STÉFANO HERNANI DOS SANTOS
ROBERTO BORGUES JUNIOR

DESENVOLVIMENTO DE UMA APLICAÇÃO VOZ SOBRE IP

Monografia apresentada à Faculdade de Ciência da Computação do Centro Universitário Eurípides de Marília, mantido pela Fundação de ensino Eurípides Soares da Rocha, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação. (Área de Concentração: Redes de Computadores).

Orientador:
Prof. Dr. Antonio Carlos Sementille

MARÍLIA
2005

SANTOS, Stéfano Hernani dos;
BORGUES JUNIOR, Roberto
Desenvolvimento de uma aplicação Voz sobre IP / Stéfano Hernani dos Santos; Roberto Borgues Junior; orientador: Antonio Carlos Sementille. Marília, SP: [s.n.], 2005.

Monografia (Graduação em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha.

1. Redes de Computadores 2. Voz sobre IP 3. Soquetes

CDD: 004.6

STÉFANO HERNANI DOS SANTOS
ROBERTO BORGUES JUNIOR

DESENVOLVIMENTO DE UMA APLICAÇÃO VOZ SOBRE IP

Banca examinadora do trabalho de conclusão de curso apresentado ao Programa de Graduação da UNIVEM,/F.E.E.S.R. para obtenção do Grau de Bacharel em Ciência da Computação. Área de Concentração: Redes de Computadores.

Resultado: _____(_____)

ORIENTADOR: Prof. Dr. _____

1º EXAMINADOR: _____

2º EXAMINADOR: _____

Marília, 07 de Dezembro de 2005

AGRADECIMENTOS

Agradecemos a Deus que, nas horas mais difíceis, sempre nos deu forças e nos momentos mais complicados, com uma luz confortadora, nos iluminou.

Agradecemos também ao nosso Orientador que com muita sabedoria e personalidade nos conduziu à realização deste trabalho.

Ninguém é tão grande que não possa aprender, nem tão pequeno que não possa ensinar.

Píndaro

SANTOS, Stéfano Hernani dos; BORGUES JUNIOR, Roberto. **Desenvolvimento de uma aplicação Voz sobre IP**. 2005. 74f. Monografia (Conclusão de Curso em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

Com a crescente utilização da comunicação entre computadores, várias aplicações neste sentido vem sendo desenvolvidas, como Chat, conferência de vídeo, comunicação de voz pela Internet entre outros. Utilizando esta nova tecnologia pode-se conseguir qualidade de voz igual e até mesmo superior à telefonia convencional e também o custo dessa comunicação é muito mais baixo. Hoje já se encontra disponível na Internet programas que fazem a comunicação de voz entre computadores de qualquer localidade para qualquer outra localidade gratuitamente. Este trabalho de Voz sobre IP aborda protocolos de transporte e comunicação, transmissão de dados pela Internet utilizando a interface sockets, e para a implementação do projeto utiliza-se a linguagem de programação C. O projeto consiste em dois programas: um programa servidor que fará o controle de conexões através de um protocolo definido pelos autores e armazenará informações sobre os conectados; e um programa cliente que fará requisições para o servidor e trocará mensagens entre outro cliente.

Palavras-Chave – Voz sobre IP, Redes de Computadores, VoIP, socket.

SANTOS, Stéfano Hernani dos; BORGUES JÚNIOR, Roberto. **Development of an application Voice over IP**. 2005. 74f. Monografia (Conclusão de Curso em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

With the increasing use of the communication among computers, some applications in this direction come being developed, as Chat, video conference, voice communication to Internet among others. Using this new technology, equal voice quality can be obtained and even though superior to the conventional telephony and also the cost of this communication is very much low. Today we find in the Internet programs available that gratuitously make the voice communication among computers of any locality for any another locality. This work of Voice over IP approaches transport protocols and communication, transmission of data for the Internet using the sockets interface and for the implementation of the project uses it C programming language. The project consists of two programs: one programs server who will make the control of connections through a protocol defined for the authors and will store information on the connected ones; e one programs client who will make solicitations for the server and will change to messages among other clients.

Keywords: Voice over IP, Computer Networks, VoIP, Sockets.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACK – ACKnowledgment

ATM – Asynchronous Transfer Mode

C – Linguagem de Programação

CCITT– Comité Consultatif Internationale Télégraphique et Téléphonique

DECnet – Digital Equipment Corporation networking

DLL – Dynamic Link Library

IBM – International Business Machines

IDP – Internet Datagram Protocol

IEEE – Institute of Electrical and Electronics Engineers

IETF – Internet Engineering Task Force

IMP – Interface Messaging Processor

IP – Internet Protocol

IPX – Internetwork Packet Exchange

ISO – International Standards Organization

LAN – Local Área Network

LAT – Local Area Transport

MP3 – MPEG Layer 3.

OSI – Open System Interconnection

PC – Personal Computer

PUP – Parc Universal Packet

RTCP – Real Time Control Protocol

RTP – Real-Time Transport Protocol

SIP – Session Initiation Protocol

SNA – Systems Network Architecture

SPX – Sequenced Packet Exchange

TCP – Transmission Control Protocol

TTL – Time To Live

UDP – User Datagram Protocol

VoIP – Voice over Internet Protocol

WAN – Wide Área Network

XNS – Xerox Network Services

SUMÁRIO

INTRODUÇÃO

As telecomunicações elétricas modernas tiveram seu início no século XIX, com o telégrafo e a telefonia. A partir desta época se teve uma grande evolução do serviço telefônico ao longo dos anos até chegar ao que se tem atualmente. Com os computadores não foi diferente. A partir de sua criação, uma série de evoluções e novas criações ocorreram para se conseguir chegar ao computador veloz, compacto e eficiente de hoje.

A criação das redes de computadores proporcionou a comunicação entre os mesmos e com a crescente utilização desta comunicação, várias aplicações neste sentido vem sendo desenvolvidas, como Chat, conferência de vídeo, comunicação de voz pela Internet entre outros.

Segundo Tanenbaum (2003), "A conta telefônica de um consumidor médio provavelmente é maior que sua conta da Internet, e assim as operadoras de redes de dados viram na telefonia da Internet um modo de ganhar um bom dinheiro extra sem terem que instalar sequer um novo cabo de fibra. Assim, se deu o início da telefonia da Internet, ou como também é conhecida voz sobre IP".

Utilizando esta nova tecnologia pode-se conseguir qualidade de voz igual e até mesmo superior à telefonia convencional e também o custo dessa comunicação é muito mais baixo. Hoje já se encontra disponível na Internet programas que fazem a comunicação de voz entre computadores de qualquer localidade para qualquer outra localidade gratuitamente.

Neste contexto, o presente projeto diz respeito ao estudo e implementação de um sistema de Voz sobre IP, o qual realizará a captura, reprodução e transmissão da voz pela Internet utilizando os serviços de sockets, a biblioteca DirectX e a linguagem de programação C.

Este trabalho começará foi organizado da seguinte forma:

No Capítulo 1, é apresentada a comunicação de voz pela Internet, bem como suas características, vantagens , desvantagens e comparações em relação ao sistema de telefonia convencional. Ainda no Capítulo 1, será descrito o protocolo RTP (Protocolo de Tempo Real) cuja finalidade é transmitir os dados multimídia. Os protocolos para estabelecimento, controle e término de conexões também serão descritos neste capítulo.

No Capítulo 2, são descritos os serviços de socket utilizando Windows Sockets 2, utilizados para fazer conexão e transferência de dados entre computadores .

No Capítulo 3 são apresentados os objetivos, estruturação, projeto lógico e detalhes da implementação do trabalho.

No Capítulo 4, é realizada uma análise de resultados obtidos a partir de testes da implementação do projeto.

Finalmente são apresentados as conclusões finais e trabalhos futuros.

CAPÍTULO 1 – COMUNICAÇÃO DE VOZ PELA INTERNET

1.1. Caracterização

A comunicação de voz pela Internet, mais conhecida como VoIP que vem do termo em inglês Voice over Internet Protocol que traduzido significa Voz Sobre o Protocolo da Internet e se caracteriza pelo Transporte da voz sob a infra-estrutura IP podendo ser uma LAN ou WAN.

A plataforma VoIP transforma os sinais analógicos da voz em sinais digitais, compactando e empacota-os, e transmite a um portal (Gateway), esse por sua vez torna a compacta-los e transforma-os novamente em sinais analógicos retransmitindo-os a um receptor.

Existem basicamente três arquiteturas para se utilizar a VoIP.

A primeira é a comunicação PC para PC, onde dois computados se computadores se comunicam sem a necessidade de algum *Gateway* ou outro componente específico além de um software que de suporte a esse tipo de comunicação.

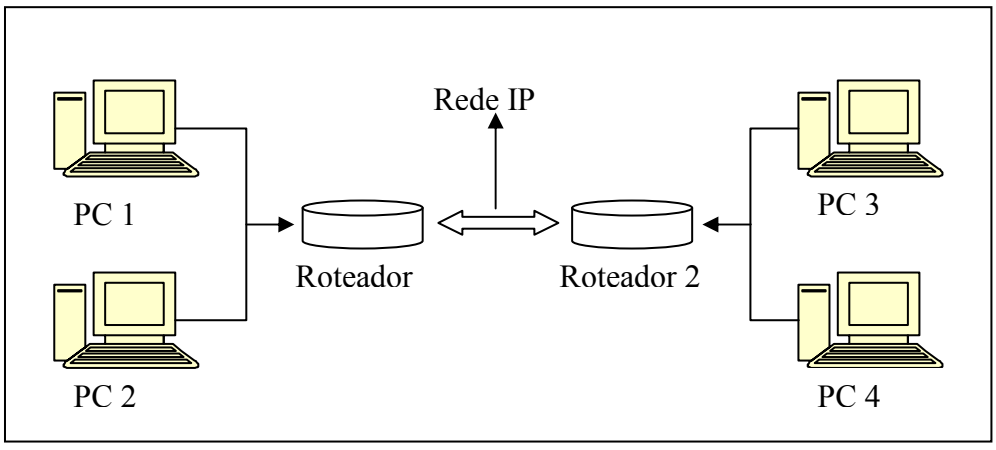

Figura 1 – Arquitetura computador para computador

A segunda forma é a comunicação PC para Telefone, onde o PC através de um software se conecta a um *gateway* e esse *gateway* se encarrega de converter os sinais digitais em sinais analógicos que serão entendidos e transmitidos pela rede de telefonia convencional.

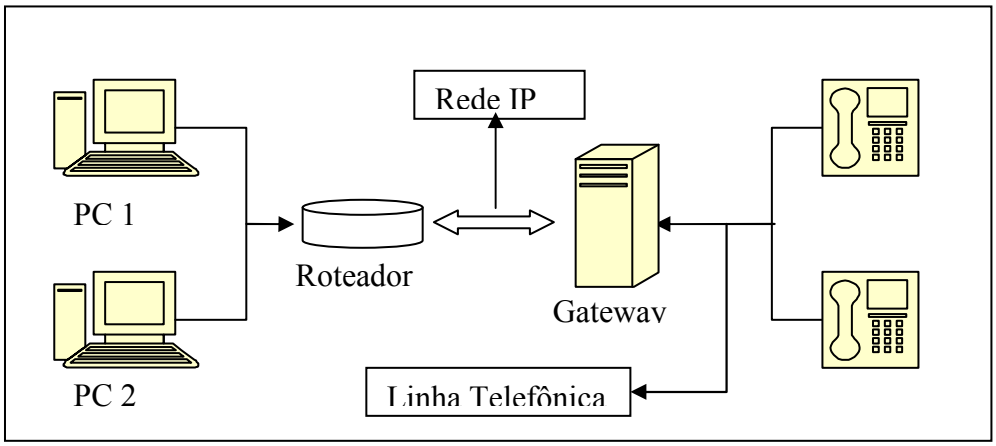

Figura 2 – Arquitetura telefone para computador

A terceira forma é a comunicação Telefone para Telefone, onde os telefones convencionais que já utilizam a arquitetura existente se conectam a um *Gateway* pela rede telefônica. Este por sua vez comunica-se com outro *Gateway* utilizando a Rede IP que se comunica com o destino utilizando a rede telefônica convencional.

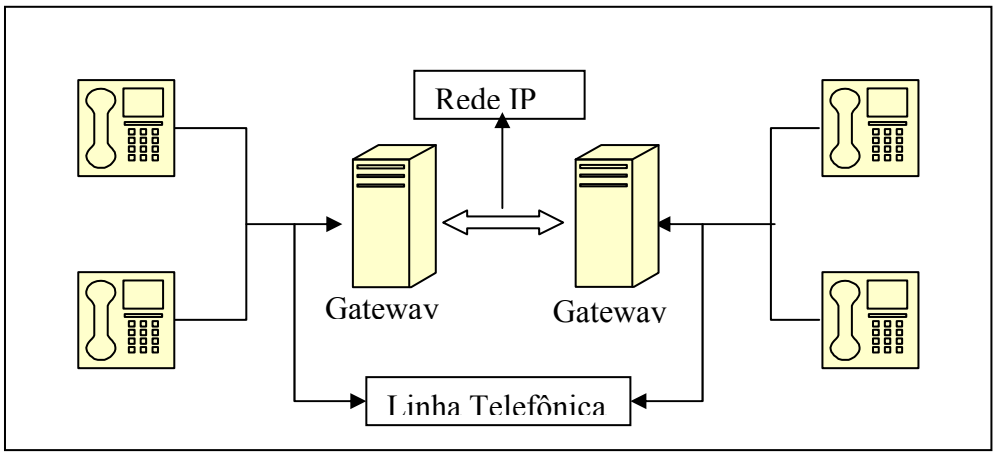

Figura 3 – Arquitetura telefone para telefone

1.2. Comparação da VoIP com relação à telefonia convencional

1.2.1. Principais Vantagens

As principais vantagens de VoIP estão relacionadas ao custo/benefício, pois é possível se reduzir os custos com telefonia como por exemplo efetuar chamadas de longa distancia a custos locais e integrar telefonia móvel com fixa. Uma outra vantagem dessa tecnologia é o fato de não haver a necessidade de se implantar uma nova estrutura física para o seu uso podendo usar a estrutura já existente no lugar implantado, caso esta já não esteja obsoleta, além do fato de não haver a necessidade de depender de uma operadora como na telefonia convencional, basta apenas um aparelho de telefonia IP e uma conexão com a internet.

Segundo CISCO (2005) "No caso da VoIP, acreditamos que mais de 70% das médias e grandes empresas já estejam se beneficiando desta tecnologia na comunicação entre matriz e filial. O Retorno de Investimento varia de 4 a 15 meses, dependendo da quantidade de tráfego de voz".

1.2.2. Desvantagens

Uma das principais desvantagens da Telefonia VoIP é o caso da estrutura física onde será implantada estar obsoleta, pois o custo de implantação dessa nova estrutura é alto. Outro ponto que ainda não favorece a telefonia IP são os atrasos. Estes acontecem geralmente por causa da compactação de descompactação dos pacotes, largura da banda e estrutura física (rede). Para um melhor desempenho é indicado o uso de Internet banda-larga.

1.3. Protocolos Utilizados na VoIP

Existem vários produtos de VoIP que possuem características variadas. Essas características geralmente são definidas por grupos que determinam os padrões empregados em redes de entrega de pacotes multimídia. São eles a ITU e IETF (Internet Engineering Task Force – Força Tarefa de Engenharia Internet). Apesar de esses dois órgãos determinarem regras e padrões para o desenvolvimento dessas aplicações, alguns fabricantes utilizam características próprias para seus produtos. Nas seções seguintes serão apresentados alguns protocolos que podem ser utilizados na VoIP, não necessariamente todos sejam utilizados em produtos específicos, cabendo muitas vezes aos fabricantes a utilizarem o protocolo que mais se adeqüe ao tipo de atividade que seu produto vai ser utilizado.

1.3.1. Protocolo RTP (*Real Time Protocol – Protocolo de Tempo Real)*

O RTP é um protocolo usado para transporte de áudio e vídeos mais comuns na internet como PCM, GSM, MP3, MPEG e H.263 e também para transporte de formatos proprietários de som e vídeo. Ele é amplamente usado em projetos e pesquisas e também complementa outros protocolos de tempo real como o SIP e o H323 (KUROSE ; ROSS, 2005).

O protocolo RTP utiliza como base para rodar o UDP. O remetente encapsula dados de mídia (voz, vídeo) em um pacote RTP que por sua vez é encapsulado em um pacote UDP que logo em seguida é transmitido ao protocolo IP. O lado que receber esse pacote extrai o pacote RTP do pacote UDP e logo em seguida extrai os dados de mídia do pacote RTP para a sua decodificação.

Uma característica desse protocolo é o fato de não haver a certeza de que o pacote será entregue, seja entregue a tempo ou que seja entregue na ordem correta, pois os roteadores não distinguem datagramas IP que carregam pacotes RTP de Datagramas que não carregam. Outra característica desse protocolo é o fato de não se destinar somente a aplicações unicast, também podem ser utilizados em multicast, como por exemplo em vídeo-conferência.

A Figura 4 descreve o cabeçalho do protocolo RTP.

Figura 4 – Cabeçalho do Protocolo RTP

O campo Tipo de carga útil possui 7 bits de comprimento e para aplicações envio de áudio serve para identificar o tipo de codificação de áudio que está sendo usado.

Número do tipo de carga útil	Formato do áudio	Taxa de amostragem	Vazão
0	PCM lei	8 kHz	64 kbps
1	1016	8 kHz	4,8 kbps
3	GSM	8 kHz	13 kbps
7	LPC	8 kHz	2,4 kbps
9	G.722	16 kHz	2,4 kbps
14	áudio MPEG	90 kHz	-
15	G.728	8 kHz	16 kbps

Tabela 1 – Tipos de carga útil suportados pelo RTP

1.3.2. Protocolo de Controle RTP (RTCP)

O protocolo RTCP é especificado pelo RFC 1889, que é um protocolo onde uma aplicação multimídia pode usar juntamente com o RTP, onde pacotes RTP são transmitidos por cada participante de uma sessão RTP utilizando o IP multcast. Eles se diferenciam pela

utilização de números de porta diferentes, sendo que o número da porta do RTCP deve ser igual ao número da porta do RTP mais uma unidade.

Os pacotes RTCP não contêm dados sobre áudio ou vídeo, eles são enviados periodicamente do remetente e/ou receptor com informações estatísticas que poderão ser úteis para a aplicação. Esses dados estatísticos incluem informações como o número pacotes enviados, número de pacotes perdidos e variação de atrasos entre chegadas (KUROSE; ROSS, 2005).

Para cada cadeia de pacotes que o receptor recebe, são gerados relatórios, onde o receptor os envia via multicast para todos os participantes da sessão. Segundo Kurose e Ross abaixo estão relacionados os principais campos do relatório:

- O SSRC da cadeia RTP para a qual o relatório de recepção está sendo gerado.
- A fração de pacotes perdida dentro da cadeia RTP. Cada receptor calcula o número de pacotes RTP perdidos divido pelo número de pacotes RTP enviados como parte da cadeia. Se um remetente receber relatórios de recepção indicando que os receptores estão recebendo apenas uma pequena fração dos pacotes transmitidos por ele, ele poderá passar para uma taxa de codificação mais baixa, com o objetivo de reduzir o congestionamento da rede e melhorar a taxa de recepção.
- O último número de seqüência recebido na corrente de pacotes RTP.
- A variação de atraso entre chegadas, que é uma estimativa amortecida da variação do intervalo de tempo entre chegadas de pacotes sucessivos na corrente RTP.

Para cada corrente RTP que um remetente estiver transmitindo, ele criará e transmitirá pacotes de relatório de remetente RTCP. Esses pacotes contêm informações sobre a corrente RTP incluindo:

- O SSRC da corrente RTP.
- A marcada de tempo e o tempo do relógio físico (tempo real) do pacote da corrente RTP gerado mais recentemente.
- O número de pacotes enviados na corrente.
- O número de bytes enviados na corrente.

O protocolo RTCP apresenta um problema de escalabilidade. Se os receptores periodicamente gerar pacotes RTCP, a taxa de transmissão de pacotes RTCP excederá a de pacotes RTP, sendo que o tráfego dos pacotes RTP não aumenta a medida que o número de receptores aumenta, ao contrário do tráfego do RTCP que aumenta linearmente á medida que o número de receptores aumenta. Para a solução desse problema o RTCP muda a freqüência que um participante envia pacotes RTCP. Isso se dá na tentativa de limitar seu tráfego a 5 por cento da largura da banda.

1.3.3. Protocolo H.323

O H.323 é uma alternativa ao protocolo SIP, que será descrito adiante, e um padrão popular utilizado em vídeo-conferência. Esse conjunto de recomendações consiste em uma série de especificações que definem funções de sinalização e também formatos relacionados a áudio e vídeo.

Redes H.323 consistem de *gatekeepers* e *gateways*. Os *gatekeepers* funcionam com um concentrador de chamadas. É ele que vai receber as chamadas, gerenciar a largura da

banda e a sinalização das chamadas. Os *gateways* funcionam como um ponto final de uma terminação H.323.

A Figura 5 mostra a arquitetura de uma rede H.323.

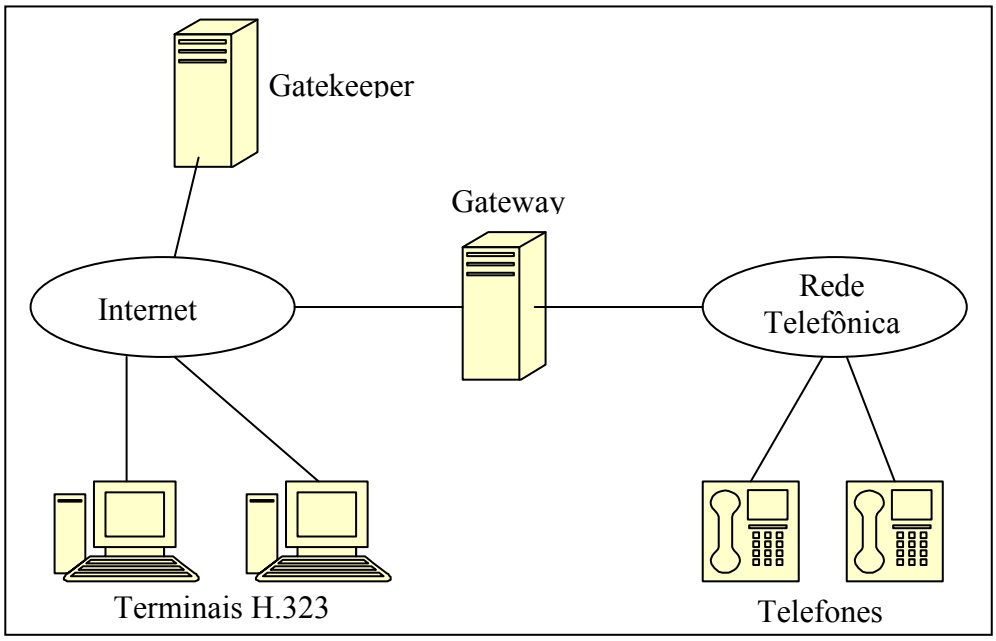

Figura 5 – Arquitetura da uma rede H.323.

Apesar de não ser obrigatório, um *gatekeeper* pode ajudar as redes H.323 na sua escalabilidade, separando chamadas e ajudando os *gateways* em sua função de gerência.

O H.323 não possui uma grande fatia do mercado entre fornecedores de serviços, mas na utilização VoIP empresariais detém uma boa posição.

1.3.4. Protocolo SIP

O protocolo SIP (*Session Initiation Protocol* – Protocolo de Inicialização de Sessão) é um protocolo recentemente definido para aplicações em telefonia IP. Baseado em código texto, foi definido no RFC 3261 e consistem de uma camada de aplicação e um protocolo de controle que permite criar, controlar e terminar sessão entre dois ou mais usuários.

Com o SIP o usuário não precisa saber o endereço IP da pessoa com quem quer iniciar uma conversa, basta saber o endereço SIP dessa pessoa que pode ser algo do tipo nome@dominio.com. O SIP utiliza portas diferentes para mandar as mensagens de convite para uma conversa das portas utilizadas para envio de dados multimídia.

Uma mensagem de convite SIP ficaria dessa forma:

```
INVITE sip:usuário@dominio.com SIP/2.0
VIA: SIP/2.0/UDP 267.180.112.24
From: sip:usuario2@dominio.com
Call-ID: 0d2a@dominio.com
Content – Type: application/sdp
Content-Length: 885
C=IN IP4 167.180.112.24
```

Figura 6 – Mensagem de convite SIP.

Item	**H.323**	**SIP**
Projetado por	ITU	IETF
Compatibilidade com PSTN	Sim	Ampla
Compatibilidade com internet	Não	Sim
Arquitetura	Monolítica	Modular
Completeza	Pilha de protocolos completa	O SIP lida apenas com a configuração
Negociação de parâmetros	Sim	Sim
Sinalização de chamadas	Q3931 sobre TCP	SIP sobre TCP ou UDP
Formato de mensagens	Binário	ASCII
Transporte de mídia	RTP/RTCP	RTP/RTCP
Chamadas de vários participantes	Sim	Sim
Conferência de multimídia	Sim	Não
Endereçamento	Número de host ou telefone	URL
Término de chamadas	Explícito ou encerramento por TCP	Explícito ou por timeout
Transmissão de mensagens instantâneas	Não	Sim
Criptografia	Sim	Sim
Tamanho do documento de padrões	1.400 páginas	250 páginas
Implementação	Grande e complexa	Moderada
Status	Extensamente distribuído	Boas perspectivas de êxito

Tabela 2 – Comparação entre o H.323 e SIP

CAPÍTULO 2 – A INTERFACE SOCKET

Um socket pode ser definido como um ponto final de uma comunicação entre dois programas que estão sendo executados na rede. A interface socket disponibiliza um conjunto de primitivas de transporte que são amplamente utilizadas em programação para a Internet (TANENBAUM, 2003).

O sistema operacional Windows fornece a uma biblioteca completa de socket para a programação na linguagem C disponível em wsock32.lib. A seguir serão descritas as principais funções e estruturas utilizadas para a programação com socket e uma explicação de seu funcionamento.

Os dois principais tipos de socket utilizados são socket Stream (TCP) e socket Dgram (UDP) que serão descritos nas seções seguintes. Como algumas primitivas e estruturas de dados são comuns aos dois tipos de socket, as características que os diferem serão descritas nas seções 2.5 e 2.6, o restante são utilizados para ambos os tipos.

2.1. O cabeçalho winsock2.h

A interface socket para Windows já está em sua segunda versão. A primeira é a versão 1.0 logo depois foi criada a versão 1.1, e posteriormente aperfeiçoamentos foram feitos para que se chegasse a versão 2 na qual o projeto foi baseado.

Primeiramente para começar um programa utilizando socket na linguagem de programação C é necessário incluir um cabeçalho para winsock2.h no inicio do programa, como mostra a Figura 7. Este cabeçalho contém todas as estruturas e funções utilizadas em socket.

```
#include <winsock2.h>
          .
          .
          .
```

Figura 7 – O cabeçalho winsock2.h

2.2. Inicializando a DLL do socket Windows

No início de um programa que utiliza socket é necessário inicializar a DLL do socket Windows. Esta inicialização é importante pois, todo programa que utiliza esta DLL pode querer fazer uma chamada da função WSAStartup, que verifica qual a versão do socket Windows instalada no computador. A Figura 8 mostra como fazer esta inicialização.

```
WORD versao;
WSADATA data;
versao = MAKEWORD( 2,0 );
int err;
err = WSAStartup (versão, &data);
if( err != 0)
     printf  ("ERRO DE VERSÃO\n");
```

Figura 8 – Inicializando a DLL do socket Windows.

Onde *versão* é o número da versão, no caso foi utilizada a versão 2.0, e isso exigirá que o computador tenha a versão 2.0 da DLL do socket Windows.

2.3. Criando um socket

Depois de feita a inicialização da DLL do socket é necessário criá-lo. Para isso, é feita a utilização da estrutura SOCKET, mas esta ação apenas declara um socket. Para

inicializar um socket faz-se o uso da função socket(), que retorna um socket com as propriedades passadas por parâmetro. A figura 9 mostra a declaração e inicialização de um socket.

```
SOCKET s;
s = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
     if (s == INVALID_SOCKET)
          printf ("ERRO NA CRIAÇÃO DO SOCKET\n");
```

Figura 9 – Declarando e inicializando um socket.

O primeiro argumento da função *socket()* especifica a família de protocolos a ser utilizada, isto é, especifica como interpretar endereços quando eles são fornecidos. Atualmente existem muitas famílias de protocolos que se pode utilizar como argumento, como mostra a Tabela 3.

Família	Valor	Descrição
AF_UNIX	1	Local to host (pipes, portals)
AF_INET	2	Internetwork: UDP, TCP, etc.
AF_IMPLINK	3	Endereços Arpanet IMP
AF_PUP	4	Protocolos PUP : e.g. BSP
AF_CHAOS	5	Protocolos CHAOS
AF_NS	6	Protocolos XEROX NS
AF_IPX	AF_NS	Protocolos IPX : IPX, SPX, etc.
AF_ISO	7	Protocolos ISO
AF_OSI	AF_ISO	OSI é ISO
AF_ECMA	8	European Computer
AF_DATAKIT	9	Protocolos Datakit
AF_CCITT	10	Protocolos CCITT, X.25 etc
AF_SNA	11	IBM SNA
AF_DECnet	12	DECnet
AF_DLI	13	Direct Data Link Interface
AF_LAT	14	LAT
AF_HYLINK	15	NSC Hyperchannel
AF_APPLETALK	16	AppleTalk
AF_NETBIOS	17	Endereços NetBios-style
AF_VOICEVIEW	18	Voice View
AF_FIREFOX	19	Protocolos do Firefox
AF_UNKNOWN1	20	Alguém está usando este!
AF_BAN	21	Banyan
AF_ATM	22	Serviços Nativos do ATM
AF_INET6	23	Serviços Nativos do ATM
AF_CLUSTER	24	Microsoft Wolfpack
AF_12844	25	IEEE 1284.4 WG AF
AF_MAX	26	-

Tabela 3 – Famílias de protocolos para socket().

O segundo argumento da função *socket()* especifica o tipo do socket a ser utilizado. A Figura 9 está exemplificando o uso de SOCK_STREAM. Além deste tipo de socket, existem vários outros disponíveis na versão 2.0 como mostra a Tabela 4.

Tipo	Valor	Descrição
SOCK_STREAM	1	Socket Stream
SOCK_DGRAM	2	Socket Datagram
SOCK_RAW	3	Interface do Protocolo Raw
SOCK_RDM	4	Reliably-Delivered Message
SOCK_SEQPACKET	5	Sequenced Packet Stream

Tabela 4 – Tipos de socket.

O terceiro e último argumento da função *socket()* é responsável por especificar qual o protocolo será utilizado na aplicação. Atualmente existem 11 tipos de protocolos que podem ser utilizados no desenvolvimento de aplicações baseadas em socket. Estes protocolos são mostrados na Tabela 5.

Protocolo	Valor	Descrição
IPPROTO_IP	0	Modelo para IP
IPPROTO_ICMP	1	Control Message Protocol
IPPROTO_IGMP	2	Internet Group Management Protocol
IPPROTO_GGP	3	Gateway^2
IPPROTO_TCP	6	TCP
IPPROTO_PUP	12	PUP
IPPROTO_UDP	17	UDP
IPPROTO_IDP	22	XNS IDP
IPPROTO_ND	77	UNOFFICIAL Net Disk Proto
IPPROTO_RAW	255	Pacote Raw IP
IPPROTO_MAX	256	-

Tabela 5 – Protocolos para socket.

Depois de especificados todos os 3 argumentos, a função *socket()* retornará, em caso de sucesso, um socket. Caso ocorra algum erro, a função retornará o valor INVALID_SOCKET, que é uma constante com o valor -1.

2.4. Associando um endereço local a um socket

Depois que um socket foi criado, se faz necessário o uso de uma primitiva para estabelecer um endereço local ao mesmo (COMER, 1998).

A interface socket fornece a primitiva *bind()* para que se possa associar um endereço local a um socket, pois um socket recém-criado não contém endereço de rede (TANEMBAUM, 2003).

Esta primitiva possui três argumentos, o primeiro é o próprio socket a ser utilizado. O segundo argumento é um ponteiro para uma estrutura *sockaddr_in* que tem os detalhes do endereçamento tais como endereço IP e porta, entre outros. Esta estrutura varia de acordo com o protocolo utilizado, no caso do TCP e do UDP, a estrutura é mostrada na Figura 10.

```
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

Figura 10 – Estrutura *sockaddr_in*.

O terceiro e último argumento é o tamanho da estrutura *sockaddr_in* passada como segundo argumento.

Para inicializar a estrutura *sockaddr_in* é necessário atribuir valores para todos os seus elementos, onde, *sin_family* deve conter a família de protocolos, *sin_port* é o número da porta a ser utilizada pelo socket, *sin_addr* é onde deve ser colocado o endereço IP do computador, este endereço pode ser colocado no formato dotted decimal ou pode-se utilizar INADDR_ANY que utiliza o endereço do computador onde o programa está sendo

executado. O array sin_zero[8] não é utilizado, geralmente inicia-se este array com *0*. A Figura 11 mostra como a estrutura *sockaddr_in* deve ser inicializada.

```
struct sockaddr_in soq_addr;
soq_addr.sin_family = AF_INET;
soq_addr.sin_port = htons(4950);
soq_addr.sin_addr.s_addr = inet_addr("200.100.29.30");
     //soq_addr.sin_addr.s_addr = INADDR_ANY; outra forma:
                                              //usa o endereço
                                              //local
for(int i = 0; i < 8; i++)
{
soq_addr.sin_zero[i] = (char)0;
}
```

Figura 11 – Inicializacão da estrutura *sockaddr_in*.

"Quando na chamada ao bind, o endereço de porta é especificado como zero, cabe ao sistema escolher um endereço ocioso e associá-lo ao socket" (SOARES; LEMOS; COLCHER, 2004).

2.5. Socket Stream

Este tipo de socket é baseado no protocolo de transporte TCP/IP que é um protocolo confiável e orientado a conexão, ou seja, por ser um protocolo confiável, há uma confirmação da mensagem enviada para outra máquina através da mensagem ACK (*acknowledgment*), em relação ao protocolo TCP/IP ser orientado a conexão é importante saber que deve haver o estabelecimento de uma conexão entre duas máquinas para que haja a troca de mensagens.

2.5.1. Escutando uma porta e aceitando uma conexão

O programa destinado a receber conexões deve, antes de fazê-las, ficar à escuta de uma porta pronto para recebê-las, para isto deve-se fazer o uso da primitiva *listen()* que deixa o socket em modo passivo pronto para receber conexões (COMER, 1998).

A primitiva *listen()* exige dois argumentos, o primeiro é o socket que será utilizado e o segundo é o número de conexões pendentes que poderá ser armazenadas na fila (TANENBAUM. 2003).

Após receber um pedido de conexão é necessário aceitá-la, para isso uma outra primitiva deve ser usada. A interface socket disponibiliza a função *accept()* para que o programa possa aceitar as conexões da fila entrante criada por *listen()*. Para fazer a chamada desta função três argumentos se fazem necessários, o primeiro é o socket utilizado na chamada *listen()*, o segundo é um ponteiro para a estrutura *sockaddr_in* e o terceiro argumento é o tamanho desta estrutura, que já foram discutidos anteriormente. Em caso de erro, a função *accept()* retorna o valor *-1*.

2.5.2. Conexão com socket Stream

Como já mencionado no início da seção 2.5, o protocolo TCP é orientado a conexão, ou seja, uma máquina precisa estabelecer uma conexão com outra máquina para que haja troca de dados. Para fazer isto utilizando socket é necessário fazer o uso da função *connect()*, que serve para estabelecer a conexão entre dois sockets. (SOARES; LEMOS; COLCHER, 2004). Esta função exige três argumentos.

O primeiro argumento é o socket que será utilizado para a conexão. O segundo argumento é um ponteiro para uma estrutura *sockaddr_in*, esta estrutura é a mesma estrutura

mostrada na Figura 11, mas deverá conter o endereço da máquina na qual será feita a conexão. O terceiro e último argumento da função *connect()* é o tamanho da estrutura passada no segundo argumento.

Em caso de sucesso a função *connect()* irá retornar o valor *0*, será estabelecida uma conexão e o socket estará pronto para enviar e receber dados, caso contrário, o valor *-1* será retornado.

2.5.3. Envio e recebimento de dados com socket Stream.

A capacidade de enviar e receber dados é a principal característica dos sockets, para isso existem duas primitivas que fazem este serviço usando socket Stream. Para enviar dados, utiliza-se a primitiva *send()* e para receber utiliza-se a primitiva *recv()* (SOARES; LEMOS; COLCHER, 2004).

Tanto a primitiva *send()* quanto a primitiva *recv()* exigem quatro argumentos cada uma. O primeiro é comum às duas primitivas, este argumento é o socket utilizado para fazer o envio e o recebimento de dados respectivamente. O segundo argumento, para *send()* é o buffer com os dados a serem enviados, para *recv()* é o buffer com os dados a serem recebidos. O terceiro argumento é o tamanho do buffer passado no argumento anterior e o quarto e último argumento é um flag que geralmente é ajustado em zero e serve para controle da transmissão e recepção.

A primitiva *send()* retorna o numero de bytes enviados, por sua vez *recv()* retorna o numero de bytes recebidos. A seguir a Figura 12 ilustra o recebimento dados e o envio de dados.

```
numbytes = recv (s, buf, tam_msg, 0);

numbytes = send (s, buf, tam_msg, 0);
```

Figura 12 – As primitivas send() e recv().

2.6. Socket Dgram

O socket Dgram é baseado no protocolo de transporte UDP. Diferentemente do protocolo TCP/IP, este é protocolo não confiável e não orientado a conexão, portanto os dados enviados de um computador pata outro não terão uma confirmação de recebimento e também não haverá a necessidade de estabelecer uma conexão entre duas máquinas para que haja a troca de dados.

Portanto as primitivas *listen()*, *accept()* e *connect()* não são utilizadas na programação com socket Dgram, além disso, as primitivas *send()* e *recv()* são substituidas por *sendto()* e *recvfrom()* em socket Dgram e serão descritas a seguir.

2.6.1. Envio e recebimento de dados com socket Dgram

A interface socket disponibiliza duas primitivas para enviar e receber dados utilizando socket Dgram, estas primitivas são *sendto()* e *recvfrom()*, respectivamente.

Estas primitivas se diferenciam das primitivas utilizadas em socket Stream por possuírem dois argumentos a mais do que *send()* e *recv()*.

As funções *sendto()* e *recvfrom()*, portanto, exigem seis argumentos. Os quatro primeiros argumentos, comuns às duas funções, são os mesmos das primitivas *send()* e *recv()*,

o primeiro é o descritor do socket utilizado, o segundo é o buffer onde estão armazenados os dados, o terceiro é o comprimento do buffer de dados e o quarto é o flag. Os dois argumentos restantes são, no caso da primitiva *sendto()*, um ponteiro para uma estrutura *sockaddr_in* que contém o endereço, porta e outras informações para identificação da máquina destino e o último argumento é o comprimento da estrutura *sockaddr_in* passada no argumento anterior.

No caso de *recvfrom()*, os dois últimos argumentos são os mesmos do *sendto()*, mas o ponteiro para a estrutura *sockaddr_in* traz as informações da máquina remetente. A Figura 13 ilustra a sintaxe destas funções.

```
numbytes = sendto(s, buffer, tam_buffer, 0,
       (LPSOCKADDR)&dest, sizeof(struct sockaddr_in));

numbytes = recvfrom(s, buffer, tam_buffer, 0,
       (LPSOCKADDR)&rem, sizeof(struct sockaddr_in));
```

Figura 13 – As primitivas sendto() e recvfrom().

As primitivas sendto() e recvfrom(), se executadas com sucesso, retornam o numero se bytes enviados e o numero de bytes recebidos, respectivamente.

2.7. Finalizando um socket

No caso do socket Stream depois de estabelecida uma conexão, qualquer um dos lados podem querer se desconectar. É muito simples fazer esta desconexão, para isso a interface socket disponibiliza a primitiva *closesocket()*. Esta função possui apenas um argumento que é o próprio descritor do socket. Para a programação utilizando socket Dgram, como não há conexão, esta primitiva apenas finaliza um socket.

Também se pode optar pelo uso da primitiva *shutdown()*, esta primitiva finaliza o envio e/ou o recebimento de dados pelo socket. Ela possui dois argumentos, o primeiro é o próprio socket e o segundo é uma opção do que se deseja encerrar. Este segundo argumento pode assumir três valores e cada um tem um significado diferente, como mostra a Tabela 6.

Valor	Descrição
SD_RECEIVE	Recebimento de dados pelo socket é interrompido
SD_SEND	Envio de dados pelo socket é interrompido
SD_BOTH	Envio e recebimento de dados pelo socket são

Tabela 6 – Opções de valores para *shutdown()*.

2.8. Modelo Cliente/Servidor utilizando socket

Antes de surgir o modelo Cliente/Servidor o compartilhamento de dados era feito através da utilização de mainframes com vários terminais ligados a ele. Mas os problemas encontrados, como alto custo e rigidez, fizeram com que esta forma de compartilhamento de dados perdesse a força.

O modelo Cliente/Servidor é uma arquitetura de rede onde existem dois módulos: o servidor e os clientes. O servidor é o responsável por servir os clientes da rede com aquilo que lhe é solicitado. O cliente, por sua vez, poderá solicitar informações na qual o servidor estará programado para fornecer.

A arquitetura Cliente/Servidor é hoje uma das tecnologias mais utilizadas em ambientes corporativos. Esta arquitetura mostrou-se mais flexível devido a utilização dos micros em rede, cada vez mais complexos e versáteis, com o compartilhamento de recursos de cada uma das máquinas.

2.8.1. Modelo cliente/servidor com socket Stream

Depois que se obteve um bom conhecimento de socket Stream, já é possível se pensar em um programa Cliente/Servidor utilizando esta interface. Por se tratar de uma aplicação utilizando o protocolo TCP onde há uma necessidade de conexão, utilizar-se-á as funções descritas na seção 2.5.

Através da Figura 14 pode-se ter uma idéia de como se comportam os módulos Cliente e Servidor.

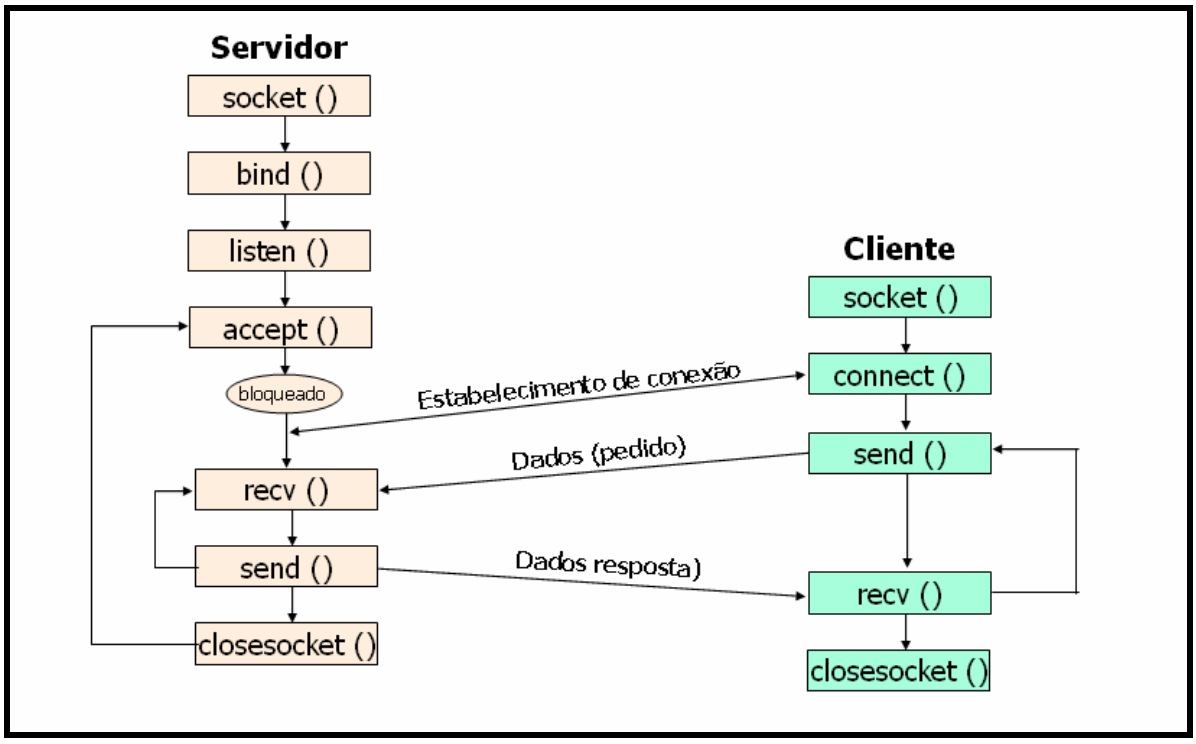

Figura 14 – Modelo Cliente/Servidor utilizando socket Stream

2.8.2. Modelo cliente/servidor com socket Dgram

Assim como é possível desenvolver um programa Cliente/Servidor utilizando socket Stream, também é possível desenvolver um programa Cliente/Servidor utilizando socket

Dgram. Por ser uma aplicação baseada no protocolo UDP, é um pouco mais simples, pois não é necessário estabelecer uma conexão entre programa cliente e o programa servidor.

O esquema utilizado para desenvolver um programa Cliente/Servidor baseado no uso de socket Dgram é o que mostra a Figura 15.

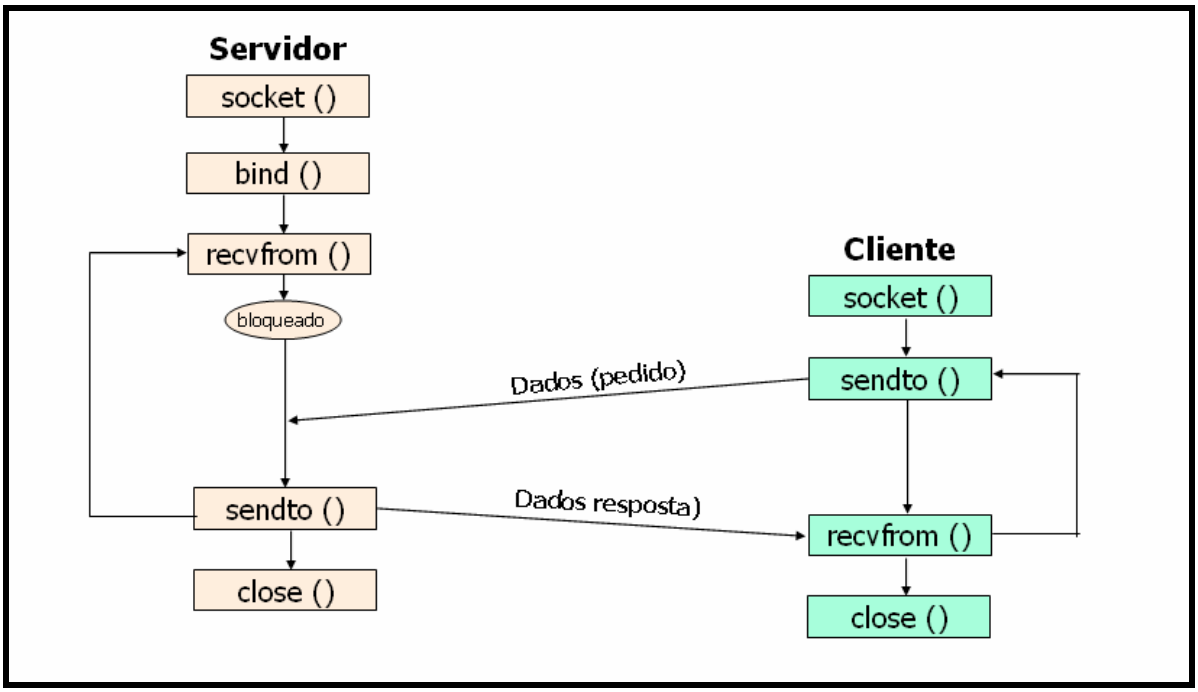

Figura 15 - Modelo Cliente/Servidor utilizando socket Dgram

2.9. Socket Multicast

Segundo Deering (1989), "O IP multicasting é a transmissão de um Datagrama IP para um *host group*, um conjunto de zero ou mais hosts identificados por um simples endereço IP destino".

Utilizando um endereço multicast é possível enviar um datagrama IP para todos os membros pertencentes ao grupo destino com a mesma confiabilidade que se tem enviando um datagrama unicast, isto é, uma vez que o protocolo utilizado é o UDP, o recebimento do

datagrama não é garantido para todos os membros do grupo destino ou na mesma ordem dos outros datagramas.

A interface socket fornece várias primitivas e estruturas de dados para que seja possível o envio de datagramas para grupos de computadores, estas primitivas serão discutidas nas subseções seguintes.

2.9.1. Endereçamento multicast

Para conseguir enviar dados para um único host basta especificar o endereço IP do computador destino e enviar os dados. Com o multicast não é diferente, a diferença é que um endereço IP estará representando um grupo de hosts.

Quando se fala em endereçamento multicast, tem-se que associar os endereços da classe D, isto é, todos os endereços IP onde os quatro primeiros bits do primeiro, dos quatro quartetos, sejam 1110.

A escala de endereços do multicast começa em 224.0.0.0 e vai até 239.255.255.255, sendo que o endereço 224.0.0.0 é reservado, ou seja, não pode ser utilizado por nenhum grupo, e o endereço 224.0.0.1 é associado para o grupo permanente de todos os hosts IP.

2.9.2. Configurando um socket para o multicasting

Para utilizar os recursos de socket enviando dados para um grupo de hosts é necessário fazer algumas configurações no socket utilizado pela aplicação.

A interface socket fornece uma primitiva que permite fazer uma séria de configurações em um socket, esta primitiva é a *setsockopt()* e possui cinco argumentos. O primeiro é o próprio socket que se deseja configurar, o segundo argumento define o nível no

qual a opção será definida, o terceiro é a opção que se deseja alterar, o quarto argumento é onde se especifica o novo valor para a opção selecionada e o quinto e último argumento é o tamanho do valor especificado no argumento anterior.

A configuração de algumas opções exige o conhecimento da estrutura de dados *ip_mreq*, que define a interface e o grupo utilizado. Esta estrutura se encontra no arquivo de cabeçalho ws2tcpip.h. A Figura 16 mostra os campos desta estrutura.

```
struct ip_mreq {
     struct in_addr imr_multiaddr;
     struct in_addr imr_interface;
};
```

Figura 16 – Estrutura *ip_mreq*

O campo *imr_multiaddr* especifica o endereço do grupo e o campo *imr_interface* especifica a interface.

Quando se utiliza a comunicação multicast o valor requerido para o segundo argumento, nível na qual a opção será definida, é sempre IPPROTO_IP. Para o terceiro argumento, existem várias opções que podem ser configuradas utilizando o nível IPPROTO_IP, as principais são mostradas na Tabela 7.

Opção	Valor	Descrição
IP_MULTICAST_IF	9	Configura o socket para o envio a um grupo multicast em uma interface particular.
IP_MULTICAST_TTL	10	Ajusta o tempo-de-vida para pacotes do multicast. O valor para esta opção no quarto argumento é sempre entre 0 e 255.
IP_MULTICAST_LOOP	11	Ajusta o loopback multicast, determinando se as mensagens transmitidas serão ou não entregues ao emissor.
IP_ADD_MEMBERSHIP	12	Adiciona ao grupo multicast um endereço especificado no quarto argumento.
IP_DROP_MEMBERSHIP	13	Remove do grupo multicast o endereço especificado no quarto argumento.

Tabela 7 – Opções de *setsockopt()* para o nível IPPROTO_IP.

Para conseguir enviar dados para um grupo multicast basta simplesmente especificar o endereço do grupo, o qual deseja enviar, na primitiva *sendto()*. Feito isso os dados serão enviados a todos os hosts pertencentes ao grupo. Para adicionar ou remover um elemento de um grupo multicast utiliza-se a opção IP_ADD_MEMBERSHIP e IP_DROP_MEMBERSHIP, respectivamente.

CAPÍTULO 3 – A BIBLIOTECA DirectX

Para a captura e reprodução da voz pelo sistema pensou-se e fazer através da biblioteca de desenvolvimento multimídia e jogos da Microsoft DirectX. Esta biblioteca consiste de API´s (*Aplication Programming Interface*) que dão suporte a desenvolvimento de aplicações 3D e 2D e programas multimídia de alta performance.

3.1. Componentes da DirectX

A biblioteca DirectX disponibiliza uma gama de componentes para as mais diversas aplicações, como gráfico e áudio. Em sua última atualização alguns componentes foram removidos como é o caso do *DirectShow* e algumas bibliotecas a Microsoft não recomenda o seu uso como o *DirectDraw*, o *DirectPlay* e o *DirectMusic*. A seguir uma breve descrição dos componentes da DirectX, pois o foco principal nesse estudo do DirectX é o *DirectSound*, componente que será analisado mais profundamente.

Com o DirectX *Graphics* é possível desenvolver gráficos 3D e 2D com agilidade e qualidade, o *DirectInput* prove suporte a diversos dispositivos de entrada como por exemplo *joysticks* e *gamepads*, por fim, o *DirectSound* possibilita o desenvolvimento aplicações de alta performance para captura e reprodução de som no formato *wave*.

3.2. Estudos sobre DirectSound

Como citado anteriormente, o componente *DirectSound* é o foco do nosso estudo sobre o DirectX. A seguir serão apresentadas algumas características do *DirectSound* como

também explicações de duas funções para desenvolvimento de aplicações utilizando a linguagem C++ .

Com o *DirectSound* é possível tocar arquivos no formato *wave*, tocas vários sons simultaneamente, controlar os dispositivos de áudio como a placa de som, adicionar efeitos 3D a sons, capturar sons por um microfone ou outro dispositivo de entrada.

Para esse tipo de aplicação que resolvemos desenvolver o *DirectSound* prove uma função que diz para a aplicação que ela será uma aplicação *FullDuplex* (captura e reprodução do som), que a função *DirectSoundFullDuplexCreate8*, contudo essa função é suportada a partir do Windows XP. Por esse motivo os dispositivos de captura e reprodução devem ser criados separadamente.

O primeiro passo para a criação de uma aplicação que utiliza *DirectSound* é a criação do dispositivo de reprodução. A Tabela abaixo mostra os objetos que são usados para a reprodução do som.

Objeto	Número	Propósito	Interfaces principais
Dispositivo	Um por aplicação	Gerenciar o dispositivo e criar os buffers	IDirectSound8
Buffer secundário	Um para cada som	Gerenciar um som estatico ou de streaming e tocar ele no buffer primário.	IDirectSoundBuffer8, IDirectSound3DBuffer8, IDirectSoundNotify8
Buffer Primário	Um em cada aplicação	Mixar e tocar sons do buffer secundário e controlar parametros globais 3D.	IDirectSoundBuffer, IDirectSound3DListener8
Efeito	Zero ou mais para cada Buffer Secundário.	Transforma o som em um Buffer Secundário	Interface para um efeito particilar, como IDirectSoundFXChorus8.

Tabela 8 – Objetos usados para a reprodução do som.

Para criar o dispositivo de reprodução utiliza-se o método *DirectSoundCreate8*. Caso não seja informado nenhum parâmetro é utilizado o dispositivo padrão. Caso o dispositivo esteja sendo utilizado por outra aplicação ou não exista, o método retorna erro que falhou. O trecho de código a seguir exemplifica a criação do dispositivo.

```
LPDIRECTSOUND8 lpds;
HRESULT hr = DirectSoundCreate8(NULL, &lpds, NULL));
```

Figura 17 – Criação do dispositivo de reprodução

Para criar o dispositivo de captura é utilizada a função *DirectSoundCaptureCreate8* que retona um ponteiro para a interface *IDirectSoundCaptureCreate8*. Após criar o dispositivo de captura também é necessário criar o buffer de captura que é obtido pelo método *IdirectSoundCapture8::CreateCaptureBuffer*. Abaixo está o cabeçalho do método e seus parâmetros.

```
HRESULT CreateCaptureBuffer(
  LPCDSCBUFFERDESC pcDSCBufferDesc,
  LPDIRECTSOUNDCAPTUREBUFFER * ppDSCBuffer,
  LPUNKNOWN pUnkOuter
);
```

Figura 18 – Cabeçalho do método *IdirectSoundCapture8::CreateCaptureBuffer*

O parâmetro *pcDSCBufferDesc* é um ponteiro para uma estrutura DSCBUFFERDESC que contem valores para o buffer de captura ser criado. O parâmetro

ppDSCBuffer é o endereço da variável que recebe um ponteiro da interface *IDirectSoundCaptureBuffer*. E o parâmetro *pUnkOuter* é o endereço para controlar a interface *IUnknown* para agregação COM.

A interface *IDirectSoundCaptureBuffer8* é usada para manipular o buffer de captura. Essa interface é obtida com a chamada dao método *IDirectSoundCapture8::CreateCaptureBuffer*. A interface *IDirectSoundCaptureBuffer8* disponibiliza os seguintes métodos para manipulação do buffer de captura:

Método	**Descrição**
Lock	Trava uma porção do buffer. Travando o buffer é retornado um ponteiro permitindo a aplicação ler ou gravar audio na memória.
Start	Inicia a captura do som dentro do buffer.
Stop	Para a captura do som dentro do buffer.
Unlock	Destrava o buffer.
GetFxStatus	Retorna o status do efeito de captura.
GetObjectInPath	Retorna uma interface de um objeto de efeito associado com o buffer.
Initialize	Inicializa o objeto buffer de captura.
GetCaps	Retorna a capacidade do buffer.
GetCurrentPosition	Retorna um ponteiro para uma variável que indica o início do buffer de captura.
GetFormat	Retorna de formato do wave do buffer de captura.
GetStatus	Retorna o status do buffer de captura.

Tabela 9 – Métodos para manipulação do buffer de captura

CAPÍTULO 4 – UM SISTEMA DE VOZ SOBRE IP

4.1. Introdução

Atualmente a forma mais utilizada para troca de mensagens de voz é através da telefonia pública convencional. Esta aplicação é importante pois fornece ao usuário uma maneira alternativa de troca de mensagens de voz. Além disso, a qualidade obtida poderá ser igual ou até mesmo superior, comparando com a telefonia convencional.

Este trabalho refere-se ao uso da tecnologia de Voz sobre IP, cuja finalidade é construir uma aplicação capaz de transmitir voz sobre o protocolo IP para a plataforma Windows.

4.2. Objetivos

Este trabalho tem como objetivos principais o estudo de técnicas e protocolos para aplicações Voz sobre IP, estruturação e implementação de uma aplicação Voz sobre IP e a validação da aplicação desenvolvida através da realização de testes e análise de resultados.

4.3. Projeto lógico do sistema

O projeto consiste em dois programas, um programa servidor que fará o controle de conexões através de um protocolo definido pelos autores e armazenará informações sobre os conectados e um programa cliente que fará requisições para o servidor e trocará mensagens de voz entre outro cliente.

Foi projetado também o arquivo de cabeçalho h_voip.h que contém todas as estruturas de dados e funções utilizadas.

Logo no início na execução do programa do usuário (cliente) este envia uma mensagem para o servidor indicando sua conexão, o servidor armazenará informações sobre o mesmo, como IP da máquina, ID do usuário, entre outras, em uma lista de conectados.

Quando um usuário deseja chamar outro usuário, este faz uma requisição para o servidor passando o ID do usuário a que deseja se conectar, o servidor imediatamente devolve as informações necessárias para o requisitante afim de que haja a transferência de mensagens entre usuários. O usuário destino receberá uma mensagem informando que há alguém tentando se conectar com ele, se ele aceitar a conexão, a partir daí se começa a transmissão de mensagens, caso contrário, o usuário requisitante receberá uma mensagem informando que sua chamada não foi aceita. A qualquer momento os usuários podem fazer uma requisição ao servidor afim de que possam receber uma tabela atualizada de todos os usuários conectados no momento.

Ao final, quando um usuário resolve se desconectar, uma mensagem é enviada para o servidor informando-o da desconexão para que ele possa removê-lo da lista de conectados. A Figura 19 representa logicamente o sistema.

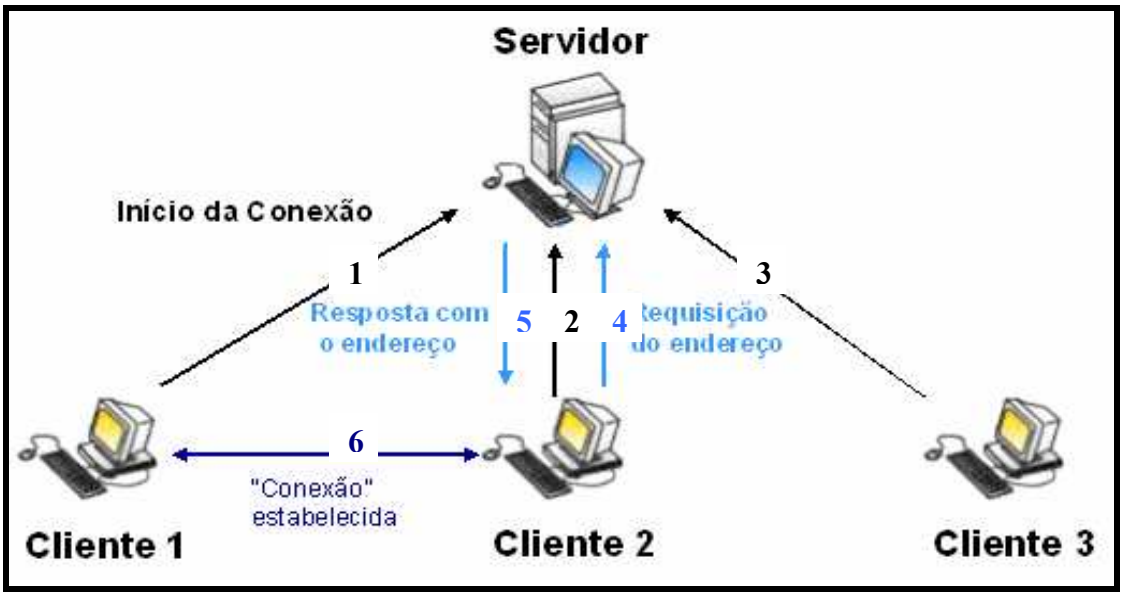

Figura 19 – Projeto lógico do sistema

4.4. O cabeçalho h_voip.h

O arquivo h_voip.h foi desenvolvido para dar suporte aos programas cliente e servidor contendo todas as estruturas de dados e funções projetadas especialmente para servi-los. A seguir serão descritas com detalhes todas as estruturas e funções contidas neste arquivo.

A estrutura **struct reg** é responsável por registrar todos os detalhes de cada cliente conectado e é composta por cinco elementos. O primeiro elemento é uma variável *flag* do tipo inteiro que sinalizará se este cliente está ou não conectado no momento e também se ele está ocupado, ou seja, em uma ligação com outro cliente. Quando o valor de flag for *1* significa que está conectado e livre para fazer e receber ligações, quando o valor for *0* significa que não está conectado e quando o valor de *flag* for *2* significa está conectado, porém em uma ligação. O segundo é uma variável ip, esta variável é do tipo *struct in*_addr e é utilizada para registrar o endereço IP do cliente conectado. O terceiro e a variável *porta* que registra o número da posta utilizada pelo cliente conectado. O quarto é um array de caracteres denominado *id*_nome, esta variável armazena o ID do cliente criado pelo próprio usuário. O quinto e

último elemento é um inteiro definido como *id_num*, esta variável funciona como um identificador do cliente e é criado pelo servidor.

É através desta estrutura que o programa servidor buscará informações sobre os usuários conectados a fim de possa enviar respostas aos clientes quando solicitado.

A estrutura **struct tabela** representa uma lista de várias estruturas do tipo *reg*, com esta estrutura o programa servidor armazenará todos os clientes que estão conectados. É composta por dois elementos, o primeiro é a variável *n_elem* que indica o número total de clientes conectados no momento e o segundo elemento é uma lista de registros do tipo *reg*.

A estrutura **struct dados** representa os dados que serão enviados e recebidos de um cliente para outro e também entre os clientes e o servidor. Possui três elementos, o primeiro é a variável *msg* que representa os dados. O segundo é um inteiro não sinalizado denominado *tipo* que representa o tipo de mensagem enviada entre o cliente e o servidor. E o terceiro é um inteiro definido como *n_seq* que representa o número de seqüência da mensagem.

O procedimento **void inicia_lista_tab(struct tabela *tab)** é utilizado pelo programa servidor e serve para iniciar a tabela ,passada por parâmetro, para o primeiro uso, ela ajusta os flags de todos os elementos para *0*.

A função **int conectar(struct tabela tab*, struct reg aux)** é utilizada pelo programa servidor para incluir na tabela de conectados o cliente que acabara de se conectar. Possui dois argumentos, o primeiro é a tabela de conectados e o segundo é um registro do cliente a ser inserido na tabela. Retorna *0* em caso de sucesso e *-1* em caso de insucesso.

O procedimento **void desconectar(struct tabela *tab, int aux)** é utilizado pelo programa servidor para retirar o cliente que acabara de se desconectar. Exige dois argumentos, o primeiro é a tabela dos conectados e o segundo é uma variável auxiliar que indicará o *id_num* do cliente para que o servidor possa removê-lo.

A função **int devolve_conectados(struct tabela tab, char conectados[])** é responsável por concatenar à variável *conectados[]* todos os IDs (apelidos) dos clientes conectados no momento de sua invocação, sempre que solicitada. Esta tabela contém o *id_nome* de todos os usuários conectados. Retorna *0* em caso de sucesso e *-1* em caso de insucesso.

A função **int devolve_ip(struct tabela tab, char nome[])** faz uma busca na tabela *tab* procurando pelo usuário conectado cujo *id_nome* seja igual a *nome[]*. Caso o usuário seja encontrado, a função retorna o índice do registro, caso contrário, o valor *-1* será retornado.

A função **int ocupado(struct tabela *tab, char nome[])** é chamada sempre que um usuário começa uma ligação com outro usuário, os dois clientes enviam uma mensagem ao servidor indicando que estão ocupados, o servidor por sua vez faz uma chamada a esta função passando o *id_nome* como argumento e muda o valor de *flag* para *2*, indicando que ele está ocupado.

A função **int desocupado(struct tabela *tab, char nome[])** é utilizada pelo programa servidor sempre que um cliente lhe envia uma mensagem informando que se desocupou, ou seja, que acabara de se desconectar de outro cliente e está pronto para receber e realizar chamadas novamente. O servidor faz a chamada desta função que muda o valor de *flag* do registro cujo *id_nome* é igual a *nome[]* para *1*.

4.5. Protocolo de comunicação entre Cliente/Servidor

Para estabelecer uma maneira para que programa servidor consiga identificar as requisições dos clientes e os clientes possam identificar as mensagens de outros clientes, foi projetado um protocolo próprio de comunicação entre os programas cliente e servidor. Assim, quando um cliente envia uma mensagem para o servidor, este por sua vez identificará

o tipo da requisição através do valor contido na variável *tipo*, presente na estrutura *dados* do arquivo h_voip.h.

A Tabela 10 mostra como o programa servidor conseguirá identificar as mensagens.

Descrição	**Tipo da Mensagem**	**0 = sucesso -1 = fracasso**	**Resposta**
Pedido de conexão com o Cliente/Servidor	2	0 ou -1	-
Pedido do endereço IP de um cliente	3	0 ou -1	ID do usuário destino
Tabela de conectados	4	0 ou -1	Tabela de conectados
Conexão Cliente/Cliente	5	-	-
Desconexão Cliente/Cliente	6	-	-
Desconexão com servidor	Demais valores	-	-

Tabela 10 – Protocolo de comunicação

4.6. O programa servidor

O programa servidor é de extrema importância para o projeto, pois é ele que dará suporte aos clientes respondendo as requisições e atuando como intermediário na comunicação entre os clientes. Durante a execução do programa servidor, todos os clientes poderão interagir com ele através de mensagens (requisições), por sua vez, o servidor está programado para respondê-las. Sempre que necessário, o servidor poderá invocar as funções e estruturas de dados contidas no arquivo de cabeçalho h_voip.h

No início de sua execução, o programa servidor fará uma série de inicializações. Depois desta etapa, entrará em um loop infinito para que ele sempre possa esperar por

requisições dos clientes. Incluído nas mensagens que os clientes enviam para o servidor sempre estará o elemento *tipo*, que é uma variável do tipo inteiro. É através do valor desta variável, definido pelo protocolo de comunicação entre o cliente e o servidor, que o programa servidor identificará o tipo do serviço que o cliente está requerendo.

Quando o servidor recebe uma requisição de um cliente que deseja se conectar, ele faz uma chamada à função *conectar()*, do arquivo h_voip, incluindo o cliente requisitante em uma tabela de conectados. Caso contrário, se por algum motivo não for possível concluir com sucesso esta requisição, o servidor retornará para o cliente uma mensagem de erro e o cliente terá que tentar se conectar novamente.

O servidor também poderá receber uma requisição pedindo uma tabela atualizada de usuários conectados. Logo depois de interpretar esta mensagem, o servidor invocará a função *devolve_conectados()*. Se houver algum erro na chamada desta função, ou por algum motivo o servidor não conseguir responder a esta pedido, uma mensagem de erro será enviada para o cliente requisitante. Em caso de sucesso, uma tabela atualizada contendo os IDs dos usuários conectados será despachada para o cliente.

Sempre que um cliente desejar se comunicar com outro cliente, primeiramente ele deve enviar uma requisição para o programa servidor a fim de que possa obter o endereço IP e outras informações para que possa haver esta comunicação. O servidor, ao receber esta requisição, faz uma busca na lista de usuários conectados, através de uma chamada à função *devolve_ip()*, para que possa obter as informações deste usuário. Se o usuário for encontrado, o servidor retornará um registro com todas as informações do usuário para o cliente que o requisitou, caso contrário, uma mensagem de erro será enviada.

A partir do momento em que o servidor envia o endereço de um cliente para outro cliente, o cliente que o receber já é capaz de realizar a comunicação com o outro cliente, e o servidor não mais intervirá entre esta comunicação.

O programa servidor também é responsável por registrar quando um cliente está ou não em uma ligação com outro cliente através de chamadas às funções *ocupado()* e *desocupado()*, respectivamente. Assim, se um cliente quiser chamar um outro cliente que já está em uma ligação, o servidor enviará uma mensagem para este cliente informando-o que o usuário que chamara está em uma ligação.

Finalmente, o servidor pode receber uma requisição de um cliente pedindo a desconexão do mesmo. No momento em que isto ocorrer, o servidor identificará o cliente requisitante e fará uma invocação da função *desconectar()*, que por sua vez irá retirar o elemento da tabela de conectados.

4.7. O programa cliente

O programa cliente terá a função de interagir com o programa servidor através de requisições e logicamente também terá a possibilidade de se comunicar diretamente com outros clientes.

Logo no inicio de sua execução, o programa cliente faz uma série de inicializações e logo depois enviará um pedido de conexão para o servidor juntamente com seu ID, a fim de que seja registrado na tabela de conectados. Em caso de sucesso, o servidor retornará uma mensagem contendo o *ID_num* no qual o cliente foi registrado.

Após esta etapa, o usuário já poderá fazer uma série de requisições para o servidor. A qualquer momento o cliente poderá requisitar ao servidor uma tabela atualizada dos usuários conectados, em caso de sucesso o servidor retornará ao cliente uma tabela contendo todos os usuários conectados, caso contrário, o cliente receberá uma mensagem de erro.

Para um cliente fazer uma chamada a outro cliente, é enviada uma mensagem para o programa servidor, contendo o ID do usuário, requisitando o endereço IP do mesmo. O

usuário não tem conhecimento deste endereço. Após a obtenção do endereço IP, o usuário já pode chamar diretamente outro usuário sem a intervenção do servidor. O usuário chamador envia uma mensagem para o usuário destino informando que um cliente o está chamando, este usuário poderá ou não aceitar a chamada. Caso a chamada seja aceita, o usuário chamador recebe uma mensagem de confirmação e a partir daí começa a troca de mensagens, caso a chamada não seja aceita, o usuário chamador recebe uma mensagem informando que sua chamada não foi aceita.

A partir do momento em que se inicia a comunicação entre dois clientes, estes enviam ao programa servidor uma mensagem informando que eles estão em uma chamada, assim, quando um outro cliente tenta chamar um usuário que já esteja em uma chamada, o programa servidor identifica que o usuário está ocupado e envia uma mensagem para o cliente que o está tentando chamar informando que o mesmo está ocupado no momento.

Por fim, quando um cliente deseja se desconectar, uma mensagem é enviada para o servidor informando a desconexão, o servidor por sua vez, retira este usuário da lista de conectados.

CAPÍTULO 5 – TESTES E ANÁLISE DE RESULTADOS

5.1. Ambiente experimental

Para a implementação do protótipo do sistema foi utilizada a linguagem de programação C sobre a plataforma Windows.

Os testes foram realizados em um laboratório com trinta e cinco computadores, sendo que para os testes apenas foram utilizados dois, os outros computadores ficaram inutilizados para dar mais precisão aos testes.

Todos os computadores do laboratório possuem processador Pentium IV 2.8GHz, memória RAM de 512MB PC2700 333MHz, Hard Disk de 40GB ATA1000 e placa de rede Intel PRO 1000. Os computadores estão interligados através de um Switch de 1Gb e por cabos furukawa categoria 5e. A conexão local estava configurada em 100Mbps. O sistema operacional que opera nas máquinas do laboratório é o Windows XP.

5.2. Descrição do teste e resultados

O teste consiste em marcar o tempo em que dados de tamanhos diferentes demoram para serem enviados e recebidos por um computador. Para a realização dos testes foram desenvolvidos dois programas para o envio e recebimento de dados.

Um dos programas tem a função de enviar dados de diferentes tamanhos para outro computador e iniciar uma contagem de tempo, em milisegundos, através da função *clock()* da linguagem C. O computador que recebe os dados, executa o outro programa. Este programa tem a função de enviar os dados que recebe para o computador que o enviou. Por sua vez, o

programa que iniciou a contagem do tempo, recebe os dados e pára a contagem através de outra chamada a função *clock()*.

Obtidos os tempos de início e fim, fez-se a diferença entre eles para obter o tempo que o dado levou, para chegar à outra máquina e voltar, em milisegundos.

Os resultados conseguidos a partir da realização deste teste são mostrados na Figura 20 que apresenta um gráfico com os resultados.

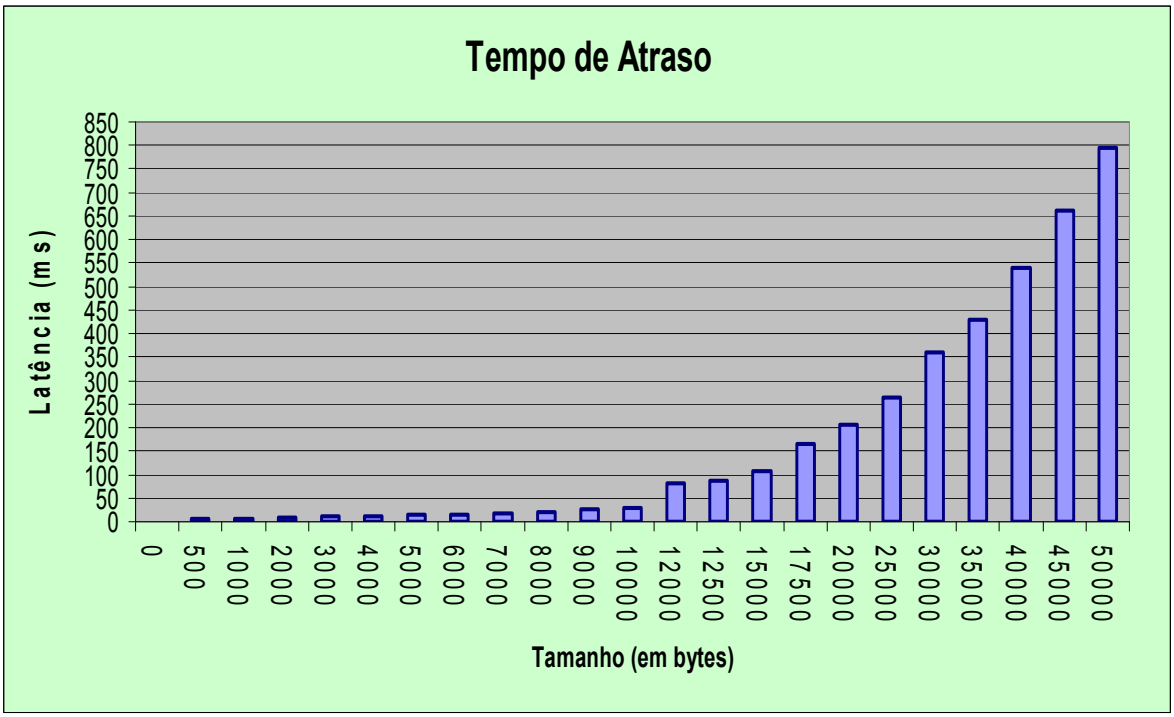

Figura 20 – Tempo de atraso.

Observando os resultados mostrados pelo gráfico obtido através do teste, tem-se a nítida impressão que à medida que o tamanho dos dados vai aumentando, aumenta também a latência. Porém se for analisar o tamanho de um determinado dado com sua respectiva latência, pode-se reparar que o atraso é justificado pelo tamanho do dado, pois se trata de tempo em milisegundos.

O tamanho máximo de um pacote *ethernet* é 1500 bytes, portanto sugere-se que o tamanho do buffer que conterá a mensagem deve ser este, além disso, o atraso para que este pacote chegue ao destino é mínimo.

CAPÍTULO 6 – CONCLUSÃO E APERFEIÇOAMENTOS FUTUROS

6.1. Conclusão

Através do estudo da tecnologia Voz sobre IP e das aplicações que já existem, pôde-se perceber que esta tecnologia será muito utilizada no futuro próximo, embora a complexidade que revela a implementação de um projeto deste tipo.

Observando as técnicas e protocolos utilizados para aplicações Voz sobre IP, tem-se a certeza de que se pode desenvolver aplicações VoIP de diversas maneiras, utilizando diversos protocolos e bibliotecas.

Para este trabalho, optou-se por não utilizar nenhum dos protocolos que geralmente são utilizados como SIP, H.323 e RTP, justamente em busca de um melhor desempenho. Decidiu-se utilizar o protocolo UDP, pois além de ser um protocolo não confiável e não orientado a conexão, também é base do protocolo RTP. Em aplicações de tempo real, dados atrasados geralmente são muito piores do que dados perdidos.

Analisando a implementação do projeto, pode-se concluir que os serviços de socket juntamente com o protocolo UDP mostram-se recursos poderosos para desenvolver aplicações VoIP. Com estes recursos pode-se também realizar o controle de conexões, como foi feito no programa servidor e cliente, e realizar a troca de mensagens sem grandes atrasos.

6.2. Aperfeiçoamentos Futuros

Nas próximas seções são sugeridas algumas implementações e aperfeiçoamentos para tornar a aplicação mais eficiente. Algumas destas sugestões não foram implementadas devido a dificuldades encontradas ao longo do desenvolvimento do trabalho.

6.2.1. Desenvolvimento de captura e reprodução da voz

Implementação da captura e reprodução de voz, através da utilização bibliotecas que disponibilizam uma série de funções que facilitam a captura e a reprodução da voz, como DirectX da Microsoft.

6.2.2. Estudo e implementação de threads

Estudar e fazer a utilização de threads para que possa ser feita uma sincronização de captura e reprodução de voz em "paralelo", para que não haja problemas como por exemplo a paralisação da captura ou reprodução.

6.2.3. Implementação de conferência de voz

Depois de feita a implementação de threads, é possível o desenvolvimento de conferência de voz no projeto utilizando o conceito de socket multicast, descrito na seção 2.9 que descreve o funcionamento do endereçamento multicast.

6.2.4. Implementação de CODECs.

Estudar e adicionar ao projeto CODECs para fazer a compactação e descompactação, visando a diminuição do tamanho dos pacotes e conseqüentemente uma diminuição no tempo de envio dos pacotes de dados e redução de trafego na rede.

6.2.5. Desenvolvimento de interface

Fazer o desenvolvimento de uma interface amigável para o usuário a fim de que ele possa manusear o software com maior facilidade e interação.

6.2.6. Implementação de um projeto utilizando outras bibliotecas

Desenvolvimento de um outro projeto, que diferentemente deste, faça a utilização de bibliotecas próprias para aplicações VoIP como a JVoIPLib e JRTPLib. É realmente muito interessante fazer a implementação de uma aplicação utilizando diferentes recursos para que possa haver uma comparação de qualidade e eficiência entre as duas.

REFERÊNCIAS BIBLIOGRÁFICAS

CISCO, Systems Brasil. O que é VoIP (Voz sobre IP) – Redação Virtual. Disponível em: http://www.ciscoredacaovirtual.com/redacao/perfistecnologicos/conectividad.asp?Id=20. Acesso em: 07 outrobro 2005.

COMER, Douglas. E. Interligação em Redes com TCP/IP: Princípios Protocolos e Arquitetura. Tradução ARX Publicações. Vol. 2. Editora Campus, 1998.

DEERING, Steve. RFC 1112: Network Working Group. Stanford University. Stanford, 1989.

GOMIDE, Cairo. Multiplexação em Sinais de Voz em redes IP. Grau : Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

KUROSE, James F. Redes de Computadores e a Internet: Uma Abordagem top-down. Tradução Arlete Simille Marques. 3. ed. Editora Addison Wesley, 2005.

MICROSOFT, Corporation. Windows Sockets 2. Disponível em http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/ windows_ sockets_start_page_2.asp> Acesso em: 01 agosto 2005.

NENO, Mylène. Saiba mais sobre VoIP. Disponível em: http://www.clubedohardware.com.br/artigos/99. Acesso em: 10 novembro 2005.

PROJETO VoIP. Disponível em: www.usp.br/cci/downloads/VoIP-CTI-20-05-2005.ppt. Acesso em: 15 novembro 2005.

SOARES, Luiz F. G.; LEMOS, Guido; COLCHER, Sergio. Redes de Computadores: Das LANs MANs e WANs às Redes ATM. 2. ed. Editora Campus, 2004.

TANENBAUM, Andrew. S. Redes de Computadores. Tradução Vanderberg D. de Sousa. 4. ed. Editora Campus. 2003.

WIKIPÉDIA. A enciclopédia livre: Voz Sobre IP. Disponível em: http://pt.wikipedia.org/wiki/Voz_sobre_IP. Acesso em: 17 novembro 2005.

APÊNDICE A – Implementação do arquivo de cabeçalho h_voip.h.

```
#include <stdio.h>
//tabela de conectados
struct reg{
	int flag;		//flag que indica se está(1) ou não conectado(0)
	struct in_addr ip;	//ip da máquina conectada
	short porta;		//porta da máquina conectada
	char id_nome[30];	//identificador da máquina, definida pelo usuário
	int id_num;		//identificador da máquina, criada pelo servidor
};

#define MAX_ELEM 30

struct tabela{
	int n_elem;		//nº de tabelas da lista
	reg t[MAX_ELEM];	//lista de registros do tipo reg
};

struct dados{
	char msg[30];		//mensagem a ser enviada
	unsigned int tipo;	//tipo da mensagem enviada ao servidor
	int n_seq;		//número de seqüência da mensagem
};

void inicia_lista_tab (struct tabela *tab){
	int i;
	for(i=0;i<MAX_ELEM;i++)
		tab->t[i].flag = 0;
	tab->n_elem = 0;
}
int conectar (struct tabela *tab, struct reg aux){
	int i=0;
```

```
	if(tab->n_elem == MAX_ELEM-1)
		return -1;
	while(tab->t[i].flag == 1){
		i++;
	}
	tab->t[i] = aux;
	tab->t[i].flag = 1;
	tab->t[i].id_num = i;
	tab->n_elem++;

	return i + 10;
}

void desconectar (struct tabela *tab, int aux){
	tab->t[aux - 10].flag = 0;
	tab->n_elem--;
}

int devolve_conectados(struct tabela tab,char conectados[]){
	if(tab.n_elem == 0)
		return -1;
	strcpy(conectados,"\0");
	for(int i=0;i<MAX_ELEM-1;i++){
		if(tab.t[i].flag == 1){
			strcat(conectados,tab.t[i].id_nome);
			strcat(conectados,"\n");
		}
	}
	return 0;
}

int devolve_ip (struct tabela tab,char nome[]){
	for(int i=0;i<MAX_ELEM-1;i++){
		if((strcmp(nome,tab.t[i].id_nome) == 0) &&
```

```
        ((tab.t[i].flag == 1)||(tab.t[i].flag == 2))){
                    break;
                }
        }
        if((i == MAX_ELEM-1) && (strcmp(nome,tab.t[i].id_nome) != 0))
                return -1;
        else{
                if(tab.t[i].flag == 2){
                    return -2;
                }
        }
        return i;
}

int ocupado (struct tabela *tab,char nome[]){
        for(int i=0;i<MAX_ELEM-1;i++){
                if((strcmp(nome,tab->t[i].id_nome) == 0) &&
        ((tab->t[i].flag == 1)||(tab->t[i].flag == 2))){
                    break;
                }
        }
        if((i == MAX_ELEM-1) && (strcmp(nome,tab->t[i].id_nome) != 0))
                return -1;

        tab->t[i].flag = 2;
        return 0;
}
int desocupado (struct tabela *tab, char nome[]){
        for(int i=0;i<MAX_ELEM-1;i++){
                if((strcmp(nome,tab->t[i].id_nome) == 0) &&
        ((tab->t[i].flag == 1)||(tab->t[i].flag == 2))){
                    break;
                }
        }
```

```
        if((i == MAX_ELEM-1) && (strcmp(nome,tab->t[i].id_nome) != 0))
                return -1;

        tab->t[i].flag = 1;
        return 0;
}
```

APÊNDICE B – Implementação do programa servidor.

```
#include <iostream.h>
#include <string.h>
#include <winsock2.h>
#include <h_voip.h>
#include <stdio.h>


void main(){

	WORD versao;
	WSADATA data;
	versao = MAKEWORD(2,0);
	int err;
	err = WSAStartup(versao,&data); //definição da versão do socket usado,
                                        // no caso V2.0.
	if(err!=0)
		printf("ERRO DE VERSÃO\n");

	struct tabela tab;
	struct reg reg_aux;
	struct dados dado,resposta;

	SOCKET sock;
	struct sockaddr_in serv,cliente;
	serv.sin_family = AF_INET;
	serv.sin_port = htons(8888);
	serv.sin_addr.s_addr = INADDR_ANY;
	for(int i=0;i<8;i++){
		serv.sin_zero[i] = (char)0;
	}

	sock = socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
	if(sock == INVALID_SOCKET)
		printf("ERRO NA CRIACAO DO SOCKET\n");

	int errobind;
	errobind = bind(sock,(LPSOCKADDR)&serv,sizeof(serv));
	if(errobind == INVALID_SOCKET)
		printf("ERRO COM O BIND()\n");

	int num_bytes,addr_tam;
	addr_tam = sizeof(struct sockaddr);
	inicia_lista_tab(&tab);
	printf("SERVIDOR INICIALIZADO\n");

	do{
```

```
num_bytes = recvfrom(sock,(char *)&dado,sizeof(dado),0,
                        (LPSOCKADDR)&cliente,&addr_tam);
if(num_bytes == INVALID_SOCKET)
        printf("ERRO COM RECVFROM()\n");

switch(dado.tipo){
        case 2:

                strcpy(reg_aux.id_nome,dado.msg);
                reg_aux.ip.s_addr = cliente.sin_addr.s_addr;
                reg_aux.porta = htons(cliente.sin_port);
                resposta.tipo = conectar(&tab,reg_aux);
                sendto(sock,(char *)&resposta,sizeof(resposta),0,
                        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));
                printf("%s conectado! \n",dado.msg);
        break;
        case 3:

                reg_aux.flag = devolve_ip(tab,dado.msg);
                if((reg_aux.flag == -1)||(reg_aux.flag == -2)){
                        sendto(sock,(char *)&reg_aux,sizeof(reg_aux),0,
                        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));
                }
                else{

                        reg_aux = tab.t[reg_aux.flag];
                        sendto(sock,(char *)&reg_aux,sizeof(reg_aux),0,
                        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));

                }

        break;
        case 4:

                resposta.tipo = devolve_conectados(tab,resposta.msg);
                if(resposta.tipo == -1){
                        strcpy(resposta.msg,"Não ha nenhum conectado\n");
                        sendto(sock,(char *)&resposta,sizeof(resposta),0,
                        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));
                }
                else{
                sendto(sock,(char *)&resposta,sizeof(resposta),0,
                        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));
                }
break;
case 5:

        reg_aux.flag = ocupado(&tab,dado.msg);
        if(reg_aux.flag == 0)
                printf("%s esta conectado com um usuario!\n",dado.msg);
```

```
                    else
                        printf("Erro ao marcar como ocupado!\n");

                break;
                case 6:

                    reg_aux.flag = desocupado(&tab,dado.msg);
                    if(reg_aux.flag == 0)
                        printf("%s se desconectou do outro usuario!\n",dado.msg);
                    else
                        printf("Erro ao marcar como desocupado!\n");

                break;
                default:

                    desconectar(&tab,dado.tipo);
                    printf("%s se desconectou\n",tab.t[dado.tipo - 10].id_nome);
        }

}while(TRUE);

printf("TERMINANDO SERVIDOR...\n");
closesocket(sock);
system("PAUSE");

}
```

APÊNDICE C – Implementação do programa cliente.

```
#include <iostream.h>
#include <string.h>
#include <winsock2.h>
#include <h_voip.h>
#include <stdio.h>

void main(){
	WORD versao;
	WSADATA data;
	versao = MAKEWORD(2,0);
	int err;
	err = WSAStartup(versao,&data); //definição da versão do socket usado,
					//no caso V2.0.
	if(err!=0)
		printf("ERRO DE VERSÃO\n");

	SOCKET sock;
	struct reg reg_aux;
	struct dados dado,dado_s;
	struct sockaddr_in eu,cliente,serv;
	struct tabela tab;

	eu.sin_family = AF_INET;
	serv.sin_family = AF_INET;
	serv.sin_addr.s_addr = inet_addr("201.0.52.167");//local onde deve ser especificado
							//o endereço do servidor
	eu.sin_addr.s_addr = INADDR_ANY;
	serv.sin_port = htons(8888);
	eu.sin_port = htons(8888);
	for(int i=0;i<8;i++){
		eu.sin_zero[i] = (char)0;
		serv.sin_zero[i] = (char)0;
	}

	inicia_lista_tab(&tab);

	sock = socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
	if(sock == INVALID_SOCKET)
		printf("ERRO NA CRIACAO DO SOCKET\n");

	int errobind;
	errobind = bind(sock,(LPSOCKADDR)&eu,sizeof(eu));
	if(errobind == INVALID_SOCKET)
		printf("ERRO COM O BIND()\n");

	char ID[30],ID_aux[30];
```

```
printf("Entre com o seu Nome ou Apelido\n");
gets(ID);
strcpy(dado.msg,ID);
dado.tipo = 2;
sendto(sock,(char *)&dado,sizeof(dado),0,
        (LPSOCKADDR)&serv,sizeof(struct sockaddr));

int num_bytes,addr_tam,ID_num;
addr_tam = sizeof(struct sockaddr);
num_bytes = recvfrom(sock,(char *)&dado,sizeof(dado),0,
        (LPSOCKADDR)&serv,&addr_tam);
system("pause");
if(num_bytes == INVALID_SOCKET){
        printf("ERRO COM RECVFROM()\n");
        exit(0);
}
else{
if(dado.tipo == -1){
printf("Não foi possivel conectar ao servidor/nTente novamente mais tarde\n");
        exit(0);
}
else{
        ID_num = dado.tipo;
}
}
int opcao;
do{
        system("cls");
        printf("Escolha uma opção e tecle ENTER\n");
        printf("1 - Ver conectados\n");
        printf("2 - Chamar uma pessoa\n");
        printf("3 - Esperar por chamada\n");
        printf("4 - Desconectar\n");
        scanf("%d",&opcao);
        switch(opcao){
                case 1:

                        system("cls");
                        printf("1 - Ver conectados.\n\n\n");
                        dado.tipo = 4;
                        sendto(sock,(char *)&dado,sizeof(dado),0,
                                (LPSOCKADDR)&serv,sizeof(struct sockaddr));
                        num_bytes = recvfrom(sock,(char *)&dado,sizeof(dado),0,
                                (LPSOCKADDR)&serv,&addr_tam);
                        if(num_bytes == INVALID_SOCKET){
                                printf("ERRO COM RECVFROM()\n");
                        }
                        else{
                                printf("Usuarios conectados: \n");
                                printf("%s \n",dado.msg);
```

```
                }
                system("pause");

        break;
        case 2:

                system("cls");
                printf("2 - Chamar um usuario.\n\n\n");
                printf("Entre com o ID da pessoa: \n");
                scanf("%s",&ID_aux);
                strcpy(dado.msg,ID_aux);
                dado.tipo = 3;
                sendto(sock,(char *)&dado,sizeof(dado),0,
                        (LPSOCKADDR)&serv,sizeof(struct sockaddr));
                num_bytes = recvfrom(sock,(char *)&reg_aux,
                       sizeof(reg_aux),0, (LPSOCKADDR)&serv,&addr_tam);
                if(num_bytes == INVALID_SOCKET){
                        printf("ERRO COM RECVFROM()\n");
                }
                else{
                        if(reg_aux.flag == -1){
                                printf("ID não existente\n");
                        }
                        else{
                        if(reg_aux.flag == -2){
                                printf("O usuario %s ja esta em uma
                                chamada!\n",ID_aux);
                                printf("Tente nomavente mais tarde.\n");
                                }
                                else{
                                cliente.sin_family = AF_INET;
                                cliente.sin_addr.s_addr = reg_aux.ip.s_addr;
                                cliente.sin_port = htons(reg_aux.porta);
                                for(int i=0;i<8;i++){
                                        cliente.sin_zero[i] = (char)0;
                                }
                                dado_s.tipo = 0;
                                strcpy(dado_s.msg,ID);
                                sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
                        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));

num_bytes = recvfrom(sock,(char *)&dado,sizeof(dado),0,
        (LPSOCKADDR)&cliente,&addr_tam);
                if(num_bytes == INVALID_SOCKET){
                        printf("ERRO COM RECVFROM()\n");
                }
                else{
                if(dado.tipo == -1){
                printf("O usuario %s recusou sua chamada!\n",ID_aux);
                }
```

```
                else{
                        dado_s.tipo = 5;
                        strcpy(dado_s.msg,ID);
                        sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
                        (LPSOCKADDR)&serv,sizeof(struct sockaddr));
                        printf("Sua chamada foi aceita!\n");
                        printf("Entre com a mensagem:\n");
                        gets(dado_s.msg);
                        do{

                                printf("%s - ",&ID);
                                gets(dado_s.msg);
                                sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
                        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));

                                if((strcmp(dado_s.msg,"desconectar"))!=0){
                           num_bytes = recvfrom(sock,(char *)&dado,sizeof(dado),0,
                                        (LPSOCKADDR)&cliente,&addr_tam);
                             if(num_bytes == INVALID_SOCKET)
                                printf("ERRO COM RECVFROM()\n");
                             else
                                printf("%s - %s \n",ID_aux,dado.msg);
                        }

                        }while((strcmp(dado.msg,"desconectar")!=0)&&
                                (strcmp(dado_s.msg,"desconectar"))!=0);
                                        dado_s.tipo = 6;
                                        strcpy(dado_s.msg,ID);
                        sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
                                (LPSOCKADDR)&serv,sizeof(struct sockaddr));
                        }
                    }
                }
            }
        }
                system("pause");
                break;

                 case 3:

                        system("cls");
                        printf("3 - Esperar por chamada.\n\n\n");
                        printf("Esperando chamada....\n");
                        num_bytes = recvfrom(sock,(char *)&dado,sizeof(dado),0,
                                (LPSOCKADDR)&cliente,&addr_tam);
                        if(num_bytes == INVALID_SOCKET)
                                printf("ERRO COM RECVFROM()\n");
                        else{
                                if(dado.tipo == 0){
```

```
                printf("O usuario %s está tentando se conectar
                com voce.\n",dado.msg);
                printf("Aceitar chamada?\n");
            do{
                        printf(" 1 - Sim.\n");
                        printf("-1 - Nao.\n");
                        printf("Digite a opcao: ");
                        scanf("%d",&dado_s.tipo);
             }while((dado_s.tipo != 1) && (dado_s.tipo != -1));
if(dado_s.tipo == -1){
        sendto(sock,(char*)&dado_s,    sizeof(dado_s),0,
        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));
        continue;
}
else{
        dado_s.tipo = 5;
        strcpy(dado_s.msg,ID);
        sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
           (LPSOCKADDR)&serv,sizeof(struct sockaddr));
        char ID_dele[30];
        strcpy(ID_dele,dado.msg);
        sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));
        gets(dado_s.msg);
do{
num_bytes = recvfrom(sock,(char *)&dado,sizeof(dado),0,
        (LPSOCKADDR)&cliente,&addr_tam);
printf("%s - %s\n",ID_dele,dado.msg);
if((strcmp(dado.msg,"desconectar"))!=0){
        printf("%s - ",ID);
        gets(dado_s.msg);
        sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
        (LPSOCKADDR)&cliente,sizeof(struct sockaddr));
                                }


        }while((strcmp(dado.msg,"desconectar")!=0)&&
                (strcmp(dado_s.msg,"desconectar")!=0));
                dado_s.tipo = 6;
                strcpy(dado_s.msg,ID);
                sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
                        (LPSOCKADDR)&serv,
                                sizeof(struct sockaddr));

                }
        }
}
system("pause");
continue;
```

```
                        break;
                        case 4:

                                system("cls");
                                printf("4 - Desconectar.\n\n\n");
                                printf("Desconectando.....\n");
                                dado_s.tipo = ID_num;
                                sendto(sock,(char *)&dado_s,sizeof(dado_s),0,
                                        (LPSOCKADDR)&serv,sizeof(struct sockaddr));
                                closesocket(sock);
                                system("pause");

                        break;
                        default:
                                printf("Opção invalida! Tente novamente.\n");
                }
        }while(opcao != 4);
}
```