Capítulo V

Análise de desempenho

O sistema implementado está codificado em linguagem C e roda em computadores PC-compatíveis sob o sistema operacional DOS. A Tabela V.1 mostra o número de linhas de código por camada, módulo ou interface, totalizando mais de 8400 linhas. Todo esse código gera um programa executável de aproximadamente 170 koctetos, com informação de depuração. O ambiente de medidas consiste em PCs equipados com CPUs 486 DX, conectados a uma rede local Ethernet através de placas padrão NE2000.

Tabela V.1: Número de linhas de código.

Camada / Módulo	Número de linhas
Gerente de Memória	973
Escalonador de Tarefas	125
Gerente de Temporizações	487
MAC	497
Interface LLC-MAC	821
Protocolo MCF	3811
Interface Sessão-Transporte	180
Usuário de medidas	1429
Programa principal	94
TOTAL	8417

Esta análise de desempenho consiste fundamentalmente em medidas de **vazão** para um usuário sobre a camada MCF, em diferentes configurações. Doravante, vazão significará o total de dados do usuário (cabeçalhos ou caudas de PDUs não são levados em consideração) transmitido em uma unidade de tempo.

O ideal seria que o tempo de transmissão fosse medido a partir do instante em que o transmissor envia a primeira mensagem até a recepção da última mensagem pela entidade remota. Entretanto, isto exigiria um sistema de medidas distribuído. Outra metodologia foi então adotada no usuário de medidas: o transmissor computa o intervalo de tempo decorrido entre o envio da primeira mensagem e a recepção do reconhecimento da última mensagem. Esta medida utiliza as interrupções do relógio do PC, geradas a cada 55 ms. Portanto, uma quantidade mínima de 200 quadros é transmitida para manter o erro de medida em um limite tolerável. Nenhuma medida de estabelecimento ou liberação de conexões foi realizada, em virtude da necessidade de interface com o usuário para a determinação das opções da conexão. Pode-se dizer também que os computadores eram dedicados à transferência de dados, em um horário de pouco tráfego na rede. O usuário realizava apenas transferências de memória para memória, evitando assim qualquer tipo de retardo introduzido por dispositivos de entrada e saída. As operações de exibição no vídeo foram inclusive reduzidas a um mínimo necessário durante a transferência propriamente dita.

As medidas incluem o *overhead* introduzido pelo cabeçalho/cauda da camada MCF e pelo cabeçalho/cauda da camada MAC, em um total de 58 octetos por mensagem. Na verdade, essas medidas de vazão são limites inferiores pois incluem o processamento do usuário na criação/destruição de mensagens, o processamento dos dados nas interfaces, alocações e dealocações de memória, etc.

A Figura V.1 mostra a vazão, em Mbps, para os usuários das camadas MAC e MCF, além da vazão obtida exclusivamente pelo *packet driver*. Para o MCF, foi medida a vazão nas seguintes configurações:

1) nenhum mecanismo opcional habilitado;
2) apenas o cálculo de *checksum* habilitado. É bom lembrar que esta opção ativa apenas o cálculo sobre o segmento de dados de PDUs DATA, pois o *checksum* é obrigatório sobre os cabeçalhos de todas as PDUs e sobre o segmento de dados das PDUs de controle (FIRST, CNTL, PASS, TOKEN, ERROR);
3) apenas o controle de fluxo por janela de transmissão habilitado;
4) *checksum*, controle de fluxo por janela e controle de erros, com retransmissão seletiva ou *go-back-N*, habilitados.

As medidas foram realizadas sobre conexões ponto-a-ponto, utilizando vários tamanhos de mensagem, entre 128 e 1024 octetos.

Teoricamente, a vazão máxima para LANs padrão IEEE 802.3 ou Ethernet é de 6 Mbps e 9,8 Mbps para quadros de tamanho mínimo e máximo, respectivamente. Estes valores podem não corresponder à realidade para a maioria das placas de rede. Por isso, foi

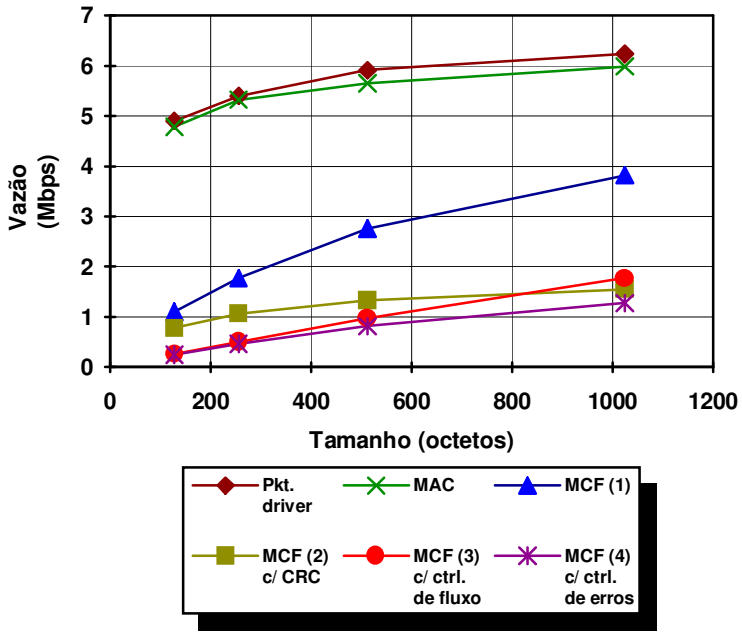

Figura V.1: Vazão × tamanho da mensagem.

desenvolvido um programa especial para a avaliação do desempenho dos *packet drivers*. Com este programa, pode-se garantir que a vazão é devida praticamente à transferência de dados entre a memória do computador e os *buffers* da placa (através de cópia ou DMA), pois o programa utiliza somente as rotinas de acesso ao *packet driver*, excluindo a interface padrão e os módulos globais de escalonamento de tarefas e gerenciamento de memória e temporizações.

Tabela V.2: Vazões obtidas para mensagens de 1024 octetos.

Camada	Vazão (Mbps)
Packet Driver	6,2
MAC	6,0
MCF	3,8
MCF c/ *checksum*	1,6
MCF c/ ctrl. de fluxo p/ janela	1,8
MCF c/ ctrl. de erros	1,3

Como mostra a Figura V.1, à medida que o tamanho da mensagem aumenta, a vazão tende a um máximo de aproximadamente 6,2 Mbps para o *packet driver*. A Tabela V.2 destaca as vazões obtidas por cada camada para mensagens de 1024 octetos. Observa-se que a camada MAC atinge quase a mesma vazão do *driver*, de onde se conclui que o *overhead* por ela introduzido (incluindo interface e módulos globais) é desprezível quando comparado ao processamento do *packet driver* e suas rotinas de acesso e ao desempenho da própria placa de rede. Para provar essa afirmação, foi implementado um *loopback* na camada MAC (Figura V.2), permitindo a presença dos fluxos de transmissão e recepção na mesma máquina através

da cópia dos dados recebidos para a fila de subida da interface MCF-MAC. A vazão alcançada pela camada MAC em *loopback* foi de quase 60 Mbps (em um 486DX2 66), demonstrando a limitação imposta pelo conjunto placa/*packet driver*.

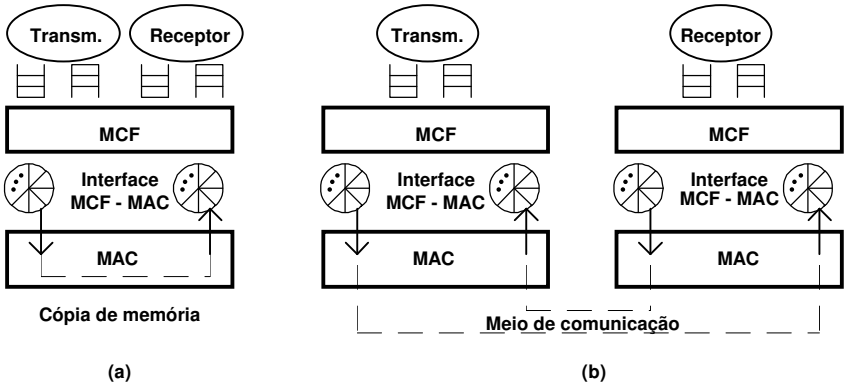

Figura V.2: Comparação do funcionamento do sistema em *loopback* (a) com o funcionamento normal via rede (b).

Ainda sobre a Figura V.1, quanto ao desempenho da camada MCF na configuração **1**, a maior responsável pela perda em relação à camada MAC é a inevitável cópia dos dados para uma região contígua de memória (especificamente, o *buffer* de transmissão da interface MCF-MAC), exigência de qualquer controlador de comunicação. Esta medida também inclui o *overhead* do cálculo de *checksum* sobre o cabeçalho das PDUs DATA, as únicas transferidas nesta medida, pois não há nenhuma espécie de controle de fluxo (não são enviadas, portanto, PDUs CNTL). Entretanto, sua contribuição para a perda de desempenho neste caso pode ser relegada a um segundo plano.

O processamento da camada MCF na configuração **3** apresenta uma perda de 76% a 53% no intervalo de 128 a 1024 octetos em relação à configuração **1**. Isto se deve ao armazenamento das mensagens no *buffer* interno de transmissão, ao processamento das PDUs CNTL de reconhecimento e, principalmente, à espera por esse reconhecimento: durante este intervalo de tempo, as transmissões ficam bloqueadas, pois o *buffer* encontra-se cheio. Além disso, o pedido de reconhecimento vai na forma de um bite `DREQ` habilitado no cabeçalho de uma PDU DATA quando o *buffer* é totalmente preenchido. Ao contrário do bite `SREQ`, que exige resposta imediata, o bit `DREQ` indica que os dados recebidos devem ser primeiramente entregues ao usuário, introduzindo aí um pequeno retardo no envio da PDU CNTL de reconhecimento. Somente quando receber esta PDU o transmissor poderá liberar o *buffer* interno e dar prosseguimento à transmissão.

A medida realizada na configuração **2** comprova a forte dependência do cálculo de *checksum* com o comprimento do campo de dados de uma PDU DATA. Até pouco mais de 800 octetos, o gráfico da Figura V.1 mostra que o processamento da função de *checksum* é

suficientemente rápido para fazer com que o desempenho da camada MCF nesta configuração supere o desempenho obtido na configuração **3**. Em compensação, a partir daquele tamanho o peso do cálculo de *checksum* no desempenho começa a se tornar visível.

Pela Figura V.1 observa-se também uma perda de 5,3% a 28% na configuração **4** em relação às medidas na configuração **3**. As mesmas medidas de vazão foram obtidas independentemente do mecanismo de retransmissão selecionado (*go-back-N* ou retransmissão seletiva). Comparando-se o desempenho nas configurações **3** e **4**, verifica-se que o resultado obtido no intervalo de 128 a 256 octetos é virtualmente idêntico, provando que o *overhead* de processamento introduzido pelo controle de erros é desprezível. Este processamento consiste no algoritmo de decisão realizado pelo transmissor com base nas informações obtidas a partir das PDUs CNTL de reconhecimento, para determinar a necessidade, ou não, de retransmissão. A partir de 256 octetos as curvas começam a se distanciar, visivelmente devido ao cálculo de *checksum* sobre os dados.

Apesar da arquitetura de implementação permitir segmentação e remontagem sem cópia (apenas ponteiros são manipulados), a segmentação penaliza a vazão devido ao processamento das PDUs adicionais. Como se percebe na Figura V.3, à medida que aumenta o tamanho da mensagem, a vazão aumenta até a próxima segmentação, quando então cai bruscamente. Esta queda é atribuída ao tempo gasto com o processamento de uma PDU extra contendo apenas um octeto de dados. Para esta medida, o tamanho máximo do campo de dados em PDUs DATA foi reduzido de 1024 para 128 octetos.

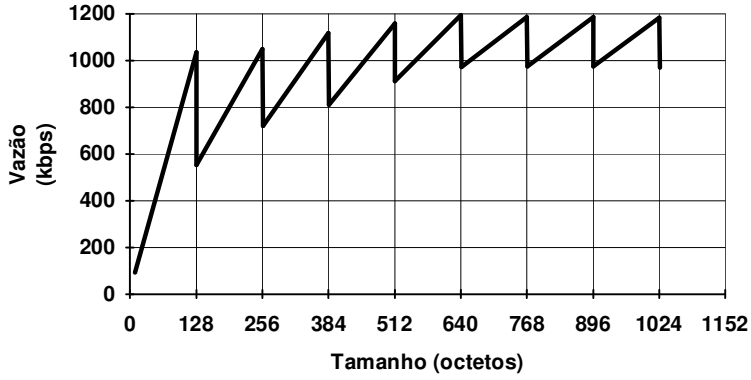

Figura V.3: Vazão × tamanho da mensagem com segmentação em 128 octetos.

Para as medidas de controle de fluxo por taxa, foram transferidos 200 quadros de 1024 octetos entre dois microcomputadores. O tamanho da rajada, sempre múltiplo de 1024, era modificado a cada medida, de modo a manter constante a razão rajada/taxa. A Tabela V.3 mostra os valores efetivamente obtidos para diferentes taxas de transmissão. A alta granularidade do mecanismo de temporização do PC (mínimo de 55ms) impede uma melhor precisão na determinação de RTIMER. A rotina de disparo de temporizadores recebe a

duração do temporizador em um número inteiro de *timer ticks* (ver Seção IV.5). Essa duração é calculada dividindo-se a rajada pela taxa, multiplicando-se por 18,2 e arredondando-se o valor obtido para o maior inteiro imediatamente superior. Conseqüentemente, RTIMER operará com uma duração, em segundos, maior do que a prevista. Isso explica os valores menores obtidos para a taxa efetiva, pois um incremento no tempo implica em um decremento na taxa. Os valores práticos tiveram uma variação percentual média de 4% em relação aos teóricos. Mas se for levado em consideração que o próprio arredondamento já introduz um erro de 7,3%, pode-se concluir que o resultado geral é satisfatório.

Tabela V.3: Controle de fluxo por taxa.

Taxa desejada (kbps)	Taxa obtida (kbps)
64	60,9
128	124,2
192	186,4
256	248,5
320	301,2
384	372,7
448	425,8
512	497,0

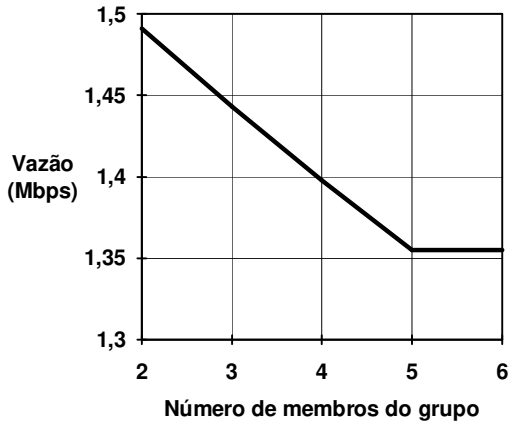

Figura V.4: Vazão × tamanho do grupo multiponto.

Para avaliar o desempenho do algoritmo *bucket* (com o cálculo de *checksum* habilitado), foi medida a vazão obtida em transmissões $1 \rightarrow N$, com N = 1, 2, 3, 4 e 5 e mensagens de tamanho máximo (1024 octetos). Os resultados estão no gráfico da Figura V.4. Neste gráfico, observa-se que a vazão diminui com o aumento do número de membros do grupo. Isto era de se esperar, pois o crescimento do grupo implica em aumento de processamento no transmissor, uma vez que aumenta o número de PDUs CNTL enviadas como resposta e que precisam ser tratadas ou descartadas pelo transmissor, dependendo dos rótulos dos *buckets* e do valor dos campos *echo*. Entretanto, a vazão se mantém constante

quando mais um membro é adicionado ao grupo de cinco. Evidentemente, a tendência é alcançar um valor mínimo de vazão no limite quando *N* tendesse ao infinito. Mas para determinar este mínimo seriam necessários muitos computadores disponíveis para formar um grupo arbitrariamente grande, o que não foi possível no ambiente de medidas. Espera-se que com uma medida com um grupo de, por exemplo, 20 participantes, possa ser traçada uma curva mais suave, ajustando-a por entre os valores obtidos. Vale observar também, a título de comparação, que uma vazão de aproximadamente 1,5 Mbps é obtida com o algoritmo *bucket* em uma transmissão ponto-a-ponto (grupo com apenas dois membros), contra os 1,3 Mbps obtidos com o conjunto *checksum* / janela / *go-back-N* (Tabela V.2). A implementação contava com dois *buckets* e uma duração de 1 *timer tick* para ST, mostrando que, nesta configuração, o algoritmo *bucket* não restringe o fluxo da transmissão. Por outro lado, como já foi mencionado anteriormente, o controle de fluxo por janela em conexões ponto-a-ponto força o transmissor a preencher o *buffer* antes do envio do pedido de reconhecimento, bloqueando a transmissão até sua recepção.