Suporte à Comunicações Multidestinatárias para Aplicações ATM Nativas

Marcelo Dias Nunes
nunes@ravel.ufrj.br

Luís Felipe Magalhães de Moraes
moraes@ravel.ufrj.br

RAVEL - Laboratório de Redes de Alta Velocidade
COPPE - Coordenação dos Programas de Pós-Graduação em Engenharia
Universidade Federal do Rio de Janeiro
Caixa POstal 68504 - 21945-970 - RJ - Brasil
Tel./Fax: 55 21 590-2552

Resumo

A partir da observação de que comunicações multiponto-a-multiponto são vitais para as mais diversas aplicações, e de que redes ATM oferecem uma plataforma de alto desempenho para o desenvolvimento de tais aplicações, este artigo propõe dois protocolos que possam ser implementados diretamente sobre ATM e ofereçam suporte para aplicações multidestinatárias nativas. Por mais complexa que possa parecer a tarefa de implementar difusão de dados sobre um meio naturalmente orientado a conexão, o processamento de ambos os protocolos é simplificado e concebido para interagir com sinalização UNI 3.1 e 4.0, embutida nos equipmentos dos principais fabricantes. São examinados ainda mecanismos de controle de fluxo e erros, em uma estrutura topológica hierárquica, que possibilitem entrega confiável de dados a todos os membros de um grupo multiponto.

Abstract

As multicast Communications are vital for a wide range of applications and ATM provides a high performance platform for the deployment of such applications, this paper proposes two native ATM protocols specifically tailored for multicast support. As complex as the task of implementing broadcast over a connection oriented médium may seem, both protocols' processing are kept simple, as they were conceived for interaction with UNI 3.1 and 4.0 signalling, which are built in all ATM hardware. Flow and error control mechanisms are also examined, in a hierarchical topology that allows reliable data delivery to all group members.

1 Introdução

O caráter orientado a conexão do Modo de Transferência Assíncrono (ATM) complica o estabelecimento de conexões multiponto-a-multiponto. A alternativa está em estabelecer tantas conexões ponto-a-ponto quantas forem necessárias. A Emulação de Redes Locais *(LAN Emulation)* [Newman94] é um exemplo deste mecanismo: o BUS *(Broadcast and UnknownServer)* mantém uma conexão unidestinatária com cada cliente LANE.

Tanto LANE quanto IP Sobre ATM [Juha93] são desvantajosos no sentido em que ambos são implementados como camadas de software (figura 1), o que introduz uma sobrecarga de processamento que restringe o aproveitamento da banda passante pela aplicação. Esta limitação levou à presente proposta, construída sobre as seguintes premissas:

- projetar um protocolo que ofereça uma interface eficiente para aplicações ATM nativas;

No SIMULT, todas as Fontes conectam-se ao Servidor de Fichas, que por sua vez encaminha os dados para todos os membros do grupo. Apesar desta concentração de tráfego parecer uma séria limitação de desempenho a primeira vista, a centralização no Servidor de Fichas permite que os recursos sejam distribuídos com maior igualdade entre as Fontes. Além disso, devemos lembrar que o BUS da Emulação de Redes Locais segue a mesma filosofia, em um produto amplamente difundido e absorvido pelo mercado.

O MULT^2 descentraliza a transferência de dados às custas de um gerenciamento de conexões mais complexo: cada Fonte estabelece sua própria árvore de distribuição ponto-a-multiponto. Isto elimina a concentração de tráfego em uma única interface mas, por outro lado, elimina também o controle de devolução das fichas. O Servidor de Fichas confia apenas em um temporizador máximo acima do qual a ficha é julgada perdida devido a falha na Fonte.

Vimos ainda como incorporar características de confiabilidade aos protocolos através da implementação de mecanismos simples de controle de fluxo e erros em uma topologia hierárquica que distribui a responsabilidade pelas retransmissões entre determinadoshosts do grupo. Dessa forma, a Fonte dos dados (ou o Servidor de Fichas em SIMULT) não mais deverá esperar pelo retorno dos reconhecimentos de todos os receptores, minimizando o tráfego desse tipo de pacote na rede e aliviando o processamento do transmissor.

Ambos os protocolos estão em fase de implementação no Laboratório de Redes de Alta Velocidade (RAVEL) da UFRJ. Este laboratório está conectado a uma rede de 155 Mbps que engloba as principais instituições de ensino e pesquisa do Rio de Janeiro, a Rede Rio 2, promovendo um ambiente de teste único. A seguir, pretendemos desenvolver modelos analíticos de ambos os protocolos *e* comparar o resultado das simulações destes modelos com os resultados práticos observados a partir da implementação.

Referências

[Newman94] P. Newman, "ATM Local Area Networks", IEEE Communications Magazine, vol. 32, pp. 86-98, Março de 1994.

[Juha93] Juha Heinanen, "Multiprotocol Encapsulation over ATM Adaptation Layer 5", IETF Request for Comments 1483, Julho de 1993.

[Gauthier97] B. Gauthier, J.-Y. Le Boudec e P. Oechslin, "SMART: a Many-to-Many Multicast Protocol for ATM", IEEE Joumal on Selected Áreas in Communications, vol. 15, pp. 458-472, Março de 1997.

[Stadler97] R. Stadler, M. Borla e C. Aurrecoechea, "Mcast: Design of the Service and its Management", Relatório Técnico, Universidade de Columbia, Dezembro de 1997.

[Cidon96] I. Cidon et al, "OPENET: na Open and Efficient Control Platform for ATM Networks", Relatório Técnico, Sun Microsystems Lab, Janeiro de 1996.

[Lazar96] A. A. Lazar, K. S. Lim e F. Marconcini, "Realizing a Foundation for Programmabiliyof ATM Networks with the Binding Architecture", IEEE Journal on Selected Áreas in Communications, vol. 14, n° 7, pp. 1214-1247, Setembro

Na figura 5, os hosts C e D (além da Fonte propriamente dita) são escolhidos como líderes de seus subgrupos e, portanto, devem armazenar os dados para eventuais retransmissões. Os hosts A, B, C e D enviam seus ACK's diretamente para a Fonte. Os hosts E, F e G enviam ACK's para o host C, enquanto H e I os enviam para D. Somente após receberem os reconhecimentos positivos de seus filhos, Fonte, C e D liberam os dados armazenados.

5.1 Controles de Fluxo e Erros

Um dos objetivos do mecanismo de recuperação local, ou seja, subdividir o grupo em "células" de retransmissão, é minimizar o tráfego de reconhecimentos na rede. Portanto, mecanismos de controle de fluxo e erros devem ser utilizados tendo em mente este objetivo principal.

Minuciosos estudos e comparações entre diversos mecanismos de *gerenciamento* de conexões, controle de reconhecimentos, fluxo e erros, já foram exaustivamente conduzidos na literatura visando otimizar ao máximo o processamento de vários protocolos [Doer90]. Por outro lado, nosso objetivo ao apresentar os protocolos SIMULT e MULT^2 é, fundamentalmente, especificar procedimentos simples que poderiam ser incorporados a protocolos de sinalização ATM. Por esta razão, optamos por um simples mecanismo de controle de fluxo por janelas, onde cada janela é composta de um certo número de pacotes de tamanho fixo. Particularmente no protocolo SIMULT, a janela corresponde ao quadro formado pela concatenação de *N* pacotes correspondentes às *N* Fontes simultâneas. O reconhecimento vale para toda a janela de pacotes.

O funcionamento integrado dos mecanismos de controle de fluxo e erros é simples. Em ambos os protocolos, introduziremos uma nova PDU, o ACK. Após a transmissão de uma janela (ou quadro), a Fonte ou líder de subgrupo dispara um temporizador e aguarda os ACK's de seus filhos. Se pelo menos um dos ACK's não for recebido neste intervalo de tempo (por não ter chegado ao seu destino ou por ter chegado com erros), toda a janela é retransmitida no subgrupo. Este processo é repetido tantas vezes quanto necessário até que todos os filhos de um determinado transmissor tenham recebido os dados corretamente.

Evidentemente, este não é um processamento otimizado. Para reduzir ainda mais o tráfego de reconhecimentos na rede, poderia ser implementado algum mecanismo de retransmissão seletiva com supressão de reconhecimentos (*NACK avoidance*) [Floyd95, Levine98a]. Contudo, voltamos a frisar que tais técnicas exigem implementação cuidadosa e têm procesamento complexo, o que foge ao escopo deste projeto.

6 Considerações Finais

Como pode-se observar das máquinas de estado apresentadas nas seções 3 e 4, os protocolos SIMULT e MULT^2 são simples e interagem diretamente com a sinalização UNI. A sinalização PNNI é transparente para os protocolos: é necessário estabelecer uma árvore de distribuição ponto-a-multiponto, independente do número de comutadores envolvidos ou do número de circuitos virtuais exigidos. Implícita no funcionamento dos protocolos é a sinalização para estabelecer e terminar os circuitos virtuais. Esta interação possibilita que ambos os protocolos sejam implementados em topologias arbitrárias e hardware de qualquer fabricante.

seus requisitos de armazenamento. O HIP promove também interoperabilidade entre protocolos de roteamento multiponto de baixo nível. Uma única árvore de distribuição multiponto para todos os receptores é montada, mas informações de controle são confinadas a domínios ou níveis particulares.

Apesar de os protocolos mencionados não terem sido originalmente concebidos para uso em uma rede ATM, sua topologia pode ser facilmente estruturada hierarquicamente, como mostrado na figura 5. Isto encoraja o uso de estruturas em árvore para os protocolos SIMULT e MULT^2. Portanto, nós adotaremos uma variação do protocolo em árvore sem supressão de NACK's descrito em [Levine98a].

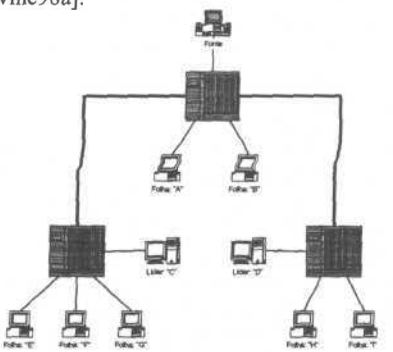

Figura 5: Arquitetura em Árvore.

O grupo é organizado em subconjuntos hierárquicos que formam uma árvore *enraizada* na fonte de dados. Cada nó intermediário deve enviar ACK's para seu líder (nível lógico superior), armazenar os dados em *buffers* para eventuais retransmissões e esperar pelos ACK's de seus "filhos" (nível lógico inferior). Folhas (o nível mais baixo) enviam ACK's apenas para seus líderes.

A figura 5 mostra uma configuração típica de uma rede ATM, onde três comutadores formam um *backbone*. Topologicamente, os comutadores seriam líderes naturais. Entretanto, isto seria tecnicamente impraticável e leva a uma escolha estática dos nós intermediários. Do ponto de vista operacional, Fontes, Destinos ou o Servidor de Fichas podem ser líderes. Considerando N hosts no grupo:

- a árvore de distribuição conecta um host (Fonte no MULT^2 ou o Servidor de Fichas no SIMULT) a todos os outros N-2 hosts;

- a conexão multiponto atravessa M comutadores.

Para cada comutador, um host será escolhido arbitrariamente como líder. O subgrupo compreende todos os hosts (que pertencem ao grupo) conectados ao comutador. Nesta etapa do projeto, os hosts serão configurados estaticamente com o endereço do líder, de tal forma que os reconhecimentos sejam enviados ao destino correto. Portanto, o número de comutadores na rede ditará o número de subgrupos de retransmissão.

```
        break;

        case "fim de transmissão": /* definido pela aplicação -/
        if (estado == Tx) (
                interrompe T1;
                termina svc p2mp;
                estabelece SVC p2p com ST;
                envia RET;
                estado = Rx;
        }
        break;

        case REG:
        if (estado == Tx) {
                termina SVC p2p;
                adiciona host ao svc p2mp;
        }
        break;

}
```

Rotina de tratamento de interrupção (assíncrona):

```
Timeout T1:
if (estado == Tx) {
        interrompe transmissão;
        envia RET;
        termina SVC p2mp;
        estado = Rx;
}
```

5 Incorporando confiabilidade aos protocolos SIMULT e MULT²

Até agora, consideramos apenas transferência de dados livre de erros. Nesta seção, serão propostas extensões para proporcionar comunicação confiável na presença de uma probabilidade não nula de perda de pacotes. Esta nova característica será incorporada na forma de mecanismos de controle de fluxo e erros nos protocolos SIMULT e MULT².

A confiabilidade em comunicações multidestinatárias está diretamente relacionada ao problema de implosão de reconhecimentos: à medida que o número de receptores em um grupo aumenta, o transmissor e a rede como um todo podem ficar sobrecarregados com reconhecimentos negativos (NACK's) solicitando retransmissões ou reconhecimentos positivos (ACK's) de dados corretamente recebidos.

Na literatura podem ser encontrados exemplos *de* modelos analíticos demonstrando que a organização do conjunto de receptores em uma estrutura hierárquica é o método mais escalável. Em [Levine98a], os autores classificam e comparam cinco protocolos ponto-a-multiponto confiáveis: iniciados pelo transmissor (por exemplo, o protocolo XTP [XTP92]), iniciados pelo receptor, iniciados pelo receptor com supressão de NACK's (por exemplo, o SRM [Floyd95]), protocolos em árvore (por exemplo, o RMTP [Paul97]) e protocolos em anel. Os autores provam matematicamente que a vazão dos protocolos em árvore é independente do número de participantes do grupo.

Protocolos em árvore utilizam a técnica de recuperação local [Nonn98]: um ou mais hosts, além da fonte dos dados, são designados para agir como retransmissores locais. Um exemplo da utilização de recuperação local pode ser encontrado em [Levine98b]. Nesse artigo é introduzido um protocolo que organiza os receptores deterministicamente em uma estrutura lógica hierárquica, onde cada receptor é capaz de traçar seu caminho de volta à fonte e usar esta informação para determinar a melhor maneira de obter recuperação local e um controle de congestionamento eficiente. Outro exemplo é o Protocolo HIP [Shields98]. Este protocolo prove roteamento multiponto hierárquico de alto nível enquanto mantém baixos o tráfego na rede e

Observação: as PDU's TKN e REJ devem ser capazes de transportar a lista de afiliação. Isto impõe uma limitação prática que restringirá o tamanho máximo do grupo.

4.2 Fontes e Destinos

Além dos estados definidos para as Fontes e Destinos no protocolo SIMULT, o protocolo MULT[2] adiciona o estado WFCNF (*wait for confirmation*). Todo host candidato a Fonte deve primeiro registrar-se como Destino e usar o temporizador T1 (mesma função do temporizador homônimo usado pelo SIMULT) para limitar localmente o tempo de permanência com a ficha. Segue abaixo a máquina de estados:

```
switch (evento) l

        case "pedido da aplicação para entrar em um grupo":
        if (estado == IDLE) {
                estabelece SVC p2p com ST;
                envia REG;
                estado = WFCNF;
        }
        break;

        case CNF:
        if (estado == WFCNF) (
                termina SVC p2p;
                if (lista de membros está não vazia)
                        Cor (i=1;i<=num_fontes na lista de membros;i++) f
                                estabelece SVC p2p com host;
                                envia REG;
                        }
                estado = Rx;
        }
        break;

        case "pedido da aplicação para sair do grupo":
        if (estado == Rx || Tx) i
                if (estado== Tx) {
                        interrompe transmissão e T1;
                        termina SVC p2mp;
                }
                for (i=1;i<=num_fontes;i++)
                /* as Fontes das quais o host está recebendo dados */
                estabelece SVC p2p com Fonte;
                envia DPY;
                estabelece SVC p2p com ST;
                envia EXIT;
                estado = IDLE;
        )
        break;

        case "dados recebidos da aplicação":
        if (estado == Rx) f
                estabelece SVC p2pcom ST;
                envia REQ;
                estado = WFTK;
        }
        break;

        case TKN:
        if (estado == WFTK || Rx) {
                termina SVC p2p;
                estabelece SVC p2mp com todos os hosts da lista de membros;
                dispara T1;
                incia transmissão; /* envia um certo número de PDU's DT */
                estado = Tx;
        }
        break;

        case REJ:
        if (estado== WFTK) f
                termina SVC p2p;
                for (i=1;i<=num_fontes na lista de membros;i++) (
                        estabelece SVC p2p com host;
                        envia REG;
                }
                notifica aplicação;
                estado = Rx;
}
```

- não existe mais a PDU NOW, uma vez que os dados não precisam mais passar pelo Servidor de Fichas, ou seja, não há mais como controlar o tempo máximo de permanência de uma ficha em uma Fonte;

- uma nova PDU, EXIT, indica a ST que um host saiu do grupo (importante para que ST mantenha a afiliação do grupo atualizada);

- CNF é também uma nova PDU, usada por ST para confirmar a adesão ao grupo;

- algumas PDUs, conforme veremos na máquina de estados, levarão consigo uma lista de membros, que pode ser simplesmente uma cópia da tabela de afíliação de ST.

4.1 Servidor de Fichas

A estrutura do Servidor de Fichas no protocolo MULT^2 é bem mais simples que no SIMULT. Não existem filas de entrada ou *buffer* de dados, uma vez que não são mantidas conexões ponto-a-ponto com ST para transmissão de dados da aplicação. No MULT^2, confia-se na temporização realizada localmente nas Fontes. Mesmo assim, ST mantém um temporizador (7), iniciado assim que a ficha é enviada e com valor maior que o do temporizador da Fonte. Esta temporização é importante para regenerar uma ficha considerada perdida por filha na Fonte. Portanto, temos a seguinte máquina de estados para ST:

```
switch [evento) i

        case REG:
        insere host na tabela;  /* tipo = Destino */
        envia CNF com lista de membros;
        break;

        case EXIT:
        termina SVC p2p;
        remove host da tabela;
        break;

        case REQ:
        procura host na tabela;
        if (token_counter> 0) {
                envia TKN com lista de membros;
                token_counter --;
                tipo = Fonte;
        }
        else f
                envia REJ com lista de membros;
                insere host na fila de espera;
        }
        break;

        case RET:
        termina SVC p2p;
        tipo = Destino;
        if (fila de espera está não vazia) {
                estabelece svc p2p com host;
                envia TKN com lista de membros;
                tipo = Fonte;
                remove host da fila de espera;
        i
        else
                token_counter ++;
        break;

}
```

Rotina de tratamento de interrupção (assíncrona):

```
Timeout T:
token_counter ++;
tipo = Destino;
```

```
        token_counter++;
        tipo = Destino;
```

3.2 Fontes e Destinos

Todo host deve, primeiramente, registrar-se no grupo como Destino. A qualquer momento, Fontes e Destinos podem estar em um dos estados seguintes: *IDLE*, o estado inicial, *WFTK*, estado de espera pela ficha, *Tx*, estado de transmissão de dados (Fonte ativa) e *Rx*, estado de recepção de dados (Destino). Abaixo, a máquina de estados para ambos:

```
switch (evento) {

        case "pedido da aplicação para entrar em um grupo":
        if (estado == IDLE) (
                estabelece SVC p2p com ST;
                envia REG;
                estado = Rx;
        }
        break;

        case "pedido da aplicação para sair do grupo":
        estabelece SVC p2p com ST;
        if (estado == Tx) f
                interrompe transmissão e T1;
                envia DPY;
                estado = IDLE;
        }
        else if (estado == Rx) !
                envia DPY;
                estado = IDLE;
        }
        break;

        case "dados recebidos da aplicação":
        if (estado == Rx) (
                estabelece SVC p2p com ST;
                envia REQ;
                estado = WFTK;
        }
        break;

        case TKN:
        if (estado == WFTK || Rx) {
                inicia transmissão; /* envia um certo número de PDU's DT */
                estado = Tx;
        i
        break;

        case REJ:
        if (estado == WFTK) {
                notifica aplicação;
                estado = Rx;
        }
        break;

        case "fim de transmissão": /* definido pela aplicação */
        if (estado == Tx) {
                sends RET;
                estado = Rx;
        )
        break;

        case NOW:
        interrompe transmissão;
        envia RET;
        estado = Rx;
        break;

}
```

4 Protocolo *MultipleMulticast Connections* (MULT^2)

Ao protocolo MULT^2 aplicam-se as mesmas PDU's utilizadas pelo protocolo SIMULT, com algumas diferenças:

```
                switch (PDU) {

                        case REG:
                        termina SVC p2p; /* ponta-a-ponto */
                        dealoca fila de entrada;
                        num_p2p --;
                        insere host na tabela; /* tipo = Destino */
                        if (num_entradas_tabela == 1)
                                estabelece SVC p2mp /* ponto-a-multiponto */ com host;
                        else
                                adiciona host ao SVC p2mp;
                        break;

                        case DPY:
                        remove host do SVC p2mp;
                        remove host da tabela;
                        if [tipo == Fonte) {
                                interrompe timer; /* T1 ou T2 */
                                token_counter ++;
                        }
                        termina SVC p2p;
                        dealoca fila de entrada;
                        num_p2p --;
                        break;

                        case REQ:
                        procura host na tabela;
                        if (token_counter> 0) {
                                envia TKN;
                                token_counter --;
                                dispara T1;
                                tipo = Fonte;
                        }
                        else {
                                envia REJ;
                                insere host na fila de espera;
                        }
                        break;

                        case RET:
                        interrompe timer; /* T1 or T2 */
                        termina SVC p2p;
                        dealoca fila de entrada;
                        num_p2p --;
                        tipo = Destino;
                        if (fila de espera está não vazia) {
                                envia TKN; /* pelo SVC previamente estabelecido quando do
                                              pedido da ficha */
                                dispara T1;
                                tipo = Fonte;
                                remove host da fila de espera;
                        }
                        else
                                token_counter ++;
                        break;

                        case DT:
                        copia a PDU da fila de entrada para o buffer;
                        remove PDU da fila de entrada;
                        break;

                }

        )

        ponteiro_p2p ++; /* indice da lista de filas de entrada, para apontar para a próxima
                            (contador circular) */
        if (buffer não vazio && ponteiro_p2p == 0) /* o segundo termo do if indica que todas
                                                      as filas já foram varridas */
                transmite quadro para SVC p2mp;

) while (num_p2p > 0);

}
```

Rotinas de tratamento de interrupção (assíncronas):

```
Timeout T1:
envia NOW;
dispara T2;

Timeout T2:
termina SVC p2p;
dealoca fila de entrada;
num_p2p --;
```

3.1 Servidor de Fichas

O Servidor de Fichas mantém, internamente, várias estruturas de dados, a saber:

- um contador de fichas (*token_counter*);
- uma fila FCFS (*first come, first serve*) que armazena os endereços dos candidatos a Fonte;
- um número de filas de entrada correspondente ao número máximo de fichas, ou seja, ao número máximo de Fontes simultâneas (a fila é alocada assim que ST recebe um pedido de estabelecimento de conexão ponto-a-ponto);
- uma variável que registra o número de conexões ponto-a-ponto estabelecidas (*num_p2p*);
- um *buffer* com capacidade suficiente para armazenar um quadro resultante da concatenação dos pacotes (supostos de tamanho fixo) recebidos por cada uma das fontes;
- uma tabela de configuração do grupo, onde cada entrada é composta por um índice, o endereço do host e seu tipo (Fonte ou Destino).

No Servidor de Fichas é implementado um escalonador que verifica uma fila de entrada após a outra, sucessivamente; se a fila estiver não vazia, o pacote é copiado para o *buffer* (ou não, de acordo com o tipo de PDU) e removido da fila. Quando a varredura estiver completa (todas as filas já foram percorridas), o quadro contido no *buffer* é transmitido para todos os Destinos (figura 4).

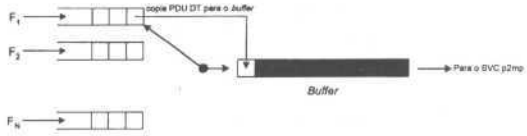

Figura 4: Escalonador do Servidor de Fichas.

São também definidos dois temporizadores (*timers*):

- T1, que marca o tempo máximo com o qual a Fonte pode permanecer com uma ficha (disparado logo após o envio da ficha);
- T2, tempo adicional máximo, após o estouro de T1, durante o qual a Fonte deve devolver a ficha.

Segue abaixo a máquina de estados para ST. O evento de atendimento aos pedidos de estabelecimento de conexão ponto-a-ponto é assíncrono, isto é, o Servidor de Fichas interrompe o processamento da máquina de estados, atende à solicitação e incrementa *num_p2p*.

```
if (num_p2p > 0) {
      do {
            if (fila de entrada não vazia) {
```

No protocolo MULT^2, a responsabilidade pela transferência de dados para o grupo é distribuída pelas Fontes. Assim, haverá tantas conexões multiponto quantas forem as Fontes simultâneas. Na figura 3, podemos ver que o Servidor de Fichas ainda está presente, enquanto única entidade que possui uma visão global do grupo. Destinos devem conectar-se a ST para se registrarem e para comunicarem seu abandono, de modo a permitir que ST mantenha atualizada sua tabela de afiliação. Cada novo Destino recebe, como confirmação de sua adesão, uma lista com todas as Fontes ativas. O Destino deve solicitar a cada Fonte que o adicione à sua conexão multiponto. Este é um requisito do UNI 3.1, onde a origem (raiz) da conexão ponto-a-multiponto é a única entidade capaz de adicionar novos receptores. Apesar de, no atual estágio de desenvolvimento, o protocolo MULT^2 não contar com esta funcionalidade, sua máquina de estados pode ser facilmente modificada para suportar adesão iniciada pelo receptor (*leajinitiatedjoin)*, introduzida pelo padrão UNI 4.0 [ATM96]. Entretanto, já na versão 3.1, o processo de saída da árvore de distribuição pode ser iniciado pelos receptores. Similarmente, quando solicita uma ficha, a nova Fonte recebe também uma lista com todos os membros. A Fonte procederá analogamente aos Destinos com relação às demais Fontes da lista e estabelecerá uma conexão ponto-a-multiponto com todos os membros.

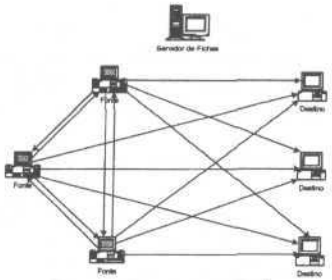

Figura 3: Protocolo MULT^2.

As próximas seções apresentam especificações informais para ambos os protocolos, detalhando suas máquinas de estado.

3 Protocolo *Single Multicast Connection* (SIMULT)

As PDU's (*protocoldata units*) seguintes são utilizadas pelo protocolo SIMULT:

REG	registro de host
DPY	chamada da rotina *drop party* do UNI 3.1
REQ	pedido de ficha
TKN	ficha
REJ	pedido de ficha rejeitado (não há fichas)
RET	devolução da ficha
NOW	devolução imediata da ficha
DT	Dados

Os formatos das PDU's não são definidos no presente artigo.

Unindo as idéias dos exemplos acima com o trabalho apresentado em [Nunes96],este artigo oferece duas propostas de protocolos multidestinatários para redes ATM, o SIMULT (*Single Multicast Connection*) e o MULT^2 (*Multiple Multicast Connections*), com as seguintes características básicas:

- suporte total a comunicações multidestinatárias fazendo uso dos protocolos de sinalização implementados na maioria dos equipamentos ATM: UNI 3.1 [ATM94b] e PNNI Fase 0;
- interoperabilidade com o hardware de qualquer fabricante sem modificações;
- independência da topologia da rede, isto é, independência do número de comutadores que compõe a rede;
- especificação de procedimentos simples que possam ser eventualmente incorporados como extensões ao UNI, a exemplo do que ocorre com o OPENET.

Este artigo está organizado em 6 seções. A seção 2 introduz os protocolos de maneira superficial. A seção 3 detalha o protocolo SIMULT, enquanto a seção 4 o faz com o protocolo MULT^2. Na seção 5, extensões para garantia de confiabilidade são propostas. A seção 6 comenta o atual estágio de desenvolvimento dos protocolos e apresenta perspectivas futuras.

2 Duas abordagens para realização de conexões multiponto-a-multiponto

Ambos os protocolos apresentados neste artigo utilizam fichas para controlar o direito de transmissão simultânea, à semelhança dos padrões *Token-Bus*, *Token-Ring* ou FDDI. O número de fichas dita a quantidade de transmissores simultâneos. Qualquerhost que desejar difundir dados para outros deve primeiramente adquirir uma ficha junto a um Servidor de Fichas (ST), entidade presente em ambos os protocolos.

No protocolo SIMULT, o Servidor de Fichas concentra toda a informação pertinente ao grupo em uma tabela: quantos participantes e de que tipo são, isto é, Fontes ou Destinos. No SIMULT, ST é a raiz da árvore multidestinatária: sempre que um host desejar enviar dados, deve estabelecer uma conexão ponto-a-ponto com ST e pedir uma ficha. ST então adiciona a nova Fonte à conexão multiponto e concede uma ficha, se houver alguma disponível. Se não houver nenhuma, ST armazena o endereço do host solicitante em uma fila. A Fonte então envia seus dados ao Servidor de Fichas pela conexão ponto-a-ponto já estabelecida. O Servidor de Fichas funciona como um concentrador dos dados provenientes de todas as Fontes, e os redistribui para todos os Destinos (figura 2).

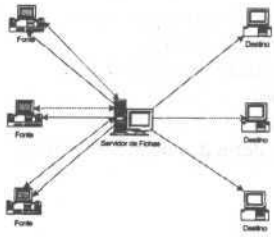

Figura 2: Protocolo SIMULT.

- prover suporte a comunicações multiponto-a-multiponto e, assim, viabilizar uma vasta gama de aplicações, que vão desde uma simples videoconferência até serviços detelemedicina.

Como pode-se ver na figurai, nosso perfil apresenta o mesmo número de camadas que o IPOA. A principal diferença é que nossos protocolos interagem diretamente com *o driver* de interface com o AAL5, eliminando assim quaisquer restrições impostas pelo uso dos protocolos TCP/IP.

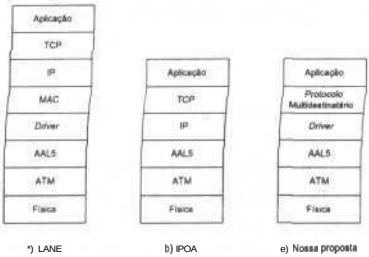

Figura 1: Comparação dos principais tipos de emulação sobre ATM.

Várias propostas para ATM multiponto podem ser encontradas na literatura, por exemplo, o SMART [Gauthier97], o Mcast [Stadler97] e o OPENET [Cidon96], que serão aqui brevemente revistas.

O SMART (*Shared Many-to-Many ATM Reservations*) é um protocolo multiponto com compartilhamento de recursos sob demanda. Isto significa que os recursos de uma conexão multidestinatária são compartilhados dinamicamente, sob demanda, entre todas as fontes potenciais. O protocolo utiliza n árvores de distribuição, uma para cada fonte, correspondentes a *n* VCC's multiponto, onde $n > 1$. Um host conectado a todos os VCCs envia dados para uma ou mais árvores, bastando para isso selecionar o(s) par(es) VPI/VCI adequado(s). Entretanto, as árvores de distribuição devem ser previamente estabelecidas, uma vez que o SMART não prove mecanismo algum de sinalização ou roteamento, o que eqüivale dizer que a afiliação do grupo não se modifica ao longo do tempo de vida dos VCC's. Além disso, o SMART requer suporte embutido nos comutadores ATM.

O Mcast é um serviço multiponto orientado a objeto. Diferentemente do SMART, o Mcast especifica os mecanismos necessários para a manutenção dinâmica de grupos. Este serviço prove apenas comunicações ponto-a-multiponto e foi projetado por uma equipe da Universidade de Columbia como base experimentos de gerenciamento sobre a plataforma Xbind [Lazar96].

O OPENET é uma plataforma de alto desempenho para redes ATM visando interoperabilidade e escalabilidade. Foi concebido na forma de extensões ao padrão PNNI [ATM94a] e possui mecanismos para o estabelecimento de conexões multiponto-a-multiponto, ainda que usadas apenas para o intercâmbio de informações de gerenciamento e controle de topologia. O OPENET implementa no comutador ATM uma entidade chamada *Control Point*, que intercepta pedidos recebidos pela UNI e os mapeia na representação interna de QoS.

de 1996.

[ATM94a] ATM Fórum, "ínterim Inter-Switch Signaling Protocol", Dezembro de 1994.

[Nunes96] M. D. Nunes e O. C. M. B. Duarte, "MCF: Um protocolo Multidestinatário Configurável com Gerenciamento de Acesso por Fichas de Permissão", XIV Simpósio Brasileiro de Redes de Computadores, Fortaleza, pp. 159-177, Maio de 1996.

[ATM94b] ATM Fórum, "ATM User-Network Interface Specification V3.1", 1994.

[ATM96] ATM Fórum, "ATM User-Network Interface Specification, V4.0", Julho de 1996.

[Levine98a] B. N. Levine e J. J. Garcia-Luna-Aceves, "A Comparison of Reliable Multicast Protocols", Multimedia Systems, vol. 6, n° 5, Agosto de 1998.

[XTP92] W. T. Strayer e A. Weaver, "XTP: the Xpress Transfer Protocol", Wesley PublishingCompany, 1992.

[Floyd95] S. Floyd et al, "A Reliable Multicast Framework for Light-Weight Sessions and Application Levei Framing", ACM Computer Communication Review, pp. 342-356, Agosto de 1995.

[Paul97] S. Paul et al, "Reliable Multicast Transport Protocol (RMTP)", IEEE Journal on Selected Áreas in Communications, vol. 15, n° 3, pp. 407-421, Abril de 1997.

[Nonn98] J. Nonnenmacher et al, "How Bad is Reliable Multicast without Local Recovery?", IEEE INFOCOM, Março de 1998.

[Lavine98b] B. N. Levine, S. Paul e J. J. Garcia-Luna-Aceves, "Organizing Multicast Receivers Determinisricallyby Packet-Loss Correlation", 6th. ACM International Multimedia Conference, Bristol, Setembro de 1998.

[Shields98] C. Shields e J. J. Garcia-Luna-Aceves, "The HIP Protocol for Hierarchical Multicast Routing", 7th. Annual ACM SIGACT-SIGOPS Symposiumon Principies of Distributed Computing, Puerto Vallarta, Junho de 1998.

[Doer90] W.A. Doeringer et al, "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.