Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Comunicações

Desenvolvimento de uma Plataforma Multimídia Utilizando a Linguagem Python

Autor:
Jahyr Gonçalves Neto

Orientador:
Prof. Dr. Max Henrique Machado Costa

Dissertação apresentada à Faculdade de Engenharia Elétrica e Computação da Unicamp como parte dos requisitos para obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA.

Campinas, Novembro de 2007

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Comunicações

Desenvolvimento de uma Plataforma Multimídia Utilizando a Linguagem Python

Autor:
Jahyr Gonçalves Neto

Orientador:
Prof. Dr. Max Henrique Machado Costa

Dissertação apresentada à Faculdade de Engenharia Elétrica e Computação da Unicamp como parte dos requisitos para obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA.

Banca Examinadora:

Prof. Dr. Max Henrique Machado Costa	FEEC/UNICAMP
Prof. Dr. José Raimundo de Oliveira	FEEC/UNICAMP
Prof. Dr. Leonardo de Souza Mendes	FEEC/UNICAMP
Prof. Dr. Henrique J. Q. de Oliveira	NPT/UMC

Campinas, Novembro de 2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

G586d	Gonçalves Neto, Jahyr Desenvolvimento de uma plataforma multimídia utilizando a linguagem Python / Jahyr Gonçalves Neto. -- Campinas, SP: [s.n.], 2007. Orientador: Max Henrique Machado Costa Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação. 1. Programação em tempo-real. 2. Internet (redes de computação). 3. Processamento do som por computador. 4. Videoconferência. 5. Sistemas multimídia. I. Costa, Max Henrique Machado. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Development of a multimedia platform using Python language
Palavras-chave em Inglês: Audio streaming, Video streaming, Videoconference, Multimedia, Python, IPTV
Área de concentração: Telecomunicações e Telemática
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: José Raimundo de Oliveira, Leonardo de Souza Mendes e Henrique Jesus Quintino de Oliveira
Data da defesa: 06/11/2007
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Jahyr Gonçalves Neto

Data da Defesa: 6 de novembro de 2007

Título da Tese: "Desenvolvimento de uma Plataforma Multimídia Utilizando a Linguagem Python"

Prof. Dr. Max Henrique Machado Costa (Presidente): _____

Prof. Dr. Henrique Jesus Quintino de Oliveira: _____

Prof. Dr. José Raimundo de Oliveira: _____

Prof. Dr. Leonardo de Souza Mendes: _____

Resumo

Nesta dissertação apresentamos o desenvolvimento de uma plataforma multimídia baseada no modelo cliente-servidor voltada para aplicações de *streaming* de áudio e vídeo. Essa plataforma deverá evoluir para um sistema de videoconferência em um projeto futuro. A plataforma permite a comunicação de áudio, vídeo e texto a partir de um ponto (o servidor) para vários outros pontos (os clientes). Uma das inovações do projeto está no desenvolvimento em Python, que é uma linguagem interpretada, orientada a objetos e dinamicamente tipada.

Abstract

This dissertation presents the development of a client-server platform designed initially for audio and video streaming applications. This platform will evolve into a videoconference system as part of a future project. The platform allows audio, video and text communication from a point (the server) to several others points (the clients). One of the project innovations is the implementation Python Language, which is an interpreted, object-oriented and dynamically typed language.

Aos meus pais, Jair e Ana.

À Tatiane Carla de Azevedo, que tornou os desafios muito mais fáceis de serem vencidos, estando sempre a meu lado, nos bons e maus momentos.

Agradecimentos

Gostaria de agradecer a todas as pessoas que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Agradeço especialmente aos meus pais, Jair Gonçalves Junior e Ana Maria Martinelli Gonçalves e a minha irmã Letícia Martinelli Gonçalves, sem o apoio deles não teria conseguido ultrapassar todos os obstáculos que a vida nos impõe.

Também agradeço especialmente ao meu orientador, Prof Max Henrique Machado Costa, que além do seu apoio profissional, competente e aplicado, também me ofereceu a sua amizade atenciosa e compreensiva.

Agradeço a meus avós Jahyr Gonçalves, Floripes Jorge Gonçalves, Ana Lourdes Martinelli e Afonso Martinelli (*in memoriam*) pelos ensinamentos durante toda a vida, a Tatiane Carla de Azevedo pelo seu carinho, afeto e compreensão e a minha tia Cássia Denise Gonçalves pelas longas conversas e pelos anos de convivência.

Agradeço a Eric Magalhães Delgado, Fabrício C. de A. Oliveira e a todos amigos do Coml@b pelo companheirismo, orientação e apoio.

Agradeço a toda comunidade da FEEC e da Unicamp.

Agradeço a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro.

Campinas, Novembro de 2007.
Jahyr Gonçalves Neto

Sumário

Lista de Figuras

Glossário de Abreviaturas

CD – Compact Disk

CDMA - Code Division Multiple Access

DRM - Digital Radio Mondiale

DVB - Digital Video Broadcasting

DVD – Digital Video Disk

FFT - Fast Fourier Transform

FTP - File Transfer Protocol

GSM - Global System for Mobile Communications

HDTV – High Definition Television

IBOC - In Band On Channel

IDLE – Python's Integrated Development Environment

IP – Internet Protocol

IPC - Interprocess Communication

IPTV - Internet Protocol Television

ISDB - Integrated Services Digital Broadcasting

ISO - International Organization for Standardization

JVM - Java Virtual Machine

MP3 - MPEG-1 Audio Layer 3

MPEG - Moving Pictures Expert Group

PSF - Python Software Foundation

PSA - Python Software Activity

SVGA – Super Video Graphics Array

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

UML - Unified Modeling Language

VHLL - Very High Level Language

Capítulo 1

Introdução

A integração de serviços como dados, voz e imagem, impulsionada pelo sucesso e expansão da *World Wide Web*, aliada ao aumento no poder de processamento dos computadores pessoais, aumentaram a demanda por maior banda passante nas redes de comunicação de dados. Isto ocorre em todas as categorias de redes LANs passando pelas MANs até as WANs ao longo de toda a Internet.

O tráfego na rede sempre aumenta quando fazemos transferência de arquivos (*downloads*) ou executamos arquivos de áudio e vídeo da Internet. Embora este processo seja relativamente fácil e possibilite armazenar dados para posterior utilização, pode tomar um longo tempo. Atualmente, segundo Snow [1], *streaming* de vídeo está ganhando popularidade como uma alternativa ao *download*. Seja para uma aplicação educacional ou mesmo um programa de entretenimento, *streaming* de vídeo está se tornando um modo popular de se transmitir dados.

Streaming, que tem um significado semelhante a "fluxo contínuo", é uma forma de transmitir áudio e vídeo através da Internet (ou alguma outra rede), mas com uma particularidade especial: não é necessário baixar um arquivo inteiro para reproduzir o áudio ou o vídeo, já que permite ouvir e visualizar os arquivos enquanto se executa o *download*. A tecnologia de *streaming* é utilizada para tornar mais leve e rápido o *download* e a execução de áudio e vídeo na *web*.

A motivação para o desenvolvimento desta plataforma multimídia é desenvolver um projeto que envolva transmissão de áudio e vídeo utilizando a rede IP (*Internet Protocol*) impulsionado pela expansão dessa tecnologia e torná-la um projeto inicial para um sistema de videoconferência que deverá ser desenvolvido em uma etapa seguinte. Possivelmente o sistema deverá ser integrado ao projeto "*Infovia Municipal*" da cidade de Pedreira-SP e ao projeto KyaTera da FAPESP.

O projeto da Infovia é um convênio da prefeitura da cidade com a Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, que vai permitir acesso a diversos serviços de telecomunicações através de um equipamento denominado *Set-Top Box*. O KyaTera [2] é um projeto que possui como um de seus objetivos estimular a colaboração científica entre as instituições participantes e, a partir da tecnologia *fiber-to-the-lab*, desenvolver tecnologias e aplicações da Internet do futuro.

Durante o andamento do projeto, surgiu outra interessante possibilidade: direcionar o projeto para aplicações de *Internet Protocol Television* (IPTV). IPTV é um novo método de transmissão de áudio e vídeo (sinais televisivos) que utiliza a rede IP para o transporte dos sinais, além de possuir recursos de interatividade. IPTV vem se tornando popular nos anos recentes [3] como uma interação conveniente entre produtores de conteúdo e usuários de Internet.

O transporte de áudio e vídeo utilizando a rede IP é parte do sistema proposto nesta dissertação.

As vantagens da programação em linguagem Python neste projeto podem ser resumidas em três pontos principais:

1. Simplicidade [4]. A curva de aprendizagem é considerada suave. O tempo necessário para se aprender a linguagem é menor do que no caso de outras linguagens. Várias estruturas de dados úteis, como listas e dicionários, já estão embutidas na linguagem, de uma maneira que prioriza a facilidade do seu uso por parte do programador.

2. Código aberto e gratuito [4]. É *software* livre em dois sentidos. Não custa nada usar Python, ou incluir suas bibliotecas em uma aplicação. Além disso, o Python pode ser livremente modificado e redistribuído porque, embora sua linguagem tenha direitos reservados, sua disponibilidade está sob licença *open source* [5].

3. Grande disponibilidade de pacotes úteis para implementação de aplicações para redes e multimídia, além de pacotes para desenvolvimento de interfaces gráficas [4].

4. Fácil integração com outras plataformas como C/C++, Java e o .NET.

A linguagem Python é uma linguagem de altíssimo nível, interpretada, orientada a objetos com uma semântica dinâmica [6]. Suas estruturas de alto nível, combinadas com sua tipagem dinâmica tornam a linguagem muito atrativa para desenvolvimento de grandes aplicativos assim como para uso como linguagem de *script* ou de colagem. A sintaxe

simples do Python encoraja a reutilização de código simplificando a manutenção e a normalização de dados em módulos e pacotes distintos.

Esta linguagem, bem como seu código fonte, se encontra gratuitamente disponível na Internet podendo ser vasculhada por qualquer interessado sem ônus algum. Apesar disto, a linguagem Python é marca registrada do *Stichting Mathematisch Centrum* de Amsterdam [7].

A linguagem Python foi desenvolvida pelo holandês Guido Van Rossum, no final de 1990 [8]. Hoje a linguagem Python vem sendo utilizada por milhares de pessoas ao redor do mundo, unidos via Internet, formando a PSA (*Python Software Activity*).

Nesta dissertação apresentamos o desenvolvimento de uma plataforma multimídia para aplicações de *streaming* de áudio e vídeo baseado na linguagem Python. Este projeto tem por objetivo ser o núcleo de outro projeto que visa implementar um sistema de videoconferência.

No contexto apresentado, este trabalho apresenta as seguintes contribuições:

- Propõe-se um sistema de captura de áudio e vídeo.
- Propõe-se um sistema multimídia assíncrono baseado no modelo cliente-servidor. O sistema permite a comunicação de áudio, vídeo e texto.

1.1 Trabalhos Relacionados

Os trabalhos correlatos aqui citados motivaram de alguma forma o desenvolvimento desta dissertação. Dentre eles temos o tocador VLC [27], o *website* Youtube [36] e o *software* Skype [37].

1.1.1 VLC

O VLC é um tocador multimídia de código aberto. Possui suporte a vários formatos de vídeo e áudio. Além disso, pode ser usado como servidor de vídeo em redes de alta

velocidade. Por meio da biblioteca Ffmpeg o VLC é compatível com diversos formatos de áudio e vídeo, sendo um dos poucos programas capazes de rodar vários formatos de mídia sem o auxílio de um programa externo.

O VLC possui a capacidade de acessar o microfone e a *webcam* do computador para capturar e transmitir as seqüências de áudio e vídeo, característica principal do sistema proposto. Porém, para isso utiliza o protocolo RTP (*Real Time Protocol*) juntamente com o protocolo UDP (*User Datagram Protocol*). O sistema proposto utiliza apenas o UDP. A implementação do RTP juntamente com o UDP é um dos trabalhos futuros propostos para o sistema desenvolvido.

1.1.2 Youtube

O Youtube é um *website* popular que permite que seus usuários carreguem, assistam e compartilhem vídeos [35]. O usuário faz o *streaming* do conteúdo, que permite visualização dos dados ao longo em que chegam.

Com a análise dessa ferramenta, foi visto que o *streaming* é uma boa alternativa para a transmissão de dados, pois permite a visualização dos dados quase em tempo real.

1.1.3 Skype

O Skype é um programa gratuito que tem como finalidade proporcionar comunicação de voz e/ou vídeo de alta qualidade. Cada usuário possui um endereço skype. O programa permite estabelecer sessões de áudio conferência e trocar de mensagens de texto [33].

Skype é largamente considerado a ferramenta mais popular de telefonia IP pela alta taxa de chamada completadas e pela superior qualidade de som. [34]

O Skype possui uma arquitetura peer-to-peer (P2P) onde não existe centralização de processamento. Uma vez completada uma chamada, o computador que estabeleceu a comunicação entre as partes sai de operação, deixando apenas as partes se comunicando. O trabalho proposto é baseado no modelo cliente servidor, onde há a centralização de processamento no servidor que captura as seqüências de áudio e vídeo e as transmite para

os clientes requisitantes. Uma alternativa como trabalho futuro é usar a arquitetura do Skype no sistema proposto.

1.2 Organização do Trabalho:

O Capítulo 2 apresenta conceitos relativos ao sistema de transmissão de áudio e vídeo. O Capítulo 3 apresenta uma exposição teórica a respeito da Linguagem Python. O Capítulo 4 introduz a arquitetura do sistema proposto. O Capítulo 5 mostra os resultados experimentais. Finalmente o Capítulo 6 apresenta as conclusões do trabalho e sugestões para trabalhos futuros. Em Anexo, segue os códigos fonte das principais funções utilizadas no desenvolvimento do servidor de vídeo e do cliente de vídeo.

Capítulo 2

Fundamentação Teórica

Este capítulo trata de conceitos relacionados ao processo de captura e transmissão de áudio e vídeo na rede, tema deste trabalho de dissertação. Dentre os conceitos apresentados, estão os de multimídia (Seção 2.1); compressão de dados (Seção 2.2) necessários para o transporte de áudio e vídeo na rede; de transporte na rede (Seção 2.3); de arquitetura cliente-servidor (Seção 2.4); de *sockets* (Seção 2.5), utilizados para a comunicação; de protocolos de transporte (Seção 2.6); de *streamming* (Seção 2.7); de linguagem interpretada (Seção 2.8); de interpretador (Seção 2.9) e de *Buffers* (Seção 2.10).

2.1 Multimídia

Originalmente os computadores suportavam apenas processos relativos a pesquisas. Ao longo do tempo com o avanço da tecnologia os computadores se tornaram poderosos e são atualmente parte importante do nosso dia a dia. Estes poderosos computadores formam hoje um ambiente computacional [9], uma vez que tecnologia se move rapidamente para uma integração das facilidades de computação ao redor do mundo.

Multimídia é o modo de como a informação (áudio, vídeo, texto ou imagem) é manipulada pelo ambiente computacional. O termo se refere à combinação de duas ou mais mídias, onde pelo menos uma é discreta (texto, imagem) e outra é continua (áudio, vídeo). Literalmente, multimídia significa múltiplas mídias.

2.1.1 Sistema Multimídia

Um sistema multimídia é caracterizado por processar, armazenar, gerar, manipular e entregar informação multimídia. Sistemas multimídia possuem vários componentes como entrada (captura), dispositivos de armazenamento, redes de comunicação, sistemas de computador e dispositivos com *displays*. São eles:

- Dispositivos de Captura: Microfone, teclado, câmera de vídeo, mouse, etc
- Dispositivos de Armazenagem: CDs, DVDs.
- Rede de Comunicação *Ethernet*.
- Dispositivos com *display*: Monitores, HDTV, SVGA

Em qualquer sistema multimídia os componentes usados dependem da localidade da informação (fonte) a da localidade para a qual a informação é transportada (destino).

No presente trabalho o sistema manipula áudio, vídeo e texto.

2.2 Compressão de dados

A compressão de dados é o ato de reduzir o espaço ocupado por dados num determinado dispositivo. Essa operação é realizada através de diversos algoritmos de compressão, reduzindo a quantidade de bits necessários para representar um dado, sendo esse dado pode ser uma fonte de vídeo, áudio, voz, imagem, texto, ou mesmo um arquivo qualquer.

A operação de compressão pode ser realizada pela remoção de redundâncias da fonte de dados, que são informações que podem ser eliminadas de alguma forma sem comprometer o conteúdo da mídia original.

Além da economia de espaço em dispositivos de armazenamento, como discos rígidos, as técnicas de compressão foram essenciais no processo de digitalização das mídias tradicionais como música e fotografia. A compressão também é essencial para viabilizar

sistemas de comunicação digitais tais como televisão (DVB e ISDB), rádio (DRM e IBOC), telefonia celular (GSM, CDMA, etc.), entre outros.

Portanto a principal meta de um algoritmo de compressão digital é produzir uma representação mínima de um sinal que, quando descomprimido e reproduzido, é indistinguível do original. Em outras palavras, a mesma informação pode ser transmitida usando uma taxa ou quantidade de dados menor.

2.2.1 Compressão de Vídeo

A compressão de vídeo do presente trabalho se baseia no padrão MPEG. O nome MPEG é na verdade a sigla para um grupo de especialistas em imagens em movimento (*Moving Pictures Expert Group*), o qual foi criado pela Organização Internacional de Padrões (ISO) para a determinação de padrões para compressão e transmissão de áudio e vídeo.

O MPEG é um padrão de compressão genérico que foi designado a atingir taxas de compressão que variam de 50:1 até 200:1

O primeiro padrão estabelecido foi o MPEG-1 [10], o qual é composto por três diferentes versões, denominadas *layers*, com crescente complexidade e ganho de codificação. Na seqüência, surgiu o padrão MPEG-2 [10], cuja aplicabilidade se mostrou bem mais ampla. Estes dois padrões são os representantes da família MPEG no contexto do presente trabalho. Além deles, outros padrões foram desenvolvidos para diferentes aplicações, como o MPEG-4, o MPEG-7 e o MPEG-21.

2.2.2 Compressão de Áudio

A compressão de áudio do presente trabalho se baseia no MPEG 1 *Layer III.* Esta é a estrutura de codificação dos arquivos MP3 largamente usados na atualidade. O MP3 (*MPEG-1 Audio Layer 3*) foi um dos primeiros tipos de compressão de áudio com perdas quase imperceptíveis ao ouvido humano. O método de compressão com perdas consiste em retirar do áudio tudo aquilo que o ouvido humano normalmente não conseguiria perceber, devido a fenômenos de mascaramento de sons e de limitações da audição.

A codificação MPEG de áudio suporta 32, 44.1, e 48 KHz [9]. Em 16 bits por amostra, o áudio sem compressão necessitaria de 1,5 Mbps de *bitrate*. Com a compressão esse número é drasticamente diminuído. Para canais mono, fica entre 32 e 192 Kbps e para canais estéreos fica entre 128 e 384 Kbps.

2.3 Modelo TCP/IP

O modelo TCP/IP é uma arquitetura de rede em camadas, com o objetivo de padronização internacional dos protocolos de redes de computadores. Além disso facilita a comunicação entre diferentes tipos e modelos de computadores [15]. As camadas são:

- Camada Application: A camada de aplicação consiste de aplicativos que utilizam a rede
- Camada Transport: A camada de transporte fornece a entrega de dados de um extremo a outro da rede. Ela é a ligação entre a aplicação e a rede [32]
- Camada Internet: A camada Internet define o datagrama e manipula o direcionamento desses datagramas.
- Camada Phisical. O modelo faz uso dos padrões fornecidos por organizações

2.4 Transporte na Rede

Uma parte das informações que estão sendo trocadas pela Internet é devido ao tráfego de voz e vídeo. Uma rede baseada em pacotes, tal como a Internet, envia dados através de pacotes na rede. Os pacotes contêm uma identificação do destino e são roteados independentemente uns dos outros, usando uma técnica conhecida como serviço de melhor esforço (*best-effort*). Este serviço pode não atender aos requisitos do tráfego de voz porque o atraso e a largura de banda não são fixos. Apesar desta restrição, existe uma série de

soluções para se usar a rede IP como um meio de transporte de tráfego de voz e vídeo, pois a rede IP pode fornecer vários benefícios, como o baixo custo e uma maior eficiência no uso da banda disponível.

2.5 Arquitetura Cliente – Servidor

A arquitetura cliente-servidor é uma arquitetura de rede, onde existem dois módulos básicos na rede: o servidor e os clientes. O servidor, de *hardware* ou de *software* é a parte que é responsável por servir os clientes com o que lhe é solicitado. Cliente é a parte que solicita serviços ou informações que estão contidas no servidor.

O servidor provê um serviço que é requisitado por um ou mais clientes, utilizadores do serviço. Tem como propósito esperar por requisições de clientes, servir esses clientes e então esperar por mais requisições [11]. Por outro lado, os clientes estabelecem contato com um servidor predeterminado para uma requisição particular, enviam quaisquer dados necessários e aguardam a resposta do servidor ou completando a requisição ou indicam o motivo da falha.

Enquanto servidores processam uma ou mais requisições, clientes fazem apenas uma requisição por vez, recebem este serviço e assim concluem a transação. Um mesmo cliente pode efetuar outra requisição posteriormente, porém será considerada uma transação separada da primeira requisição.

Um exemplo comum da arquitetura "cliente-servidor" é ilustrado na Fig. 2.1. Um ou mais usuários clientes estão requisitando informações de um servidor ao longo da rede LAN (*Local Area Network*). Este é apenas um exemplo de arquitetura, dentre vários outros existentes.

A arquitetura cliente-servidor é usada extensivamente em *softwares* modernos. Seu uso garante alta velocidade de execução, que possibilita dividir carga computacional em várias máquinas e suportar várias interfaces de usuários referentes a diferentes sistemas de computadores [12]. Segundo Davey e Tatnall [13] a arquitetura cliente-servidor domina no desenvolvimento de aplicações de rede.

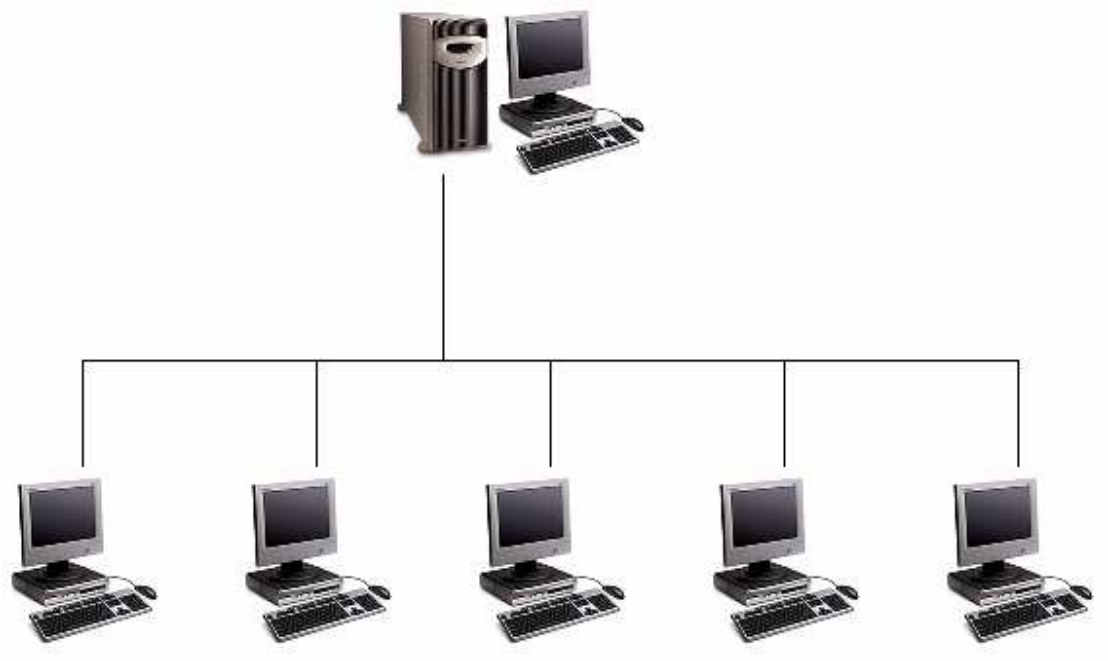

Fig. 2.1: Exemplo da Arquitetura Cliente-Servidor

Portanto uma rede cliente-servidor é uma rede distribuída que consiste de um sistema de desempenho rápido, o servidor, e de vários sistemas de menor performance, os clientes. O servidor é uma unidade de registro central assim como é o único fornecedor de conteúdo e serviço. Um cliente apenas faz requisição de conteúdo ou executa os serviços [14].

2.5.1 Arquitetura Cliente-Servidor de *Hardware*

Um exemplo comum de servidor de hardware é o servidor de impressão. Os servidores de impressão processam trabalhos de impressão e os enviam para a impressora conectada à rede de computadores. Esse servidor é acessível pela rede e as máquinas clientes podem enviar requisições de impressão.

Outro exemplo de servidor de *hardware* é o servidor de arquivos. São tipicamente máquinas com alta capacidade de armazenamento a qual é acessível remotamente para os clientes. A máquina cliente usa o disco rígido da máquina servidora como se fosse o seu.

2.5.2 Arquitetura Cliente-Servidor de Software

Os serviços básicos providos por servidores de *software* incluem execução de programas, transferência de dados, atualizações e outros tipos de manipulação de dados. Um dos servidores de *software* mais comuns hoje em dia são os servidores *Web*. Uma máquina corporativa é configurada com páginas *Web* e/ou aplicações *Web* e então o servidor é inicializado. O trabalho desse servidor é esperar requisições, aceitar essas requisições e enviar páginas da *Web* para os clientes. O propósito desses servidores seria "servir para sempre", porém não é possível tal ação. Eles servem até que sejam interrompidos por desligamento da máquina ou queda de energia.

Servidores de banco de dados é outro tipo de servidores de *software*. Eles aceitam requisições de clientes para armazenar ou recuperar de dados. Eles executam o serviço e então esperam por mais trabalho. São também designados a servir "para sempre".

O trabalho proposto por esta dissertação é baseado neste tipo de arquitetura cliente-servidor para aplicações de fluxo de áudio e vídeo.

2.6 Sockets

Sockets são interfaces de comunicação bidirecional entre processos e são representados como descritores de arquivos que permitem a comunicação entre processos distintos na mesma máquina ou em máquinas distintas, através de uma rede (*Ethernet*). Os *sockets* são tipicamente utilizados para implementar aplicações que envolvem comunicações em redes TCP/IP (*Transmission Control Protocol/Internet Protocol)*, mas também podem ser utilizados em comunicações entre processos no interior de um mesmo computador.

Sockets são estruturas de dados em redes de computadores que incorporam o conceito de "comunicação de ponto final" [11]. Aplicações de redes devem criar *sockets* antes de iniciar qualquer tipo de comunicação.

Essas estruturas foram desenvolvidas nos anos 70 na Universidade da Califórnia. Foram originalmente criados para aplicações na mesma máquina onde permitiam que um

processo em execução se comunicasse com outro processo também em execução. Isto é conhecido como IPC (*Interprocess Communication*).

Um detalhe interessante é que os *sockets* foram inventados antes de existirem as redes de computadores. Isso comprova que *sockets* nem sempre foram utilizados apenas para aplicações em redes.

Quando as redes de computadores se tornaram uma realidade, pesquisadores acreditaram que comunicação entre processos pudessem ocorrer em diferentes máquinas. Estes *sockets* mais novos possuem nome próprio de família, AF_INET, ou "endereço de família: Internet".

2.6.1 Endereços *Sockets*: Par Endereço IP e Porta de Comunicação

Se compararmos um *socket* a um telefone, um aparelho com infra-estrutura que permite comunicação, então o endereço IP e o número da porta de comunicação serão como a combinação de código de área e do número do telefone. Apenas dispor do *hardware* e da habilidade para comunicação não significa nada se não sabemos para quem e para onde "discar"

Um endereço de *Internet* é formado por um par de informações contendo o endereço IP da máquina e um número de porta de comunicação. Uma porta é um número de 16 bits que denota um terminal de comunicação dentro de um programa. A combinação de endereço IP e de número de porta identifica unicamente uma conexão de rede em um processo [15]. Esse par de informação é essencial para a comunicação em rede.

A numeração valida das portas de comunicação varia de 0 a 65535, porém os números de portas menores que 1024 são reservados para uso especifico do sistema.

2.6.2 Formas de Comunicação

Existem dois diferentes estilos de comunicação entre *sockets.* O primeiro tipo é orientado à conexão, o que basicamente significa que uma conexão deve ser estabelecida entre as partes antes de ocorrer a comunicação. O segundo tipo é não orientado à conexão, onde não há a necessidade de estabelecer uma conexão antes de iniciar a comunicação.

2.6.2.1 Comunicação Orientada à Conexão

Comunicação Orientada à Conexão oferece entrega de dados seqüenciada, confiável, sem duplicações. Isto significa que cada mensagem que é quebrada em múltiplos pedaços tem garantia de chegada ao seu destino sendo reconstruída de forma correta e entregue à aplicação que a espera.

O protocolo que implementa esse tipo de comunicação é o TCP (*Transmission Control Protocol*). Para criar *sockets* TCP em Python, devemos usar SOCK_STREAM como tipo de *sockets* que queremos criar.

2.6.2.2 Comunicação Não Orientada à Conexão

Este tipo de comunicação é baseado em datagramas. Não é necessário estabelecer uma conexão entre as partes para se iniciar a comunicação. Porém, não há garantias de seqüenciamento, confiabilidade ou não duplicação no processo de entrega de dados.

As mensagens que são entregues usando datagramas podem ser comparadas ao serviço de correios. Cartas e pacotes podem não chegar na ordem em que foram enviados. De fato, podem até não chegar. Contribuindo para esses problemas, em redes de computadores, é possível a duplicação de mensagens.

Com tantos pontos negativos, porque usar datagramas? Pois todas as garantias providas pelos *sockets* orientados à conexão geram muito *overhead* para estabelecer e manter a conexão. Nos datagramas o *overhead* é muito menor pois não há necessidade de estabelecimento de conexões e confirmações de entrega e recebimentos das mensagens. Assim sendo possuem uma melhor performance na rede.

O protocolo que implementa esse tipo de comunicação é o UDP (*User Datagram Protocol).* Para criar *sockets* UDP em Python, devemos usar SOCK_ DGRAM como tipo de *sockets* que queremos criar.

No presente trabalho usamos a comunicação não orientada a conexão para um melhor desempenho do sistema com relação a banda e atraso.

2.7 Protocolos de Transporte

Existem duas formas de estabelecimento de uma ligação cliente-servidor. Uma é orientada à conexão, a outra não. O TCP é o protocolo de transporte orientado à conexão em que o cliente estabelece uma conexão com o servidor e ambos trocam múltiplas mensagens de tamanhos variados, sendo a aplicação do cliente quem termina a sessão. Já o protocolo UDP não é orientado à conexão. Nele o cliente constrói uma mensagem e a envia num pacote UDP para o servidor, que responde sem estabelecer uma conexão permanente com o cliente.

2.7.1 Protocolo Datagrama de Usuário (User Datagram Protocol – UDP)

O UDP permite que os dados sejam transferidos pela rede a um custo relativamente baixo. Este custo está associado com a entrega não confiável de dados. Não há método no protocolo para verificar se os dados alcançaram o destino exatamente como foram enviados. Os dados podem ser perdidos, duplicados ou chegar sem validade.

Entretanto, essas limitações não tornam o UDP sem uso. Em aplicações de fluxo de dados em tempo real como o *streaming* de áudio e vídeo em uma comunicação bidirecional, por exemplo, é mais importante uma baixa latência do que garantia de entrega. De fato, é até preferível a perda de pacotes ao atraso ocasionado pela troca de mensagens associada a uma retransmissão de pacotes.

2.7.2 Protocolo de Controle de Transmissão (Transmission Control Protocol – TCP)

O TCP verifica se os dados são entregues em ordem e sem alteração. Associado a esse recurso está uma despesa extra na geração e na manutenção de uma conexão.

O TCP fornece à transmissão um fluxo de bytes confiável e orientado a conexões. A confiabilidade do TCP resulta da inclusão de um *checksum* em cada pacote de dados transmitido. Na recepção, um *checksum* é gerado e comparado ao *checksum* incluído no cabeçalho do pacote de dados. Se os *checksum* não combinarem, o receptor comunica esse

fato ao emissor e os dados são automaticamente enviados novamente. O TCP é considerado do tipo orientado a conexões porque os dois terminais conversam amigavelmente antes que a transmissão de dados possa começar. Esse diálogo garante ao emissor que o receptor está ativo e pronto para aceitar os dados.

A utilização do protocolo TCP pode ser viável em uma comunicação bidirecional em tempo real quando há garantia de largura de banda para a comunicação. Neste cenário praticamente não haverá retransmissões devido a perdas de pacote, o que minimiza o atraso que é tão prejudicial em aplicações de vídeo conferência.

2.8 *Streaming*

O *streaming* é uma forma de transmitir áudio e vídeo através da Internet ou alguma outra rede. A tecnologia de *streaming* torna mais leve e rápido o *download* e a execução de áudio e vídeo na *web*.

Para exibir um conteúdo multimídia na rede, o *streaming* evita termos que descarregar o arquivo inteiro em nosso computador. Ele permite a reprodução e visualização do conteúdo ao mesmo tempo em que se faz o *download.* Além disso, o uso de *streaming* otimiza o consumo de banda na rede, uma vez que a transmissão pode ser realizada a uma taxa constante bem mais baixa que a taxa normalmente utilizada no *download* de um arquivo. No *download* de arquivos, como nas aplicações de FTP (*File Transfer Protocol*), toda a banda disponível é consumida se não houver nenhuma limitação no tráfego.

O *streaming* funciona da seguinte maneira. Inicialmente o computador cliente se conecta ao servidor e este, começa a lhe enviar o fluxo de bits. À medida que recebe o arquivo, o cliente constrói um *buffer* onde começa a salvar informações. Quando se enche o *buffer* com uma pequena parte do fluxo, o cliente inicia a reprodução ao mesmo tempo que o *download* é realizado, de modo que quando o fluxo termina, também finaliza-se a visualização. Se em algum momento a conexão sofre atrasos, se utiliza a informação que existe no *buffer*, de modo que se pode agüentar um pouco esse atraso. Porém, se a comunicação cai durante muito tempo, o *buffer* se esvazia e a execução do arquivo pára até

que se restaure o sinal. Resumindo, o *buffer* é fundamental, pois compensa pequenas variações nos tempos de chegada dos pacotes (*jitter*) que poderiam prejudicar significativamente a decodificação e a reprodução do fluxo.

2.9 Linguagem Interpretada

Uma linguagem de computador interpretada (também chamada linguagem de *script*) é uma linguagem de programação executada em um interpretador. Por esse motivo, os *scripts*, que são as instruções formais escritas com as linguagens interpretadas, são diferentes de programas de computador; enquanto programas de computador são convertidos em formatos executáveis binários para serem usados, *scripts* mantêm seu formato e são interpretados comando a comando cada vez que são executados. As linguagens interpretadas foram criadas para simplificar o ciclo tradicional de programação "edição – compilação – ligação – execução". Entretanto, nada impede que uma linguagem interpretada seja compilada para código de máquina ou que uma linguagem de programação tradicional seja interpretada sem compilação.

2.10 Interpretador

Interpretadores são programas de computador que lêem um código fonte de uma linguagem de programação interpretada e os convertem em código executável. Seu funcionamento pode variar de acordo com a implementação. Em muitos casos o interpretador lê linha-a-linha e converte em código objeto à medida que vai executando o programa. Linguagens interpretadas são mais dinâmicas por não precisarem escrever-compilar-testar-corrigir-compilar-testar-distribuir, e sim escrever-testar-corrigir-escrever-testar-distribuir.

2.11 Buffers

Os *buffers* são áreas de memória criadas pelos programas para armazenar dados que estão sendo processados nas aplicações. Na utilização de *buffers*, deve-se tomar cuidados para que não ocorra um fenômeno conhecido como *buffer overflow* (quando o programa recebe mais dados do que está preparado para armazenar no buffer) ou um *buffer underflow* (quando os dados recebidos são poucos e não permitem a reprodução).

Capítulo 3

Linguagem Python

Este capítulo apresenta uma visão geral da linguagem Python utilizada no trabalho. O capítulo está dividido em 4 seções. A Seção 3.1 descreve em detalhes a linguagem Python; A Seção 3.2 apresenta a ferramenta de desenvolvimento gráfico Tkinter; A Seção 3.3 trata da integração da linguagem com outras plataformas. A Seção 3.4 mostra aplicações que utilizam a linguagem Python.

3.1 A linguagem Python

Python é uma linguagem de programação de altíssimo nível (VHLL – Very High Level Language), criada pelo holandês Guido Van Rossum sob o ideal de "Programação de Computadores para todos" [16]. Este ideal fez com que o desenvolvimento de Python tivesse sempre em mente a liberdade (gratuita, código aberto), disponibilidade (Python é compatível com os sistemas Windows, Linux e Macintosh, além de rodar em Palms, celulares, etc) e principalmente a clareza de sintaxe, que hoje é responsável pela alta produtividade só conseguida com Python.

Serão apresentadas a seguir algumas características da linguagem.

3.1.1 Python como Linguagem Interpretada

Linguagens de programação são freqüentemente classificadas como compiladas ou interpretadas. Nas compiladas, o texto (ou código-fonte) do programa é lido por um programa chamado compilador, que cria um arquivo binário, executável diretamente pelo hardware da plataforma-alvo.

Em contrapartida, programas escritos em linguagens interpretadas, como a linguagem Python, não são convertidos em um arquivo executável. Eles são executados utilizando um outro programa, o interpretador, que lê o código-fonte e o interpreta diretamente, durante a sua execução.

Para executar programas escritos em Python podemos utilizamos o interpretador interativo IDLE (*Python's Integrated Development Environment*) que acompanha a distribuição oficial da linguagem Python ou utilizar o *prompt* de comando do próprio Windows.

Para executar um programa escrito e armazenado num arquivo com o nome *hello.py* na pasta raiz do Python, a partir do interpretador interativo IDLE, utiliza-se o comando *import nome_do_arquivo* como é mostrado na Fig. 3.1:

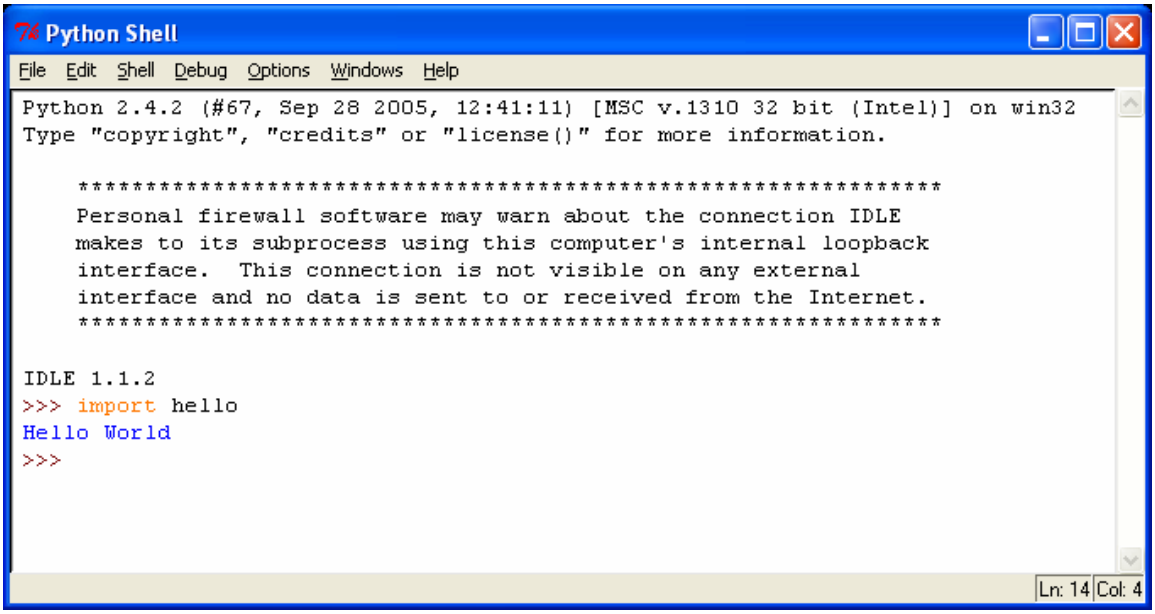

Fig. 3.1: Interpretador Interativo IDLE

O programa *hello.py* tem como função imprimir a mensagem de texto "*Hello World*"na tela.

No caso do interpretador, a extensão .py não é incluída no comando *import*, apenas o nome principal do programa. Na próxima seção veremos mais características do interpretador interativo IDLE.

No caso de invocarmos o *prompt* de comando do Windows para executar o mesmo programa *hello.py*, utiliza-se o comando *python nome_do_arquivo.py* como é mostrado na Fig. 3.2. No caso do *prompt*, a extensão *.py* é incluída no comando *python.*

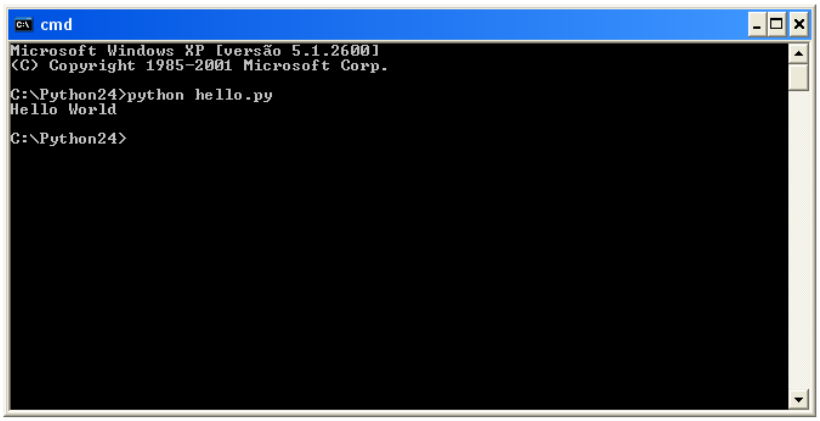

Fig. 3.2: *Prompt* de comando do Windows

Note que o programa que executamos diretamente é o interpretador Python (o *prompt*), não havendo necessidade de um compilador. Passamos como parâmetro o próprio nome do arquivo com código-fonte *hello.py*. Não há o passo de geração de executável; o interpretador transforma o programa especificado à medida que é executado. O nome do programa é sempre considerado um primeiro parâmetro. Existe a possibilidade de passar mais parâmetros, desde que sempre depois do nome do programa a ser interpretado.

3.1.2 Interpretador Interativo

Python está disponível para várias plataformas. Cada plataforma possui um pacote específico para instalação, que normalmente inclui um grande número de bibliotecas de código e um programa executável python. O interpretador interativo IDLE que acompanha o pacote Python para a plataforma Windows pode ser visto na Fig. 3.1.

Podemos executar o interpretador interativamente, no modo *Shell*. Para isso entramos com o código-fonte diretamente para avaliar expressões e comandos, sem criar um arquivo separado.

O símbolo >>> indica que o interpretador está aguardando um comando. O *shell* Python oferece uma excelente forma de testar código como vemos na Fig. 3.3.

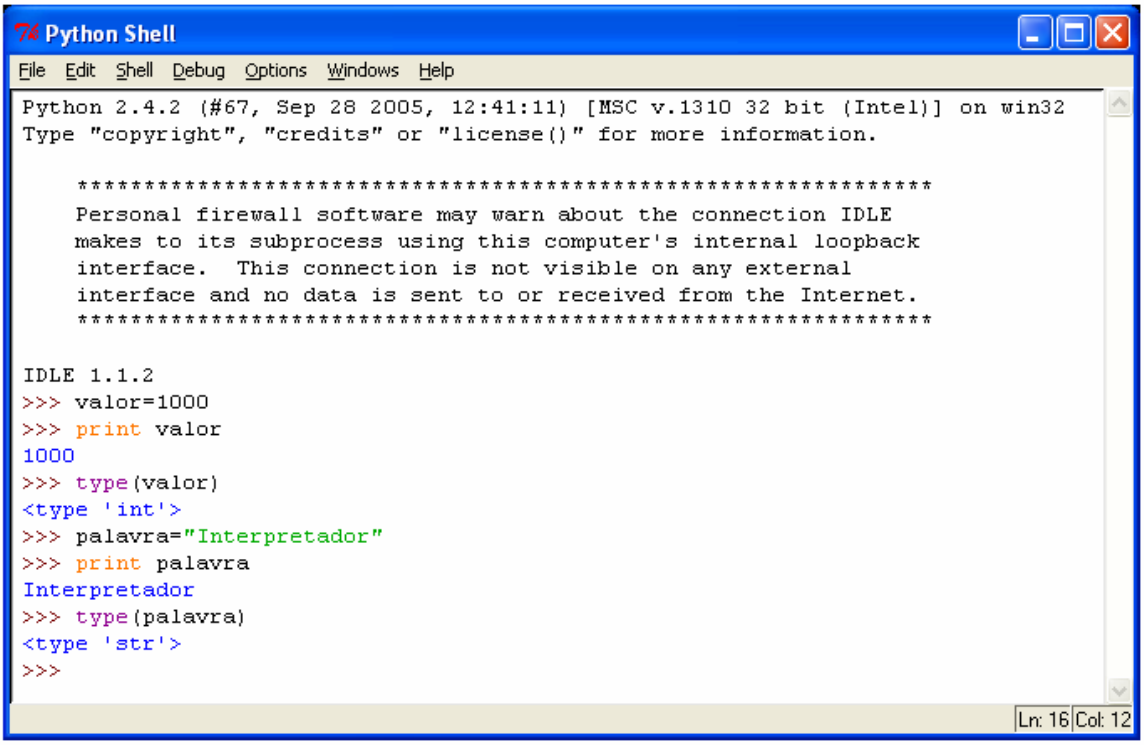

Fig. 3.3: Utilização Interativa do Interpretador.

Neste exemplo introduzimos alguns aspectos básicos: atribuímos um valor a variável 'valor'. Depois verificamos o valor atribuído (número 1000) invocando a instrução *print,* que exibe valores e conteúdo de variáveis na tela e por fim verificamos o seu tipo ('*int*' de inteiro) com a instrução *type*, que retorna o tipo da variável, atribuída dinamicamente. Repetimos esse processo com a variável 'palavra' que recebe a palavra "Interpretador" como parâmetro e exibe o tipo '*str*' de *string*.

Dessa forma é possível digitar comandos linha a linha, observando a cada passo o como o computador interpreta e executa esses comandos.

O interpretador interativo do Python pode executar comandos Python arbitrários. Isso é útil para depuração, desenvolvimento rápido e execução de testes.

A forma interativa de executar o Python é conveniente, no entanto, não armazena os códigos digitados, servindo apenas para testes e procedimentos simples. Para programas mais complexos, o código-fonte é normalmente escrito e armazenado em um arquivo.

Para criar um arquivo contendo um programa, basta usar editor de texto comum ou o que acompanha a distribuição Python, digitar os comandos e salvar na pasta raiz do Python com a extensão "*.py*". Em Python, um arquivo contendo instruções da linguagem é chamado de módulo e poderá ser usado dentro de outros módulos. Nos casos mais simples pode-se usar um único módulo, executando-o diretamente, no entanto, é interessante saber que é possível subdividir o programa em arquivos separados e facilmente integrar as funções definidas neles. Veremos mais sobre este assunto na seção 3.1.7.

Para chamar um módulo no interpretador interativo, basta usar a instrução "*import*" e passar como parâmetro o nome do arquivo que contém o código a ser executado, como fizemos na Fig. 3.1 com o arquivo previamente armazenado *hello.py*.

3.1.3 Tipagem dinâmica

Um dos conceitos básicos em programação é a variável, que é uma associação entre um nome e um valor. Na maioria das linguagens, temos que declarar o tipo de uma variável, antes de associá-la. Em Python, não precisamos declarar variáveis.

Python possui o que é conhecido como tipagem dinâmica: o tipo ao qual a variável está associada pode variar durante a execução do programa. Não quer dizer que não exista tipo específico definido (a chamada tipagem fraca), embora em Python não o declaremos explicitamente, as variáveis sempre assumem um único tipo em um determinado momento.

Tipagem dinâmica, além de reduzir a quantidade de planejamento prévio (e digitação) para escrever um programa, é um mecanismo importante para garantir a simplicidade e flexibilidade das funções Python. Como os tipos dos argumentos não são explicitamente declarados, não há restrição sobre o que pode ser fornecido como parâmetro.

3.1.4 Controle de bloco por indentação

Funções Python não têm marcações explícitas de começo e fim como "*begin"* e "*end"* ou chaves. O único delimitador é o sinal de dois-pontos (":") e a própria indentação do código. Blocos de código (funções, comandos if, laços for, etc.) são definidos pelo uso de uma indentação visual. Cada linha lógica possui um valor chamado de nível de identação que é o número da primeira coluna não branca da linha

Indentar o código inicia um bloco, remover a indentação termina o mesmo, ou seja, a primeira linha não indentada após o bloco está fora da função. Não há sinais específicos ou palavras-chave. Isso significa que espaço em branco é significativo e deve ser consistente.

Esta propriedade faz com que o código seja muito claro e legível, afinal, garante que a indentação esteja sempre correta, porém requer costume e um controle mais formal.

3.1.5 Tipos de alto nível

Além dos tipos básicos (inteiros, números de ponto flutuante, booleanos), alguns tipos pré-determinados em Python merecem atenção especial:

List: (Listas):

Como um vetor em outras linguagens, a lista é um conjunto (ou seqüência) de valores acessados (indexados) por um índice numérico, inteiro, começando em zero. A lista em Python pode armazenar valores de qualquer tipo.

Tuple (Tuplas):

Tuplas são também seqüências de elementos arbitrários; comportam-se como listas com a exceção de que são imutáveis. Objetos imutáveis possuem a característica que uma vez criados não podem ser alterados. A única forma de modificar um objeto imutável é substituir o objeto antigo por um novo objeto.

Strings:

A cadeia de caracteres, uma forma de dado muito comum; a string Python é uma seqüência imutável, alocada dinamicamente, sem restrição de tamanho.

Dictionaries (Dicionários):

Dicionários são seqüências que podem utilizar índices de tipos variados, bastando que estes índices sejam imutáveis (números, tuplas e *strings*, por exemplo). Os dicionários são conhecidos em outras linguagens como *arrays* associativos ou *hashes*.

File (Arquivo):

Python possui um tipo pré-definido para manipular arquivos. Este tipo permite que seu conteúdo seja facilmente lido, alterado e escrito.

Class (Classes):

As classes são estruturas especiais que servem para apoiar programação orientada a objetos e determinam um tipo customizado com dados e operações particulares. As instâncias são as expressões concretas destas classes. A orientação a objetos em Python será descrita em maiores detalhes na seção 3.1.6.

3.1.6 Orientação a Objetos

Python é uma linguagem orientada a objetos, um paradigma que facilita o controle sobre a estabilidade dos projetos quando estes começam a tomar grandes proporções.

A orientação a objetos é uma forma conceitual de estruturar um programa. Ao invés de definirmos variáveis e criarmos funções que as manipulam, definimos objetos que possuem dados próprios e ações associadas. O programa orientado a objetos é resultado da ”colaboração” entre estes objetos.

Em Python, todos os dados podem ser considerados objetos: qualquer variável (mesmo as dos tipos básicos e pré-definidos) possui um valor e um conjunto de operações que pode ser realizado sobre este.

Como a maior parte das linguagens que são consideradas “orientadas a objeto”, a linguagem Python oferece um tipo muito especial para definir objetos customizados: a

classe. Python suporta também funcionalidades comuns na orientação a objetos: herança, herança múltipla, polimorfismo, reflexão e introspecção.

3.1.7 Módulos e o comando *"import"*

3.1.7.1 Módulos

Um arquivo contendo código Python é denominado módulo. Na grande maioria das ocasiões utilizamos um ou mais módulos Python em combinação, pois como vimos na seção 3.1.2, o interpretador interativo é adequado para realizar experimentos curtos, mas não para escrever código com muitas instruções.

Um módulo Python consiste de código-fonte contido em um arquivo denominado da forma "*nome.py*", que pode conter variáveis, funções e classes. Para fins de nomenclatura, qualquer um destes elementos contidos em um módulo é considerado um atributo do módulo.

Python, através do módulo, oferece excelentes mecanismos para modularizar o código-fonte. Esta modularização pode ter diversas motivações, o programa pode ser grande demais, ter sub-partes reusáveis que devem ser separadas, ou ainda necessitar de atributos existentes em módulos escritos por terceiros.

Há um grande número de módulos desenvolvidos para a linguagem Python que podem ser utilizados como base para o desenvolvimento de novas ferramentas [17]. Existem também os módulos *built-in*, que implementam importantes funcionalidades, tais como manipulação de arquivos, chamadas de sistema, *sockets*, entre outras.

Um módulo em Python é semelhante a uma biblioteca na linguagem C/C++.

3.1.7.2 Importando módulos e atributos de módulos

A instrução básica para manipulação de módulos é a instrução *"import"*. Da mesma forma que usamos esta instrução para importar um módulo no interpretador interativo, podemos usá-la para importar atributos (classes, funções, variáveis) de um módulo para outro módulo no próprio código fonte do programa. Desta forma o programa fica

estruturado, podendo reutilizar códigos já existentes. Esta é uma das vantagens da linguagem Python.

Esta característica de reutilização de códigos entre os módulos exige que os mesmos devem estar contidos na pasta raiz do Python.

Existem duas formas de usar o *import*. Na primeira forma, todo atributo importado de um módulo deve levar o nome do módulo de origem como prefixo, separados por um ponto, como vemos no editor de texto que acompanha o Python na Fig. 3.4. É importante esta observação, pois ao usar a forma "*import* módulo", acessamos os atributos do módulo usando esta sintaxe.

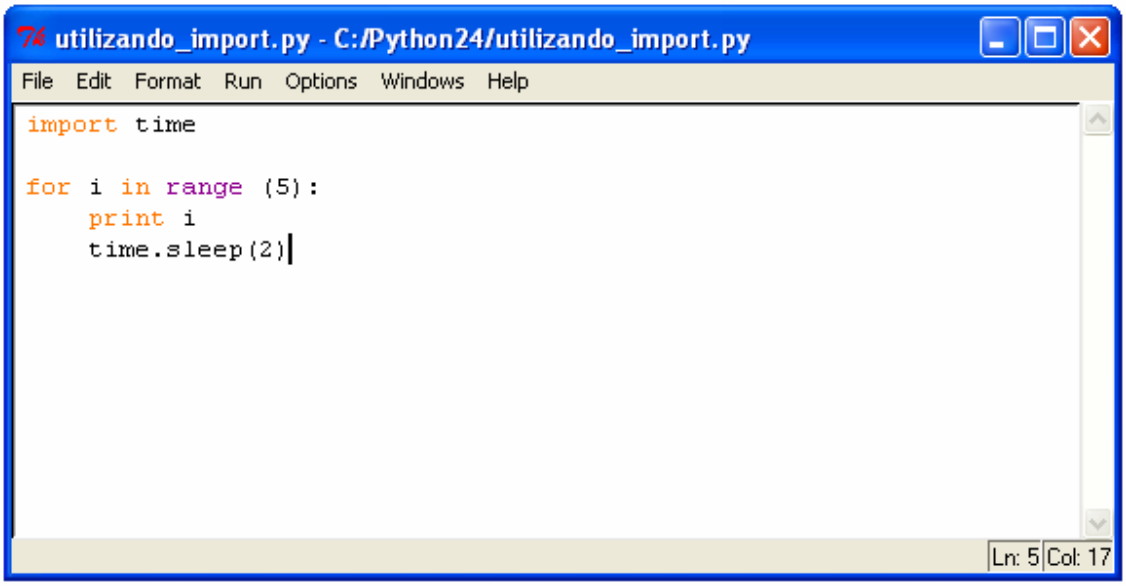

Fig. 3.4: Primeira forma da instrução import.

Neste exemplo criamos um módulo chamado "*utilizando_import.py*". Dentro dele, importamos outro módulo chamado "*time*". Observe que a extensão .py não é incluída no comando *import* apenas o nome principal do módulo. A partir daí, utilizamos os atributos do módulo *time* colocando o nome do módulo antes da função utilizada, separados por ponto. Neste caso utilizamos o formato *time.sleep()* para utilizar a função *sleep()*.

Existe a segunda forma do comando *import* que funciona de forma diferente. Ao invés de importar o módulo inteiro, o que nos obriga a usar as funções prefixadas pelo

nome do módulo de origem, este formato importa um atributo do módulo. Isto se da modificando a primeira linha do exemplo anterior. A Fig. 3.5 mostra essa modificação.

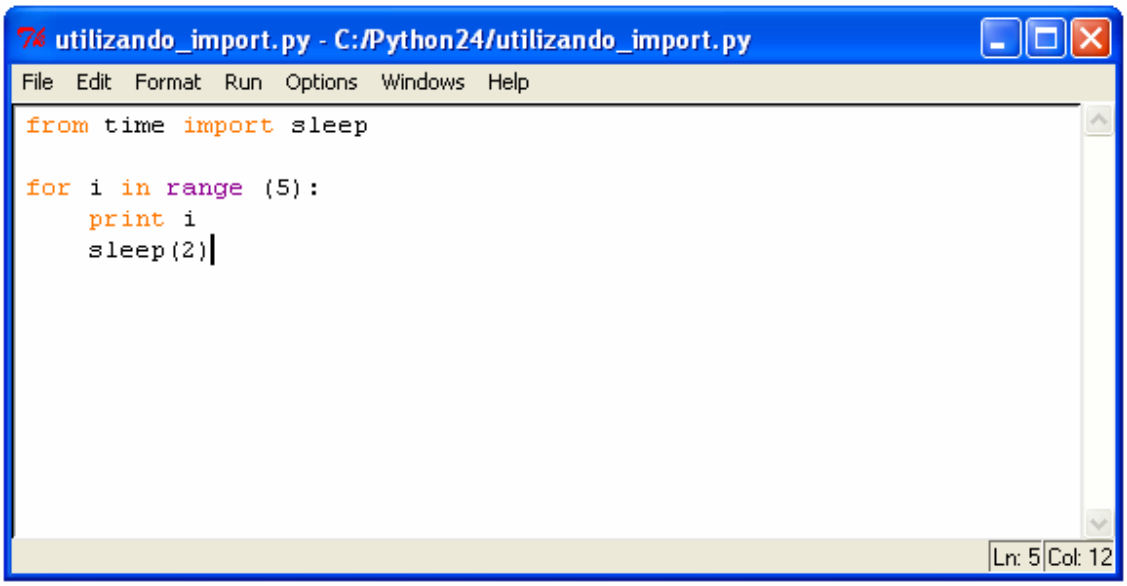

Fig. 3.5: Segunda forma da instrução import.

A função *sleep()* é importada diretamente, podendo ser usada sem prefixo ou seja ela pode ser utilizada localmente.

Executando qualquer um dos dois exemplos anteriores, serão impressos na tela os números de zero a quatro com dois segundos de intervalo entre eles.

3.1.7.3 Funções Úteis

Há algumas funções pré-definidas no interpretador bastante úteis quando lidamos com módulos e seus atributos

- dir (nome_módulo) retorna uma lista dos nomes dos atributos contidos em um módulo, o que permite que você descubra interativamente quais símbolos e funções o compõem.
- help (nome_módulo) retorna uma lista dos nomes dos atributos do módulo, contendo uma descrição detalhada de cada dos atributos.

3.2 A Ferramenta de Desenvolvimento Gráfico *Tkinter*

A maioria das aplicações profissionais interage com os usuários por uma Interface Gráfica de Usuário (*Graphical User Interface* - GUI) [18]. As GUIs se tornaram um importante modo de interação com os *softwares* [31]. Desenvolver uma aplicação GUI é similar a um artista produzir uma pintura. Convencionalmente, existe uma única tela onde o artista coloca todo seu trabalho.

A GUI é normalmente programada por um kit de ferramentas, onde há uma biblioteca de funções que implementam controles (conhecidos como *widgets*). Os controles são botões, campos de entrada de texto, menus, etc. As ferramentas da GUI permitem desenvolver controles de uma forma coerente, mostrando-os na tela, interagindo com o usuário, recebendo dados de entrada via periféricos como o teclado e o *mouse.*

Python provê vários kits de ferramentas de desenvolvimento de GUI's. A maioria delas trabalha em várias plataformas (multi-plataforma). Porém, a mais comum e popular é a Tkinter. Tkinter utiliza a ferramenta multi-plataforma Tk que é usada em outras linguagens interpretadas como Tcl e Pearl.

Tkinter acompanha a distribuição oficial do interpretador Python. No Windows distribuição oficial do Python também inclui os componentes Tcl/Tk necessários para o funcionamento de Tkinter.

Com a flexibilidade de desenvolvimento de GUI da ferramenta Tk, aliado a simplicidade de linguagens interpretadas, Tkinter provê ferramentas para um rápido desenvolvimento de interfaces gráficas.

3.2.1 Fundamentos Tkinter

O módulo Tkinter torna fácil a construção de aplicações gráficas. Simplesmente importamos o módulo Tkinter, criamos, configuramos e posicionamos os controles que desejamos. A aplicação se torna dirigida a eventos, o que significa que o usuário interage com os controles, causando eventos, e a aplicação responde via funções desenvolvidas para lidar com cada controle.

A ferramenta possui funções como *Label* ou *Button*, que quando chamadas, criam os respectivos controles, como entrada de dados e botões e os retorna como resultados.

Tkinter coloca os controles diretamente na janela de aplicação principal. Se desejarmos, temos a opção de especificar janelas principais e janelas secundárias. Não é necessário ligar variáveis aos controles *Labels* ou *Buttons*. Seus nomes já especificam a configuração dos controles. Devemos apenas chamar um método conhecido como *pack* (empacotar) em cada controle passando o controle da geometria dos controles para um *layout manager* (administrador de planejamento) conhecido como *packer* (empacotador). Um administrador de planejamento é um componente invisível cujo trabalho é posicionar controles dentro de outros controles (conhecidos como containeres), lidando com questões de planejamento geométrico.

3.3 Integração com outras plataformas

A linguagem Python possui recursos para integração com outras plataformas de desenvolvimento como, por exemplo, as plataformas C/C++ , Java da Sun e o .NET da Microsoft.

3.3.1 Integração com a linguagem C/C++

A ligação entre as linguagens C/C++ e Python pode servir tanto para utilizar bibliotecas binárias já existentes como para criar módulos em aplicações que exigem um desempenho além do possível na linguagem Python. Existem pelo menos três formas de se desenvolver as ligações:

- Escrever as ligações na mão

Pode-se escrever as ligações em C/C++, que é considerado um ato de baixo grau de complexidade utilizando a *Python C/API (Application Programmer's Interface*). A API define um conjunto de funções para a ligação entre as linguagens [19].

Esse tipo de ligação garante que o código que o programador desenvolveu fique exatamente a seu gosto.

- Usar geradores de ligações

Existem algumas ferramentas geradoras de ligações. Uma das ferramentas mais populares é o SWIG. Essa aplicação permite a geração de ligações para diversas linguagens de programação, entre elas a linguagem Python.

- Usar um construtor de chamadas

Uma das táticas mais interessantes para acessar uma biblioteca escrita em outra linguagem é com o uso do módulo Ctypes. O Ctypes é uma ligação com uma biblioteca chamada *libffi,* que permite que chamadas à funções C/C++ sejam construídas em tempo de execução tornando desnecessário o uso de qualquer outra linguagem que não seja o Python.

3.3.2 Integração com a linguagem Java

Para a integração com a linguagem de programação Java pertencente à empresa Sun, temos o Jython [20]. Esta ferramenta é um interpretador Python escrito em Java que permite que aplicações Python sejam executadas de dentro de qualquer plataforma Java.

O Jython é que um arquivo com extensão *class* da linguagem Java. Essa é uma vantagem quando se pretende rodar aplicativos Python em lugares onde não existe o Python instalado, somente uma *Java Virtual Machine* (JVM).

Essa ferramenta também provê um ambiente interativo onde se podem instanciar objetos da linguagem Java no console de comandos da linguagem Python.

3.3.3 Integração com a plataforma .NET

Para integração com a plataforma de desenvolvimento .NET pertencente à empresa Microsoft, existe o IronPython. O IronPython é uma implementação da linguagem Python integrada com a plataforma .NET [21].

É implementada em C#, permitindo a execução de qualquer biblioteca .NET a partir do Python.

3.4 Aplicações que utilizam a Linguagem Python.

Existem varias instituições ao redor do mundo que utilizam aplicações desenvolvidas em Python:

- Google – Partes de seu *software* são desenvolvidas em Python
- Yahoo - usa Python para o *site* de grupos.
- Nasa – possui módulos gráficos usado em missões de planejamento do espaço.
- Industrial Light & Magic - Produz filmes da série *Star Wars* usando Python para computação gráfica

No Brasil algumas das empresas que trabalham com a Linguagem Python são: Embratel, Conectiva, Serpro, Haxent, Async, GPr, Hiperlógica.

Algumas instituições de ensino superior estão usando a Linguagem Python como ferramenta de ensino de programação devido à facilidade de aprendizagem. São elas: Universidade Federal de São Carlos (UFSC), Escola Politécnica da Universidade de São Paulo (POLI – USP), Pontifícia Universidade Católica de Campinas (PUC), etc.

Capítulo 4

Arquitetura do Sistema

Este capítulo apresenta uma descrição do funcionamento do sistema. Mostra particularidades das etapas de acesso aos dispositivos de captura de áudio e vídeo, de captura dos sinais, de transmissão e recepção, de troca de mensagens de texto, além de apresentar detalhes da interface gráfica do usuário. Por fim apresenta a modelagem do sistema.

4.1 Introdução

O projeto foi planejado com o objetivo de desenvolver um modelo de comunicação cliente-servidor baseado em *sockets* [22]. Um *socket* é uma interface de comunicação bidirecional entre processos. É representado como descritor de arquivos e permite a comunicação entre processos distintos na mesma máquina ou em máquinas distintas através de uma rede. Os *sockets* são a base da comunicação em redes TCP/IP e também são muito usados em comunicações entre processos no interior de um mesmo computador.

Ao criar um *socket*, deve ser escolhido o protocolo que irá efetuar a comunicação, ou seja, o mecanismo usado para a transferência dos dados. É importante que os *sockets* de ambos os lados usem o mesmo protocolo.

Como em nosso sistema, usamos datagramas, o protocolo de transporte correspondente é o UDP. O UDP [23] providencia a entrega de datagramas sem a necessidade de uma conexão estabelecida entre cliente e servidor, o que torna a comunicação mais ágil. Porém, não há garantia nem confirmação da entrega dos pacotes. O

UDP utiliza o conceito de melhor esforço (paradigma que é adotado em redes de pacotes, onde os dados são transportados pelo melhor caminho que se apresenta no momento da transmissão). É também usado em sistemas que transmitem pequenas quantidades de dados ao mesmo tempo ou possuem necessidades em tempo real.

Durante a transmissão, um fator a ser considerado é o tamanho dos registradores (*buffers*) a serem usados. Os *buffers* são áreas de memória criadas pelos programas para armazenar dados que estão sendo processados pelas aplicações.

O *buffer* também é importante para combater problemas de variação de tempo de chegada (*jitter*) na rede. Esta variação se dá em redes de pacotes pelos diversos caminhos trafegados pelos pacotes e pelas diferentes condições de tráfego ao longo da rede.

Para resolver os problemas de *jitter*, já se iniciaram testes utilizando o *Real Time Protocol* (RTP) juntamente com o UDP. O RTP permite gerenciar o tráfego de pacotes pela rede. No entanto, estes testes envolvem uma interação entre a linguagem Python e a linguagem C/C++, já que as funções do protocolo RTP serão implementadas utilizando a linguagem C/C++ e serão chamadas dentro do programa escrito em Python.

O sistema proposto trabalha da seguinte maneira. É iniciado o programa servidor de vídeo. Para executar a função de servidor devem ser estabelecidas duas portas de comunicação, uma para o áudio e outra para o vídeo. O cliente solicita o envio dos dados utilizando o endereço IP do servidor e as mesmas portas de comunicação previamente estabelecidas.

No servidor ocorre a captura dos sinais de áudio e de vídeo através do acesso aos dispositivos (microfone e *webcam*). Em seguida ocorre a codificação e compressão dos dados. A partir daí os dados são transmitidos diretamente para o(s) cliente(s) sem a necessidade de serem armazenados no computador servidor.

No cliente ocorre a reprodução do áudio e do vídeo. Durante a transmissão, as partes podem trocar mensagens de texto através de um bate papo. Deve-se apenas especificar outra porta para essa comunicação, que é bidirecional.

Como contribuição do trabalho, o servidor permite o armazenamento dos dados durante a transmissão.

4.2 Descrição do Sistema

4.2.1 Acesso aos dispositivos de áudio e vídeo

A etapa inicial foi baseada no acesso ao microfone e a *webcam* (previamente acoplada na entrada USB) do computador. Para acessar o dispositivo de vídeo utilizamos funções da biblioteca VideoCapture [24] da linguagem Python. Essa biblioteca permite, a partir da classe chamada *Device*, criar uma instância dentro do programa que representará o dispositivo de vídeo. A partir daí, podemos a cada momento capturar uma imagem da *webcam* e montar a seqüência de vídeo.

O acesso ao microfone é feito utilizando a biblioteca Pymedia da linguagem Python, assim como a codificação e compressão dos dados. Falaremos dela no próximo tópico.

4.2.2 Captura

Para a captura de áudio e vídeo, foram utilizadas funções da biblioteca Pymedia da linguagem Python. Essa biblioteca é própria para manipulação de arquivos de áudio e vídeo [25].

Como já foi citada, a biblioteca provê o acesso ao microfone do computador. Isso é possível através da classe *Input* da biblioteca. Deve-se passar como parâmetro para a classe a freqüência de amostragem e o número de canais desejados (normalmente 44100 Hz e 2 canais). A partir daí, permite captura o som do microfone.

O conjunto de funções desenvolvidas com a ajuda da biblioteca Pymedia efetua a codificação e a compressão dos dados de áudio e vídeo. Para isso, são criados dois codificadores, um que recebe os parâmetros de áudio e outro que recebe os parâmetros de vídeo, previamente definidos no programa. Esses parâmetros incluem os formatos de compressão.

Um recurso adicional da plataforma possibilita armazenar o áudio e o vídeo antes ou durante a transmissão. A partir de uma função Python chamada '*open'* pode-se manipular arquivos.

4.2.3 Transmissão – Recepção

A interação entre os processos usa o modelo de comunicação cliente-servidor. Algumas características importantes desse modelo são as seguintes.

- O cliente conhece o endereço e forma de acesso ao servidor e toma a iniciativa da comunicação.
- O servidor é uma entidade passiva, apenas recebendo pedidos dos clientes e respondendo aos mesmos.
- O servidor oferece um serviço específico a seus clientes.
- O cliente envia uma requisição de serviço e aguarda uma resposta do servidor.
- As implementações do cliente e do servidor são independentes e autônomas; apenas as seqüências de mensagens trocadas durante a comunicação, que caracterizam o serviço, devem ser respeitadas.

Como servidor e cliente têm comportamentos distintos em relação a comunicação, suas implementações seguem também padrões distintos. Ambos criam *sockets* para se comunicar, mas operam de forma diferente. O servidor cria *sockets* que o mantêm em estado de espera, aguardando por requisições. O cliente cria *sockets* que efetuam requisições. A partir do momento que os *sockets* do servidor recebem as requisições, é permitido o envio dos sinais. No cliente após fazer as requisições, os *sockets* ficam prontos para receber os sinais.

Desta forma, são implementadas duas interfaces gráficas distintas, uma para o servidor e outra para o cliente.

O computador acoplado com uma *webcam*, que faz a captura dos dados torna-se o servidor de áudio e vídeo. Conseqüentemente, o computador que faz a requisição dos dados torna-se o cliente de áudio e vídeo.

Na interface gráfica do servidor, para executar o *software* na função de servidor (alternativamente pode se apenas usar o programa para gravação) é necessário especificar duas portas de comunicação, uma para o áudio e outra para o vídeo.

Portas de comunicação são conexões do computador para uma rede externa e vice-versa. Existem portas reservadas para aplicações específicas e outras para uso de aplicações

em geral, como é o caso da nossa aplicação. As portas específicas [26] são de baixa numeração, como por exemplo, a porta 80, usada pelo protocolo de Internet HTTP. Portanto, em nosso sistema, usaremos tipicamente portas com numeração alta.

A partir da especificação das portas de comunicação, o servidor é inicializado, ficando sempre ativo, em sincronia, esperando possíveis requisições de clientes.

Na interface gráfica do cliente, para acessar o servidor, o cliente deve especificar o IP da máquina onde o servidor está sendo executado e as duas portas de comunicação previamente escolhidas no servidor. Estas informações são usadas para enviar a requisição e iniciar a comunicação. Durante o tempo em que recebe os sinais de áudio e vídeo o cliente faz a reprodução dos dados em tempo real.

O sistema possui um comportamento assíncrono, ou seja, o servidor pode prover os serviços a mais de um cliente ao mesmo tempo.

4.2.4 Mensagens de Texto

Além da transmissão de áudio e vídeo o servidor permite o estabelecimento de um bate papo com troca mensagens de texto com o cliente. Para isso é necessário que seja especificada uma nova porta de comunicação no servidor. O cliente envia a requisição de texto para essa porta e são abertas duas janelas de texto independentes das interfaces gráficas, uma para o servidor e outra para o cliente. Com isso inicia-se uma comunicação com mensagens de texto bidirecional entre servidor e cliente.

4.2.5 Interface Gráfica do Usuário

As interfaces gráficas do usuário foram desenvolvidas baseadas no módulo de desenvolvimento de interfaces gráficas Tkinter. Tkinter é umas das GUI's (*Graphical User Interface - Interface Gráfica do Usuário)* disponíveis para Python. É a biblioteca padrão da linguagem para desenvolvimento de interfaces e acompanha a distribuição oficial do interpretador Python. Sua grande vantagem é que é portátil em qualquer ambiente em que funcione o Python.

As interfaces gráficas gerenciam e controlam as funções de acesso aos dispositivos de áudio e vídeo, captura dos sinais, armazenamento, transmissão e recepção e por fim a reprodução dos sinais.

A interface servidora permite que o áudio e o vídeo capturados para transmissão sejam armazenados, antes ou durante a transmissão, permite tirar fotos e também iniciar uma comunicação de texto com o servidor via janela de bate papo, onde mensagens de texto são trocadas pelas partes em uma comunicação bidirecional.

A interface gráfica do cliente permite efetuar requisições para o servidor, reproduzir em tempo real os dados recebidos, além de trocar mensagens de texto com o servidor.

Para inicializar as interfaces gráficas utilizamos o *prompt* de comando do Windows como mostrado na Fig. 3.2. Há também a possibilidade de inicializar as interfaces pelo interpretador interativo IDLE, porém nesse caso podem ocorrer conflitos com a interface do IDLE.

Como existe a opção de evolução do programa, ele ainda é inicializado por linhas de comando. Posteriormente será disponibilizado como um pacote de instalação com todos os arquivos e bibliotecas necessárias para seu funcionamento, podendo ser inicializado através do menu Iniciar da barra de ferramentas do Windows.

4.3. Modelagem do Sistema

Nesta seção são apresentados os diagramas UML (*Unified Modeling Language*) do sistema desenvolvido. O sistema (parte servidor e parte cliente) é apresentado através de um conjunto de diagramas, onde cada diagrama se refere a uma visão parcial do mesmo. Todo o sistema foi implementado seguindo as bases da Programação Orientada a Objetos (POO), que traz benefícios quanto a modularidade, encapsulamento, reuso, simplificação e redução do custo de manutenção.

4.3.1 Diagramas de Casos de Uso

Para representar os requisitos do sistema foram utilizados diagramas de casos de uso. O diagrama de casos de uso descreve a interação do cliente com as ações que o servidor realiza. O servidor p casos de uso que possui alguns casos de uso que o cliente não tem.

Os requisitos gerais identificados para o sistema proposto são apresentados na Fig. 4.1.

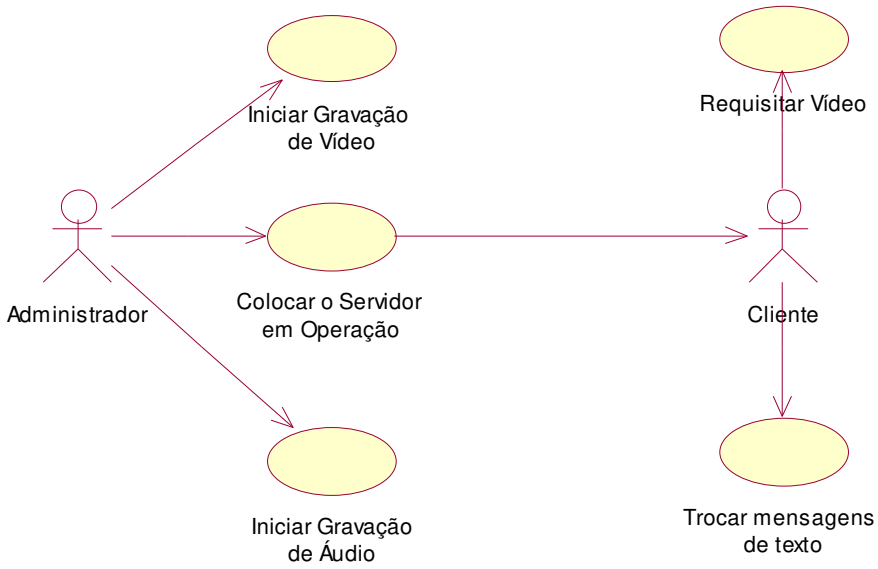

Fig. 4.1 – Diagrama de Casos de Uso

O detalhamento dos requisitos gerais do diagrama de casos de uso da Fig. 4.1 é mostrado abaixo:

Iniciar Gravação de Vídeo

Este caso de uso permite que o administrador grave vídeo para posterior utilização. Pode-se gravar antes ou durante a transmissão do vídeo. É necessário especificar o nome do arquivo de saída juntamente com a extensão do vídeo (mpeg, avi ou wmv) e o tempo de gravação.

Iniciar Gravação de Áudio

Este caso de uso permite que o administrador grave áudio para posterior utilização. Pode-se gravar antes ou durante a transmissão do vídeo. É necessário especificar o nome do arquivo de saída juntamente com a extensão do áudio (mp3) e o tempo de gravação.

Colocar o Servidor em Operação

Este caso de uso permite que o administrador coloque o servidor em estado de espera por requisições dos clientes. É necessário especificar duas portas de comunicação, uma para o áudio e outra para o vídeo. Quando receber requisições, o servidor captura as seqüências de áudio e vídeo, codifica as seqüências e as transmite, sem a necessidade de gravação. Ao longo que recebe uma requisição, o servidor já fica preparado para receber a próxima requisição. É possível ainda estabelecer uma porta de comunicação trocar mensagens de texto com os clientes.

Requisitar Vídeo

Este caso de uso permite que o cliente faça a requisição para transmissão do vídeo. Ao longo em que o cliente vai recebendo as seqüências de áudio e vídeo ocorre a reprodução quase e tempo real.

Trocar Mensagens de Texto

Este caso de uso permite que o cliente faça a requisição para troca de mensagens de texto.

4.3.2 Diagramas de Classes

Os diagramas de classes são usados para descrever a estrutura do sistema e suas relações. É nele que encontramos informações sobre métodos, atributos, nomes de funções

e como serão integradas. Uma classe geralmente é descrita como o modelo ou a forma a partir da qual um objeto é criado e cada objeto deve ser responsável pela realização de um conjunto específico de tarefas relacionadas. Um objeto nunca deveria manipular diretamente os dados internos de outro objeto e toda comunicação entre eles deve ser através de mensagens (chamadas a métodos).

A Fig. 4.2 apresenta o diagrama de classes do servidor. A classe *MyApp_Server* gerencia as classes: *Chat_TxThread* (estabelece o bate-papo), *Picture_Thread* (tira fotos), *Audio_Save* (armazena áudio), *Video_Save* (armazena vídeo), *Video_TxThread* (transmite vídeo) e *Audio_TxThread* (transmite áudio).

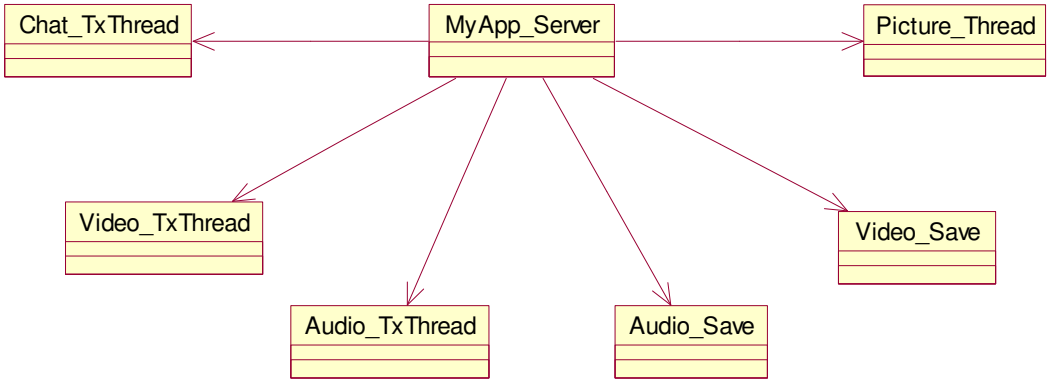

Fig. 4.2 – Diagrama de Classes do Servidor

A Fig. 4.3 apresenta o diagrama de classes do cliente. A classe *MyApp_Client* gerencia as classes: *Chat_RxThread* (estabelece o bate-papo), *Video_RxThread* (faz a requisição e recebe sinal de vídeo) e *Audio_RxThread* (faz a requisição e recebe sinal de áudio).

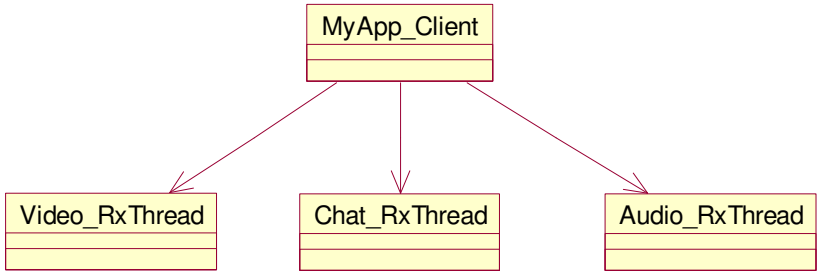

Fig. 4.3 – Diagrama de Classes do Cliente

4.3.3 Diagramas de Seqüência

Os diagramas de seqüência mostram a interação entre os objetos ao longo do tempo e apresenta os objetos que participam da interação e a seqüência de mensagens trocadas. Para que o comportamento do sistema seja entendido, os diagramas de seqüência apresentados a seguir contêm algumas chamadas de métodos de classes.

O diagrama de seqüência da Fig. 4.4 define passo-a-passo as etapas necessárias para que servidor e cliente estabeleçam comunicação.

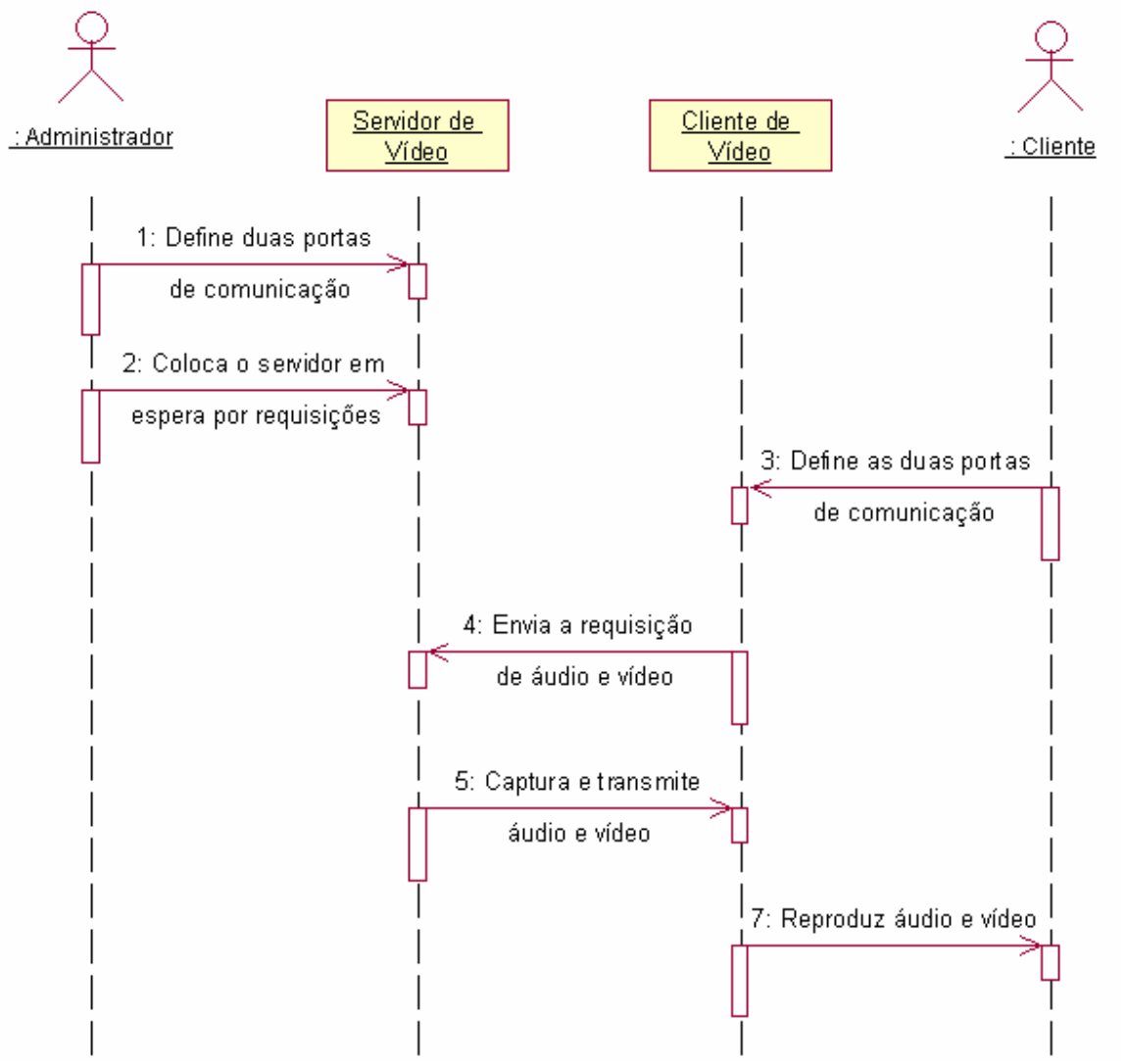

Fig. 4.4 – Diagrama de Seqüência da comunicação entre servidor e cliente

No diagrama da Fig. 4.4 vemos que inicialmente o administrador do servidor coloca o mesmo em operação. A partir daí o cliente faz requisição para o envio dos sinais de áudio e vídeo. O servidor captura e transmite os sinais. O cliente recebe e reproduz os sinais ao longo em que chegam.

O diagrama de seqüência específico da transmissão é mostrado na Fig. 4.5.

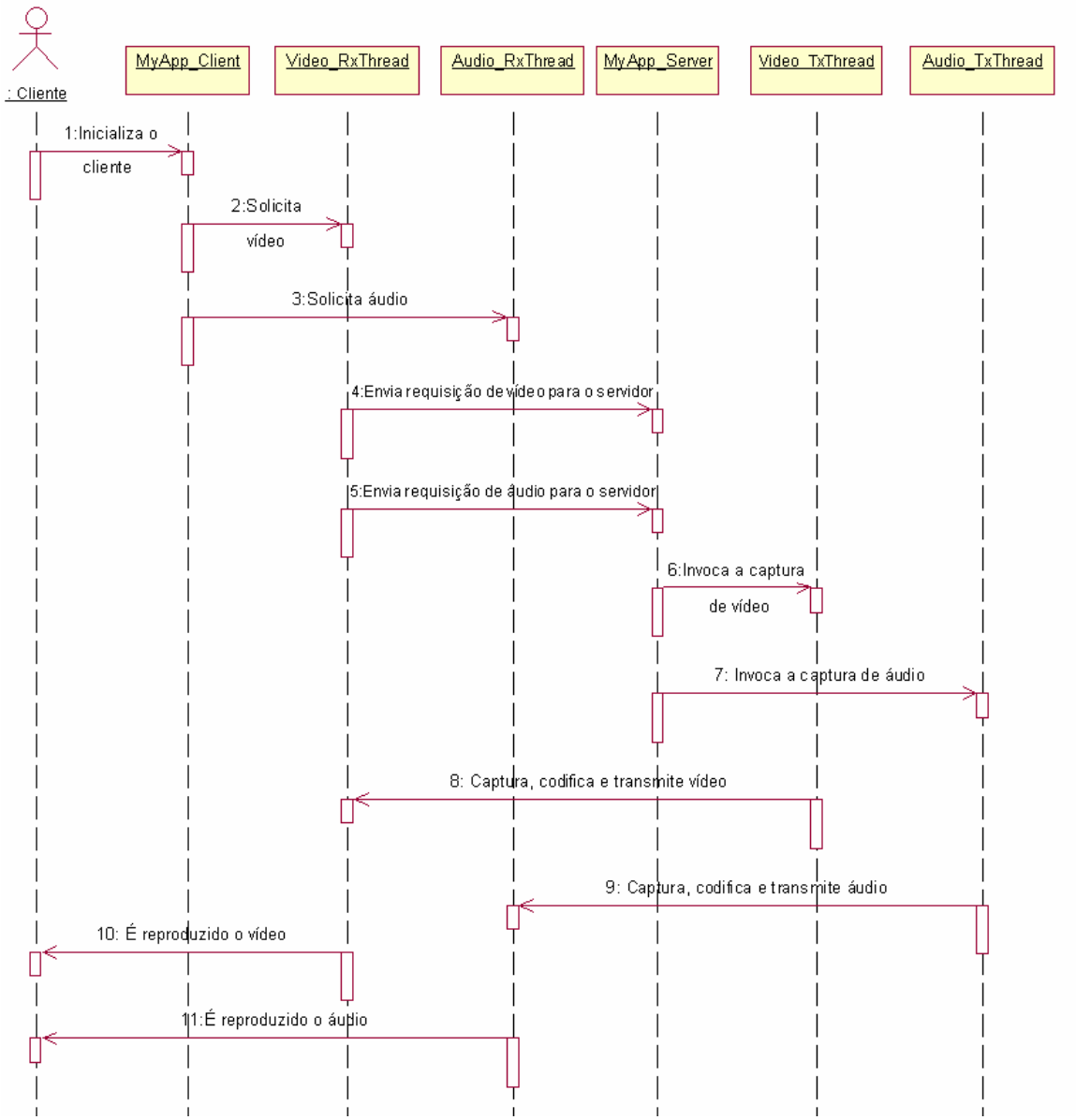

Fig. 4.5 – Diagrama de Seqüência da transmissão de áudio e vídeo

As duas principais classes mostradas no diagrama da Fig. 4.5 são *MyApp_Server* que cria interface gráfica do servidor e *My_App_Client* que cria a interface gráfica do cliente. Quando há uma requisição do cliente, *My_App_Client* chama *Video_RxThread* e A*udio _RxThread* que se comunicam com *MyApp_Server. MyApp_Server* invoca *Video_TxThread* e A*udio_RxThread* que captura e transmite os sinais para *Video_RxThread* e A*udio _RxThread.* A partir daí os dados são reproduzidos.

Capítulo 5

Resultados Experimentais

Neste capítulo descrevemos os resultados experimentais obtidos a partir da implementação do sistema descrito no capítulo anterior.

5.1 Metodologia

A primeira parte do projeto se refere à etapa de captura dos sinais de áudio e vídeo. Nesta parte, acessamos os dispositivos e capturamos as seqüências (*streams*) de áudio e vídeo para serem transmitidos na rede em tempo real ou para serem salvos no computador.

Durante esta etapa iniciamos o desenvolvimento de uma das interfaces gráficas de usuário, baseada na biblioteca de desenvolvimento Tkinter, que no decorrer do trabalho foi aprimorada e se tornou a interface final do servidor.

Para sincronização das seqüências de áudio e vídeo, pretendíamos utilizar a função "*muxer*" [25] da biblioteca Pymedia do Python. Porém, obtivemos problemas com a sincronização dessas seqüências no momento da captura. Em contato com um dos mantenedores da linguagem Python, *Dmitry Borisov*, soubemos que essa função possui problemas, e que não desempenha um bom funcionamento quando trabalhamos com áudio e vídeo juntos.

Para contornar esses problemas enviamos as seqüências em separado e com a reprodução feita em paralelo no cliente. O áudio e o vídeo são transmitidos por *sockets* distintos, cada um utilizando uma porta de comunicação deferente.

Para tornar o programa totalmente *multi-threading*, utilizamos a biblioteca *Threading* do Python. Essa biblioteca provê as funções de *threads* para a linguagem. *Threads* são porções de código que são executados em "paralelo" dentro de um processo

principal [11]. Este tipo de programação conhecida como programação paralela, permite que um programa possa executar várias tarefas diferentes ao mesmo tempo, independentemente umas das outras. Isto é o que possibilita vários clientes se conectem ao mesmo tempo com o servidor.

A segunda etapa envolve a transmissão e a recepção de sinal propriamente ditas.

Na recepção, a reprodução do áudio é feita pelo cliente utilizando apenas funções da linguagem Python. Para a reprodução do vídeo, além das funções Python, utilizamos o *VLC media player* [27] como parte cliente do sistema. Esse *software* é um tocador (*player*) multimídia de código aberto, altamente portátil para vários formatos de áudio e vídeo.

5.2 O Servidor – Interface Gráfica

Inicializando o programa servidor de vídeo, será carregada a interface gráfica do servidor conforme mostrado na Fig. 5.1. Na interface gráfica do servidor podem ser vistos a barra de menus e os botões disponíveis na janela principal do programa.

A barra de menu é formada por cinco menus: Arquivo, Áudio, Vídeo, Foto e Ajuda.

Através dos itens do menu principal podem ser acessadas as seguintes funcionalidades:

- Arquivo – Sair: ao se pressionar este item de menu, o usuário sai da janela principal, fechando o programa.
- Áudio – Parâmetros: ao se pressionar este item de menu, será aberta uma janela como mostrado na Fig. 5.2.

Na janela da Fig. 5.2 o usuário pode definir alguns parâmetros do áudio como *Bitrate, Sample Rate* e *Channels.* Através da opção de *Bitrate* pode-se escolher dentre alguns valores pré-definidos de processamento de *bits* por segundo. A opção *Sample Rate* permite escolher a freqüência de amostragem do sinal de áudio. E por fim com a opção *Channels* pode-se definir quantos canais de áudio serão usados: 1 canal – Mono ou 2 canais – Estéreo.

Fig. 5.1: Interface do programa Servidor de Vídeo.

Fig. 5.2: Opções de Áudio.

- Vídeo - Parâmetros: ao se pressionar este item de menu, será aberta uma janela como mostra a Fig. 5.3.

Fig. 5.3: Opções de Vídeo.

Na janela da Fig 5.3 o usuário pode definir alguns parâmetros do vídeo como Codificação e *Frame Rate*. Através da opção de Codificação, o usuário poderá escolher entre relativos os padrões de compressão MPEG 1 e MPEG2 (valores 2,7 Mbps e 9,8 Mbps respectivamente). A opção *Frame Rate* dá a oportunidade ao usuário de escolher quantos quadros por segundo devem ser processados.

- Ajuda – Ajuda: ao se pressionar este item de menu, abre-se uma janela contendo alguns detalhes a respeito do funcionamento do programa.
- Ajuda – Sobre: este item de menu apresenta uma janela contendo informações sobre o programa servidor de vídeo, tais como autor, contato e o nome da universidade como podemos ver na Fig. 5.4.

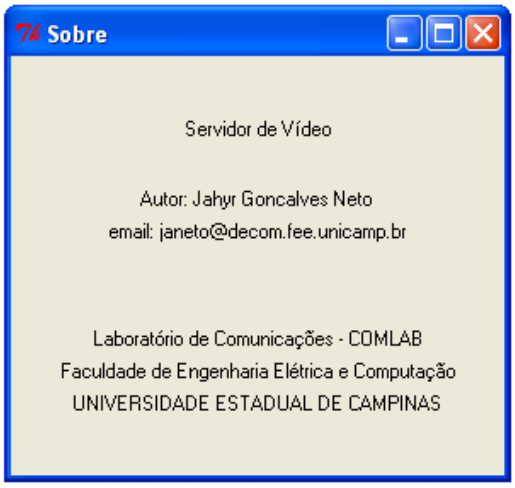

Fig. 5.4: Sobre.

A janela principal da interface é formada por seis botões. São eles: Gravar Vídeo, Gravar Áudio, Servidor, Tirar Foto, *Chat* e Sair.

A principal função do programa Servidor de Vídeo é fazer transmissões de áudio e vídeo. Como contribuição, além da transmissão, a interface permite fazer capturas de áudio e vídeo para armazenar no computador, tirar fotos e ainda iniciar um bate-papo de mensagens de texto.

Através dos botões da janela principal podem ser acessadas as seguintes funcionalidades:

- Servidor:

Ao pressionar o botão Servidor, será aberta a janela de endereço do servidor. Essa janela é mostrada na Fig. 5.5. Na janela de endereço do servidor deve-se especificar duas portas de comunicação, uma para o áudio e a outra para o vídeo.

Fig. 5.5: Janela de Endereço do Servidor.

Pressionando-se o botão OK da janela da Fig. 5.5, o servidor entra em estado de espera, podendo receber requisição de algum cliente. Caso o usuário deseje retornar a janela principal desistindo da transmissão, deve pressionar o botão Cancelar.

Quando o cliente faz uma requisição, o servidor inicia o processo de captura das seqüências de áudio e vídeo e as envia em tempo real para o endereço do cliente que efetuou a requisição. Quando o cliente é encerrado pelo usuário, o servidor volta para o estado de espera por novas requisições.

- Gravar Vídeo:

Pressionado o botão Gravar Vídeo na interface principal, será aberta uma janela de opções de vídeo como vemos na Fig. 5.6.

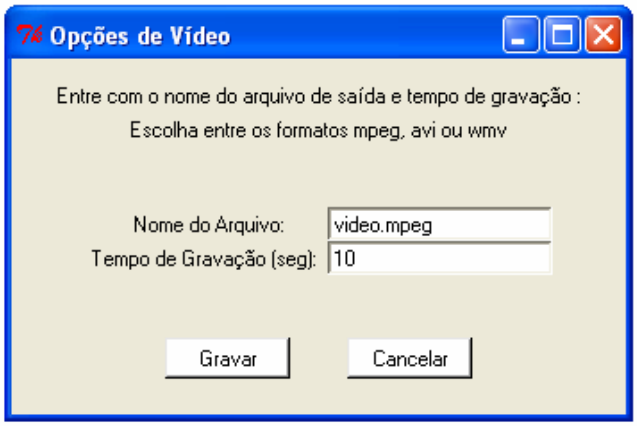

Fig. 5.6: Opções de Vídeo 2.

Na janela da Fig. 5.6 o usuário deve informar o nome do arquivo de saída do vídeo e o tempo que deseja fazer a gravação. É possível escolher o formato de saída do arquivo de vídeo: mpeg, avi ou wmv. O arquivo será armazenado na pasta raiz do Python. Caso o usuário deseje retornar a janela principal desistindo da gravação do vídeo, deve pressionar o botão Cancelar.

- Gravar Áudio:

A gravação do áudio funciona da mesma forma. Ao pressionar Gravar Áudio, será aberta a janela Opções de Áudio (Fig. 5.7), na qual o usuário informa o nome do arquivo de saída e o tempo de gravação.

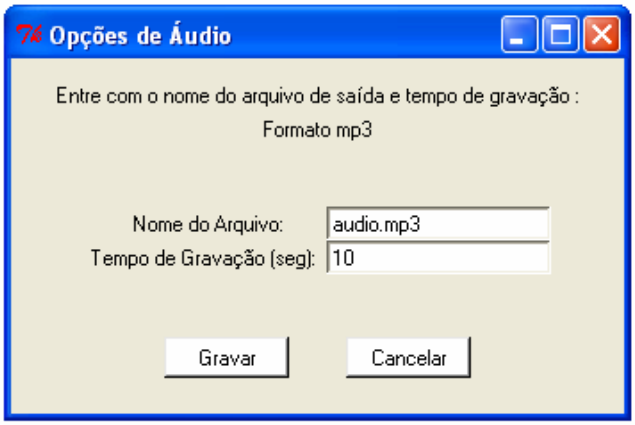

Fig. 5.7: Opções de Áudio 2.

Caso o usuário deseje retornar a janela principal desistindo da gravação do áudio, deve pressionar o botão Cancelar.

- Tirar Foto:

  O programa permite tirar fotos. Os arquivos serão salvos na pasta raiz do Python.

- *Chat*:

O programa permite que seja estabelecida uma comunicação através de mensagens de texto entre o servidor e o cliente. O servidor e o cliente podem iniciar uma conversa de texto antes ou durante transmissão. Para isso ao se pressionar o botão Chat na interface do servidor será aberta uma janela como mostra a Fig. 5.8.

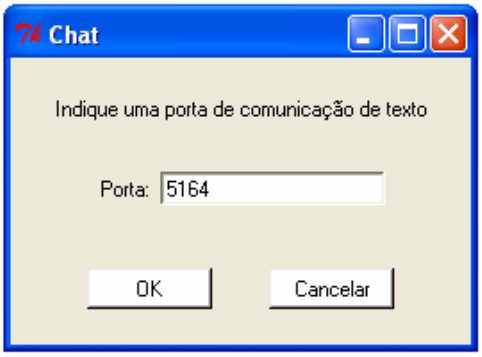

Fig. 5.8: Janela de Endereço do Chat.

Definindo se a porta de comunicação de texto e pressionando o botão OK da janela da Fig. 5.8, ficará disponível uma janela independente, o *prompt* de comando, para a troca de mensagens como mostra a Fig. 5.9:

Fig. 5.9: Janela para troca de mensagens

- Sair:

  Finalizam-se os processos e encerra-se o programa

5.3 O Cliente – Interface Gráfica

A interface gráfica do cliente é mostrada na Fig. 5.10.

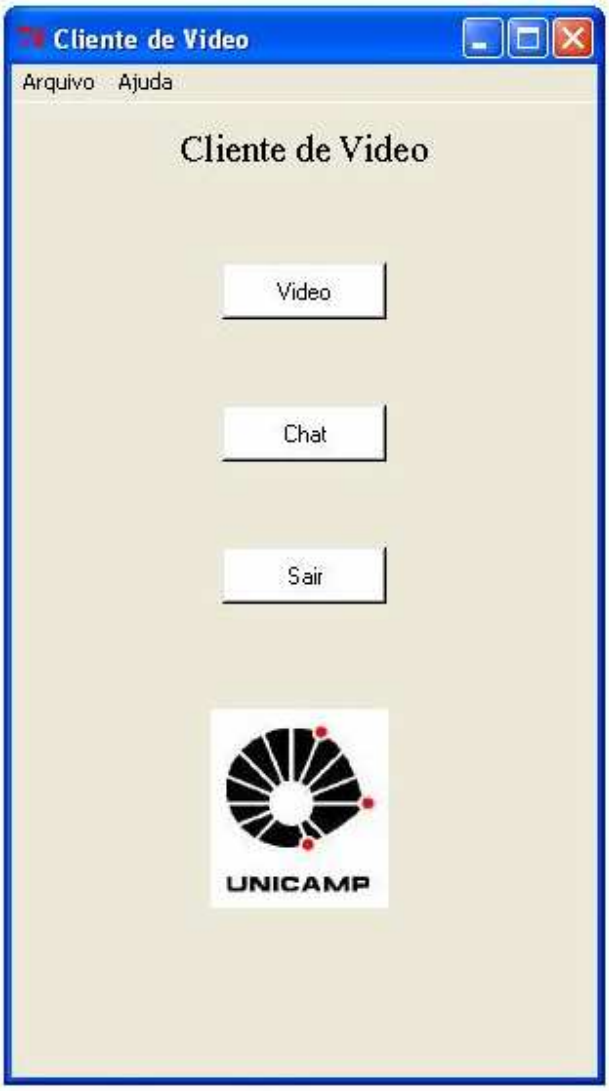

Fig. 5.10: Interface do Cliente.

Para efetuar uma requisição, o usuário deve pressionar o botão Vídeo na interface principal (Fig. 5.10). Será aberta uma janela de endereços como mostra a Fig. 5.11. Nesta

janela da Fig. 5.11, o usuário deve informar o endereço IP da máquina que funciona como servidor e as duas portas de comunicação para que ele possa estabelecer uma comunicação e começar a receber os dados.

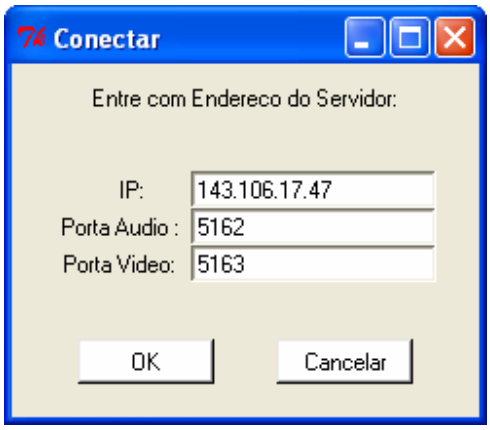

Fig. 5.11: Janela de Endereços.

Pressionando-se o botão OK na janela da Fig 5.11, o cliente entra em sincronia com o servidor, recebendo as seqüências de áudio vídeo. A seqüência de áudio é reproduzida automaticamente e a seqüência de vídeo utiliza o tocador VLC para a reprodução como vemos na Fig. 5.12.

Fig. 5.12: Tocador VLC como parte do Cliente.

Uma das principais características do tocador VLCé a exibição de vídeo recebido através da rede.

O botão Sair da Fig 5.10 finaliza o programa.

Para iniciar a comunicação de texto, o cliente pressiona o botão *Chat*. Será aberta a janela mostrada na Fig. 5.13.

Fig. 5.13: Janela de Endereço do *Chat* no cliente.

Na janela da Fig. 5.13, o cliente especifica o IP e a porta de comunicação previamente escolhida no servidor. A partir daí será aberta uma janela para a troca de mensagens de texto, como podemos ver na Fig. 5.14.

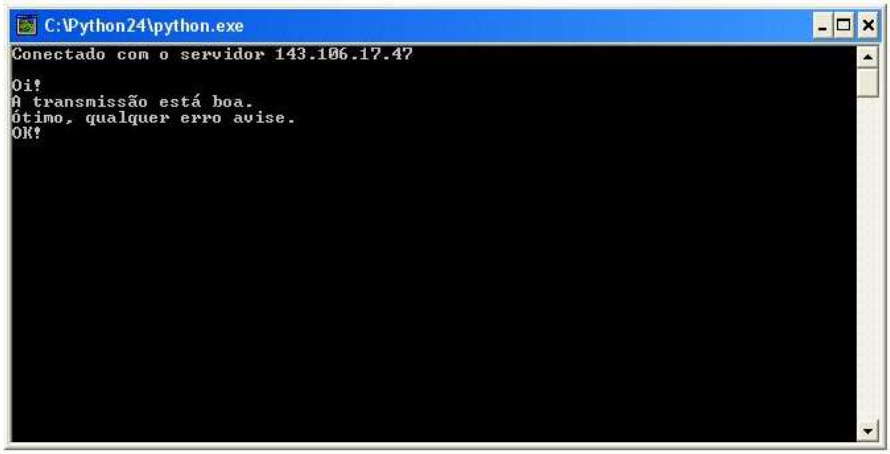

Fig. 5.14: Janela do Cliente para troca de mensagens de texto.

5.4 Formato dos Pacotes

As informações trocadas entre servidor e cliente envolvem a transmissão de pacotes de dados na rede. Os pacotes possuem informações que desejamos transmitir e cabeçalho, que possui dados fundamentais para a transmissão.

Para a verificação dos protocolos utilizados na formação do pacote, além da possibilidade de visualização de envio e chegada dos pacotes de áudio e vídeo, utilizamos o *software* Ethereal. Esse *software* é denominado um analisador de protocolos de redes [28]. Com ele capturamos os pacotes no momento da transmissão e podemos assim ver o formato exato dos pacotes, ou seja, os dados propriamente ditos e os protocolos que os encapsulam.

5.4.1 Pacotes de Áudio

Os pacotes de áudio possuem em torno de 400 a 500 Bytes de tamanho. Escolhemos um deles, dentre os vários capturados para uma breve análise dos protocolos.

```
⊟ Frame 34 (472 bytes on wire, 472 bytes captured)
    Arrival Time: Jun  6, 2007 15:15:35.774923000
    [Time delta from previous packet: 0.001205000 seconds]
    [Time since reference or first frame: 0.285114000 seconds]
    Frame Number: 34
    Packet Length: 472 bytes
    Capture Length: 472 bytes
    [Protocols in frame: eth:ip:udp:data]
⊟ Ethernet II, Src: Intel_57:87:c9 (00:11:11:57:87:c9), Dst: AsustekC_0d:0c:42 (00:13:d4:0d:0c:42)
  ⊟ Destination: AsustekC_0d:0c:42 (00:13:d4:0d:0c:42)
      Address: AsustekC_0d:0c:42 (00:13:d4:0d:0c:42)
      .... ...0 .... .... .... .... = Multicast: This is a UNICAST frame
      .... ..0. .... .... .... .... = Locally Administrated Address: This is a FACTORY DEFAULT address
  ⊟ Source: Intel_57:87:c9 (00:11:11:57:87:c9)
      Address: Intel_57:87:c9 (00:11:11:57:87:c9)
      .... ...0 .... .... .... .... = Multicast: This is a UNICAST frame
      .... ..0. .... .... .... .... = Locally Administrated Address: This is a FACTORY DEFAULT address
    Type: IP (0x0800)
⊟ Internet Protocol, Src: 143.106.17.47 (143.106.17.47), Dst: 143.106.17.48 (143.106.17.48)
    Version: 4
    Header length: 20 bytes
  ⊟ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
      0000 00.. = Differentiated Services Codepoint: Default (0x00)
      .... ..0. = ECN-Capable Transport (ECT): 0
      .... ...0 = ECN-CE: 0
    Total Length: 458
    Identification: 0xbf85 (49029)
  ⊟ Flags: 0x00
      0... = Reserved bit: Not set
      .0.. = Don't fragment: Not set
      ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 128
    Protocol: UDP (0x11)
  ⊟ Header checksum: 0x386a [correct]
      [Good: True]
      [Bad : False]
    Source: 143.106.17.47 (143.106.17.47)
    Destination: 143.106.17.48 (143.106.17.48)
⊟ User Datagram Protocol, Src Port: 6160 (6160), Dst Port: 4222 (4222)
    Source port: 6160 (6160)
    Destination port: 4222 (4222)
    Length: 438
    Checksum: 0x0480 [correct]
  Data (430 bytes)
```

Fig. 5.15: Formato de um pacote de áudio.

A Fig. 5.15 mostra a interface do Ethereal, que permite a visualização dos detalhes do pacote de áudio escolhido para análise. Podemos ver informações de tempo, de tamanho do pacote, de endereços, além de uma série de outras informações particulares de cada etapa. Podemos ver a pilha de protocolos utilizados para o envio do pacote de áudio. As linhas sombreadas mostram os protocolos. Este pacote em particular possui 430 Bytes de informação propriamente dita, passa por encapsulamento do protocolo de transporte UDP, do protocolo de rede IP e do protocolo de enlace Ethernet. Em cada ponto desse processo, são adicionadas informações ao cabeçalho do protocolo, até que ele chega ao final do processo com 472 Bytes.

O *software* permite também a visualização completa dos dados propriamente ditos dentro do pacote (neste caso 430 Bytes). Eles estão no formato hexadecimal e são apresentados na Fig. 5.16.

```
0000  00 13 d4 0d 0c 42 00 11  11 57 87 c9 08 00 45 00   .....B.. .W....E.
0010  01 ca bf 85 00 00 80 11  38 6a 8f 6a 11 2f 8f 6a   ........ 8j.j./.j
0020  11 30 18 10 10 7e 01 b6  04 80 00 40 9a 00 24 04   .0...~.. ...@..$.
0030  6e f6 bb 4b 0c 44 25 2e  94 e5 b9 33 69 2f 72 5b   n..K.D%. ...3i/r[
0040  1b b2 61 27 d3 4e 22 26  c8 81 2b 0a b1 f4 9c c0   ..a'.N"& ..+.....
0050  b0 e4 53 ff fb 92 64 14  80 04 2a 6a 4f db 86 1b   ..S...d. ..*jO...
0060  72 40 41 a9 fb 6d e6 26  10 a1 89 41 6e 18 6d c8   r@A..m.& ...An.m.
0070  fb 94 68 29 b4 09 28 26  4d 9f 82 4a 7a 67 40 e3   ..h)..(& M..Jzg@.
0080  6f e1 50 7b 95 29 18 ed  01 0d 04 98 71 35 83 92   o.P{.).. ....q5..
0090  a1 93 40 8a a2 24 d8 53  ac 46 42 20 f5 ff 50 ca   ..@..$.S .FB ..P.
00a0  69 fb 1b a9 0b f8 48 68  20 5a d7 bf 4c e5 23 98   i.....Hh  Z..L.#.
00b0  91 ca 39 90 ed 65 5f 73  26 0a 7b a1 d2 73 b4 81   ..9..e_s &.{..s..
00c0  e8 8f d9 1f 8c d0 11 d6  80 ea 39 f8 db 34 41 b9   ........ ..9..4A.
00d0  93 c8 f6 c1 89 e8 1e 08  09 4a 03 a3 7e 92 1c 81   ........ .J..~...
00e0  50 4a 11 9d 22 e1 e1 c4  64 ca 81 da 60 b2 94 14   PJ.."... d...`...
00f0  48 e3 c2 09 b2 a0 54 c9  97 a1 d4 a5 7b 7d db fd   H.....T. ....{}..
0100  4e 76 95 69 fb ff ff f9  64 7d 62 91 5a 96 64 fb   Nv.i.... d}b.Z.d.
0110  54 9c 86 a0 01 81 80 02  d8 3e 83 94 3e 90 0c 2a   T....... .>..>..*
0120  02 75 e1 f8 8b 76 95 40  d2 b6 9b 42 1c 29 ce 7a   .u...v.@ ...B.).z
0130  41 0c 3d 2f 96 a8 3a 4f  2c d9 c4 07 36 90 a5 e4   A.=/..:O ,...6...
0140  3d ed 4c 10 49 32 17 30  da 82 cb 29 20 82 41 86   =.L.I2.0 ...) .A.
0150  14 2b 1f 05 08 50 ee cb  99 91 63 e2 d1 cc 14 12   .+...P.. ..c.....
0160  e4 86 19 14 91 45 9b 8b  cc 1d 62 52 c9 08 cd d6   .....E.. ..bR....
0170  22 30 d0 4b 90 64 90 56  21 e9 11 bc d6 45 15 9d   "0.K.d.V !....E..
0180  3d 5e 94 fd a5 f5 3b 6d  74 15 b6 d6 35 26 11 a7   =^....;m t...5&..
0190  cf 95 47 ed 30 74 af 36  90 de 80 00 02 00 00 13   ..G.0t.6 ........
01a0  15 ad 5a 01 75 9f a8 7e  57 62 24 3c 22 30 69 d9   ..Z.u..~ Wb$<"0i.
01b0  f7 eb c6 f8 c7 bf 7e 8b  cf fc f6 6f 7e fa a6 c2   ......~. ...o~...
01c0  b1 80 4e b6 57 08 df f5  7e f4 f9 4a 6a 7f fd 1d   ..N.W... ~..Jj...
01d0  fc 9c bf af a4 6f fd 75                           .....o.u
```

Fig. 5.16: Dados do pacote de áudio.

5.4.2 Pacotes de Vídeo

Os pacotes de vídeo são maiores que os pacotes de áudio. Eles possuem em torno de 1200 a 1500 Bytes.

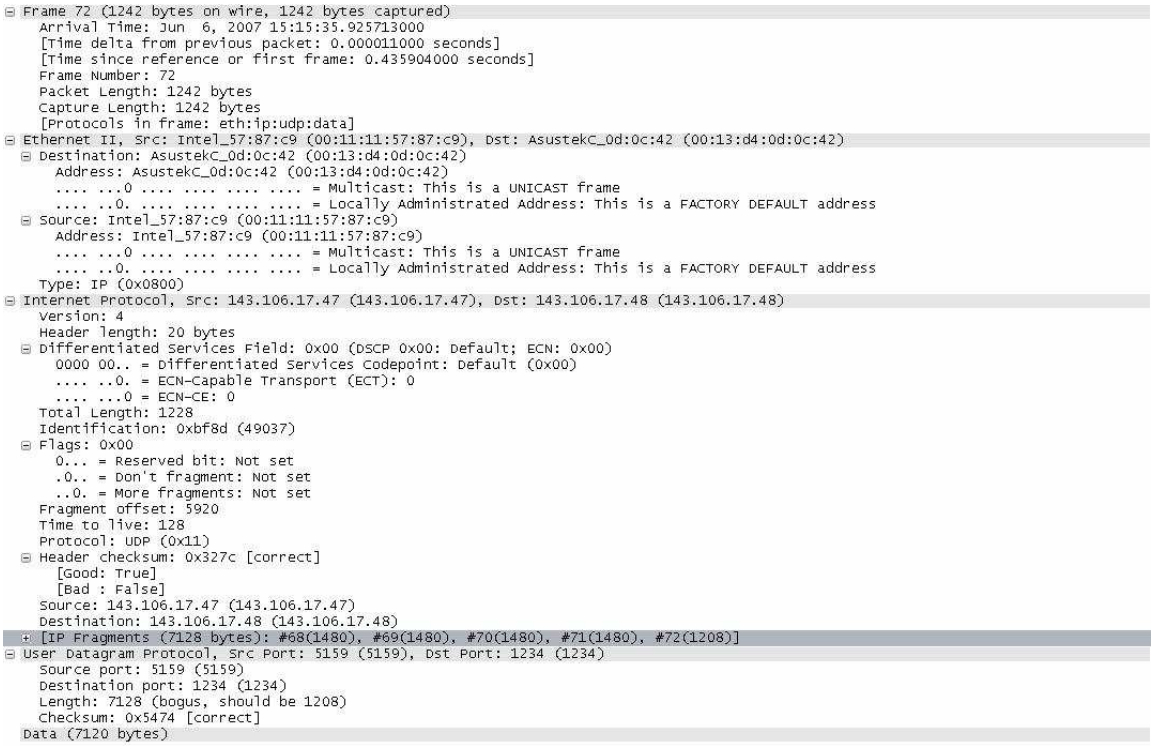

Fig. 5.17: Formato de um pacote de vídeo.

O formato de um pacote de vídeo em particular é mostrado na Fig. 5.17. Da mesma forma que acontece no pacote de áudio, a figura exibe uma série de informações de tempo, de tamanho, de endereços, além de uma série de informações particulares de cada protocolo. Podemos ver as etapas de encapsulamento de cada protocolo (linhas sombreadas), que são similares as etapas percorridas pelos pacotes de áudio. Este pacote de vídeo em destaque possui 1242 bytes de comprimento.

5.5 Consumo de Banda

Foram desenvolvidos testes para análise do consumo de banda no servidor em um dado intervalo de tempo durante a transmissão. Para estes testes foi usado o software NetLimiter, que permite monitorar o consumo de banda dos processos ativos.

Os gráficos a seguir ilustram o consumo de banda com um, dois e três clientes conectados ao servidor durante um período de vinte segundos. O eixo y mostra valores em Kbps (kilo *bits* por segundo) e o eixo x mostra valores em segundos.

O primeiro teste apresenta o consumo de banda do servidor com um cliente conectado (Fig. 5.18).

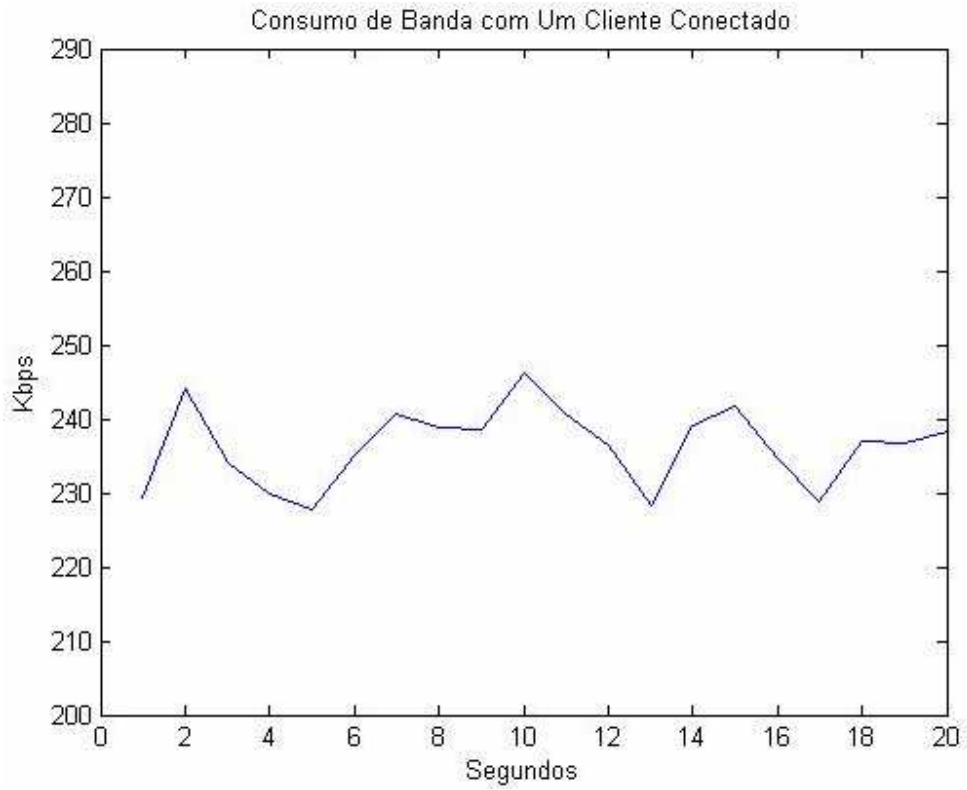

Fig. 5.18: Gráfico do consumo de banda com um cliente conectado.

O gráfico da Fig. 5.18 mostra que o consumo de banda do servidor quando está enviando dados para um cliente varia em torno de 225 Kbps a 245 Kbps.

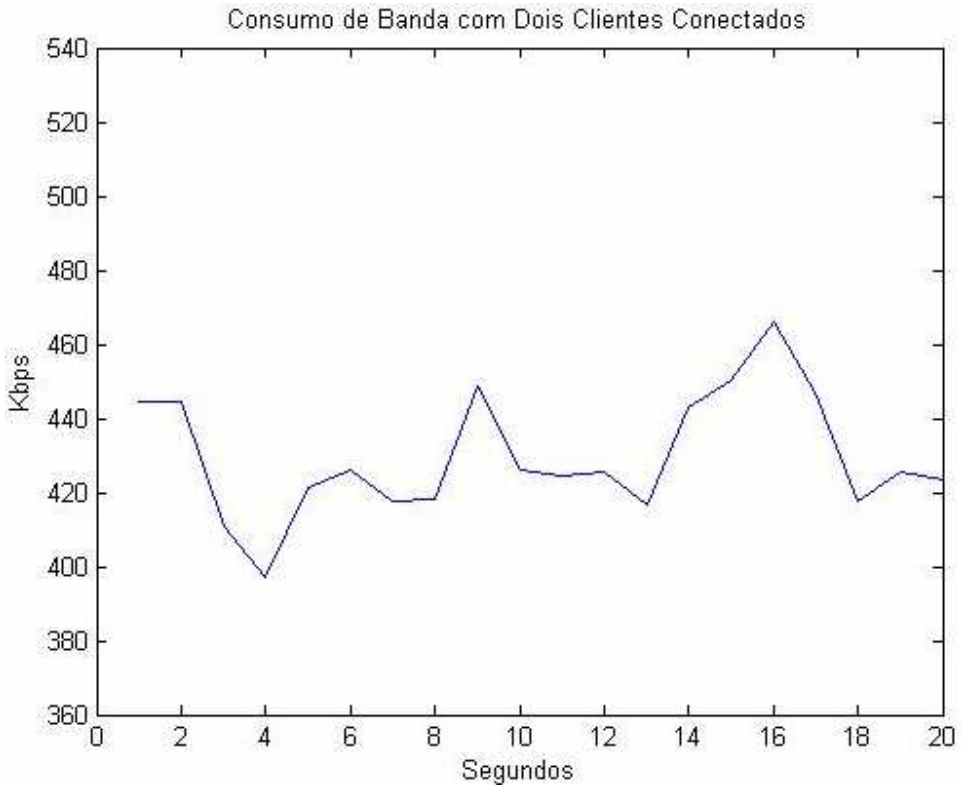

Fig. 5.19: Gráfico do consumo de banda com dois clientes conectados.

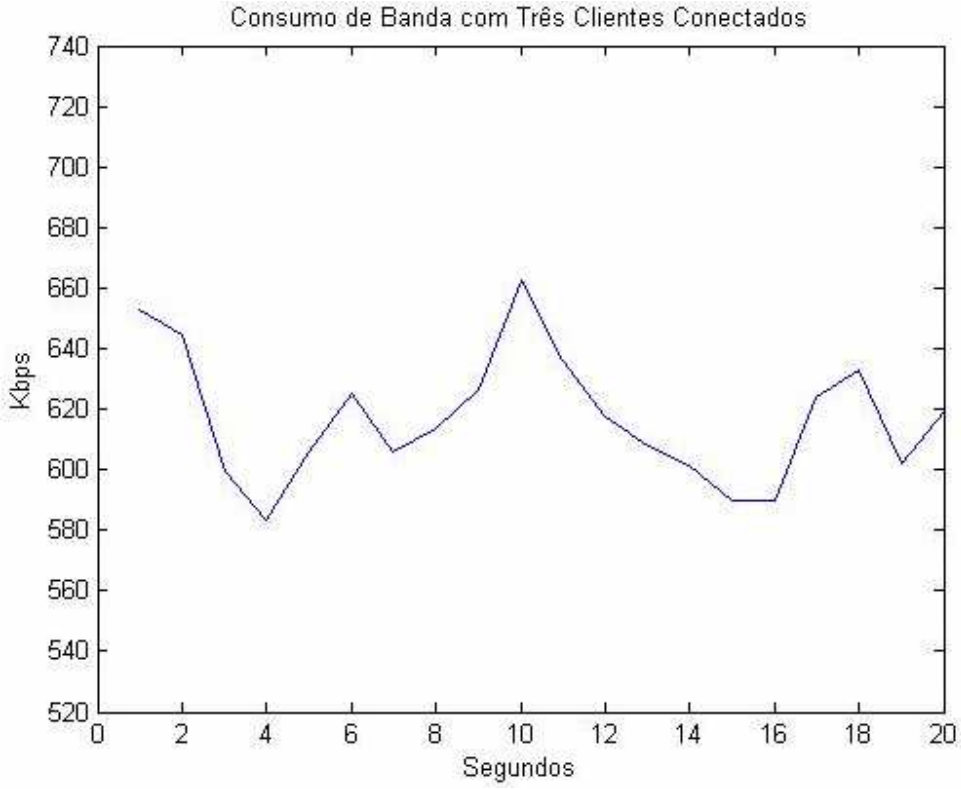

Fig. 5.20: Gráfico do consumo de banda com três clientes conectados.

O segundo teste é apresentado na Fig. 5.19, que mostra o consumo de banda com dois clientes conectados ao servidor. Neste segundo teste podemos ver que o consumo de banda varia em torno de 400 Kbps a 460 Kbps.

Por fim a Fig. 5.20 apresenta o consumo de banda com três clientes conectados ao servidor. No gráfico da Fig. 5.20 podemos verificar que o consumo de banda varia em torno de 580 Kbps a 660 Kbps.

Capítulo 6

Conclusão

Neste trabalho propõe-se um sistema multimídia de *streaming* de áudio e vídeo baseado no modelo cliente-servidor, implementado a partir do uso das bibliotecas da linguagem Python.

Este sistema é composto de um servidor que provê serviços e de clientes que executam requisições desses serviços. O servidor, assim como o cliente, é um programa desenvolvido em linguagem Python e com a interface gráfica baseada no módulo de desenvolvimento gráfico Tkinter. Os dois componentes, contribuição prática deste trabalho, desenvolvidos para o sistema operacional Windows podem ser utilizados para estabelecimento de comunicação unidirecional de áudio e vídeo e comunicação bidirecional de mensagens de texto.

A linguagem Python utilizada no trabalho se mostrou eficiente e estável no desenvolvimento de aplicações multimídia, se mostrando uma boa alternativa dentre as existentes para esse tipo de aplicação. Além disso, Python possui uma suave curva aprendizagem. Suas estruturas são de altíssimo nível e a grande variedade de bibliotecas prontas facilitaram bastante a programação. Foi visto que com um tempo considerado relativamente pequeno, foi possível desenvolver aplicações com funções consideradas avançadas. Porém, observa-se que, algumas classes e funções ainda possuem pouca documentação.

A limitação da linguagem Python encontrada durante o andamento do trabalho foi na tentativa do uso da função "*muxer*" que supostamente faria a sincronização dos *streams* de áudio e vídeo no momento da captura, para serem transmitidos juntos. Como já foi dito, a solução encontrada para esse problema foi a transmissão em paralelo dos *streams.* O ponto positivo dessa implementação é que como os pacotes de áudio e vídeo são

independentes, caso se perca a comunicação de vídeo, a comunicação de áudio continua, conforme visto nos testes em laboratório.

O uso de programação paralela (*threads*) proporcionou ao sistema trabalhar de forma assíncrona podendo assim o servidor tratar de várias requisições ao mesmo tempo.

Na programação gráfica (desenvolvimento da interface gráfica) foi proposto a utilização do módulo de desenvolvimento gráfico Tkinter. Este módulo mostrou possuir muitos recursos, permitindo ao programador saber em detalhes cada etapa do desenvolvimento, uma vez que deve elaborar o código completamente.

Um aspecto positivo da implementação em Python, demonstrado com o uso do tocador VLC como parte do cliente, é que as informações transmitidas na rede podem ser reconhecidas por *softwares* comerciais desenvolvidos em outras linguagens, como era de se esperar, uma vez que são utilizados protocolos padrões de transporte.

Nos testes realizados em laboratório, numa rede superdimensionada, mostramos que o uso do protocolo UDP não comprometeu a reprodução sincronizada do áudio e do vídeo na parte cliente da aplicação.

No contexto teórico, a contribuição deste trabalho é uma pesquisa bibliográfica baseada em livros, artigos publicados, páginas de Internet e normas técnicas. A partir desta pesquisa foram trabalhados vários conceitos relacionados com os temas:

- Captura, codificação e compressão de áudio e vídeo;
- Transmissão de áudio e vídeo;
- Programação Paralela (*Threads*) e Programação Gráfica;

6.1 Sugestões Para Trabalhos Futuros

- Utilizar o *Real Time Protocol* (RTP) [29] para melhorar as condições de tráfego dos pacotes na rede. Para isso será necessário implementar funções do protocolo RTP na linguagem C/C++ e fazer a ligação com a linguagem Python através da *Python/C API.* Testes para a ligação das linguagens já estão sendo desenvolvidos.
- Implementar o *Session Initiation Protocol* (SIP). O SIP [30] é o protocolo geralmente empregado no controle e estabelecimento de sessões multimídia. Cada

usuário terá um endereço SIP, permitindo assim verificação de localização e disponibilidade do usuário, além da determinação de parâmetros da mídia a ser trocada entre as partes.

- Tornar a transmissão do áudio e do vídeo bidirecional, já que atualmente apenas a comunicação de mensagens de texto é bidirecional. Portanto, o sistema deixaria de possuir arquitetura cliente servidor e passaria a possuir a arquitetura *peer-to-peer* (P2P), já que não haveria distinção entre a entidade que está provendo ou recebendo serviços. Além disso, os participantes passariam a compartilhar parte dos seus recursos de hardware [14]. Porém seria interessante antes dessa etapa, utilizar o protocolo RTP para ter um melhor controle dos pacotes na rede.
- Utilizar o sistema proposto para aplicações de IPTV. Em IPTV o conteúdo é enviado apenas em *streaming*, porém com garantia de qualidade na entrega. O receptor é uma aparelho chamado *set-top box*, que é conectado a televisão.

Está em desenvolvendo um artigo [38] sobre o trabalho proposto, que deverá ser submetido para algum congresso na área de transmissão multimídia.

Referências Bibliográficas

[1] Snow, K. **Streaming Video Over the Network.** Disponível em: <www.fullofarticles.com/Article/Streaming-Video-Over-TheNetwork/16963>. Acesso em: 01 fev. 2007.

[2] Página da Rede KyaTera. Disponível em:<http://www.kyatera.fapesp.br>. Acesso em: 01 abril 2007.

[3] Luo, J. G, Tang, Y., Zhang, M., Zhao, L. e Yang, S.Q. **Design and Deployment of a Based IPTV System over Global Internet.** IEEE First International Conference on Communications and Networking in China, 2006. ChinaCom '06.

[4] Python Beginner's Guide. Disponível em: <http://wiki.python.org/moin/BeginnersGuide/Overview> . Acesso em: 01 junho 2007

[5] Python License. Disponível em:<http://www.python.org/psf/license.html>. Acesso em: 01abril 2007.

[6] Página da Comunidade Python no Brasil. Disponível em: <www.pythonbrasil.com.br> Acesso em: 01 março 2007.

[7] The Python home page. Disponível em: <http://www.python.org/download/release//2.3.2/license/. Acesso em: 01 julho 2007.

[8] Lidstrom, G. **Programming With Python**. IT Professional

[9] Raghavan, S. V. e Tripathi, S. K. **Networked Multimedia Systems – Concepts, Architecture and Design.** New Jersey: Ed. Prentice Hall, 1998, p. 1 – 17.

[10] Barbedo, J. G. A. **Avaliação Objetiva de Qualidade de Sinais de Áudio e Voz** Tese de Doutorado, Unicamp, 2004.

[11] Chun , W. J. **Core Python Programming**. Prentice Hall, First Edition, 2000, p.742 – 777.

[12] Apostolopoulos, T. K. e Pramataris, K. C. **A Client-Server Architecture for Distributed Problem Solving**. Networks, 1995. Theme: 'Electrotechnology 2000: Communications and Networks'. [in conjunction with the] IEEE Singapore International Conference on Information Engineering.

[13] Davey, B. e Tatnall, A. **Tools for Client Server Computing.** Software Engineering: Education and Practice, 1996. Proceedings.IEEE International Conference

[14] Schollmeier, R. **A Definition of Peer to Peer Networking for the Classification of Peer to Peer Architectures and Applications.** First International Conference on Peer to Peer Computing, 2001. Proceedings.

[15] Dumas,A. **Programando WINSOCK**. Rio de Janeiro: AXCEL Books, 1995 p.39-42

[16] Labaki, J. Introdução a Python – Módulo A. Universidade Estadual Paulista

[17] Grigoletti, P. S. " Funcionalidades da Linguagem Python para Programação Distribuída ", Instituto de Informática – Universidade Federal do Rio Grande do Sul.

[18] Martelli, A. **Python in a Nutshell**. O'Reilly, 2003. p. 257 – 262.

[19] Van Rossum, G. e Drake, F.L. **Extending and Embedding the Python Interpreter.** Disponível em: <http://docs.python.org/ext/ext.html>. Acesso em: 01 julho 2006.

[20] "The Jython home Page".Disponível em: <http://www.jython.org/Project/index.html>. Acesso em: agosto 2007.

[21] "The IronPython home page". Disponível em: <www.codeplex.com/?IronPython>. Acesso em: agosto 2007.

[22] Mcmillan, G. **Sockets Programming HowTo.** Dísponivel em: <www.amk.ca/python/howto/sockets>. Acesso em: 01 out. 2006.

[23] Postel, J. **User Datagram Protocol – RFC 768.** Internet Engineering Task Force, Agosto 1980.

[24] “The VideoCapture home page”Disponível em <http://videocapture.sourceforge.net> Acesso em: 01 julho 2007.

[25] “The PyMedia home page.” Disponível em: <http://www.pymedia.org>. Acesso em: 01 setembro 2007.

[26] Internet Assigned Numbers Authority. Disponível em: < www.iana.org/assignments/ port- numbers>. Acesso em: 01 agosto 2006.

[27] “The Video Lan home page.” Disponível em: <http://www.videolan.org>. Acesso em: 01 agosto 2007.

[28] “The Ethereal home page.” Disponível em:< http://www.ethereal.com>. Acesso em: 01 maio 2007.

[29] Schulzrinne,H.; Casner, S.; Frederick, R. e Jacobson,V.: **RTP: a transport protocol for real – time applications – RFC 1889**. Internet Engineering Task Force, Janeiro 1996.

[30] Rosenber, J.; Schulzrinne, H.; Camarillo, G.; Johnston, A.; Peterson, J.; Sparks, R.; Handley, M. e Schooler, E. **SIP: Session Initiation Protocol – RFC 3261.** Internet Engineering Task Force, Junho 2002.

[31] Memon, A. M., Pollack, M. E. e Soffa M. L. **Hierarchical GUI Test case Generation Using Automated Planning.** IEEE Transactions on Software Engineering, 2001.

[32] Hsieh, H.Y. e Sivakumar, R.**Parallel transport: a new transport layer paradigm for enabling Internet quality of service.** IEEE Communications Magazine, April 2005

[33] Baset, S. A. e Schulzrinne H. G. **An Analysis of the Skype Peer-to-Peer Internet Telephone Protocol.** 25th IEEE International Conference on Computer Communications. INFOCOM 2006.

[34] Lisha, G. e Junzhou, L. **Performance Analysis of a P2P Based VoIP Software.** International Conference on Internet and Web Applications and Services / Advanced International Conference on Telecommunications, 2006. AICT-ICIW '06

[35] Boll, S. **Multi-Tube - - Where Multimedia and Web 2.0 Could Meet.** Multimedia, IEEE. Volume 14, April-June 2007 p. 12-15

[36] The Youtube home page. Disponível em: <http://www.youtube.com>. Acesso em: 01 novembro 2007

[37] The Skype home page. Disponível em: <http://www.skype.com>. Acesso em: 01 outubro 2007

[38] Gonçalves, J. N. e Costa, M. H. M. **Desenvolvimento de uma Plataforma Multimí - dia Utilizando a Linguagem Python** (Em desenvolvimento).

Apêndice A. Funções do programa Servidor de Vídeo:

Este apêndice apresenta os principais trechos de código fonte do programa servidor de vídeo.

Funções de captura e transmissão do Vídeo:

```
def CaptureImage(self,e):
    global terminated
    global timer
    self.img = cam.getImage()
    s = pygame.image.fromstring(self.img.tostring(), self.img.size, self.img.mode)
    # Criando um frame
    ss = pygame.image.tostring(s, "RGB")
    bmpFrame = vcodec.VFrame( vcodec.formats.PIX_FMT_RGB24, s.get_size(),
    (ss,None,None))
    yuvFrame = bmpFrame.convert( vcodec.formats.PIX_FMT_YUV420P )
    self.d = self.e.encode( yuvFrame )
    #Envio dos dados
    self.mySocket.sendto(self.d.data,(self.addr_v[0],1234) )
    if not terminated:
       self.timer = threading.Timer(0.03,self.CaptureImage, [e])
       self.timer.start()

def makeVideo(self):
    global timer

   #parametros
    bitrate= 7000000
```

```
    frame_rate=3100
    img = cam.getImage()
    s = pygame.image.fromstring(img.tostring(), img.size, img.mode)

   params= { \
        'type': 0,'gop_size': 12,'frame_rate_base': 125,
        'max_b_frames': 0,'height': s.get_height(),'width': s.get_width(),
        'frame_rate': frame_rate,'deinterlace': 0,'bitrate': bitrate,'id': vcodec.getCodecID(
        'mpeg1video' )
     }
    #criando o codificador
    self.e = vcodec.Encoder( params )
    self.timer = None
    self.CaptureImage(self.e)
    time.sleep(86400)
    self.timer.cancel()
```

Funções de captura e transmissão do Áudio:

```
def voiceRecorder(self):
    #parametros
    bitrate=128000
    sample_rate=44100
    channels=2
    aparams= { 'id': acodec.getCodecID( 'mp3' ),'bitrate': bitrate,'sample_rate': sample_rate,
                'channels': channels }
    #criando o codificador
    self.ac= acodec.Encoder( aparams )
    self.snd= sound.Input( 44100, 2, sound.AFMT_S16_LE )
    self.snd.start()
```

```
    global terminated
    while not terminated:
      self.s= self.snd.getData()
      if self.s and len( self.s ):
       for fr in self.ac.encode( self.s ):
         #Envio dos dados
         self.mySocket2.sendto( fr,self.addr_a)

       self.snd.stop()
```

Funcões da Gravacão do Áudio:

```
  def voiceRecorder_save( self ):
    f= open( self.name_audio, 'wb' )
    #parametros
    bitrate=128000
    sample_rate=44100
    channels=2
    aparams= { 'id': acodec.getCodecID( 'mp3' ),'bitrate': bitrate,'sample_rate': sample_rate,
    'channels': channels }

    #criando o codificador
    ac= acodec.Encoder( aparams )
    snd= sound.Input( 44100, 2, sound.AFMT_S16_LE )
    snd.start()

    while snd.getPosition()<= self.time_audio:
      print "Gravando Audio"
      s= snd.getData()
      if s and len( s ):
        for fr in ac.encode( s ):
```

```
            f.write( fr )
        else:
            time.sleep( .003 )

    snd.stop()
```

Funções de Gravacão do Vídeo:

```
    def CaptureImage1(self, e, fw):
        global timer
        print 'Gravando Video'
        img = cam.getImage()
        s = pygame.image.fromstring(img.tostring(), img.size, img.mode)
        ss = pygame.image.tostring(s, "RGB")
        bmpFrame = vcodec.VFrame( vcodec.formats.PIX_FMT_RGB24, s.get_size(),
       (ss,None,None))
        yuvFrame = bmpFrame.convert( vcodec.formats.PIX_FMT_YUV420P )
        d = e.encode( yuvFrame )
        fw.write( d.data )
        timer = threading.Timer(0.03,self.CaptureImage1, [e, fw])
        timer.start()

    def Video( self ):
        global timer
        bitrate= 7000000
        img = cam.getImage()
        s = pygame.image.fromstring(img.tostring(), img.size, img.mode)
        params= { \
          'type': 0,'gop_size': 12,'frame_rate_base': 125,
          'max_b_frames': 0,'height': s.get_height(),'width': s.get_width(),
          'frame_rate': 2997,'deinterlace': 0,'bitrate': bitrate,
```

```
  'id': vcodec.getCodecID( 'mpeg1video' )
  }
print 'Setting codec to ', params
e = vcodec.Encoder( params )
t = time.time()

fw= open( self.name_video, 'wb' )

timer = None
self.time_video1=self.time_video-(self.time_video*0.1)
self.CaptureImage1(e, fw)
time.sleep(self.time_video1)
timer.cancel()
fw.close()
```

Apêndice B. Código fonte do programa Cliente de Vídeo:

Este apêndice apresenta os trechos do código fonte do programa cliente de vídeo.

Funções de requisição e reprodução do áudio:

```
def audioReproduce(self):
    cparams= { 'id': acodec.getCodecID( 'mp3' ),
       'bitrate': 128000,
       'sample_rate': 44100,
       'channels': 2 }

    self.dc= acodec.Decoder( cparams )
    self.dm= muxer.Demuxer( 'mp3' )
    self.bytesComing=0

    self.sndOut= sound.Output( 44100, 2, sound.AFMT_S16_LE )

    arquivo= open( "buffer.mp3", 'wb' )

    self.f=open("buffer.mp3", 'rb' )
    print 'Audio'

    global terminated
    while not terminated:
      (fr,addra)=self.cliente2.recvfrom(999999)
      arquivo.write(fr)
      self.bytesComing+=len(fr)
      self.s=self.f.read(self.bytesComing)
```

```
            self.frames= self.dm.parse( self.s )
            for fra in self.frames:
                self.r= self.dc.decode( fra[ 1 ] )
                self.sndOut.play( self.r.data )

        arquivo.close()
```

Funções de requisição e reprodução do Vídeo:

```
self.cliente.sendto("Video",self.ponto)
os.startfile('C:\Python24/teste_vlc.bat')
```

Testes Subjetivos: Avaliação do Sistema

Foram realizados testes com 3 voluntários para avaliação do sistema. Os voluntários assistiram em diferentes momentos a 10 minutos da transmissão do áudio e vídeo do sistema proposto e responderam algumas questões através de notas. Todos os testes foram realizados em laboratório.

As questões foram:

1 - Como foi a reprodução da seqüência de vídeo no cliente?
2 - Como foi a reprodução da seqüência de áudio no cliente?
3 – Como foi a sincronização das seqüências?
4 – Como se apresentou o áudio armazenado no servidor?
5 – Como se apresentou o vídeo armazenado no servidor?

As respostas dos voluntários se basearam na tabela de descrição de notas abaixo.

Nota	Definição	Descrição
5	Excelente	Sinal Perfeito
4	Bom	Alta Qualidade
3	Razoável	Requer Esforço
2	Pobre	Baixa Qualidade
1	Ruim	Sinal Quebrado

Tabela 1: Descrição das notas

As respostas dos voluntários foram:

Voluntário 1:
Questão 1 – 4
Questão 2 – 5

Questão 3 – 4

Questão 4 – 5

Questão 5 – 5

Voluntário 2:

Questão 1 – 3

Questão 2 – 5

Questão 3 – 3

Questão 4 – 5

Questão 5 – 5

Voluntário 3:

Questão 1 – 4

Questão 2 – 4

Questão 3 – 3

Questão 4 – 5

Questão 5 – 5