Proposta e Implementação de um Protocolo Multidestinatário Configurável

Marcelo Dias Nunes
email:bill@coe.ufrj.br

Otto C. M. B. Duarte
email: otto@coe.ufrj.br

Universidade Federal do Rio de Janeiro
COPPE/EE - Programa de Engenharia Elétrica
Caixa Postal 68504 - CEP 21945-970 - Rio de Janeiro - RJ - Brasil
FAX: +55 21 290.6626

Resumo

Este artigo apresenta a proposta, implementação e teste de um protocolo configurável, para comunicações multidestinatárias estatisticamente confiáveis, com gerenciamento de acesso por por fichas de permissão, apropriado para ambientes de redes locais. Os serviços oferecidos correspondem às funcionalidades de uma camada transporte e no ambiente de desenvolvimento utilizam os serviços da sub-camada MAC nas configurações ponto-a-ponto e ponto-a-multiponto. A implementação do protocolo, realizada em linguagem C, segue um modelo de arquitetura em camadas com mecanismos que visam alto desempenho. O sistema como um todo contém mais de 8.000 linhas de código e opera normalmente sem erros, apresentando um bom desempenho.

Abstract

This article presents the proposal, implementation and test of a programmable protocol, well suited for statistically reliable multipoint communications, with a token-based access management adequate for LAN environments. It offers the functionality of a transport layer and uses the services provided by the MAC sublayer in point-to-point and multicast configurations over the development environment. The protocol's implementation, coded in C language, presumes a layered-model architecture with high performance mechanisms. The overall system consists in more than 8,000 lines of source code and usually runs error-free, showing a good performance.

I - Introdução

Neste artigo, é apresentada uma proposta de protocolo multidestinatário estatisticamente confiável, configurável e com gerenciamento de acesso por fichas de permissão, bem como a análise de desempenho realizada sobre uma implementação do mesmo em linguagem C. Entretanto, uma vez que a principal etapa na definição de um novo protocolo consiste no estudo dos mecanismos de comunicação usados em outros protocolos de alto desempenho, será feita nesta seção uma análise comparativa de diferentes mecanismos necessários para o suporte dos seguintes serviços [Doer90] [Wats81] [Wats87]:

- **conexões**, que podem ser **negociadas** ou **implícitas**;
- **reconhecimentos**, que podem ser **submetidos ao transmissor** ou **independentes do transmissor**;
- **controle de fluxo**, que pode ser feito por **janelas de transmissão** ou através do controle da **taxa de transmissão**;

- **controle de erros**, em que as retransmissões podem ser disparadas por **reconhecimento positivo** (PAR) ou **pedido de repetição automática** (ARQ). As retransmissões propriamente ditas são feitas através dos mecanismos ***go-back-N*** ou **retransmissão seletiva**.

Um estudo de alguns protocolos de alta velocidade (Datakit, Delta-t, NETBLT, VMTP e XTP) pode ser encontrado em [Nune94b].

Os dois mecanismos básicos de gerenciamento de conexões são a negociação (*handshake*) e conexões implícitas baseadas em temporizadores. Conexões negociadas têm a desvantagem de introduzirem um *overhead* de processamento significativo para aplicações transacionais ou, de uma maneira geral, para qualquer conexão de curta duração ou que envolva a transmissão de pequena quantidade de dados. Mas quando se trata da transferência de grandes volumes de dados em alta velocidade, esse *overhead* pode ser desprezado. Mecanismos implícitos, por sua vez, simplificam o estabelecimento de conexões dispensando trocas de mensagens adicionais, mas introduzem o inconveniente de se determinar o valor ideal para os temporizadores da conexão, normalmente calculados sobre estimativas de tempo de ida e volta (*round trip delay*). Conexões implícitas também requerem a implementação de mecanismos de "envelhecimento" de mensagens (campo de tempo de vida), mais um fator de degradação de desempenho. Portanto, para o protocolo proposto optou-se por adotar o mecanismo de *handshake* para abertura e fechamento de conexões ponto-a-ponto. Conexões multidestinatárias utilizam uma espécie de estabelecimento implícito, porém sem temporizadores para manutenção da conexão, como será explicado na Seção II. A principal vantagem do *handshake* que pesou nessa decisão é a possibilidade de negociar **parâmetros** da conexão.

No que diz respeito à geração de reconhecimentos submetida ao transmissor, a vinculação do envio de reconhecimentos à recepção de mensagens decididamente aumenta a complexidade do processamento no receptor, que já é naturalmente prejudicado por exercer um maior processamento por mensagem que o transmissor. Nesse caso, a segunda possibilidade, ou seja, enviar um reconhecimento apenas quando explicitamente solicitado pelo transmissor, é preferencial em termos de processamento. Além disso, devido à complexidade introduzida pela manipulação de temporizadores, mais necessários em outros mecanismos, a possibilidade de gerar reconhecimentos idependentemente do transmissor foi prontamente descartada.

A utilização de janelas de transmissão, de tamanho fixo ou não, para controle de fluxo é, acima de tudo, um mecanismo de controle da utilização dos *buffers* de transmissão e, principalmente, de recepção, para evitar problemas de sobre-escrita (*overrun*). O único cuidado a ser tomado para vencer a latência no meio físico é a escolha de uma janela suficientemente grande que garanta a continuidade da transmissão. Porém, essa escolha depende das características de atraso da rede e das disponibilidades de armazenamento nas estações. Por sua vez, o controle de fluxo através do controle da taxa de transmissão é aparentemente melhor habilitado a suportar diferentes padrões de geração de tráfego. Em suma: ambos os mecanismos são válidos, pois endereçam diferentes aspectos da comunicação: uma aplicação de transmissão de voz pode exigir um controle mais preciso da taxa de transmissão, por exemplo a 64 kbps. Por outro lado, uma aplicação de transferência de arquivos está mais preocupada em não encher os *buffers* de recepção. Logo, os dois mecanismos de controle de fluxo são oferecidos nesta implementação.

Uma vez que o mecanismo de geração de reconhecimentos adotado na proposta é a submissão a comandos do transmissor, pois evita o uso de temporizadores e um simples bite do cabeçalho da última PDU de dados de uma mensagem pode servir como pedido de reconhecimento, o mecanismo de disparo de retransmissões que melhor se adapta a esse contexto é o pedido de repetição automática (ARQ). Quanto à forma de recuperação de dados, a recuperação por *go-back-N* tem a vantagem de um processamento mais simples, mas apresenta o incoveniente de acarretar a

retransmissão de dados corretamente recebidos, porém descartados. Essas retransmissões desnecessárias são contudo evitadas se um mecanismo de retransmissão seletiva for utilizado. Porém, tal mecanismo tem a desvantagem de exigir a capacidade de armazenamento e resseqüenciamento das mensagens no receptor, ou seja, maior processamento. Ficou então decidido deixar para a aplicação a opção do modo de recuperação de erros que a ela melhor se adapte.

É necessário agora definir o grau de confiabilidade a ser oferecido em conexões ponto-a-multiponto [Diot]. Conexões não confiáveis não são adequadas à maioria das aplicações. Em conexões atômicas ou totalmente confiáveis, o transmissor precisa esperar por todos os reconhecimentos para liberar seus *buffers*. Tal degradação evidentemente cresce com o aumento do tamanho do grupo. A melhor solução parece ser um meio termo, ou seja, conexões estatisticamente confiáveis. Assim, pelo menos teoricamente, um grupo tem tamanho ilimitado, pois nenhum membro, seja ele transmissor ou receptor, precisa conhecer o número total de membros ou seu *status*. Basta conhecer o endereço do grupo. O algoritmo *bucket*, utilizado pelos protocolos XTP [PEI92] [Sant92] [Stra92] e HSTP [Cohn91], permite a implementação de conexões estatisticamente confiáveis, tendo sido portanto implementado no protocolo proposto. Mas resta ainda decidir a melhor maneira de oferecer conexões **multiponto-a-multiponto**, que é um dos objetivos do protocolo. Assim, foi criado um mecanismo de fichas que permite a um subconjunto do grupo atuar simultaneamente como transmissor. São utilizadas *N* fichas, onde *N* é menor do que o número máximo de participantes da comunicação (limitado, na prática, pela memória disponível).

Por questões de simplicidade, o protocolo proposto neste artigo será doravante referenciado pela sigla que resume suas principais características: **MCF** (protocolo **M**ultidestinatário **C**onfigurável com gerenciament de acesso por **F**ichas de permissão). A Seção II contém uma descrição do protocolo proposto, detalhando cada um de seus procedimentos. A Seção III expõe os resultados das análises de desempenho realizadas, enfatizando medidas de vazão. Finalmente, conclusões são apresentadas na Seção IV.

II - O Protocolo MCF

II.2 - Formato das PDUs

Por questões de desempenho, é indiscutível que os campos no cabeçalho de uma PDU devem ter tamanho fixo, alinhados preferencialmente por 4 ou 8 octetos, para aproveitar o potencial das máquinas de 32 e 64 bites, bem como acelerar o processamento de decodificação dos campos. Assim, foi implementado neste protocolo o mesmo formato das PDUs dos protocolos XTP e HSTP, que utilizam campos alinhados em 32 bites. No presente trabalho, alguns campos definidos por aqueles protocolos foram suprimidos, da mesma forma em que outros foram criados ou adaptados. O protocolo MCF utiliza seis PDUs, a saber:

- DATA PDU de dados do usuário;
- CNTL PDU de informações de controle;
- FIRST PDU de estabelecimento de conexão ponto-a-ponto (também pode conter dados do usuário);
- TOKEN PDU representativa da ficha em conexões multidestinatárias;
- PASS PDU de pedido de ficha ao servidor;
- ERROR PDU de diagnóstico em situações de erro.

II.3 - Primitivas de serviço

Para uma melhor compreensão dos procedimentos de abertura e fechamento de conexões e transferência de dados, é necessário apresentar as primitivas utilizadas pelo usuário para a solicitação de algum serviço. Por conter funcionalidades de um protocolo de transporte, como segmentação e remontagem, juntamente com garantias de confiabilidade fim-a-fim, pode-se considerar que a interface com o usuário se encaixa na interface entre as camadas de Sessão e Transporte do Modelo OSI. Por isso, a nomenclatura das primitivas segue o Modelo OSI para o serviço de Transporte. A Tabela 1 mostra todas as primitivas definidas para o acesso aos serviços deste protocolo experimental, com uma breve descrição e os parâmetros para cada uma delas.

Tabela 1: Primitivas de serviço.

Primitivas	Descrição	Parâmetros
T-CONNECT.Request	Pedido de estabelecimento de conexão ponto-a-ponto	Endereço destino Endereço Origem Qualidade de Serviço
T-CONNECT.Indication	Indicação de pedido de estabelecimento de conexão ponto-a-ponto	Endereço destino Endereço Origem Qualidade de Serviço
T-CONNECT.Response	Resposta ao pedido de estabelecimento de conexão ponto-a-ponto	Endereço respondedor Resposta (aceitação ou recusa) Qualidade de serviço
T-CONNECT.Confirm	Confirmação do pedido de estabelecimento de conexão ponto-a-ponto	Endereço Respondedor Resposta Qualidade de serviço
T-DISCONNECT.Request	Pedido de liberação de conexão ponto-a-ponto	Tipo de desconexão (normal ou aborto)
T-DISCONNECT.Indication	Indicação de pedido de liberação de conexão ponto-a-ponto	Razão de desconexão
T-DISCONNECT.Response	Resposta ao pedido de liberação de conexão ponto-a-ponto	não tem
T-DISCONNECT.Confirm	Confirmação do pedido de liberação de conexão ponto-a-ponto	não tem
T-DATA.Request	Pedido de transmissão de dados do usuário	Dados do usuário
T-DATA.Indication	Indicação de recepção de dados do usuário	Dados do usuário
T-MULTIPOINT.Request	Adesão a um grupo multiponto	Categoria (servidor ou não) Qualidade de serviço
T-TOKEN.Request	Pedido de ficha para transmissão em grupos multiponto	Endereço Destino Endereço Origem
T-TOKEN.Indication	Indicação de recepção ou devolução de ficha	não tem
T-TOKEN.Response	Comando de devolução da ficha	não tem
T-LEAVE.Request	Desligamento de um grupo multiponto	não tem

II.4 - Abertura de conexão

II.4.1 - Ponto-a-ponto

Toda estação possui um endereço lógico que consiste em um número de 0 a 255. Quando uma estação deseja conectar-se a outra, o primeiro dado pedido consiste no endereço lógico da

estação remota. A partir daí, uma série de opções são oferecidas ao usuário no que diz respeito à configuração da conexão. Essas opções, para conexões ponto-a-ponto, são as seguintes:

- cálculo de *checksum* sobre os dados em PDUs DATA;

- tipos de controle de fluxo oferecidos:
    1. janela de transmissão;
    2. controle de fluxo por taxa (nesse caso, os parâmetros de taxa, em octetos por segundo, e rajada, em octetos, são solicitados);

- tipos de recuperação de erros oferecidos:
    1. *go-back-N*;
    2. retransmissão seletiva.

Após a configuração das opções, uma primitiva T-CONNECT.Request é gerada, desencadeando o envio de uma PDU FIRST com aquelas opções. Na entidade destinatária, a recepção de uma PDU FIRST gera uma primitiva T-CONNECT.Indication, informando o usuário a respeito das opções escolhidas pela entidade iniciadora para a conexão. A entidade respondedora tem então três alternativas: aceitar a conexão sem nenhuma alteração, aceitar somente após a modificação de uma ou mais opções e recusar a conexão. Na primeira alternativa, uma primitiva T-CONNECT.Response indicando a aceitação da conexão é gerada e uma PDU CNTL é transmitida. A entidade iniciadora, ao receber esta PDU, envia uma primitiva T-CONNECT.Confirm ao usuário, abrindo definitivamente a conexão. A segunda alternativa segue o mesmo procedimento, com a diferença de que a PDU CNTL dessa vez estará levando uma contra-proposta de configuração ao iniciador. Se este último não desejar levar adiante a conexão nestes novos termos, pode simplesmente optar por fechá-la ou abortá-la. Na terceira alternativa, a primitiva T-CONNECT.Response indica a recusa da conexão, causando o envio de uma PDU ERROR com *e-code* = 1, significando operação recusada. Ao receber esta PDU, o iniciador da conexão envia ao usuário uma primitiva T-DISCONNECT.Indication, fechando a conexão.

II.4.2 - Multiponto

Não existe propriamente um estabelecimento de conexões multidestinatárias. Um dos principais objetivos deste protocolo é oferecer um serviço de comunicações de grupo **dinâmico**, ou seja, a qualquer momento estações podem entrar ou sair de um grupo, sem que isso afete a comunicação. O protocolo MCF utiliza um gerenciamento de grupo centralizado. Cada grupo deve ter um servidor, ou centralizador, de fichas de transmissão, cujo endereço lógico é bem conhecido por todas as estações da rede local.

No protocolo MCF, o servidor de fichas define um endereço de grupo de acordo com a rede subjacente (IP Multiponto, Ethernet), que também deve ser de conhecimento geral. Portanto, qualquer estação de posse desse endereço pode transmitir dados para o grupo (uma vez que a ficha lhe tenha sido concedida) e receber dados direcionados ao grupo, desde que esteja configurada para tanto, sem que nenhuma conexão seja estabelecida. A opção pela centralização foi feita com base na simplificação do projeto que esta abordagem traria, em termos de processamento e de máquina de estados. Esta simplificação se torna evidente, por exemplo, no gerenciamento de múltiplas fichas implementado por este protocolo, em que um temporizador é associado a cada ficha concedida, como será visto na Seção II.7. Esses temporizadores são mantidos apenas no servidor, liberando as estações transmissoras do processamento extra causado pela manutenção de mais um temporizador, além dos já utilizados no controle de fluxo por taxa ou no algoritmo *bucket*. Ou seja, a única estação que poderá ter seu desempenho de certa forma comprometido quando comparado às demais

estações é o próprio servidor, pois este último também tem a capacidade de transmitir dados, sendo portanto mais do que uma simples entidade de monitoramento.

Uma estação que deseje participar de um grupo deve primeiro configurar as opções da transmissão de dados, que para conexões multidestinatárias são as seguintes:

- cálculo de *checksum* sobre os dados em PDUs DATA;
- controle de fluxo por taxa (nesse caso, os parâmetros de taxa, em octetos por segundo, e rajada, em octetos, são solicitados);
- controle de erros pelo algoritmo *bucket*;
- recuperação de erros por *go-back-N*.

É importante frisar que tais parâmetros não são negociados com nenhuma outra estação, pois em um grupo cada estação tem controle apenas sobre a transmissão de seus próprios dados. Por exemplo, uma estação pode optar por não realizar nenhum tipo de controle de erros, mas não pode deixar de responder ao pedido de reconhecimento de alguma outra estação que tenha decidido utilizar o algoritmo *bucket*.

Após a configuração das opções, uma primitiva T-MULTIPOINT.Request é gerada, colocando o protocolo automaticamente no estado de transferência de dados (Active). Nessa ocasião também é enviado um comando à camada subjacente, para configurar o *hardware* de comunicação, permitindo assim que a estação receba as mensagens endereçadas ao grupo.

II.5 - Fechamento de conexão

II.5.1 - Ponto-a-ponto

Qualquer uma das entidades envolvidas em uma conexão ponto-a-ponto pode decidir terminá-la ou abortá-la. Em uma desconexão normal, uma primitiva T-DISCONNECT.Request é gerada, causando o envio de uma PDU CNTL com os bites `WCLOSE`, `RCLOSE` e `DREQ` habilitados. No caso de uma aborto, uma PDU CNTL com o bite `END` habilitado é enviada e a conexão automaticamente fechada no lado iniciador. Quando recebe uma PDU CNTL com os bites `WCLOSE`, `RCLOSE` e `DREQ` habilitados, a entidade respondedora envia uma primitiva T-DISCONNECT.Indication ao usuário, que deve responder imediatamente com uma primitiva T-DISCONNECT.Response. Uma PDU CNTL com os bites `WCLOSE`, `RCLOSE` e `END` habilitados é então enviada de volta. Por outro lado, se uma PDU CNTL apenas com o bite `END` habilitado é recebida, uma primitiva T-DISCONNECT.Indication é enviada ao usuário indicando o aborto. Prosseguindo com a desconexão normal, o lado iniciador da conexão, ao receber a PDU CNTL com os bites `WCLOSE`, `RCLOSE` e `END` habilitados gera uma primitiva T-DISCONNECT.Confirm, fechando definitivamente a conexão.

II.5.2 - Multiponto

Da mesma forma que não há uma etapa explícita de estabelecimento de conexões multidestinatárias, não há também uma etapa de liberação. Um usuário pode simplesmente deixar um grupo, através da primitiva T-LEAVE.Request, o que coloca a máquina de estados diretamente em IDLE. Nenhuma notificação de saída é enviada ao servidor ou a qualquer membro do grupo.

II.6 - Transferência de dados

Seja em conexões ponto-a-ponto ou multidestinatárias, o usuário dispõe de apenas uma primitiva de transferência de dados, a primitiva T-DATA.Request. Internamente ao protocolo, existem *buffers* distintos para transmissão e recepção. Ao receber uma primitiva T-DATA.Request, a entidade MCF transmissora pode armazenar ou não os dados do usuário no *buffer* de transmissão, de acordo com as opções de controle de fluxo e de erros, conforme será visto adiante. Os dados são transmitidos à estação remota, ou ao grupo, através de PDUs DATA, seqüencialmente numeradas (começando em zero) com o intuito de permitir a detecção de falhas na recepção por seqüenciamento, e são entregues ao usuário em primitivas T-DATA.Indication.

II.6.1 - Segmentação e remontagem

Função típica de protocolos de transporte, a segmentação é necessária no caso em que o tamanho total dos dados recebidos do usuário, ou o tamanho da SDU (*Service Data Unit*), é maior que o comprimento máximo definido para o segmento de dados de PDUs DATA. Assim, a SDU deve ser segmentada em várias PDUs DATA seqüencialmente numeradas, de forma a permitir sua remontagem na entidade remota. Além disso, a PDU DATA correspondente ao último segmento da mensagem deve conter o bite `EOM` habilitado. Por essa razão, os dados são sempre armazenados no *buffer* de recepção até que o fim da mensagem seja detectado, isto é, até que uma PDU DATA com o bite `EOM` habilitado seja recebida. Se todos os segmentos da mensagem tiverem sido recebidos intactos e na ordem correta, uma primitiva T-DATA.Indication com a mensagem remontada é enviada ao usuário.

II.6.2 - Seqüenciamento

Já foi visto que para garantir a entrega ordenada dos dados, bem como para realizar o controle de erros, detectando PDUs perdidas, um correto seqüenciamento se faz necessário. Com esse objetivo, diferentes variáveis representando números de seqüência foram definidas: *seq* (número de seqüência de uma PDU DATA), *rseq* (número de seqüência imediatamente superior ao número de seqüência da última PDU DATA corretamente recebida) e *dseq* (número de seqüência imediatamente superior ao da última PDU DATA cujos dados foram realmente entregues ao usuário). Entretanto, duas variáveis auxiliares, que devem pertencer ao conjunto das variáveis de estado do receptor, são também importantes para um perfeito seqüenciamento dos dados. São elas:

- *hseq*, que representa o número de seqüência da última PDU DATA recebida (corretamente ou não);
- *recv_sync*, que armazena no receptor o valor do campo *sync* de PDUs CNTL recebidas com o bite `SREQ` habilitado, ou o valor do campo *seq* de PDUs DATA recebidas.

Observação: todas essas variáveis devem ser inicializadas com zero.

II.7 - Gerenciamento da transmissão em um grupo através da concessão de fichas

Como já foi mencionado na Seção II.4.2, o protocolo MCF define basicamente duas classes de entidades que podem participar de comunicações multidestinatárias: as entidades que tem a capacidade de transmitir e receber dados endereçados ao grupo, que serão doravante chamadas de **produtores**, e o **servidor**, que além das características inerentes aos produtores, também tem a capacidade de conceder **fichas**, sem as quais a transmissão não é possível, e armazenar pedidos de fichas. Na verdade, a máquina de estados do protocolo permite que qualquer estação seja um

servidor, bastando para isso ser adequadamente configurada. Mas em cada grupo haverá somente um servidor.

O mecanismo de concessão de fichas é simples. O servidor, cujo endereço deve ser conhecido em toda a rede, fixa *a priori* um endereço de grupo (nada impede que estejam presentes vários servidores em uma rede, cada um tendo fixado um endereço de grupo diferente). A estação que desejar transmitir dados a um grupo deve primeiramente solicitar uma ficha ao servidor correspondente àquele grupo, através de uma primitiva T-TOKEN.Request, que causará o envio de uma PDU PASS ao servidor. Este último, quando recebe um pedido, coloca o endereço da estação solicitante em uma fila FIFO. Se não houver espaço na fila, uma PDU ERROR é retornada. Nessa ocasião, o servidor verifica a disponibilidade de fichas, através de um contador local. Se houver ainda alguma ficha em poder do servidor, o primeiro endereço da fila é retirado, uma PDU TOKEN é enviada a este endereço e um temporizador, o *token timer*, ou TTIMER, é disparado. Este temporizador representa o tempo máximo com que uma estação pode permanecer com a ficha. Ao receber a ficha, na forma de uma PDU TOKEN, a entidade produtora habilita uma variável booleana, *Token_Granted*. Essa variável indica a posse da ficha pela estação, permitindo a transmissão de dados. O protocolo deve notificar ao usuário a concessão da ficha através de uma primitiva T-TOKEN.Indication, autorizando assim o início da transmissão. Quando uma estação terminar de transmitir sua mensagem, ela deve ordenar a devolução da ficha através de uma primitiva T-TOKEN.Response, fazendo com que uma PDU TOKEN seja enviada ao servidor. Este, ao receber a PDU, interrompe o temporizador TTIMER associado àquela ficha. Tão logo uma ficha seja devolvida, o servidor verifica a fila de pedidos e, se não estiver vazia, concede a ficha recém-devolvida à primeira estação da fila.

Entretanto, se o temporizador TTIMER estourar, uma PDU CNTL com o bite `RETTK` habilitado, solicitando a devolução imediata da ficha, é enviada ao endereço correspondente à ficha associada ao temporizador estourado. Um novo temporizador é então disparado, o *wait timer* ou WTIMER, de duração evidentemente bem menor que TTIMER. Na estação, a devolução obrigatória da ficha ocorre ao nível do protocolo (o usuário é apenas notificado, através de uma primitiva T-TOKEN.Indication, que seu tempo de acesso terminou e a ficha foi devolvida ao servidor). Se no decorrer de WTIMER a ficha não for devolvida, este temporizador é sucessivamente reiniciado até completar três tentativas. Se até o estouro de WTIMER pela terceira vez a ficha ainda não tiver sido devolvida (talvez por problemas operacionais na estação remota), o servidor considera aquela ficha perdida e incrementa o contador de número de fichas.

II.8 - Controle de fluxo

II.8.1 - Janela de transmissão

O mecanismo de controle de fluxo por janelas de transmissão implementado pelo protocolo MCF é bem simples. O tamanho da janela corresponde ao tamanho do *buffer* de transmissão, que por sua vez está limitado apenas pela quantidade de memória disponível. Entretanto, os dados do usuário recebidos não são sempre armazenados no *buffer*, mas apenas quando o mecanismo de controle de fluxo selecionado for o de janelas e/ou quando o controle de erros estiver habilitado.

No protocolo MCF, o controle por janelas segue a idéia do controle realizado pelo protocolo NETBLT. Naquele protocolo, além do controle da taxa de transmissão durante o envio de um *buffer*, há uma espécie de controle de fluxo por janela, que impede a transmissão de um novo *buffer* até que a correta recepção do anterior tenha sido reconhecida. Semelhantemente, no protocolo MCF, o *buffer* de transmissão deve ser completamente preenchido para que um pedido de reconhecimento seja enviado, na forma da habilitação do bite `DREQ` no cabeçalho da PDU que

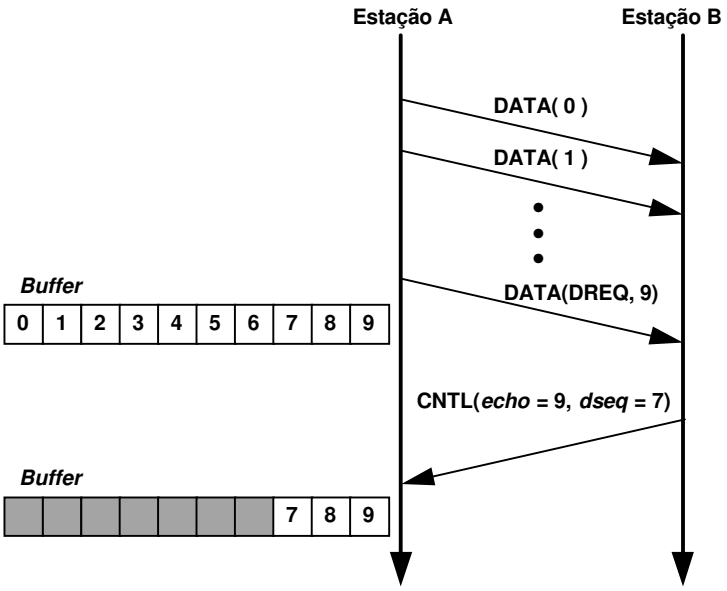

Figura 1: Exemplo de controle de fluxo por janela de transmissão.

carregará o último segmento de dados do *buffer*. O número de seqüência desta PDU é armazenado em uma variável local, *saved_sync*. No receptor, o valor do número de seqüência da PDU DATA contendo o bite `DREQ` habilitado é salvo na variável *rcv_sync* e será copiado para o campo *echo* da PDU CNTL que será ser enviada como resposta ao pedido de reconhecimento, mas apenas após os dados terem sido entregues ao usuário, como dita o bite `DREQ`. No transmissor, o valor do campo *echo* é comparado com *saved_sync*. Se forem iguais, o valor de *dseq* recebido é utilizado para liberar os dados armazenados no *buffer*. Evidentemente, se o controle de erros não estiver habilitado, *dseq* corresponderá à próxima PDU DATA a ser transmitida. Logo, todo o *buffer* será liberado. A Figura 1 ilustra um exemplo, com um *buffer* hipotético de 10 elementos.

A situação de exceção está no caso em que a mensagem do usuário não ocupa integralmente um *buffer*, ou não corresponde a um número inteiro de *buffers*. Nessa ocasião, o mesmo procedimento de pedido de reconhecimento será realizado quando for detectado o final da mensagem, mesmo que o *buffer* não esteja totalmente preenchido.

III.8.2 - Taxa de transmissão

Também a exemplo do NETBLT, o mecanismo de controle de fluxo por taxa de transmissão implementado pelo protocolo MCF utiliza os parâmetros taxa, fornecida em octetos por segundo, e rajada, fornecida em octetos. Esses parâmetros, negociados na etapa de estabelecimento de conexões ponto-a-ponto, ou configurados na adesão de um novo membro a um grupo multiponto, são utilizados para determinar o valor de um temporizador, o *rate timer*, ou RTIMER, através da relação rajada/taxa. O temporizador RTIMER é disparado quando a primeira PDU DATA for enviada na fase de transferência de dados. Uma variável local, **crédito**, é inicializada com o valor da rajada e decrementada cada vez que uma PDU DATA for transmitida, de uma quantidade igual ao comprimento dos dados na PDU. Quando todas as PDUs correspondentes a uma rajada forem transmitidas, nenhuma outra poderá ser enviada até que o temporizador RTIMER estoure e o valor da rajada seja restituída à variável **crédito**.

É claro que o controle de fluxo por taxa realiza na verdade uma redução da vazão máxima alcançável. Esta vazão é determinada com o desligamento de todos os mecanismos opcionais (cálculo de *checksum*, controle de fluxo e de erros), reduzindo assim o processamento ao mínimo necessário. Com isso, tem-se um teto para a vazão, abaixo do qual a escolha adequada de valores

para taxa e rajada permite um controle eficaz. Mas assim os dados não podem ser armazenados no *buffer* de transmissão, pois isso introduziria um processamento não considerado na determinação da vazão máxima.

II.9 - Controle de erros

II.9.1 - *Go-back-N*

Os dois principais mecanismos de recuperação de erros são oferecidos opcionalmente no protocolo MCF: *go-back-N* e retransmissão seletiva. Em ambos os casos, o controle de fluxo por janela deve estar necessariamente habilitado, para que os dados sejam armazenados no *buffer* de transmissão, permitindo assim possíveis retransmissões.

Se o mecanismo *go-back-N* for selecionado, os campos *num_msgs* e *msgs* da PDU CNTL enviada em resposta ao pedido de reconhecimento do transmissor (feito através do bite `DREQ`) devem conter os valores 1 e *hseq*, respectivamente. Dessa forma, o transmissor poderá identificar o pedido de retransmissão por *go-back-N*, entendendo que todas as PDUs DATA com números de seqüência compreendidos no intervalo *rseq* a *hseq* devem ser retransmitidas. Portanto, valores iguais para *rseq* e *hseq* indicarão que nenhum dado precisa ser retransmitido.

II.9.2 - Retransmissão seletiva

Para o mecanismo de retransmissão seletiva, foi implementada uma mistura dos mecanismos utilizados pelos protocolos NETBLT e XTP. No XTP, existem dois campos semelhantes a *num_msgs* e *msgs*, mas que representam, respectivamente, o número de intervalos de octetos corretamente recebidos (*spans*) e os pares de números de seqüência que limitam esses intervalos. Mas como no XTP a numeração é feita sobre octetos, diferentemente do protocolo MCF, em que a numeração é feita sobre mensagens, foi aproveitada a idéia do NETBLT, em que a mensagem de reconhecimento conduz uma lista das mensagens incorretamente recebidas em um *buffer* (na forma de um mapa de bites). Portanto, *num_msgs* conterá o número exato de PDUs DATA cuja retransmissão está sendo solicitada pelo transmissor, ao passo que *msgs* representa a lista de números de seqüência destas PDUs. Essa abordagem inclusive simplifica bastante o processamento do receptor, pois torna mais fácil a atualização da lista de números de seqüência, e do transmissor, que não terá que se preocupar em extrair as falhas (*gaps*) a partir dos intervalos corretos. Pode-se observar também que o tamanho máximo de *msgs* está diretamente ligado ao tamanho do *buffer* de recepção.

Foi visto na seção anterior que, no mecanismo *go-back-N*, *num_msgs* e *msgs* eram sempre preenchidos com 1 e *hseq*, fazendo com que o transmissor tenha que comparar *rseq* e *hseq* para decidir ou não pela retransmissão. No modo seletivo, por outro lado, zero em *num_msgs* indicará que nenhuma retransmissão é necessária.

III.9.3 - Algoritmo *bucket*

O algoritmo *bucket* sugerido pelo protocolo XTP como mecanismo de controle de erros para conexões multidestinatárias também é oferecido como opcional no protocolo MCF. O *bucket* é uma estrutura de dados que armazena algumas variáveis de estado relevantes à comunicação, essencialmente números de seqüência. Um temporizador denominado *switching time* (ST) é utilizado para "envelhecer" o *bucket*, em um processo que será explicado a seguir. Esse temporizador visa dar tempo suficiente para que múltiplos receptores respondam a pedidos de *status*

do transmissor, sendo calculado com base em estimativas de tempo de ida e volta. Quando o temporizador ST estoura, o transmissor envia uma mensagem CNTL de pedido de reconhecimento (bite `SREQ` habilitado) e um *bucket* vazio é instanciado com o valor do campo *sync* contido no CNTL transmitido. Normalmente, *sync* recebe o valor de um contador local incrementado toda vez que uma mensagem de dados é transmitida. Este *bucket* armazenará então informações combinadas das mensagens CNTL recebidas em resposta à solicitação do transmissor, desde que possuam o campo *echo* igual ao índice, ou *label*, do *bucket*. O transmissor deve ter um vetor de *buckets* prontos para uso. Um novo *bucket* somente é alocado após o estouro de ST, quando uma nova mensagem CNTL é transmitida e ST é reiniciado. Enquanto houver *buckets* vazios, o estouro de ST causa apenas a instanciação de um novo *bucket*. Mas quando todos os *buckets* estiverem preenchidos, a informação do *bucket* mais "velho" (com menor índice) é utilizada para atualizar o estado do transmissor, que libera parte do *buffer* de transmissão de acordo com essa informação. Este *bucket* pode assim ser reutilizado e reinicializado com o valor de *sync* contido no CNTL enviado nessa ocasião, passando a ser o *bucket* mais "novo" (com maior índice). Qualquer CNTL recebido com um valor de *echo* menor que o menor índice de *bucket* existente é imediatamente descartado. Dessa forma, o algoritmo pune os receptores com menor capacidade de processamento, conferindo uma certa homogeneidade ao grupo.

Apesar de ser um eficiente mecanismo de controle de erros que possibilita um melhor gerenciamento de retransmissões em grupos de tamanho arbitrário, pode ser também de certa forma utilizado para controlar o fluxo da transmissão, dependendo do tamanho do *buffer* disponível. Se o *buffer* de transmissão for completamente preenchido antes que o *bucket* mais "velho" seja liberado e sua informação utilizada para atualizar o estado do transmissor, liberando os dados armazenados até *dseq*, nenhuma nova PDU DATA poderá ser transmitida. O tamanho do *buffer* de transmissão, o *switching time* (ST) e o número de *buckets* disponíveis podem ser ajustados de maneira a que o *buffer* nunca fique cheio, garantindo assim uma maior continuidade na transmissão. A relação entre esses parâmetros pode ser equacionada através da seguinte fórmula [PEI92]:

Tamanho do *buffer* = número de *buckets* * ST * B,

onde B é a banda-passante da rede.

Logo, como o algoritmo *bucket* "amarra" de certa forma o controle de fluxo, o mecanismo de janelas de transmissão não é oferecido como opção pelo protocolo MCF para conexões multidestinatárias. Além disso, o mecanismo de recuperação de erros utilizado em conjunto com o algoritmo *bucket* é geralmente o mecansimo *go-back-N*, pois a utilização de um mecanismo seletivo implicaria em um tamanho variável para o *bucket* ou a alocação de um tamanho máximo, prejudicando o processamento.

III - Análise de Desempenho

O protocolo MCF foi implementado no contexto da arquitetura de implementação proposta pelo Grupo de Teleinformática e Automação (GTA) da UFRJ [Bagi91a] [Bagi91b] [Bagi92] [Nune94a]. Esta arquitetura define interfaces padrão a serem utilizadas para a comunicação entre entidades adjacentes, em um sistema com divisão por camadas, bem como módulos especializados de gerenciamento de memória, escalonamento de tarefas e gerenciamento de temporizações, chamados **módulos globais**. O sistema está codificado em linguagem C e roda em computadores PC-compatíveis sob o sistema operacional DOS. O código fonte totaliza mais de 8400 linhas, gerando um programa executável de aproximadamente 170 koctetos, com informação de depuração. O ambiente de medidas consiste em PCs equipados com CPUs 486 DX conectadas a uma rede local Ethernet através de placas padrão NE2000.

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 2 mostra a vazão, em Mbps, para os usuários das camadas MAC e MCF, além da vazão obtida exclusivamente pelo *packet driver*. Para este último, foi medida a vazão nas seguintes configurações:

1) nenhum mecanismo opcional habilitado;
2) apenas o cálculo de *checksum* habilitado. 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 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.

Como mostra a Figura 2, à medida que o tamanho da mensagem aumenta, a vazão tende a um máximo de aproximadamente 6,2 Mbps para o *packet driver*. Observa-se que a camada MAC atinge quase a mesma vazão do *driver*, de onde se conclui que o *overhead* por ela introduzido

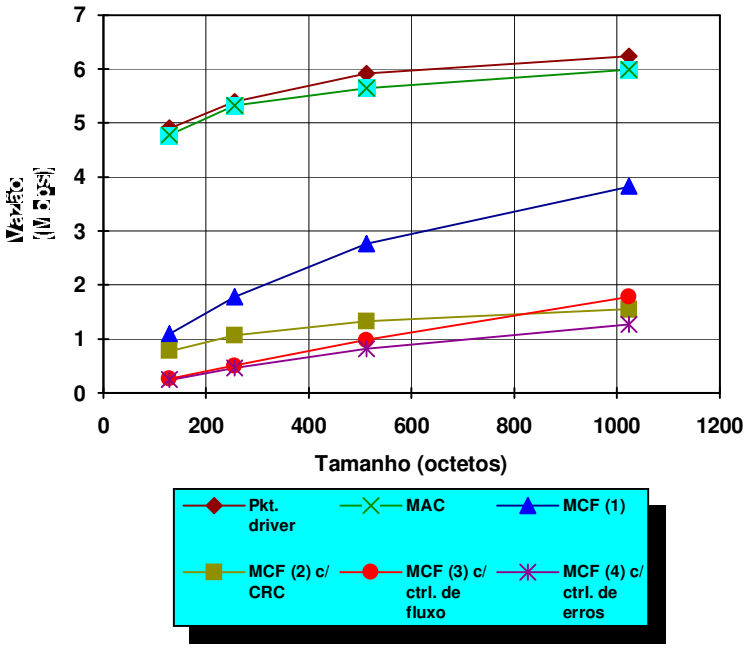

Figura 2: Vazão × tamanho da mensagem.

(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 3), 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*.

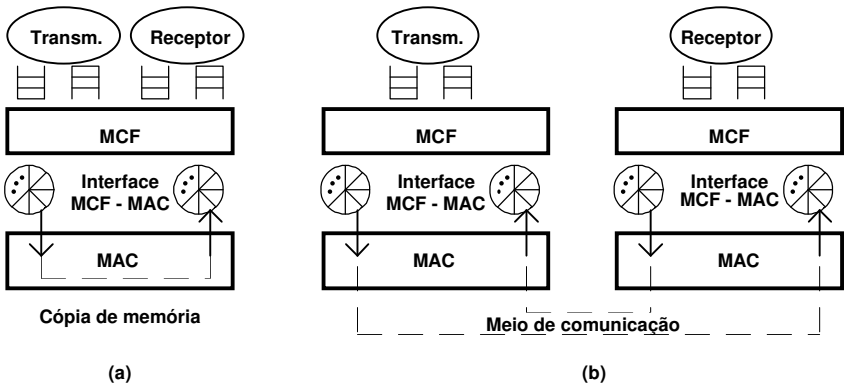

Figura 3: Comparação do funcionamento do sistema em *loopback* (a) com o funcionamento normal via rede (b).

Ainda sobre a Figura 2, 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 2 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 2 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 4, 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.

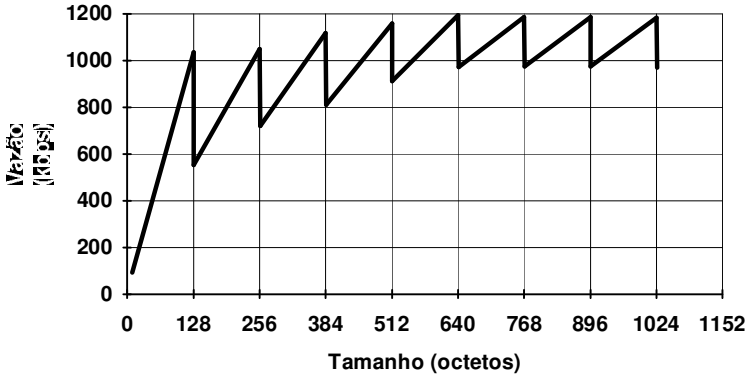

Figura 4: Vazão × tamanho da mensagem com segmentação em 128 octetos.

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 2 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*. 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 2: 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

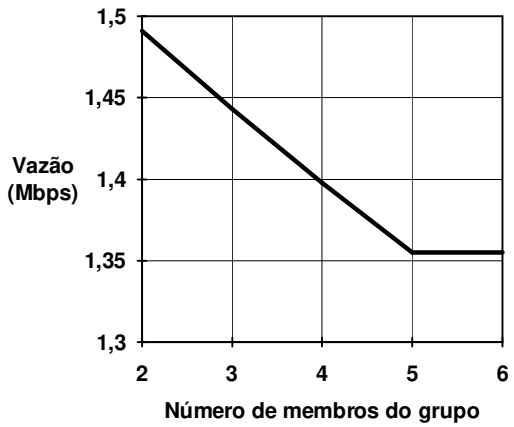

Figura 5: Vazão × tamanho do grupo multiponto.

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 5. 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, mostrando que as PDUs CNTL por ele geradas não prejudicam significativamente o desempenho do transmissor. 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. 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*. 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.

IV - Conclusões

Neste trabalho foi apresentado um protocolo multiponto-a-multiponto com configuração dos parâmetros da comunicação e com gerenciamento de acesso baseado em fichas, o protocolo MCF. Com a popularização dos serviços multimídia, há um sentimento generalizado de que os próximos protocolos deverão ser adaptáveis aos diferentes requisitos de desempenho de cada aplicação. Neste sentido, o protocolo MCF oferece a possibilidade de seleção dos mecanismos que eram outrora inerentes aos protocolos de transporte convencionais:

- Cálculo e verificação de *checksum* opcionais sobre o campo de dados do usuário em PDUs DATA.
- Dois mecanismos de controle de fluxo: janelas de transmissão e controle de taxa.
- Dois mecanismos de recuperação de erros: *go-back-N* e retransmissão seletiva.
- Algoritmo *bucket* opcional para comunicações multidestinatárias.

Assim, uma aplicação de transferência de arquivos pode optar por comprometer um pouco seu desempenho em troca de maior confiabilidade para os dados transmitidos, acionando o cálculo de *checksum*, o controle de fluxo e a recuperação de erros, enquanto um serviço de datagrama mais preocupado com o aproveitamento máximo da banda-passante disponível pode optar por não acionar nenhum mecanismo opcional. Por outro lado, o controle de fluxo através do controle da taxa de transmissão mostra-se adequado à aplicações de tempo real em que rígidos requisitos de banda-passante e atraso se fazem necessários. Apesar de já apresentar um bom desempenho, a precisão do controle de fluxo por taxa pode ser melhorada mediante a programação direta do controlador de temporizações do PC, reduzindo o intervalo do *timer tick*.

Por maior que possa parecer uma perda de aproximadamente 37% em relação ao desempenho obtido por um usuário da camada MAC, esta perda é causada principalmente pela inevitável cópia dos dados em uma região contígua de memória para atender às exigências dos controladores de comunicação. Mesmo assim, uma vazão de 1,28 Mbps é conseguida para uma conexão ponto-a-ponto em uma configuração com *checksum*, controle de fluxo por janela e recuperação de erros por retransmissão seletiva, ou seja, o máximo *overhead* de processamento possível.

Quanto às suas características multidestinatárias, o algoritmo *bucket* permite que grupos arbitrariamente grandes sejam formados. Uma vazão de aproximadamente 1,36 Mbps é conseguida com o algoritmo *bucket* em um grupo de seis participantes. Este algoritmo implica em cálculo de *checksum* e em recuperação de erros por *go-back-N*, apresentando um desempenho ainda melhor que o de uma conexão ponto-a-ponto com configuração semelhante. Além disso, o caráter notadamente cliente/servidor da implementação, com um servidor de fichas de endereço fixo, permite uma constante alteração no tamanho do grupo, sem afetar o andamento da comunicação. O mecanismo de fichas funcionou corretamente para duas fichas, sendo que este número pode ser facilmente aumentado, permitindo assim que um número arbitrário de membros possa transmitir simultaneamente seus dados para o grupo. Entretanto, a maneira como as aplicações administrarão os dados recebidos de diferentes fontes foge ao escopo deste trabalho.

Referências

[Bagi91a] L.F. Baginski, F.M.C. de Barros, R. Schatzmayr e O.C.M.B. Duarte, "Implementação e Análise de Desempenho em um Sistema de Transferência de Dados", *X Congresso da Sociedade Brasileira de Computação*, Vitória, Brasil, pp. 157-169, julho de 1991.

[Bagi91b] L.F. Baginski e O.C.M.B. Duarte, "Um Modelo de Implementação de Alto Desempenho para Sistemas Abertos", *IX Congresso da Sociedade Brasileira de Telecomunicações*, São Paulo, Brasil, pp. 1941-1945, setembro de 1991.

[Bagi92] L.F. Baginski, "Ambiente de Implementação para Sistemas de Alto Desempenho", *Tese de Mestrado*, PEE-COPPE/UFRJ, janeiro de 1992.

[Cohn91] M. Cohn, "High Speed Transport Protocol Specification", *Contribution to ISO/IEC JTC1 SC6/WG4 on the High Speed Transport Protocol*, setembro de 1991.

[Diot] C. Diot, "A survey of Multicast Services and Protocols".

[Doer90] W.A. Doeringer, D. Dykeman, M. Kaiserswerth, B.W. Meister, H. Rudin e R. Williamson, "A Survey of Light-Weight Transport Protocols for High-Speed Networks", *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 2025-2039, novembro de 1990.

[Nune94a] M.D. Nunes, C.V.N. Albuquerque e O.C.M.B. Duarte, "Performance Measurements in a Manufacturing Communication System", *ISCAS '94*, Londres, maio de 1994.

[Nune94b] M.D. Nunes e O.C.M.B. Duarte, "Análise de Mecanismos para Protocolos de Alto Desempenho", *VI Simpósio Brasileiro de Arquitetura de Computadores e Computação de Alto Desempenho*, Caxambu, 1-5 de agosto de 1994.

[PEI92] Protocol Engines Inc., "Xpress Transfer Protocol Definition", Revisão 3.6, janeiro de 1992.

[Sant92] H. Santoso e S. Fdida, "Transport Layer Multicast: XTP Bucket Error Control and Its Enhancement", *4th. IFIP Conference on High Performance Networking*, Liège, Bélgica, 16-18 de dezembro de 1992.

[Stra92] W.T. Strayer, B.J. Dempsey e A.C. Weaver, "XTP: The Xpress Transfer Protocol", Addison Wesley Publishing Company, Inc., 1992.

[Wats81] R.W. Watson, "Timer-Based Mechanisms in Reliable Transport Protocol Connection Management", *Computer Networks*, vol. 5, 1981.

[Wats87] R.W. Watson e S.A. Mamrak, "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices", *ACM Transactions on Computer Systems*, vol. 5, no. 2, pp. 97-120, maio de 1987.