Capítulo III

O Protocolo MCF

Neste capítulo, são comparados os mecanismos descritos no Capítulo I, destacando suas vantagens e desvantagens, numa tentativa de delinear o perfil de um protocolo de transferência de propósito geral, visando alto desempenho. Também é apresentada uma descrição do protocolo proposto, com os formatos das PDUs (*Protocol Data Units*) e os mecanismos de gerenciamento de conexões, controles de fluxo e de erros.

III.1 - Comparação de mecanismos

III.1.1 - Gerenciamento de conexões

Dois mecanismos de gerenciamento de conexões foram apresentados no Capítulo I: negociação (*handshake*) e conexões implícitas baseadas em temporizadores. Conexões negociadas têm a desvantagem de introduzir 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, como transmissão de vídeo em tempo real, 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*). Tal parâmetro, entretanto, pode ser altamente variável de acordo com as condições de tráfego na rede, exigindo uma interação constante entre o protocolo e a rede subjacente no sentido de atualizar o temporizador. Este procedimento é altamente oneroso quando o interesse está em simplificar o processamento no sentido de aumentar o desempenho. 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 nas Seção III.5.2. A principal vantagem do *handshake* que pesou nessa decisão é a possibilidade de negociar **parâmetros** da conexão, como tipo de controle de fluxo e de erros, pois a diversidade das aplicações atuais com diferentes exigências de banda-passante e confiabilidade, entre outras, pede um protocolo **configurável**, a exemplo do protocolo XTP.

III.1.2 - Reconhecimentos

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. Por outro lado, a vantagem da geração de reconhecimentos independente do transmissor é que o receptor fica totalmente liberado de qualquer processamento extra para determinar o momento em que um reconhecimento deve ser gerado. Isso, é claro, às custas da manutenção de um temporizador no receptor. Logo, devido à complexidade introduzida pela manipulação de temporizadores, mais necessários em outros mecanismos, como será visto adiante, optou-se pela geração de reconhecimentos submetida à comandos explícitos do transmissor.

III.1.3 - Controle de fluxo

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 transbordo de memória. 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. Evidentemente, tal controle é somente válido se a granularidade da temporização permitir um mínimo de precisão na taxa efetivamente obtida, e se a rede subjacente puder interagir com o protocolo para a determinação de valores iniciais de taxa, rajada ou intervalo entre mensagens (*interpacket gap*), e também permitir a posterior atualização desses valores para ajustar a taxa às condições de tráfego, evitando assim congestionamentos. Sem uma rede com essa capacidade, o controle de fluxo por taxa ainda é possível, mas os parâmetros devem ser negociados na fase de estabelecimento da conexão, não podendo ser alterados até que uma

nova conexão seja estabelecida. 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.

III.1.4 - Controle de erros

Na Seção III.1.2, chegou-se a conclusão que o mais simples mecanismo de geração de reconhecimentos é a submissão a comandos do transmissor, pois evita o uso de temporizadores. Um simples bite do cabeçalho da última PDU de dados de uma mensagem pode servir como pedido de reconhecimento. Portanto, 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, ambos os mecanismos analisados possuem vantagens e desvantagens, qualificando-os para uma ou outra aplicação. 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. No entanto, a degradação no desempenho causada por tal fato será dificilmente notada em uma rede razoavelmente confiável. 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. Logo, a implementação suporta os dois mecanismos. Além disso, números de seqüência nas PDUs são utilizados para detecção de perdas e *checksums* são utilizados para detecção de erros, separadamente para o cabeçalho e para o segmento de dados (este último, opcional na PDU de dados do usuário).

III.1.5 - Semântica das conexões multidestinatárias

É necessário agora definir o grau de confiabilidade a ser oferecido em conexões ponto-a-multiponto. Conexões multidestinatárias não confiáveis se destinam melhor a aplicações de tempo real que não possam esperar por retransmissões, uma vez que longos atrasos são inerentes a esse mecanismo. Conexões atômicas ou totalmente confiáveis exigem um conhecimento do tamanho do grupo e dos endereços de todos os seus membros pela entidade transmissora. Entretanto, como atualmente as distâncias são encurtadas pela Internet, onde são possíveis conferências entre membros de diversos países, qualquer limitação no tamanho de um grupo é inadmissível. Quanto ao desempenho, com conexões totalmente confiáveis a

velocidade da transmissão pode ser seriamente comprometida, pois 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*, discutido em detalhes na Seção II.5.1, permite a implementação de conexões estatisticamente confiáveis, tendo sido portanto utilizado no protocolo proposto. Mas resta ainda decidir a melhor maneira de oferecer conexões **multiponto-a-multiponto**, que é um dos objetivos do protocolo implementado. Os protocolos VMTP, XTP, ST-II, MTP e RAMP analisados no capítulo anterior se referem apenas a conexões ponto-a-multiponto. Porém, o protocolo MTP, com seu mecanismo de fichas, permite que haja vários transmissores em um grupo. Mas, em cada instante, apenas um transmissor estará em poder da ficha e, portanto, enviando dados, enquanto todos os demais estarão apenas "ouvindo". E isso não é o bastante pois, por definição, em uma conexão multiponto-a-multiponto, uma ou mais entidades poderão estar transmitindo **simultaneamente**. No protocolo proposto, 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 trabalho será doravante referenciado pela sigla que resume suas principais características: **MCF** (protocolo **M**ultidestinatário **C**onfigurável com gerenciamento de acesso por **F**ichas de permissão).

III.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 [Cohn91], que utilizam campos alinhados em 32 bites. O HSTP consiste basicamente no protocolo XTP mapeado sobre a estrutura RM-OSI, ou seja, define funções e procedimentos idênticos ao XTP. No presente trabalho, alguns campos definidos por aqueles protocolos foram suprimidos, da mesma forma em que outros foram criados ou adaptados.

III.2.1 - Cabeçalho

Todas as PDUs definidas neste protocolo utilizam o mesmo cabeçalho, de tamanho fixo, com sete campos de 32 bites, mostrado na Figura III.1.

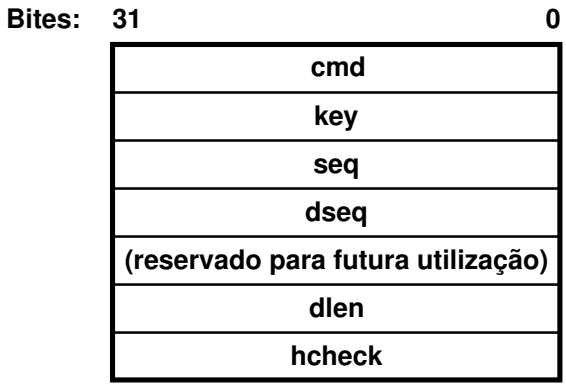

Figura III.1: Cabeçalho das PDUs.

Campo** cmd **- Command

O campo *cmd* (Figura III.2) contém informações quanto ao tipo da PDU (1 octeto), o número de octetos (1 octeto) utilizados para completar o campo de dados do usuário (em PDUs FIRST ou DATA), tornando seu comprimento múltiplo de 32 bites (*padding*), e as opções de configuração do protocolo (2 octetos).

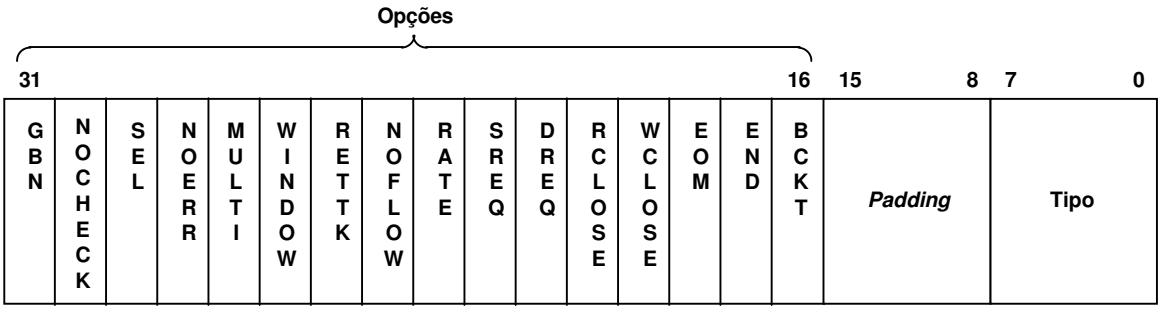

Figura III.2: Campo *cmd*.

Tipos de PDU:

- 0x00: DATA PDU de dados do usuário;
- 0x01: CNTL PDU de informações de controle;
- 0x02: FIRST PDU de estabelecimento de conexão ponto-a-ponto (também pode conter dados do usuário);
- 0x04: TOKEN PDU representativa da ficha em conexões multidestinatárias;

- 0x06: PASS PDU de pedido de ficha ao servidor;
- 0x08: ERROR PDU de diagnóstico em situações de erro.

Opções de configuração:
Os dois octetos mais significativos do campo *cmd* constituem um conjunto de 16 bites configuráveis de acordo com a escolha do usuário. São eles:

`GBN`	Habilitado, indica opção pelo mecanismo *go-back-N* para recuperação de mensagens.
`NOCHECK` (*Suspend CRC checking*)	Habilitado, indica que o *checksum* sobre os **dados** não deve ser calculado.
`SEL` (*Selective retransmission*)	Habilitado, indica opção pelo mecanismo de retransmissão seletiva para recuperação de mensagens.
`NOERR` (*Suspend retransmission*)	Habilitado, indica que nenhum controle de erros será realizado. Nesse caso, os bites `GBN`, `SEL` e `BCKT` (para conexões multidestinatárias) não são considerados.
`MULTI` (*Multicast operation*)	Habilitado, indica conexão multidestinatária
`WINDOW`	Habilitado, indica opção pelo mecanismo de janelas de transmissão para controle de fluxo.
`RETTK` (*Return Token*)	Habilitado em PDUs CNTL enviadas pelo servidor para que uma determinada estação devolva a ficha que está em seu poder.
`NOFLOW` (*Suspend flow control*)	Habilitado, indica que nenhum controle de fluxo será realizado. Nesse caso, os bites `WINDOW` e `RATE` não são considerados.
`RATE`	Habilitado, indica opção pelo mecanismo de controle de fluxo através da taxa de transmissão.
`SREQ` (*Status request*)	Habilitado em PDUs DATA ou CNTL, indica que a entidade destinatária deve enviar imediatamente uma PDU CNTL como resposta. Serve portanto como pedido de reconhecimento pela entidade transmissora.

`DREQ` (*Delivery status request*)	Semelhante ao `SREQ`, com a diferença que a PDU CNTL é somente enviada após a entrega, ao usuário, de quaisquer mensagens corretamente recebidas que ainda estejam armazenadas em *buffers*.
`RCLOSE` (*Reader closed*)	Utilizado no fechamento de uma conexão ponto-a-ponto, indica à entidade remota o "fechamento do canal de leitura", ou seja, PDUs DATA não serão mais aceitas.
`WCLOSE` (*Writer closing*)	Análogo ao anterior, indica o "fechamento do canal de escrita", ou seja, PDUs DATA não serão mais transmitidas.
`EOM` (*End of message*)	Deve ser habilitado na última PDU DATA de uma mensagem.
`END`	Habilitado, indica o fechamento definitivo da conexão.
`BCKT`	Habilitado, indica a opção pelo mecanismo de *buckets* para controle de erros em conexões multidestinatárias.

***Campo** key*

O campo *key* é utilizado para associar uma PDU a uma determinada conexão. Em outras palavras, *key* é o número da conexão.

***Campo** seq* **- Current sequence number**

O campo *seq* representa o número de seqüência de uma PDU DATA. Este campo deve conter o valor **zero** em qualquer outra PDU.

***Campo** dseq* **- Delivered sequence number**

O campo *dseq* também deve ser interpretado em PDUs DATA e CNTL, e representa o número de seqüência imediatamente superior ao da última PDU DATA cujos dados foram realmente entregues ao usuário. Quando recebido pelo transmissor, indica que todas as PDUs armazenadas em *buffers* para retransmissão, com números de seqüência até *dseq* **exclusive**, podem ser liberadas.

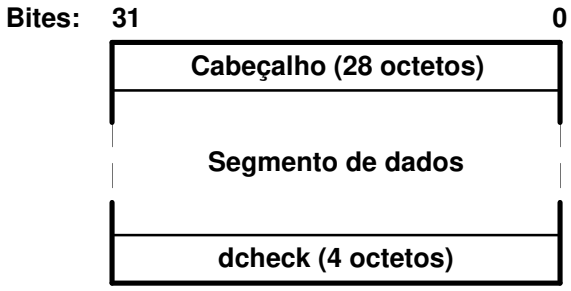

Figura III.3: Formato de uma PDU.

Campo** dlen **- Data length

O campo *dlen* representa o número total de octetos do segmento de dados. No caso de segmentos de informação (PDUs FIRST e DATA), inclui também quaisquer octetos de *padding* que tenham sido acrescentados para tornar o campo de dados do usuário múltiplo de 32 bites.

Campo** hcheck **- Header checksum

O campo *hcheck* contém o cálculo da função de *checksum* (maiores detalhes na Seção III.4.1) sobre os sete campos de 4 octetos do cabeçalho. Esse *checksum* é obrigatório e não pode ser desabilitado pelo bite `NOCHECK`.

III.2.2 - Cauda

A cauda das PDUs consiste em apenas um campo de 4 octetos, *dcheck*. O campo *dcheck* corresponde à função de *checksum* aplicada aos octetos do segmento de dados da PDU, de tamanho variável, com diferentes formatos para diferentes PDUs. Por exemplo, em PDUs DATA o segmento de dados consiste apenas dos dados do usuário, enquanto que em PDUs CNTL o segmento de dados contém vários campos correspondentes às variáveis de estado relevantes à conexão, como números de seqüência, parâmetros de taxa de transmissão, etc. (o segmento de dados de todas as PDUs será examinado em detalhes nas próximas seções). Nas PDUs FIRST, CNTL, PASS, TOKEN e ERROR, o cálculo de *dcheck* é obrigatório. Apenas na PDU DATA o seu cálculo é facultativo, sendo regido pela configuração do bite `NOCHECK`. A Figura III.3 mostra o formato geral de uma PDU.

III.2.3 - PDU FIRST

A PDU FIRST é utilizada para estabelecer conexões no modo **ponto-a-ponto**, podendo conter dados do usuário para aplicações transacionais, por exemplo. Seu segmento de dados (Figura III.4) contém os seguintes campos:

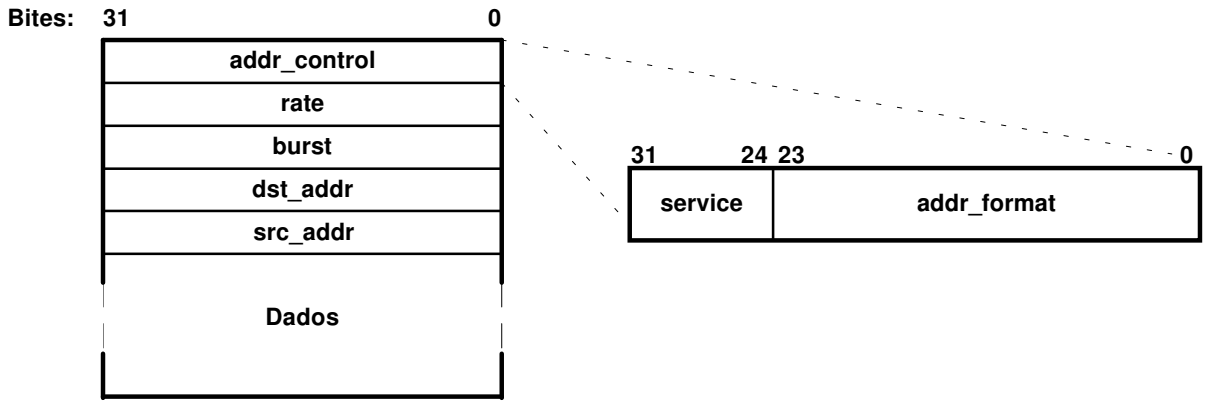

Figura III.4: Segmento de dados da PDU FIRST.

Campo** addr_control **- Address Control

O campo *addr_control* subdivide-se em dois outros campos: *service* e *addr_format*. O campo *service* é usado para especificar o modo de operação da comunicação. Na versão atual, apenas dois modos estão disponíveis: modo 0, para conexões ponto-a-ponto, e modo 1, para conexões multidestinatárias. Futuras extensões do protocolo poderão incluir, por exemplo, transações e datagramas. O campo *addr_format* identifica a sintaxe de endereçamento de acordo com a Tabela III.1. Como a atual versão está implementada sobre uma sub-camada MAC padrão IEEE 802.3, conforme será visto no Capítulo IV, o valor de *addr_format* utilizado é 0x04.

Tabela III.1: Valores de *addr_format*.

Valor	Sintaxe de endereçamento
0	Nenhum endereço
0x01	Endereçamento IP
0x02	Padrão ISO
0x03	XNS
0x04	Estilo IEEE 802
0x05	MODSIM
0x06	Endereçamento NetBios

***Campo** rate*

O campo *rate* das PDUs FIRST permite ao iniciador da conexão sugerir à entidade remota um valor inicial para a taxa de transmissão (em octetos por segundo), que poderá ser aceita ou não. Na Seção III.5.1, o procedimento de negociação de parâmetros será visto em detalhes. Se o procedimento de controle de fluxo por taxa não estiver sendo utilizado, os quatro octetos do campo *rate* devem ser preenchidos com o caracter 0xFF.

***Campo** burst*

O campo *burst* das PDUs FIRST permite ao iniciador da conexão sugerir à entidade remota um valor inicial para a rajada de mensagens (em octetos), que poderá ser aceita ou não. Se o procedimento de controle de fluxo por taxa não estiver sendo utilizado, os quatro octetos do campo *burst* devem ser preenchidos com o caracter 0xFF.

Campo** dst_addr **- Destination address

O campo *dst_addr* contém o identificador lógico da entidade respondedora da conexão.

Campo** src_addr **- Source address

O campo *src_addr* contém o identificador lógico da entidade iniciadora da conexão.

Campo de dados

O campo de dados tem tamnho variável e é utilizado para transmitir os dados do usuário. Esse campo é opcional em PDUs FIRST. Se não estiver presente, o campo *dlen* do cabeçalho deve ser preenchido com o valor 20, correspondente aos 5 campos mencionados acima. Se o tamanho total dos dados não for múltiplo de 4 octetos, um número de 0 a 4 octetos (com o valor 0x00) deve ser acrescentado ao final do campo de dados para alinhá-lo em 32 bites. O número de octetos acrescentados deve constar do campo *padding* do cabeçalho.

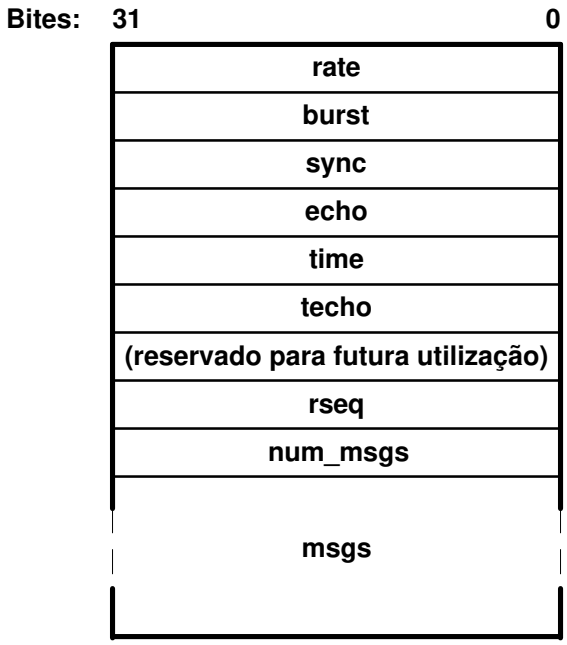

Figura III.5: Segmento de dados da PDU CNTL.

III.2.4 - PDU DATA

É na PDU DATA onde os dados do usuário são efetivamente transmitidos. Seu segmento de dados consiste apenas no campo de dados visto na seção anterior, incluindo os octetos de *padding* que venham a ser necessários.

III.2.5 - PDU CNTL

A PDU CNTL é utilizada para troca de informações de controle entre entidades pares, como números de seqüência, reconhecimentos, parâmetros de taxa de transmissão, entre outras. Seu segmento de dados (Figura III.5) contém os seguintes campos:

***Campo** rate*

O campo *rate* das PDUs CNTL permite ao respondedor da conexão sugerir à entidade iniciadora uma contra-proposta para a taxa de transmissão (em octetos por segundo), que poderá ser aceita ou não. Na Seção III.5.1, o procedimento de negociação de parâmetros será visto em detalhes. Se o procedimento de controle de fluxo por taxa não estiver sendo utilizado, os quatro octetos do campo *rate* devem ser preenchidos com o caracter 0xFF.

***Campo** burst*

O campo *burst* das PDUs CNTL permite ao respondedor da conexão sugerir à entidade iniciadora uma contra-proposta para a rajada de mensagens (em octetos), que poderá ser aceita ou não. Se o procedimento de controle de fluxo por taxa não estiver sendo utilizado, os quatro octetos do campo *burst* devem ser preenchidos com o caracter 0xFF.

Campo** sync **- Control synchronization

O campo *sync* é usado para permitir a sincronização de estado entre duas ou mais entidades através da troca de PDUs CNTL (vide Seção II.5.1 sobre o algoritmo *bucket*). *Sync* recebe o valor de uma variável local ao transmissor que é somente incrementada quando PDUs DATA são transmitidas.

Campo** echo **- Control echo

O campo *echo* é utilizado em conjunto com *sync* para permitir a sincronização de estado entre duas ou mais entidades. Quando o receptor envia uma PDU CNTL em resposta à alguma solicitação do transmissor, o valor de *sync* recebido deve ser copiado para o campo *echo*. Dessa forma, o transmissor poderá associar aquela resposta ao seu pedido de *status* ou reconhecimento. É importante lembrar que PDUs DATA com o bite `SREQ` habilitado também funcionam como pedidos de reconhecimento. Nesse caso, o número de seqüência da PDU DATA recebida é considerado como um valor de *sync*.

Campo** time **- Round trip time

O campo *time* é usado para calcular uma estimativa do tempo de ida-e-volta (*round trip time*) através de uma operação de sincronização por meio do par *sync*/*echo*.

Campo** techo **- Time echo

O campo *techo* é usado em conjunto com *time* para calcular uma estimativa do tempo de ida-e-volta (*round trip time*) através de uma operação de sincronização por meio do par *sync*/*echo*. Quando o receptor envia uma PDU CNTL em resposta a algum pedido do transmissor, o valor de *time* recebido é copiado para o campo *techo*. No transmissor, a diferença entre *techo* recebido e *time* enviado corresponde à estimativa do tempo de ida-e-volta.

Campo** rseq **- Received sequence number

O campo *rseq*, como já foi dito anteriormente, representa o número de seqüência imediatamente superior ao número de seqüência da última PDU DATA corretamente recebida. Diferentemente de *dseq*, entretanto, não significa que os dados até este ponto tenham sido entregues ao usuário.

Campo** num_msgs **- Number of message sequence numbers

O campo *num_msgs* contém o total de números de seqüência de PDUs DATA presente no campo *msgs*. Se nenhum número de seqüência estiver presente em *msgs*, *num_msgs* deve obrigatoriamente ser preenchido com 0.

Campo** msgs **- Message sequence numbers

O campo *msgs* é utilizado pelas entidades receptoras para a recuperação de erros através de retransmissões seletivas. De tamanho variável, consiste em uma lista dos números de seqüência das PDUs DATA que tenham sido incorretamente recebidas ou perdidas. A quantidade desses números deve constar do campo *num_msgs*.

III.2.6 - PDU TOKEN

A PDU TOKEN representa a ficha de permissão de transmissão para conexões multidestinatárias. A ficha deve ser solicitada diretamente ao servidor e deve ser devolvida ao final da transmissão, ou é tomada de volta pelo servidor ao estouro de um temporizador. Seu segmento de dados consiste simplesmente nos campos *dst_addr* e *src_addr*, discutidos na Seção III.2.3.

III.2.7 - PDU PASS

A PDU PASS é utilizada pelos participantes de um grupo para solicitarem a ficha ao servidor. A resposta à essa PDU é a PDU TOKEN, se houver fichas disponíveis. Seu segmento de dados é igual ao da PDU TOKEN.

III.2.8 - PDU ERROR

A PDU ERROR é utilizada para fornecer diagnósticos para diversas situações de erro do protocolo. Seu segmento de dados (Figura III.6) contém os seguintes campos:

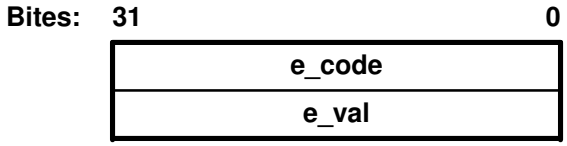

Figura III.6: Segmento de dados da PDU ERROR.

Campo** e-code **- Error code

Esse campo especifica as condições de erro que causam o envio de PDUs ERROR, segundo a Tabela III.2.

Campo** e-val **- Error value

O campo *e-val* destina-se a fornecer informações complementares para cada código definido em *e-code*. Não está implementado nesta versão.

Tabela III.2: Condições de erro.

e-code	Condição de erro
0	Operação inválida
1	Operação recusada
2	Destino desconhecido
3	Usuário inativo
4	Não há recursos
5	Erro do protocolo
6	Conexão abortada
7	Fila cheia no servidor
8	Fim do tempo de acesso (ficha deve ser tomada de volta)
9	Ficha enviada pelo servidor e não recebida pela estação
10	Ficha devolvida ao servidor mas não recebida

III.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 III.3 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 III.3: 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

Tabela III.3: Primitivas de serviço (continuação).

Primitivas	Descrição	Parâmetros
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

III.4 - Validação de PDUs

A fase de validação corresponde ao descarte de PDUs que não obedecerem aos seguintes critérios:

- o identificador de tipo da PDU deve ser um dos seis definidos em III.2.1;
- o valor do *checksum* calculado sobre o cabeçalho deve ser igual ao valor de *hcheck*;
- o valor do *checksum* calculado sobre o segmento de dados de PDUs FIRST, CNTL, ERROR, PASS e TOKEN deve ser equivalente ao valor de *dcheck*;
- se `NOCHECK` não estiver habilitado, o valor do *checksum* calculado sobre o segmento de dados de PDUs DATA deve ser equivalente ao valor de *dcheck*.

III.4.1 - *Checksum*

A função de *checksum* implementada foi o algoritmo CXOR utilizado pelo protocolo XTP [PEI92]. O resultado CXOR, de 32 bites, consiste na concatenação de dois resultados de 16 bites: XOR (parte baixa) e RXOR (parte alta). Com os dados alinhados em 32 bites, XOR é calculado sobre os 16 bites menos significativos e RXOR é calculado sobre os 16 bites mais significativos. O procedimento é o seguinte para o cálculo de XOR:

- o valor inicial de XOR deve ser zero;
- para cada palavra de dados de 16 bites, é feito um ou-exclusivo sobre XOR.

E para o cálculo de RXOR:

- o valor inicial de RXOR deve ser zero;
- para cada palavra de dados de 16 bites, é feito um ou-exclusivo sobre RXOR após este último ter sido deslocado por 1 bite para a esquerda.

III.5 - Abertura de conexão

III.5.1 - 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 (Seção III.2.8). Ao receber esta PDU, o iniciador da conexão envia ao usuário uma primitiva T-DISCONNECT.Indication, fechando a conexão.

A Tabela III.4 representa a tabela de estados para o estabelecimento de conexões ponto-a-ponto.

Estados:

- Idle: estado de repouso, conexão fechada.
- Listen: lado respondedor da conexão espera resposta do usuário.
- Connect-Conf: lado iniciador da conexão espera resposta da entidade par.
- Active: estado de transferência de dados, conexão aberta.

Predicados:

- p01: aceitação do pedido de conexão.

Ações:

- [01]: configurar opções da conexão.
- [02]: enviar T-CONNECT.Indication.
- [03]: enviar T-CONNECT.Confirm.
- [04]: enviar T-DISCONNECT.Indication.
- [05]: enviar FIRST.
- [06]: enviar CNTL.
- [07]: enviar ERROR(*e-code* = 1)

Tabela III.4: Estabelecimento de conexão ponto-a-ponto

ESTADOS EVENTOS	IDLE	LISTEN	CONNECT-CONF
T-CONN.Req	[01] [05] CONNECT-CONF		
T-CONN.Rsp		p01 [01] [06] ACTIVE ^p01 [07] IDLE	
FIRST	[01] [02] LISTEN		
CNTL			[01] [03] ACTIVE
ERROR(*e-code*=1)			[04] IDLE

III.5.2 - 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. Isso corresponde mais ou menos à idéia de clientes/servidores da rede Internet. Naquele ambiente, servidores de FTP e TELNET, por exemplo, oferecem seus serviços em "portas" com endereços fixos.

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 (maiores detalhes no Capítulo IV), 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 III.8. 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*. No caso específico do controle de fluxo por taxa, ao ser pedido o parâmetro **taxa** deve ser informado o valor máximo permitido. Este valor é calculado *a priori* e depende do ambiente de implementação.

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.

III.6 - Fechamento de conexão

III.6.1 - 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 um 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.

A Tabela III.5 mostra a tabela de estados para a liberação de uma conexão ponto-a-ponto.

Estados:

- Close: lado iniciador da desconexão espera resposta da entidade remota.
- Close-Conf: lado respondedor da desconexão espera resposta do usuário.

Predicados:

- p02: desconexão normal.
- p03: aborto.

Ações:

- [08]: enviar T-DISCONNECT.Confirm.

- [09]: enviar CNTL(WCLOSE, RCLOSE, DREQ).
- [10]: enviar CNTL(WCLOSE, RCLOSE, END).
- [11]: enviar CNTL(END).

Tabela III.5: Liberação de conexão ponto-a-ponto.

ESTADOS EVENTOS	ACTIVE	CLOSE-CONF	CLOSE
T-DISCONN.Req	p02 [09] CLOSE p03 [11] IDLE		
T-DISCONN.Rsp		[10] IDLE	
CNTL(WCLOSE, RCLOSE, DREQ)	[04] CLOSE-CONF		
CNTL(END)	[04] IDLE		
CNTL(WCLOSE, RCLOSE, END)			[08] IDLE

III.6.2 - 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 abandono é enviada ao servidor ou a qualquer membro do grupo.

III.7 - 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.

III.7.1 - 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.

III.7.2 - 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*, *rseq* e *dseq*, descritas na Seção III.2. 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.

Segue abaixo o algoritmo de verificação do número de seqüência de PDUs DATA recebidas, em pseudo-código, sabendo-se que *seq* é equivalente ao valor do campo correspondente na PDU DATA:

```
se seq < recv_sync e seq < rseq
        descartar PDU (trata-se de uma duplicata);
senão
        recv_sync recebe seq;
        se o controle de erros está habilitado
                se seq = rseq e a PDU não está corrompida
                        incrementar rseq;
                fim-se
        senão
                se seq = rseq
```

```
                    incrementar rseq;
                fim-se
            fim-se
            se seq ≥ hseq
                incrementar hseq;
            fim-se
            Processar DATA;
fim-se.
```

III.8 - Gerenciamento da transmissão em um grupo através da concessão de fichas

Como já foi mencionado na Seção III.5.2, 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.

As Tabelas III.6 e III.7 representam as tabelas de estado do gerenciamento de fichas para um produtor e um servidor, respectivamente.

Estados:

- Active: em conexões multidestinatárias, uma entidade estará em Active com ou sem a ficha de transmissão. Ela poderá sempre receber dados, mas somente poderá transmiti-los se a variável *Token_Granted* for verdadeira.
- Wait-Token: estado de espera pela ficha; dados podem ser recebidos.

Predicados:

- p04: *Token_Granted* = FALSE.
- p05: endereço não consta da fila de pedidos.
- p06: primeiro endereço da fila de pedidos = endereço local.
- p07: fila de pedidos não cheia.
- p08: fila de pedidos não vazia.
- p09: número de fichas no servidor > 0.
- p10: número de fichas no servidor < máximo.
- p11: contador de tentativas > 0.
- p12: endereço associado a TTIMER = endereço local.

Ações:

- [12]: atribuir FALSE a *Token_Granted*.
- [13]: atribuir TRUE a *Token_Granted*.
- [14]: colocar endereço na fila de pedidos.
- [15]: disparar temporizador TTIMER.
- [16]: interromper temporizador TTIMER.

- [17]: disparar temporizador WTIMER.
- [18]: atribuir 3 ao contador de tentativas.
- [19]: decrementar contador de tentativas.
- [20]: interromper temporizador WTIMER.
- [21]: interromper temporizador associado à ficha recebida (pode ser TTIMER ou WTIMER).
- [22]: incrementar número de fichas no servidor.
- [23]: decrementar número de fichas no servidor.
- [24]: enviar T-TOKEN.Indication.
- [25]: enviar CNTL(`RETTK`).
- [26]: enviar PASS.
- [27]: enviar TOKEN.
- [28]: enviar ERROR(*e-code* = 7).
- [29]: enviar ERROR(*e-code* = 9).
- [30]: enviar ERROR(*e-code* = 10).

Tabela III.6: Gerenciamento de fichas no produtor.

ESTADOS EVENTOS	ACTIVE	WAIT-TOKEN
T-TOKEN.Req	[26] WAIT-TOKEN	
T_TOKEN.Rsp	[27] [12]	
CNTL(`RETTK`)	p04 [30] [24] ^p04 [27] [24] [12]	[29] [24] ACTIVE
TOKEN		[24] [13] ACTIVE
ERROR(*e-code* = 7)		[24] ACTIVE

Tabela III.7: Gerenciamento de fichas no servidor.

ESTADOS EVENTOS	ACTIVE	WAIT-TOKEN
T-TOKEN.Req	p05 p07 [14] WAIT-TOKEN p09 p06 [24] [13] ACTIVE ^p06 [27] [15] [23] ^p07 [24]	
T-TOKEN-Rsp	p10 [22] [16] [12] p08 p06 [24] [13] ^p06 [27] [15] [23]	
TOKEN	p10 [22] [21] p08 [27] [15] [23]	p10 [22] [21] p08 p06 [24] [13] ^p06 [27] [15] [23]
PASS	p05 p07 [14] p09 [27] [15] [23] ^p07 [28]	
ERROR(*e-code* = 9)	[20] [22]	
ERROR(*e-code* =10)	[20] [22]	
Estouro TTIMER	p10 p12 [24] [22] [12] ^p12 [25] [17] [18]	
Estouro WTIMER	p11 p10 [25] [17] [19] ^p11 [22]	

III.9 - Controle de fluxo

III.9.1 - 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 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 a 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 III.7 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.9.2 - 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

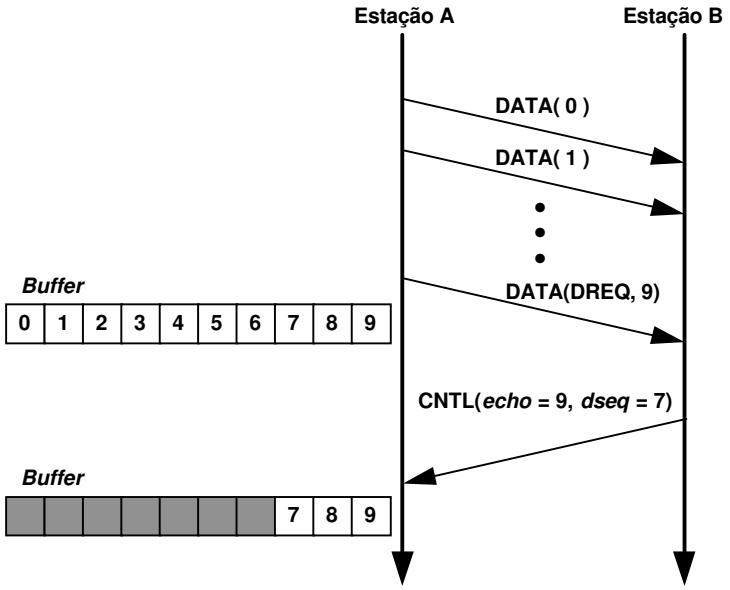

Figura III.7: Exemplo de controle de fluxo por janela de transmissão.

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 restituído 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.

III.10 - Controle de erros

III.10.1 - *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.

III.10.2 - 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 receptor, 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.10.3 - 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. Seu funcionamento já foi visto em detalhes na Seção II.5.1. 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.