de componentes (*Modular Component Architecture*) é a base sobre a qual todo o projeto da biblioteca OPENMPI é construída. MCA fornece todos os componentes de serviços que o resto do sistema utiliza. Embora esse seja o coração do sistema, sua implementação é pequena e leve, pois foi concebida para ser pequena, rápida e eficiente. Além disso, oferece alguns outros serviços, como localização, carga e remoção de componentes.

Framework Estrutura que fornece uma interface pública utilizada pelo código externo e também pelos próprios serviços internos. Um *framework* MCA utiliza os serviços MCA para encontrar e carregar componentes em tempo de execução. Um exemplo é o *framework* MPI chamado BTL (*Byte Transfer Layer*), que é usado para enviar e receber dados em diferentes tipos de redes. Por conseguinte, a biblioteca OPENMPI tem componentes BTL para memória compartilhada, TCP, Infiniband, Myrinet, etc.

Componente Um componente MCA é uma implementação da interface de um *framework*. Outra denominação comumente utilizada para componente é "plugin", que é uma porção de código que pode ser inserido na biblioteca OPENMPI, tanto em tempo de execução quanto em tempo de compilação. *Um novo componente BTL foi implementado pelo autor para acrescentar o suporte ao protocolo LLC na biblioteca.*

Módulo Um módulo MCA é uma instância de um componente (instância no sentido de uma instância de uma classe C++). Por exemplo, se um nodo que está executando uma aplicação OPENMPI tem várias interfaces de rede Ethernet, a aplicação OPENMPI conterá um componente BTL TCP, porém vários módulos BTL TCP. Essa diferença entre componentes e módulos é importante porque os módulos tem um estado particular e os componentes não.

Os *frameworks*, componentes e módulos podem ser dinâmicos ou estáticos. Assim sendo, podem estar disponíveis na forma de *plugins* ou ainda serem compilados estaticamente nas bibliotecas.

Frameworks da Biblioteca OPENMPI

Existem três tipos de *frameworks* na biblioteca OPENMPI [4, 40, 41], aqueles que estão na camada MPI (OMPI), os que se encontram na camada de execução (ORTE) e ainda os que estão presentes na camada que fica em contato com o sistema operacional e a plataforma de destino (OPAL). A seguir podemos localizar aonde na estrutura de *frameworks* foi inserido o módulo BTL LLC codificado pelo autor:

Frameworks da camada OMPI

- **allocator** *Memory allocator*
- **bml** *BTL Management Layer (managing multiple devices)*
- **btl** *Byte Transfer Layer (point-to-point byte movement)*
  - **LLC** responsável pelo transporte de dados utilizando o protocolo LLC. Foi desenvolvido um novo componente desse *framework* especificamente para tratar do protocolo LLC.
  - **TCP** responsável pelo transporte de dados utilizando os protocolos TCP/IP.
- **coll** *MPI collective algorithms*

io *MPI-2 I/O functionality*

mpool *Memory pool management*

pml *Point-to-point Management Layer (fragmenting, reassembly, top-layer protocols, etc.)*

rcache *Memory registration management*

topo *MPI topology information*

O *framework* BTL é o responsável por criar a camada de abstração de transferência de dados entre estações ponto-a-ponto. Todas as implementações do *framework* BTL que tratam dos protocolos subjacentes, como por exemplo TCP/IP e LLC, são chamadas de componentes BTL. O autor desenvolveu um componente BTL específico para o protocolo LLC. Sendo que esse componente é descrito a seguir.

4.2.3 Implementação do Componente BTL LLC

A interface de movimentação de bits (BTL) é responsável pela comunicação entre o protocolo MPI e o protocolo de rede subjacente. Essa interface é responsável por comunicar-se com os outros nodos do cluster, independente do meio utilizado. Pelo fato de utilizar-se um novo protocolo (LLC), foi necessário implementar um novo módulo BTL para o protocolo LLC. O BTL LLC é descrito a seguir.

Inicialização

Durante a inicialização da biblioteca, todos os componentes BTL disponíveis são carregados e abertos através das suas funções `mca_base_open_component_fn_t`, onde *fn* refere-se ao protocolo utilizado, por exemplo, llc ou tcp.

A função de inicialização do componente BTL deve registrar junto à camada MCA quaisquer parâmetros MCA utilizados para ajustar o comportamento do componente BTL utilizando as funções:

- `mca_base_param_register_int`
- `mca_base_param_register_string`

Uma característica dessas funções é de que irão falhar no caso da falta de qualquer recurso necessário para utilização do componente BTL não estar disponível. Dessa maneira, por exemplo, caso o acesso ao protocolo LLC não esteja disponível no sistema, o componente não será carregado nem disponibilizado.

A seguir, a função `mca_btl_base_component_init_fn_t` é invocada para cada um dos componentes que foi carregado com sucesso. A função `component_init` retorna um dos seguintes valores:

1. Uma lista nula de módulos BTL, caso o meio (protocolo) de transporte utilizado não esteja disponível.
2. Uma lista contendo um único módulo BTL, que fornece uma camada de abstração sobre múltiplos dispositivos físicos (por exemplo, várias interfaces de rede).
3. Uma lista contendo vários módulos BTL, e cada módulo corresponde a um único dispositivo físico.

Durante a inicialização, o módulo deverá comunicar quaisquer informações de endereçamento requisitadas pelos outros nodos, por exemplo, a porta de escuta aberta tanto pelo módulo TCP como pelo módulo LLC para conexões de entrada. Essas informações são enviadas aos outros nodos utilizando a interface fornecida pela função `mca_pml_base_modex_send`. Embora essas informações possam não estar disponíveis durante a inicialização do módulo, elas estarão disponíveis durante a fase de seleção do componente BTL, executada pela função `mca_btl_base_add_proc_fn_t`.

Seleção do BTL

A camada superior (MCA) cria uma lista ordenada dos módulos BTL disponíveis, ordenados pela sua classificação de exclusividade. Essa classificação relativa é utilizada para determinar o conjunto de módulos BTL que será usado para atingir/comunicar-se com determinado destino. Durante a inicialização dos módulos BTL, a função `mca_btl_base_add_proc_fn_t` de cada módulo é responsável por informar quem é capaz de atingir determinado destino. O módulo BTL com a maior pontuação (essa pontuação é retornada com base na latência obtida pelo módulo na comunicação com o destino) que retornar com sucesso é selecionado. Os módulos subseqüentes são selecionados somente se tiverem a mesma ordem de exclusividade.

BTL	**Exclusividade**	**Comentários**
LO	100	Selecionado exclusivamente para processos locais
SM	50	Selecionado exclusivamente para outros processos na máquina
LLC	0	Selecionado dependendo da acessibilidade através da rede
TCP	0	Selecionado dependendo da acessibilidade através da rede

Tabela 4.1: *Classificação de Exclusividade de Seleção do Componente BTL*

A tabela 4.1 mostra um exemplo de classificação de módulos BTL. Pode-se observar que, para se comunicar localmente, o módulo que será utilizado é o módulo *LO*, uma vez que sua classificação é a maior dentre os módulos listados. Da mesma forma, para acessar uma estação remota, tanto o módulo *LLC* quanto o módulo *TCP* podem ser selecionados uma vez que ambos tem a mesma classificação. É importante observar que essa classificação pode ser mudada pelo usuário para favorecer a utilização de um ou outro módulo e que o valor inicial é obtido a partir do código do módulo BTL.

Quando um módulo BTL é selecionado, ele retorna um ponteiro para uma estrutura de dados `mca_btl_base_endpoint_t` para o *framework* PML. Esse ponteiro é utilizado como um "tratador" (*opaque handler*) pelo *framework* PML e é retornado ao BTL em chamadas de transferências de dados correspondentes ao processo de destino. O conteúdo da estrutura de dados utilizada no módulo BTL é definido exclusivamente em cada implementação e é utilizado para armazenar informações de endereçamento e conexões ativas, como por exemplo, um soquete LLC.

Transferência de Dados BTL

Após selecionado o componente BTL responsável por transmitir os dados, é estabelecido o anel de comunicação do cluster. Após estabelecido o anel, todos os módulos trocam informações necessárias para estabelecer a comunicação fim-a-fim de um nodo a outro no *cluster*. Após essa troca de informações, todas as comunicações em nível MPI ocorrem de maneira transparente.

Finalização do BTL

Ao término da execução do programa MPI, os módulos BTL são informados para finalizarem. Durante o processo de finalização, o anel de comunicação é desfeito e todos os recursos alocados anteriormente (estruturas de dados, etc) são desalocados.

Comentários a Respeito do Componente BTL

Devido ao fato de o protocolo LLC utilizado (nível 2) operar de maneira semelhante ao protocolo TCP, a estrutura base da implementação do componente BTL LLC seguiu a estrutura existente do componente BTL TCP, observadas as necessárias modificações a respeito de endereçamento e características próprias de cada protocolo. Desta maneira, utilizando uma estrutura comum, fica fácil estabelecer uma comparação entre os protocolos apresentados utilizando aplicações que empreguem a biblioteca OPENMPI. Uma descrição mais detalhada da implementação do componente BTL LLC pode ser encontrada no anexo A.2.

CAPÍTULO 5

AVALIAÇÃO DE DESEMPENHO

Para avaliar a implementação do componente BTL LLC e a implementação do protocolo LLC no *kernel*, optou-se por executar aplicações que empregam a biblioteca OpenMPI. Essas aplicações compreendem dois grupos distintos: dois *benchmarks* de rede (NETPIPE e MPPTEST) e aplicações computacionalmente intensivas, uma com maior número de troca de mensagens (*implementação da transformada rápida de fourier*) e a outra com mais processamento do que comunicação (*radix sort*). Estas são brevemente descritas a seguir.

NETPIPE – *Network Protocol Independent Performance Evaluator* É uma ferramenta desenvolvida para representar visualmente o desempenho da rede sob uma grande variedade de condições [43]. Essa ferramenta executa um teste ping-pong entre duas máquinas, enviando mensagens cujos tamanhos crescem gradativamente, seja entre dois processos que estão em máquinas separadas por uma rede, seja em uma máquina multiprocessada. Cada ponto amostrado envolve vários testes ping-pong[1] executados para medir com precisão o tempo.

MPPTEST – *Measuring MPI Performance* O programa MPPTEST [44] tem a finalidade de medir o desempenho de algumas rotinas de troca de mensagens do padrão MPI em uma grande variedade de situações. Além do teste ping-pong, pode ser medido o desempenho com vários processos participando do teste (expondo assim problemas de escalabilidade e contenção) e pode-se adaptativamente escolher os tamanhos de mensagens de maneira a isolar mudanças bruscas no desempenho.

[1]Em um teste ping-pong, uma mensagem é enviada da máquina A para a máquina B e, ao receber a mensagem, a máquina B devolve-a para a máquina A.

Fast Fourier Transform A implementação da transformada rápida de Fourier [45] utilizada nesse estudo é a mesma utilizada no trabalho [46]. O objetivo aqui é comparar os tempos obtidos na execução da aplicação, com os dois protocolos de comunicação TCP/IP ou LLC, de forma a prover uma base de comparação para aplicações já existentes.

Ordenação Radix O algoritmo de ordenação radix [47] é muito utilizado para ordenar uma grande quantidade de números em poucas passadas sobre os dados. É freqüentemente utilizado em aplicações de banco de dados e aplicações espaciais, e esse algoritmo se apresenta como uma alternativa a outros algoritmos de ordenação de alto desempenho como, por exemplo, *heapsort* e o *mergesort*. Seguindo o mesmo princípio utilizado para o FFT, pretende-se obter uma base de comparação para o desempenho dos dois protocolos.

5.1 Ambiente de Execução dos Testes

Os ambiente de testes utilizado nesse estudo compõe-se de um *cluster* de computadores composto por 16 máquinas homogêneas com as seguintes características:

Hardware

- Processador: AMD Athlon 1.2 GHz
- Memória Cache L1 Instruções: 64KBytes
- Memória Cache L1 Dados: 64KBytes
- Memória Cache L2: 64KBytes
- Memória RAM: 128 MBytes
- Placa de Rede: Realtek 8139 séries C/C+ 100Mbits/s
- Rede: Ethernet
- Switch: DLINK DES-3226 10/100Mbits

Software

- Kernel Linux: 2.6.14-rc2-g8420e1b5 (modificado para suportar o socket LLC)
- OPENMPI: 1.0 stable + código BTL LLC escrito para esse estudo
- configurações específicas para o roteamento de pacotes LLC

5.2 Análise Estatística

Para a análise estatística dos dados amostrais adquiridos com a execução dos testes, utilizamos o teste da diferença entre duas médias utilizando as distribuições *t de Student*.

A hipótese nula (H_0) foi estabelecida como sendo a igualdade das duas médias. A hipótese alternativa (H_1) foi definida então como a diferença entre as duas médias.

Segundo [48, 49], uma maneira de obter uma forte evidência a partir dos dados com relação à afirmação feita pela hipótese testada é utilizar o nível de significância de 1%. Dessa maneira foi escolhido o nível de significância de 1% e o intervalo de confiança de 99% para a execução do teste *t de Student*.

Embora [50] diga que "quando a diferença entre as duas médias é testada com o uso de distribuições t, é necessária a suposição de que as variâncias das populações sejam iguais"; o software estatístico R [51] permite estimar as variâncias independentemente uma da outra, sendo que nesses casos, é utilizada a modificação de Welch para os graus de liberdade do teste *t de Student*.

Do teorema do limite central [50]: *À medida que se aumenta o tamanho da amostra, a distribuição de amostragem da média se aproxima da forma da distribuição normal, qualquer que seja a forma da distribuição da população*. Em [50], é possível encontrar: "Na prática, a distribuição de amostragem da média pode ser considerada como aproximadamente normal sempre que o tamanho da amostra for $n \geq 30$". Para que a distribuição das médias encontradas se aproximasse ainda mais da distribuição normal, foi escolhido um tamanho de amostra $n = 50$.

5.3 NETPIPE

Em um cluster Linux, assim como em qualquer multi-computador, a taxa de comunicação entre processadores é o maior fator limitante para a utilidade do sistema. A taxa de comunicação limita a eficiência de uso do poder de processamento disponível e a capacidade das aplicações de "escalarem" para um número maior de processadores.

O primeiro passo na direção de melhorar o desempenho geral de sistemas que se utilizam de bibliotecas de troca de mensagens é identificar aonde o desempenho está sendo perdido e determinar o por que dessa perda.

O NETPIPE tem com principal objetivo identificar aonde se encontram as ineficiências na comunicação entre processadores e encontrar a sua causa, sendo utilizado para comparar a latência e a vazão atingidas utilizando bibliotecas de troca de mensagens entre as diferentes versões existentes dessas bibliotecas.

Utilizou-se o NETPIPE para comparar a eficiência dos protocolos subjacentes à biblioteca OPENMPI. Dessa maneira, utilizando a mesma biblioteca e variando apenas o protocolo, é possível avaliar o desempenho do protocolo no que se refere à execução de aplicações paralelas.

O NETPIPE executa um teste ping-pong, enviando mensagens de vários tamanhos entre dois processadores. Os tamanhos de mensagem são escolhidos a intervalos regulares de tempo e cada ponto amostrado envolve vários testes ping-pong para fornecer uma medida de tempo precisa.

5.3.1 O Projeto e Metodologia de Testes do NETPIPE

O NETPIPE pode ser dividido em duas partes: um *driver* independente de protocolo e uma seção de comunicação específica para um determinado protocolo.

A seção de comunicação contém as funções necessárias para estabelecer uma conexão, enviar e receber dados e ainda encerrar a conexão. A parte de comunicação é diferente para cada

protocolo. Entretanto, a interface entre o *driver* e o módulo do protocolo permanece a mesma para todos os protocolos. Sendo assim, o *driver* não precisa ser alterado para que seja modificado o protocolo de comunicação subjacente.

Da mesma maneira como o desempenho de um computador não pode ser precisamente descrito utilizando uma computação de um único tamanho de mensagem, tampouco o desempenho de rede pode ser medido com um único tamanho na transferência de dados. O NETPIPE aumenta o tamanho do bloco de dados k de um único byte até o tempo de transmissão exceder 1 segundo. Para tornar os resultados comparáveis entre as diversas plataformas para as quais foi portado, os resultados do NETPIPE são medidos em bits ao invés de bytes.

Para cada tamanho de bloco c a ser testado, três medidas são efetuadas: $c-p$, c e $c+p$ bytes, aonde p é um fator de perturbação que pode ser inserido no pacote de dados, aumentando e/ou reduzindo o tamanho [43]. Nos testes executados, optou-se por não utilizar a perturbação, uma vez que a perturbação invalidaria a comparação com resultados de outros testes, sendo assim $p = 0$.

5.3.2 Resultados

A figura 5.1, que ilustra os resultados do NETPIPE, mostra no eixo x (em escala logarítmica) o tamanho da mensagem em *bits* e no eixo y o tempo em *microsegundos* decorrido para a execução dos testes.

A figura 5.1 mostra o tempo necessário para o envio de um pacote de dados de um nó a outro da rede, executando um teste ping-pong. Nesta mesma figura, encontram-se além das duas linhas dos protocolos LLC e TCP/IP uma terceira linha cujos valores indicam a porcentagem de ganho do LLC em relação ao TCP/IP, qual é a diferença do percentual de tempo em que o LLC é mais rápido que o TCP/IP.

Observa-se que para mensagens de tamanho inferior a 100 bits o ganho do LLC chega perto de 20%, caindo em seguida à medida que o tamanho da mensagem aumenta. É possível observar

que, mesmo para um tamanho de mensagem de 8192 bits (1024 bytes) o protocolo LLC ainda tem um ganho de cinco pontos percentuais.

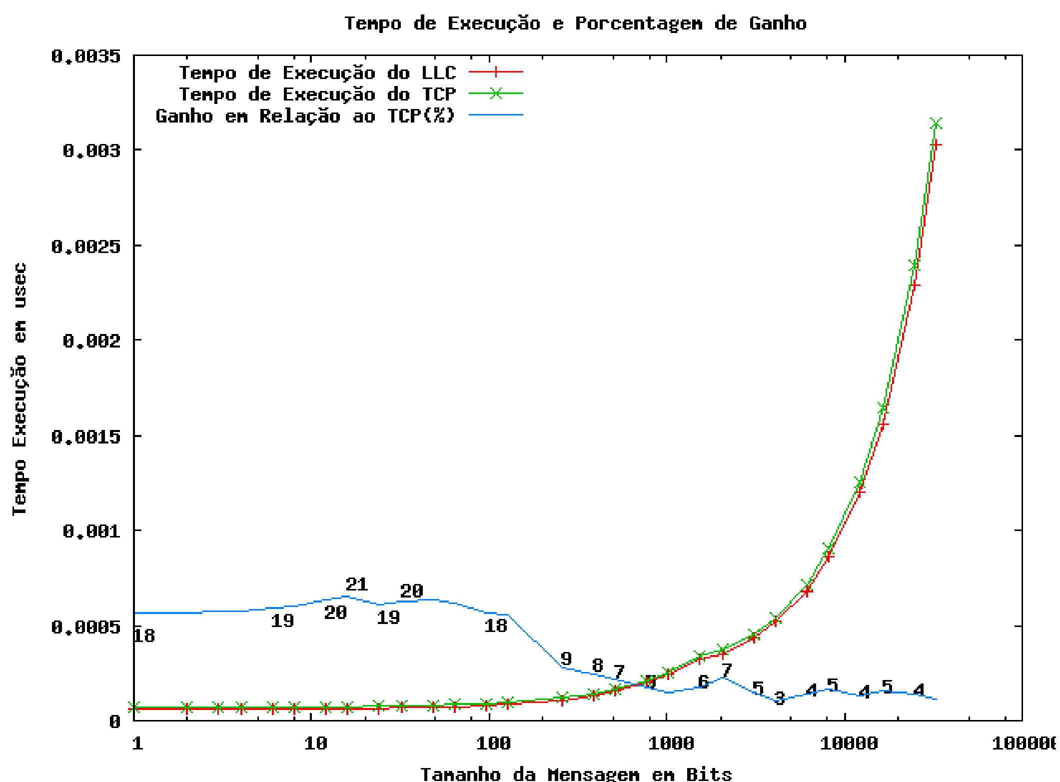

Figura 5.1: *Tempo de Execução*

A figura 5.2 ilustra os resultados do NETPIPE mostrando no eixo *x* (em escala logarítmica) o tamanho da mensagem em *bits* e no eixo *y* a vazão atingida em *megabits/s*.

Observa-se nas figuras 5.1 e 5.2 que o ganho do LLC para mensagens de tamanho inferior a 100 bits é alto com relação às mensagens maiores. Esse fato é explicado porque, ao contrário do TCP, o LLC não emprega um buffer de transmissão de dados. Sendo assim, é possível observar que a partir do momento em que o tamanho da mensagem cresce, preenchendo-se o buffer de saída do TCP mais rapidamente, a porcentagem de ganho do LLC se reduz até estabilizar-se em um patamar ao redor de 5%. Devido ao algoritmo utilizado pelo TCP para otimizar a transmissão de mensagens, e também o fato de o tamanho de buffer utilizado pelo TCP no Linux ser variável, observa-se que para mensagens menores aonde é necessário um maior número de mensagens/dados para preencher o buffer do TCP o ganho do protocolo LLC

é maior. Entretanto, pode-se observar que mesmo com a utilização de mensagens maiores, o protocolo LLC continua sendo mais rápido.

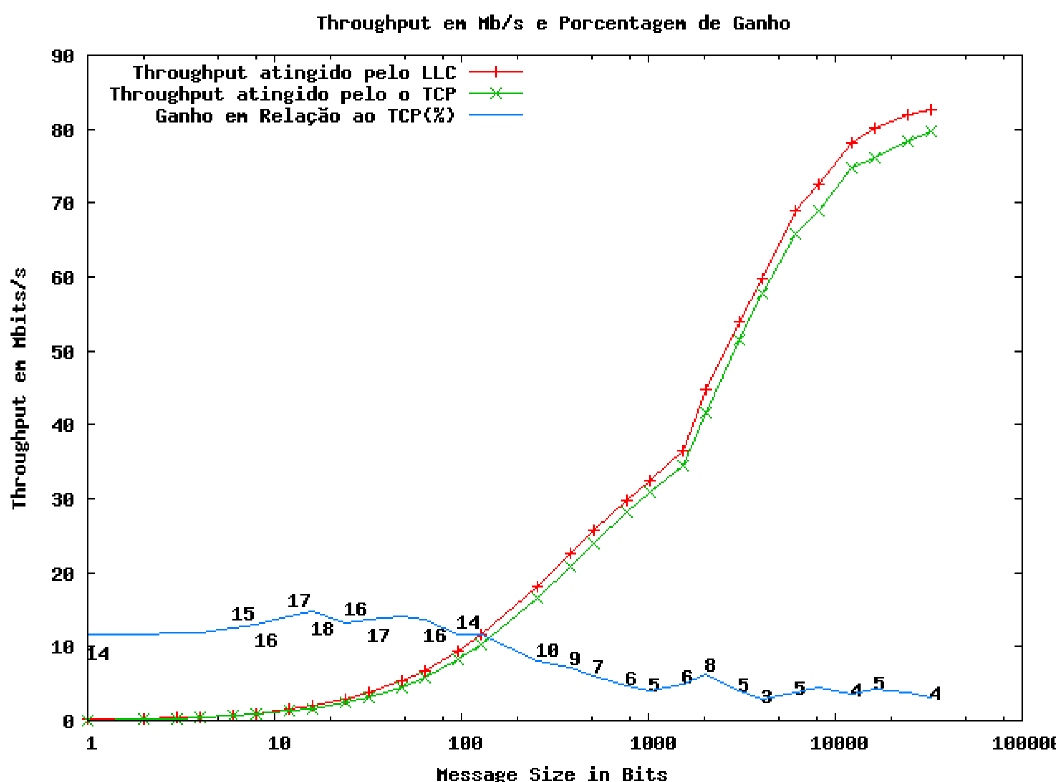

Figura 5.2: *Vazão Atingida*

A tabela A.1, na página 72, contém os resultados do teste *t de Student* com intervalo de confiança de 99% para os tamanhos de mensagens ilustrados nas figuras 5.1 e 5.2. A partir destes dados pode-se concluir que as médias estatísticas dos dois protocolos são diferentes, e portanto o ganho apresentado para o protocolo LLC é significativo e real. Observa-se ainda que o valor de *p* para o teste *t de Student* em todos os tamanhos de mensagens é extremamente pequeno, conferindo maior credibilidade aos resultados. Vale ainda lembrar que todos os valores ilustrados são obtidos a partir da média aritmética de 50 execuções do NETPIPE.

Além do tempo de execução, convém observar a vazão dos dados na rede, uma vez que quanto maior a capacidade de transmissão de dados, maior a eficiência do protocolo. A figura 5.2 ilustra o comportamento da vazão (em megabits por segundo) atingida utilizando-se o NETPIPE.

Assim como na figura 5.1, é possível observar que para tamanhos de mensagens inferiores a 100 bits o ganho do LLC é maior, de aproximadamente 15%, descrescendo à medida em que o tamanho do bloco aumenta, porém permanecendo em um patamar de 5% de ganho para os tamanhos ao redor de 8192 bits (1024 bytes).

É importante notar que os resultados produzidos pelo NETPIPE tem como fator limitante na execução do teste o tempo de envio das mensagens. O NETPIPE tenta enviar o máximo de mensagens possível dentro do intervalo de um segundo. O comportamento observado nas regiões em que os tamanhos de mensagens variam de 100 a 1000 bits e de 5000 a 10000 bits, onde o ganho do LLC reduz-se, é explicado pelo fato de o NETPIPE preencher o *buffer* de envio dos protocolos TCP/IP.

Pode-se observar na tabela A.2, que mostra os principais resultados para o teste *t de Student* com intervalo de confiança de 99%, a informação de que as médias dos dois protocolos são diferentes, e portanto fica validada a conclusão de que o LLC atinge uma vazão maior que o TCP/IP.

Em resumo, destes testes conclui-se que para tamanhos pequenos de mensagens, de até 100 bits, o *buffer* de envio do TCP influencia consideravelmente no tempo de transmissão dos dados, retardando o envio e portanto dando vantagem ao LLC, que não se utiliza de *buffer* de envio de dados. Para mensagens maiores que 100 bits, os ganhos sào da ordem de 3 a 8%.

5.4 MPPTEST – *Measuring MPI Performance*

O programa (MPPTEST [44]) tem a finalidade de medir o desempenho de algumas rotinas de troca de mensagens do padrão MPI. Embora distribuído juntamente com a biblioteca MPICH (uma implementação do padrão MPI), o MPPTEST segue algumas regras previamente estabelecidas com o objetivo de poder comparar as diferentes implementações do padrão MPI; dessa maneira foi possível utilizá-lo para avaliar a implementação OPENMPI.

A regra fundamental da execução dos testes com o MPPTEST é que o teste possa ser repetível, e portanto, a execução do mesmo teste várias vezes deve retornar, com um erro experimental, o mesmo resultado. Um fato bem conhecido é que a execução do mesmo programa pode produzir resultados bastante diferentes a cada vez que ele é executado.

O único tempo que é reprodutível é o tempo mínimo de um número de testes, e essa foi a métrica escolhida para ser medida nos testes do MPPTEST [44]. Por padrão, o MPPTEST executa 30 vezes o mesmo teste, sempre capturando o tempo mínimo. Para igualar aos outros testes executados nesse estudo, modificamos esse número para 50 observações.

5.4.1 Tamanho das mensagens

Obter as amostras de tempo utilizando intervalos de tempo de tamanho regulares pode levar à resultados errôneos, uma vez que deve-se levar em conta o tempo de transmissão dos dados no fio, latência e também a influência que hierarquia de memória acarreta em todo o sistema. O *benchmark* MPPTEST pode escolher automaticamente os tamanhos de mensagens. A regra utilizada pelo MPPTEST é tentar eliminar a artificialidade no gráfico de saída. Essa regra é seguida, calculando-se três vezes os seguintes valores: $f(n_0)$, $f(n_1)$, e $f((n_0 + n_1)/2)$, aonde $f(n)$ é o tempo de envio de *n* bytes. A seguir, o MPPTEST estima o erro na interpolação entre n_0 e n_1 com uma reta, calculando a diferença entre $(f(n_0) + f(n_1))/2$ e $f((n_0 + n_1)/2)$. Se esse valor for maior que um limite de aceitação previamente definido, o intervalo $[n_0, n_1]$ é subdividido em dois intervalos e o teste é repetido. Essa operação pode continuar até que um intervalo mínimo entre tamanhos de mensagens seja atingido.

5.4.2 Escalonamento dos Testes

Os eventos que causam perturbações na medição do tempo de um programa podem demorar alguns milisegundos. Sendo assim, uma abordagem simples que procure pelo mínimo de médias de tempo de pequena duração pode ser obscurecido por um simples evento de longa duração.

Portanto, é importante distribuir os testes para cada tamanho de mensagem sobre o tempo total da execução do teste. O esqueleto de um laço de medição apropriado é mostrado a seguir:

```
for ( num de repetições ) {
   for ( tamanhos ) {
      Mede o tempo de execução desse tamanho
        Se esse tempo medido é o mais rápido aceite-o.
   }
}
```

Figura 5.3: *Laço de Medição do* MPPTEST

É importante observar que essa abordagem só produz um conjunto de dados de resultado ao final dos testes. Entretanto, essa abordagem vem de encontro às boas práticas de teste, aonde executa-se várias vezes o mesmo teste e armazena-se os melhores resultados.

Testes com resultados extremamente díspares, em relação aos resultados vizinhos, são automaticamente refeitos para determinar se esse tempo reflete uma propriedade do sistema de comunicação, ou são resultado de uma carga momentânea do sistema. Essa abordagem também ajuda na obtenção de resultados reprodutíveis.

Observe também que é importante executar várias vezes o laço antes de refinar os intervalos de mensagens. De outra maneira, algum ruído nas medidas pode gerar refinamentos desnecessários.

5.4.3 Execução dos Testes

A seguinte política foi utilizada na execução dos testes: em todos os testes executados, com exceção do teste de sobreposição de computação e comunicação, foi utilizada a opção *-bisect* que informa ao MPPTEST que ele deve utilizar todos os processadores disponíveis, sendo que metade deles enviará os dados para a outra metade. A distribuição das aplicações nos nodos é

feita pela biblioteca OpenMPI e o MPPTEST é quem define a atuação de cada nodo como cliente ou servidor.

Para efetuar uma medida sem utilizar os dados presentes na cache, os testes de comunicação podem executar utilizando bytes sucessivos em um grande *buffer*. Selecionando o tamanho do buffer com um tamanho maior que o tamanho da cache, todas as mensagens se encontrarão localizadas na memória principal.

Para efetuar um teste em que ocorre a sobreposição da comunicação e computação, foi escolhido o tamanho de mensagem de 1500 bytes visando causar o maior impacto possível nos protocolos, de maneira a avaliar o "pior" caso possível, uma vez que o tamanho de 1500 bytes é exatamente o tamanho do MTU da Ethernet.

5.4.4 Análise dos Resultados

Nos gráficos apresentados nesta seção (5.4, 5.5, 5.6, 5.7e 5.9), o eixo x mostra a variação no tamanho da mensagem (em *bytes*) e o eixo y mostra a variação no tempo de transmissão da mensagem de um nodo a outro (em *microsegundos*).

Observando os gráficos 5.4, 5.5, 5.6 e 5.7, que representam o mesmo teste para 2,4,8 e 16 processadores, distingüe-se três regiões com comportamentos distintos:

$[0, 1500]$ **bytes** onde se observa um crescimento quase linear na vazão medida pelo MPPTEST. Esse crescimento pode ser explicado pelo fato de o MTU da *Ethernet* ser de 1500 bytes. Assim, todas as mensagens com tamanho[2] até 1500 bytes cabem em um quadro único, evitando a fragmentação das mensagens em vários quadros.

$[1500, 1800]$ **bytes** esse intervalo mostra uma certa "instabilidade", pelo fato de se estar utilizando tamanho de mensagens com limite inferior muito próximo do *MTU*. Dessa maneira, a mensagem não cabe em um único quadro de dados, sofrendo fragmentação. A quantidade de dados restante para o segundo quadro é pequena em comparação com o

[2]Incluindo dados de controle dos protocolos subjacentes.

tamanho total do quadro, o que eleva o custo da fragmentação para esse segundo quadro, aumentando assim o tempo de processamento relativo ao envio da mensagem.

$[1800, 4200]$ **bytes** nesse terceiro intervalo observa-se que volta a ocorrer o crescimento da vazão. Embora não se trate mais de um crescimento quase-linear como no primeiro intervalo, obtêm-se um crescimento considerável, mas com uma derivada menor. O fato desse crescimento não ser tão acentuado quanto no primeiro intervalo advém da necessidade de se realizar a fragmentação dos dados em vários quadros, aumentando assim o tempo de processamento para cada mensagem e reduzindo a vazão. Em torno de 3000 bytes espera-se um comportamento semelhante ao ocorrido ao redor de 1500 bytes, porém com um degrau menos acentuado.

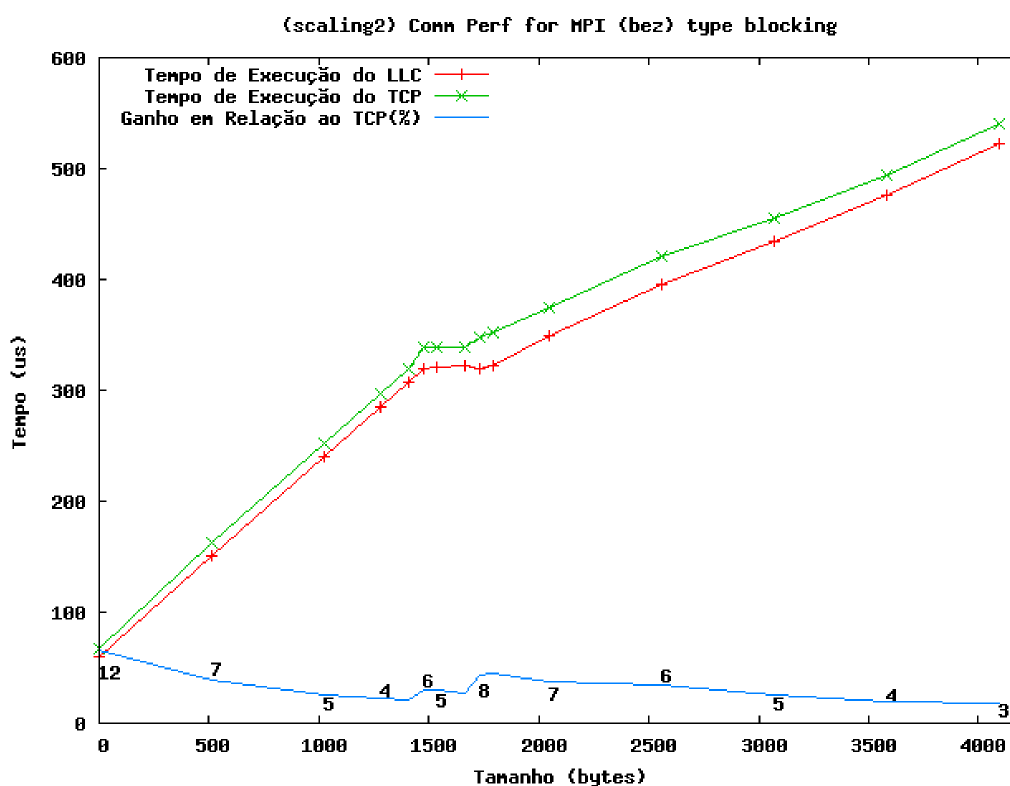

Figura 5.4: *Execução do* MPPTEST *em 2 computadores*

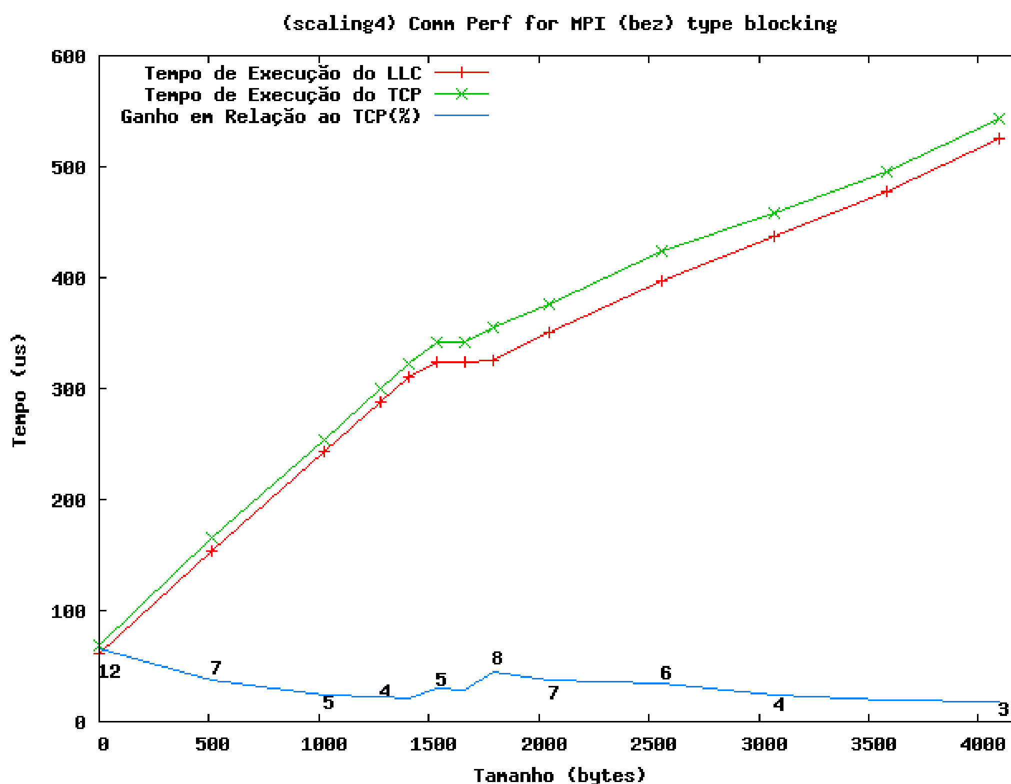

Figura 5.5: *Execução do* MPPTEST *em 4 computadores*

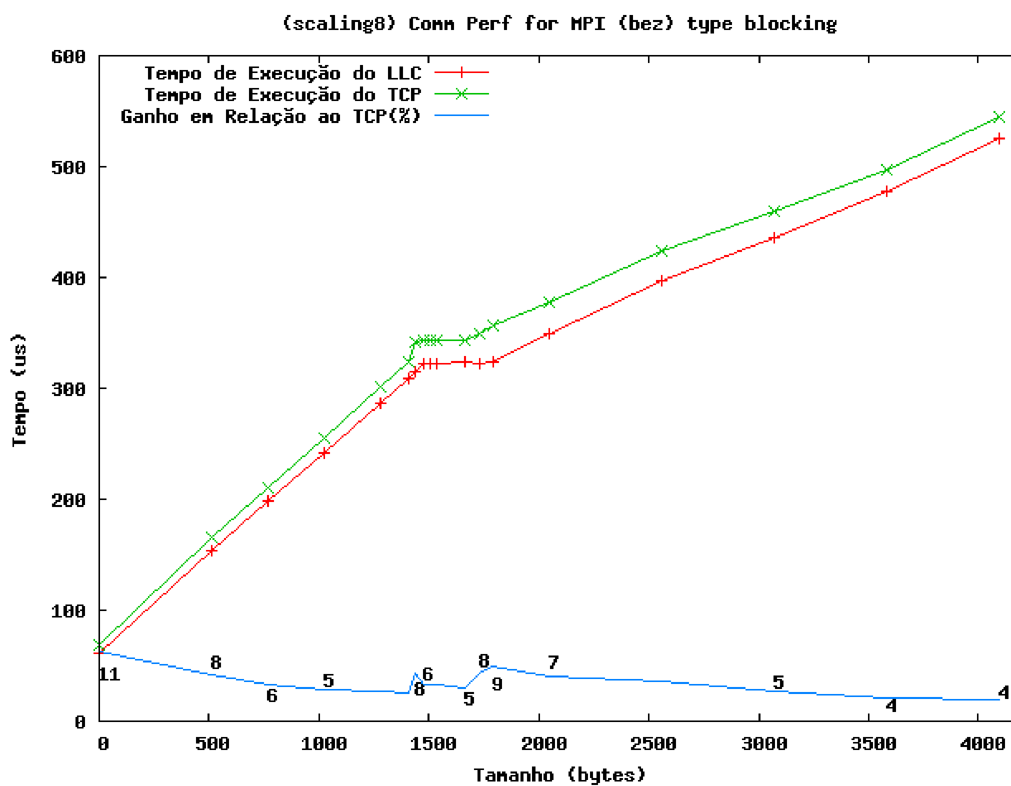

Figura 5.6: *Execução do* MPPTEST *em 8 computadores*

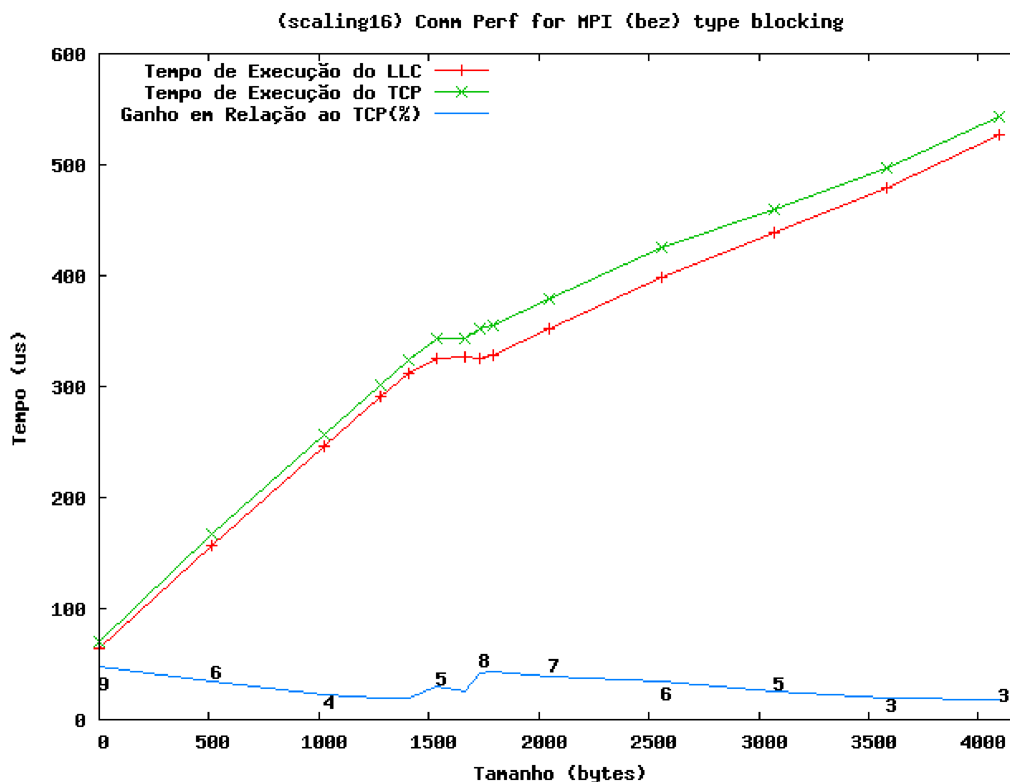

Figura 5.7: *Execução do* MPPTEST *em 16 computadores*

Os gráficos nas figuras 5.4, 5.5, 5.6 e 5.7 mostram que quando se aumenta o número de processadores, o ganho percentual do protocolo LLC com relação ao TCP/IP se mantém. Nos gráficos 5.4 e 5.5 é possível observar que o comportamento dos protocolos é semelhante, independendo do número de processadores.

O gráfico 5.6 explicita uma tendência comum nos testes: a região para tamanhos de mensagens de 1400 a 1800 bytes mostra uma variação mais acentuada no ganho do protocolo LLC. A subida abrupta no ganho de desempenho, de 5% para 8%, ocorre de maneira mais suave nos gráficos das figuras 5.4, 5.5, e 5.7. O ponto de menor ganho nessa região é aquele para tamanho de mensagem de *1408* bytes, e o ponto com o máximo para o tamanho de *1472* bytes. Estes valores são os tamanhos do campo de dados de mensagens que podem ser transportadas pelos protocolos TCP/IP e LLC sem que haja fragmentação das mensagens.

É possível observar outro salto no ganho de desempenho do protocolo LLC que ocorre a partir do tamanho de mensagem *1792*. Esse tamanho de mensagem corresponde ao último ta-

manho comum testado na faixa de valores de tamanhos de mensagens nos quais a fragmentação e a remontagem dos pacotes agregam um custo elevado à transmissão dos dados.

Analisando os gráficos 5.4, 5.5, 5.6 e 5.7, observa-se uma redução no ganho, e portando há convergência nos tempos de execução quando se aumenta o tamanho das mensagens. A figura 5.8 ilustra somente as porcentagens de ganho do protocolo LLC com relação ao TCP/IP.

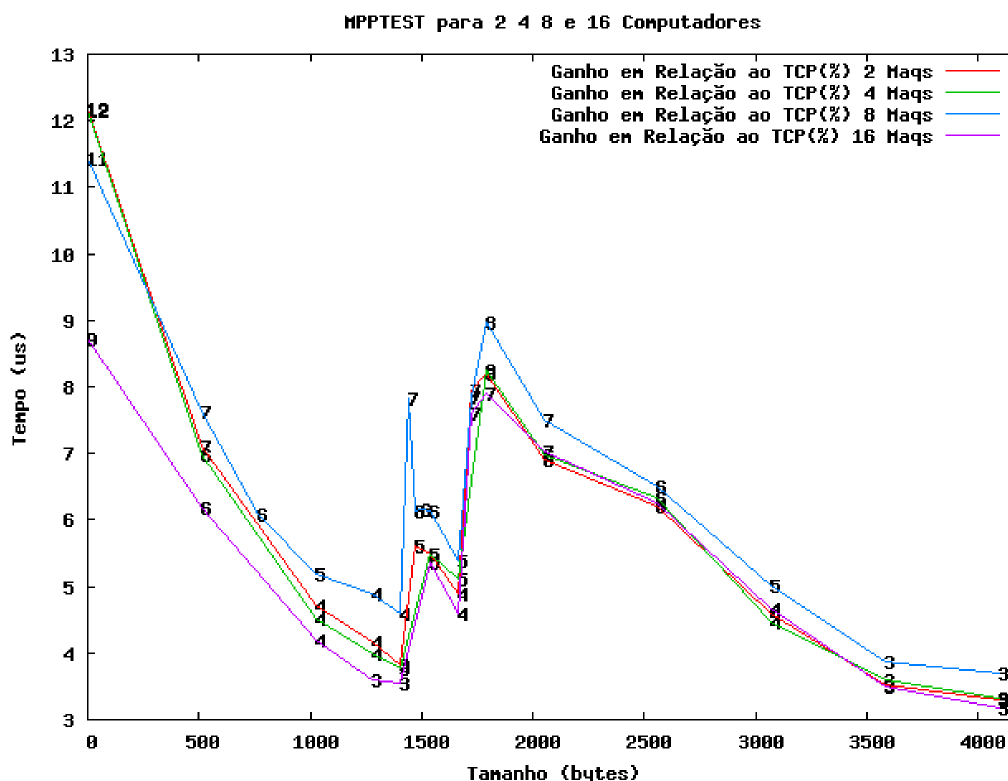

Figura 5.8: *Porcentagens de Ganho para os Testes com 2,4,8 e 16 Computadores*

Ao usar um buffer de dados tamanho maior que o tamanho da memória cache, evita-se que os dados a serem enviados sejam mantidos em cache, permitindo assim avaliar a influência desta no desempenho dos testes [44]. Utilizou-se esse artifício em um teste com dois processadores, cujo resultado pode ser visto no gráfico 5.9. Comparando os gráficos 5.4 e 5.9 é possível observar que a relação entre os tempos de execução dos protocolos LLC e TCP/IP é praticamente a mesma. A diferença entre os dois gráficos é que, no teste que elimina os efeitos da memória cache, os tempos obtidos são 10 microsegundos mais longos em média. Quando se reduzem, ou eliminam, os efeitos dos acertos em cache, o resultado do teste fica uniformemente mais lento.

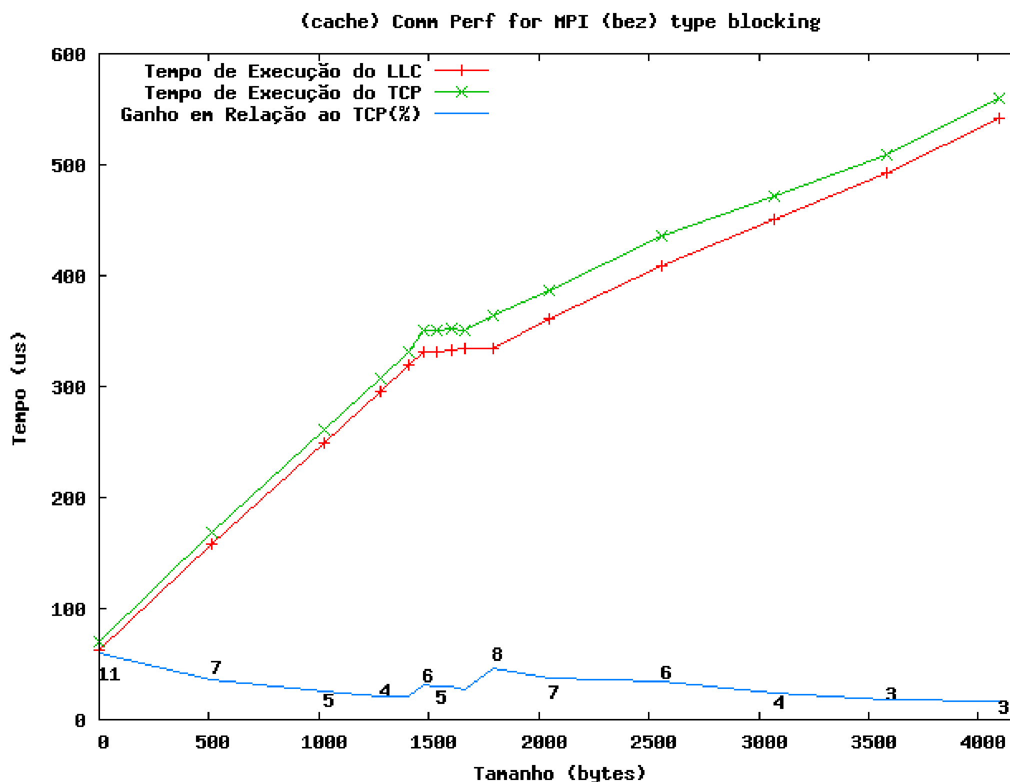

Figura 5.9: *Eliminação os Efeitos da Cache*

5.5 A Transformada de Fourier

As transformadas de Fourier [52] são utilizadas em várias aplicações científicas na física, na teoria dos números, combinatória, processamento de sinais, teoria das probabilidades, estatística, criptografia, acústica, oceanografia, ótica, geometria e outras áreas. Essa grande gama de aplicações decorre de várias propriedades das transformadas, dentre as quais podemos citar

- as transformadas são operações lineares e, com uma normalização apropriada, são unitárias;
- as transformadas são inversíveis;
- utilizando o teorema da convolução, as transformadas de Fourier transformam a complicada operação de convolução em simples multiplicações, tornando-se assim, uma maneira

eficiente de calcular as operações baseadas em convolução, tais como a multiplicação polinomial e a multiplicação de grandes números.

A fórmula para a transformada de Fourier transformar uma seqüência de N números complexos $x_0, \ldots, x_{N-1}$ na seqüência de N números complexos $X_0, \ldots, X_{N-1}$ é mostrada na equação 5.1.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \ldots, N-1 \tag{5.1}$$

No processamento de sinais, as transformadas de Fourier podem isolar os componentes individuais de um sinal (freqüência e amplitude), concentrando-os para uma melhor detecção/remoção. Um grande número de técnicas de processamento de sinais consiste em aplicar a transformada de Fourier à um sinal (áudio ou imagem), manipular os dados resultantes e reverter a transformada. Outra aplicação da transformada de Fourier é na representação compacta de um sinal, por exemplo, a compressão JPEG utiliza a transformada de Fourier de pequenas áreas de uma imagem digital de forma a armazenar uma versão reduzida da imagem original.

5.5.1 O Algoritmo Paralelo da Transformada Rápida de Fourier

O *kernel*[3] FFT (*Fast Fourier Transform*) utilizado nos testes é uma versão complexa e unidimensional do algoritmo descrito em [45].

O algoritmo utiliza-se de um conjunto de dados composto por:

- uma matriz de entrada de *N* números complexos
- uma matriz tamanho NxN, composta de números complexos que representam as *N* raízes complexas do número real 1, sendo esses valores denominados *Raízes da Unidade*.

[3] núcleo

- uma terceira matriz, também de tamanho NxN, utilizada como estrutura intermediária de dados para armazenar os valores já calculados pelo algoritmo.

Tanto a matriz de entrada, quanto a matriz de armazenamento de dados, são lidas e escritas durante a execução do programa. A matriz de raízes complexas é apenas lida, sendo preenchida uma única vez, no início da execução.

A distribuição dos dados entre os processadores ocorre da seguinte maneira: as estruturas a serem enviadas para os processadores destino são subdivisões das matrizes de tamanho *N*, observado o tamanho de sub-matrizes de $\sqrt{N}$x$\sqrt{N}$ elementos.

Os seguintes passos são realizados durante a fase de execução do algoritmo:

1 A matriz de entrada é transposta na matriz intermediária.

2 São realizadas FFTs unidimensionais em cada linha da matriz intermediária, sendo armazenados os resultados na própria matriz intermediária.

3 Aplica-se os elementos correspondentes da matriz de raízes da unidade aos elementos da matriz intermediária.

4 A matriz intermediária é transposta na matriz de entrada.

5 Aplica-se FFTs unidimensionais em cada linha da matriz de entrada, armazenando os resultados na matriz de entrada.

6 A matriz de entrada é transposta na matriz intermediária.

A comunicação entre os processadores ocorre apenas nas fases em que ocorre a transposição da matriz (passos 1,4,6). Existe um bloco de dados que é transposto localmente, (o bloco pertencente ao processador local) sendo que os outros *P - 1* blocos sempre são transferidos para os demais processadores. Durante a fase de transposição, ocorre a comunicação todos para todos entre os processadores, gerando assim um período de tráfego intenso na rede.

5.5.2 Testes e Execução do Algoritmo FFT

Como o principal objetivo do presente estudo é avaliar a interferência dos protocolos das camadas inferiores (*LLC e TCP*), utilizou-se sempre a mesma massa de dados de entrada para o algoritmo; independentemente do número de processadores utilizados na resolução do algoritmo, a matriz de entrada teve sua ordem fixada em 20, por este ser o tamanho máximo de matriz suportado pela memória RAM dos microcomputadores que compõem o *cluster*.

O algoritmo foi modificado para calcular o tempo de processamento em cada um dos processadores em que a computação foi distribuída. Note que o tempo de comunicação do algoritmo FFT está diretamente relacionado ao tempo de processamento, sendo que o tempo de processamento aferido só é finalizado após a última troca de mensagens do algoritmo, onde todos os nodos contém a matriz completa resolvida. Ao final da execução o programa retorna o tempo que cada processador dispendeu no cálculo do FFT. O valor utilizado para o cálculo dos resultados aqui apresentados é composto pela média aritmética dos tempos de execução dos processadores utilizados. Sendo assim, para dois processadores, têm-se como valor utilizado o resultado da média aritmética dos tempos de execução de cada um dos dois processadores. O valor médio foi utilizado porque a variância dos tempos de execução observada é muito pequena.

Para assegurar uma qualidade estatística aos dados obtidos, os testes foram repetidos 50 vezes. Os valores utilizados nos gráficos são correspondentes aos valores da média aritmética dos tempos de execução dessas 50 vezes e podem ser vistos na figura 5.10.

5.5.3 Análise dos Resultados Obtidos

A tabela A.3 no anexo A.1.2 mostra os resultados da aplicação do teste *t de Student*. Pode-se concluir desses resultados que as médias dos tempos dos protocolos LLC e TCP são diferentes, e portanto há ganho real na utilização do protocolo LLC. Esse ganho é variável, de acordo com o número de processadores utilizado. A tabela 5.1 ilustra os ganhos obtidos em porcentagem.

P	Ganho (%)
2	2.74
4	3.29
8	6.65
16	6.81

Tabela 5.1: *Ganhos em Porcentagem do LLC com Relação ao TCP*

A figura 5.10 mostra o tempo de execução do algoritmo FFT para os dois protocolos TCP/IP e LLC, juntamente com o ganho do LLC em relação ao TCP. O eixo *y* mostra o tempo de execução em microsegundos e o eixo *x*, em escala logarítmica, mostra o número de processadores utilizados nos testes.

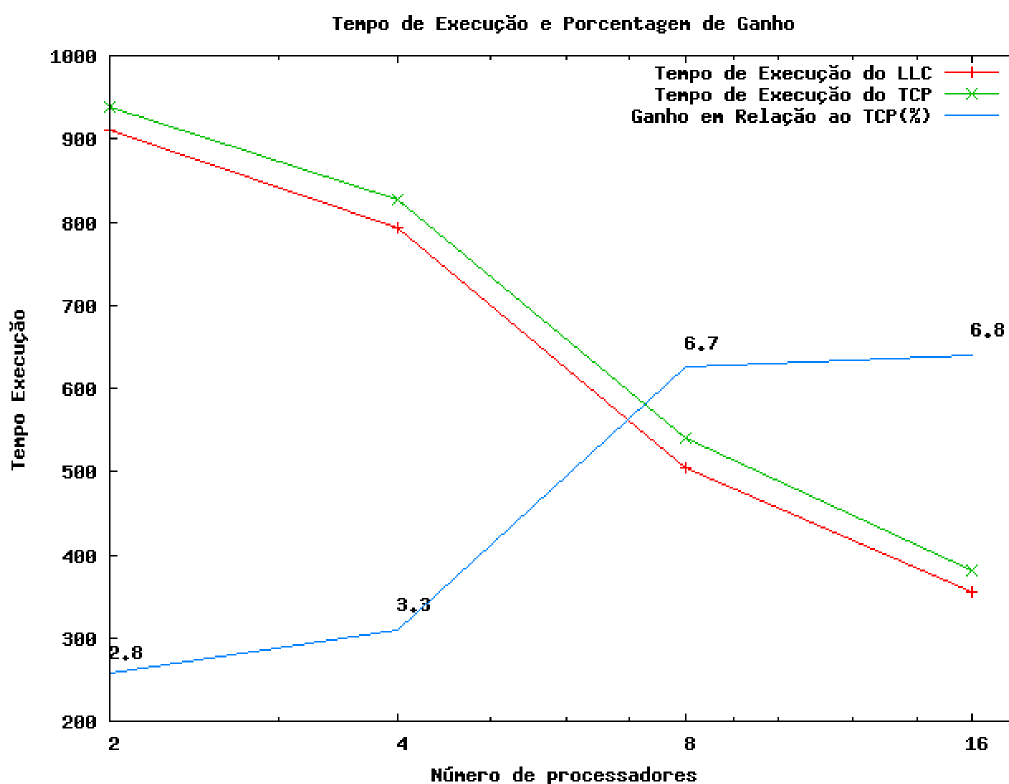

Figura 5.10: *Tempo de Execução do Algoritmo FFT*

Pode-se observar que à medida em que aumenta o número de processadores, para um conjunto fixo de dados (uma matriz de ordem 20), o tempo de execução do programa FFT decresce linearmente, e o ganho relativo do protocolo LLC aumenta consideravelmente. A redução no

tempo de execução é esperada uma vez que ao dividir o problema em partes menores, cada processador resolve mais rápidamente sua parte, restando o tempo de agregar todos os resultados no resultado final.

O ganho obtido pelo protocolo LLC pode ser explicado da seguinte maneira. Para dois e quatro processadores, o ganho decorre da redução no tempo de comunicação porque o protocolo LLC é mais eficiente, e como os conjuntos de dados são relativamente grandes, há intensa comunicação entre os processadores.

Para oito e dezesseis processadores, os conjuntos de dados são relativamente menores, o número de mensagens trocadas é maior mas as mensagens são menores. Portanto, o ganho nestes dois casos decorre da maior eficiência do LLC com mensagens pequenas. O pequeno ganho de oito para dezesseis processadores decorre dos pequenos conjuntos de dados, sendo ultrapassado o ponto ótimo na relação computação/comunicação.

5.6 O Algoritmo de Ordenação Radix

O algoritmo de ordenação radix [47] é um algoritmo de ordenação estável que pode ser utilizado para ordenar ítens que são identificados por chaves únicas. Cada chave é uma seqüência de caracteres (*string*) ou um número, e o "*radix sort*" ordena essas chaves em uma ordem semelhante à lexicográfica. O método de ordenação radix também é conhecido como ordenação postal [47, 53].

O algoritmo de ordenação radix executa em $O(n \times k)$, onde n é o número de itens e k a média do tamanho das chaves. Esse algoritmo foi originalmente utilizado para ordenar cartões perfurados utilizando várias passadas.

O algoritmo consiste de 3 fases:

1. separe o bit/conjunto de bits menos significativo de cada chave.

2. ordene a lista de elementos com base no dígito separado, porém mantenha a ordem dos

elementos com o mesmo dígito (esta é a definição de ordenação estável).

3. Repita a ordenação para cada um dos dígitos/bits mais significativos que restam na chave.

A ordenação executada no segundo passo é feita utilizando o algoritmo *counting sort* [54]. A ordenação *counting sort* inicialmente determina um *ranking* para cada chave a ser ordenada – determina a sua posição na ordem de saída – e depois disso, permuta as chaves para as suas respectivas localizações.

5.6.1 O Algoritmo de Ordenação Radix Paralelo

Inicialmente cada processador do cluster recebe um mesmo número N de chaves a serem ordenadas, e a seguinte seqüência é executada:

1. O processador constrói um histograma local com base nas chaves que possui. Esse histograma local representa a distribuição dos dígitos na porção das chaves em poder do processador.

2. Todos os processadores combinam seus histogramas locais em um histograma global que indica a soma da distribuição total de dígitos nas chaves.

3. Os processadores fazem uso do histograma global para fazer a troca das chaves que cada um possui.

4. As chaves são escritas uma a uma em um vetor e enviadas ao processador de destino.

5. O algoritmo repete as fases anteriores até que todos os dígitos tenham sido verificados e a ordenação tenha sido finalizada.

Um fator importante e que influencia decisivamente no desempenho do algoritmo radix é o valor r utilizado na ordenação. O algoritmo executa uma iteração para cada bloco de *r-bits* nas chaves, o que implica em quanto maior o valor de r, menor o número de iterações necessárias.

Por outro lado, quanto maior o valor de *r*, mais espaço é necessário para alocar as estruturas que armazenam, em memória, os histogramas local e global.

A comunicação no algoritmo radix ocorre principalmente em dois pontos distintos: (1) na permutação de chaves entre os processadores e (2) na combinação dos histogramas locais para formar o histograma global e na redistribuição deste histograma para todas as máquinas.

Durante a fase de permutação das chaves o algoritmo agrupa as chaves destinadas a um mesmo processador em posições consecutivas de memória de forma a possibilitar o envio dessas chaves em uma única transferência de dados, reduzindo assim o tráfego de mensagens na rede e aumentando a eficiência do algoritmo.

5.6.2 Análise dos Resultados Obtidos

No eixo *x* da figura 5.11 é mostrado o número de processadores (em escala logarítmica), e no eixo *y* o tempo de execução do algoritmo radix sort em *segundos*. A figura 5.11 mostra o tempo de execução do algoritmo de radix para 2,4,8 e 16 processadores, com uma massa de *65536 chaves por processador*[4]. Pode-se observar que o tempo de execução para o protocolo TCP/IP cresce à medida em que o número de processadores aumenta. Para o protocolo LLC, observa-se um "degrau" no tempo de execução. Embora o tempo de execução usando protocolo LLC seja maior para 2 e 4 processadores, a relação do tempo de execução se inverte quando o número de processadores passa para oito. Esse fato é explicado pela relação *(tempo de computação x tamanho e quantidade de mensagens)*. Com dois e quatro processadores, o tamanho das mensagens transmitidas é maior e a taxa de comunicação/computação é menor, uma vez que cada processador tem que ordenar um número grande de chaves. Quando se aumenta o número de processadores para oito, obtém-se uma relação *(tempo computação x taxa de comunicação)* mais equilibrada.

Note-se que como a relação computação/comunicação desse algoritmo é maior que 1, a diferença entre os tempos de execução para os protocolos LLC e TCP/IP é mínima porque o

[4]Esse é o maior número de chaves possível de ser ordenado pelo radix no *cluster* de utilizado para o teste.

sistema de comunicação tem pouca influência no desempenho global.

Quando se executa o radix em 2 ou 4 processadores o protocolo LLC se mostra menos eficiente do que o TCP/IP. Esse comportamento pode ser explicado porque nessa configuração o número reduzido de troca de mensagens entre os processadores e as mensagens longas favorecem a estrutura utilizada pelo protocolo TCP/IP. Nesse sentido o TCP/IP leva vantagem porque usa um *buffer* para o envio de mensagens, e as funções de envio de dados retornam mais rapidamente para a aplicação. Quando se aumenta o número de processadores e conseqüentemente cresce o número de mensagens pequenas, o protocolo LLC volta a ser o mais eficiente.

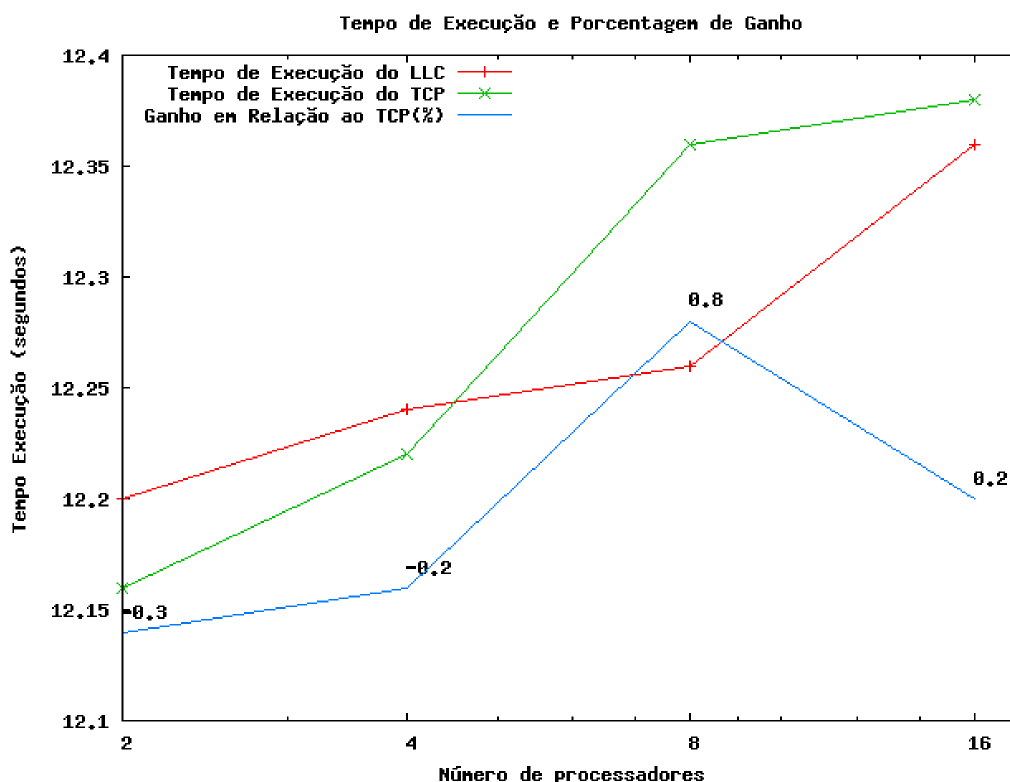

Figura 5.11: *Tempo de Execução do Kernel Radix*

É importante notar que a diferença observada nos resultados entre os protocolos para o radix é insignificante, isso advém do fato de o radix é computacionalmente intensivo e tem seu algoritmo otimizado para evitar a troca de mensagens entre os nodos.

Dada a hipótese de haver uma diferença entre as médias de tempo observadas na execução do algoritmo radix, fez-se necessária a avaliação estatística dos resultados. A tabela A.4 contém

os valores obtidos para a aplicação do teste *t de Student* para as observações/execuções do radix. Analisando-se os resultados na tabela A.4, conclui-se que, embora exista uma diferença entre as médias, essa diferença não é significativa. O *valor p* do teste nos mostra que a hipótese de haver alguma diferença entre as médias foi refutada, uma vez que esse valor é maior que o intervalo indicado para aceitação da hipótese (0.01%).

CAPÍTULO 6

CONCLUSÃO

A utilização de aglomerados (*clusters*) de PCs, interligados por uma rede local tem suprido a crescente demanda por poder computacional em aplicações científicas. Para a programação de aplicações segundo o modelo de troca de mensagens em multicomputadores, são utilizadas bibliotecas de troca de mensagens, como MPI e PVM.

Os protocolos TCP/IP são utilizados como meio de transporte para a troca de mensagens utilizada nas bibliotecas de comunicação. Por conta da incerteza a respeito da rede na qual é empregado, o protocolo TCP incorpora uma série de mecanismos para garantir a entrega confiável das mensagens, bem como para manter a rede operando em condições razoáveis de tráfego e sem congestionamento.

Se a estrutura de rede na qual os dados trafegam for conhecida e restrita a uma rede local Ethernet, o que acontece em muitas instalações de aglomerados de computadores, o protocolo de enlace da Ethernet (o padrão IEEE 802.2 – *Logical Link Control - LLC*) pode ser utilizado para substituir os protocolos TCP/IP. O protocolo LLC incorpora uma série de funcionalidades que são similares ou equivalentes a um sub-conjunto das funcionalidades do TCP, porém limitadas a uma rede local.

Por ser um protocolo mais simples e restrito que o protocolo TCP, é de se supor que um aglomerado de PCs construído sobre uma biblioteca que utilize LLC como seu protocolo de transporte seja mais eficiente —no ambiente de rede local Ethernet— do que um aglomerado baseado na mesma biblioteca mas com TCP como seu protocolo de transporte.

Este trabalho descreve uma adaptação da implementação da biblioteca MPI para executar com o protocolo LLC em uma rede local Ethernet e compara o desempenho desta

implementação com uma implementação da MPI baseada em TCP/IP. Para a avaliação de desempenho foram utilizados os programas de teste NETPIPE e MPPTEST, além de dois núcleos de aplicativos *FFT* e *ordenação Radix*.

Nos testes executados utilizando o NETPIPE pode-se observar que, independente do tamanho da mensagem e para os valores testados, o protocolo LLC é mais eficiente que os protocolos TCP/IP. O maior ganho no desempenho do LLC ocorre na faixa de mensagens com tamanho até 13 bytes (100 bits), com ganho variando de 16 a 21% com relação ao TCP/IP. Embora o ganho percentual do LLC é menor com o aumento do tamanho da mensagem, esse ganho estabiliza-se na faixa de 3 a 5%, mesmo para mensagens de tamanho de 1500 bytes.

Os resultados do MPPTEST mostram que ao escalar o número de processadores utilizados no teste, o protocolo LLC tem ganho de desempenho que varia na faixa de 3 a 12%, ganho que varia na razão inversa ao tamanho das mensagens.

O mesmo comportamento é observado para os resultados da execução do programa da *Transformada Rápida de Fourier* (FFT), quando o LLC obtém ganhos que variam de 2.8% para 2 processadores a 6.8% para 16 processadores. O ganho de desempenho cresce com o número de processadores, quando o tamanho do conjunto de dados é fixo.

Os resultados com a *ordenação Radix* não apontam ganhos significativos de desempenho para o LLC, porque neste programa o tempo de computação predomina sobre o tempo de comunicação de tal forma que não há demanda significativa sobre o subsistema de comunicação.

De tudo isso pode-se concluir que a utilização do protocolo LLC, no contexto de uma rede local Ethernet interligando um aglomerado de computadores, é válida para as aplicações descritas e pode ocasionar na redução do tempo de execução de outras aplicações que empregam a biblioteca MPI. Note-se também que o escopo de abrangência do protocolo LLC não se limita somente a uma rede pertencente a um aglomerado de computadores, mas toda rede local que empregue os protocolos IEEE *802.2* e *802.3**.

Trabalhos Futuros

Para complementar o trabalho descrito nessa dissertação, seria interessante que fossem efetuados os seguintes trabalhos: (1) verificar a escalabilidade do protocolo LLC para *clusters* maiores do que 16 nós; (2) analisar o impacto que a utilização de dispositivos de rede com capacidade de transmissão em Gigabit tem sobre o comportamento destes protocolos; (3) verificar a influência de pacotes Ethernet "jumbo", com tamanho de 9000 bytes, nos ganhos de desempenho; e (4) efetuar um estudo estatístico do valor de convergência dos tempos de execução dos protocolos para os diferentes tamanhos de mensagens.

APÊNDICE A

APÊNDICE

A.1 Teste t de Student

A.1.1 NetPIPE

tam. msg.	Média LLC	Média TCP	valor p	Interv. confiança de 99%
1	6.41824e-05	7.30252e-05	$<$ 2.2e-16	[-8.991158e-06,-8.694442e-06]
2	6.41826e-05	7.31138e-05	$<$ 2.2e-16	[-9.072253e-06,-8.790147e-06]
3	6.41324e-05	7.33514e-05	$<$ 2.2e-16	[-9.362617e-06,-9.075383e-06]
4	6.41490e-05	7.35074e-05	$<$ 2.2e-16	[-9.5024e-06,-9.2144e-06]
6	6.41482e-05	7.38710e-05	$<$ 2.2e-16	[-9.864924e-06,-9.580676e-06]
8	6.4164e-05	7.4256e-05	$<$ 2.2e-16	[-1.023911e-05,-9.944885e-06]
12	6.42116e-05	7.49508e-05	$<$ 2.2e-16	[-1.088477e-05,-1.059363e-05]
16	6.44202e-05	7.56880e-05	$<$ 2.2e-16	[-1.140664e-05,-1.112896e-05]
24	6.67126e-05	7.69826e-05	$<$ 2.2e-16	[-1.041954e-05,-1.012046e-05]
32	6.76090e-05	7.86864e-05	$<$ 2.2e-16	[-1.122305e-05,-1.093175e-05]
48	7.03372e-05	8.23492e-05	$<$ 2.2e-16	[-1.215650e-05,-1.186750e-05]
64	7.34952e-05	8.55136e-05	$<$ 2.2e-16	[-1.217602e-05,-1.186078e-05]
96	8.00444e-05	9.11892e-05	$<$ 2.2e-16	[-1.128774e-05,-1.100186e-05]
128	8.56804e-05	9.78100e-05	$<$ 2.2e-16	[-1.228539e-05,-1.197381e-05]
256	0.0001089866	1.202912e-04	$<$ 2.2e-16	[-1.145557e-05,-1.115363e-05]
384	0.0001311548	1.427926e-04	$<$ 2.2e-16	[-1.177701e-05,-1.149859e-05]
512	0.0001536584	1.657570e-04	$<$ 2.2e-16	[-1.224281e-05,-1.195439e-05]
768	0.0001986082	2.103432e-04	$<$ 2.2e-16	[-1.189187e-05,-1.157813e-05]
1024	0.0002428900	2.549716e-04	$<$ 2.2e-16	[-1.223447e-05,-1.192873e-05]
1536	0.0003221116	3.414326e-04	$<$ 2.2e-16	[-1.950327e-05,-1.913873e-05]
2048	0.0003498882	3.767446e-04	$<$ 2.2e-16	[-2.697621e-05,-2.673659e-05]
3072	0.0004358784	4.574456e-04	$<$ 2.2e-16	[-2.173278e-05,-2.140162e-05]
4096	0.0005242316	5.424172e-04	$<$ 2.2e-16	[-1.833472e-05,-1.803648e-05]
6144	0.0006817038	7.137708e-04	$<$ 2.2e-16	[-3.22863e-05,-3.18477e-05]
8192	0.0008625420	9.095682e-04	$<$ 2.2e-16	[-4.748224e-05 ,4.657016e-05]

Tabela A.1: *Teste t de Student para o benchmark NetPIPE - Tempo*

tam. msg.	Média LLC	Média TCP	valor p	Interv. confiança de 99%
1	0.1188960	0.1044979	$< 2.2e-16$	[0.01415611,0.01464005]
2	0.2377870	0.2087388	$< 2.2e-16$	[0.02858937,0.02950711]
3	0.3569622	0.3120914	$< 2.2e-16$	[0.04416963,0.04557185]
4	0.4758360	0.4152398	$< 2.2e-16$	[0.0596579,0.0615345]
6	0.7137491	0.6197886	$< 2.2e-16$	[0.09258293,0.09533819]
8	0.9514421	0.8221074	$< 2.2e-16$	[0.1274406,0.1312287]
12	1.426071	1.221737	$< 2.2e-16$	[0.2015675,0.2071005]
16	1.895235	1.613080	$< 2.2e-16$	[0.2786975,0.2856112]
24	2.745259	2.378941	$< 2.2e-16$	[0.3609405,0.3716968]
32	3.611731	3.103241	$< 2.2e-16$	[0.5017876,0.5151938]
48	5.207383	4.447707	$< 2.2e-16$	[0.7504896,0.7688626]
64	6.645139	5.710832	$< 2.2e-16$	[0.9218132,0.9467998]
96	9.151313	8.032869	$< 2.2e-16$	[1.104131,1.132757]
128	11.399267	9.985586	$< 2.2e-16$	[1.395537,1.431825]
256	17.92215	16.23799	$< 2.2e-16$	[1.661782,1.706545]
384	22.33866	20.51797	$< 2.2e-16$	[1.798866,1.842529]
512	25.42254	23.56704	$< 2.2e-16$	[1.833487,1.877505]
768	29.50278	27.85720	$< 2.2e-16$	[1.623824,1.667322]
1024	32.16532	30.64122	$< 2.2e-16$	[1.504894,1.543299]
1536	36.38148	34.32279	$< 2.2e-16$	[2.039412,2.077982]
2048	44.65734	41.47401	$< 2.2e-16$	[3.169402,3.197258]
3072	53.77092	51.23604	$< 2.2e-16$	[2.515795,2.553979]
4096	59.61126	57.61270	$< 2.2e-16$	[1.982402,2.014733]
6144	68.76167	65.67277	$< 2.2e-16$	[3.068240,3.109561]
8192	72.46089	68.71487	$< 2.2e-16$	[3.710361,3.781675]

Tabela A.2: *Teste t de Student para o benchmark NetPIPE - Megabytes*

A.1.2 FFT – Fast Fourier Transform

P	Média LLC	Média TCP	valor p	Interv. confiança de 99%
2	910.68	936.40	$= 1.294e-06$	[-38.24947,-13.19053]
4	793.42	820.64	$< 2.2e-16$	[-32.79992,-21.64008]
8	503.64	539.52	$< 2.2e-16$	[-42.21578,-29.54422]
16	354.64	380.60	$< 2.2e-16$	[-28.02395,-23.89605]

Tabela A.3: *Resultados do Teste t de Student para o algoritmo FFT*

A.1.3 Algoritmo de Ordenação Radix

P	Média LLC	Média TCP	valor p	Interv. confiança de 99%
2	12.20	12.16	= 0.607	[-0.1636501,0.2436501]
4	12.24	12.22	= 0.8145	[-0.2032854,0.2432854]
8	12.26	12.36	= 0.2843	[-0.3440550,0.1440550]
16	12.36	12.38	= 0.838	[-0.2761797,0.2361797]

Tabela A.4: *Resultados do Teste t de Student para o algoritmo Radix*

A.2 Componente BTL LLC

Aqui são descritas algumas das funções e características do componente BTL TCP que foram alteradas no desenvolvimento do componente BTL LLC.

- O sistema de compilação da biblioteca (que usa autoconf e automake para gerar os Makefiles) foi modificado de modo a incluir o novo componente em uma configuração automática.
- Foi desenvolvida a função *opal_ifindextohdrinfo* do *framework* OPAL, que retorna o endereço MAC associado à interface de rede.
- O comportamento de criação de processos, que no BTL TCP permite a criação de processos em diferentes redes de computadores, foi modificado de forma a permitir apenas a criação de processos na rede local Ethernet.
- A estrutura *mca_btl_llc_module* foi desenvolvida de acordo com as características suportadas pelo protocolo LLC.
- A função *mca_btl_llc_register* registra as funções de envio e recebimento que o componente suporta.

O ciclo de vida e as funções executadas pelo componente BTL, tanto o TCP quanto o LLC, são descritos a seguir. Aonde lê-se *btl proto* pode-se ler tanto *btl tcp* quanto *btl llc*.

1. Inicialização

```
  btl proto component open
  btl proto component init
  btl proto component create instances
* btl proto create
* btl proto component create_listen
* btl proto setsocket options
* btl proto component exchange
* btl proto add procs
  endpoint  construct (executada tantos vezes quantos os endpoints presentes)
  btl proto del procs
  btl proto register
```

2. Troca de Informações

- Lado do Servidor

```
  btl proto alloc
  btl proto send
  endpoint  send
* endpoint  start connect
  btl proto setsock options
  endpoint  send handler
  endpoint  complete connect
  endpoint  send connect ack
  endpoint  send blocking
  endpoint  recv handler
  endpoint  recv connect ack
  endpoint  recv blocking
  endpoint  connected
```

- Lado do Cliente

```
component recv_handler
component accept
btl proto setsock options
btl proto recv handler
* endpoint  accept
endpoint  close
endpoint  send connect ack
endpoint  send blocking
endpoint  connected
```

3. Anel Estabelecido, Troca de Componentes

    • Lado do Servidor

```
:->loop
  endpoint    send handler
  btl proto   frag send
  endpoint    recv handler
  btl proto   frag recv
até que o processo finalize, vá para :->loop
```

    • Lado do Cliente

```
:->loop
  endpoint   recv handler
  btl proto  frag receive
  btl proto  alloc
  btl proto  send
  endpoint send
até que o processo finalize, vá para :->loop
```

4. Transferência de Dados de um Nodo a Outro

    • Envio de Pacote

```
btl proto  prepare src
btl proto  send
endpoint   send
btl proto  frag send
btl proto  free
```

    • Recepção de Pacote

```
:->início
  endpoint   recv handler
  btl proto  frag receive
fim, vá para :-> loop até que o pacote termine
```

5. Finalizando as conexões

    • Cliente e Servidor Executam

```
btl proto  finalize
:->loop
  endpoint close
```

```
  endpoint destruct
vá para :->loop até que não haja mais nenhum endpoint a ser fechado
component  close
```

Como o componente BTL LLC é baseado no código do componente BTL TCP, todos os itens marcados com um * ao lado, foram significativamente modificados enquanto que os demais foram adaptados para o novo código. O total de linhas do *patch* para modificar o componente BTL TCP em um componente BTL LLC criado pelo comando `diff -ruN llc tcp` é *9494* disponível em `http://www.lezz.org/`.

BIBLIOGRAFIA

[1] Message Passing Interface Forum. The message passing interface (mpi) standard, February 2006. http://www-unix.mcs.anl.gov/mpi/.

[2] PVM Group. Parallel virtual machine, February 2006. http://www.csm.ornl.gov/pvm/.

[3] David E. Culler and Jaswinder Pal Singh. *Parallel Computer Architecture: a Hardware/-Software Approach*. Morgan Kaufmann Publishers, Inc, 1999.

[4] OpenMPI Group. Openmpi frequently asked questions, April 2006. http://www.open-mpi.org/faq/.

[5] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. http://www.ietf.org/rfc/rfc793.txt.

[6] Robbert van Renesse. Masking the overhead of protocol layering. In *SIGCOMM '96*, pages 96–104. ACM Press, 1996. http://doi.acm.org/10.1145/248156.248166.

[7] Vijay Karamcheti and Andrew A. Chien. Software overhead in messaging layers: Where does the time go? *ASPLOS-VI*, 28, 29(11, 5):51–60, November 1994. http://doi.acm.org/10.1145/195473.195499.

[8] A A Chien, M D Hill, and S S Mukherjee. Design challenges for high-performance network interfaces. *IEEE Computer*, 31(11):42–44, November 1998. http://ftp.cs.wisc.edu/markhill/Papers/computer98_niguest.pdf.

[9] Roberto A Hexsel. *A Quantitative Performance Evaluation of SCI Memory Hierarchies*. PhD dissertation, University of Edinburgh, Dept of Computer Science, October 1994. Tech Report CST-112-94. http://www.inf.ufpr.br/roberto/absMemHierAbstr.html.

[10] Peter A. Steenkiste. A systematic approach to host interface design for high speed networks. *IEEE Computer*, 27(3):47–57, 1994. http://dx.doi.org/10.1109/2.268886.

[11] Piyush Shivam and Jeffrey S. Chase. On the elusive benefits of protocol offload. In *ACM SIGCOMM Workshop on Network-I/O convergence*, pages 179–184. ACM Press, 2003. http://doi.acm.org/10.1145/944747.944750.

[12] Jeffrey C. Mogul. TCP offload is a dumb idea whose time has come. *HotOS IX: The 9th Workshop on Hot Topics in Operating Systems*, April 2003. http://www.usenix.org/events/hotos03/tech/full_papers/mogul/mogul_html/index.html.

[13] Arnaldo Carvalho de Melo. TCPfying the Poor Cousins. *Linux Symposium*, 2(11):367–370, 2004. http://www.linuxsymposium.org/proceedings/reprints/Reprint-Melo-OLS2004.pdf.

[14] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a Realistic Model of Parallel Computation. *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993. http://doi.acm.org/10.1145/155332.155333.

[15] John L. Hennessy and David A. Patterson. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers, Inc., 3rd edition, 1997.

[16] P Druschel, M B Abbot, M A Pagels, and L L Peterson. Network subsystem design. *IEEE Network*, 7(4):8–17, July 1993.

[17] IEEE. ANSI IEEE 802.2 Standard. Technical report, IEEE, May 1998. http://standards.ieee.org/getieee802/download/802.2-1998.pdf.

[18] Yi-Chun Chu and Toby J. Teorey. Modeling and analysis of the unix communication subsystems. *Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research*, 1996.

[19] Chandramohan A. Thekkath and Henry M. Levy. Limits to low-latency communication on high-speed networks. *ACM Transactions on Computer Systems (TOCS)*, 11(2):179–203, 1993. `http://doi.acm.org/10.1145/151244.151247`.

[20] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987. `http://www.ietf.org/rfc/rfc1034.txt`.

[21] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987. `http://www.ietf.org/rfc/rfc1035.txt`.

[22] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1098, April 1989. `http://www.ietf.org/rfc/rfc1098.txt`.

[23] Maurice J. Bach. *The Design of the Unix operating System*. Prentice Hall, 2nd edition, 1990.

[24] Richard W. Stevens. *Advanced Programming in the UNIX Environment*. Addison-Wesley, 1992.

[25] Andrew S. Tanenbaum. *Redes de Computadores*. Editora Campus, 3rd edition, 1997.

[26] Z. Su. RFC 781: Specification of the Internet Protocol (IP) timestamp option, May 1981. `ftp://ftp.internic.net/rfc/rfc781.txt`.

[27] Douglas E. Comer. *Interligação em rede com TCP/IP volume 1*. Editora Campus, 3rd edition, 2001.

[28] William Stallings. *Data and Computer Communications*. Macmillan Publishing Company, 3rd edition, 1991.

[29] Douglas E. Comer and David L. Stevens. *Internetworking with TCP/IP volume II*. Prentice Hall, 2nd edition, 1994.

[30] IEEE. ANSI IEEE 802.3 Standard. Technical report, IEEE, May 1998. http://standards.ieee.org/getieee802/download/802.3ae-2002.pdf.

[31] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, 2nd edition, 1992.

[32] Giovanni Chiola and Giuseppe Ciaccio. Efficient parallel processing on low-cost clusters with gamma active ports. *Parallel Computing*, 26(2-3):333–354, 2000. http://dx.doi.org/10.1016/S0167-8191(99)00108-8.

[33] G. Chiola and G. Ciaccio. Porting MPICH ADI on GAMMA with Flow Control. http://citeseer.ist.psu.edu/228803.html.

[34] Giovanni Chiola and Giuseppe Ciaccio. Gamma: a low cost network of workstations based on active messages, 1997. http://citeseer.ist.psu.edu/chiola97gamma.html.

[35] Julio Ortega, Antonio F. Díaz, Mancia Anguita, Antonio Cañas, F. J. Fernández, and Alberto Prieto. An efficient os support for communication on linux clusters. In *ICPP Workshops*, pages 397–402, 2001. http://csdl.computer.org/comp/proceedings/icppw/2001/1260/00/12600397abs.htm.

[36] Antonio F. Díaz, Julio Ortega, Antonio Cañas, F. J. Fernández, and Alberto Prieto. The Lightweight Protocol CLIC: Performance of an MPI implementation on CLIC. In *CLUSTER*, pages 391–398, 2001. http://csdl.computer.org/comp/proceedings/cluster/2001/1116/00/11160391abs.htm.

[37] Antonio F. Díaz, Julio Ortega, Antonio Cañas, F. J. Fernández, Mancia Anguita, and Alberto Prieto. The Lighweight Protocol CLIC on Gigabit Ethernet. In *IPDPS*, page 200a, 2003. http://csdl.computer.org/comp/proceedings/ipdps/2003/1926/00/19260200aabs.htm.

[38] V. S. Sunderam. PVM: a framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990. `http://citeseer.ist.psu.edu/sunderam90pvm.html`.

[39] William Gropp and Ewing L. Lusk. Goals guiding design: Pvm and mpi. `http://citeseer.ist.psu.edu/568858.html`.

[40] Richard L. Graham, Timothy S. Woodall, and Jeffrey M. Squyres. Open MPI: A flexible high performance MPI. In *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*, Poznan, Poland, September 2005.

[41] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, September 2004.

[42] B. Barrett, J. M. Squyres, A. Lumsdaine, R. L. Graham, and G. Bosilca. Analysis of the Component Architecture Overhead in Open MPI. In *Proceedings, 12th European PVM/MPI Users' Group Meeting*, Sorrento, Italy, September 2005.

[43] Quinn O. Snell, Armin R. Mikler, and John L. Gustafson. Netpipe: A network protocol independent performance evaluator, April 2006. `http://www.scl.ameslab.gov/netpipe/paper/full.html`.

[44] William Gropp and Ewing L. Lusk. Reproducible measurements of mpi performance characteristics, 1999. `http://www.mcs.anl.gov/~gropp/bib/papers/1999/pvmmpi99/mpptest.pdf`.

[45] David H. Bailey. FFTs in External or Hierarchical Memory. *Journal of Supercomputing*, 1(4):23–35, 1990.

[46] Sergio Luiz Marques Filho. Uma Abordagem Multithread em Aplicações Paralelas Usando MPI. Master's thesis, Universidade Federal do Paraná, Departamento de Informática, 2005.

[47] Wikipedia. Radix sort — Wikipedia, the free encyclopedia, January 2006. http://en.wikipedia.org/wiki/Radix_sort.

[48] David Roxbee Cox. *Planning of experiments*. John Wiley, 1992.

[49] George E. P. Box, William G. Hunter, and J. Stuart Hunter. *Statistics for Experimenters An introduction to Design, Data Analysis, and Model Building*. John Wiley, 1st edition, 1976.

[50] Leonard J. Kazmier. *Estatística Aplicada a Economia e Administração*. Mc Graw-Hill, 1982.

[51] R Project. The r project for statistical computing, January 2006. http://www.r-project.org/.

[52] Wikipedia. Fast fourier transform — Wikipedia, the free encyclopedia, March 2006. http://en.wikipedia.org/wiki/Fast_fourier_transform.

[53] Guy E. Blelloch, Charles E. Leiserson, Bruce M. Maggs, C. Greg Plaxton, Stephen J. Smith, and Marco Zagha. A Comparison of Sorting Algorithms for the Connection Machine CM-2. *In Symposium on parallel Agorithms and Architechtures*, 1:3–16, 1991.

[54] Wikipedia. Counting sort algorithm — Wikipedia, the free encyclopedia, February 2006. http://en.wikipedia.org/wiki/Counting_sort.