

JAUBER LOPES URBIÊTA

ESTUDO DAS SOLUÇÕES DE TRANSMISSÃO DE VÍDEO UTILIZANDO
SOFTWARE LIVRE

Monografia de Pós-Graduação “*Lato Sensu*” apresentada
ao Departamento de Ciência da Computação para
obtenção do título de Especialista em “Administração
em Redes Linux”

Orientador
Prof. Joaquim Quinteiro Uchôa

LAVRAS
MINAS GERAIS – BRASIL
2007

JAUBER LOPES URBIÊTA

ESTUDO DAS SOLUÇÕES DE TRANSMISSÃO DE VÍDEO UTILIZANDO
SOFTWARE LIVRE

Monografia de Pós-Graduação “*Lato Sensu*” apresentada
ao Departamento de Ciência da Computação para
obtenção do título de Especialista em “Administração
em Redes Linux”

APROVADA em 29 de Abril de 2007

Prof. Heitor Augustus Xavier Costa

Prof. Denilson Vedoveto Martins

Prof. Joaquim Quinteiro Uchôa
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

Agradecimentos

Primeiramente a Deus pelo que Ele representa pra mim,
aos meus pais pelo apoio e aconselhamento e
a minha noiva pela paciência dos finais de semana.

RESUMO

Este trabalho tem como objetivo mostrar a transmissão de vídeo utilizando somente *software* livre em todos os níveis, desde a criação do formato que contém o conteúdo multimídia até o posterior envio por um servidor de *streaming*. Aborda definições de *streaming*, CODEC e *container*. Demonstra a utilização de ferramentas livres para suporte e discute estudo de caso de alguns aplicativos servidores de *streaming* mais conhecidos como Flumotion, Icecast, VLC e um não tão conhecido como JRoar. Procura demonstrar na prática estas tecnologias e apresenta conclusões geradas a partir de sua utilização.

Sumário

1. Introdução.....	1
2. Streaming.....	4
2.1. Vídeo sob demanda (Vídeo <i>On Demand</i> - VoD).....	6
2.2. <i>Streaming</i> ao vivo de Vídeos e Áudios (<i>live streaming</i>).....	8
2.3. <i>Unicast</i> x <i>Multicast</i>	9
3. Recursos Tecnológicos.....	11
3.1. Protocolos.....	11
3.2. Diferenças entre <i>Container</i> e CODEC.....	15
3.2.1. CODEC.....	15
3.2.2. <i>Container</i>	17
3.3. Ferramentas de Suporte.....	25
4. Estudo de Casos de Aplicação de <i>streaming</i>.....	29
4.1. Flumotion.....	29
4.2. Icecast.....	46
4.3. VLC.....	51
4.4. JRoar.....	57
4.5. Cortado.....	59
4.6. Resultados Obtidos.....	62
5. Conclusão.....	67
6. Referências Bibliográficas.....	69
Apêndice A72

Lista de Figuras

Figura 1: Fluxo de <i>stream</i> (TOPIC, 2002).....	4
Figura 2: Arvore de tipos e maneiras de <i>streaming</i> (TOPIC, 2002).....	6
Figura 3: <i>Streaming</i> de Vídeo <i>On Demand</i> (KUROSE; ROSS, 2004).....	7
Figura 4: Comparação de <i>unicast</i> x <i>multicast</i> (KUMAR, 1997).....	10
Figura 5: Protocolos para <i>streaming</i>	12
Figura 6: Estrutura de página do <i>container</i> Ogg (PFEIFFER, 2003).....	19
Figura 7: Representação de um arquivo Matroska (MATROSKA, 2007).....	23
Figura 8: Flumotion - Tela inicial do <i>wizard</i>	32
Figura 9: Flumotion - Informações do gerenciador.....	32
Figura 10: Flumotion - Tela de <i>login</i> e senha.....	33
Figura 11: Flumotion - Tipo de origem da transmissão.....	34
Figura 12: Flumotion - Configuração do dispositivo de vídeo de entrada.....	35
Figura 13: Flumotion - Logotipo, texto por cima do vídeo (<i>Overlay</i>).....	35
Figura 14: Flumotion - Configuração da origem de áudio.....	36
Figura 15: Flumotion - Configuração do <i>container</i> a ser transmitido.....	37
Figura 16: Flumotion - Configuração do CODEC Theora.....	38
Figura 17: Flumotion - Configuração do CODEC Vorbis.....	38
Figura 18: Flumotion - tipo de <i>streaming</i> , opção de gravação.....	39
Figura 19: Flumotion - Ponto de montagem, acesso aos usuários.....	40
Figura 20: Flumotion - Escolha da licença de uso.....	41
Figura 21: Flumotion - Administração.....	42
Figura 22: Flumotion - Estado do componente.....	43
Figura 23: Flumotion - Estado intermediário.....	44
Figura 24: Flumotion - Estado final correto.....	44
Figura 25: Flumotion - Janela de depuração de erros.....	45

Figura 26: Icecast, <i>source client</i> e <i>listener client</i> , funcionamento (NIXON, 2004)	47
Figura 27: Icecast: Visão de rede (NIXON, 2004).....	48
Figura 28: VideoLAN <i>Streaming Solution</i> (LATRE, 2005).....	52
Figura 29: VLC gráfico, modos de trabalho.....	53
Figura 30: VLC - Opções avançadas de <i>streaming</i>	54
Figura 31: JRoar executando em um terminal.....	58
Figura 32: <i>applet</i> Cortado em execução.....	62

Lista de Tabelas

Tabela 1: Tipo de cabeçalho de páginas Ogg (PFEIFFER, 2003).....	20
--	----

Lista de Códigos e Listagens

Listagem 1: Flumotion - Seqüência de uso dos comandos.....	31
Listagem 2: Icecast - diretórios e arquivos.....	49
Listagem 3: Icecast - arquivo de configuração <i>icecast.xml</i>	50
Listagem 4: Compilação do servidor JRoar.....	57
Listagem 5: Cortado para <i>stream</i> sob demanda.....	60
Listagem 6: Cortado para <i>stream live</i>	61

Lista de Siglas

AAC (*Advanced Audio Coding*) – Uma melhoria do esquema de codificação de áudio MP3.

AC3 (*Dolby Digital*) – Nome de marketing para uma série de compressões de áudio da Dolby Laboratories.

ALSA (*Advanced Linux Sound Architecture*) – Provê funcionalidades de áudio e MIDI para o sistema operacional Linux.

API (*Application Programming Interface*) – Especificação de métodos do sistema operacional.

AVI (*Audio Video Interleave*) – Formato de arquivo da Microsoft

BSD (*Berkeley Software Distribution*) – Sistema Operacional Unix desenvolvido pela Universidade de Berkeley.

CDN (*Content Delivery Network*) – Rede planejada especificamente para distribuir conteúdo.

CODEC (*Compression/DECcompression*) – Algoritmo de compressão.

CPU (*Central Processing Unit*) – Conhecido como microprocessador, um dos principais componentes do computador.

DVD (*Digital Versatile Disc*) – Formato popular para distribuição de conteúdo digital.

EBML (*Extensible Binary Meta Language*) – Extensão binária simplificada do XML, com objetivo de armazenar e manipular dados na forma hierárquica com tamanho de campos variáveis.

FLAC (*Free Lossless Audio Codec*) – Formato de compressão *lossless* para áudio.

FPS (*Frames per Second*) – A medida do filme ou resolução temporal do vídeo.

GIF (*Graphics Interchange Format*) – Formato de compressão *lossless* para imagens gráficas.

GNU (*Gnu's NOT Unix*) – Acrônimo recursivo para “Gnu não é Unix”, sigla do projeto GNU, que tem como objetivo propagar o *software* livre.

GPL (*General Public License*) – Licença Pública que concretiza a filosofia defendida por Richard Stallman para o *software* livre.

GUI (*Graphical User Interface*) – Interface Gráfica do Usuário.

HD (*Hard Disk*) – Disco Rígido, memória secundária para armazenamento dos dados do usuário.

HTTP (*Hypertext Transfer Protocol*) – Protocolo que define como os elementos das páginas *web* são transportadas de um servidor para o navegador.

ID (*IDentification*) – Identificação única, normalmente em formato de número.

ID3 (*IDentify an MP3*) – *Tags* que identificam um arquivo de áudio MP3, elas

contém informações como autor, ano, título, álbum e faixa.

IP (*Internet Protocol*) – Método pelo qual pacotes de dados são transmitidos pela Internet.

JDK (*Java Development Kit*) – Pacote de desenvolvimento da linguagem *Java*.

JRE (*Java Runtime Environment*) – Pacote que contém a máquina virtual *Java*.

JPEG (*Joint Pictures Expert Group*) – Método padronizado de compressão de imagem.

LAN (*Local Area Network*) – Sigla que designa uma rede local interconectada.

MCF (*Media Container Format*) – Formato *container* de uso geral.

MIDI (*Musical Instrument Digital Interface*) – Música sintetizada, normalmente sem voz.

MMS (*Microsoft Media Streaming protocol*) – Protocolo derivado do *Real Time Protocol*.

MOV (formato *Quicktime*) – Formato *container* da Apple QuickTime.

MP3 (*MPEG audio layer-3*) – Formato de arquivo que contém áudio codificado MPEG layer-3.

MP4 (*MPEG-4 part 14*) – Formato *container* de áudio e vídeo da especificação MPEG-4.

MPEG (*Moving Picture Experts Group*) – Grupo que desenvolve padronizações para formatos multimídia / Extensão do arquivo multimídia do padrão MPEG-1.

OGM (*Ogg Media File*) – Formato de arquivo multimídia que possibilita cruzar CODEC Ogg Vorbis com outro CODEC não-livre.

OSS (*Open Sound System*) – Primeira tentativa de unificar a arquitetura de áudio digital para UNIX. OSS são um conjunto de *drivers* que fornecem uma API uniforme para UNIX.

P2P (*Peer to Peer*) – Um método de compartilhar dados digitais.

PCM (*Pulse-Code Modulation*) - Forma mais primitiva de armazenamento de áudio em formato digital.

PID (*Process ID*) – Identificação numérica única de cada processo.

QoS (*Quality of Service*) – Qualidade de serviço são normas, diretivas e estudo de dados para melhorar um serviço.

RTCP (*RTP Control Protocol*) – Protocolo usado com RTP para assegurar sincronização.

RTP (*Real-time Transport Protocol*) - Protocolo usado para transportar dados de *streaming*.

RTSP (*Real-Time Streaming Protocol*) – Protocolo de controle para *streaming*, possibilita habilitar pausa, avanço rápido, retrocesso, etc.

SMS (*Short Message Service*) – Serviço de mensagens curtas.

SSA (*Sub Station Alpha*) – Formato de legendas.

SSL (*Secure Socket Layer*) – Uma biblioteca que permite enviar dados encriptados pela Internet.

TCP (*Transmission Control Protocol*) – Protocolo de entrega de pacotes baseado no “melhor-esforço”.

TIFF (*Tagged Image File Format*) – Formato de arquivo de imagem.

UDP (*Unreliable Datagram Protocol*) – Protocolo simples, não confiável para entrega de dados.

URL (*Uniform Resource Locator*) – Endereço de páginas da Internet.

USB (*Universal Serial Bus*) – Padrão de conexão de periféricos.

V4L (*Video for Linux*) – Conjunto de API que possibilita acesso a dispositivos de vídeo e captura.

VCR (*VideoCassette Recorder*) – Vídeo Cassete.

VLC (*VideoLan Client*) – *Software* para envio/recebimento de *streaming*, bem como *player* de variados formatos.

VoD (*Video On Demand*) – Tipo de *streaming* para transmissão de multimídia onde os dados de origem estão pré-gravados.

WAV (*WAVEform audio format*) – Formato padrão da Microsoft para armazenamento de áudio digital.

XML (*Extensible Markup Language*) – Linguagem de marcação extensível, possibilita interoperabilidade e transporte de dados via Internet.

1. Introdução

O número de usuários que utilizam a Internet tem crescido ao longo dos anos. O relatório inicial do NIC.br (Núcleo de Informação e Coordenação do Ponto br) demonstra que tem aumentado o número de usuários que utilizam a Internet em casa, em LAN¹ *house*, em cafés e em escolas. A maioria destes usuários tem utilizado a rede mundial de computadores para fins pessoais/privativos e educacionais (CETIC.br, 2006).

Este crescimento da utilização da Internet tem promovido o desenvolvimento e o aprimoramento de diversos serviços nela inseridos, por exemplo **internet banking**: para realizar transações eletrônicas, pagamentos; **telefonía**: mensagens SMS (*Short Message Service*), interação com aparelhos móveis, VoIP²; **informativos**: notícias, artigos, pesquisas; **lazer**: jogos *online*, culinária, compra de ingressos de cinema e muitos outros.

Até pouco tempo atrás, quando se ouvia falar em vídeo pela Internet, tinha-se a idéia de espera e lentidão por causa do tamanho do arquivo de vídeo. Para visualizar *trailer*, vídeos musicais e animações gráficas, era necessário o *download* do conteúdo, antes de poder tocá-lo em seu *player* favorito. Este conceito tem mudado, pois gradativamente empresas e internautas têm utilizado a tecnologia de *streaming* para divulgação de vídeos, tanto pessoais quanto publicitários, como é o caso do YouTube, Google Vídeo, iFilm e Metacafe³.

Outras possibilidades são desenvolvidas na Internet com base nesta forma de transmissão de vídeo, por exemplo conferências, palestras, vídeo-aulas, biblioteca multimídia, treinamento a distância com vídeo/áudio aliada a uma ferramenta interativa (*chat*) e sistema de vigilância utilizando *hardware* como *webcam* ou câmera IP (*Internet Protocol*).

1 *Local Area Network*

2 VoIP – Voz sobre IP, tecnologia que permite ligação com baixo custo utilizando a Internet

3 <http://www.youtube.com>, <http://www.google.com>, <http://video.google.com>, <http://www.ifilm.com>, <http://www.metacafe.com>, respectivamente.

Os recursos tecnológicos criados e implementados no contexto da transmissão de vídeo por *streaming* são diversos, vão desde a captura por placas de captura, *webcam* e câmeras digitais, passando por *software* especializado de edição com tratamento de imagem e filtros, até a transmissão propriamente dita com servidores dedicados e dezenas de aplicativos que prometem a entrega do *stream*⁴. Muitos destes recursos tecnológicos são patenteados, ou seja, possuem algum tipo de licença proprietária sobre eles, que permite ou nega a utilização comercial e inclui ou não taxas de uso e comercialização. Algumas destas licenças permitem o uso pessoal, mas não o uso de produção e a maioria possui a característica de fechar o código fonte impedindo a mudança, estudo e melhoria por outros usuários.

A licença de uso de *software* livre, ao contrário do *software* proprietário, dá permissão e direitos livres sobre o programa, e dependendo do tipo de licença adotada, o usuário pode copiar, distribuir e modificar o conteúdo do *software* sem a cobrança de taxas. Estes fatores permitem o constante estudo e aprimoramento do *software* por muitos usuários espalhados pela Internet.

Este trabalho visa o estudo e implementação da tecnologia de *streaming* sobre a infraestrutura da Internet, além de demonstrar o uso de *software* livre em todas as etapas da transmissão. O presente trabalho apresenta ainda estudos de caso de alguns aplicativos de *streaming*, desde os materiais necessários para construção até sua utilização com exemplos práticos.

O trabalho está dividido em cinco capítulos, disponibilizados da seguinte forma. O Capítulo 2 apresenta uma introdução e uma definição do termo *streaming*, bem como sua classificação. No Capítulo 3, são mostrados protocolos, *containers*, CODEC livres e ferramentas de suporte. O Capítulo 4 é o centro do trabalho, nele estão demonstrados estudos de caso de *software* de transmissão e os resultados obtidos. Então, pelo Capítulo 5 discute-se as

⁴ Nome dado ao conteúdo do *streaming* que trafega na rede

considerações finais e conclusões. Ao final, no Apêndice, é acrescentado um Glossário que contém alguns termos estrangeiros utilizados durante a apresentação do trabalho.

2. Streaming

Streaming é a tecnologia que permite tocar conteúdos multimídia no momento em que se inicia o recebimento dos dados. Este conceito difere do *download* onde é preciso esperar o arquivo ser transferido para o computador para que então possa ser acessado. Meios de entretenimento e informação utilizam uma forma de *streaming* como televisão e rádio (TOPIC, 2002).

A filosofia do *streaming* é: “Enquanto você está vendo, o resto está chegando”, situação mostrada na Figura 1.

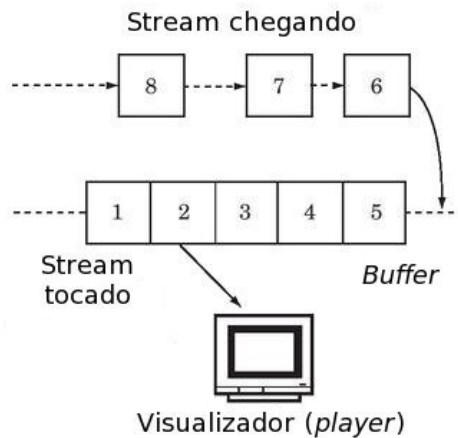


Figura 1: Fluxo de *stream* (TOPIC, 2002)

Alguns fatores podem causar atraso no recebimento dos *stream*: tráfego na rede, a largura de banda disponível, o número de roteadores no caminho, a qualidade e o tamanho do vídeo. Estes fatores, por sua vez, podem gerar parada repentinas, congelamento de imagem/áudio, áudio fora da seqüência do vídeo e lentidão no vídeo. Com o intuito de melhorar a recepção para o usuário, o *player* utiliza um recurso chamado *buffer* (Figura 1).

O *buffer* pode auxiliar na questão de pacotes perdidos, retendo os

pacotes que chegaram em ordem e aguardando os pacotes que necessitam ser retransmitidos. No início da transmissão de *streaming*, o *player* armazena alguns segundos do vídeo no *buffer* para depois tocá-lo enquanto recebe mais pacotes *streams*. A dificuldade nesta abordagem é saber o tamanho do *buffer* apropriado, pois *buffer* de tamanho menor pode ocasionar constantes paradas, e de tamanho maior assemelhar com o *download*.

A tecnologia de *streaming* mostra-se sob duas formas: sob demanda (*on demand*) e ao vivo (*live*). Sob demanda, os dados são armazenados no servidor por um longo período de tempo aguardando alguma requisição por parte do usuário para começar a transmitir a solicitação. Em *streaming* ao vivo, os dados são somente disponíveis em um período específico como por exemplo novelas e esportes (CANAN; RAABE, 2004).

O usuário no modelo sob demanda possui a liberdade de controlar individualmente o que está visualizando, ao contrário do que ocorre no modelo ao vivo, onde o controle é limitado por causa do compartilhamento do *streaming*. A Figura 2 ilustra os tipos de *streaming* possíveis, o modo de transmissão (*unicast*, *multicast*, unidirecional e bidirecional) e a experiência do usuário no recebimento dos dados (individual ou compartilhada).

A transmissão *streaming* de áudio/vídeo discutida neste trabalho baseia na estrutura da Internet, contudo, nada impede que os mesmos conceitos, com diferentes soluções, possam ser aplicadas em redes privadas (redes locais), fato comentado na Figura 2.

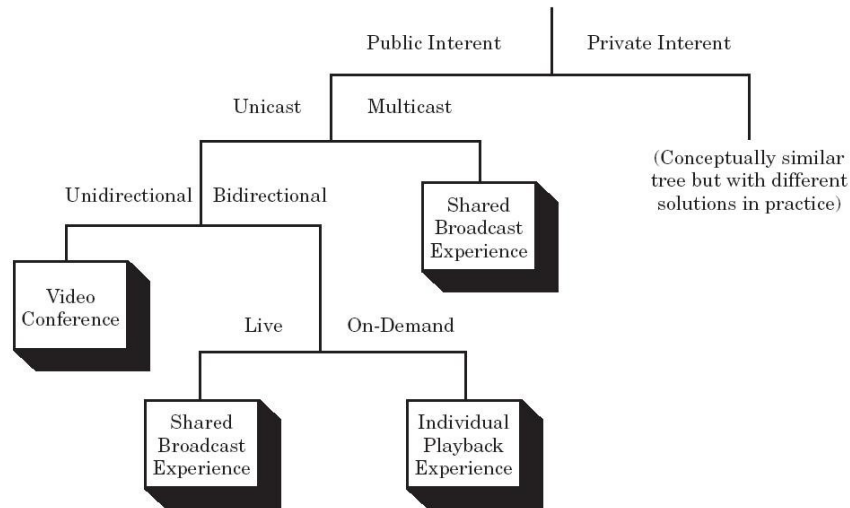


Figura 2: Arvore de tipos e maneiras de *streaming* (TOPIC, 2002)

2.1. Vídeo sob demanda (*Vídeo On Demand - VoD*)

Vídeo sob demanda é a forma de *streaming* onde a origem multimídia é pré-gravada e armazenada no disco em formato de arquivo. O cliente neste modelo pode iniciar o vídeo na hora que quiser, pausar, pressionar a barra de progressão, avançar (*forward*) e retroceder o vídeo (*rewind*), como um VCR (*VideoCassette Recorder*). A Figura 3 mostra o processo de transmissão de vídeo sob demanda, onde pode-se visualizar o acúmulo de dados (eixo Y) em relação ao tempo decorrido (eixo X). Os itens (1) e (2) da mesma figura, demonstram os passos dados pelo servidor para capturar um vídeo e posteriormente enviá-lo para a rede. A barra vertical assinalada na Figura 3 mostra o momento em que o vídeo é tocado em relação a recepção do vídeo transmitido.

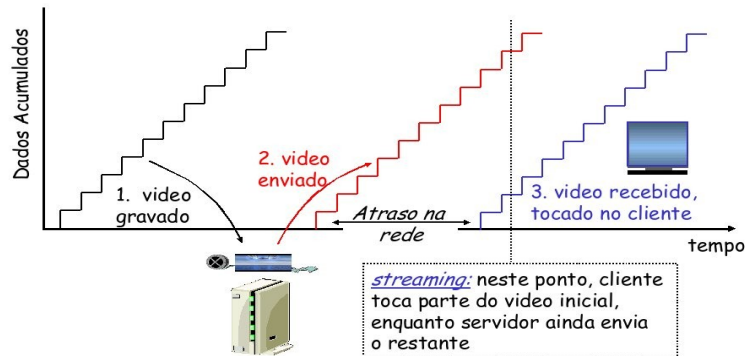


Figura 3: Streaming de Vídeo On Demand (KUROSE; ROSS, 2004)

É notado em pesquisas pela Internet que muitos servidores de *streaming*, sob a forma VoD, fazem uso do protocolo HTTP (*Hypertext Transfer Protocol*) sobre TCP (*Transmission Control Protocol*) para este tipo de transmissão. Isto porque, realizar *streaming* sob demanda, não requer o mesmo tempo real imprescindível que a forma ao vivo requer. Além disso a facilidade de dispor os arquivos em um servidor *web* e deixar para o TCP o controle de congestionamento tende a ser a melhor solução para este modo.

Os arquivos de vídeo podem ser armazenados em um servidor *web* (ex: Apache⁵) e o acesso ser disponibilizado utilizando, por exemplo, um endereço do tipo:

```
http://localhost/video/fisl07.ogg
```

Para tocar a mídia, basta utilizar um *player* que suporte o formato de arquivo armazenado e a possibilidade de tocar arquivos remotos, por exemplo: VLC (*VideoLan Client*), Mplayer e Xine. Exemplo de utilização com o *player* VLC é mostrado a seguir:

```
vlc http://streaming.noip.com/video/john-ripper.mkv
```

⁵ <http://www.apache.org/>

Na seção 4.5 é apresentada uma melhoria do acesso ao conteúdo multimídia utilizando um *player* em Java rodando no navegador do usuário chamado Cortado⁶ e retirando a necessidade de possuir um *player* com CODEC específico. Esta mesma tecnologia pode ser comparada ao uso de vídeos em Flash⁷, utilizados por *sites* como YouTube e Google Vídeo, que fazem uso dos recursos deste *plugin* proprietário para distribuir conteúdo multimídia.

2.2. Streaming ao vivo de Vídeos e Áudios (*live streaming*)

Este tipo de *streaming* permite ao usuário assistir vídeos ao vivo, ou seja, independente de quando o usuário começar a acessar o servidor de vídeos, poderá assistir somente ao que estiver passando no momento. Pode-se comparar esta forma com a transmissão de filmes pela televisão, o usuário não consegue assistir uma programação desde o início que começa às 17:00h, se liga a TV e sintoniza o canal às 18:00h.

Live streaming permite que transmissões ao vivo sejam enviadas pela Internet, utilizando equipamentos como *webcam*, placas de captura e filmadoras digitais. Este modelo não possibilita somente o tipo de transmissão em tempo real, pode-se enviar conteúdo de DVD (*Digital Versatile Disc*), vídeos armazenados, mas na forma de “ao vivo”, sem a possibilidade de retrocesso.

Este método possibilita a pausa e a continuidade do vídeo tocado, contudo tendo uma demora na sincronização do vídeo. O avanço neste caso é impossível. Exemplos deste tipo de *streaming* são inúmeros como *talk show* de rádio, notícias e reportagens pela Internet, palestras, transmissão de esportes e olimpíadas.

⁶ <http://www.flumotion.net/cortado/>

⁷ http://www.adobe.com/shockwave/download/index.cgi?Lang=BrazilianPortuguese&P1_Prod_Version=ShockwaveFlash

2.3. Unicast x Multicast

Streams podem ser enviados do servidor para o cliente por *unicast* ou *multicast*. No modo *unicast*, é estabelecida uma conexão ponto a ponto entre o servidor e o cliente, a maioria das redes operam neste modelo, usuários requisitam um arquivo e o servidor envia somente para estes o arquivo. A vantagem de utilizar este método é os computadores clientes poderem requisitar diretamente o *stream* do servidor e a desvantagem é cada cliente que conectar ao servidor receber um *stream* separado, o que ocasiona o aumento de consumo da largura de banda da rede.

IP *multicast* refere-se a técnica no qual um computador envia uma única cópia do *stream* pela rede e muitos computadores o recebem. Diferentemente do *broadcast*, roteadores podem controlar onde um *multicast* pode trafegar. A vantagem de *multicasting* é somente uma cópia do *stream* ser enviada através da rede, o que preserva a largura de banda. A desvantagem é ser *connectionless* (sem controle de conexão), clientes não tem controle sobre o *stream* recebido, eles apenas recebem.

Na Figura 4, é mostrada uma conexão de 28,8 *kbps* em modo *unicast* e *multicast*. Nota-se que no primeiro modo a largura de banda é dividida por cada um dos clientes, sendo decrescida a medida que novos clientes requisitam o *stream*. No modo *multicast*, a velocidade de conexão é a mesma para os clientes, mesmo que o número de clientes aumente.

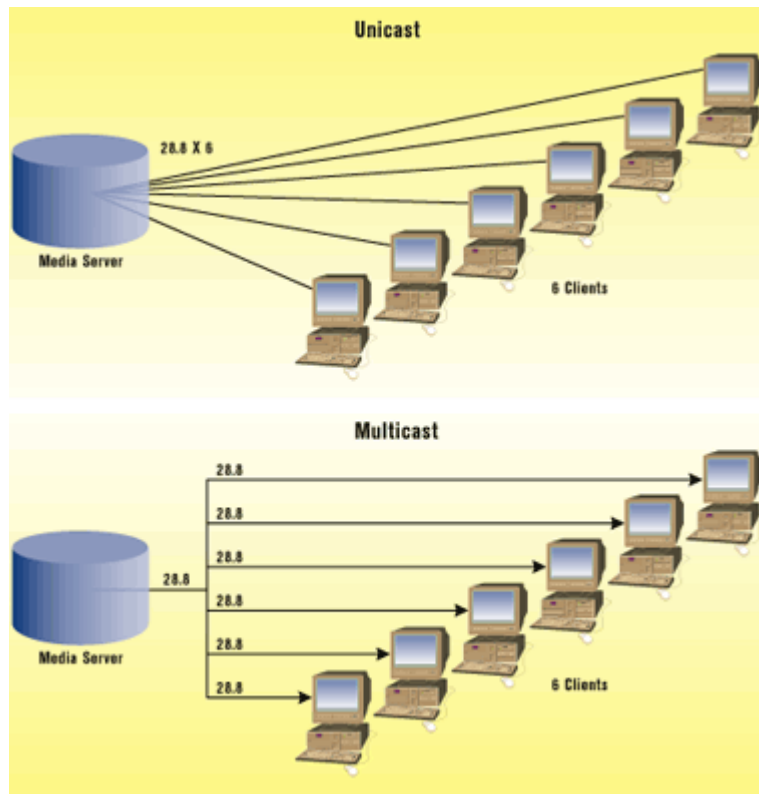


Figura 4: Comparação de *unicast* x *multicast* (KUMAR, 1997)

Os roteadores modernos suportam *multicast* e podem ser configurados para permiti-lo ou proibí-lo. Para que o *stream multicast* chegue no visualizador, os roteadores entre o servidor e o cliente precisam permitir *multicast*. Ele é proibido pela maioria dos roteadores públicos da Internet. Se isto não fosse uma verdade terrível, serviços de *streaming* seriam altamente disponíveis para qualquer um (UNREAL MEDIA, 2006).

A implementação de *multicast* possui maior funcionalidade em LAN onde o controle sobre os tipos de roteadores é maior. As implementações demonstradas no Capítulo 4 - Estudo de caso utilizaram tecnologia *unicast*.

3. Recursos Tecnológicos

Realizando buscas na Internet e em livros sobre o termo *streaming* o autor deparou com vários termos técnicos associados a este método, além de várias nomenclaturas que podem vir a confundir o leitor na implementação descrita neste trabalho. Este capítulo, portanto, tem por objetivo apresentar estas tecnologias e os termos a respeito de *streaming*.

3.1. Protocolos

Protocolos servem como padronizações da forma de transporte dos dados entre computadores em uma rede computacional, neles são definidos de que modo os dados trafegarão na rede. Os protocolos possibilitam o tráfego de variados formatos de dados, como músicas, vídeos, documentos, imagens, *chat*, *email* e conexão remota.

Esta seção aborda os principais protocolos de uso livre utilizados no transporte de dados *stream*, focando nas suas características, qualidade e/ou desvantagens específicas no uso de *streaming*. Não é foco deste trabalho a utilização de protocolos proprietários como MMS (*Microsoft Media Streaming protocol*⁸).

IP (*Internet Protocol*) é o protocolo mais básico usado sobre a Internet, provê as funções necessárias para entregar um pacote de *bits* de uma origem para um destino sobre um sistema de redes interconectado. Oferece um serviço de datagrama não confiável, ou seja, o pacote vem quase sem garantias; pode chegar desordenado e/ou duplicado ou pode ser perdido por inteiro (DARPA[1], 1981). É implementado na camada de rede da pilha TCP/IP, como mostra a Figura 5. Os outros protocolos de *streaming* e mecanismos residem sobre o IP.

⁸ <http://sdp.ppona.com/>

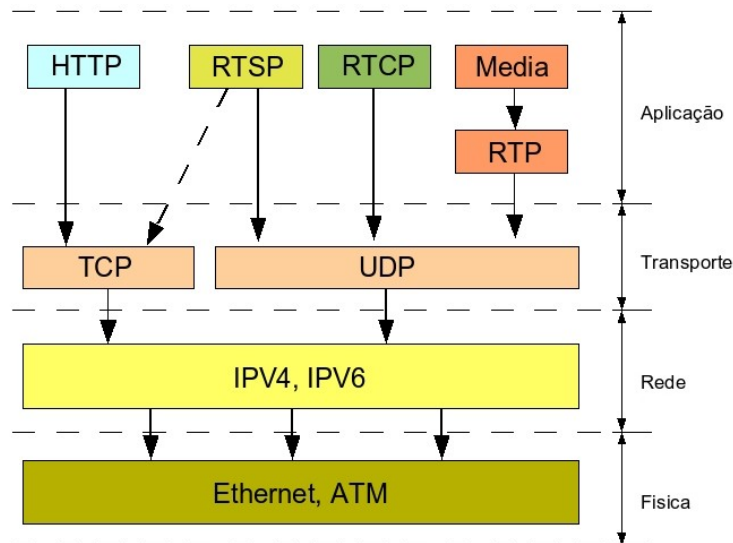


Figura 5: Protocolos para *streaming*

UDP (*Unreliable Datagram Protocol* – protocolo não-confiável) é a escolha usual de tipos de pacotes a serem enviados sobre IP quando a chegada dos pacotes no destino não é crítica, pode ser utilizado para vídeos *live*, onde o tempo é essencial e a preocupação sobre pacotes perdidos não é interessante. O protocolo UDP possui outras características como: não orientado a conexão, sem controle de fluxo, sem controle de congestionamento e tolerante a perdas, possui baixo *overhead* que é útil especialmente para RTP (*Real-time Transport Protocol*). O UDP é indicado para aplicações *streaming*, contudo muitas regras de *firewall* bloqueiam sua passagem (POSTEL, 1980). Trabalha na camada de transporte como mostra Figura 5.

TCP (*Transmission Control Protocol*) foi desenvolvido para prover transmissão robusta e sem erros nos dados, tendo outras características como: ponto a ponto, confiável, controle de congestionamento e fluxo, *buffer* no

transmissor e receptor, orientado a conexão, retransmissão de pacotes perdidos e ordenação. Ele opera no mesmo nível que o RTP, mas é usado nas comunicações de Internet onde o tempo não é preocupante ou conexões que requerem alta confiabilidade (DARPA[2], 1981). Pode ser utilizado para *streaming* ao vivo, contudo por suas características não é indicado. Muito utilizado como base para o HTTP em transmissões de *streaming* sob demanda.

HTTP (*Hypertext Transfer Protocol*) não foi criado para *streaming* de dados. Comunicação sobre o protocolo HTTP é *stateless*⁹. Usualmente respostas do HTTP utilizam *buffer*. A versão HTTP 1.1 adicionou suporte para *streaming* através do cabeçalho *keep-alive*, onde uma conexão permanece aberta até ser finalizada pelo cliente ou pelo servidor (FIELDING, 1999). Entre as características do HTTP destacam-se:

- Não tenta fazer *stream* em tempo real, para realizar esta característica a largura de banda da rede precisa ser maior que a taxa de dados do vídeo. Caso a largura de banda não possua tamanho suficiente, o usuário ainda sim poderá visualizar o vídeo com um certo atraso, já que o *player* do usuário terá que armazenar uma certa quantidade de dados antes de tocar o vídeo;
- A maioria dos esquemas de *firewall* e configurações de rede permitem o tráfego do protocolo HTTP sem alterações;
- Por rodar sobre TCP, o fator tempo é irrelevante e a sensibilidade na perda de pacotes é alta; por isso, o HTTP não é adequado para *streaming* ao vivo.

RTP (*Real-time Transport Protocol*) é um protocolo de transporte de tempo-real que fica sobre o UDP e foi originalmente desenvolvido para suportar conferência *multicast*-multimídia, mas ele pode ser usado em outras aplicações.

Como RTP provê serviços, como números de seqüência e *timestamps*

⁹ sem conhecimento do estado anterior, não guarda informação sobre uma requisição anterior.

etc. para aplicações multimídia, ele pode ser visto em uma subcamada da camada de transporte (SCHULZRINNE[1], 2003). O RTP possui as seguintes características:

- Pode ser usado para transmissão *live* e *multicast*;
- *Streaming* em tempo real permite visualizar vídeos longos ou transmissões contínuas sem armazenar mais do que alguns segundos de dados locais;
- Usando transmissão RTP sobre o controle RTSP (*Real-Time Streaming Protocol*), um usuário pode pular para qualquer ponto de um filme no servidor sem precisar baixar o material;
- Pode transmitir uma única trilha sobre RTP, visto que HTTP transmite somente todo o vídeo;
- Pode utilizar outros protocolos de transporte como TCP.

RTCP (*RTP Control Protocol*) fornece, em conjunto com o RTP, um canal de controle útil para monitorização de qualidade. Os servidores enviam pacotes RTCP periodicamente para que o servidor saiba a qualidade do *stream* que recebe. O servidor pode baixar a qualidade do *stream*, se necessário. Este protocolo possibilita a geração de relatórios periódicos sobre o estado do *stream* para a camada de aplicação que inclui estatísticas sobre o número de pacotes enviados, número de pacotes perdidos e espera entre intervalos de chegada (SCHULZRINNE[1], 2003).

RTSP (*Real-Time Streaming Protocol*) é um protocolo no nível de aplicação que usa os elementos de baixo-nível do RTP para gerenciar múltiplos *streams* que precisam ser combinados para criar uma seção multimídia. Estes *streams* podem ser originados de muitos lugares, inclusive de locais geográficos diferentes. Ele estabelece e controla um único *stream* ou vários *streams* de conteúdo contínuo (SCHULZRINNE[2], 1998). O RTSP possui as seguintes características:

- Provê meios para escolha de canais de entrega como UDP, *multicast* UDP e TCP;
- Atua como uma "rede de controle remoto" para servidores multimídia;
- Sua operação não depende do mecanismo de transporte usado para carregar os dados;
- Possui sintaxe e operação similar ao HTTP versão 1.1;
- Disponibiliza funcionalidades de um vídeo cassete como pausa, retrocesso, avanço e posicionamento.

Em termos gerais, RTP é um protocolo de transporte para entrega de dados em tempo-real, incluindo *streaming* de áudio e vídeo. RTCP é uma parte do RTP que auxilia na sincronização e no gerenciamento de QoS (*Quality of Service*). RTSP é um protocolo de controle que inicializa e direciona entrega de *streaming* multimídia dos servidores. RTSP não entrega dados, embora a sua conexão possa ser utilizada como túnel para tráfego RTP, facilitando seu uso com *firewall* e outros dispositivos de rede.

RTP e RTSP podem ser usados juntos em muitos sistemas, contudo cada protocolo pode ser utilizado sem o outro. A documentação do protocolo RTSP¹⁰ contém uma seção que apresenta o uso do RTP com RTSP.

3.2. Diferenças entre *Container* e CODEC

Esta seção aborda a diferença entre *container* e CODEC, um conceito que confunde muitos usuários, e mostra a definição de *container* exemplificando com alguns *containers* de uso livre, bem como para o CODEC.

¹⁰ <http://www.ietf.org/rfc/rfc2326.txt>

3.2.1. CODEC

São programas que codificam (CO-dec) um dado de origem para um formato, na maioria das vezes comprimido, e/ou decodificam (co-DEC) um dado codificado de volta para seu estado original. O CODEC foi desenvolvido principalmente para compressão e transporte de dados, pode ser classificado em dois grupos (RICHARDSON, 2003):

*lossy*¹¹ – método de compressão que descarta dados para comprimir melhor do que o normalmente possível, utiliza taxas de compressão muito altas gerando arquivos menores que os originais. Por causa das altas taxas, o método *lossy* gera perdas no arquivo final, contudo não são identificadas facilmente pelo usuário. Exemplo: compressões JPEG (*Joint Pictures Expert Group*), Vorbis, Theora, XviD, Speex e MP3 (*MPEG audio layer-3*) (RICHARDSON, 2003);

*lossless*¹² – método de compressão que preserva os dados originais, não descartando nenhuma informação na sua utilização. Zip é um formato de compressão *lossless* comum de propósito geral; FLAC (*Free Lossless Audio Codec*) é um formato de compressão *lossless* que foi desenvolvido especificamente para áudio, outros exemplos são RealPlayer, GIF (*Graphics Interchange Format*), TIFF (*Tagged Image File Format*) e Huffiyuv (RICHARDSON, 2003).

O conjunto de CODEC de vídeo/áudio/legenda podem ser agrupados em formatos conhecidos como *container*. Estes não possuem algoritmo de compressão, sendo sua funcionalidade a de prover mecanismos de distribuição de conteúdo multimídia.

O *container* Matroska encapsula praticamente todos os CODEC de vídeo/áudio e legenda. Exemplos desta habilidade encontra-se no *site matroska*

11 <http://www.bobulous.org.uk/misc/audioFormats.html>

12 <http://www.bobulous.org.uk/misc/audioFormats.html>

Sample Page¹³ onde dois *trailers* de filmes *.mkv* são mostrados: um com CODEC de vídeo RealPlayer, áudio AAC (*Advanced Audio Coding*) e 11 legendas SSA (*Sub Station Alpha*) e o outro arquivo codificado com vídeo XviD, áudio AAC e 11 legendas SSA. No mesmo *site*, encontra-se outro vídeo codificado com CODEC de vídeo XviD e áudio Vorbis.

No caso do *container* Ogg, é possível encapsular CODEC da própria empresa Xiph como Speex, Vorbis, Theora, FLAC e Writ, não se tem informações a respeito de outro CODEC encapsulado por ele.

3.2.2. *Container*¹⁴

Um *container* é um formato desenvolvido para conter uma combinação de dados de vídeo/áudio e algumas vezes algo mais como legendas, de tal modo que a aplicação *player* possa lê-lo confiantemente e tocá-lo corretamente, isso significa com áudio e vídeo sincronizados. Exemplos de *containers*: Ogg, Matroska, AVI (*Audio Video Interleave*), MP4 (*MPEG-4 part 14*) e MOV (formato *Quicktime*).

Ogg¹⁵ - é o nome do formato de *container* aberto e livre pertencente a Xiph.org. Esta fundação sem fins lucrativos dedica-se a proteção do livre uso de padrões abertos de multimídia na Internet, combatendo os formatos de particulares. Xiph.org têm o objetivo de desenvolver protocolos, *software* e padrões livres que possam ser utilizados por todos, retirando a preocupação de utilizar recurso tecnológico patentado.

O desenvolvimento do *container* Ogg possibilitou aos usuários a utilização de um formato não proprietário que permite encapsular áudio, vídeo e

13 <http://www.matroska.org/samples>

14 http://www.divxland.org/container_formats.php

15 <http://xiph.org/ogg/>

metadados. Ogg pode ser utilizado para realizar *streaming* eficiente bem como ser uma alternativa para os formatos AVI, MP4 e MOV quando encapsulado com áudio e vídeo.

Freqüentemente, o termo Ogg é designado ao formato de arquivo de áudio Ogg Vorbis, que é um áudio *vorbis* codificado no *container* Ogg. Outro CODEC da Xiph freqüentemente encapsulado no Ogg é o CODEC de vídeo Theora e o formato de compressão de áudio para fala humana Speex¹⁶.

O formato *bitstream*¹⁷ do Ogg é livremente disponível e livre para ser reimplementado em *software*. Este formato consiste de pedaços de dados cada um chamado de Ogg *page* e cada uma destas páginas começa com uma *string* "OggS" para identificar o arquivo como sendo de formato Ogg (Figura 6).

Um número serial e um número de página no cabeçalho identifica cada página como parte de uma série de páginas criando um *bitstream*. Múltiplos *bitstream* podem ser multiplexados (agrupados) em um arquivo onde as páginas de cada *bitstream* são ordenadas pelo tempo de procura dos dados contidos. *Bitstream* podem ainda ser acrescentados a arquivos existentes, um processo conhecido como *chaining*¹⁸, para causar uma decodificação em seqüência dos *bitstream*.

Uma biblioteca chamada *libogg*¹⁹, de licença BSD²⁰(*Berkeley Software Distribution*), é disponível na Internet para realizar codificação (*encoding*) e decodificação (*decoding*) dos dados de *stream* Ogg.

16 Respectivamente <http://www.vorbis.com/>, <http://www.theora.org/> e <http://www.speex.org/>

17 Fluxo de *bits* ordenado representando um dado.

18 corrente, correnteza, encadeamento

19 <http://xiph.org/downloads/>

20 <http://www.opensource.org/licenses/bsd-license.html>

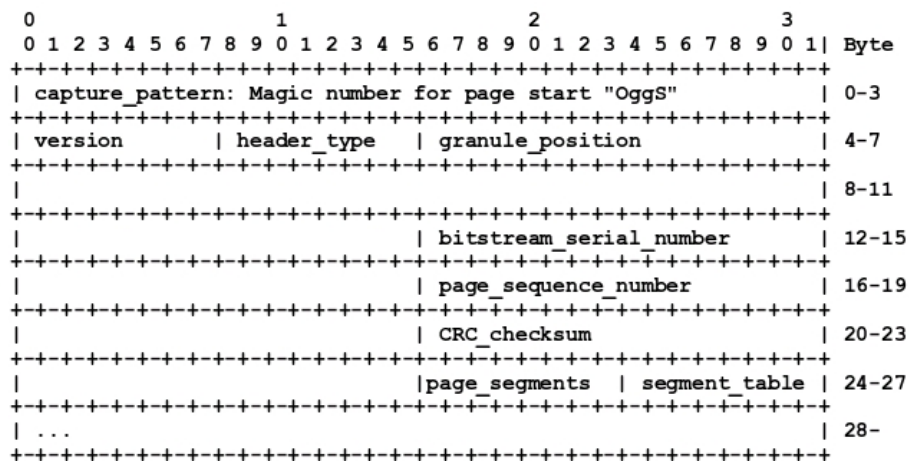


Figura 6: Estrutura de página do *container* Ogg (PFEIFFER, 2003)

A estrutura de uma página Ogg pode ser visualizada pela Figura 6, tendo seus campos descritos como segue (PFEIFFER, 2003) :

- **capture_pattern - 32 bits (4 Byte)** - O *capture pattern* ou código de sincronismo é um número mágico usado para garantir sincronização quando repassa arquivos *ogg*; Cada página começa com uma seqüência de 4 *bytes* contendo o texto *OggS*. Isto ajuda no ressincronismo e, nos casos onde dados são perdidos ou estão corrompidos, é utilizado para checagem da consistência de cada página antes de ser iniciado a análise da estrutura da página. O caracter e sua respectiva representação em hexadecimal é mostrado a seguir:

'O' - 0x4f
 'g' - 0x67
 'g' - 0x67
 'S' - 0x53

Este campo fornece ajuda para um decodificador encontrar os limites da

página e obter a sincronização depois de repassar um *stream* corrompido. Uma vez que o *capture pattern* é encontrado, o decodificador verifica o sincronismo e a integridade da página computando e comparando o *checksum*.

- **version - 8 bits** - Este campo indica a versão do formato *bitstream ogg*, permitido para futura expansão. Atualmente, é exigido ser 0.
- **header_type - 8 bits** - Este é um campo de opções, o qual indica o tipo de página que segue. Como descrito pela Tabela 1.

Bit	Setado (S) Não Setado (NS)	Tipo de Página
0x01	S	a página contém dados de um pacote continuado de páginas anteriores
	NS	página contém um pacote novo
0x02	S	esta é a primeira página de um <i>logical bitstream</i> , (BOS - <i>Beginning of Stream</i>)
	NS	esta página não é a primeira página
0x04	S	esta é a última página de um <i>logical bitstream</i> , (EOS - <i>End of Stream</i>)
	NS	esta página não é a última página

Tabela 1: Tipo de cabeçalho de páginas Ogg (PFEIFFER, 2003)

- **granule_position - 64 bits** - Marcador de tempo e posição em arquivos *ogg*. Ele é um valor abstrato, determinado pelo CODEC. Pode conter, por exemplo, para áudio *streams*, o número total de amostras codificadas de PCM (*Pulse-Code Modulation*²¹) depois que foi incluído todos os *frames* acabados nesta página. Para vídeo *stream*, ele pode conter o número total de *frames* codificados depois desta página.
- **bitstream_serial_number - 32 bits** - Número serial que identifica uma página pertencente a um particular *logical bitstream*. Cada *logical bitstream* em um

²¹ Forma mais primitiva de armazenamento de áudio em formato digital

arquivo tem um único valor e este campo permite implementações de entrega de páginas para o decodificador apropriado.

- **page_sequence_number - 32 bits** - Contém a seqüência de números das páginas, para que o decoder possa identificar páginas perdidas. Esta seqüência de números é aumentada em cada *logical bitstream* separado.
- **CRC_checksum - 32 bits** - Provê uma checagem de dados em toda página, realizada com o campo *checksum* setado em 0. Isto permite verificar os dados que não tenham sido corrompidos desde a criação. Páginas com falha no *checksum* devem ser descartadas.
- **page_segments - 8 bits** - Fornece o número de entradas de segmentos codificados na tabela de segmento.
- **segment_table** - A tabela de segmento é um vetor de 8 *bit* que indica o tamanho de cada segmento dentro do corpo da página. O número de segmentos é determinado pelo campo precedente *page_segments*. O tamanho de cada segmento está entre 0 e 255 *bytes*.

Matroska²² - É um projeto ambicioso de padrões abertos que permite o livre uso do formato, bem como das especificações que definem o seu *bitstream*. O *container* Matroska tem por objetivo criar e documentar um formato moderno, flexível, multiplataforma e livre, tendo o desejo de competir com formatos proprietários como MOV da Quicktime e AVI da Microsoft. O projeto Matroska é derivado do MCF (*Media Container Format*²³) e é baseado no EBML (*Extensible Binary Meta Language*) um binário derivado do XML (*Extensible Markup Language*) que gera possibilidade de ser extensível. Outras características incluem (MATROSKA, 2007):

- rápido posicionamento no arquivo;
- alta recuperação de erros;

²² <http://www.matroska.org/>

²³ <http://mcf.sourceforge.net/> último *post* do *site* data de 6 de Setembro de 2003.

- suporte a capítulos;
- legendas selecionáveis;
- seleção de *stream* de áudio;
- suporte a *menu* como DVD.

O formato *container* Matroska suporta a inclusão de vídeo, áudio, legendas, fontes e imagens. O nome é grafia romanizada de *матрёшка*²⁴, ou *matrioska* - bonecas típicas russas, no qual uma boneca grande contém outra menor, que contém outra, e assim sucessivamente. Analogamente, o formato do Matroska permite conter dados resultantes de diferentes tipos de codificações de vídeo e áudio (CODEC).

Os tipos de arquivos utilizados por este formato são: *.mks* para legenda, *.mka* para áudio e *.mkv* para vídeo e formato de *container*, com trilhas de legenda e áudio. Uma simples representação de como é formado um arquivo Matroska é mostrada pela Figura 7 (MATROSKA, 2007), onde são listados os seguintes campos:

- **header** – contém informações a respeito de qual versão de EBML foi utilizada para criação deste arquivo e qual é o tipo de arquivo, no caso Matroska;
- **metaseek information** – é um índice que possibilita encontrar os outros grupos de nível 1 como trilhas, capítulos, *tags* e anexos. Tecnicamente este campo não é exigido, mas sem ele deve-se percorrer o arquivo para encontrar uma dada informação;

²⁴ <http://www.answers.com/topic/matryoshka>



Figura 7: Representação de um arquivo Matroska (MATROSKA, 2007)

- ***segment information*** – contém informação básica a respeito do arquivo inclusive o título, ID (*IDentification*) único para ser identificado ao redor do mundo e, se for parte de uma série de arquivos, o ID do próximo arquivo;
- ***track*** – seção que retêm todas as informações a respeito de cada trilha no arquivo Matroska, podendo conter o tipo da trilha (vídeo, áudio ou legenda) e opções específicas dependendo do tipo da trilha inserido, por exemplo resolução e *bitrate* para vídeo;
- ***chapters*** – listam os capítulos do arquivo que estabelecem os pontos de saltos, tanto de vídeo como de áudio;
- ***clusters*** – seção que contém os *frames* de vídeo e áudio para cada trilha;
- ***cueing data*** – seção que contém os *cues*. *Cues* são índices para cada uma das trilhas, muito parecido com o *metaseek*, mas é usado para procurar um tempo específico quando é tocado o arquivo;
- ***attachment*** – seção que permite anexar qualquer tipo de conteúdo ao arquivo Matroska, por exemplo figuras, páginas da Internet, programas e o CODEC necessário;

- **tagging** – contém os *tags* utilizados no arquivo relacionadas a ele ou as trilhas. *Tags* são como os ID3²⁵ *tags* encontrados em arquivos MP3 com a informação de autor, ano de publicação, título, diretor, etc.

O projeto Matroska está em constante desenvolvimento, contudo seu formato é utilizado em alguns aplicativos espalhados pela Internet. Futuramente, seu formato será combinado com um CODEC API (*Application Programming Interface*) aberto para formar um *framework* livre e aberto de multimídia (MATROSKA, 2007).

Ogg Vs Matroska - Ogg e Matroska tem diferentes filosofias. Ogg tem foco na alta compatibilidade e utilização, tornando ideal para distribuição de conteúdo, bem como *streaming*, possui uma sobrecarga baixa e limitada. Matroska, por outro lado, tem como foco competir com MOV do Quicktime e objetivos de sobrepor AVI e VfW/Dshow²⁶ como *container* de uso geral. Matroska oferece acesso hierárquico, muito parecido com o formato de arquivos da Apple's Quicktime. O *container* Matroska se comparado com Ogg, possui posicionamento mais rápido além de ser mais complexo. Pode-se usar características de DVD como *menu*, capítulos e seleção de legendas em Matroska (MATROSKA, 2007).

Container OGM²⁷ (Ogg Media File) - Este *container* é um filtro *DirectShow* para Ogg, criado por Tobias Waldvogel²⁸, permite o encapsulamento de CODEC não livres, por exemplo DivX com áudio Vorbis e possibilita tocar arquivos *ogg* com um *player* do tipo Windows Media Player. Algumas características incluem:

25 *IDentify an MP3*

26 Video for Windows / DirectShow são utilizados para manipulação de multimídia em WinXX

27 <http://www.answers.com/topic/ogg-media>

28 Tobias foi integrado a fundação Xiph.org

- suporte a capítulo;
- múltiplas trilhas de legendas;
- múltiplas trilhas de áudio de vários formatos como: MP3, AC3 (*Dolby Digital*), Vorbis, WAV (*WAVEform audio format*).

Freqüentemente, o OGM carrega vídeo codificado em formato Xvid e áudio em Vorbis ou AC3 (*Dolby Digital*).

3.3. Ferramentas de Suporte

Esta seção tem por objetivo abordar algumas ferramentas auxiliares para criação, implementação, utilização e, porventura, transmissão de *streams*.

Gstreamer²⁹ (*Open Source Framework Multimedia*) - É um *framework* livre para desenvolvimento de aplicações multimídia, oferece base de suporte de *streaming* para o Flumotion e outras aplicações. Na instalação deste *software*, é disponibilizado comandos para as mais variadas funções como: captura de vídeo (utilizando V4L³⁰), transcodificação e *source client*³¹. É sugerido pelo autor a instalação de todos os módulos do *gstreamer*. Nas próximas linhas são mostrados exemplos práticos da utilização dos *plugins* internos deste *framework*.

O exemplo a seguir, utiliza o comando *gst-launch* para visualizar um dispositivo conectado ao */dev/video0*, com resolução de saída de 176x144:

```
gst-launch-0.10 v4lsrc device=/dev/video0 ! "video/x-raw-yuv",
width=176,height=144 ! xvimagesink
```

Um exemplo de captura de *webcam*, gravando seu conteúdo em um arquivo no formato Ogg Theora é mostrado na linha seguinte:

```
gst-launch-0.10 v4lsrc device=/dev/video0 ! "video/x-raw-yuv",
width=320,height=240 ! ffmpegcolorspace ! theoraenc bitrate=23 !
```

²⁹ <http://www.gstreamer.net/>

³⁰ *Video for Linux*

³¹ ver Icecast

```
oggmux name=mux ! filesink location=meuarquivo.ogg
```

Pode-se utilizar o *gstreamer* para criar um *source client* que encaminha o vídeo capturado para um servidor do tipo Icecast ou JRoar. Este procedimento é mostrado na linha a seguir:

```
gst-launch-0.10 v4lsrc device=/dev/video0 ! "video/x-raw-yuv",  
width=176,height=144 ! ffmpegcolorspace ! theoraenc bitrate=23 !  
oggmux name=mux ! shout2send ip=127.0.0.1 port=8000  
password=hackme mount=webcam.ogg { mux. }
```

Transcode³² - Conjunto de utilitários de linha de comando para transcodificação de CODEC vídeo/áudio e para conversão de diferentes *container*. Este *software* possui suporte para codificação e decodificação Ogg Vorbis.

Para capturar por exemplo, um áudio de DVD do décimo capítulo do primeiro título e salvá-lo em formato Ogg Vorbis, deve-se fazer:

```
transcode -i /dev/dvd -x dvd -T 1,10,1 -a 0 -y ogg -m track10.ogg
```

MediaCoder³³ - *Software* de transcodificação para sistema operacional Windows, suporta variados formatos como Vorbis, Theora, OGM e Speex.

FFmpeg³⁴ - *Software* muito conhecido por converter vários formatos e possibilitar captura de áudio/vídeo. Este programa suporta CODEC Vorbis, Theora e decodificação de arquivos Matroska.

Fffmpeg2theora³⁵ - *ffmpeg2theora* é um *software* simples para criação de arquivos Ogg Theora. Exemplo de conversão de um arquivo de vídeo no formato AVI para Ogg Theora, é mostrado a seguir:

32 <http://www.transcoding.org/cgi-bin/transcode>

33 <http://mediacoder.sourceforge.net/index.htm>

34 <http://ffmpeg.mplayerhq.hu/>

35 <http://www.v2v.cc/~jffmpeg2theora/download.html>

```
ffmpeg2theora arquivo_video.avi
```

Alterando um pouco o exemplo anterior, reduzindo o *bitrate* para 70 e alterando o nome de saída surge a seguinte linha de comando:

```
ffmpeg2theora arquivo_video.avi -V 70 -o novo_nome.ogg
```

oggfwd³⁶ - Utilitário para receber um *pipe ogg* e enviá-lo para um servidor que suporte *libshout* como *icecast* e *JRoar*. O *oggfwd* é utilizado em combinação com o *ffmpeg2theora* para criação de *source client*.

O exemplo a seguir demonstra a conversão do formato MPEG para Ogg transmitindo ao mesmo tempo para um servidor:

```
ffmpeg2theora /Video/arquivo_video.mpg --inputfps 15 -x 160 -y 120 -o - |  
oggfwd 127.0.0.1 8000 senha /theora.ogg
```

Caso seja possível a captura de um dispositivo, tem-se a linha descrita a seguir, sendo enviado a captura para um servidor no formato *ogg*:

```
ffmpeg2theora --v4l /dev/video0 --inputfps 15 -x 160 -y 120 -o - | oggfwd  
127.0.0.1 8000 senha /theora.ogg
```

mencoder³⁷ - Ferramenta de linha de comando para codificação de vídeos. Distribuído juntamente com o *MPlayer*, o *mencoder* permite converter variados tipos de formatos, inclusive Ogg Vorbis.

MKVToolnix³⁸ - Ferramentas para visualizar, extrair e criar arquivos Matroska. Para obter informações a respeito de um arquivo *.mkv* de nome *video_teste.mkv* faz-se como mostrado a seguir:

```
mkvinfo video_teste.mkv
```

Um exemplo de criação de arquivo *.mkv* agrupando uma faixa de vídeo AVI e uma faixa de áudio Vorbis é mostrado na linha seguinte:

```
mkvmerge -o video_final.mkv video_original.avi som_ambiente.ogg
```

36 <http://www.v2v.cc/~jffmpeg2theora/oggfwd/>

37 <http://www.mplayerhq.hu/design7/dload.html>

38 <http://www.bunkus.org/videotools/mkvtoolnix/index.html>

Lembrando que o Matroska suporta a criação de várias faixas de áudio para o mesmo vídeo, bem como várias faixas de legenda.

Vorbis Tools³⁹ - Conjunto de ferramentas que possibilita criar e manipular arquivos Ogg Vorbis. Um exemplo de codificação para um arquivo *vorbis* com qualidade 4 é mostrado na linha seguinte:

```
oggenc -q4 -o som_saida.ogg som_entrada.wav
```

OGMtools⁴⁰ - Conjunto de utilitários para manipular arquivos *.ogm*, que não suporta Theora. Um exemplo de uso encapsulando um arquivo de vídeo AVI e áudio Vorbis é apresentada a seguir:

```
ogmmerge -o MyMovie-with-sound.ogm MyMovie.avi MyMovie.ogg
```

VLC⁴¹ - O VLC, apesar de ser conhecido mais pela utilização como *player*, permite transcodificação no momento do envio para o cliente (*on-the-fly*) e possibilita conversão de CODEC e armazenamento do *stream* em meio digital.

Libshout⁴² - Biblioteca para uso e criação de *source client*. Estes possibilitam encadear fontes a servidores como: Icecast e JRoar.

FreeJ⁴³ - *Software* livre de mixagem que permite a manipulação em tempo real de vídeos, podendo utilizar sobreposições, efeitos e filtros. FreeJ possibilita a captura de dispositivos e o encaminhamento para servidores do tipo Icecast (em formato *ogg*).

39 <http://www.bebits.com/app/1058>

40 <http://www.bunkus.org/videotools/ogntools/>

41 <http://www.videolan.org/vlc/>

42 <http://www.icecast.org/download.php>

43 <http://freej.org/>

4. Estudo de Casos de Aplicação de *streaming*

Neste capítulo, será estudado, de forma prática, os aplicativos Flumotion, Icecast, VLC, JRoar e Cortado que possibilitam a transmissão e/ou a recepção de *streams*. Essas tecnologias de *streaming* foram escolhidos pela sua popularidade em fóruns e por serem *software* livre utilizando formatos livres. A plataforma adotada para a implementação foi a distribuição Slackware versão 11.0, com *kernel* 2.6.17, GCC 3.4.6, ambiente gráfico X (X.Org) com gerenciador Fluxbox 1.0r2.

Para resolução de dependências, foi utilizada a distribuição *dropline* GNOME⁴⁴, que é uma distribuição GNOME para o Slackware. Esta distribuição permite a instalação facilitada do GStreamer, bem como a de algumas bibliotecas utilizadas pelos aplicativos descritos neste capítulo.

A infraestrutura de testes utilizada foram três computadores interligados em uma LAN, sendo dois cliente-servidor Linux, com *webcam* cada um, e um cliente Windows, rodando *player* VLC e Cortado. Também foram utilizados, como clientes, microcomputadores externos conectados a um dos servidores via Internet.

4.1. Flumotion

Flumotion Streaming Server⁴⁵ é um servidor de *streaming live*, desenvolvido pela Fluendo⁴⁶. A versão básica do produto é livre, distribuído sob a licença GPL, e realiza *stream* nos formatos livres Ogg/Vorbis, Theora, Speex. Na sua versão avançada, para uso comercial, possui mais recursos e pode trabalhar com formatos proprietários como o MPEG.

⁴⁴ <http://www.droplinegnome.net/>

⁴⁵ <http://www.flumotion.net/>

⁴⁶ <http://www.fluendo.com/products.php?product=flumotion>

Este servidor de *streaming* é baseado em duas tecnologias, GStreamer e Twisted⁴⁷. A tecnologia GStreamer é um *framework* para aplicações multimídia, podendo ser utilizada para construir gráficos, manipular áudio/vídeo, realizar processamento de vídeo, disponibilizar a utilização de filtros/*plugins*, fornecer recursos de *streaming*, etc. A tecnologia Twisted, escrita em Python⁴⁸, permite acrescentar funções não diretamente ligada a *stream*, mas relacionada a eles, como configuração de seção, conexão do cliente e autenticação, provê camada de abstração de rede, entre outros (STICHELE; SCHALLER, 2005).

Flumotion permite obter vídeos através de três tipos de *hardware*: *webcam*, placa de captura de TV e aparelhos com conexão *firewire*. Em relação a captura de áudio, tem-se: placas de som e aparelhos com conexão *firewire*.

A transmissão de *stream* pode ser realizada somente sobre o par de protocolos HTTP/TCP, podendo ser utilizado o protocolo SSL (*Secure Sockets Layer*⁴⁹) para conexão segura.

A instalação do servidor Flumotion (versão utilizada 0.3.1) requer a prévia instalação do Twisted (versão utilizada 1.3.0 – via *dropline* GNOME), GStreamer (versão utilizada 0.10.11) e PIL (Python Imaging Library – pacote utilizado Imaging-1.1.6⁵⁰).

A utilização do servidor é realizada com o uso de 3 comandos gerados na instalação (STICHELE; SCHALLER, 2005), são eles:

- *flumotion-manager* – Gera um processo *manager* que gerenciará o trabalho dos processos *worker* e os componentes nos níveis mais baixos;
- *flumotion-worker* – Gera um processo *worker* que conecta ao *manager* e espera o servidor lhe dar comandos, tipicamente ele inicia os

47 <http://twistedmatrix.com/trac/>

48 <http://www.python.org/>

49 <http://www.verisign.com/ssl/ssl-information-center/how-ssl-security-works/index.html>

50 <http://www.pythonware.com/products/pl/>

componentes;

- *flumotion-admin* – Inicia uma interface gráfica para administração do servidor (a interface modo texto não foi bem sucedida, *flumotion-admin-text*).

A Listagem 1 mostra a seqüência padrão de utilização do Flumotion, na máquina local, sem segurança (SSL), e com os processos *manager* e *worker* rodando em *background*. Tem-se os seguintes parâmetros: -v (mostrar detalhes), -T protocolo (TCP/SSL), -u usuário (*login* a ser utilizado para conectar ao servidor), -p senha (senha do usuário), *caminho/para/o/planet.xml* é a localização do arquivo *planet.xml* o qual contém a configuração do servidor, normalmente localizado dentro do código-fonte (*flumotion-0.3.1/conf/managers/default/planet.xml*).

Listagem 1: Flumotion - Seqüência de uso dos comandos

```
# flumotion-manager -v -T tcp caminho/para/o/planet.xml &
# flumotion-worker -v -T tcp -u user -p test &
# flumotion-admin -v
```

O último comando na Listagem 1 (*flumotion-admin -v*) inicializa a interface gráfica para configuração, como mostrado na Figura 8. Nesta figura, tem-se as seguintes opções: “abrir conexões existentes” - para conectar em um servidor iniciado e “conectar a um gerenciador aberto” - para que seja utilizado o gerenciador carregado pelo comando *flumotion-manager*. É escolhida a segunda opção.

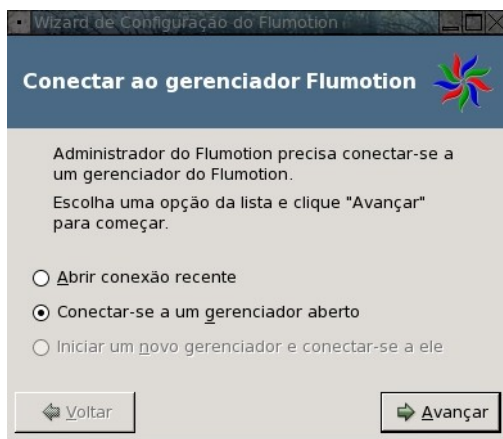


Figura 8: Flumotion - Tela inicial do *wizard*

Na tela seguinte, Figura 9, é informado o local onde é executado o gerenciador e se utilizará conexão segura.

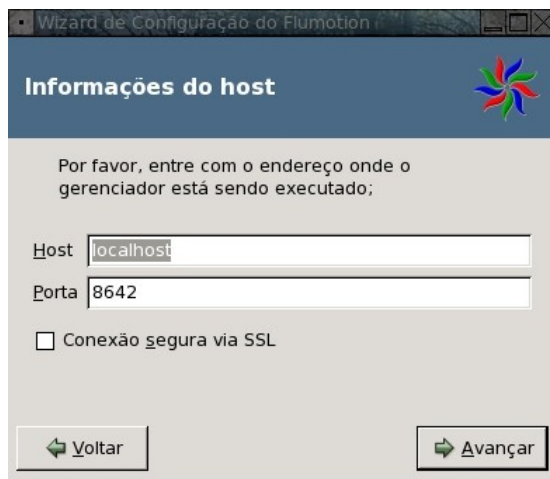


Figura 9: Flumotion - Informações do gerenciador

A tela mostrada pela Figura 10 solicita os dados de usuário e a sua senha para que o processo administrativo conecte ao processo gerenciador. Para isso, é

utilizada a conta padrão: *login user* e *senha test*.

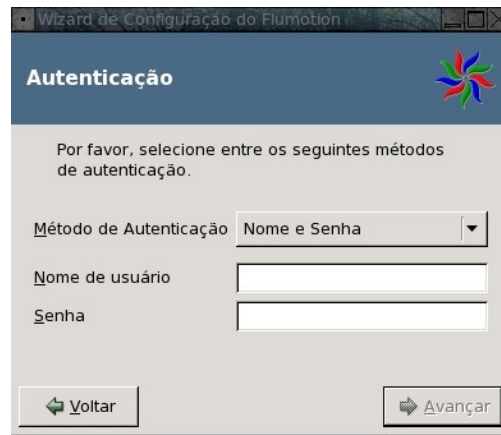


Figura 10: Flumotion - Tela de *login* e senha

Após a tela de *login*, é escolhida a origem do *streaming*, Figura 11, onde são mostradas as opções de vídeo: *webcam*, via *firewire*, placa de captura de vídeo ou a escolha padrão “*Test video source*”, que possibilita um teste padrão sem a necessidade de instalar o *hardware* indicado. Para áudio, as opções são: placa de som, via *firewire* e “*Test audio source*” (apenas para teste). Foi escolhida, para demonstração prática, a origem de vídeo *webcam* e para áudio a placa de som, portanto as telas posteriores irão se basear nestas escolhas, como mostra a Figura 11. As telas do *wizard* deste ponto em diante podem ser diferentes, caso seja escolhida outra origem de áudio/vídeo.

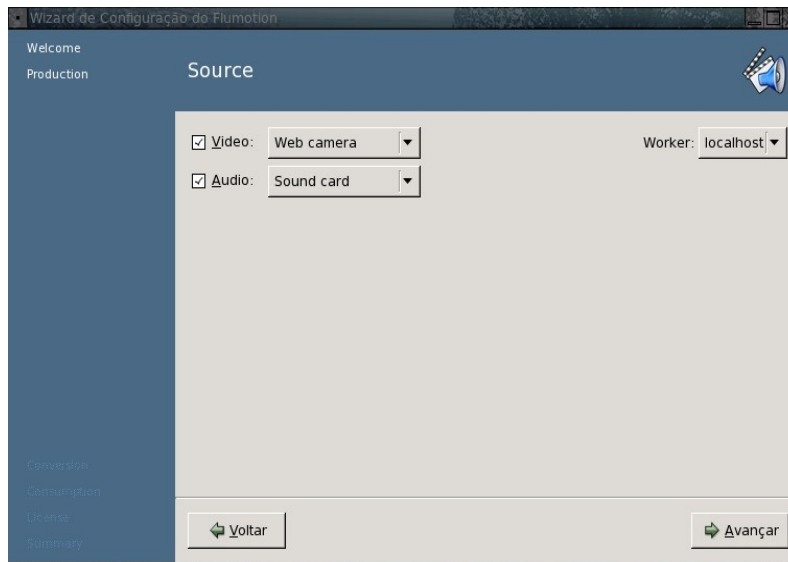


Figura 11: Flumotion - Tipo de origem da transmissão

A próxima tela do *wizard* – Figura 12 – é configurado o dispositivo de vídeo de entrada, no caso *webcam*. Lembrando que o *hardware* deve estar devidamente instalado para que esta etapa não gere erros. Na tela da Figura 12, é escolhida a localização do dispositivo `/dev/video0`, largura e altura a ser capturada e a taxa de *frame* por segundo (*framerate*).

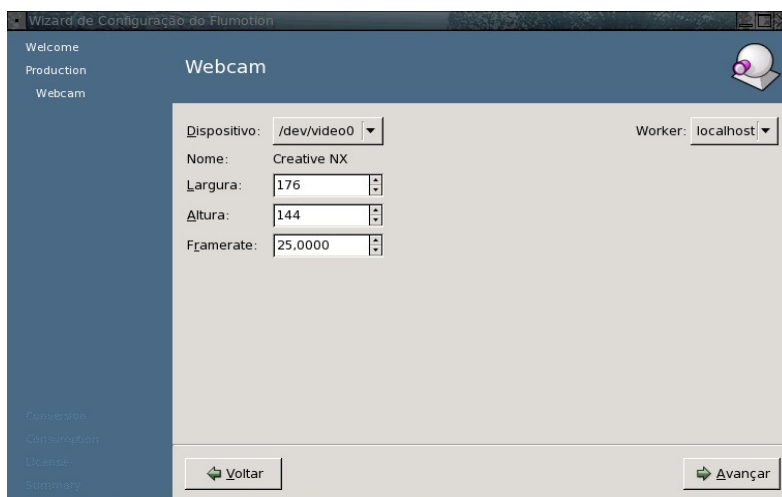


Figura 12: Flumotion - Configuração do dispositivo de vídeo de entrada

Para mostrar um texto no canto superior esquerdo e/ou mostrar o logotipo do Fluendo e Xiph.org no canto inferior esquerdo por sobre o vídeo, é apresentada a próxima tela do *wizard*, a tela *overlay*, Figura 13.

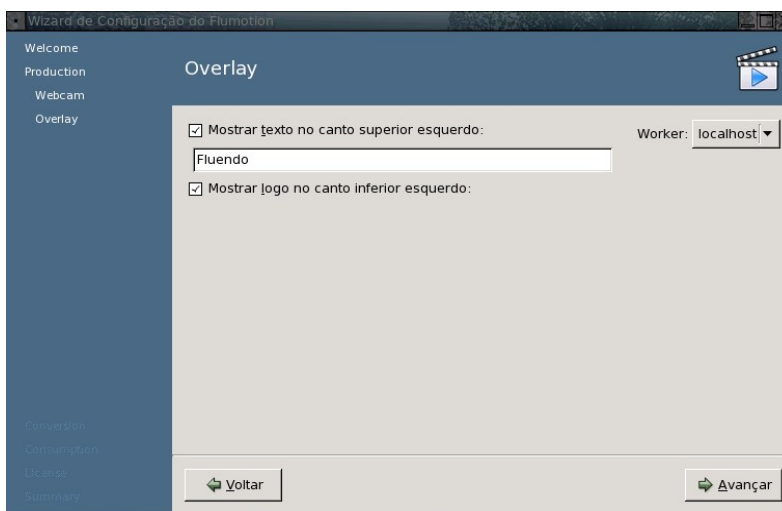


Figura 13: Flumotion - Logotipo, texto por cima do vídeo (*Overlay*)

A próxima tela configura a entrada de áudio para transmissão, como foi escolhido o *hardware* placa de som na Figura 14, surgem as opções de sistema

de som a ser capturado OSS (*Open Sound System*) ou ALSA (*Advanced Linux Sound Architecture*) e a possibilidade de adequar a qualidade: canais, taxa de amostragem e quantidade de *bits*.

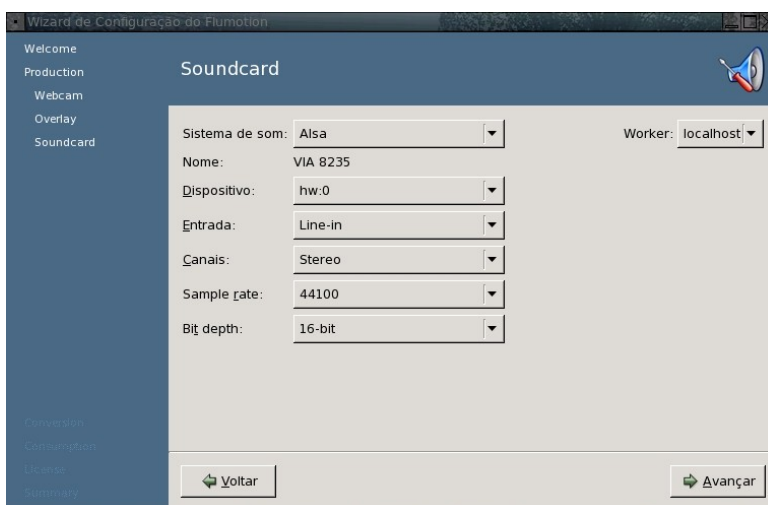


Figura 14: Flumotion - Configuração da origem de áudio

A tela seguinte, Figura 15, configura-se o formato de saída a ser codificado (*encoding*) para transmissão, bem como o CODEC de vídeo e áudio incorporados ao *container*, nativamente o Flumotion transmite em formato Ogg (Vorbis e Theora), uma outra alternativa para áudio é o CODEC Speex.

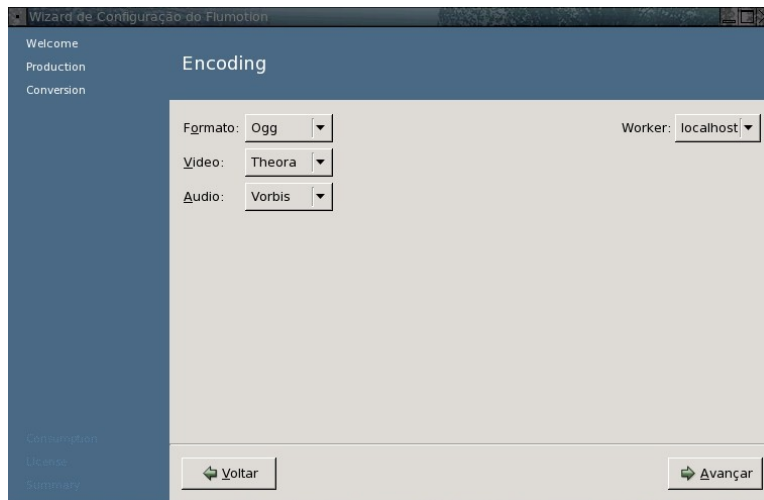


Figura 15: Flumotion - Configuração do *container* a ser transmitido

A Figura 16 mostra algumas configurações para o CODEC de vídeo Theora, onde poderão ser feitas alterações baseadas no *bitrate* ou na qualidade do vídeo. A primeira opção permite prever o consumo de largura de banda que será utilizada em Kbps (Kbit/segundo), não importando a qualidade do vídeo. A segunda opção mantém uma certa constância na qualidade do vídeo, não importando se utilizará muita ou pouca banda.

Como esta execução é realizada em uma rede local, não é necessário alterar o valor padrão mostrado na Figura 16. Nos testes para transmissão pela Internet, o valor do *bitrate* foi alterado para 80 Kbit/s.

A Figura 17 da tela do *wizard* segue os mesmos princípios que a Figura 16. Neste caso, configura-se opções do CODEC Vorbis.

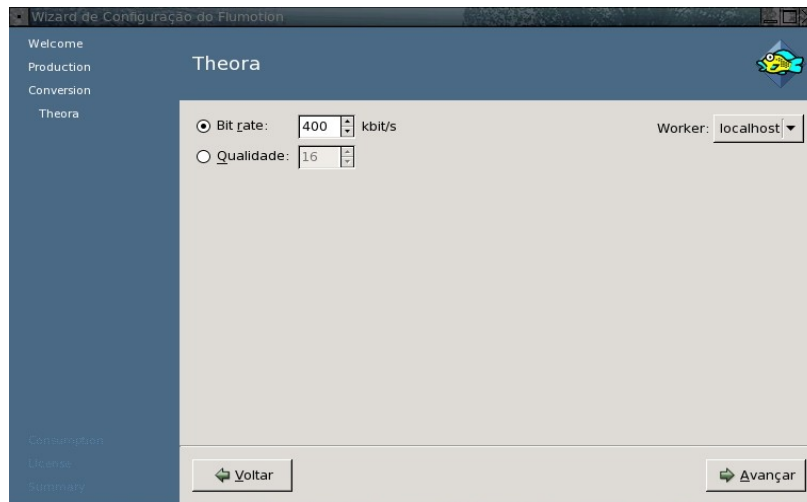


Figura 16: Flumotion - Configuração do CODEC Theora

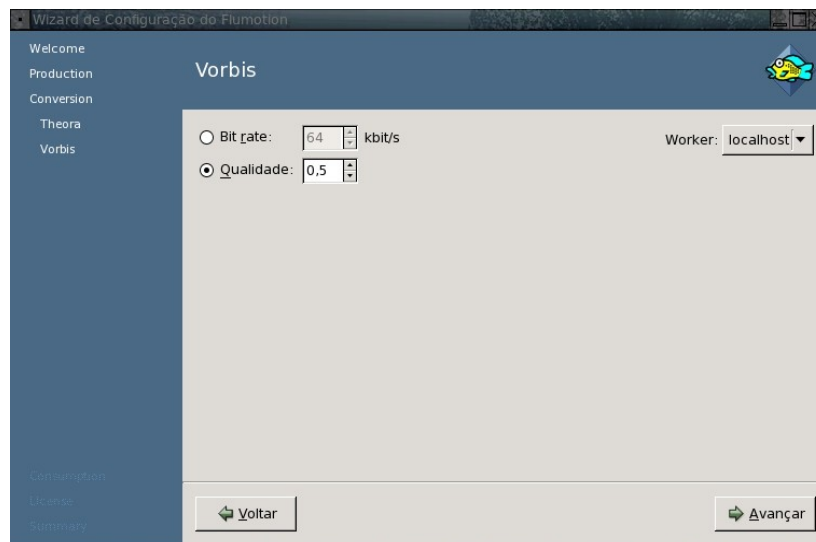


Figura 17: Flumotion - Configuração do CODEC Vorbis

É muito difícil recomendar boas configurações de *bitrate* e de qualidade, tanto de vídeo como de áudio, pois transmissões via Internet agregam muitas

variáveis que influenciam sobre a qualidade final do *streaming*, tais como: número de usuários, suas respectivas taxas de *download*, a taxa de *upload* do servidor, resolução do vídeo, quantidade de movimento, iluminação e quantidade de saltos entre roteadores.

Outros fatores que podem interferir na qualidade são as variáveis relacionadas ao tipo de *hardware* utilizado, se estes possuem suficiente velocidade de vazão do HD (*Hard Disk*), para vídeos sob demanda, se o servidor contém alto processamento, no caso de vídeos *live*, e se a velocidade dos dispositivos de captura são adequados ao tipo de transmissão que se deseja fazer.

Seguindo com o *wizard*, surge a Figura 18 – *Consumption*, onde é configurado a forma de *stream* (possível somente sobre HTTP), a opção de enviar somente vídeo, áudio ou os dois, e a opção de gravar para o disco o que está sendo transmitido.

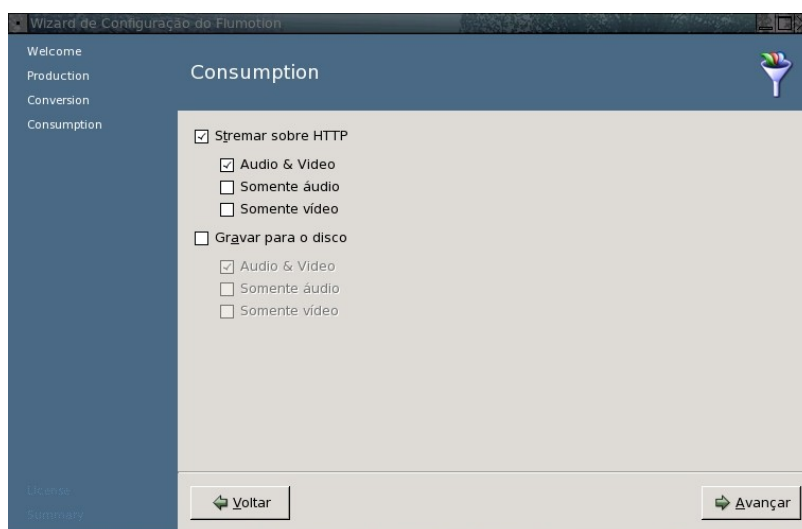


Figura 18: Flumotion - tipo de *streaming*, opção de gravação

Na Figura 19, configura-se qual será o ponto de montagem e porta para

acesso dos usuários e quais serão os limites impostos a conexão criada, como a quantidade de usuários e largura de banda.

O endereço *http://ip-do-servidor:8800/webcam.ogg* é utilizado para que o usuário possa acessar o *streaming* do servidor, fazendo uso da configuração mostrada na Figura 19.

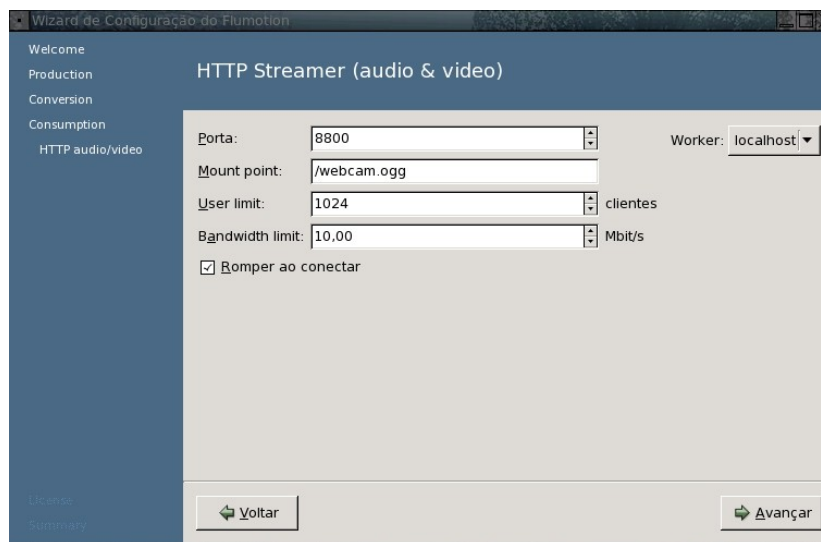


Figura 19: Flumotion - Ponto de montagem, acesso aos usuários

A penúltima tela do *wizard*, Figura 20, permite a escolha de duas licenças: Creative Commons⁵¹ e comercial, caso seja optado pela primeira licença e a opção da Figura 13 - “Mostrar logo no canto inferior esquerdo” estiver habilitada será mostrado o logotipo desta licença juntamente com os outras logotipos.

⁵¹ <http://creativecommons.org/licenses/by-nc-sa/2.0/br/>

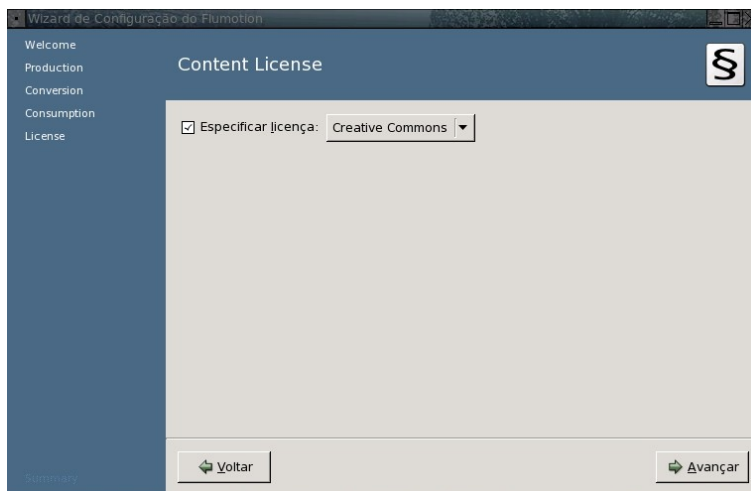


Figura 20: Flumotion - Escolha da licença de uso

A última tela do *wizard* é apenas informativa, ela relata que a configuração foi feita e que o servidor está pronto pra uso, contudo, os passos dados até o momento pelo *wizard* foram apenas de configuração, não é sabido se o servidor irá funcionar a contento. Após a passagem da tela de sumário (última), é apresentada a tela mostrada pela Figura 21.

The screenshot shows a window titled 'User@localhost:8642 - Administração do Flumotion'. The window has a menu bar with 'Conexão', 'Gerenciar', 'Debug', and 'Ajuda'. Below the menu is a toolbar with icons for file operations and a search icon. The main area contains a table with the following data:

Humor	Componente	Worker	PID	% CPU
😊	audio-encoder	localhost		
😊	audio-source	localhost		
😊	http-audio-video	localhost		
😊	muxer-audio-video	localhost		
😊	video-encoder	localhost		
😊	video-overlay	localhost		
😊	video-source	localhost		

Figura 21: Flumotion - Administração

Nesta tela, existem 5 campos: Humor que configura o estado do componente, nome do Componente, localização do Worker, PID (*Process ID*) - número do processo deste componente e % CPU (*Central Processing Unit*) - quantidade de uso do processador. Cada linha indica um componente criado e seus valores nos campos citados.

O estado do componente pode assumir algumas “caras” que facilitam saber se o componente foi bem sucedido no carregamento ou não. Estas “caras” e campos são melhor demonstrados pela Figura 22.



Figura 22: Flumotion - Estado do componente

Percebe-se que o estado inicial indica indiferença pois não foi ativada uma configuração no componente. No segundo estado, preparação, são carregadas as opções e as configurações de cada componente e, a partir deste, o componente pode ir para o estado 3 (Tudo OK) ou 4 (Falha).

A Figura 23 e a Figura 24 demonstram a passagem dos estados dos componentes até o pleno sucesso, gerando, na Figura 24, o endereço final a ser acessado pelo usuário, além de estatísticas da conexão.

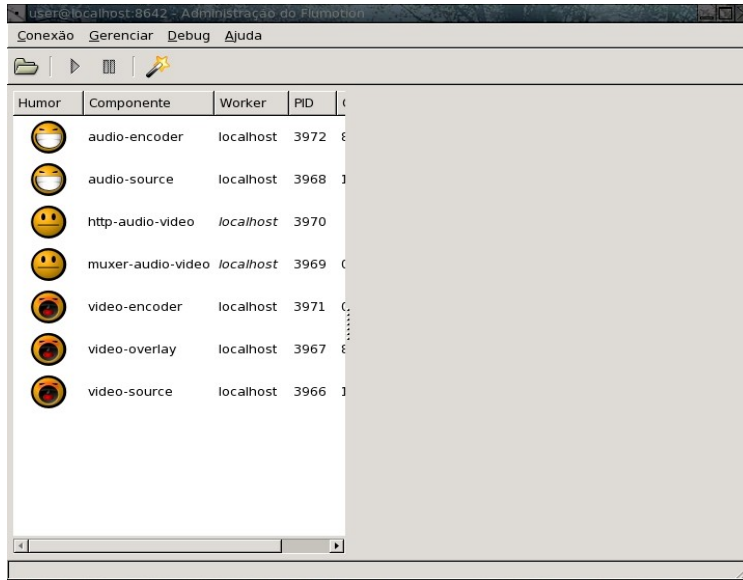


Figura 23: Flumotion - Estado intermediário

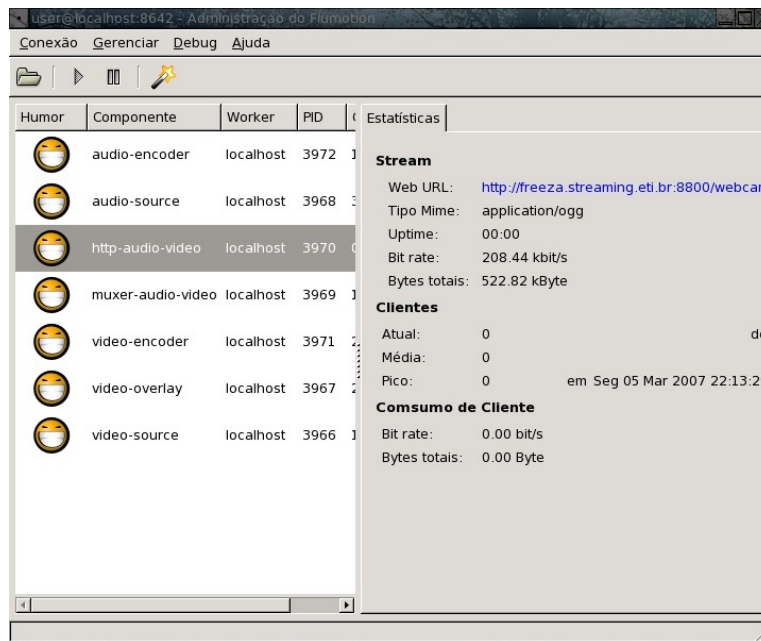


Figura 24: Flumotion - Estado final correto

Caso no carregamento de um componente ocorra erro, poderá o erro ser estudado na tela de depuração, Figura 25.

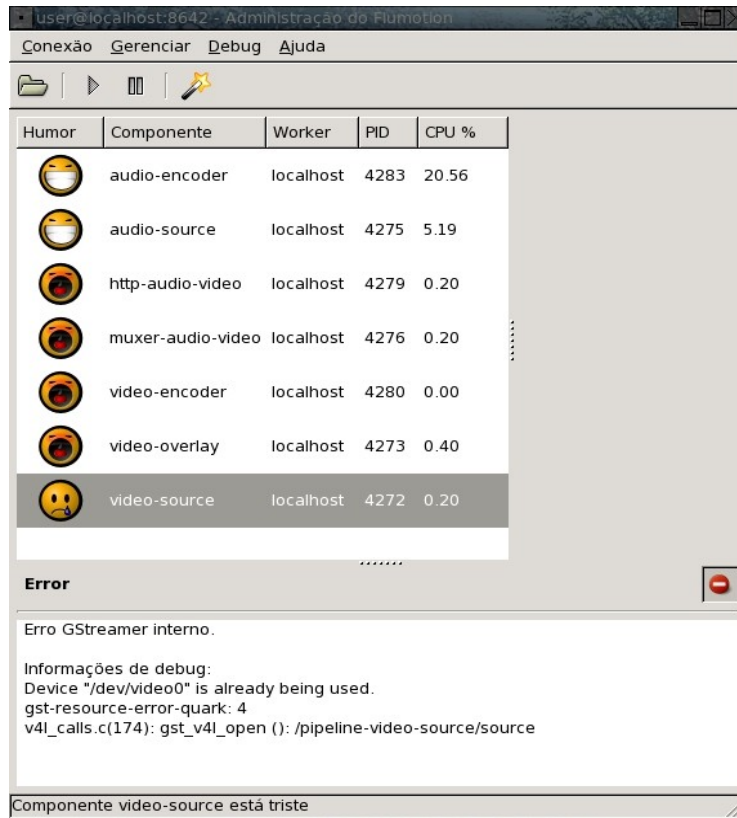


Figura 25: Flumotion - Janela de depuração de erros

Neste exemplo, o componente da origem do vídeo (*video-source*) gerou um erro indicando que este dispositivo */dev/video0* é ocupado por outra aplicação. Nota-se que os componentes relacionados com o áudio, com exceção do componente *http-audio-video*, estão prontos para servir, ao contrário dos componentes que dependem do componente falho.

4.2. Icecast

Mantido pela Xiph.org, Icecast⁵² foi inicialmente utilizado para construir estações de rádio realizando *streaming* somente de áudio nos formatos proprietário MP3 e livre Ogg Vorbis. Este servidor de *streaming* atualmente, na sua versão 2.3.1, possibilita a transmissão de vídeos sob o formato Ogg Theora⁵³, além de ter incorporado os formatos Ogg: Speex, FLAC e MIDI (*Musical Instrument Digital Interface*). A forma de transmissão *streaming* deste servidor é realizada sobre o par HTTP/TCP.

O Icecast é distribuído debaixo da licença GNU GPL, versão 2, possui em sua instalação o requerimento das bibliotecas *libxml2*⁵⁴ (*XML parser library*), *libxslt*⁵⁵ (*XML transformation library*) e, no caso de realizar *stream* no formato Vorbis e Theora, requer *libogg*, *libvorbis*, *libtheora* e a requerido a ferramenta de manipulação de URL (*Uniform Resource Locator*), *curl*⁵⁶ (ICECAST, 2006).

Este servidor utiliza programas externos, os chamados *source client* para prover a origem do *stream*, em outras palavras, o conteúdo a ser transmitido pelo Icecast, é recebido por ele através de dados codificados pelos *source client*. Apesar de ser referenciado como *client* (do servidor Icecast), os *source client* são a fonte de áudio/vídeo para um ponto de montagem particular (NIXON, 2004). A Figura 26 permite uma melhor visualização de como é realizado este processo.

52 <http://www.icecast.org/>

53 na verdade este recurso está presente desde a versão 2.2.0

54 obtido via dropline-GNOME ou <http://xmlsoft.org/downloads.html>

55 obtido via dropline-GNOME ou <http://xmlsoft.org/XSLT/downloads.html>

56 <http://curl.haxx.se/download.html>

Stream structure per mount point

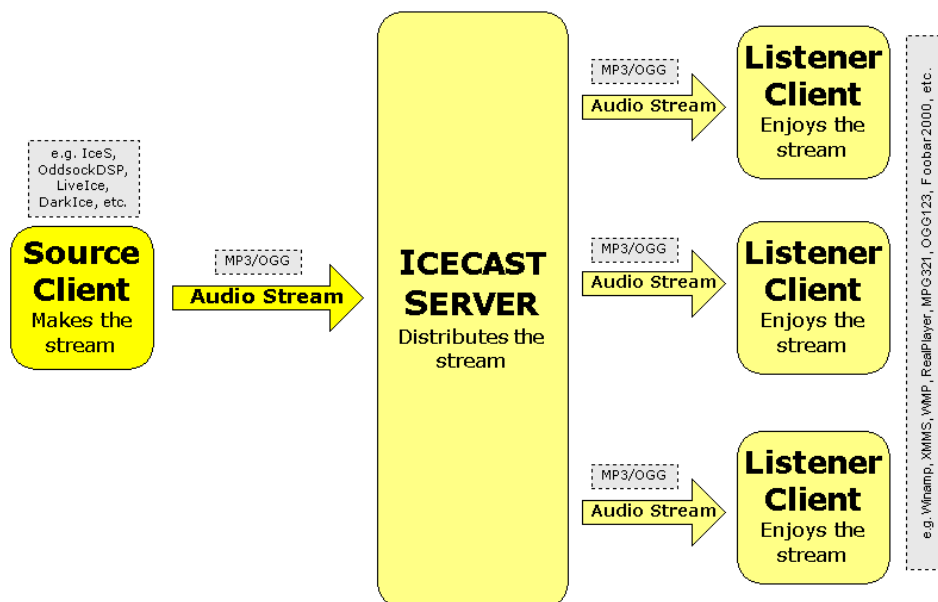


Figura 26: Icecast, *source client* e *listener client*, funcionamento (NIXON, 2004)

Os chamados *listener client*, demonstrados na Figura 26, são os programas de usuários finais que irão tocar o conteúdo multimídia, tais como: XMMS, ogg123, Rhythmbox, VLC e MPlayer.

Alguns *source client* que suportam Icecast são Oddcast, ices2, ices0.3, DarkIce, LiveIce e *oggfwd*. Para que estes *source client* sejam utilizados ou para construir este tipo de *software*, é utilizada a biblioteca *libshout* (versão utilizada 2.2.2).

Icecast suporta *relaying*⁵⁷ (retransmissão) de *streams*. *Relaying* é o processo pelo qual o servidor espelha um ou mais *streams* de um servidor remoto. Os servidores não necessitam ser do mesmo tipo, por exemplo Icecast

⁵⁷ http://www.icecast.org/docs/icecast2.3.1/icecast2_relay.html

pode realizar *relay* do SHOUTcast⁵⁸, usado principalmente para grandes *broadcasts* que necessitam distribuir conteúdo para ouvintes (*listening client*) através de múltiplas máquinas (ICECAST, 2006).

A Figura 27 mostra o esquema de trabalho entre o servidor Icecast, os servidores Icecast *Relay*, os *source client* e os ouvintes (*listener client*) através de uma rede. A figura faz uso do nome *channel* (canais), referenciando aos pontos de montagem utilizados pelo Icecast. Estes pontos de montagem são um recurso que representam um único *stream broadcast*, possuindo a similiaridade com os nomes de arquivos por exemplo: */mystream.ogg* e */mymp3stream*. Quando usuários conectam ao Icecast, eles necessitam especificar o *mountpoint*⁵⁹ requisitado na forma mostrada a seguir (ICECAST, 2006):

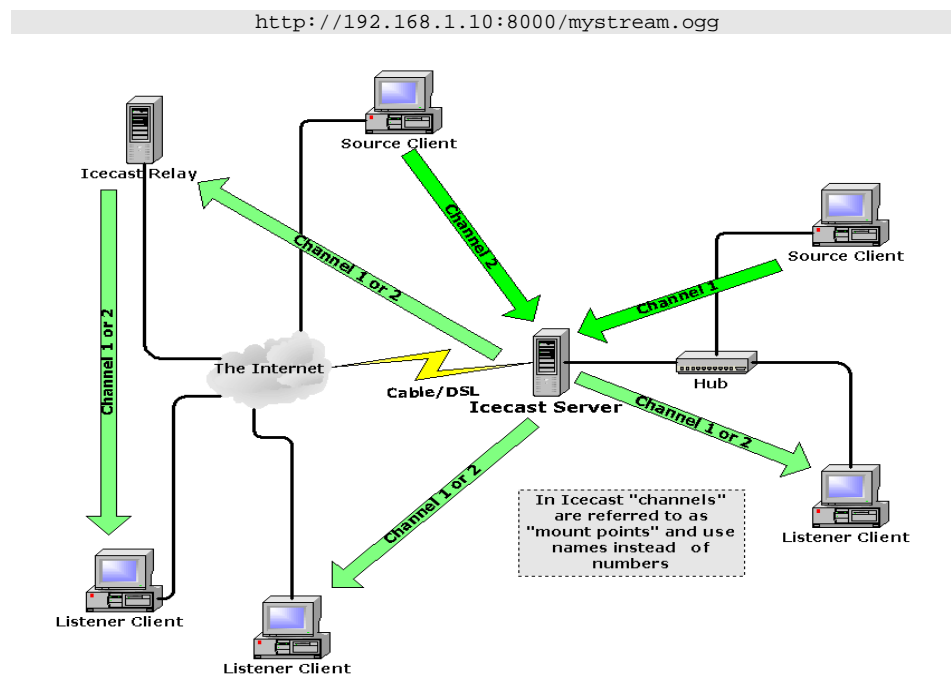


Figura 27: Icecast: Visão de rede (NIXON, 2004)

⁵⁸ <http://www.shoutcast.com/>
⁵⁹ Ponto de montagem

Os dois componentes utilizados para transmitir *streaming* pelo Icecast são os *source client* e o próprio servidor Icecast, podendo ambos estarem ou não na mesma máquina (normalmente estão em máquinas distintas). Icecast pode prover vários pontos de montagem, enquanto os *listeners* podem se conectar em apenas um destes pontos por vez.

Para utilizar o servidor Icecast, após a instalação dos programas necessários, foi criado um diretório principal (*/icecast*⁶⁰) e seus subdiretórios (*admin/*, *log/* e *web/*) como mostrado na Listagem 2.

Listagem 2: Icecast - diretórios e arquivos

```
/icecast/  
+-- icecast.xml  
+-- admin/  
+-- log/  
|   +-- access.log  
|   +-- error.log  
+-- web/  
    +-- Breaking.ogg  
    +-- Forca.ogg  
    +-- Tracy.ogg
```

Foi criada uma cópia do arquivo *icecast_minimal.xml.dist* (no *source* localizado em *icecast-2.3.1/conf/*) para o diretório */icecast/*, alterando o nome para *icecast.xml*⁶¹. Este arquivo foi escolhido por conter uma configuração mínima para o servidor rodar e seu conteúdo foi alterado para refletir os diretórios criados, entre outras configurações. As linhas que importam ser alteradas nesta configuração básica são listadas pela Listagem 3 (ICECAST, 2006).

60 Escolha do autor

61 Escolha do autor

Listagem 3: Icecast - arquivo de configuração *icecast.xml*

```
<source-password>hackme</source-password>
<admin-password>hackme</admin-password>
<hostname>192.168.1.10</hostname>
<port>8000</port>
<logdir>/icecast/log</logdir>
<webroot>/icecast/web</webroot>
<adminroot>/icecast/admin</adminroot>
```

Onde tem-se:

- *source-password* – senha para os *source client*;
- *admin-password* – senha para acesso a características administrativas;
- *hostname* e *port* – qual endereço e porta irá escutar;
- *logdir* – diretório dos arquivos *log* (*/icecast/log*);
- *webroot* – diretório dos arquivos a serem transmitidos (*/icecast/web*);
- *adminroot* – diretório contendo arquivos administrativos *xslt* (*/icecast/admin*).

Após estes passos, para iniciar o servidor, basta executar o comando:

```
# icecast -c /icecast/icecast.xml
```

É bom esclarecer que o Icecast permite a transmissão de *streams* tanto de vídeo sob demanda quanto ao vivo. Na forma de transmissão sob demanda, basta adicionar arquivos *.ogg* no diretório *web/* mostrado pela Listagem 2. Os usuários poderão acessar o conteúdo através de um endereço como:

```
# mplayer http://192.168.1.10:8000/Breaking.ogg
```

Interessa ressaltar que a preocupação sobre a velocidade de transmissão, a dimensão e a qualidade do áudio/vídeo, o sincronismo, entre outros fatores, está ligada a criação do respectivo arquivo *.ogg* assim o servidor irá se preocupar

somente com fornecimento de pontos de montagem para acesso às mídias, não se ocupando em transcódificações para melhoria da transmissão (esta é uma característica da forma de *streaming* sob demanda).

Para a forma de *streaming* ao vivo, é necessária a utilização de *software* externo para introdução de dados no servidor, por exemplo os *source client*. No exemplo a seguir, foi utilizado o *software ffmpeg2theora* para conversão para *.ogg*, juntamente com o *oggfwd* que possibilita o envio de *stream* para o servidor Icecast.

```
ffmpeg2theora /Video/arquivo_video.mpg --inputfps 15 -x 160 -y 120 -o - |  
oggfwd 127.0.0.1 8000 hackme /theora.ogg
```

É notado que, na forma de *streaming* ao vivo, o arquivo ou a entrada de áudio/vídeo não precisa necessariamente estar no formato *.ogg*, pois o mesmo pode sofrer transcodificação antes de ser enviado ao servidor. No exemplo anterior, *ffmpeg2theora* codifica o arquivo de entrada em *.ogg* com alteração no FPS (*Frames per Second*) e nas dimensões. Assim, é realizado um redirecionamento para o *software oggfwd*, que envia o *stream* para o servidor.

4.3. VLC

VLC *media player*⁶², originalmente chamado VideoLAN Client, é o principal projeto do grupo francês VideoLAN. Este *software* é distribuído sob a licença GPL e consiste de uma solução completa de vídeo, incluindo *player* para variadas plataformas (Linux, Windows, Mac OS X, BeOS, *BSD), possibilita a captura de vários dispositivos (DVD, arquivo, placa de captura, *webcam*), capaz de ler vários formatos de arquivo (DivX, *network stream*, MPEG, Ogg, Flash, AVI e Matroska), permite realizar transcodificação *on-the-fly* de formatos

⁶² <http://www.videolan.org/vlc/>

(*transcoding*) e principalmente possibilita a transmissão de *streaming* ao vivo (LATTRE, 2005).

O *streaming* sob demanda utilizando este *software* se realiza na forma de um servidor *web* (ex: Apache) contendo os arquivos multimídia, e a utilização do VLC para tocar estes arquivos armazenados. Um exemplo de linha de comando para tocar é apresentado a seguir:

```
vlc http://servidor_web/videos/slackware.ogg
```

O esquema das várias possibilidades de uso deste *software* é apresentado pela Figura 28.

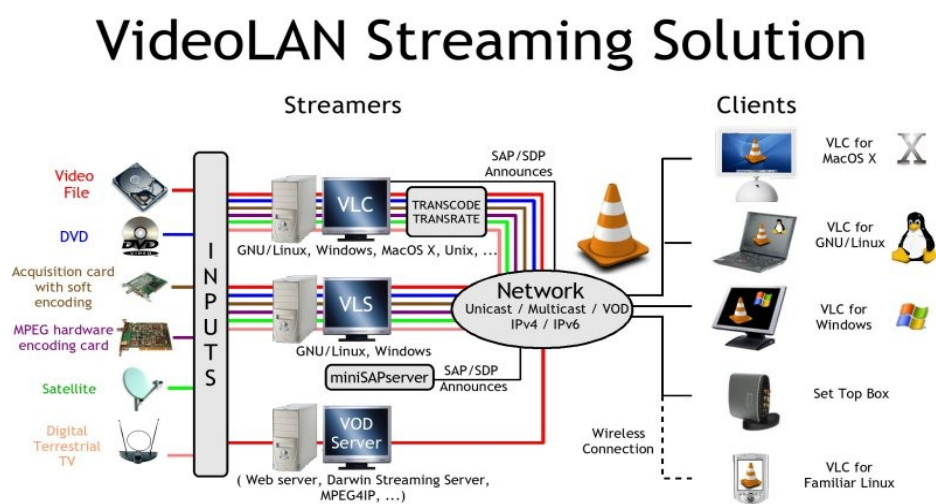


Figura 28: VideoLAN Streaming Solution (LATTRE, 2005)

O modo de utilização do VLC pode ser realizada através de linha de comando ou GUI (*Graphical User Interface*). Para os exemplos, foi utilizada a versão 0.8.6a⁶³ (compilada pela fonte) e, dependendo dos recursos a serem utilizados, será necessária a compilação/instalação de bibliotecas de terceiros

⁶³ <http://www.videolan.org/vlc/download-sources.html>

como: *libogg*, *libtheora*, *matroska*, *wxWidgets*, GNOME1/2 e *V4L*⁶⁴.

A Figura 29 mostra a tela gráfica do VLC onde são apresentadas duas formas de trabalho, a primeira indicada pelo número 1 permite utilizar o VLC como *player*, abrindo arquivos, DVDs, *stream* de rede e dispositivos que utilizem a biblioteca *v4l* (*video4linux*⁶⁵). A segunda indicação, referida na Figura 29 pelo número 2, permite que a escolha feita pela indicação 1 seja transmitida por *streaming* ao vivo, em outras palavras, para que o VLC funcione somente como *player*, deve-se utilizar a indicação 1, caso a intenção seja realizar *streaming*, escolhe-se a origem (indicação 1) e opções de envio (indicação 2).

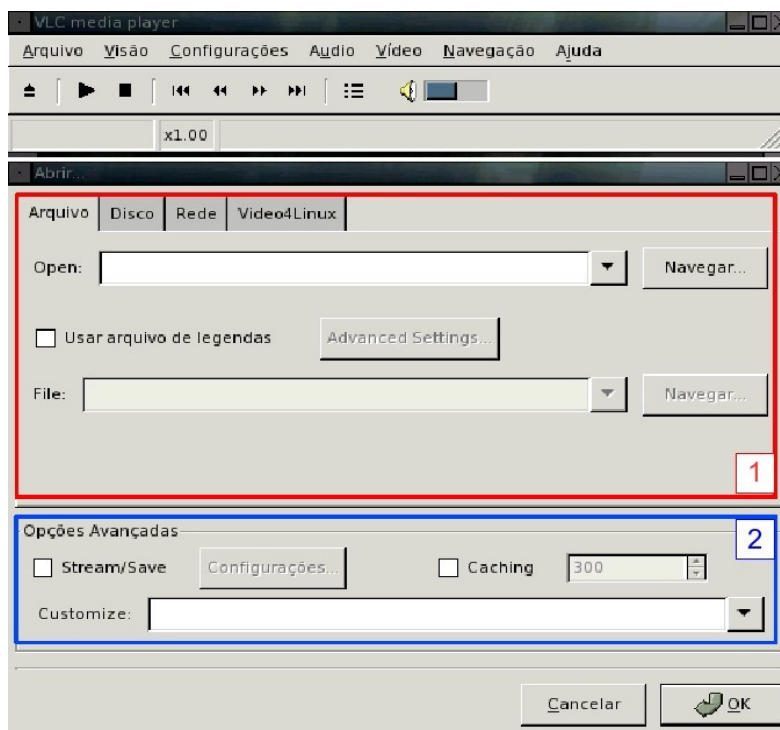


Figura 29: VLC gráfico, modos de trabalho

64 <http://www.videolan.org/developers/vlc.html>

65 <http://linux.bytesex.org/v4l2/>

As opções de transmissão de *streaming* ao vivo podem ser seleccionadas escolhendo o item “Stream/Save” e “Configurações” mostrada pela Figura 29 - indicação 2. Estas opções podem ser visualizadas pela Figura 30 que, no exemplo mostrado, fará o *streaming* via protocolo HTTP, pelo endereço 192.168.254.30 porta 8800, utilizando o *container* Ogg e dentro deste os CODEC Theora (*theo*) e Vorbis (*vorb*) tendo alguns ajustes específicos.

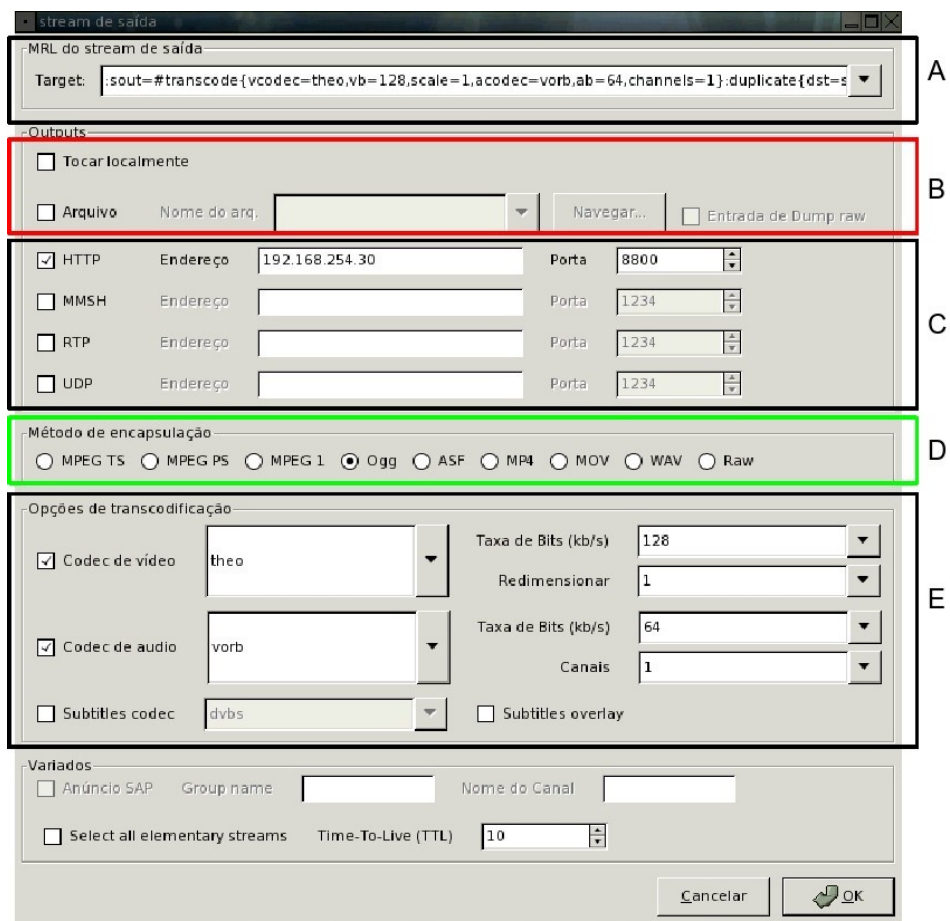


Figura 30: VLC - Opções avançadas de *streaming*

As opções de configuração de *stream* podem ser divididas em 5 partes, como mostra a Figura 30. O item A permite que o usuário obtenha as configurações escolhidas por ele em modo de comando, este item facilita a utilização destes comandos em um terminal. O item B da mesma figura permite que o elemento aberto (arquivo, DVD, V4L, *stream* de rede) possa ser tocado localmente e/ou gravado em um arquivo (com a configuração do *container*/CODEC). Os itens de configuração indicados pela letra C possibilitam a escolha da forma de transmissão: qual protocolo, endereço e porta respectiva. No item E da mesma figura, é possível escolher qual CODEC de vídeo/áudio/legenda será transmitido bem como alguns ajustes (*bitrate*, redimensionamento, canais). No item D, escolhe-se qual método de encapsulamento (*container*) conterá os CODEC selecionados.

É importante ressaltar que as opções disponíveis pelas indicações C, D e E da Figura 30 devem estar em concordância com as definições disponíveis no tópico ***Streaming features list*** do site VideoLAN *Streaming* (<http://www.videolan.org/streaming-features.html>) onde são mostradas as tabelas:

- ***VLC Stream output*** – combinação do protocolo com o tipo de sistema operacional servidor;
- ***Output method/muxer matrix*** – combinação do *container* para um tipo de protocolo;
- ***Muxer/audio and video formats matrix*** – combinação do CODEC de vídeo / áudio para um tipo de *container*.

Seguindo os tópicos das características de *streaming* apresentadas, pode-se diminuir os erros na hora de transmitir um *stream* pelo VLC.

O modo gráfico do VLC facilita tocar arquivos multimídia, receber *streams* de rede, abrir DVDs, permitir criação de *playlist*, transcodificar,

visualizar mensagens das operações internas entre vários outros recursos. O modo console, assim como o modo gráfico, permite realizar as tarefas descritas, contudo ele possibilita maior controle das inúmeras opções que o VLC possui, proporciona uma visualização mais detalhada das mensagens e permite um modo mais avançado de manipulação do *software*. No exemplo a seguir, é utilizado o modo texto para visualizar a *webcam* ou outro dispositivo conectado em */dev/video0* sem áudio, com resolução de 160x120:

```
vlc -v v4l:/dev/video0:norm=secam:size=160x120:channel=0 --no-audio
```

Para realizar *streaming*, por exemplo de um arquivo multimídia via HTTP para o endereço *server.example.org* porta 8080 do servidor, pode-se utilizar a seguinte linha de comando⁶⁶:

```
vlc -v arquivo_multimidia \  
--sout '#standard{access=http,mux=ogg,dst=server.example.org:8080}'
```

Alterando a linha de comando anterior, acrescentando a transformação do arquivo para um outro formato (Vorbis/Theora), tem-se que o exemplo seguinte se mostra adequado:

```
vlc -vvv arquivo_multimidia \  
--sout '#transcode{vcodec=theo,acodec=vorb,vb=128,ab=64}: \  
standard{access=http,mux=ogg,dst=server.example.org:8080}'
```

No exemplo anterior, é escolhida a taxa de *bitrate* do vídeo em 128 e do áudio em 64. Para que o usuário acesse o *streaming*, utiliza-se a seguinte linha de comando:

```
vlc http://server.example.org:8080
```

Algo similar ao comando anterior pode ser utilizado para tocar vídeos sob demanda utilizando VLC, onde o servidor *web* contém um diretório denominado */videos* e neste estão os arquivos multimídia a serem acessados, a

⁶⁶ <http://www.videolan.org/doc/streaming-howto/en/ch04.html>

linha de comando para tocar um arquivo chamado *FOSDEM2007-Xorg.ogg* seria como apresentada a seguir:

```
vlc http://server.example.org/videos/FOSDEM2007-Xorg.ogg
```

4.4. JRoar

JRoar⁶⁷ é um servidor de *streaming live* para Ogg implementado em Java puro, desenvolvido pela JCraft e licenciado sob a GNU GPL. Este *software* possibilita realizar transmissões *stream* exatamente como faz o servidor IceCast2, aceitando conexões de *source client* e tendo suporte ao CODEC livre Ogg Vorbis, Speex e Theora. JRoar possui a característica diferenciada de trabalhar como um *proxy* para *stream Ogg live* (JROAR, 2004).

Para a demonstração de uso deste servidor, foi utilizada a versão JRoar 0.0.9 (última versão conhecida datada de 17/Out/2004), JRE (*Java Runtime Environment*) 1.5.0 e JDK (*Java Development Kit*) 1.5.0⁶⁸. No intuito de facilitar o uso deste servidor são listados os passos para compilá-lo e deixá-lo pronto para uso, Listagem 4.

Listagem 4: Compilação do servidor JRoar

```
1# unzip jroar-0.0.9.zip
2# cd jroar-0.0.9
3# CLASSPATH=`pwd`
4# export CLASSPATH
5# javac com/jcraft/jogg/*.java
6# javac com/jcraft/jorbis/*.java
7# javac com/jcraft/jroar/*.java

8# find / -iname *.ogg > lista.lst
9# java com.jcraft.jroar.JRoar -port 9000 \
  -playlist /test1.ogg lista.lst -icepasswd hackme
```

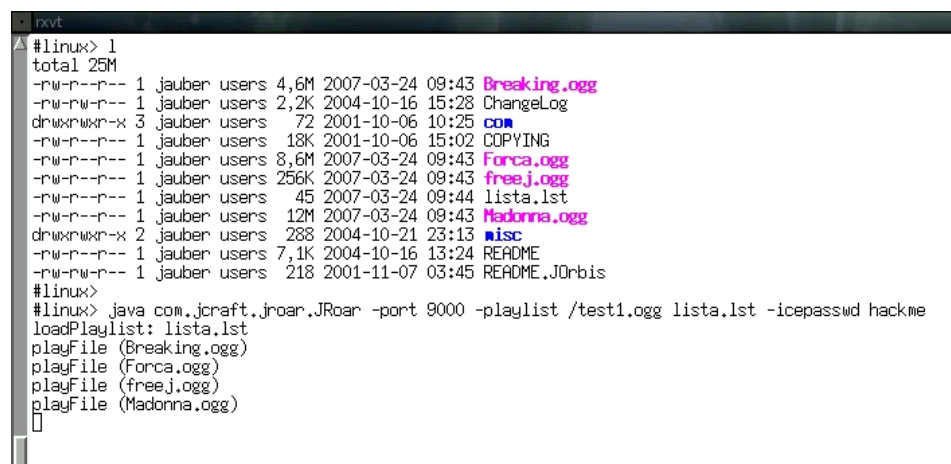
⁶⁷ <http://www.jcraft.com/jroar/>

⁶⁸ Utilizado pacote da distribuição

Na linha 8 da Listagem 4, utiliza-se um comando do Linux para criar um arquivo de nome *lista.lst* que contém os arquivos multimídia (Vorbis e Theora) a serem transmitidos. Na linha 9, é demonstrado como transmitir utilizando a porta 9000, no ponto de montagem */test1.ogg* (este contendo os arquivos encontrados pelo comando da linha 8) e senha definida como *hackme* para prover um nível básico de segurança ao servidor no recebimento de *stream* encaminhado por um *source client* (ex: *ices*, *Oddcast*, *Darkice*).

Apesar da semelhança com a transmissão sob demanda, a execução da linha 9 da Listagem 4 realiza uma transmissão ao vivo, pois o servidor não espera requisições do usuário para iniciar uma transmissão, ele transmite os arquivos independentemente de ser solicitado e, no caso do JRoar, fica transmitindo em *looping*⁶⁹.

A Figura 31 mostra a listagem do diretório *joar-0.0.9*, a execução do servidor e os arquivos transmitidos. Nesta tela do servidor, é gerado o *log* de acesso.



```
pxvt
#linux> l
total 25M
-rw-r--r-- 1 jauber users 4,6M 2007-03-24 09:43 Breaking.ogg
-rw-rw-r-- 1 jauber users 2,2K 2004-10-16 15:28 ChangeLog
drwxrwxr-x 3 jauber users 72 2001-10-06 10:25 com
-rw-r--r-- 1 jauber users 18K 2001-10-06 15:02 COPYING
-rw-r--r-- 1 jauber users 8,6M 2007-03-24 09:43 Forca.ogg
-rw-r--r-- 1 jauber users 256K 2007-03-24 09:43 freej.ogg
-rw-r--r-- 1 jauber users 45 2007-03-24 09:44 lista.lst
-rw-r--r-- 1 jauber users 12M 2007-03-24 09:43 Madonna.ogg
drwxrwxr-x 2 jauber users 288 2004-10-21 23:13 misc
-rw-r--r-- 1 jauber users 7,1K 2004-10-16 13:24 README
-rw-rw-r-- 1 jauber users 218 2001-11-07 03:45 README.Vorbis
#linux>
#linux> java com.jcraft.jroar.JRoar -port 9000 -playlist /test1.ogg lista.lst -icepasswd hackme
loadPlaylist: lista.lst
playFile (Breaking.ogg)
playFile (Forca.ogg)
playFile (freej.ogg)
playFile (Madonna.ogg)
□
```

Figura 31: JRoar executando em um terminal

⁶⁹ quando a transmissão alcança o final do arquivo é transmitido novamente desde o começo

4.5. Cortado

Cortado⁷⁰ é um *applet* Java escrito pela Fluendo, cuja função é visualizar *streams* de vídeo sob demanda e *live*, substituindo a necessidade de instalar um *player* específico. Possui a característica de tocar os formatos livres Ogg Vorbis e Theora, podendo ser integrado a um servidor *web* para facilitar a distribuição de *streams* sob demanda, no caso de *streams live* pode ser utilizado em conjunto com o Flumotion.

Características do Cortado incluem: posicionamento em arquivos sob demanda, autenticação básica do HTTP e *buffering*. O *applet* pode ser obtido na sua forma empacotada em *jar*⁷¹ ou construída através dos fontes⁷². Um exemplo de sua utilização na forma de vídeo sob demanda é apresentado na Listagem 5, onde tem o código *applet* Cortado dentro de um arquivo HTML.

⁷⁰ <http://www.flumotion.net/cortado/>

⁷¹ <http://www.flumotion.net/jar/cortado/>

⁷² <http://www.flumotion.net/src/cortado/>

Listagem 5: Cortado para *stream* sob demanda

```
<html>
<head><title>Monografia sobre Streaming</title>
</head>
<body>
<center>
<applet archive="cortado-ovt-debug-0.2.2.jar"
code="com.fluendo.player.Cortado.class"
width="352" height="288">
  <param name="url" value="http://localhost/test001.ogg"/>
  <param name="local" value="false"/>
  <param name="duration" value="232"/>
  <param name="keepAspect" value="true"/>
  <param name="video" value="true"/>
  <param name="audio" value="true"/>
  <param name="bufferSize" value="200"/>
  <param name="userId" value="user"/>
  <param name="password" value="test"/>
  <param name="seekable" value="true"/>
  <param name="statusheight" value="14"/>
  <param name="showStatus" value="auto"/>
</applet>
</center>
</body>
</html>
```

Atualmente, Cortado 0.2.2 possui mais de 12 parâmetros que possibilitam ajustá-lo melhor⁷³. Alguns destes parâmetros são:

- *url* – Localização do arquivo de vídeo/áudio;
- *video* – utilizará vídeo?;
- *audio* – utilizará áudio?;
- *bufferSize* – Tamanho do *buffer* utilizado, pode-se decrementar este parâmetro para obter menos espera. Valor padrão 200 em KB;
- *autoPlay* – Toca automaticamente;
- *showStatus* – Controla como o status é mostrado;
- *live* – receberá vídeo/áudio ao vivo?

⁷³ <https://core.fluendo.com/flumotion/trac/browser/cortado/trunk/README>

Utilizando Flumotion, seção 4.1, somente com vídeo, ponto de montagem */webcam.ogg* e porta 8802, faz-se adequada a configuração do Cortado para *streams* ao vivo demonstrada pela Listagem 6.

Listagem 6: Cortado para *stream live*

```
<html>
<head><title>Monografia sobre Streaming</title>
</head>
<body>
<center>
  <applet archive="cortado-ovt-debug-0.2.2.jar"
code="com.fluendo.player.Cortado.class" width="320" height="240">
  <param name="url" value="http://localhost:8802/webcam.ogg"/>
  <param name="live" value="true"/>
  <param name="local" value="false"/>
  <param name="duration" value="232"/>
  <param name="keepAspect" value="true"/>
  <param name="video" value="true"/>
  <param name="audio" value="false"/>
  <param name="bufferSize" value="30"/>
  <param name="userId" value="user"/>
  <param name="password" value="test"/>
  <param name="seekable" value="false"/>
  <param name="statusheight" value="14"/>
  <param name="showStatus" value="auto"/>
  </applet>
</center>
</body>
</html>
```

A Figura 32 mostra o resultado final, o *applet* Cortado em execução.



Figura 32: *applet* Cortado em execução

4.6. Resultados Obtidos

Em um ambiente de testes, quando são implementados vários aplicativos, é de se esperar que alguns tragam maior facilidade de instalação e uso, enquanto outros proporcionam noites de intensa pesquisa. Esta seção tem por objetivo abordar as experiências, os erros e os resultados obtidos pela implementação dos recursos tecnológicos mostrados no Capítulo 4.

Após alguns testes de transmissão de vídeo sob demanda, foi notado por este autor que esta forma exige pouco processador e memória de trabalho em relação a transmissão ao vivo (exemplo: computador com processador de 1Ghz⁷⁴ e 128MB⁷⁵ de RAM⁷⁶). Isso se deve a transcodificação não ser utilizada em *streaming* sob demanda, os arquivos multimídia devem estar no formato

⁷⁴ Gigahertz, 1 (um bilhão de instruções por segundo)

⁷⁵ Megabytes

⁷⁶ Random Access Memory, também conhecida como memória de trabalho

preestabelecido para envio, não necessitando de alterações antes da transmissão. Uma forma de transmitir sob demanda requer apenas que se tenha um servidor *web* e um diretório com os arquivos multimídia. Assim, o trabalho é realizado pelo *player* do usuário.

A forma sob demanda é de fácil implementação e exige-se, para atendimento de vários usuários, a configuração adequada da qualidade dos arquivos. Quando se fala em qualidade, significa dimensionar o *bitrate* de cada arquivo multimídia para que o consumo de conexões não ultrapasse a largura de banda da rede (por usar *unicast* o consumo de cada conexão é a taxa de *bitrate* do respectivo arquivo). Este modelo pode ser aplicado quando se tem vídeos/áudios armazenados e tem-se a intenção de disponibilizar a qualquer momento o conteúdo para o usuário. Pode-se criar páginas de *web* para facilitar o acesso⁷⁷.

Para *streaming* sob demanda, é utilizado um servidor *web* ou um servidor de *streaming* como Icecast (Seção 4.2) que disponibiliza funções extras ao usuário, como *playlist* e administração remota. No lado do usuário pode-se utilizar o *software* VLC, Mplayer, Xine e Cortado, dentre outros, para tocar os arquivos contidos nestes servidores.

A transcodificação, que é o ato de alterar um formato de origem para um outro formato codificado, pode ou não ser utilizado ou não na transmissão de *streaming* ao vivo. Comumente, a transcodificação é usada em transmissões onde a origem dos dados são dispositivos de captura, como *webcam*, placas de captura de vídeo e dispositivos de vídeo com conexão *firewire*.

Streaming ao vivo requer muito processador e memória de trabalho quando utiliza transcodificação, portanto algum *software* apresentado pode depender destes fatores para uma boa transmissão.

⁷⁷ Como o modelo das páginas do YouTube e Google Video.

Na hora de configurar e transmitir, o VLC⁷⁸ ganha pela sua versatilidade em enviar *stream* sob vários formatos, variados protocolos, diferentes origens de captura, rodar sob muitas plataformas e possuir ambiente gráfico de configuração. O VLC é adequado ao usuário que deseja aprender a transmitir *streaming*, contudo não foi obtido muita estabilidade em transmitir no formato livre Ogg com Theora e Vorbis, principalmente no momento da transcodificação para Ogg, onde o autor teve dificuldades em afinar as opções dos CODEC sem causar interrupção do programa (*crash*).

Pela preferência em instalar via *source*, o *software* Flumotion⁷⁹ demonstrou certa complexidade para o autor, não pela compilação do *software* em si, mas pela dependência dos aplicativos GStreamer e Twisted. A maneira de trabalho do Flumotion, utilizando os processos gerados *admin*, *worker* e *manager*, não está completamente entendida, pois o manual carece de exemplos práticos, ilustrações e explicações sobre o funcionamento. O *software* não possibilita a utilização de outros tipos de dispositivos além da *webcam*, placas de captura de vídeo e de áudio e *hardware* com conexão *firewire*, contudo nos dispositivos suportados e testados, o *software* comportou-se muito bem, tanto na captura quanto no envio nativo de Ogg com Theora e Vorbis.

A configuração do Flumotion é facilitada por sua tela gráfica, a utilização em modo console não foi bem sucedido. Pelos testes realizados executando captura, transcodificação e transmissão na mesma máquina se comportou como um consumidor voraz de recursos, contudo pôde-se transmitir com boa recepção. O autor não implementou a forma de trabalho distribuída do Flumotion, porém testou e recomenda a sua utilização com o *player* Cortado para transmissão e recepção de *streaming* ao vivo.

O Icecast para transmissão ao vivo permite descentralizar a origem dos

⁷⁸ Seção 4.3 VLC

⁷⁹ Seção 4.1 Flumotion

dados, como também o processamento pela captura e transcodificação através dos *source client*. Este servidor mostrou-se fácil de instalar e configurar, a dificuldade se tornou presente na combinação das ferramentas para criação dos *source client*. Para usuários não acostumados com linha de comando, pode-se mostrar complicado, pois não possui ambiente gráfico de configuração e a criação de *source client* via *pipe* pode assustar. Este *software* é muito utilizado para transmissão de *stream* de áudio, sendo iniciado para transmissão de vídeo, o autor recomenda mais testes ou a utilização do Flumotion para *streaming* ao vivo.

JRoar⁸⁰ assemelha ao Icecast no que tange na utilização de *source client* para criação de *stream*, contudo difere por não transmitir arquivos sob demanda. Apesar da vantagem de ser construído em Java, o autor não obteve sucesso em utilizar este *software* em ambiente diferente do Linux (Windows). A data de atualização deste projeto foi 17-Out-2004 na versão 0.0.9 não tendo novidades ou correções desde então.

Uma observação se faz necessária em relação ao *applet* Cortado⁸¹: apesar de estar juntamente com os programas servidores de *streaming*, ele é um *player* Java que roda no navegador do usuário (como Flash), dispensando o uso de um aplicativo específico para tocar os *stream* enviados. Pode ser utilizado com qualquer um dos servidores descritos neste trabalho que suporte transmissão do *container* Ogg.

O *container* Matroska não foi comentado para transmissão com estes servidores por não estar em sua forma plena e por não ter suporte para a maioria deles, está sendo desenvolvido para alcançar versão estável. Atualmente, o projeto está na etapa 3.5 de um total de 4, seu formato é utilizado individualmente por alguns aplicativos.

80 Seção 4.4 JRoar

81 Seção 4.5 Cortado

Alguns servidores de *streaming* que são distribuídos como *open source*, não foram comentados neste trabalho por não terem suporte a formatos livres para transmissão, como: LIVE555⁸², Darwin Streaming Server⁸³ da Apple e MPEG4IP⁸⁴.

82 <http://www.live555.com/liveMedia/>

83 <http://developer.apple.com/opensource/server/streaming/index.html>

84 <http://mpeg4ip.sourceforge.net/>

5. Conclusão

A transmissão de vídeo/áudio pela Internet possibilita a geração de serviços como biblioteca áudio-visual, sistemas de segurança, videoconferência, vídeo-aula, rádios *online*, canais de notícias, publicação de vídeos pessoais e publicitários. A maioria destes serviços utiliza *software* e formatos proprietários para sua criação/transmissão.

O presente trabalho procurou demonstrar a utilização de *software* livre para transmissão de conteúdo multimídia, desde o formato até posterior envio pelo servidor de *streaming*. Foram estudados alguns aplicativos servidores do mercado mais conhecidos e utilizados, bem como foram levantados os prós e contras de cada um. O trabalho procurou ainda definir e classificar a teoria sobre *streaming*, diferenciar o confuso *container* com o CODEC e mostrar formatos *open source* destes dois.

Em relação aos servidores discutidos no Capítulo 4, nota-se que o *software* de *streaming* Flumotion se destacou dos demais pela sua facilidade em capturar dispositivos conhecidos e possibilidade de transmissão estável no formato livre Ogg. Na opinião do autor, o aplicativo VLC tornou-se uma ferramenta de uso geral, *player* e servidor, tendo negligenciado de certa forma o aprimoramento do *stream* dos formatos livres. JRoar e Icecast possuem formas de trabalho semelhantes, contudo o servidor Icecast destaca-se pelo suporte, atualizações e posição consolidada ante os usuários multimídia espalhados pela Internet. A utilização de *player* integrado a um navegador, como Cortado, tende a crescer, principalmente como facilitador da distribuição de conteúdo multimídia.

Desde o surgimento do conceito, a tecnologia de *streaming* proporciona melhor distribuição de conteúdo áudio/vídeo pela Internet, contudo muito precisa ser feito para que a transmissão alcance um nível satisfatório de

qualidade, principalmente na transmissão de vídeo. Dessa maneira, propõe-se para trabalhos futuros investigar a implementação de servidores livres utilizando protocolos padronizados para *streaming* como RTCP e RTP, estudo da qualidade de serviço (QoS) associada a transmissão, levantamento de infraestruturas alternativas para tráfego, como CDN (*Content Delivery Network*⁸⁵), comparação de desempenho entre *software open source* e proprietário, comparativo da transmissão de *streaming* em P2P (*Peer to Peer*⁸⁶) com as formas conhecidas de transmissão.

85 http://whatis.techtarget.com/definition/0,289893,sid9_gci1187046,00.html

86 <http://www.peercast.org/>

6. Referências Bibliográficas

APPLE. *Reference Library: Streaming QuickTime Over RTP or HTTP*. [online]. Disponível na internet via [www. url: http://developer.apple.com/documentation/QuickTime/RM/Streaming/StreamingClient/B-Chapter/chapter_1000_section_4.html](http://developer.apple.com/documentation/QuickTime/RM/Streaming/StreamingClient/B-Chapter/chapter_1000_section_4.html). Arquivo capturado em 15 de janeiro de 2007.

CANAN, Rafael; RAABE, A. L. A. *Um Ambiente para Transmissão de Vídeos Instrucionais sob Demanda*. [online]. Disponível na internet via [www. url: http://www.cinted.ufrgs.br/renote/mar2004/artigos/22-umambiente_transmissao.pdf](http://www.cinted.ufrgs.br/renote/mar2004/artigos/22-umambiente_transmissao.pdf). Arquivo capturado em 18 de maio de 2007.

CETIC.br. *TIC DOMICÍLIOS e USUÁRIOS 2006: Pesquisa sobre o uso das tecnologias da informação e da comunicação no Brasil*. [online]. Disponível na internet via [www. url: http://www.cetic.br/usuarios/tic/2006/index.htm](http://www.cetic.br/usuarios/tic/2006/index.htm). Arquivo capturado em 19 de janeiro de 2007.

DARPA[1]. *Internet Protocol*. Internet Engineering Task Force (IETF). Setembro 1981. (Request for Comments: 791). [url: ftp://ftp.rfc-editor.org/in-notes/rfc791.txt](ftp://ftp.rfc-editor.org/in-notes/rfc791.txt).

DARPA[2]. *Transmission Control Protocol*. Internet Engineering Task Force (IETF). Setembro 1981. (Request for Comments: 793). [url: ftp://ftp.rfc-editor.org/in-notes/rfc793.txt](ftp://ftp.rfc-editor.org/in-notes/rfc793.txt).

FIELDING, R. *Hypertext Transfer Protocol -- HTTP/1.1*. Internet Engineering Task Force (IETF). Junho 1999. (Request for Comments: 2616). [url: ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt](ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt).

ICECAST. *Icecast 2 Documentation Table of Contents*. [online]. Disponível na internet via [www. url: http://www.icecast.org/docs/icecast-2.3.1/](http://www.icecast.org/docs/icecast-2.3.1/). Arquivo capturado em 25 de janeiro de 2007.

JROAR. *README*. [online]. Disponível na internet via [www. url: http://www.jcraft.com/jroar/](http://www.jcraft.com/jroar/). Arquivo capturado em 01 de março de 2007.

KUMAR, Vinay. *Real-time multimedia Broadcasts with the Internet Multicast Backbone*. [online]. Disponível na internet via [www. url: http://www.microsoft.com/mind/0297/mbone/mbone.asp](http://www.microsoft.com/mind/0297/mbone/mbone.asp). Arquivo capturado em 17 de fevereiro de 2007.

KUROSE, James F; ROSS, Keith W. *Computer Networking: A Top-Down Approach Featuring the Internet*. 3.ed. EUA; Addison Wesley, 2004.

LATTRE, Alexis de; BILIEN, Johan; DAOUD, Anil; *VideoLAN Streaming Howto*. [online]. Disponível na internet via www. url: <http://www.videolan.org/doc/streaming-howto/en/streaming-howto-en.html>. Arquivo capturado em 01 de março de 2007.

MATROSKA. *Diagram*. [online]. Disponível na internet via www. url: <http://www.matroska.org/technical/diagram/index.html>. Arquivo capturado em 03 de março de 2007.

NIXON, Stephen. *How does Icecast v2 all fit together?*. [online]. Disponível na internet via www. url: <http://liveice.sourceforge.net/understanding.html>. Arquivo capturado em 01 de fevereiro de 2007.

ORNELLAS, Fabio Pugliese. *Video e Áudio digital: usando MPEG-4 e OGG Vorbis no Linux*. [online]. Disponível na internet via www. url: <http://www.geocities.com/neofpo/files/linuxvideo-1.2.1.html>. Arquivo capturado em 23 de novembro de 2006.

PFEIFFER, S. *The Ogg Encapsulation Format Version 0*. [online]. Disponível na internet via www. url: <ftp://ftp.rfc-editor.org/in-notes/rfc3533.txt>. Arquivo capturado em 18 de janeiro de 2007.

POSTEL, J. *User Datagram Protocol*. Internet Engineering Task Force (IETF). Agosto 1980. (Request for Comments: 768). url: <ftp://ftp.rfc-editor.org/in-notes/rfc768.txt>.

REVISTA TEMA. *Sintonia Mundial: Nova era do software livre une governo, empresas, pesquisadores e usuários num esforço estratégico contra a escravidão em relação às licenças proprietárias*. [online]. Disponível na internet via www. url: http://www.serpro.gov.br/publicacoes/tema_187/materias/sol_187_01. Arquivo capturado em 31 de janeiro de 2007.

RICHARDSON, Iain E. G. *H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia*. 1.ed. EUA; Wiley, 2003.

SCHULZRINNE[1], H. *RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force (IETF). Julho 2003. (Request for Comments: 3550). url: <ftp://ftp.rfc-editor.org/in-notes/rfc3550.txt>.

SCHULZRINNE[2], H. *Real Time Streaming Protocol (RTSP)*. Internet Engineering Task Force (IETF). Abril 1998. (Request for Comments: 2326). url: <ftp://ftp.rfc-editor.org/in-notes/rfc2326.txt>.

STICHELE, Thomas V.; SCHALLER, Christian F. K. *Flumotion manual*. [online]. Disponível na internet via [www](http://www.flumotion.net/doc/flumotion/manual/html/index.html). url: <http://www.flumotion.net/doc/flumotion/manual/html/index.html>. Arquivo capturado em 27 de fevereiro de 2007.

TAFFAREL, Armando Neto. *Streaming de áudio e vídeo LIVRE*. [online]. Disponível na internet via [www](http://midia.linuxchix.org.br/palestras/4ENLB/streaming.pdf). url: <http://midia.linuxchix.org.br/palestras/4ENLB/streaming.pdf>. Arquivo capturado em 21 de novembro de 2006.

TOPIC, Michael. *Streaming Media Demystified*. 1.ed. EUA: McGraw-Hill Professional, 2002.

UNREAL MEDIA. *Practical advices for setting up IP streaming services*. [online]. Disponível na internet via [www](http://www.umediaserver.net/source.html). url: <http://www.umediaserver.net/source.html>. Arquivo capturado em 20 de fevereiro de 2007.

Apêndice A

Glossário

bandwidth – largura de banda da rede, medido normalmente em Kbps, Mbps.

bitrate – taxa de *bits* por segundo que pode ser lido/processado, medido em Kbps.

bitstream ou ***bit stream*** - é uma seqüência contínua de *bits*, representando um fluxo de dados transmitidos continuamente sobre um caminho de comunicação. Esta forma permite tocar o arquivo sem possuir o início ou mesmo todo o seu conteúdo.

connectionless – não orientado a conexão

container – envólucro que armazena um ou mais CODEC, não possui compressão por si só.

demultiplexing – é o ato de desagrupar um ou mais *streams* de vídeo e/ou áudio de um *container*.

mountpoint – ponto de montagem; método de acesso;

multiplexing - ato de unir um ou mais *streams* de vídeo e/ou áudio em um único arquivo, freqüentemente chamado de *container*.

overhead – sobrecarga

reliable – confiável

transcoding – é o ato de converter um formato codificado para outro, alterando ou não configurações como *bitrate*, resolução, qualidade, etc. Este processo demanda processador e memória de trabalho na sua execução (RAM). Principalmente utilizado na transmissão de *streaming* ao vivo.