Um Ambiente de Distribuição de Vídeo MPEG2 com Suporte Multicast em Código Aberto para o Projeto I2TV

Proposta de apresentação de trabalho na área de Vídeo Digital para o 4º Workshop RNP2

Alexandre Schulter schulter@inf.ufsc.br
Leonardo A. Ribeiro lar@inf.ufsc.br
Valdecir Becker valdecir@nurcad.ufsc.br
Marcos F. Caetano caetano@lisha.ufsc.br
Carlos Montez montez@das.ufsc.br
Edison Melo melo@npd.ufsc.br
Antônio A. M. Fröhlich guto@inf.ufsc.br

Núcleo de Redes de Alta Velocidade e Computação de Alto Desempenho -NURCAD
Centro Tecnológico – Universidade Federal de Santa Catarina – CTC/UFSC
Caixa Postal 476 – 88040-900 – Florianópolis – SC

1. Introdução

O projeto I2TV [1] busca estabelecer uma infra-estrutura para o desenvolvimento e testes de programas e ferramentas de TV digital interativa no Brasil. Sua abrangência de pesquisa engloba *set-top boxes* (hardware), servidores de vídeo e novas formas de distribuição de vídeos (*streaming*) considerando parâmetros de QoS da rede, e até o desenvolvimento de conteúdo para o novo tipo de mídia que vem surgindo com essas tecnologias.

Este trabalho trata, especificamente, da implementação de um ambiente de transmissão de vídeo MPEG-2, com suporte a difusão (*multicast)* e código aberto baseado no software distribuidor de vídeo *Videolan Server* (VLS) [4]. Este ambiente está sendo proposto pela equipe da Universidade Federal de Santa Catarina – UFSC – visando sua integração aos experimentos do Projeto I2TV. Concentramos nossos relatos em alguns desafios que nosso grupo teve de superar para implementar nosso ambiente de distribuição de vídeos.

2. Ambiente de Testes e Primeiros experimentos

Grande parte da infra-estrutura instalada para o Projeto de Redes Metropolitanas de Alta Velocidade de Florianópolis [2] foi aproveitada no projeto I2TV. Além do suporte de redes (tais como roteadores configurados para *multicast*), nosso projeto teve como ponto de partida a disponibilidade de uma câmera digital de alta definição, placa codificadora de MPEG2 Apollo Expert, da DV Studio [3], e placas decodificadoras MPEG2 Hollywood Plus [7].

O primeiro desafio que nos deparamos foi a utilização da placa codificadora em nossas transmissões, pois esta é originalmente projetada para autoria de DVD, e o formato gerado (MPEG2-PS – *Program Stream*) é inadequado para ser transmitido em rede.

Inicialmente, implementamos diversas versões de programas em linguagem C que interagiam com a placa através de mecanismos de comunicação do sistema operacional (*pipe*) e transmitiam pela rede o conteúdo gerado (áudio e vídeo MPEG2-PS). No outro ponto final da rede, implementamos programas que usavam a placa decodificadora para a recepção e apresentação do vídeo. Infelizmente, essa solução se mostrou insatisfatória, pois além de ser uma abordagem adequada apenas para transmissões ponto a ponto (*unicast*), o uso do formato MPEG2-PS exige bastante código adicional no programa servidor, no sentido de ajustar a taxa de transmissão (*shaping*).

3. Experimentos *multicast*

Devido à limitação descrita, optamos por adotar o software distribuidor de vídeo *Videolan Server* (VLS) [4], que começou a ser desenvolvido por alunos da École Centrale Paris, na França, e hoje conta com desenvolvedores de vários países. Esse projeto nos atraiu por dois grandes motivos: (i) possui código aberto, e (ii) permite a transmissão *multicast* de vídeo no

formato MPEG2-TS. Outras características interessantes desse software é que ele funciona em redes IPv4 e IPv6, usa os protocolos UDP e RTP, e suporta vários sistemas operacionais.

Na extremidade final da rede, parte cliente, passamos a utilizar também o software *Videolan Client* (VLC) [4], desenvolvido pelo mesmo grupo de pesquisa, e que consegue fazer a decodificação de vídeo sem auxílio de hardware. Essa é uma característica bastante atraente, pois permite que os microcomputadores convencionais possam ser usados como clientes da transmissão de vídeo.

O software VLS distribui vídeos lendo arquivos MPEG2-PS (*Program Stream*), e transformando-os em MPEG2-TS (*Transport Stream*). A diferença entre os dois formatos, é que este segundo é preparado para ser transportado em redes, carregando informações úteis para transmissões em ambientes onde há ruído e a perda de pacotes é freqüente [6].

Iniciamos as transmissões testando a eficiência do VLS com arquivos MPEG1 e MPEG2 armazenados. O resultado foi uma transmissão ponto a ponto estável. Contudo, estávamos interessados na transmissão de vídeos gerados em tempo real, em uma rede que envolvesse suporte *multicast*, conforme mostrado em nosso ambiente de transmissão apresentado na Figura 1.

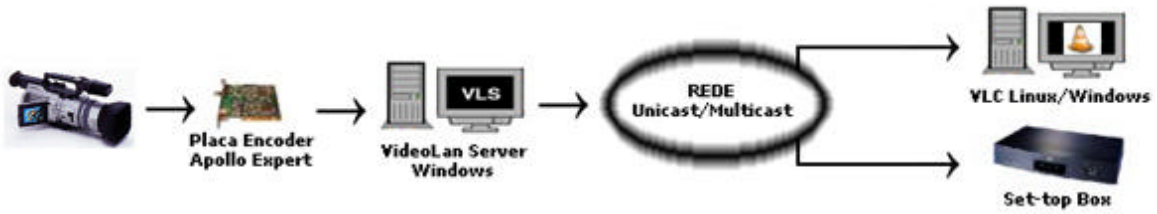

Figura 1. Visão geral do ambiente de transmissão.

Concentramos nossos testes em transmissões *multicast* de vídeos no formato MPEG2, usado nas TVs digitais. No ambiente de transmissão, a placa codificadora Apollo Expert recebe o vídeo capturado de um vídeo-cassete comum ou de uma filmadora e entrega no formato MPEG2-PS ao software distribuidor VLS. Esse software converte em tempo real o vídeo para o formato MPEG2-TS e o distribui, preferencialmente usando o suporte *multicast* da rede. O nosso objetivo principal é que o vídeo seja recebido e apresentado por set-top boxes, preparados para apresentação de programação interativa. Entretanto, num primeiro momento, passamos a usar o software VLC para receber e mostrar os vídeos.

Para concretizar esse ambiente, nos deparamos com três novos desafios:

(i) era necessário fazer a comunicação entre o vídeo gerado pela placa codificadora (a qual só tínhamos o código executável) e o VLS;

(ii) o software da placa codificadora só executava no Windows, contudo o *multicast* não funcionava no VLS quando este era executado no Windows; e principalmente,

(iii) o vídeo gerado em tempo real, não apresentava recepção estável, apresentando paradas intermitentes, como se quadros estivessem sendo perdidos.

Os três problemas citados foram exacerbados pelo fato de desejarmos executar o VLS no Windows. A placa codificadora só possui *drivers* para esse sistema operacional, e o VLS não apresenta suporte adequado no Windows (este software possui suporte pleno apenas em versões UNIX). Apesar de ser possível compilá-lo no Windows, seu funcionamento nesse sistema operacional apresenta uma série de limitações.

4. Resolução dos problemas

O primeiro desafio foi o mais simples de solucionar, pois já havíamos resolvido o mesmo problema anteriormente quando fizemos alguns programas para a comunicação direta com a placa codificadora, usando suporte para comunicação via *pipe* (*named pipe*) do Windows.

No segundo desafio, o do não funcionamento do *multicast*, após eliminada a possibilidade de ser problema da rede, começamos a inspecionar o código-fonte do VLS, e descobrimos alguns erros de codificação desse programa em sua versão para Windows, que impediam a configuração correta do valor do parâmetro de rede TTL (*Time to Live*).

O terceiro desafio sem dúvida foi o mais complexo. Quando executávamos nossos experimentos, percebíamos que se iniciássemos a geração de um arquivo MPEG2-PS com a placa Apollo Expert, e depois de alguns segundos de codificação, executássemos o VLS para difundir o vídeo pela rede, a sincronização Apollo/VLS não era perfeita. Aparentemente, o VLS transmitia mais rápido do que a Apollo codificava, e após algum tempo, o vídeo recebido apresentava paradas intermitentes, como se quadros estivessem sendo perdidos. Quanto maior o tempo que deixávamos codificando antes de iniciar o VLS, mais tempo durava a transmissão. Por exemplo, com cinco segundos de codificação antes de iniciar o servidor, a transmissão durava cerca de dois minutos; e com 60 segundos, uma hora e meia.

Com objetivo de resolver esse problema, foi preciso estudar quase todo o código-fonte do programa, e entender a arquitetura geral do VLS, apresentada na Figura 2. Essa arquitetura pode ser dividida em três partes: a *thread* produtora, a *thread* consumidora e a parte de gerenciamento [5]. Estas partes são subdivididas em módulos, e os módulos a serem utilizados dinamicamente pelo programa, dependem do formato de vídeo, da fonte de leitura e do destino do fluxo de vídeo.

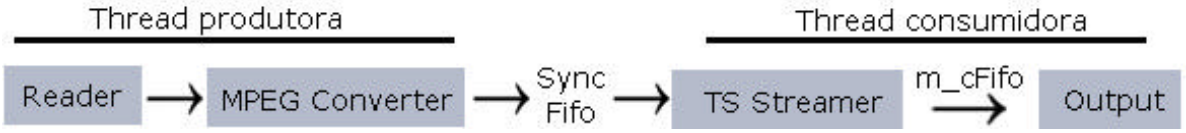

Figura 2. Arquitetura Geral do VLS.

A parte do programa responsável pelo gerenciamento faz a configuração da transmissão, inicia as duas *threads*, e controla a administração remota. A *thread* produtora usa um módulo de leitura, tal como o *filereader*, que faz leitura de arquivos ou como o *videoreader*, que faz leitura de placas codificadoras. Os pacotes lidos são então convertidos para outro formato, neste caso, do MPEG2-PS para MPEG2-TS, e escritos num *buffer* (*SyncFifo*), para sincronização com a *thread* consumidora. A thread produtora lê do *named pipe* e escreve o mais rápido possível para poder encher o *SyncFifo*. A *thread* consumidora, módulo *streamer*, controla a velocidade da transmissão levando em conta o campo *Program Clock Reference* (PCR), presente nos pacotes do *TS*. O módulo *streamer* envia os pacotes para outro *buffer* que é usado pelo módulo de saída para agrupar vários pacotes TS em um pacote UDP, juntamente com cabeçalhos do protocolo RTP. O módulo de saída pode ser a rede ou um arquivo.

As modificações principais foram feitas no módulo *filereader*, que cria o *named pipe* e faz sua leitura. Foram trocadas algumas funções de manipulação de arquivos, incompatíveis com *named pipes* por suas respectivas funções na API do Windows. O *filereader* modificado detecta a natureza do arquivo, *named pipe* ou arquivo normal, e usa os procedimentos adequados para cada caso. Configurando-se o VLS para ler um arquivo MPEG2-PS de determinado *named pipe*, e a Apollo para escrever no mesmo, a transmissão começa apresentando o mesmo problema: o vídeo continua com pequenas "travadas", como se estivesse "perdendo" quadros.

Mantendo o buffer do *named pipe* com no mínimo 200 KB, o problema não ocorre. A

razão disso ainda é desconhecida. Este é um dos motivos para a não utilização do módulo *videoreader*, pois nesse caso, a taxa de bits (*bitrate*) não seria levada em conta, e a velocidade da transmissão seria governada pela disponibilidade de dados no *buffer* do *named pipe*, ou seja, pela velocidade de escrita da placa codificadora. Conseqüentemente, o buffer ficaria sempre inferior aos 200 KB.

Outras modificações foram feitas no *filereader* para que bloqueie o VLS até o *buffer* do *named pipe* alcançar o patamar de 200 KB. Isto levou a uma transmissão de vídeo adequada, sem a sensação da perda de quadros. No entanto, após certo tempo, vinte minutos aproximadamente, as "travadas" recomeçavam. Teoricamente, a quantidade de dados no *buffer* do *pipe* deveria se manter estável, mas foi detectado que a quantidade de dados neste tende a diminuir com o tempo, devido a rápidas acelerações da leitura (picos de leitura). O motivo pelo qual ocorrem esses picos de leitura ainda é desconhecido. Acreditamos que possa ser algum defeito da conversão PS-TS, ou na temporização realizada pelo *streamer*. Problemas relacionados à codificação realizada pela placa já foram descartados.

De qualquer forma, a solução adotada para este problema foi fazer com que a quantidade de dados no *buffer* do *pipe* se "regenere" após um pico de leitura. Isto foi feito aumentando-se o patamar do *buffer* para 1000 KB, que após vários picos de leitura, torna-se inferior a 200 KB. Logo que isto acontece, a *thread* produtora é suspensa até que o *buffer* volte ao seu patamar. Devido às suspensões da *thread* produtora serem de curta duração, e aos *buffers* internos do VLS, as interrupções na transmissão são imperceptíveis no cliente.

5. Considerações finais

Conseguimos implementar um ambiente de transmissão de vídeo MPEG2 estável, através de um servidor flexível a outras modificações, uma vez que sua implementação está dominada, e que permite ao projeto uma base sólida para os experimentos de interatividade a serem desenvolvidos a seguir. É nesta parte que surgem os desafios, e onde percebemos o quanto ainda se tem por fazer. Acreditamos que o próximo passo é reduzir ainda mais o atraso (*delay*) do vídeo entre a captura e a transmissão, hoje em três segundos, integrar o servidor com aplicações de TV interativa, multiplexando vários canais de vídeo, e melhorar o serviço de vídeo sob demanda, já disponível no VideoLan Client [4], usando-se o protocolo HTTP.

Outro importante passo seria o de implementar um módulo que possibilite ao VLS trabalhar com uma variedade maior de placas e sistemas operacionais, uma vez que o esforço foi dedicado ao sistema Windows devido a restrições de *drivers* da placa codificadora.

6. Referências

[1] Projeto I2TV - Infra-estrutura para Desenvolvimento e Teste de Ferramentas e Programas para TV Interativa, http://www.i2tv.ufsc.br/

[2] Rede Metropolitana de Alta Velocidade de Florianópolis. http://www.rmav-fln.ufsc.br

[3] DV Studio Technologies. http://www.dv-studio.com/

[4] Projeto VideoLAN, http://www.videolan.org/

[5] DEGUET, Cyril; LETEURTRE, Tristan. "VideoLAN Server developer documentation", Disponível em: http://developers.videolan.org/vls/doc/developer/vls-devel.pdf.

[6] ISO/IEC Standard 13818-1:2000, "Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Systems".

[7] Sigma Designs, http://www.sigmadesigns.com.