DynaVideo – Um Serviço de Distribuição de Vídeo baseado em Configuração Dinâmica

Luiz Eduardo Cunha Leite
leduardo@natalnet.br

Guido Lemos de Souza Filho
guido@natalnet.br

Thais Vasconcelos Batista
thais@dimap.ufrn.br

Laboratório NatalNet – Departamento de Informática e Matemática Aplicada – DIMAp
Universidade Federal do Rio Grande do Norte – UFRN

RESUMO

A maioria das soluções apresentadas como candidatas à implementação do serviço de distribuição de fluxos contínuos de vídeo têm sido projetadas levando-se em consideração determinadas condições de infra-estrutura ou são moldadas para dar suporte a formatos de vídeo ou tipos de clientes específicos. Este artigo apresenta um serviço de distribuição de vídeo denominado DynaVideo (Dynamic Video Distribution Service). Este serviço foi projetado para distribuir vídeo de forma independentemente de formato e com suporte para interação com diferentes tipos de clientes. O serviço pode ser utilizado para distribuir vídeo em qualquer tipo de rede, entretanto um dos focos do projeto é a Internet. Uma das principais características do DynaVideo é a capacidade de configurar dinamicamente o serviço para atender a uma determinada demanda.

ABSTRACT

Most solutions proposed to implement audio and video distribution services have been designed considering specific infrastructures or have been tailored to specific application requirements, such as stream types or clients, which will be supported by the service. The performance of distributed services is becoming increasingly variable due to changing load patterns and user mobility. This paper presents the Dynamic Video Distribution Service – DynaVideo. This service was designed to distribute video in a way that is independent of the video format and to interact with different types of clients. The service may be used to distribute video over any digital network, however it is focused on the Internet. The main feature of the DynaVideo is the possibility of configuring dynamically the service to a specific demand.

Keywords: multimedia, DynaVideo, Video Server, Dynamic Configuration.

1 Introdução

Fatores como a diminuição do custo das redes banda larga e o desenvolvimento de microprocessadores mais potentes e mais baratos, aliados à consolidação de padrões para codificação de áudio e vídeo para aplicações como TV Digital (Digital TV – DTV), Vídeo sob demanda, Televisão de Alta Definição (High Definition Television – HDTV) e TV Interativa, tornam cada vez maior a necessidade de se estudar formas adequadas para transmissão de fluxos de vídeo e áudio através de redes digitais como a Internet.

Um fator de extrema relevância para serviços de distribuição de vídeo são os formatos de áudio e vídeo por ele suportado. É importante que o formato permita o maior grau de compressão possível, sem afetar a qualidade dos vídeos, facilitando a transmissão pela rede. O comitê MPEG (Moving Pictures Experts Group) foi criado em 1988 com o objetivo imediato de padronizar a codificação de áudio e vídeo para CD (Compact Discs). Através dos esforços da ISO (International Standards Organisation) e da IEC (International Electrotechnical Commission) foi publicado em 1992 um padrão para codificação de áudio e vídeo conhecido como MPEG-1 [3]. O esquema básico utilizado por este padrão é a predição temporal de cada quadro. Embora o MPEG-1 tivesse como alvo inicial as aplicações de armazenamento digital de mídia, ele é tido como um padrão genérico, no sentido de ser independente de uma aplicação em particular. Assim sendo, apenas a sintaxe de codificação e o esquema de decodificação são especificados.

O padrão MPEG-2 [4] consolidou-se em 1994. Sua aplicação inicial era a transmissão digital de canais de TV comum, sendo posteriormente incluído suporte para HDTV. O esquema de codificação de vídeo utilizado no MPEG-2 é também genérico e similar àquele utilizado no MPEG-1, entretanto com mais refinamentos e considerações especiais no que diz respeito às fontes interlaçadas. “Profiles” e “Levels”, descrevendo funcionalidades e resoluções respectivamente, foram introduzidos, de forma a permitir a compressão de vídeo considerando diferentes referenciais de qualidade e respectivas taxas de compressão.

O formato AVI (*Audio Video Interleave*) [5] foi definido pela Microsoft e é um caso especial do RIFF (*Resource Interchange File Format*). AVI é o formato mais comum para dados de áudio/vídeo no PC, permitindo a multiplexação de vídeo comprimido e áudio. Utiliza bibliotecas dinâmicas do sistema operacional Microsoft Windows em conjunto com *codecs* associados ao tipo de codificação, se for o caso.

QuickTime é um formato para criação e distribuição de vídeo, áudio e outros formatos [6]. Embora agora seja um padrão da ISO, ele foi inicialmente desenvolvido pela Apple para utilização nos computadores MacIntosh. Uma extensão ao formato QuickTime foi produzida pela Apple e denomina-se QuickTime VR. Este formato permite o controle interativo das animações QuickTime, que podem estar no formato de cenas navegáveis ou objetos manipuláveis. Estas animações podem conter links para outros objetos, tais como texto, arquivos de som ou documentos Web.

Existem vários formatos proprietários de áudio e vídeo. Alguns conseguem bons resultados, como o formato da Real Networks, mas devido à falta de documentação torna-se difícil usá-los em pesquisas, já que o intuito das mesmas geralmente é buscar padronizações para facilitar a distribuição e interoperabilidade. Como exemplos de formatos proprietários, podem ser citados os formatos da *Real Networks (RM), Microsoft (ASF), Vivo (VIV) e Vosaic (VDO).*

Atualmente no mercado, existem várias implementações de sistemas capazes de transmitir fluxos de áudio e vídeo na Internet. Dentre estes podem ser citados:

- Real System da Real Networks [7]. Este serviço consegue transmitir fluxos de áudio e vídeo nos formatos: MPEG-1, AVI, WAV, VIVO, AIF e RM, sendo este último um formato privado da empresa. Estes fluxos podem ser transmitidos utilizando-se: IP Multicast [16], TCP [19], UDP [18] ou HTTP [17].

- Windows Media Service [10] da Microsoft. Este serviço codifica no seu formato privado (ASF), os seguintes formatos de áudio e vídeo: BMP, WAV, WMA, WMV, ASF, AVI, e MPEG-1, podendo transmiti-los utilizando os seguintes protocolos: UDP, TCP HTTP / TCP e IP Multicast.
- VídeoCharger [8] da IBM. Transporta fluxos nos formatos: MPEG-1, MPEG-2, AVI, WAV, LBR e QuickTime, utilizando os protocolos: RTP [13], TCP, HTTP ou IP Multicast. Quando se está utilizando o sistema operacional AIX, o ReSerVation Protocol (RSVP) [21] e o Path MTU podem ser utilizados para melhorar o transporte do fluxo de áudio e vídeo em redes IP.
- VivoActive Producer da Vivo [11]. Fluxos no formato AVI são codificados e transmitidos no seu formato privado (VIV).

Este trabalho apresenta um serviço de distribuição de vídeo denominado DynaVideo (Dynamic Video Distribution Service). O DynaVideo foi concebido para distribuir vídeo de forma independente de formato para diferentes tipos de clientes em redes digitais. Uma ênfase especial foi dada à adequação do sistema para distribuição de vídeo na Internet. Aplicações que envolvem a distribuição de vídeo, notadamente TV Digital Interativa, tem por característica variações bruscas na demanda pelo serviço. O número, o tipo e a localização dos usuários pode variar muito em intervalos de tempo curtos, bastando para tal que "entre no ar" um programa de grande audiência. Sendo assim, outra característica marcante do DynaVideo é a possibilidade de configuração e reconfiguração dinâmica, característica não encontrada nos sistemas comerciais supracitados. No DynaVideo, a reconfiguração baseia-se no uso de servidores secundários que pretendemos implementar como agentes móveis, permitindo assim que sejam posicionados servidores em pontos estratégicos da topologia da rede, em tempo de execução, com o objetivo de atender, da melhor forma possível, uma determinada demanda. O uso de agentes móveis em sistemas baseados em reconfiguração dinâmica é também discutido em [27]. Nesse trabalho, os Servidores Secundários são denominados refletores e os agentes são utilizados para alterar a configuração, ou o código, dos servidores secundários. A abordagem, no entanto, é diferente da adotada no DynaVideo. Em nosso caso, o equivalente a modificação de código feita no trabalho proposto em [27] pelos agentes, é feita através do envio de um novo servidor secundário, ou seja, todo o novo código para um ponto específico da rede. A principal diferença reside no fato de que nossa motivação para uso dos agentes foi ajustar a configuração a flutuações na demanda e não corrigir e atualizar código, foco principal do trabalho apresentado em [27]. Outros trabalhos [28, 29] também adotam a idéia de replicar servidores, entretanto não fornecem suporte à reconfiguração do sistema durante a difusão ao vivo de vídeo.

Este artigo está estruturado da seguinte maneira. A seção 2 apresenta o serviço de distribuição dinâmica de vídeo. A seção 3 descreve a modelagem do servidor primário de difusão de vídeo. A seção 4 comenta o papel do servidor secundário de difusão de vídeo. A seção 5 discute a implementação do servidor de transmissão de áudio e vídeo usando os protocolos TCP, HTTP, UDP, IP Multicast e RTP. A seção 6 menciona os resultados obtidos nos experimentos de transmissão. Por fim, a seção 7 contém as conclusões deste trabalho.

2 DynaVideo – Dynamic Video Distribution Service

O DynaVideo é um serviço de distribuição de vídeo cuja principal característica é o dinamismo de sua configuração. A idéia básica que norteou a concepção do sistema foi a de que sua configuração deve ser capaz de se ajustar automaticamente à demanda pelo serviço. Isto é, o sistema deve continuamente buscar uma configuração ótima para atender uma dada demanda, onde a demanda é caracterizada principalmente pela quantidade e localização dos clientes do serviço de distribuição de vídeo. As aplicações alvo do sistema são o suporte a difusão do vídeo para aplicações de televisão e vídeo sob demanda. Uma característica inerente a tais aplicações é a variação da demanda (principalmente no caso de televisão) que pode passar de poucos clientes a milhões deles em um curto intervalo de tempo, bastando, para tal, que comece a ser transmitido um programa de grande interesse popular. Segue uma breve descrição dos módulos que compõem o DynaVideo, cuja arquitetura é exibida na exibida na Figura 1.

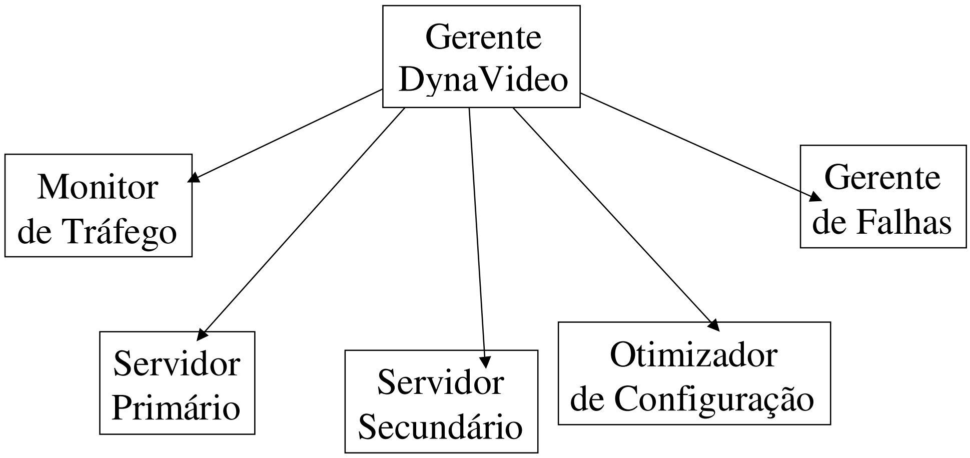

Figura 1 – Arquitetura do DynaVideo

O *Gerente DynaVideo* é o responsável pelo gerenciamento do serviço. À medida que clientes solicitam conexões ao serviço este módulo procura um servidor com capacidade disponível para atendê-los. Caso encontre um, o *Gerente DynaVideo* aloca o cliente para o servidor em questão. Caso contrário, é solicitada a iniciação de um servidor secundário para atender o cliente. Cabe ressaltar que, no primeiro momento, a preocupação é atender o cliente o mais rápido possível, e não buscar uma configuração otimizada para a distribuição do vídeo.

A entrada de um cliente no serviço caracteriza uma mudança na configuração da aplicação. Em resposta a esta modificação, o *Monitor de Tráfego* é ativado para rastrear a rota dos servidores ativos até o cliente recém chegado. Assim, o papel do *Monitor de Tráfego* é manter atualizada uma estrutura de dados (um grafo) que armazena as rotas dos servidores ativos para os clientes do serviço. O módulo *Monitor de Tráfego* está descrito detalhadamente em [1]. Na entrada de um cliente no serviço, o *Gerente DynaVideo* aloca um servidor para atendê-lo sem considerar aspectos de otimização já que seu objetivo é simplesmente providenciar, o mais rápido possível, um servidor com capacidade disponível para atender o cliente. Após finalizada uma atualização do grafo pelo monitor de tráfego, o *Gerente DynaVideo* ativa o *Otimizador de Configuração* para que este, usando técnicas de otimização, compute uma configuração ótima para o serviço. Assim, o *Otimizador de Configuração* é executado em *background*, buscando sempre melhores formas de configurar o serviço de distribuição de vídeo para uma dada demanda, exibida no grafo de rotas. Uma vez calculada uma configuração melhor que a atual, o *Otimizador de Configuração* solicita ao *Gerente DynaVideo* que mova os clientes de um servidor para outro, mova servidores secundário de um local para outro ou crie e exclua servidores secundários, sintonizando a configuração do serviço com o objetivo de otimizar o uso dos recursos de transmissão, processamento e armazenamento.

Os *Servidores Primários* caracterizam-se por acessar diretamente fontes de vídeo, as quais podem ser codificadores de vídeo operando em tempo real, servidores de arquivos onde estão armazenados os vídeos, etc.

Os *Servidores Secundários* diferenciam-se dos *Servidores Primários* por não acessar diretamente as fontes de vídeo, mas sim acessar um servidor primário. Pretende-se, posteriormente, incluir a capacidade de mobilidade aos servidores secundários, de maneira que, quando necessário, eles possam migrar para diferentes locais em uma rede.

Por fim, o *Gerente de Falhas (GF)* tem por objetivo identificar componentes em falha e providenciar sua substituição através da reconfiguração da aplicação. Por exemplo, caso um servidor entre em falha o *GF* detecta o evento e toma as providencias para que seus clientes passem a ser atendidos por outro servidor.

3 Modelagem do Servidor Primário de Difusão de Vídeo

O servidor primário de distribuição de fluxos de áudio e vídeo, aqui descrito, pode ser configurado de acordo com as necessidades do usuário, expressas através do formato do vídeo e do tipo do cliente utilizado. Para especificar este servidor, foi utilizada uma modelagem orientada a objetos, apresentada através da notação UML (*Unified Modeling Language*) [2].

O servidor de distribuição de fluxos de áudio/vídeo oferece mecanismos para: acessar os vídeos gerados por uma ou mais entidades como, por exemplo, um sistema de arquivos ou um codificador de vídeo operando em tempo real, de forma independente do formato do vídeo; transmiti-los, adaptando-os às necessidades dos clientes; e ainda fornecer informações e funções que possibilitem o gerenciamento do processo de recepção e transmissão.

O servidor está dividido em três partes: (i) *Interface com as Fontes de Dados*, responsável por acessar o fluxo de vídeo em entidades de armazenamento ou enviados por alguma entidade geradora, controlando o acesso e armazenando parâmetros relativos ao vídeo sendo transmidido, de acordo com as necessidades de cada aplicação específica; (ii) *Interface com os Clientes*, responsável por: permitir a entrada e a saída de clientes em tempo real, transmitir os fluxos aos clientes, adaptando-os às suas necessidades e medir e armazenar parâmetros relevantes a algum tipo de cliente em especial; e (iii) *Gerenciamento*, responsável por: permitir o gerenciamento do serviço por parte de um administrador; controlar a passagem dos fluxos de dados das fontes para os clientes; e controlar a entrada e saída de clientes.

A Figura 2 mostra a especificação do Servidor de Transmissão de Fluxos de áudio/vídeo em UML.

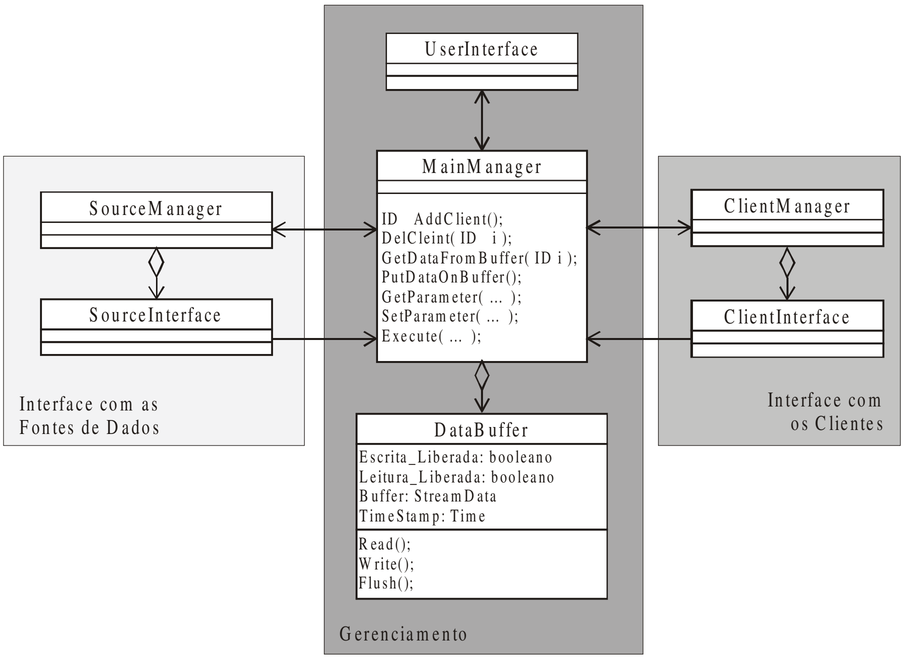

Figura 2 – Especificação do Serviço de Transmissão de Fluxos

3.1 Interface com as Fontes de Dados

A parte do serviço responsável pela interface com as fontes de dados possui duas classes: a *SourceManager* e a *SourceInterface*. A especificação dessas duas classes é dependente da entidade geradora do fluxo de vídeo. No caso de mais de uma fonte de vídeo, uma instância de cada uma destas classes é utilizada para cada fonte.

A classe *SourceInterface* é responsável por acessar segmentos de dados que constituem os fluxos de áudio/vídeo, estejam eles em arquivos ou sendo gerados em tempo real, e armazená-los em um buffer da uma instância da classe *DataBuffer*, através do método *PutDataOnBuffer()*, da classe *MainManager*. Dependendo dos parâmetros configurados nessa última classe, o método *PutDataOnBuffer()* pode ficar ou não bloqueado até que todas as entidades transmissoras tenham lido do buffer.

A classe *SourceManager* é responsável por colher parâmetros da fonte de dados, como por exemplo, a variação do retardo desde a fonte, caso esta seja um transmissor TCP, e informa-los à classe *MainManager*, através do método *SetParameter()*. A classe *SourceManager* pode ainda acessar métodos específicos da classe *SourceInterface.*

3.2 Gerenciamento

A classe *MainManager* é responsável pela troca de informações entre as classes; por controlar a inserção de dados no buffer de instâncias da classe *DataBuffer* e pela recuperação dos mesmos, através dos métodos *PutDataOnBuffer()* e *GetDadtaFromBuffer()*, respectivamente. É ainda responsável pelo armazenamento de parâmetros configurados pelo administrador do serviço ou informados por algum objeto, através do método *SetParameter()*; pela recuperação destes parâmetros através do método *GetParameter()*; pela interpretação e execução de comandos através do método *Execute()*; e pela inserção e remoção de novos clientes através dos métodos *AddClient()* e *DelClient()*.

A classe *DataBuffer* contém, além de um buffer para o armazenamento dos fluxos de dados, mecanismos para provisão de exclusão mútua entre os objetos que acessam o buffer, não permitindo que um objeto acesse um mesmo dado mais de uma vez. Esta classe contém também um *timestamp*, que indica o momento em que os dados foram inseridos no buffer, permitindo que os objetos que acessam esta classe decidam se irão ou não utilizar aqueles dados.

A interface entre o administrador do serviço e o objeto da classe *MainManager*, que torna possível a administração do sistema, é feita através da classe *UserInterface*.

3.3 Interface com os Clientes

Cada aplicação que será atendida pelo servidor de transmissão de fluxos deve estar associada uma classe do tipo *ClientManager* e outra do tipo *ClientInterface*. A primeira delas é responsável por permitir a entrada de novos clientes, sem a intervenção do administrador, negociando os parâmetros iniciais necessários. A classe *ClientManager* notifica a classe *MainManager* da chegada e saída de clientes e controlar a transmissão dos fluxos de áudio e vídeo para estes clientes, podendo inclusive renegociar com os mesmos os parâmetros estabelecidos no início da transmissão. Por exemplo, é possível mudar a forma de transmissão de *unicast* para *multicast* ou vice versa, de acordo com as condições da rede em um dado instante. A classe *ClientInterface* é responsável por acessar as informações contidas no buffer do objeto *DataBuffer*, através do método *GetDataFromBuffer()* da classe *MainManager,*, adequá-las de acordo com o tipo da aplicação cliente associada à classe (por exemplo adequar um fluxo MPEG para ser transportado utilizando-se o protocolo RTP) e finalmente transmitir estas informações.

4 Servidor Secundário de Difusão de Vídeo

Um Servidor de Fluxos de áudio e vídeo pode ser utilizado como refletor, ou Servidor Secundário de um outro Servidor de Fluxos, ou Servidor Primário. Este tipo de configuração é ilustrada na Figura 3. Podendo ser implementada especificando-se as classes *ClientInterface* e *ClientManager* no servidor primário, de forma a possibilitar a transmissão dos fluxos de vídeo e das informações de controle respectivamente, para as classes *DataManager ClientInterface* no Servidor Secundário. Estas últimas por sua vez, devem ser especializadas para permitir a recepção destes dados. Atualmente, o código do servidor secundário está sendo implementado como um agente móvel.

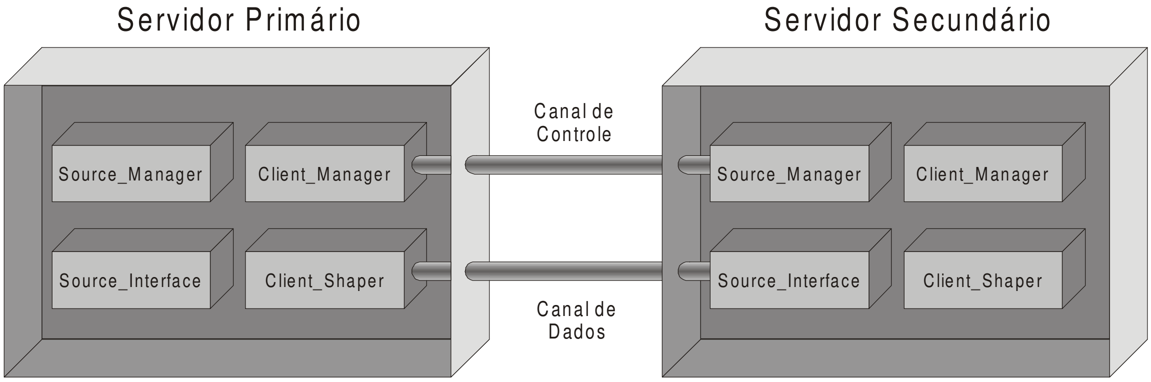

Figura 3 – Configuração ilustrativa de um Servidor Secundário

5 Implementação do Servidor de Transmissão de áudio e vídeo

O servidor de transmissão de fluxos de áudio e vídeo foi implementado, de forma a possibilitar a transmissão de fluxos de vídeo para os clientes: MTV (MPEG-1 utilizando TCP, UDP, HTTP e IP Multicast), VideoLan (MPEG-2 utilizando UDP), Windows Media Player (MPEG-1 e MPEG-2 utilizando HTTP), RealAudio (MPEG-1 utilizando HTTP), e JMF (MPEG-1 utilizando RTP).

Nos experimentos realizados no Laboratório NatalNet (Projeto ProTeM/RNP REMAV), o fluxo de vídeo foi obtido em tempo real a partir de uma placa de captura Apolo, instalada em um PC Pentium II 400 com 64 MB de memória RAM e com Sistema Operacional Microsoft Windows NT 4.0. Neste servidor foi criado um túnel mapeado no sistema de arquivos (*named pipe*) com um buffer de 8 MB. O software da placa de captura escreve constantemente o fluxo gerado pela mesma no túnel. Um programa denominado *streamer* foi implementado com base em duas *threads*. Uma delas espera por pedidos de conexão e os aceita caso não haja nenhum outro cliente (Servidor de Fluxos) conectado. A outra *thread* lê constantemente os fluxos MPEG do túnel e os transmite para o possível cliente através de uma conexão TCP de dados conforme ilustra a Figura 4-a. Caso o buffer do túnel comece a encher, o streamer esvazia o buffer do túnel e notifica o Cliente (Servidor de Fluxo) através de uma conexão TCP de controle. O diagrama de atividades para esta *thread* é mostrado na Figura 4-b.

O Servidor de Fluxos propriamente dito foi implementado utilizando-se a linguagem C++ e instalado no sistema operacional Linux. A classe *SourceInterface* foi implementada de forma a receber os fluxos de dados, enquanto a classe *SourceManager* os dados de controle, a partir do *streamer*. Como já mencionando em seções anteriores, o objeto da classe *SourceInterface* armazena os segmentos de dados que compõem os fluxos de áudio/vídeo em um buffer presente na classe *DataBuffer*, através do método *PutDataOnBuffer()* da classe *MainManager*.

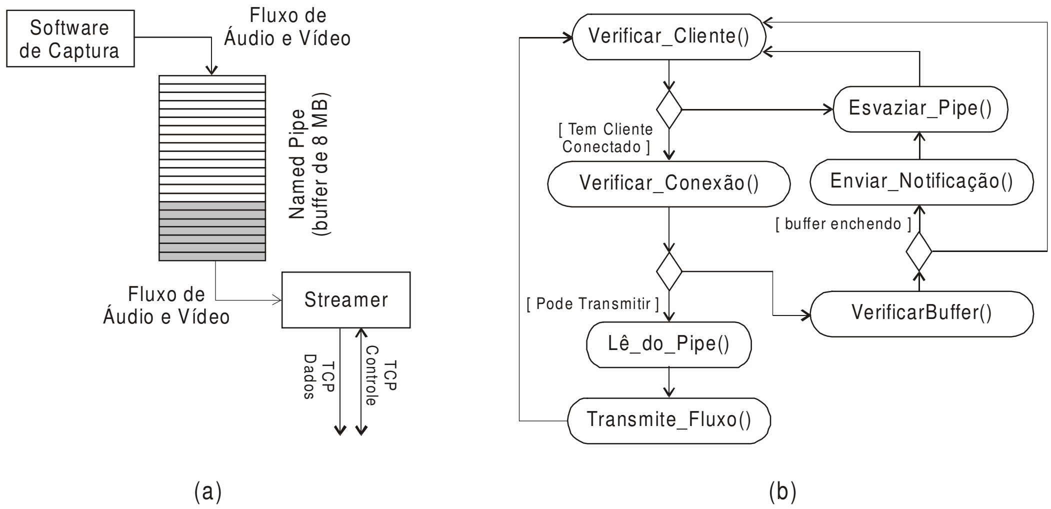

Figura 4 (a) – Ilustração da utilização de um túnel para a comunicação entre o software de captura e o *streamer*; (b) – Diagrama de atividades para a transmissão de fluxos pelo *streamer*

5.1 Implementação das classes ClientManager e ClientInterface:

As classes *ClientManager* e *ClientInterface* foram especificadas e implementadas, de forma diferente para cada um dos protocolos utilizados.

5.1.1 Classes *ClientManager* e *ClientInterface* para Transmissão utilizando o Protocolo TCP

Para transmissão do fluxo MPEG utilizando-se o protocolo TCP, a classe *ClientManager* foi implementada possuindo um método principal que fica bloqueado aguardando por pedidos de conexão. Após a chegada de um pedido de conexão, esta é aceita, o seu descritor [20] é colocado numa *lista de espera* e é solicitado à classe *MainManager* a inserção de um novo cliente através do método *AddClient()*. Em seguida este método fica bloqueado, aguardando por novos pedidos.

A classe *ClientInterface* possui um método principal que lê os dados do buffer e os envia para todos os clientes cujos descritores estão na *lista de envio*. Caso não seja possível enviar os dados a algum cliente em particular, seu descritor é removido desta lista e é solicitado ao objeto da classe *MainManager*, a remoção deste cliente. Havendo algum cliente na *lista de espera*, o método percorre os dados à procura do código de início de seqüência MPEG. Uma vez encontrado este código, o fluxo é enviado para todos os clientes cujos descritores estão na *lista de espera*, a partir desse código, e esses descritores são movidos da *lista de espera* para a *lista de transmissão*. O método então volta a ler do buffer, reiniciando todo o processo.

5.1.2 Classes *ClientManager* e *ClientInterface* para Transmissão utilizando o Protocolo HTTP

Para transmissão do fluxo MPEG utilizando-se o protocolo HTTP, a classe *ClientManager* foi implementada de forma semelhante àquela utilizada na transmissão utilizando-se o TCP. A única diferença é que antes de se inserir um cliente na lista de espera, um *handshake* utilizando-se o protocolo HTTP é realizado. Durante este *handshake*, o servidor informa ao cliente que o conteúdo não deve ser mantido em cache, através dos campos de cabeçalho: “Pragma: no-cache” e “Cache-Control: no-cache”. O formato do fluxo é informado através do campo de cabeçalho

“Content-Type: video/mpeg”. O campo de cabeçalho “Content-Length: 45000000” informa o tamanho em bytes do suposto arquivo de vídeo sendo transmitido, devendo informar um valor bastante alto. A presença deste campo se deve ao fato de que, quando o mesmo está ausente, alguns *players* só começam a exibir o vídeo ou áudio quando o arquivo é transmitido por completo do servidor para o cliente. O grande problema com este campo é que alguns *players* simplesmente param de receber quando a quantidade de bytes recebidos é igual àquela especificada por este campo.

A classe *ClientInterface* foi implementada de forma idêntica àquela do TCP.

5.1.3 Classes *ClientManager* e *ClientInterface* para Transmissão utilizando o Protocolo UDP

No caso do protocolo UDP, a classe *ClientManager* foi implementada de forma a possibilitar a inserção e remoção manual de clientes. Quando um cliente é adicionado, um *socket* [20] para o mesmo é criado e o seu identificador é inserido em uma *lista de transmissão*.

A classe *ClientInterface* possui um método principal que lê os dados do buffer e os envia a todos os clientes cujos identificadores estão presentes na *lista de transmissão*, voltando então a ler do buffer.

Para transmissão de fluxos MPEG-2, o servidor *VideoLan Server* [12], de código aberto, foi utilizado para gerar os pacotes de transmissão. Este servidor foi modificado para ler o fluxo do buffer e enviá-lo à um objeto da classe *ClientInterface*. Esta classe por sua vez foi modificada para receber o fluxo do *VideoLan Server*, ao invés de lê-lo do buffer.

5.1.4 Classes *ClientManager* e *ClientInterface* para Transmissão utilizando o IP Multicast

Para transmissão utilizando-se o IP Multicast, as classes *ClientManager* e *ClientInterface* são idênticas àquelas utilizadas para transmissão utilizando-se UDP. A única diferença é que a classe *ClientManager* vai receber um endereço IP classe D ao invés de um endereço IP unicast.

5.1.5 Classes *ClientManager* e *ClientInterface* para Transmissão utilizando RTP (MPEG-1 vídeo)

Para transmissão de um fluxo de vídeo MPEG-1 utilizando-se o protocolo de transporte RTP, a classe *ClientManager* foi implementada de forma idêntica àquela utilizada para transmissão UDP.

A classe *ClientInterface,* por sua vez, foi implementada com um método principal que lê os dados do buffer e então os percorre, de forma a identificar as estruturas de dados do padrão MPEG presentes no fluxo, bem como alguns campos dos cabeçalhos dos *Frames* MPEG, cujos valores são colocados em campos dos cabeçalhos permanente e específico do pacote RTP. Assim sendo, o vídeo MPEG-1 é empacotado da obedecendo-se às seguintes regras [15]:

- O campo Payload Type sempre contém valor 32 (MPEG-1 vídeo);
- Os campos MBZ, T, AN e N do cabeçalho específico sempre contém valor zero;
- Quando uma seqüência é identificada:
    - O segmento de dados que vai do código de início de seqüência até o início do primeiro *GOP* é colocado em um buffer local;

    - O campo S (*begin of Sequence*) do cabeçalho especifico do RTP recebe valor 1;
- Quando um *GOP* é identificado:
    - A variável global *TimeStampBase* recebe o valor do *Time Stamp* corrente;
    - O segmento de dados que vai do código de início do *GOP* até o início do primeiro *Frame* é colocado no buffer local;
- Quando um *Frame* é identificado:
    - Os valores dos campos do cabeçalho especifico do RTP: TR, P, FBV, BFC, FFV, FFC são retirados dos campos: TR, PCT, FPBV, BFC, FPFV e FFC respectivamente, que estão presentes no cabeçalho do *Frame*;
    - O *Time Stamp* é calculado com base no campo TR. Para vídeo do tipo NTSC, por exemplo, *Time Stamp* = *TimeStampBase* + TR * 3003;
    - O segmento de dados que vai do código de início do *Frame* até o início do próximo *Slice* é colocado no buffer local;
- Quando um *Slice* é identificado:
    - Os campos B (*Begin of slice*) e E (*End of slice*) do cabeçalho especifico do RTP recebem valor 1;
    - O segmento de dados que vai do código de início do *Slice* até o início do próximo *Slice*, *Frame*, *GOP* ou *Sequence* é colocado em um buffer local;
    - Quando a próxima estrutura não for um *Slice* ou o tamanho do buffer local somado ao tamanho do próximo *Slice* for maior que o tamanho máximo de um quadro RTP:
        - Se a próxima estrutura for um início uma seqüência, o bit M do cabeçalho fixo do RTP recebe valor 1;
        - O buffer local é empacotado e transmitido;
        - O campo S recebe o valor 0.

No empacotamento, se o tamanho do buffer local for maior que o tamanho máximo do pacote RTP, o mesmo é dividido em vários pacotes. Como sempre existe pelo menos um *Slice* no buffer local, o primeiro pacote recebe valor 1 para o campo B (*Begin of slice*) e valor 0 para o campo E (*End of slice*) do cabeçalho especifico do RTP. Os demais pacotes recebem valor 0 para o campo S e valor 1 para o campo E.

6 Resultados Obtidos

O *Servidor de Fluxos* foi instalado em um PC Pentium MMX 300 com 128 Mb de memória RAM, placa de rede Ethernet de 100 Mbps, e sistema operacional Linux.

Um fluxo MPEG-2, gerado em tempo real a uma taxa de 4 Mbps, foi constantemente transmitido para o *Servidor de Fluxos* através de uma conexão TCP com o programa *streamer*. Este fluxo foi gerado pela placa de captura Apolo, instalada num PC Pentium II 400 com 64 MB de memória RAM e com Sistema Operacional Microsoft Windows NT 4.0.

A Figura 5 mostra o percentual de utilização da banda passante do enlace, que conecta o *Servidor de Fluxos* a um Switch IBM 8266, em função do tempo. Nessa figura pode-se observar duas regiões distintas: uma em que o servidor está desligado e, portanto, o percentual de utilização da banda passante é praticamente nulo; e outra região na qual o *Servidor de Fluxos* está ativo e transmitindo datagramas UDP para um endereço multicast. Nesta região o percentual de utilização da banda passante é de aproximadamente 9,5 %, ou seja, 9,5 Mbps. Como o fluxo de vídeo é gerado a uma taxa constate de 4 Mbps, o recebimento e a retransmissão deste fluxo consomem 8Mbps, ou seja, 8% da banda passante. Tendo em vista que nenhuma outra aplicação relevante estava fazendo uso do enlace naquele momento, fica fácil concluir que 1,5 % da banda passante do enlace estava sendo consumida pelo *overhead* dos protocolos de transporte, enlace e rede.

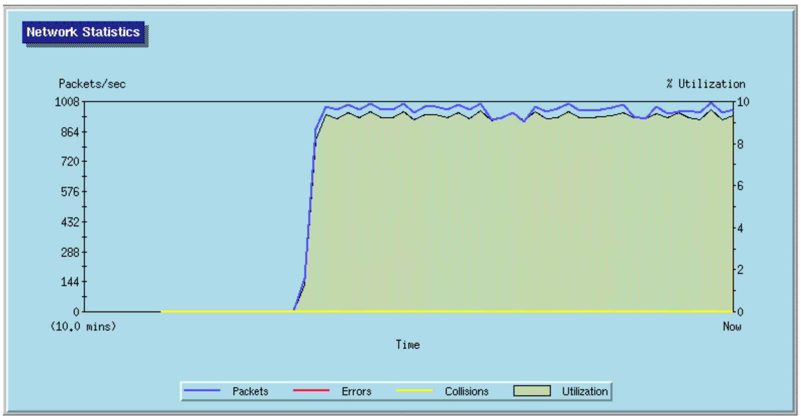

Figura 5 – Percentual de utilização da banda passante do enlace do Servidor de Fluxos.

Em um outro experimento, um *Servidor Secundário* foi configurado em um PC Pentium III 600 com 128 Mb de memória RAM, placa de rede Ethernet de 100 Mbps, e com sistema operacional Linux. Este servidor recebeu um fluxo de áudio e vídeo MPEG-1 de 4 Mbps diretamente do *Servidor de Fluxos*, e conseguiu transmiti-lo para 20 clientes MTV utilizando o protocolo UDP. Nesse caso, a utilização da banda passante do enlace que interliga o *Servidor Secundário* à rede chegou a 100%. Esse foi o fator limitante no número de clientes atendidos pelo servidor. A partir de 22 clientes, notou-se uma diminuição na qualidade do vídeo exibido pelos mesmos.

O Serviço de Transmissão de Fluxos foi testado em rede local com os seguintes clientes: *MTV* (MPEG-1 utilizando TCP, UDP, HTTP e IP Multicast), *VideoLan* (MPEG-2 utilizando UDP), *Windows Media Player* (MPEG-1 e MPEG-2 utilizando HTTP), *RealAudio* (MPEG-1 utilizando HTTP), e *JMF* (MPEG-1 utilizando RTP). Em todos estes clientes pôde-se observar um vídeo e áudio contínuo e de boa qualidade. A Figura 6 mostra um vídeo MPEG-1 sendo exibido no cliente *MTV* e a Figura 7 um vídeo MPEG-2 sendo exibido no cliente *VLC*.

O *Windows Media Player*, quando está utilizando o protocolo HTTP, possui uma característica especial, que é a de esperar receber um arquivo de tamanho limitado, ao invés de um fluxo de duração indeterminada. Quando não são informados previamente do tamanho do arquivo, estes clientes só iniciam a exibição do vídeo quando o mesmo é completamente transmitido do servidor. Porém quando são informados do tamanho de um suposto arquivo, estes clientes exibem o vídeo por um tempo correspondente ao tamanho informado.

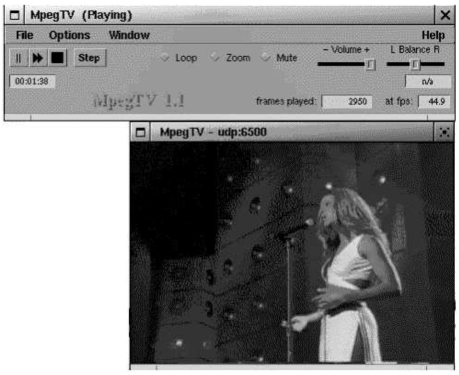

Figura 6 – Vídeo MPEG-1 sendo exibido no cliente MTV

Figura 7 – vídeo MPEG-2 sendo exibido no cliente VLC

7 Conclusões

Este trabalho apresentou um Serviço de Distribuição Dinâmica de Vídeo com ênfase na implementação do servidor de transmissão de áudio e vídeo.

O Serviço de distribuição de áudio/vídeo foi parcialmente implementado através da implementação de um *Servidor Primário* e um *Servidor Secundário*, de forma a tornar possível a transmissão de fluxos no formato MPEG, gerados em tempo real, para clientes *MTV*, *VideoLan*, *Windows Media Player*, *RealAudio* e *JMF*. Nestes clientes pôde-se observar um áudio e vídeo nítido e contínuo. Devido à forma como foram especificados, o suporte a novos tipos de clientes e fontes de dados pôde ser facilmente adicionado aos servidores, através da definição de módulos específicos para cada tipo de cliente ou fonte de dados.

Dando continuidade a este trabalho, estão sendo desenvolvidas implementações do *Servidor Secundário* como agentes móveis, de forma a permitir a migração em tempo real de *Servidores Secundários* para pontos específicos da rede. A implementação do serviço de Monitoramento de Tráfego foi concluída em um outro trabalho [1] e, atualmente, está sendo realizada a sua integração com os outros módulos do DynaVideo. Os módulos de Otimização de Configuração e de Gerência de Falhas estão sendo desenvolvidos em outros trabalhos que já se encontram em fase de implementação.

O uso de uma abordagem baseada em configuração, com o *Gerente Dynavideo* sendo o módulo responsável pela configuração e, portanto, o elemento integrador dos demais módulos, tem facilitado bastante o desenvolvimento separado dos módulos e, a integração de novos serviços à medida que é identificada a necessidade de acrescentar uma nova facilidade ao serviço de distribuição de vídeo.

Referências bibliográficas

[1] Madruga, Marcos; Batista, Thais; Lemos, Guido. "*SMTA: Um Sistema para Monitoramento de Tráfego em Aplicações Multimídia*". XXVI Conferência Latinoamericana de Informática (CLEI). Cidade do México – México. Setembro de 2000.

[2] Booch, G.; Jacobson, I..; Rumbaugh, J.. "*The Unified Modeling Language for object-oriented Development*". Documentation Set Version 0.91 Addendum UML Update, Setembro de 1996.

[3] ISO/IEC International Standard 11172. "*Coding of moving pictures and associated audio for digital storage media up to about 1,5 Mbits/s*". Novembro de 1993.

[4] ISO/IEC International Standard 13818. "*Generic coding of moving pictures and associated audio information*". Novembro de 1994.

[5] Mcgowan, John F.. "*AVI Overview*". 1998. < http://www.jmcgowan.com/avi.html>.

[6] Apple, Inc.. "*QuickTime*". <http://www.apple.com/quicktime/>

[7] RealNetworks. "*Realserver Administration Guide RealSystem G2*". <http://www.real.com/serveradminguideg2.pdf>.

[8] VideoCharger, *IBM DB2 Digital Library*. "*VideoCharger Server Key Features*". <http://www-4.ibm.com/software/data/videocharger/vcserverkey.html>. Janeiro de 2000.

[9] Vosaic, Inc.. "*Vosaic Media Server*". <http://www.vosaic.com>

[10] Microsoft, Inc., "*Windows Media Technologies*". <http://www.microsoft.com/windowsmedia/>.

[11] Vivo. "*VivoActive Producer*". <http://www.vivo.com>.

[12] École Centrale Paris. "*VideoLan*". < http://www.videolan.org>.

[13] Schulzrinne, H.. Casner, S.; Frederick, R.; V. Jacobson. "*RTP: A Transport Protocol for Real-Time Applications*". RFC 1889. Janeiro de 1996.

[14] Schulzrinne, H.. "*RTP Profile for Audio and Video Conferences with Minimal Control*". RFC 1890. Janeiro de 1996.

[15] Hoffman, D.; Fernando, G.; Goyal, V.; M. Civanlar. "*RTP Payload Format for MPEG1/MPEG2 Video*". Janeiro de 1998.

[16] Deering, S.. "*Host Extensions for IP Multicasting*". STD 5, RFC 1112. Agosto de 1989.

[17] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T.. "*Hypertext Transfer Protocol -- HTTP/1.1*". RFC 2616. Junho de 1999.

[18] J. Postel.. "*User Datagram Protocol (UDP)*". RFC 768. Agosto de 1980.

[19] J. Postel.. "*Transmission Control Protocol (TCP)*". RFC 793. Setembro de 1981.

[20] Stevens, W. R.. "*UNIX Network Programing - Networking APIs:Sockets and XTI* ". Prentice-Hall, Inc., 1998. Segunda Edição.

[21] R. Braden, Ed.; L. Zhang; S. Berson; S. Herzog; S. Jamin. "*Resource ReSerVation Protocol (RSVP)*". RFC 2205. Setembro de 1997.

[22] Soares, L. F. G.; Lemos, G.; Colcher, S.. "*Redes de Computadores: das LANs, MANs e WANs às Redes ATM*". Ed. Campus, 1995. Segunda Edição.

[23] Tanenbaum, A. S.. "*Modern Operating System*". Prentice-Hall, Inc., Engelwood Cliffs. 1992.

[24] Façanha, Roberto de Almeida. "*Gerenciamento de Banda Passante em Servidores de Vídeo*"**.** Universidade Estadual de Campinas.

[25] BMRC, Berkley Multimedia Research Center. "*Berkley Mpeg Tools*". <http://bmrc.berkeley.edu/research/mpeg/>.

[26] Banks, Doug; Rowe, Lawrence A.. "*Analysis Tools for MPEG-1 Video Streams*". Department of Computer Science – EECS. University of California at Berkeley. http://bmrc.berkeley.edu/research/publications/1997/139/dbanks.html

[27] Kon, F.; Campbell, R. et al. Dynamic Reconfiguration of Scalable Internet Systems with Mobile Agents. Technical Report, Department of Computer Science at the University of Illinois at Urbana-Champaign, 1999.

[28] Parveen Kumar, L. H. Ngoh, A. L. Ananda. A Programmable Audio/Video Streaming Framework for Broadband Infrastructures. Network Storage Symposium - NetStore '99. Seattle, WA. October, 1999.

[29] Brian Noble, Ben Fleis, Minkyong Kim, Jim Zajkowski. Fluid Replication. Network Storage Symposium - NetStore '99. Seattle, WA. October, 1999.