UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIAS E GEOCIÊNCIAS
DEPARTAMENTO DE ELETRÔNICA E SISTEMAS

Pós-Graduação em Engenharia Elétrica

Uma Aplicação de Voz Sobre IP baseada no Session Initiation Protocol

por

Jucimar Maia da Silva Junior

Dissertação de Mestrado

Recife
Agosto de 2003

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIAS E GEOCIÊNCIAS
DEPARTAMENTO DE ELETRÔNICA E SISTEMAS

JUCIMAR MAIA DA SILVA JUNIOR

Uma Aplicação de Voz Sobre IP baseada no Session Initiation Protocol

Este trabalho foi apresentado à Pós-Graduação em Engenharia Elétrica do Centro de Tecnologias e Geociências da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica.

ORIENTADOR: Prof. Rafael Dueire Lins

Agosto de 2003

Universidade Federal de Pernambuco

Pós-Graduação em Engenharia Elétrica

PARECER DA COMISSÃO EXAMINADORA DE DEFESA DE DISSERTAÇÃO DE MESTRADO DE

JUCIMAR MAIA DA SILVA JÚNIOR

TÍTULO

"Uma Aplicação de Voz Sobre IP baseada no Session Initiation Protocol"

A comissão examinadora composta pelos professores: RAFAEL DUEIRE LINS, DES/UFPE, VALDEMAR CARDOSO DA ROCHA JÚNIOR, DES/UFPE e RICARDO MASSA FERREIRA LIMA, DI/UFPE, sob a presidência do primeiro, consideram o candidato **JUCIMAR MAIA DA SILVA JÚNIOR** **APROVADO**.

Recife, 06 de agosto de 2003.

RAFAEL DUEIRE LINS

VALDEMAR CARDOSO DA ROCHA JÚNIOR

RICARDO MASSA FERREIRA LIMA

Dedico este trabalho a Deus

To follow the path:
Look to the master,
Follow the master,
Walk with the master,
Become a master
Anonymous

Agradecimentos

- Ao professor Rafael Dueire Lins, pela amizade, apoio, compreensão e por sempre me fazer voltar ao caminho correto.
- A Cláudia e meus filhos Kaio e Anna, pelo amor e paciência nos momentos que estive longe ou muito ocupado.
- Aos amigos Raimundo e Ricardo, pela amizade e nunca terem me deixado desistir.
- Aos meus pais, Jucimar e Selma, por sempre terem me incentivado a estudar.
- Ao Unilton, por ter me ajudado nas horas que precisei e estava longe de Recife.
- Ao Sr Fernando Folhadela, por ter conseguido minha liberação para a realização do mestrado em Recife.
- Ao Sr Miguel Grimm, por ter me ajudado a voltar a Recife e fazer a defesa.
- Aos amigos da FUCAPI que me ajudaram a terminar esta dissertação.
- À Universidade Federal de Pernambuco, nas pessoas dos professores e funcionários do Departamento de Eletrônica e Sistemas, por terem feito minha estadia em Recife ótima.

Resumo

Neste trabalho é implementada uma aplicação de Voz Sobre IP utilizando o *Session Initiation Protocol* (SIP) como protocolo de sinalização. Para alcançar este objetivo, são analisadas algumas tecnologias importantes, como os protocolos da Internet Engineering Task Force (IETF) para Telefonia IP, codificadores de voz, e a utilização da aplicação para verificação de atrasos, *jitter* e perda de pacotes. A aplicação foi desenvolvida em Java e pode facilmente ser modificada para adequar novos codificadores de voz e características mais recentes que novas especificações do SIP possam implementar.

Abstract

In this work is implemented an application of Voice On IP using Session Initiation Protocol (SIP) as signalling protocol. To reach this objective, some important technologies are analyzed, as the protocols of the Internet Engineering Task Force (IETF) for Telephony IP, coders of voice, and the use of the application for verification of delays, jitter and loss of packages. The application was developed in Java and can easily be modified to adjust new coders of voice and characteristics more recent than new specifications of the SIP can implement.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

Este capítulo descreve uma visão geral sobre o que é Voz Sobre IP, motivação e objetivos desta dissertação além de apresentar a idéia principal dos demais capítulos que constituem este trabalho.

1.1 Voz Sobre IP

Voz é transmitida via Rede Pública de Telefonia (*Public Switched Telephone Network* - PSTN) há mais de 100 anos. É um mercado que movimenta quase 100 bilhões de dólares anualmente. Todo esse custo, aliado ao crescimento exponencial do uso e infra-estrutura da Internet nos últimos 10 anos, levaram ao surgimento do interesse sobre a tecnologia de Voz Sobre IP (VoIP) ou como também é conhecida: *Telefonia IP*. VoIP é a entrega em tempo real de voz entre dois ou mais participantes através de uma rede que utiliza o Internet Protocol (IP). Juntamente com a voz, são enviadas informações necessárias para controlar essa troca, processo conhecido como **sinalização.** A voz é digitalizada, comprimida utilizando um *codec* (codificador-decodificador) específico. A voz codificada é dividida em pacotes e então enviada pela rede IP. Ao chegar no destino, os pacotes são ordendos, montados e a voz é decodificada (ver Figura 1.1).

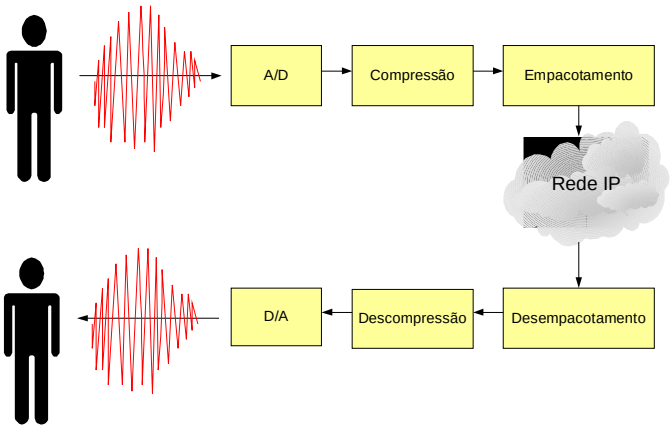

Figura 1.1 Voz Sobre IP

1.2 Por que usar VoIP?

Um usuário doméstico pode usar sua conexão Internet normal para realizar esta atividade ou *links* rápidos como *cable-modems* ou ISDN (*Integrated Services Digital Network*). Usuários corporativos podem usar toda infra-estrutura de rede de sua empresa (*intranet*) para trafegar voz. Além dessas facilidades, existem outros aspectos que levam a adoção de VoIP:

- **Custo menor de equipamento** - Para conversar PC-com-PC, basta que o usuário tenha um computador dotado de caixas de som e microfone, um software específico e uma conexão IP. A conexão IP precisa apenas de equipamentos comuns de rede (*switches*, *hubs*, roteadores). Esses equipamentos são produzidos em massa para suprir o crescimento do uso de PCs e da Internet. Toda essa demanda leva a uma concorrência intensa entre os fornecedores, causando queda de preços. Os equipamentos específicos para telefonia convencional são produzidos por um número restrito de grandes fabricantes (isso acontece porque a demanda não é tão grande quanto a de dispositivos de rede) encarecendo os preços e tornando o mercado proibitivo para empresas de pequeno porte. Normalmente esses dispositivos são desenvolvidos com tecnologia proprietária (tanto hardware quanto software), causando uma dependência do cliente com o fornecedor.
- **Queda nos custos de ligações** - Nos EUA uma chamada de longa distância custa em média $0,0171 o minuto. Com esse valor a Bell e suas subsidiárias faturam em média 8 bilhões de dólares por ano. As corporações e usuários domésticos pagam muito caro pelas ligações. Uma empresa que possua uma infra-estrutura de rede IP montada e ligando os pontos para onde essas ligações são realizadas pode economizar bastante sem

perder muito em Qualidade de Serviço (QoS). Em princípio essas ligações podem ser feitas de graça, pois o custo do link já esta embutido na infra-estrutura de rede.

- **Integração entre voz e dados** - Com a telefonia convencional, é necessário uma linha para acesso a Internet e outra para tráfego de voz. Por exemplo: um corretor de imóveis está fechando um negócio com um cliente por telefone convencional. No meio da negociação o futuro comprador diz que gostaria de ver o desenho da planta baixa da casa. O corretor tem esse arquivo no computador. Ele imprime e tenta enviar por fax, mas a qualidade da imagem fica muito ruim. Ele então é obrigado a finalizar a ligação, conectar-se a Internet via telefone e então enviar por e-mail a imagem. Com a Telefonia IP não existe esse problema, pois voz e dados compartilham a mesma rede ao mesmo tempo. O corretor poderia ao mesmo tempo em que conversava com o cliente enviar o e-mail com a imagem. VoIP permite muito mais que isso. Permitiria que fosse montada uma aplicação onde o corretor mostrasse em tempo real a planta enquanto falava as vantagens da casa. Integração total de voz e dados. Permite ainda uma grande variedade de novos serviços: *e-commerce* otimizado, videoconferência, ensino a distância entre muitos outros. Enquanto as redes de telecomunicações convencionais (telefone, fax, rádio e tv) são distintas em termos de tecnologia, interface com o usuário e dispositivos, o mesmo não acontece com a Internet.
- **Menor necessidade de largura de banda** - Quando uma ligação de telefonia convencional é completada, fecha-se uma conexão fim-a-fim exclusiva entre o transmissor e o receptor por onde a voz será transportada a uma taxa de 64Kbps. Toda banda fica exclusiva para essa conexão. Com Telefonia IP outros mecanismos mais sofisticados de compressão de voz podem ser utilizados. Isso permite transmissões de voz a 32Kbps, 16Kbps, 6.3Kbps ou 5.3Kbps. A mesma banda pode ser usada para trafegar diversas ligações simultaneamente sem que uma atrapalhe a outra.
- **Larga utilização do protocolo IP** - A Internet sofreu difusão exponencial nos últimos anos atingindo os mais diversos recantos do planeta. Dessa maneira, hoje em dia, em qualquer empresa pode ser encontrada uma rede IP (tanto LAN quanto WAN). A tendência mundial é colocar endereços IP em tudo: sistemas desktop, palmtop, laptop e etc. A infra-estrutura montada para acesso a Internet pode ser utilizada para Telefonia IP, pois esta é considerada apenas mais um tipo de serviço, como WEB ou E-mail.

- **Lucros** - VoIP está atraindo muita atenção das empresas de telefonia e Internet principalmente por causa dos lucros. O ramo de telefonia é um dos mais lucrativos do planeta. Abaixo estão alguns dados colhidos em 1999 [COL01]:

Empresas Internet	**Empresas de Telefonia**
YAHOO! – Lucro líquido de 61 milhões de dólares sobre rendimentos de 588 milhões de dólares	BellSouth – Lucro líquido de 3.5 bilhões de dólares de rendimentos de 25 bilhões de dólares (a maioria vindo de telefonia doméstica)
AMAZON.COM – Rendimentos de 1.6 bilhões de dólares. Ainda não deu lucro	Bell Atlantic – Lucro líquido de 4.7 bilhões de dólares de rendimentos de 33 bilhões de dólares

Tabela 1-1 Lucros

1.3 Por que não usar VoIP?

Apesar de todas as vantagens citadas, existem desafios reais que deverão ser vencidos pela Telefonia IP para que esta se torne comercialmente aceita:

- **Falta de padronização de protocolos** - VoIP difere dos serviços convencionais de entrega de multimídia em tempo real principalmente no estabelecimento e controle de sessões, a **sinalização**. Existem duas principais maneiras de implementar sinalização para VoIP: SIP e H.323 [COL01,DAL98,SCH98a]. H.323 é um padrão desenvolvido pela ITU-T que especifica um conjunto de protocolos, componentes e procedimentos para transmissão em tempo real de vídeo, áudio e dados em uma rede comutada por pacotes. Ele é o protocolo mais usado hoje em dia para VoIP, havendo diversos produtos comerciais baseados nele (Ex. *Microsoft NetMeeting*). Desenvolvido pela IETF, *Session Initiation Protocol* (SIP) é um protocolo de controle da camada de aplicação que pode estabelecer, modificar e terminar sessões multimídia ou chamadas. Essas sessões multimídia incluem conferências multimídia, ensino a distância, telefonia Internet e aplicações similares [HAN99].
- **Escalabilidade e confiabilidade da rede** - A rede de telefonia hoje é 99,99% disponível. Redes IP não são assim. São muito comuns as expressões: *".. a rede caiu ...*", *"...A Internet está fora ..*". Enquanto as redes IP não se tornarem disponíveis no mesmo nível da PSTN não poderão substituí-las.
- **Qualidade de Voz** - O principal desafio de VoIP é manter uma qualidade de voz nas ligações semelhante ao da telefonia convencional.

1.4 Fatores que afetam a Qualidade de Voz

Para chegar a um nível de qualidade de voz aceitável existem 4 pontos principais devem ser abordados:

- **Atraso -** É o tempo que um pacote leva para chegar ao destinatário. Usuários acham inaceitáveis atrasos superiores à 200ms [VAR02].
- ***Jitter*** **-** A variação nos atrasos (*jitter*) é causada quando os pacotes chegam fora de ordem. Eles então necessitam ser bufferizados, para depois serem reordenados. Isso acontece quando os pacotes atravessam switches e roteadores diferentes para chegar ao mesmo destino.
- **Perda de pacotes -** Apesar de comunicação por voz aceitar uma perda de até 5% dos pacotes, números maiores que esse afetarão a qualidade de voz, deixando a conversa entrecortada.
- **Codificadores de Voz –** A taxa de bits dos *codecs* de banda estreita disponíveis atualmente variam de 1,2Kbps a 64Kbps. Isso afeta a qualidade da voz reconstruída. Para medir a qualidade foi definido o *Mean Opinion Score* (MOS). Ele possui a seguinte escala:

Qualidade	MOS	Descrição
Alta	4 a 5	Similar, ou melhor, que a experiência do uso de uma chamada ISDN.
Qualidade Telefônica	3,5 a 4	Similiar a obtida com o uso do codificador G.726
Boa	3,0 a 3,5	Boa, mas com degradação facilmente audível.
Qualidade Militar	2,5 a 3	Comunicação ainda possível mas exige atenção

Tabela 1-2 Mean Opinion Score

A telefonia convencional utiliza o codificador G.711 que possui MOS de 4.3. Em VoIP, para minimizar o atraso e melhorar a utilização da banda, *codecs* modernos que variam de 3.7 a 4.0 podem ser utilizados.

1.5 Motivação

Há a necessidade da criação de um protótipo VoIP para testar a viabilidade de uma aplicação desse tipo na rede local e com isso minimizar custos de ligações. Existem poucas

implementações VoIP que utilizam o protocolo SIP, principalmente no meio acadêmico. Apesar de ser um protocolo novo, SIP já se estabeleceu como a "onda do futuro" para sinalização de VoIP [MAC00]. É muito mais simples de implementar que o H.323 e possui grande flexibilidade e poder para construir aplicações avançadas. É um protocolo desenvolvido para a Era Internet.

Para implantar uma aplicação baseada no SIP em uma rede privada é necessário encontrar um estudo sobre as possíveis causas de atrasos e maneiras de resolvê-lo. Não foi encontrado nenhum estudo sobre o assunto que envolvesse o protocolo SIP.

1.6 Objetivo

O objetivo deste trabalho é analisar e compreender os conceitos envolvidos na implementação de VoIP baseado nos protocolos desenvolvidos pela IETF. Utilizar esses conhecimentos para desenvolver uma aplicação Java para Telefonia IP baseada no *Session Initiation Protocol* que possibilite chamadas de voz (telefonia) via Internet e rede local. Através dessa aplicação, analisar a viabilidade do uso/desenvolvimento do protocolo SIP e as possíveis causas de atraso que possam ocorrer e propor parâmetros que possam melhorar o desempenho de aplicações desse tipo.

1.7 Organização da dissertação

Além deste capítulo introdutório, esta dissertação é composta dos seguintes capítulos:

Capitulo 2 – Digitalização de Voz e Protocolos
Revisão sobre digitalização de voz e dos protocolos TCP/IP que mais influenciam em uma aplicação VoIP.

Capítulo 3 – Aplicação de Telefonia IP.
Mostra o desenvolvimento da aplicação de Telefonia IP em Java. Esta aplicação será um USER AGENT SIP [SCH00].

Capítulo 4 - Análise de Desempenho

A análise consistirá em uma série de experimentos de transporte de voz sobre LAN e WAN. Estes experimentos servem para avaliar a influência de aspectos que alteram a interatividade e qualidade de serviço VoIP como sistema operacional, *jitter*, atraso e perda de pacotes.

Capítulo 5 - Conclusão

Uma análise crítica será feita em cima do desempenho esperado/alcançado. Possíveis trabalhos futuros serão propostos.

Capitulo 6 - Referências bibliográficas

Serão apresentadas as referências usadas na dissertação.

Apêndice

Código-fonte da aplicação

2 Digitalização de Voz e Protocolos

Este capítulo descreve uma visão geral sobre digitalização de voz e dos protocolos utilizados para desenvolver uma aplicação de Telefonia IP.

2.1 Digitalização de Voz

O som é vibração do ar que nossos ouvidos percebem, convertido em impulsos nervosos que são enviados ao cérebro. Sons simples tendem a ser periódicos enquanto os complexos tendem a não o ser [LIN00]. O mais simples de todos os sons é uma onda senoidal. É o deslocamento repetitivo do ar representado por uma única freqüência. A total amplitude de uma onda senoidal é a faixa que vai do pico mais negativo ao pico mais positivo da onda. A *freqüência* de uma onda senoidal expressa quão freqüentemente a onda repete um valor. O inverso da freqüência é chamado *período*. A freqüência de uma onda senoidal é tipicamente medida em ciclos por segundo. Um ciclo por segundo corresponde a um Hz, em homenagem a *Heinrich Hertz* (1857 a 1894).

Os seres humanos podem perceber sons de 30Hz a 20KHz (dependendo da idade, da saúde e de fatores externos como exposição excessiva a sons altos) [SPA]. O ouvido responde a sons de uma grande variedade de amplitudes. Desde o som do bater de asas de uma borboleta ao som de um avião a jato. Além disso, a resposta do ouvido não é linear na percepção de um barulho. Um som pode ser duas vezes mais alto o que não significa que ele tem duas vezes mais amplitude. O ouvido responde de uma maneira mais logarítmica que linear [WOO, LIN00]. A noção de faixas logarítmicas de amplitude para produzir um aumento de som que possa ser medido em decibelis. Um decibel é definido como:

1 decibel = 20* log10 (taxa de amplitude)

ou

1 decibel = 10*log10(taxa de energia) em watts

Essa diferença é porque a energia varia no quadrado da amplitude.

A noção de que um sinal de áudio, mais especificamente as notas musicais, possuiam uma relação especial *"de multiplicidade"* entre si em *"harmônicos"* era conhecida desde a antiguidade clássica pelos gregos. *Jean-Baptiste Fourier* (1768-1830) foi o primeiro a propor que um sinal periódico poderia ser representado por uma soma de série de ondas de senos e co-senos. *Claude Shannon* utilizou o farmalismo de Fourier para modelar aúdio, assumindo a sua repetição com período infinito, dando base matemática às telecomunicações. Isso mostra o conceito que um espectro de freqüências pode ser representado por um conjunto de ondas senoidais. Um som tem uma *freqüência fundamental* que é a freqüência da onda senoidal de maior amplitude, ou seja é a freqüência que "mais contribui" para a formação do sinal. No caso da voz humana, foco de interesse da telefonia, a fundamental está em torno de 1KHz.

2.1.1 Conversão analógico-digital

Um conversor analógico-digital (A/D) é um dispositivo que recebe como entrada um sinal contínuo (analógico) e produz como saída uma seqüência de valores discretos ou *amostras* através do processo de quantização. Um fluxo de amostras produzido pelo A/D representa a entrada da informação em um determinado grau de precisão. Um A/D converte um sinal analógico em uma série de amostras que representam este sinal no domínio digital. Amostras são sucessivas "fotografias" de um sinal (no caso uma onda de som). Um microfone converte um sinal acústico em um sinal analógico. Um conversor analógico-digital (A/D) converte o sinal analógico em digital. A precisão do sinal com o digital depende de sua resolução no tempo (taxa de amostragem) e de sua resolução na amplitude (quantidade de bits utilizados para representar cada amostra – quantização). A diferença entre o valor da amostra e a sua representação discreta gera um erro conhecido como *ruído de quantização*.

2.1.2 Lei de *Nyquist*

A taxa de amostragem: um sinal digital possui valores de amplitude que variam continuamente com o tempo. Para digitalizar este sinal, mede-se sua amplitude em intervalos regulares, processo que chamamos de amostragem. De acordo com *Henry Nyquist* (1889-1976) a taxa de amostragem determina a informação de freqüência mínima para que toda a informação de um

sinal limitado em banda a ser amostrado seja preservada. O critério de Nyquist estabelece que para poder regerar um sinal analógico a partir de suas amostras digitais, o sinal original deve ser amostrado a uma taxa que seja pelo menos o dobro da mais alta freqüência do sinal [ALE98, COL01]. Portanto, em telefonia, a voz é amostrada a uma taxa de 8000 amostras por segundo, pois se limitarmos a voz humana a uma banda de cerca de 4Khz temos não só um sinal inteligível, mas também a possibilidade de reconhecimento do interlocutor.

2.1.3 Tipos de Codificadores

Em Telefonia IP, os fluxos de áudio e vídeo são transportados pelo protocolo RTP (*Real-Time Transport Protocol*) usando técnicas de compressão de sinais digitais conhecidas como *codificadores de voz* ou *codecs* (*coder-decoder*). O objetivo da codificação de voz é representar a fala com um número mínimo de bits mantendo uma qualidade perceptível, transmissão e armazenamento eficiente. Existem 3 tipos de codificadores de voz [ALE98,WOO, COL01]:

- **Forma de onda** – procuram reproduzir o sinal, amostra por amostra explorando suas características estatísticas, temporais e espectrais. São codificadores de baixo atraso e pequena complexidade de implementação, mas necessitam de um uma alta taxa de transmissão acima de 16kbps. O codificador mais simples desse tipo é o *Pulse Code Modulation* (PCM).
- **Vocoders** - Vocoders ou codificadores de fonte operam usando o modelo de produção de voz. Dessa modelagem são retirados parâmetros para que a mesma seja implementada computacionalmente. Esses codificadores operam a uma taxa de transmissão média de 2,4Kbps. Infelizmente, a produção de voz humana é complexa, a implementação dessa modelagem produz sons inteligíveis, mas longe de parecerem naturais.
- **Híbridos -** Eles combinam a qualidade dos codificadores de forma de onda com a eficiência dos codificadores paramétricos. Eles são mais elaborados que os *vocoders* e necessitam de taxas de transmissão de 4,8 a 16Kbps.

2.1.4 Codificadores de voz usados em Voz Sobre IP

- **G.711** - É a maneira mais simples de se digitalizar os dados analógicos usando uma escala semilogaritmica. Isso é chamado de PCM Comprimido e serve para aumentar a resolução de sinais de baixa amplitude, enquanto sinais de amplitudes mais elevadas são tratados de maneira proporcional – operando parecido ao ouvido humano. Dois tipos

diferentes de escalas são usados na Europa (*A-law*) e nos EUA (*m-law*). O G.711 é usado em redes ISDN e na maioria dos *backbones* de telefonia digital. Sua taxa de amostragem é de 8000 amostras por segundo e é digitalizado em amostras de 8bits, necessitando então de uma taxa de transmissão de 64Kbps.

- **G.722** - Apesar de o G.711 ter uma qualidade muito boa, uma parte do espectro de voz (acima de 4KHz) é perdido. O G.722 [ITUa] codifica até 7KHz a apenas 48, 56 ou 64Kbps.
- **G.723.1** – O codificador G.723.1 [ITUd] utiliza um comprimento de quadro de 30ms e necessita de um esquema de previsão de 7,5ms. Ele possui dois modos de operação, um a 6,4 Kbps e um a 5,3 Kbps. O modo com a taxa de bits mais elevada usa o MP-MLQ (*Multipulse Maximum Likehood Quatization*) para modelar o sinal de voz enquanto o modo de menor taxa de bits usa o ACELP (*Algebric Code Excited Linear Prediction*)
- **G.726** - O codificador G.726 [ITUb] utiliza a técnica ADPCM (*Adaptative Diferencial Pulse Code Mudulation*) para codificar um fluxo de bits G.711 em palavras de dois, três ou quatro bits o que resulta em taxas de 16, 24 ou 32Kbps.
- **G.728 -** O codificador G.728 [ITUc] usa uma técnica de codificação LD-CELP (*Low-delay, Code-Excited Linear Prediction*). Possui pontuações MOS semelhantes ao G.726 a uma taxa de 16Kbps. CELP é um codificador otimizado para voz [HER02]. Esses codificadores modelam especificamente sons de voz e funcionam comparando a forma de onda a ser codificada com um conjunto de modelos de forma de onda (*codebook* de predição linear) e encontrando o que melhor coincide.
- **G.729** - O codificador G.729 [ITU] usa uma técnica de codificação chamada CS-ACELP (*Conjugated Structure Algebric Code-Excited Linear Prediction).* Ele produz quadros de 80bits codificando 10ms de fala a uma taxa de 8Kbits/s. Ele exige um esquema de previsão de 5ms.

Os protocolos têm MOS de acordo com a Tabela 2.1:

Codec	Taxa de bits	Tamanho do Frame	MOS
G.711 PCM	64	0.125	4.1
G.726 ADPCM	32	0.125	3.9
G.729 CS-ACELP	8	10	3.7
G.723.1 MP-MLQ	6,3	30	3.9

Tabela 2-1 Codificadores

2.2 Protocolos

A Internet oferece a oportunidade de desenvolver aplicações que possam eventualmente substituir a telefonia convencional [SCH98]. Diferente de rádio, televisão e telefonia convencional que devem usar tecnologias diferentes para implementar seus serviços, transportar multimídia pela Internet utiliza uma tecnologia comum não importando se a mídia transportada é som, imagem [SCH98b, POL99]. O que acontece é que existem aplicações diferentes para cada tipo de problema. Por exemplo, tocar um arquivo de música através de *streams* deve permitir ao usuário poder retroceder, avançar, pausar e aumentar o volume. Telefonia via Internet, deve permitir ao usuário "ligar" para outra pessoa, estabelecer a conexão, etc. Para dar suporte a essas aplicações foram desenvolvidos diversos protocolos representados na Figura 2.1 [VOV]. Os protocolos abordados neste capítulo são: *User Datagram Protocol* (UDP), *Real-Time Transport Protocol* (RTP), *Session Initiation Protocol* (SIP), *Session Description Protocol* (SDP).

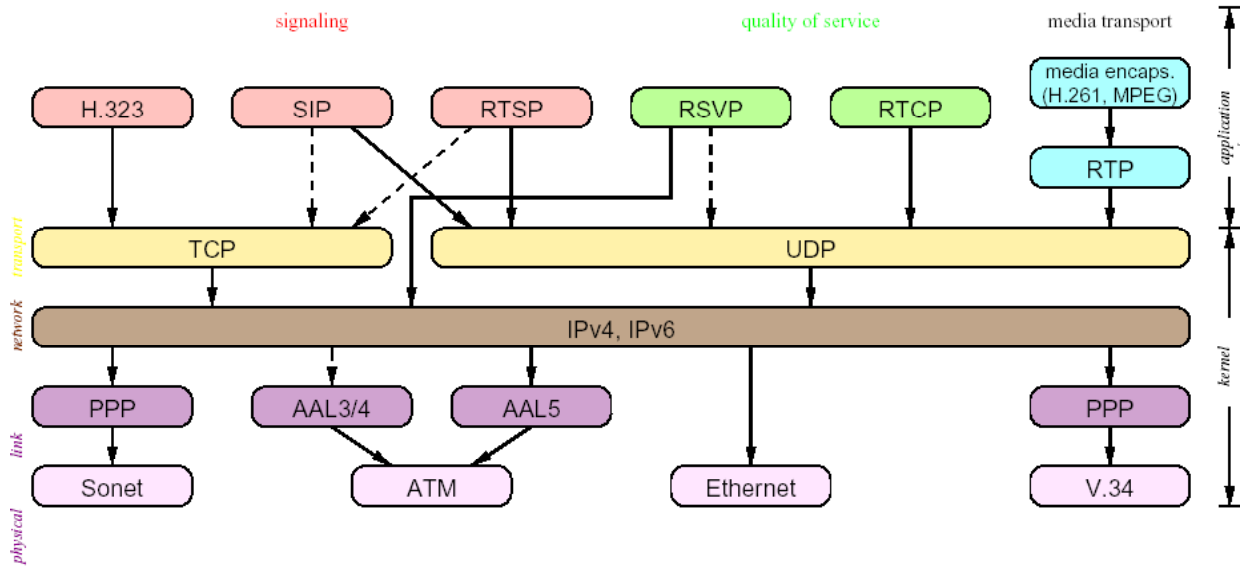

Figura 2.1 Protocolos de Telefonia IP

2.2.1 User Datagram Protocol

Uma transmissão confiável com conexão é como uma "chamada telefônica": Os softwares de rede dos sistemas operacionais de ambos os lados trocam informações para verificar se o receptor e o transmissor aceitam a "chamada". Verificam também se ambos estão prontos para ela. Quando todos os detalhes forem estabelecidos, a conexão é realizada e pode-se iniciar a transferência de dados. Durante a transferência, os protocolos de ambas as máquinas continuam

a comunicar-se para verificar se os dados são recebidos corretamente. Se houver alguma falha, ambas as máquinas detectarão e informarão aos aplicativos correspondentes. Definido na RFC 768, o *User Datagram Protocol* (UDP) fornece um serviço de transmissão **sem conexão, não confiável** [TAN97]. UDP usa o IP para transportar mensagens entre as máquinas, acrescentando a habilidade de distinguir entre múltiplos destinos em um determinado host através das portas de protocolo. Uma aplicação que usar UDP deve aceitar a responsabilidade de lidar com o problema de perda de mensagens, ordenação, duplicação e atraso, pois o protocolo não cuida desses aspectos. Segundo Comer [COM98]: "O UDP é um protocolo 'fino' porque, de modo significativo, nada acrescenta à semântica do IP".

2.2.1.1 Formato da mensagem UDP

A mensagem UDP é chamada de datagrama de usuário [COM98] e é encapsulada dentro de um único pacote IP de acordo com a Figura 2.2:

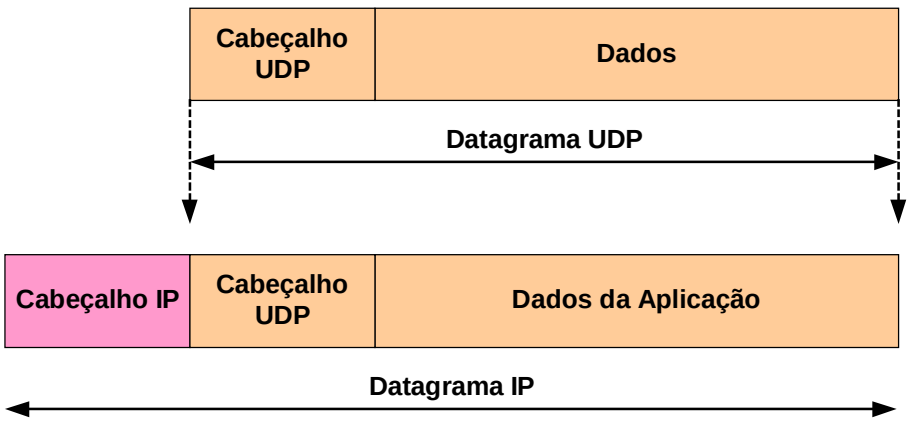

Figura 2.2 Encapsulamento UDP

O formato do datagrama UDP é mostrado na Figura 2.3:

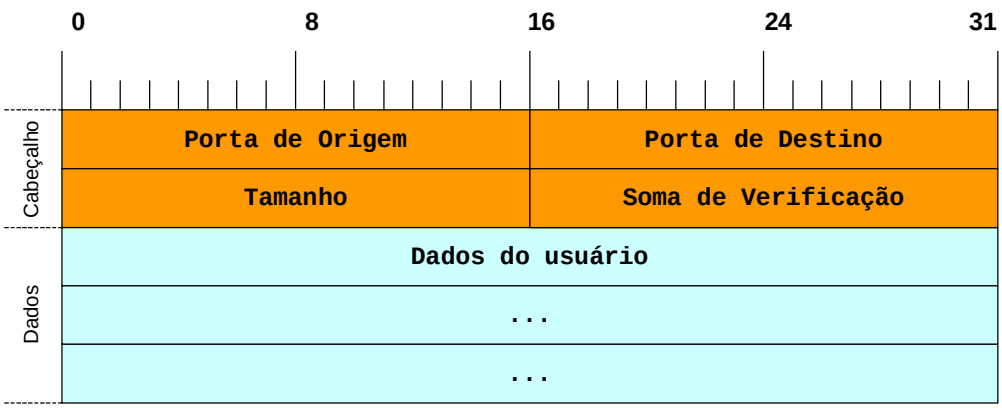

Figura 2.3 Datagrama UDP

Campo	Descrição
Porta de origem	Seu uso é opcional. Quando usada, especifica a porta no host de origem para qual as respostas devem ser enviadas.
Porta de destino	Indica a porta do host de destino para qual a mensagem foi enviada
Tamanho	Indica a contagem, em bytes, incluindo o cabeçalho e os dados do usuário.
Soma de verificação (checksum)	Seu uso é opcional. A soma de verificação UDP abrange mais informações do que consta no datagrama UDP. Para calcular o checksum, o protocolo adiciona um pseudocabeçalho ao datagrama UDP, acrescenta um octeto de zeros para preencher o datagrama e fazer dele um múltiplo exato de 16 bits. O octeto usado para o preenchimento e o pseudocabeçalho não são enviados e nem incluídos no cálculo do Tamanho do datagrama. No host de destino, o software deve extrair o endereço IP de origem e destino do cabeçalho IP, montá-los no formato do pseudocabeçalho e recalcular a soma de verificação

Tabela 2-2 Campos UDP

2.2.1.2 Pseudocabeçalho UDP

O uso do pseudocabeçalho tem como objetivo verificar se o datagrama UDP atingiu seu destino correto, baseado que o destino é formado pelo endereço IP acrescido da porta UDP. Seu formato é mostrado na Figura 2.4:

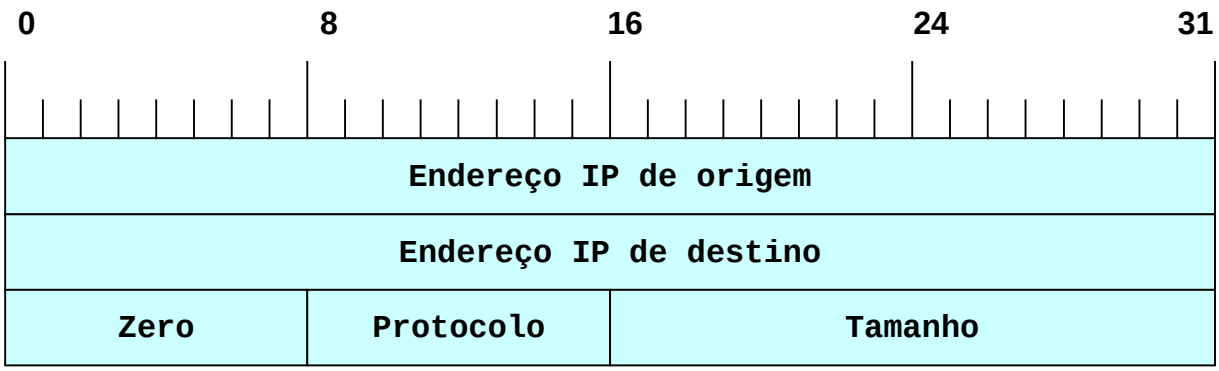

Figura 2.4 Pseudocabeçalho UDP

Campo	Descrição
Endereço IP de origem	Indica o endereço IP do host que gerou a mensagem
Endereço IP de Destino	Indica o endereço IP do host de destino para qual a mensagem foi enviada
Protocolo	Contém o código do protocolo usado. O código do UDP é 17
Tamanho	Indica o tamanho do datagrama UDP sem incluir o pseudocabeçalho

Tabela 2-3 Campos do pseucabeçalho UDP

Para calcular uma soma de verificação, o software primeiramente armazena zeros no campo Soma de Verificação e, a seguir, acumula uma soma de complementos de um de 16 bits de todo o objeto, inclusive o pseudocabeçalho, o cabeçalho UDP e dos dados do usuário.

2.2.1.3 Multiplexação e Demultiplexação do UDP

Cada aplicativo deve negociar com o sistema operacional para obter uma porta antes de enviar um pacote UDP. Ao obter uma porta, seu número constará nos pacotes UDP enviados pelo aplicativo, por essa porta, dentro do campo PORTA DE ORIGEM.

A Figura 2.5 mostra o esquema de multiplexação/demultiplexação. Ao transmitir mensagens, o UDP aceita datagramas de diversos aplicativos os passa à camada IP (multiplexação). Ao receber mensagens, o UDP aceita datagramas vindos da camada IP e os repassa aos aplicativos destinos através da PORTA DE DESTINO endereçadas em cada pacote (demultiplexação).

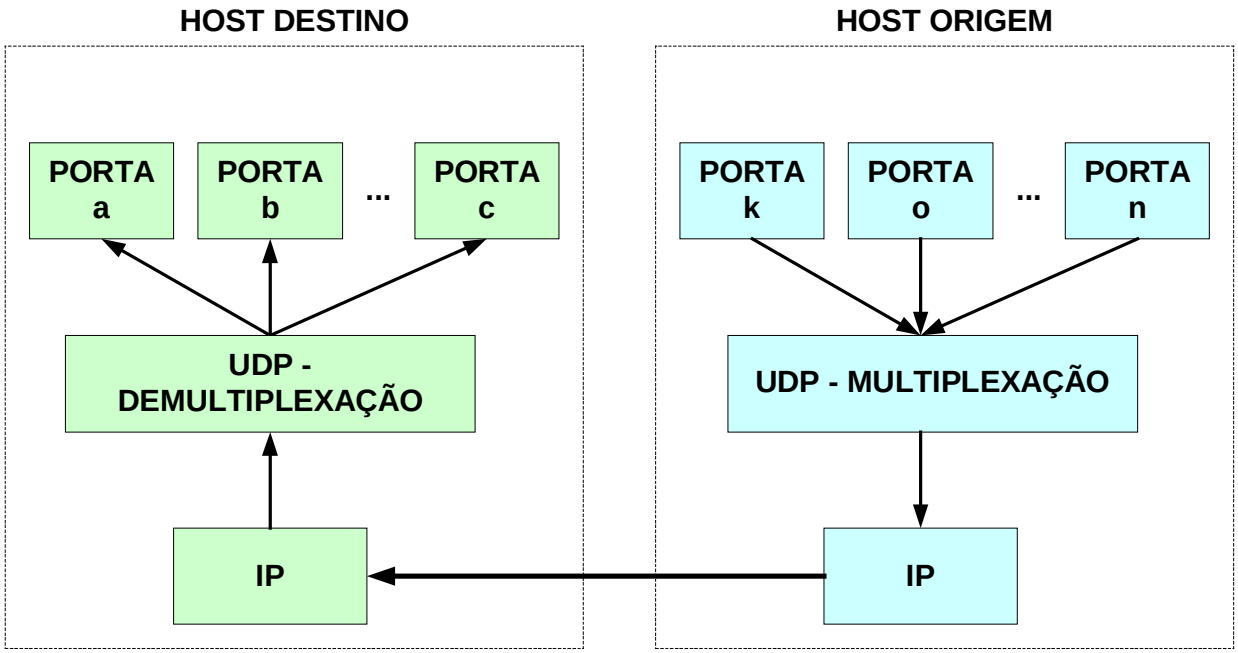

Figura 2.5 Multiplexação/Demultiplexação UDP

Essa multiplexação baseada em porta e não em conexão como o TCP que permite sua utilização com multicast, propriedade necessária para uma conferência multimídia [CRO98].

2.2.2 Real-Time Transport Protocol

O transporte de multimídia pela Internet possui algumas particularidades que o diferenciam das demais aplicações [SCH98a]:

- **Ordenação** – Os pacotes precisam ser reordenados em tempo-real ao chegarem a seu destino, pois muitas vezes eles chegam fora de ordem.

- **Perda de pacotes** – Um pacote perdido não pode ser simplesmente retransmitido, pois causaria atrasos. Esse pacote deve ser detectado e compensações devem ser feitas para não atrapalhar a inteligibilidade da mídia transmitida.
- **Reconhecimento de payload** – *Payload* é a mídia (áudio/vídeo) codificada e digitalizada para ser transmitida via rede. Algumas vezes é necessário reconhecer o tipo de mídia para ajustes. Cada pacote deve indicar o tipo de payload que está sendo transportado. Por exemplo: Em uma rede de largura de banda de 256kbps, uma conferência multimídia com dois usuários está utilizando uma determinada codificação que usa 64kbps de largura de banda (2x 64 = 128kbps somente de tráfego de mídia). Um novo usuário deseja entrar nessa conferência, e como isso novos recursos de banda deverão ser alocados (128kbps + 64kbps = 192kbps). O administrador da rede percebe que isso aumentará perigosamente o tráfego. Para revolver esse problema, ele altera a codificação utilizada na conferência para outra que usa 16kbps de largura de banda.
- **Reconhecimento de frame** – Alguns tipos de codificação de áudio e vídeo são enviados em unidades chamadas frames que possuem tamanhos fixos. O início e fim dos frames transportados devem que ser reconhecidos para que as aplicações de nível superior possam reordená-los.

Os principais protocolos de transporte usados na Internet (UDP e TCP) não possuem essas características. Pode parecer lógico usar TCP para transportar mídia, porém existem vários problemas:

- Transmissão confiável não é apropriada para dados sensíveis a atrasos como áudio e vídeo de tempo-real [RTP]. O TCP tentará sempre reenviar um pacote perdido, causando atrasos e esperas por parte do usuário. A perda de apenas um pacote (é aceitável até 5% de perda) não influirá na inteligibilidade da mídia.
- TCP não suporta multicast [COM98].
- O controle de congestionamento do TCP diminuirá a janela de congestionamento assim que detectar perda de pacotes (“*slow start*”) [MUR], diminuindo o fluxo de pacotes recebidos. Pacotes perdidos causariam graves problemas com áudio e vídeo que necessitam de um fluxo contínuo de dados sendo enviado.

Definido na RFC 1889 [SCH96], *Real-Time Transport Protocol* (RTP) é um protocolo da camada de transporte que é utilizado sobre UDP (apesar de operar sobre qualquer protocolo de transporte baseado em pacotes) e provê entrega fim-a-fim para aplicações que necessitam transmitir dados em tempo real.

O RTP não controla a perda ou reordenação de pacotes, mas provê informações para que aplicações de camadas superiores tratem desses assuntos. Com esse fim, ele possui campos como *timestamp* e número de seqüência, usados para reconstruir e sincronizar a mídia transportada.

Além do RTP, a RCF 1889 define o *RTP Control Protocol* (RTCP). Este protocolo também opera sobre UDP e fornece informações sobre a qualidade da sessão. Estas informações são compartilhadas entre os participantes via multicast. Todas as vezes que uma sessão RTP é iniciada, uma sessão RTCP também é aberta. A sessão RTP usa sempre uma porta par. O RTCP utiliza a porta ímpar imediatamente superior. A RFC 1890 [SCH96a] sugere como *default* usar as portas 5004 e 5005, respectivamente para RTP e RCTP.

2.2.2.1 Cabeçalho RTP

O cabeçalho de um pacote RTP possui o formato mostrado na Figura 4.1.1:

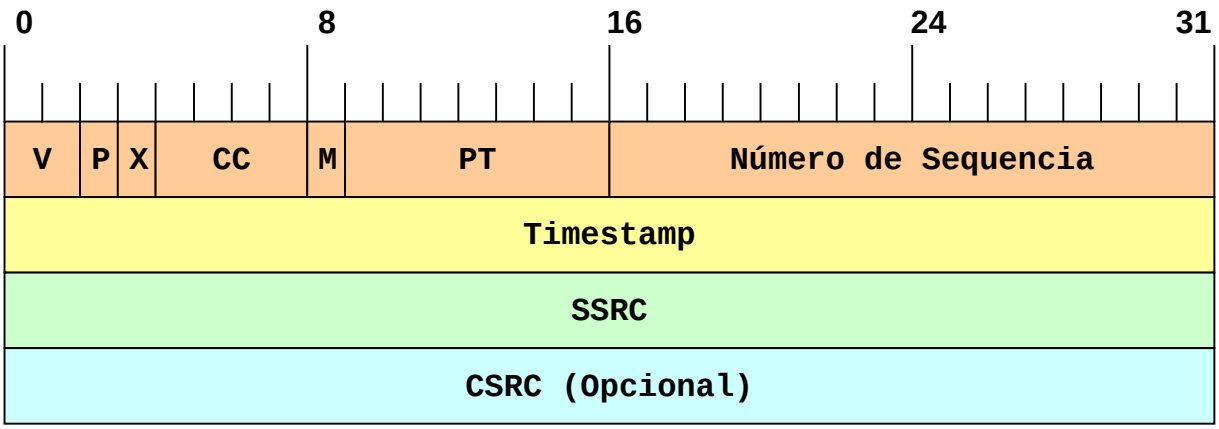

Figura 2.6 Cabeçalho RTP

Campo	Descrição	Tamanho (bits)
Version (V)	Indica a versão do RTP. A versão atual é a 2	2
Padding (P):	Indica se o pacote contém um ou mais octetos de preenchimento (*padding octets*) no final do payload, mas que não fazem parte dele. O último octeto do *padding* contém a quantidade de bytes que devem ser ignorados, inclusive ele mesmo. O *padding* pode ser necessário para assegurar o alinhamento correto para criptografia ou para carregar muitos pacotes RTP em um único pacote	1

	UDP.	
Extension (X):	Indica se o cabeçalho deve ou não estar seguindo de um cabeçalho de extensão, usado para aplicações que necessitam de informações adicionais no cabeçalho.	1
CSRC Count (CC):	Indica a quantidade de identificadores CSRC incluídos no cabeçalho, variando de 0 a 15	4
Marker (M):	A interpretação deste campo depende do tipo de *payload* que esta sendo transportado no pacote. A RFC 1890 especifica o uso deste bit	1
Payload Type (PT):	Indica o tipo de *payload* que esta sendo transportado e determina sua interpretação para as aplicações. Esses *payloads* devem ser registrados pela IANA (*Internet Assigned Numbers Authority*). O *payload* contém, dentro de si, cabeçalhos específicos para cada codificação	7
Número de Seqüência	O número de seqüência é iniciado randomicamente no início da sessão e incrementado para cada pacote RTP que é enviado. Através desse campo, o receptor então pode detectar perda de pacotes e também restaurar a seqüência original dos pacotes que por ventura, chegarem na ordem errada.	16
Timestamp	Indica o instante que o primeiro octeto do *payload* é criado, sendo que seu valor inicial é gerado randomicamente. Esse instante deve ser derivado de um relógio incrementado monoliticamente e linearmente no tempo, para que as aplicações possam gerenciar a sincronização e também calcular o *jitter*. A freqüência do relógio é dependente do formato do *payload*. O *timestamp* é incrementado de pacote em pacote, dependendo da quantidade de amostras de mídia contidas em um pacote.	32
Synchronization Source (SSRC)	Indica a entidade que é responsável por configurar o número de seqüência e o *timestamp*. Normalmente é transmissor do pacote RTP. O identificador é escolhido randomicamente pelo transmissor e não tem qualquer vínculo com endereços de rede. Este identificador deve ser único em uma sessão e preferencialmente gerado pela aplicação.	32
Contributing Source (CSRC):	Usada quando os pacotes provem de um *mixer*. Indica a SSRC original que gerou a mídia e que está por trás do *mixer*. Varia de 0 a 15 entradas de CSRC em um único pacote RTP	32

Tabela 2-4 Campos RTP

Os 12 primeiros bytes são obrigatórios para todos os pacotes RTP, enquanto o campo CSRC é usado apenas quando houver mixers envolvidos. Abaixo está a lista dos tipos de *payload* definidos na RFC 1890:

PT	Codificação	Mídia	Freqüência	Canais
0	PCMU	Áudio	8000	1
1	1016	Áudio	8000	1
2	G.726	Áudio	8000	1
3	GSM	Áudio	8000	1
4	Unassigned	Áudio	8000	1
5	DVI4	Áudio	8000	1
6	DVI4	Áudio	16000	1
7	LPC	Áudio	8000	1
8	PCMA	Áudio	8000	1
9	G.722	Áudio	8000	1
10	L16	Áudio	44100	2
11	L16	Áudio	44100	1
12	Unassigned	Áudio		
13	Unassigned	Áudio		
14	MPA	Áudio	90000	1

15	G.728	Áudio	8000	1
16-23	Unassigned	Audio		
24	Unassigned	Vídeo		
25	CelB	Vídeo	90000	
26	JPEG	Vídeo	90000	
27	Unassigned	Vídeo		
28	nv	Vídeo	90000	
30	Unassigned	Vídeo		
31	H.261	Vídeo	90000	
32	MPV	Vídeo	90000	
33	MP2T	AV	90000	
34-71	Unassigned	?		
72-76	Reservado			
77-95	Unassigned	?		
96-127	Dynamic			

Tabela 2-5 Tipos de payload

Tipos dinâmicos de *payload* usam a faixa de 96 a 127. Eles não são mapeados pela RFC 1890 ou pela IANA, mas apontam para um tipo de *payload* identificado por um protocolo externo de sinalização (como H.323 ou SIP). O mapeamento existe durante o tempo da sessão. Ou seja, no momento que o receptor “atender” a chamada, o protocolo de sinalização negocia com qual tipo de mídia/codificação a sessão será realizada e mapeia para um número de tipo de *payload* entre 96 e 127.

2.2.2.2 Mixer

É um sistema intermediário que recebe pacotes RTP de uma ou mais fontes, possivelmente modifica o formato dos dados, combina os pacotes de alguma maneira e então encaminha um novo pacote RTP. Múltiplos fluxos de mídia são combinados em apenas um.

Cada participante passa a trocar mídia com o mixer e não mais com os outros participantes. No cabeçalho do pacote enviado pelo mixer, cada fonte participante (SSRC) é acrescentada como uma fonte contribuinte (CSRC). O SSRC do pacote passa a ser o do mixer.

Um dos usos do mixer é ajustar largura de banda entre os participantes de uma sessão. Por exemplo, a Figura 2.6 mostra três participantes em uma sessão trocando mídia a 64Kbps. Se a banda disponível para cada um for apenas 64Kbps, um participante não poderá receber ou enviar mídia para outro. O mixer então combina os fluxos de 64Kbps em apenas um. Assim cada participante poderá receber e enviar dados.

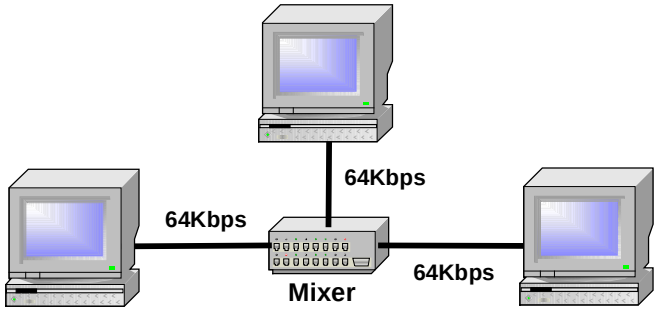

Tabela 2-6 Mixer

2.2.2.3 Tradutor

É um sistema intermediário que encaminha pacotes recebidos sem alterar o SSRC, podendo ou não haver alguma modificação no formato dos dados. São normalmente usados para possibilitar troca de mídia entre participantes dentro de *firewalls* ou que não suportam o mesmo tipo de formato de *payload* ou taxa de transmissão. Por exemplo, na Figura 2.7, um participante suporta apenas voz codificada com *payload* tipo 0 (G.711 64Kbps) enquanto o outro suporta payload tipo 2 (G.726 32Kbps). O tradutor se encarrega de converter a codificação áudio para ambos poderem conversar.

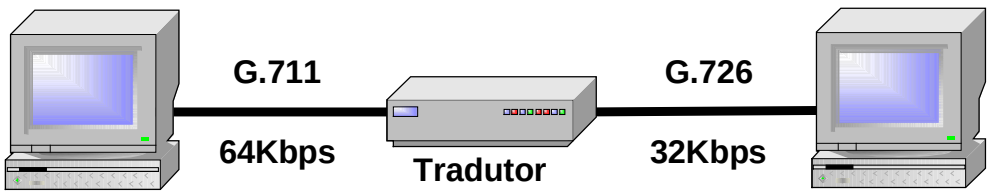

Figura 2.7 Tradutor

2.2.2.4 RTP Control Protocol

Definido na RFC 1889 juntamente com o RTP e também operando em cima do UDP, o *RTP Control Protocol* (RCTP) transmite periodicamente pacotes de controle para todos os participantes de uma sessão. Esses pacotes contêm dados sobre a qualidade da sessão que podem ser usados para detectar/resolver possíveis problemas que ocorram durante a sessão.

Existem cinco tipos de pacotes RCTP que transportam uma variedade de informações de controle:

Tipo de pacote	Descrição
SR (Sender Report)	Para transmissão e recepção de estatísticas dos participantes que são transmissores ativos

RR (Receiver Report)	Para recepção de estatísticas de participantes que não são transmissores ativos
SDES (Source Description)	Contém uma ou mais informações sobre um determinado participante de uma sessão, incluindo o canonical name (CNAME). O CNAME identifica unicamente um participante em uma sessão. O SSRC de um participante pode mudar se o computador reiniciar ou um mesmo participante pode ter vários, se estiver enviando múltiplos fluxos de mídia. Porém o CNAME permanecerá sempre o mesmo.
BYE	Indica o fim de uma participação em uma sessão
APP	Funções específicas de aplicações

Tabela 2-7 Tipos de pacote RCTP

Apesar de definidos individualmente, os pacotes devem ser enviados em um pacote composto como no exemplo da Figura 2.8. A RFC 1889 define que um pacote composto deve iniciar com um pacote de relatório (SR ou RR) e deve conter um pacote SDES.

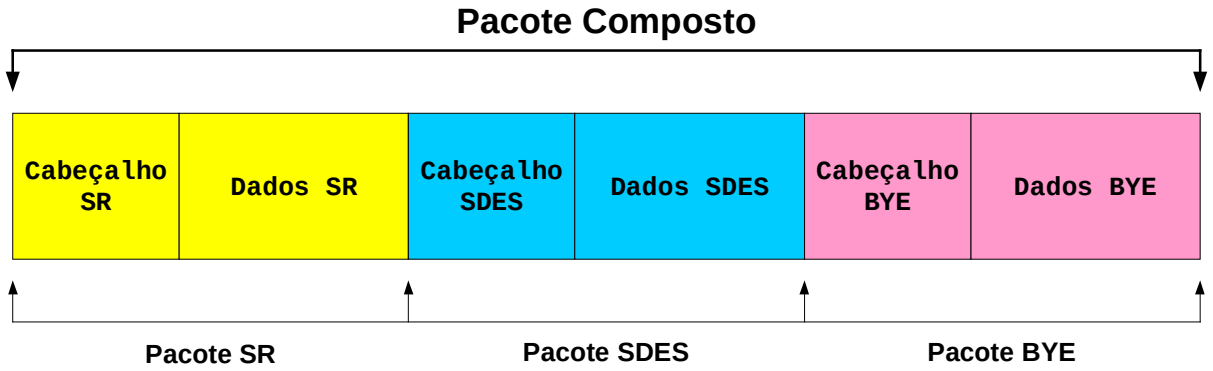

Figura 2.8 Pacote RTCP composto

2.2.2.5 Exemplo de utilização: Conferência de áudio usando multicast

Um endereço de grupo multicast e um par portas são alocadas para a sessão. Uma porta será utilizada para RTP e a outra para RTCP. Esse endereço e portas são distribuídos para os participantes da sessão.

A aplicação de cada participante começa a enviar áudio codificado em pequenos pedaços de, por exemplo, 20ms de duração. Cada pedaço (*payload*) é precedido de um cabeçalho RTP. Esse pacote RTP é encapsulado dentro de um pacote UDP.

A sessão está sujeita a "largura de banda da sessão" que é a banda usada pela codificação mais os cabeçalhos IP, UDP e RTP (40 bytes). Ou seja, para sessão que usa PCM de 64kbps seriam necessários acrescentar mais 16kbps, ficando então 80kbps. Dessa largura de banda da sessão, 5% deve ser reservado para o tráfego de pacotes RTCP que são enviados a cada 5 segundos aproximadamente [RTP]

O cabeçalho RTP indica o tipo de codificação (Ex. PCM) que está sendo utilizada. Isto servirá para possíveis ajustes de acordo com a disponibilidade de largura de banda. Ele também possui campos que possibilitam a detecção de perda de pacotes e sua reordenação quando necessário (*timestamp* e número de seqüência).

Em uma conferência é necessário conhecer os participantes e saber se eles estão recebendo o áudio de maneira satisfatória. Para isso, periodicamente, a aplicação envia por multicast relatórios indicando a qualidade da sessão através de pacotes RTCP (RTCP SR e RTCP RR). Além disso, são enviados dados sobre cada usuários da sessão (RTCP SDES). Quando um usuário sai da sessão, este envia um pacote RTCP BYE.

2.2.3 Session Initiation Protocol

Usando a definição da RFC 2543 [HAN99], *Session Initiation Protocol* (SIP) "... é um protocolo de controle da camada de aplicação que pode estabelecer, modificar e terminar sessões multimídia ou chamadas. Essas sessões multimídia incluem conferências multimídia, ensino a distância, telefonia Internet e aplicações similares...".

SIP é um protocolo cliente-servidor. Clientes enviam solicitações (*requests*) e recebem respostas (*response*). Os servidores recebem solicitações e enviam respostas.

2.2.3.1 User Agent (UA)

É a aplicação que fica com o usuário final. Pode ser implementada em hardware ou software. É formada por dois componentes:

- **User Agent Client (UAC)** Envia as requisições SIP.
- **User Agent Server (UAS)** Aceita as requisições SIP e contata o usuário.

Os dois componentes ficam no mesmo dispositivo. Agindo como um UAC, o dispositivo pode iniciar solicitações SIP. Agindo como um UAS, o dispositivo pode receber e responder solicitações SIP. Na prática, o dispositivo pode iniciar e receber chamadas, habilitando o SIP para comunicação ponto-a-ponto [ROS].

2.2.3.2 Servidor de Re-direcionamento (Redirect Server)

É um servidor que aceita solicitações SIP, relaciona o endereço de destino para zero ou mais endereços e repassa esse endereço traduzido ao criador da solicitação. Depois disso, o criador da solicitação pode enviar solicitações ao endereço traduzido devolvido pelo Servidor de Re-direcionamento.

2.2.3.3 Servidor Proxy (Proxy Server)

Clientes enviam pedidos para o Proxy, e este trata essas solicitações ele mesmo ou encaminha a outros servidores. Para esses outros servidores, aparece como se a mensagem estivesse partido do Proxy e não de uma alguma entidade escondida atrás dele. Como o Proxy envia e recebe solicitações, ele age tanto como cliente e servidor.

2.2.3.4 Servidor de Registro (Registrar)

SIP inclui o conceito de registro de usuário, por meio de que um usuário indica à rede sua disponibilidade em um endereço particular. Esse registro ocorre através de uma solicitação REGISTER feita ao Registrar pelo usuário. Normalmente, o Registrar é combinado com um Proxy e um Redirect em um mesmo dispositivo.

2.2.3.5 Estrutura das Mensagens SIP

SIP tem uma sintaxe baseada em texto, semelhante ao HTTP (*Hypertext Transfer Protocol*). Por causa disso, programas projetados para processar HTTP (Perl, Java e etc) podem ser adaptados com relativa facilidade para uso com SIP. Uma desvantagem comparada à codificação binária, utilizada por outros protocolos, é o maior consumo de largura de banda [COL01, SCH98a].

Mensagens de SIP podem ser solicitações de um cliente a um servidor ou respostas de um servidor para um cliente. Cada mensagem contém uma linha inicial seguida por zero ou mais cabeçalhos e opcionalmente seguido pelo corpo da mensagem, de acordo com a seguinte sintaxe:

Figura 2.9 Formato da mensagem SIP

SIP define dois tipos de mensagens: solicitações e respostas. Por isso a linha de início pode ser a linha de solicitação (*request-line*) que especifica o tipo de solicitação que é enviada, ou a linha de resposta (*status-line*) que indica o sucesso ou fracasso de uma determinada solicitação. No caso de fracasso, a linha indica o tipo ou a razão para o mesmo.

Os cabeçalhos de mensagem provêem informações adicionais relativas à solicitação ou resposta (a criador da mensagem, conteúdo e etc). Os cabeçalhos também oferecem os meios para levar informação adicional nas mensagens. Por exemplo, avisos de quando os usuários estarão disponíveis, assunto da conversa e etc.

O corpo de mensagem descreve o tipo de sessão a ser estabelecida, inclusive uma descrição dos tipos de mídia que serão trocados, codecs e etc. SIP não define a estrutura ou conteúdo do corpo de mensagem [SIP]. Isto é descrito usando um protocolo diferente. A estrutura mais comum para o corpo de mensagem utiliza o *Session Description Protocol* (SDP) que será descrito posteriomente.

2.2.3.6 Solicitações SIP

Uma solicitação SIP começa com uma request-line que possui a seguinte sintaxe:

Figura 2.10 Linha de início

Campos	Descrição
Método	Identifica a solicitação que foi emitida

Request-URI	Endereço da entidade para a qual a solicitação está sendo enviada. Este campo possui o formato URI (*Uniform Resource Identifier*)
Versão-SIP	Identifica a versão do SIP a ser utilizada

Tabela 2-8 Campos da linha de início de uma solicitação SIP

Os três componentes estão separados através de espaços, e a linha é terminada por um caractere de CRLF (*Carriage Return-Line Feed*). RFC 2543 define seis métodos diferentes:

Método	**Descrição**
INVITE	Método que inicia uma sessão, convidando uma pessoa a participar.
ACK	Uma vez que recebeu uma resposta final para um INVITE, o transmissor envia um ACK. Este método confirma que a resposta final foi recebida
BYE	Método que termina uma sessão. Pode ser enviado pelo transmissor ou receptor quando um deste quer terminar a sessão
OPTIONS	Este método examina um host sobre suas capacidades. Este método poderia ser usado, por exemplo, determinar se um usuário pode suportar um tipo particular de mídia ou determinar a sua disponibilidade (respondendo a um INVITE)
CANCEL	Termina uma solicitação pendente. Por exemplo, CANCEL poderia ser usado para terminar uma sessão onde um INVITE foi enviado, mas uma resposta final não foi recebida
REGISTER	Um UA usa o método de REGISTER para se logar e registrar seu endereço em um Servidor SIP, fazendo assim que o servidor saiba onde ele está localizado.

Tabela 2-9 Métodos SIP

2.2.3.7 Respostas SIP

A linha inicial de uma resposta SIP é uma *status-line*. Esta linha contém um código (um número de três dígitos) que indica o resultado de uma solicitação. Contém também um texto que descreverá o resultado. O software do cliente interpretará o código e tomará as medidas cabíveis, enquanto a descrição poderá ser apresentada ao usuário para que o mesmo possa entender melhor a resposta. A status-line possui a seguinte sintaxe:

Figura 2.11 Linha de início de uma resposta SIP

Campos	**Descrição**
Versão-SIP	Identifica a versão do SIP a ser utilizada
Código	Códigos de resposta definidos em RFC 2543 têm valores entre 100 e 699, sendo o primeiro dígito do código que indica a classe de resposta. Assim, todas as respostas codificadas entre 100 e 199 pertença à mesma classe. Cada código está acompanhado de sua descrição. As classes de respostas são as seguintes: **1XX** - Informativa **2XX** – Sucesso. Somente o código 200 é definido. Ele significa que a solicitação foi compreendida e executada. **3XX** – Redirecionamento. **4XX** - Fracasso de Pedido **5XX** - Fracasso de Servidor

	6XX - Fracasso Global Todas as respostas, com exceção de respostas de lXX, são consideradas finais e devem reconhecidas com uma mensagem ACK (se a mensagem que originou for um INVITE). A RFC 2543 especifica que respostas da classe 1XX são temporárias não precisam ser reconhecidas com ACK
Descrição	Um texto descritivo que indica o motivo ou comentário sobre a resposta

Tabela 2-10 Campos de uma linha de início de uma resposta SIP

2.2.3.8 Endereçamento SIP

As solicitações e respostas são enviadas a um endereço em particular. Em SIP, estes endereços são conhecidos como SIP URLs (*Uniforme Resource Locators*). Estes endereços têm a forma user@host que é semelhante a um endereço de e-mail. Embora pareçam semelhantes, eles são diferentes. Considerando que um endereço de e-mail usa um MAILTO URL (por exemplo, mailto: anna@yent.com), uma SIP URL tem a sintaxe :

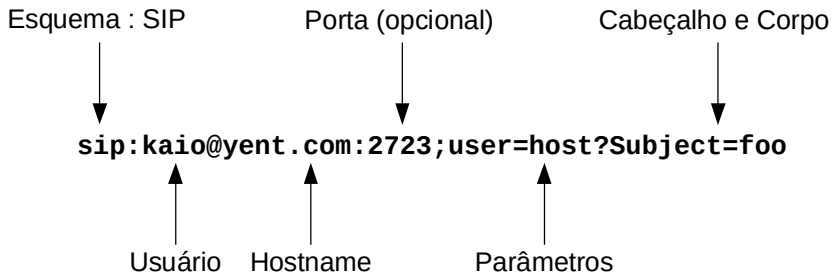

Figura 2.12 Endereçamento SIP

SIP permite endereços baseados em, por exemplo, URI telefônico, HTTP URL, MAILTO URL e etc.

2.2.3.9 Cabeçalhos de Mensagem

RFC 2543 define vários cabeçalhos (*headers*). Estes itens estão incluídos em uma solicitação ou resposta para prover informação adicional sobre a mensagem ou habilitar uma manipulação apropriada da mesma. Dependendo da solicitação ou resposta, determinados cabeçalhos podem ser obrigatórios, opcionais ou não aplicáveis. Existem quatro categorias principais [SIS]:

Categorias de cabeçalhos	**Descrição**
General Headers	Podem ser usados dentro de solicitações e respostas. Estes headers contêm informações básicas que são necessárias o gerenciamento de solicitações e respostas
Request Headers	Aplicam-se apenas às solicitações SIP e são usados para prover informação adicional para o servidor/cliente do pedido
Response Headers	Aplicam-se apenas respostas SIP (status). É usada para prover informação adicional à resposta que não pode ser incluída na *status-line*
Entity Headers	Em SIP, o corpo de mensagem contém informações sobre a sessão ou que devam ser apresentadas ao

	usuário. O propósito do *Entity* Headers é indicar o tipo e formato da informação incluída no corpo de mensagem, de forma que a aplicação apropriada possa ser chamada para agir na informação dentro do corpo de mensagem.

Tabela 2-11 Cabeçalhos SIP

2.2.3.10 Exemplo de seqüência de mensagens para Registro

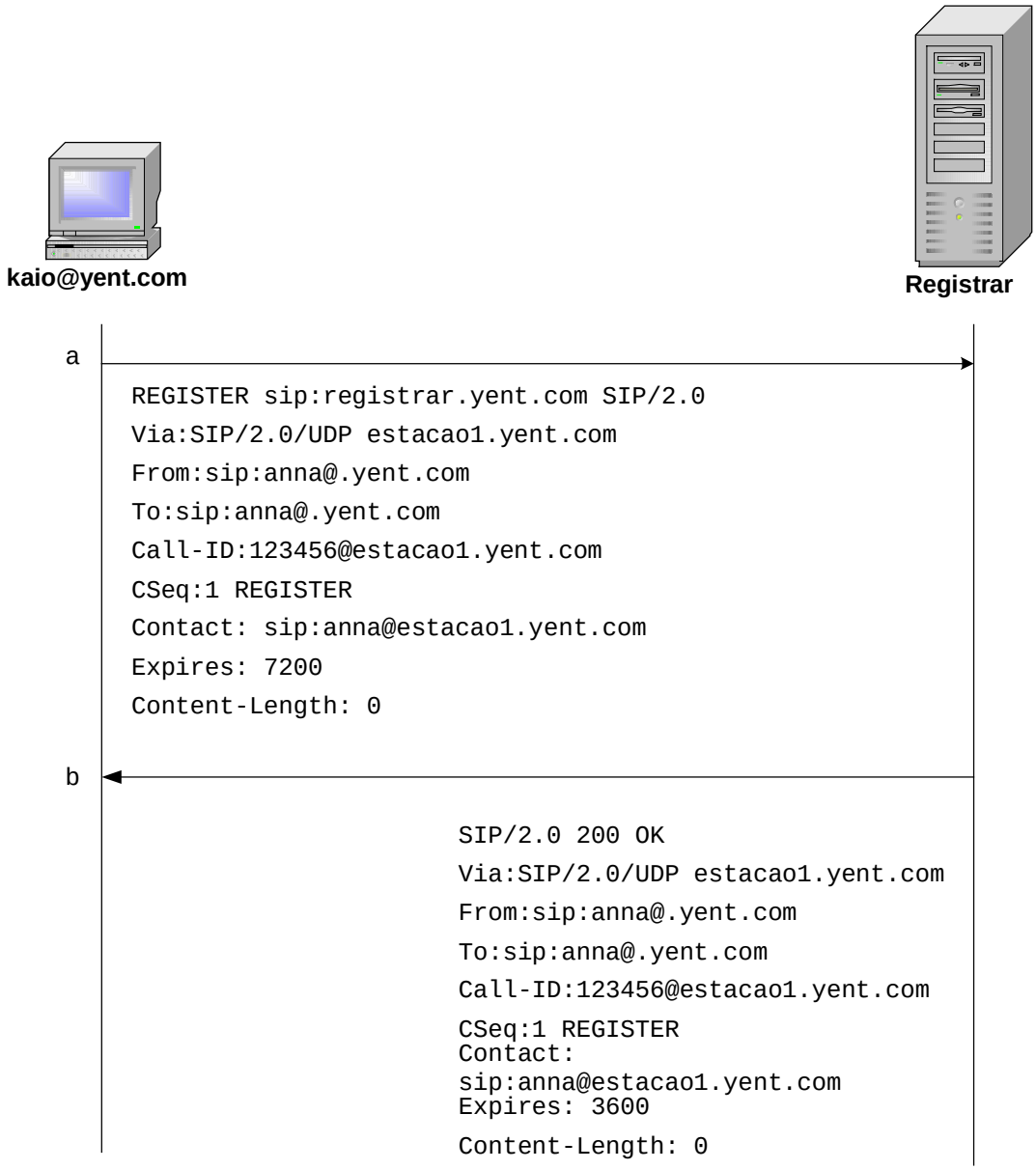

Figura 2.13 Registro

a. Anna logou-se no host estacao1@yent.com e pediu para se registrar.

Uma solicitação de registro (REGISTER) que será enviada ao Servidor de Registro Local. O endereço do Servidor de Registro Registrar é descrito na linha inicial: `sip:registrar.yent.com`.

- O campo de cabeçalho `Via:` contém o caminho tomado pela solicitação até agora (no caso estacao1.yent.com). Indica também o protocolo de transporte utilizado (UDP é *default*).
- O campo `From:` indica o endereço de quem iniciou o registro.

- O campo `To`: indica o "endereço de registro" que o Servidor Registrar irá armazenar para aquele usuário. No caso de uma requisição de registro, os campos To: e From: serão idênticos. A exceção acontece quando um usuário está registrando um outro.
- O campo `Call-ID:` identifica unicamente uma chamada.
- O campo `Cseq:` Indica o método usado na solicitação e é um número de inteiro. Solicitações consecutivas para o mesmo Call-ID incrementarão este número.
- O campo `Contact:` indica que as mensagens devem ser roteadas para sip:anna@estacao1.yent.com
- O campo `Expires:` indica que Anna pediu para o registro ficar ativo por duas horas
- Uma solicitação de registro não contém um corpo de mensagem, porque a mensagem não é usada para descrever uma sessão de qualquer tipo. Por isso o campo `Content-Legth:` esta setado para 0 (zero)

b. O Servidor de Registro responde

Resposta positiva para a solicitação de registro, como podemos observar pelo código de resposta 200 (OK) na status-line.

- Os campos `Via:, From:, To:, Call-ID:,Contact:, Content-Length:` foram copiadas com os mesmo valores da solicitação.
- O Servidor Registrar escolheu conceder apenas uma hora de registro. Quando Registrar muda o valor do Expires: muda sempre para um valor menor que o solicitado. Nunca maior. Este valor pode ser expresso em segundos ou uma data específica.

2.2.3.11 Exemplo de seqüência de mensagens para iniciar uma sessão

O convite (INVITE) é a solicitação que inicia uma sessão (estabelece uma chamada).

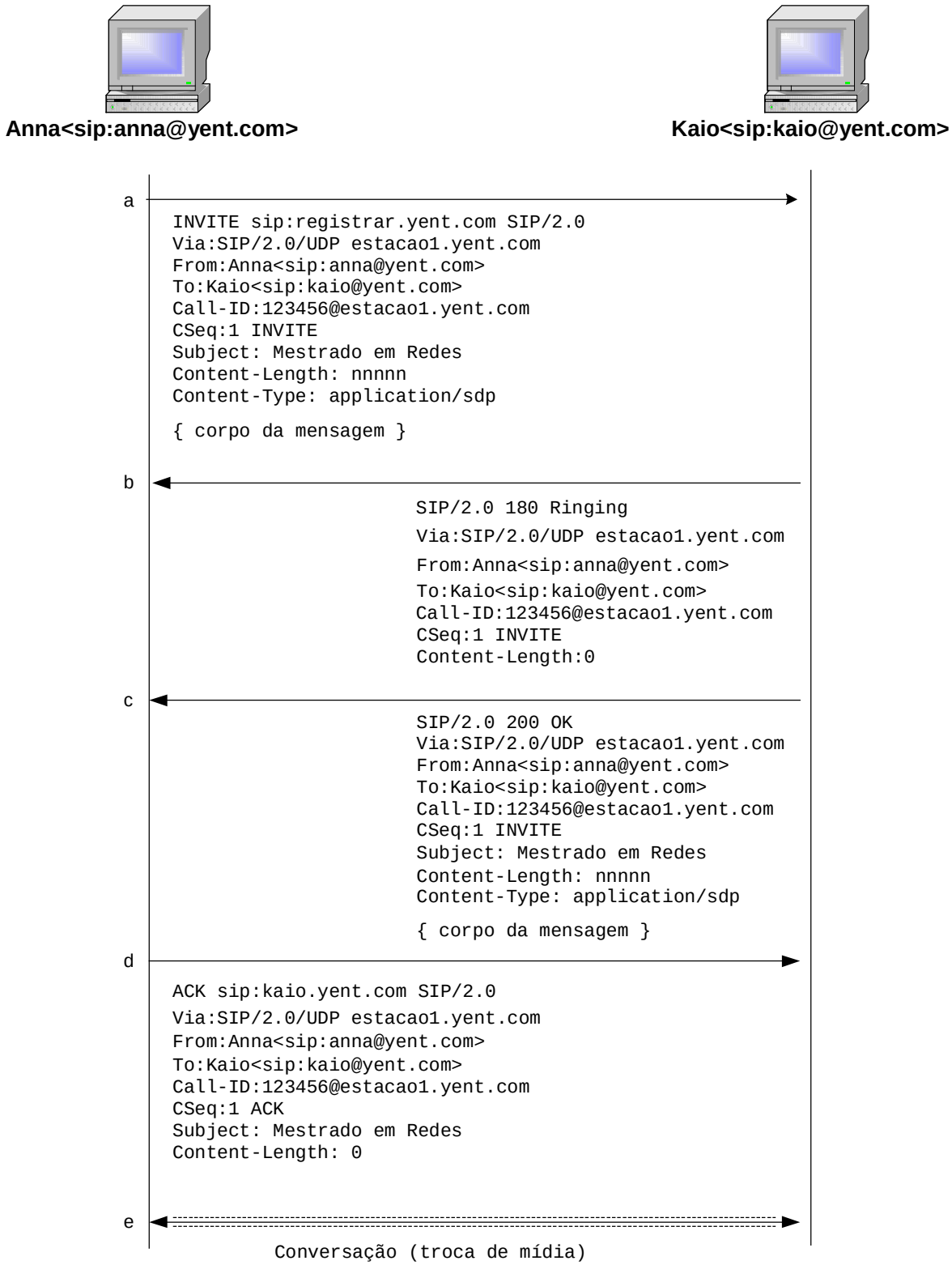

Figura 2.14 Iniciando uma sessão

a. Anna faz uma chamada para Kaio

Uma solicitação INVITE é enviada a `kaio@yent.com` como pode ser observado na Request-URI.

- O campo `To:` tem o mesmo valor da Resquest URI, porque neste caso, nós não estamos atravessando nenhum Servidor Proxy
- O campo `From:` indica que a chamada veio de `anna@yent.com`. SIP habilita a utilização de um nome para ser usado em conjunto com o SIP URL. Assim, o terminal poderia exibir o nome Anna enquanto estivesse alertando Kaio sobre a chamada.
- O campo `Subject:` é opcional e serve para indicar o assunto geral da chamada
- O campo `Content-Type:` indica neste caso que o corpo da mensagem será descrito utilizando o protocolo SDP.
- O campo `Content-Length:` deve possuir um valor NNN maior que 0(zero) dependendo do tamanho do corpo da mensagem.
- O tipo de mídia que Anna deseja usar é descrito dentro do corpo de mensagem.

b. Kaio é alertado quanto à chamada (código 180).

c. Kaio responde positivamente (código 200 OK). O corpo da mensagem descreve as mídias que a Kaio deseja usar.

d. Anna envia um ACK para confirmar o recebimento da resposta. Depois que um ACK foi enviado, as partes podem conversar (trocar mídia).

2.2.3.12 Seqüência de mensagens para finalização de uma chamada

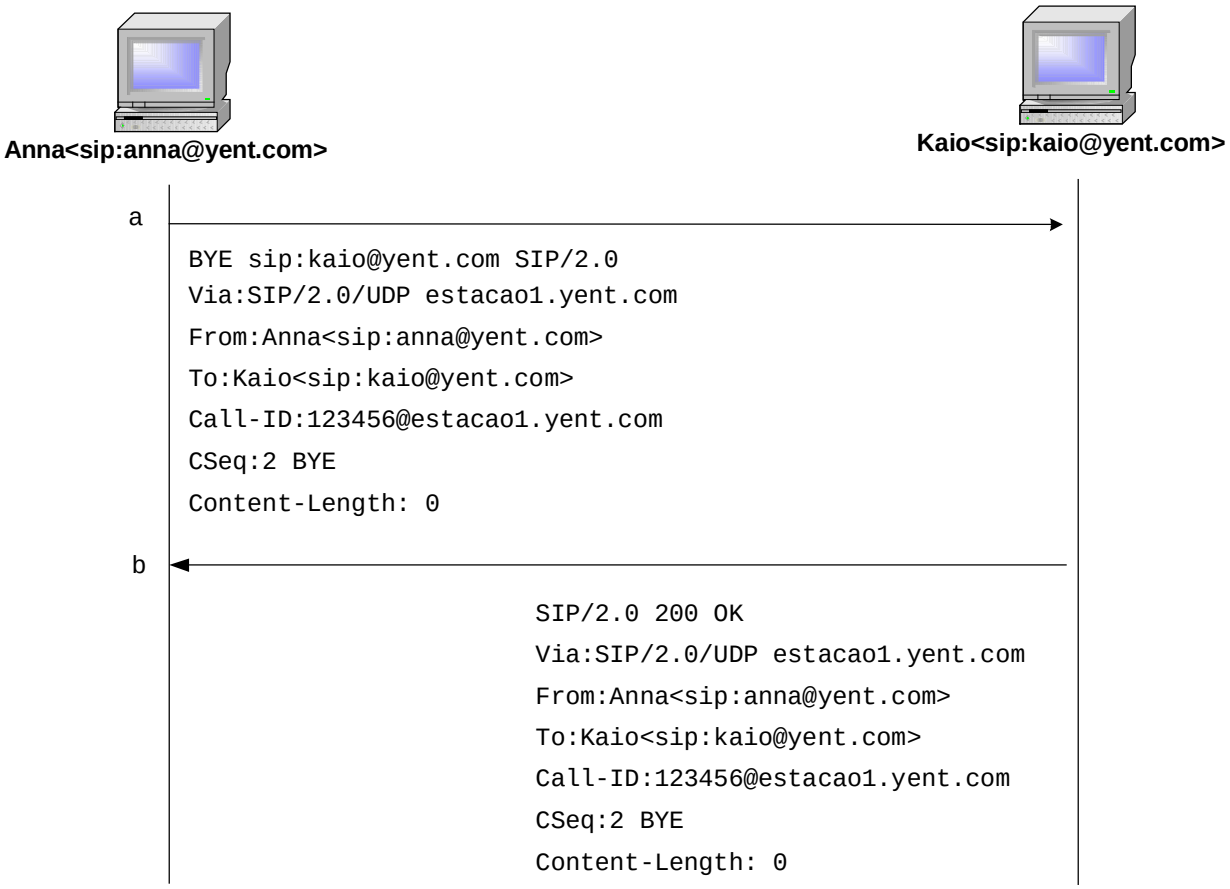

Figura 2.15 Finalizando uma chamada

A parte que deseja terminar a chamada envia uma solicitação BYE. Na Figura 2.15, Anna termina a chamada iniciada na sessão anterior com um INVITE. Ao receber o BYE, Kaio imediatamente interrompe a transmissão de mídia e responde com um OK (código 200). A chamada então é finalizada.

2.2.3.13 Seqüência de mensagens que utilizam re-direcionamento

A resposta padrão de um servidor de re-direcionamento é um novo endereço de contato como alternativa ao endereço para onde a solicitação se dirigiria. A exceções são casos onde o servidor recebeu uma solicitação que não pode ser suportada, onde devolverá respostas 4XX, 5XX, ou 6XX, ou quando o servidor recebe uma solicitação de cancelamento, que deverá ser respondida com um 200 OK. A Figura 2.16 mostra um caso típico de re-direcionamento:

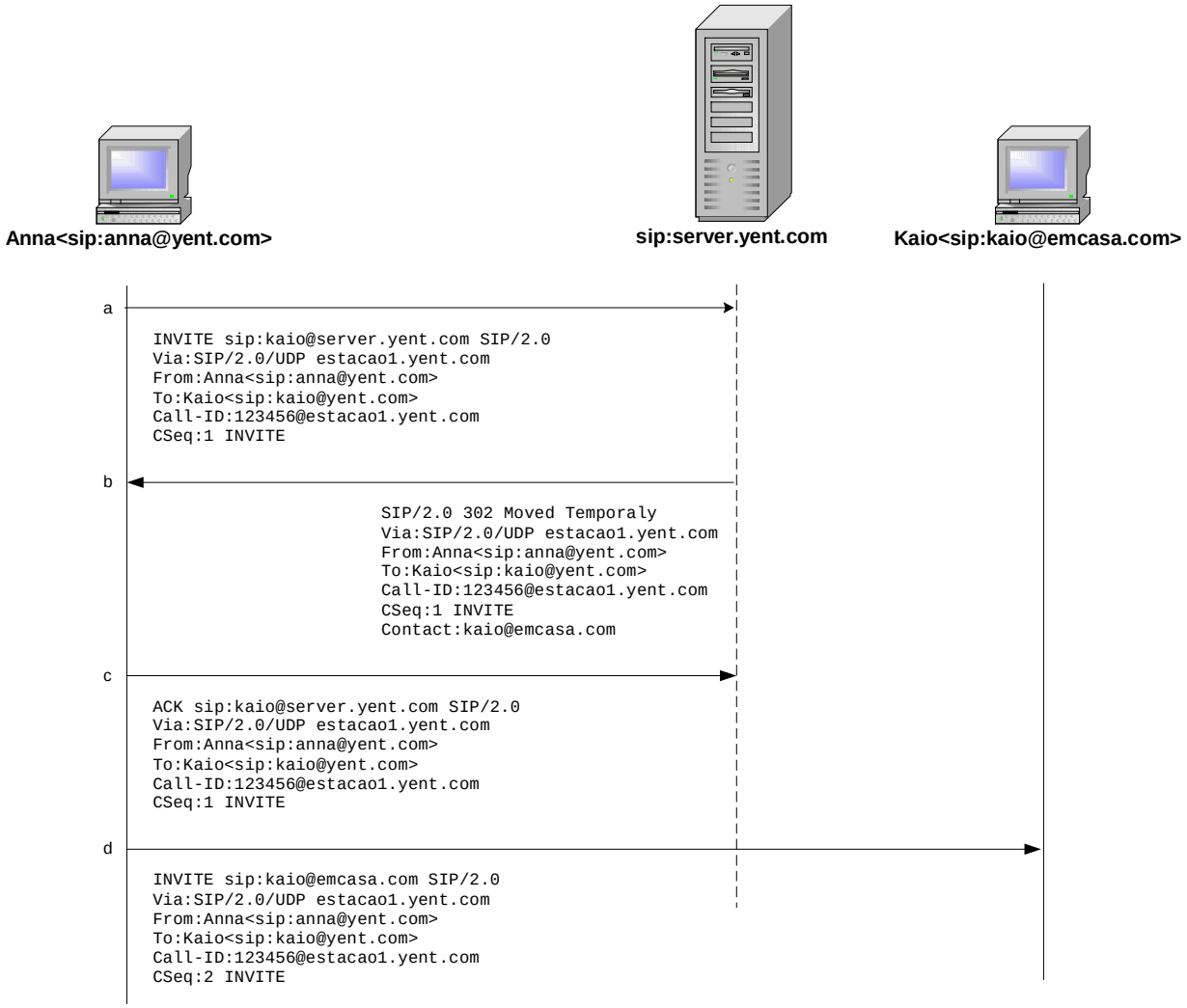

Figura 2.16 Redirecionamento

a. Anna envia uma solicitação INVITE para Kaio no Servidor de Re-direcionamento

b. O Servidor de Re-direcionamento responde com o código 302 (movido temporariamente) e, dentro da resposta, no campo `Contact:`, o endereço alternativo.

c. Anna envia um ACK deixando claro que entendeu o endereço alternativo

d. Anna cria um novo INVITE. Esse novo INVITE usa o endereço recebido do Servidor de Re-direcionamento como o Resquest-URI, enquanto o campo `To:` permanece sem alteração. O

campo Cseq: foi incrementado. O campo `To:` permaneceu igual para permitir ao cliente-destino saber que a mensagem foi originalmente enviada a um outro número. Com esse dado acrescido da identificação do remetente e do assunto (`Subject:`), o destinatário pode decidir ou não se aceita a chamada.

2.2.3.14 Seqüência de mensagens que utilizam servidor de Proxy

Um Servidor de Proxy está situado entre um UAC e o UAS remoto. Um Proxy aceita solicitações e as encaminha.

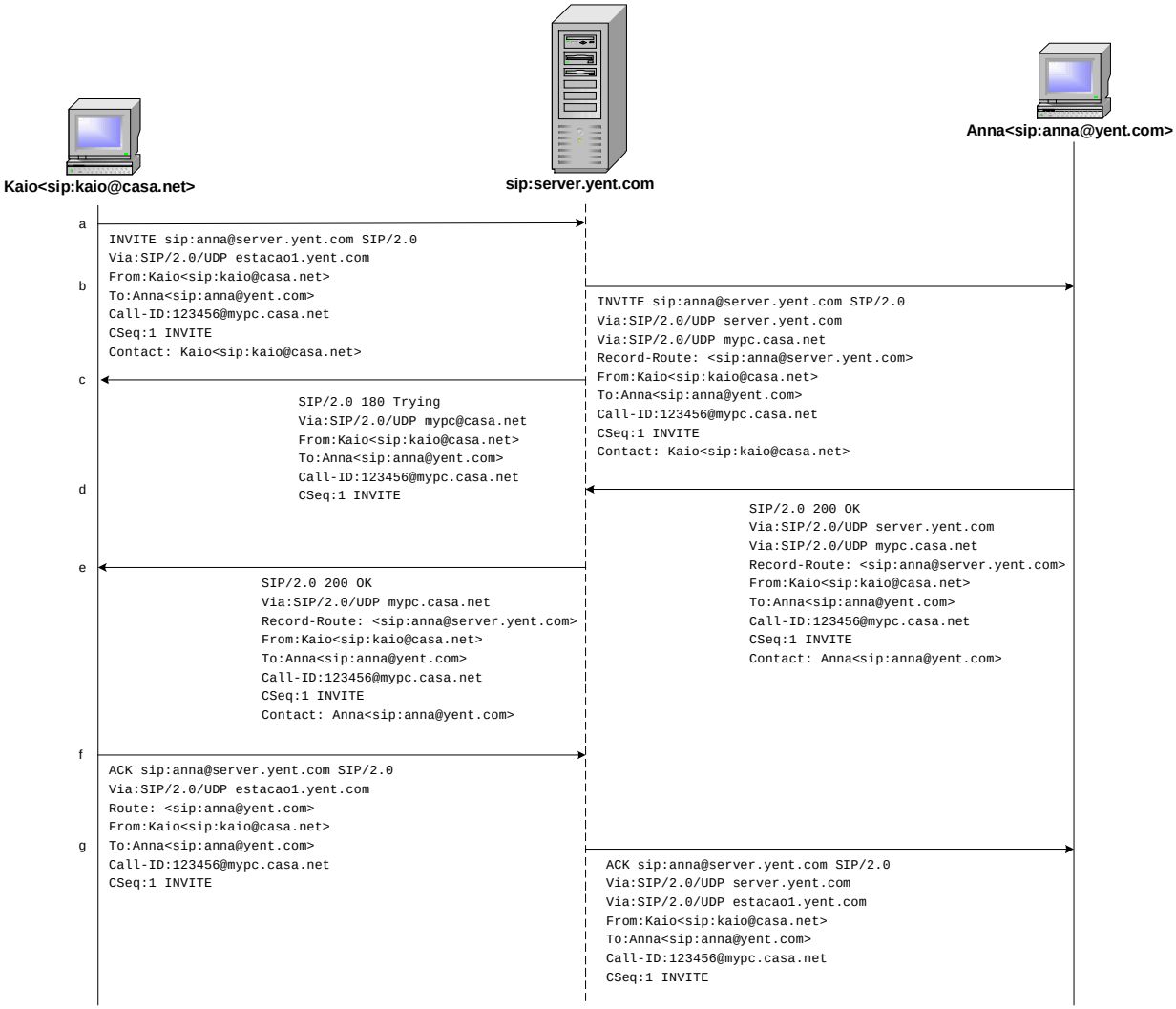

Figura 2.17 Proxy

a. Kaio faz uma chamada para Anna e chamada passa por um Proxy.

b. O Proxy recebe o INVITE.

c. O Proxy responde com o código 180 (Tentando), enquanto encaminha o INVITE. Fazendo isso, o Resquest-URI é alterado.

d. Quando uma solicitação é gerada, o cliente original insere seu próprio endereço em um campo `Via:`. Cada Proxy no caminho também insere seu endereço em um novo campo `Via:`, colocado frente de qualquer `Via:` já existente.. O campo `Via:` é usado para indicar o caminho seguido por uma solicitação até aquele momento. Então, a coleção de campos `Via:` provê um mapa do caminho levado pela rede por uma determinada solicitação. Quando um Proxy recebe

uma solicitação, este confere primeiro se seu próprio endereço já está incluído em um dos campos de `Via:`. Se estiver, indica que a solicitação já passou por ele e está acontecendo um loop. Quando isso acontece, o Proxy responderá a solicitação com o código 482 (loop detectado). Se não estiver, o Proxy encaminhará a solicitação depois de inserir seu próprio endereço em um novo campo Via.

e. Respostas também incluem campos `Via:` que são usados para enviar uma resposta através da rede ao longo do mesmo caminho que a solicitação, apenas em sentido inverso. Quando um Proxy recebe uma resposta, o primeiro campo `Via:` deveria conter seu próprio endereço. Se não contiver, um problema ocorreu e a mensagem é descartada. Assumindo que o primeiro campo `Via:` indica o próprio Proxy, então este remove o campo e checa se existe um segundo campo Via. Se não existir, então a mensagem é destinada a ele próprio. Se um segundo campo `Via:` existir, então o Proxy encaminha a resposta para o endereço contido naquele campo. Deste modo, a resposta encontra seu transmissor seguindo o caminho que a solicitação seguiu originalmente.

2.2.3.14.1 Estado do Proxy

Um Proxy pode ser sem estado (*stateless*) ou com estado (*stateful*). No primeiro caso, o Proxy recebe as solicitações que chegam, realiza alguma tradução necessária, e as encaminha, sem armazenar qualquer informação sobre elas. No segundo caso, o Proxy armazena informações sobre as solicitações que entraram e saíram. Com isso pode tomar decisões melhores em relação as subseqüentes solicitações e respostas relacionadas à mesma sessão.

No exemplo mostrado pela Figura 2.17, todas as mensagens e respostas atravessam o mesmo Proxy. Porém, isso nem sempre ocorre. Enquanto a resposta tem que voltar pelo mesmo caminho feito pela solicitação, duas solicitações sucessivas não precisam seguir o mesmo caminho. Dado que um INVITE inicial possa conter um campo `Contact:` e respostas também possam conter um campo `Contact:`, as duas partes podem ter informação de endereçamento necessária para se comunicarem diretamente. Conseqüentemente, depois de uma solicitação INVITE inicial e sua resposta, as solicitações subseqüentes e suas respectivas respostas podem ser enviadas fim-para-fim.

Um Proxy pode forçar sua permanência no caminho entre as partes para armazenar dados sobre a chamada. Ele faz isso usando o campo `Record-Route:`.

Quando possui estado, cada Proxy ao longo do caminho do INVITE insere seu próprio endereço no campo `Record-Route:`, que se torna uma lista de endereços. Como a inserção é feita no topo da lista, a primeira entrada na lista é o endereço último Proxy usado. O campo `Record-Route:` possui a lista de todos os Proxies na direção receptor-transmissor.

Quando o transmissor recebe uma resposta que contenha o campo `Record-Route:`, ele o copia na ordem inversa no campo `Route:`. Então, o campo `Route:` possui a lista de todos os Proxies na direção transmissor-receptor para servidor-destino. O transmissor adiciona o conteúdo do campo `Contact`: recebida do receptor no fim da lista de endereços, remove o primeiro endereço de Proxy e envia a próxima mensagem (por exemplo, um ACK) para o Proxy. Cada Proxy ao longo do caminho transmissor-receptor removerá a primeira entrada do campo `Route:` antes de enviar a mensagem à próxima entidade, que pode ser um outro proxy ou o servidor-destino. Esta abordagem melhora o desempenho na rede, pois cada Proxy sabe para onde encaminhar a solicitação, evitando a perda de ordem entre as mensagens.

2.2.4 Session Description Protocol (SDP)

Nas sessões anteriores, foi visto que a descrição do tipo de mídia a ser trocado estaria referenciada no corpo da mensagem. Além disso, nós vimos que o formato da descrição quase sempre será descrito usando SDP (*Session Description Protocol*) especificado em RFC 2327 [HAN98]. Nesta sessão descreveremos uma visão geral deste protocolo e sua utilização com o SIP.

2.2.4.1 Estrutura de SDP

SDP provê um formato simples para descrever informação de sessão a seus potenciais participantes. Basicamente, uma sessão consiste em vários fluxos de mídia. Então, a descrição de uma sessão envolve a especificação de vários parâmetros relacionados a cada dos fluxos de mídia e a sessão como um todo. Existem parâmetros de nível de sessão e parâmetros de nível de mídia. Os parâmetros de nível de sessão incluem informações como o nome da sessão, o criador da sessão, e o tempo em que a sessão estará ativa. Informações do nível de mídia incluem tipo, número da porta, protocolo de transporte, e formato da mídia.

Como SDP provê apenas descrições de sessão e não controla os meios por transportar ou anunciar as sessões aos potenciais participantes, ele deve ser utilizado em conjunto com outros protocolos (Por exemplo, SIP que leva informação de SDP dentro do corpo de mensagem).

Semelhante ao SIP, SDP é um protocolo baseado em texto que utiliza o conjunto de caracteres ISO 10646 em codificação UTF-8. Esta codificação habilita o uso de idiomas múltiplos e inclui o EUA-ASCII como um subconjunto. Enquanto campos do SDP usam somente EUA-ASCII, informação textual pode ser passada em qualquer língua.

2.2.4.2 Sintaxe de SDP

SDP carrega informação de sessão usando várias linhas de texto. Cada linha usa o formato *campo=valor* onde campo é exatamente um caractere (sensível ao caso) e valor depende do campo em questão. Em alguns casos, valor poderia consistir em vários pedaços separados de informação separados por espaços. Nenhum espaço é permitido entre campo e o sinal = (igual) ou entre o sinal = e o valor.

Os campos de nível de sessão devem ser incluídos primeiro, seguidos dos campos de nível de mídia.

A divisão entre dados de sessão e dados de mídia acontece na primeira ocorrência do campo de descrição de mídia (m=). Cada ocorrência subseqüente do campo de descrição de mídia marca o começo de dados relacionado a outros fluxos de mídia na sessão.

2.2.4.3 Campos

SDP inclui campos obrigatórios que devem ser incluídos em qualquer descrição de sessão, e campos opcionais (que pode ser omitidos). Alguns campos opicionais só aplicam a sessão, e alguns aplicam ao nível de mídia. Alguns campos opcionais podem ser aplicados tanto ao nível de sessão quanto ao nível de mídia. Em tais casos, o valor aplicado ao nível de mídia anula o valor de sessão para aquela instancia de mídia. As tabelas abaixo mostram os campos obrigatórios e opcionais.

Campo	**Descrição**
v =	(versão do protocolo) Este campo também marca o começo da descrição de uma sessão e o fim de qualquer descrição de sessão prévia
o =	(origem da sessão ou criador e identificador de sessão)
s =	(nome de sessão) Este campo é uma string de texto que poderia ser exibida aos potenciais participantes da sessão.
t =	(tempo da sessão) Este campo provê o tempo de começo e fim para a sessão.
m =	(mídia) Este campo indica o tipo de mídia, a porta de transporte para o qual os dados deveriam ser enviados, o protocolo de transporte (por exemplo, RTP), e o formato da mídia (tipicamente um formato de payload de RTP). O aparecimento deste campo também marca o limite entre informação de sessão e informação de mídia ou entre descrições de mídia diferentes

Tabela 2-12 Campos SDP obrigatórios

Campo	**Descrição**
i =	(informação de sessão) Este campo é uma descrição de texto da sessão e seu propósito é prover maior detalhe que o nome de sessão. Este campo pode ser especificado ao nível de sessão e ao nível de mídia
u =	(URI de descrição) Este campo é um URI (por exemplo, um endereço na WEB) onde se pode obter informações adicionais da sessão. Só um URI pode ser especificado por sessão
e =	(endereço de e-mail) Este campo é um endereço de e-mail da pessoa que é responsável pela a sessão. Várias instâncias deste campo podem existir, e estes campos seriam usados no caso que vários indivíduos poderiam ser contatados para maiores informações. Este campo é aplicado só ao nível de sessão.
p =	(número de telefone) Este campo é um número de telefone da pessoa que é responsável pela sessão. Como para o campo de endereço de e-mail, pode haver várias instâncias deste campo. Aplicável ao nível de sessão.

c =	(informação de conexão) Este campo provê dados de conexão incluindo, tipo de conexão, tipo de rede e endereço de conexão. Este campo pode ser aplicado ao nível de sessão ou ao nível de mídia.
b =	(informação de largura de banda) Especifica a largura de banda (bandwidth) necessária em kilobits por segundo (Kbps). Quando este campo está no nível de sessão, especifica a banda necessária para a conferência inteira. Quando está no nível de mídia, especifica a banda necessária para um determinado tipo de mídia.
r =	(quantidade de repetições) Indica a quantidade de vezes e a freqüência que uma sessão pré-programada será repetida.
z =	(ajustes de timezone) Em sessões pré-programadas, este campo é usado para ajustar o horário marcado. Isto acontece em decorrência da diferença de fusos horários ao redor do mundo.
k =	(chave de cifragem) Este campo provê uma chave de cifragem ou especifica um mecanismo pelo qual uma chave pode ser obtida com a finalidade de cifrar ou decifrar as mídias. O campo pode ser aplicado no nível de sessão, ao nível de mídia, ou ambos.
a =	(atributos) Este campo é usado para descrever atributos adicionais que foram aplicados à sessão ou as mídias individuais.

Tabela 2-13 Campos SDP opcionais

2.2.4.4 Ordenação dos Campos

A para evitar ambigüidades, SDP define a ordem que os campos são colocados tanto no nível de sessão quanto no nível de mídia. As tabelas abaixo mostram as ordens dos campos:

Ordem	**Campo**
1	Versão do Protocolo (v)
2	Origem (o)
3	Nome da Sessão (s)
4	Informação sobre a sessão (i) (opcional)
5	URI (u) (opcional)
6	E-mail (e) (opcional)
7	Telefone (p) (opcional)
8	Informação de Conexão (c) (opcional)
9	Largura de Banda (b)
10	Tempo(t)
11	Quantidade de repetições (r) (opcional)
12	Ajustes de timezone (z) (opcional)
13	Chave de criptografia (k) (opcional)
14	Atributos (a) (opcional)

Tabela 2-14 Ordem dos campos no nível de sessão

Ordem	**Campo**
1	Descrição da Mídia (m)
2	Informações adicionais (i) (opcional)
3	Informações de Conexão (c) (opcional se especificado no nível de sessão)
4	Informação de largura de banda (b) (opcional)

5	Chave de Cifra (k) (opcional)
6	Atributos (a) (opcional)

Tabela 2-15 Ordem dos campos no nível de mídia

2.2.4.5 Sub-campos

Alguns campos SDP possuem sub-campos. Nestes casos, os campos obedecem a seguinte sintaxe:

Figura 2.18 Sub-campos

A tabela abaixo mostra alguns importantes para uso com SIP:

Campo	Sub-campo	Descrição
Origem (o)	Username	Armazena a identidade de login do criador da sessão
	Sessão ID	Identificador único da sessão. Para assegurar que o ID é realmente único, a RFC 2327 recomenda que o ID faça uso do timestamp do Network Time Protocol (NTP).
	Versão	É um número de versão para uma sessão de particular.
	Tipo de Rede	É uma string que indica o tipo de rede. IN significa Internet
	Tipo de Endereço	É o tipo de endereço de rede. Pode ser IP4 (IP versão 4) ou IP6 (IP versão 6).
	Endereço	É o endereço de rede da máquina onde a sessão foi criada. Este endereço pode ser um nome de domínio completamente-qualificado (fully-qualified domain name) ou pode ser endereço de IP atual (somente quando a máquina não está debaixo de um LAN Proxy/NAT)
Informação de Conexão (c)	Tipo de rede	Atualmente somente o valou IN (Internet) está especificado
	Tipo de endereço	Atualmente somente IP4 (IP Versão 4) está especificado
	Endereço de conexão	É o endereço para o qual onde os dados deveriam ser enviados
Informação de Mídia (m)	Tipo de Mídia	O tipo de mídia pode ser áudio, vídeo, aplicação, dados, ou controle. Quando se tratar de voz, devemos selecionar áudio
	Porta	Indica o número da porta para o qual mídia deveria ser enviada. O número da porta depende do tipo de conexão em questão e o protocolo de transporte. Para VoIP, porém, as mídias são levadas normalmente por RTP usado em cima de UDP. Assim, o número de porto será um valor entre 1024 e 65535.
	Formato	Lista os vários tipos de mídia suportados ordenados por preferência. Normalmente, o formato será um RTP payload com o tipo de payload correspondente.
Atributos (a) usando rtpmap	Payload Type	Corresponde ao tipo de RTP payload
	Encoding Name	String que mostra o nome da codificação
	Clock Rate	Taxa em Hz
	Encoding Parameters	Podem ser usados para indicar o número de canais auditivos.

Tabela 2-16 Sub-campos

O campo atributos (a=) é usado para prover informações adicionais. O atributo *rtpmap* é um dos mais importantes em VoIP. Ele pode ser aplicado a um fluxo de mídia e pode ser particularmente útil quando o formato de mídia não é um RTP payload estático

2.2.4.6 Uso de SDP com SIP

Enquanto SIP provê os mecanismos para o estabelecimento de sessões de multimídia, SDP provê uma linguagem para descrever essas sessões. A figura abaixo mostra um exemplo de uso de SDP com SIP (para facilitar a leitura algumas alguns cabeçalhos foram omitidos):

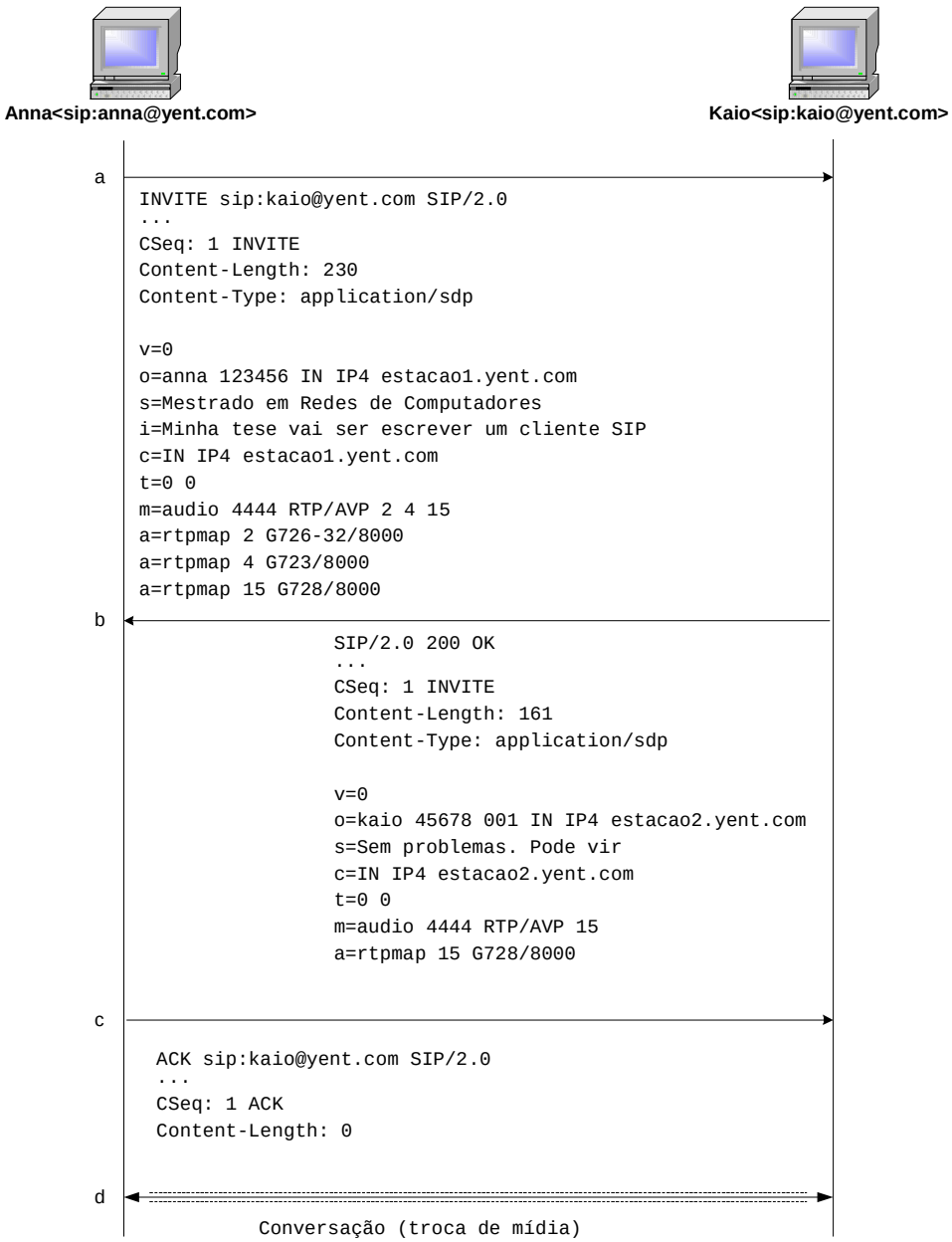

Figura 2.19 SDP com SIP

a- Anna faz uma chamada para Kaio. Anna suporta codificação de voz baseada em G.726, G.723 ou G.728. Kaio suporta somente G.728. Neste exemplo aparece o uso do atributo *rtpmap* para cada tipo de mídia suportada.

Como as duas partes suportam G.728, a conversação pode ser iniciada. Caso contrário, seria devolvida uma resposta 488 (Não Aceitável Aqui) ou uma 606 (Não Aceitável). Além disso, a resposta deve conter um campo `Warning:` com o código de advertência de 304 (tipo de mídia não suportada) ou 306 (formato de mídia incompatível).

3 Aplicação de Telefonia IP

Este capítulo descreve a aplicação de telefonia IP desenvolvida no escopo deste trabalho.

3.1 Visão geral da aplicação

Existe a necessidade de comunicar dentro da mesma rede local (100Mbps), computadores usando Windows 2000 Professional. Além disso, existem setores abrangidos pela mesma rede que utilizam Linux. A aplicação deverá permitir que as estações de trabalho troquem mensagens de voz não importando o sistema operacional. Um outro pré-requisito é poder comunicar-se com uma estação de trabalho que fica em outra rede. As redes estão ligadas via Internet através de um link de 64Kbps.

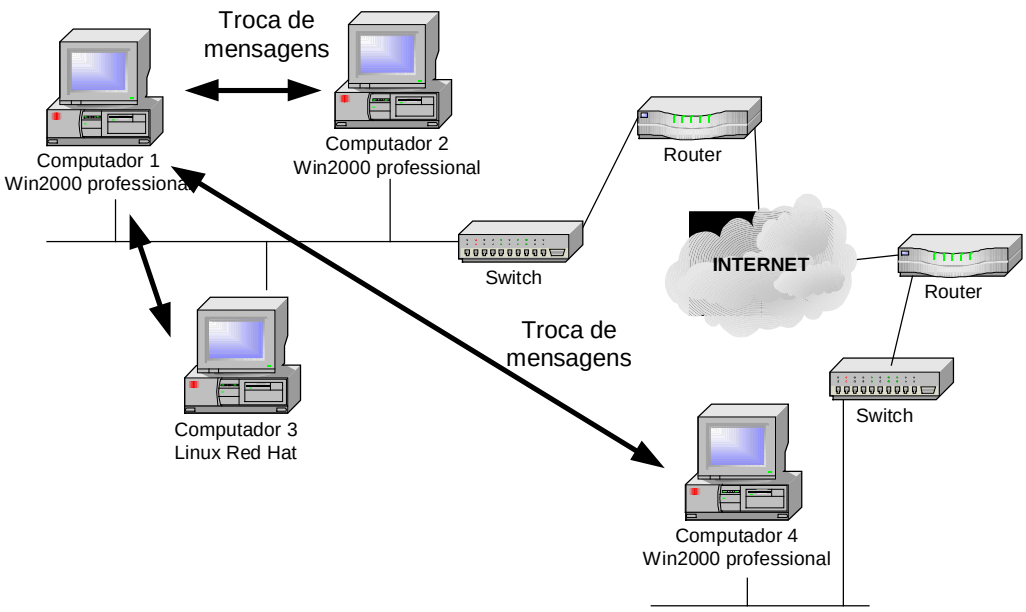

Figura 3.1 Ambiente de uso da aplicação

A aplicação construída é um USER AGENT SIP baseado nos requisitos mínimos [SCH99a]. Como existe a necessidade de utilizar plataformas de sistemas operacionais diferentes, foi escolhida a linguagem Java [JAVA]. Java possui ainda facilidades para tratar com pacotes UDP e TCP no uso de *threads*.

3.2 Arquitetura da aplicação

O funcionamento básico de uma aplicação VoIP usando SIP é mostrado na Figura 3.2:

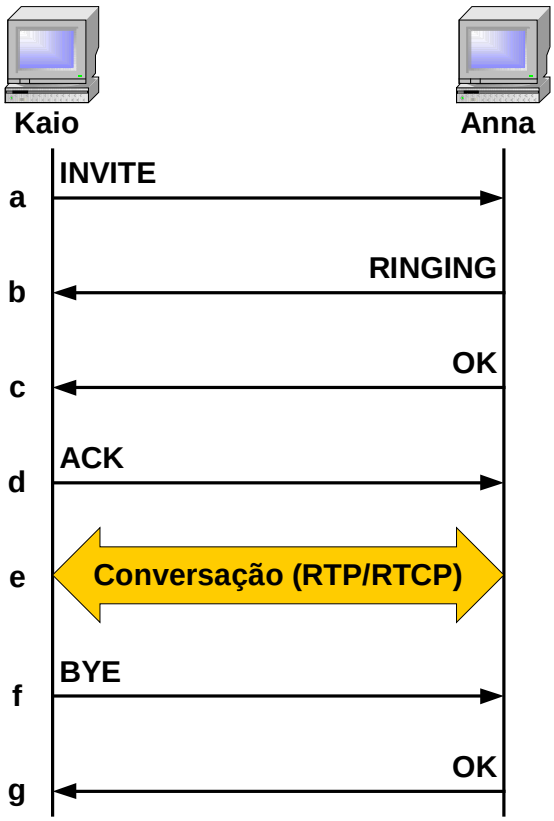

Figura 3.2 Chamada SIP

a. O processo inicia com uma mensagem SIP INVITE enviada pelo transmissor (Kaio) para o receptor (Anna) para estabelecimento de uma sessão.

b. Kaio recebe uma mensagem avisando que Anna está sendo alertada sobre a chamada (RINGING).

c. Anna responde a chamada gerando uma mensagem OK.

d. Kaio reconhece o OK enviando um ACK.

e. A partir deste momento, as mídias (voz, vídeo, etc) são trocadas através de RTP/RTCP.

f. Finalmente, uma das partes (no caso Kaio) finaliza a conversa, enviando de uma mensagem BYE.

g. Ao receber o BYE, Anna envia um OK para confirmar. Neste ponto, a chamada é finalizada.

Usando esta especificação, pode-se dividir a arquitetura da aplicação em três partes distintas:

a) Sinalização da chamada usando o protocolo SIP;
b) Digitalização de voz e envio/recebimento de pacotes RTP
c) Interface com o usuário

A aplicação foi modelada com o *Rational Rose 2000* e implementada usando o *JBuilder 9 Enterprise Edition*. O código fonte da aplicação está no apêndice desta dissertação.

3.2.1 Sinalização da chamada usando o protocolo SIP

A Figura 3.3 mostra a sinalização das chamadas usando SIP:

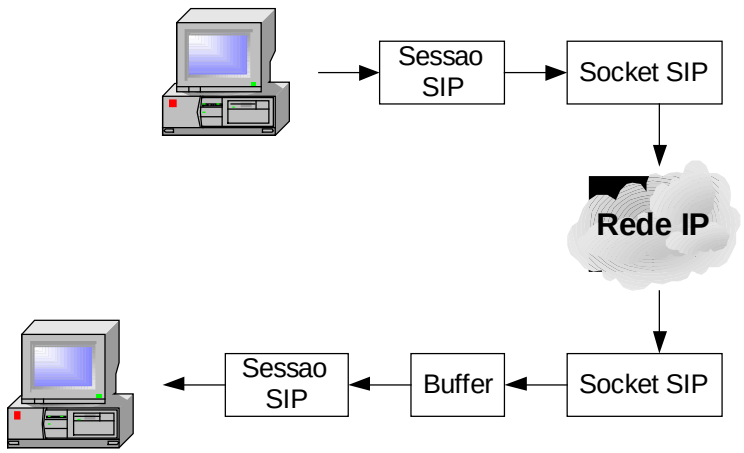

Figura 3.3 Sinalização usando SIP

a) As mensagens SIP são montadas dentro da Sessão SIP
b) São montadas em pacotes SIP e enviadas pela rede IP
c) No destinatário, são recebidas armazenadas em um *buffer*.
d) A Sessão SIP retira as mensagens do *buffer*, as interpreta e realiza as operações solicitadas.

O problema foi especificado usando as classes:

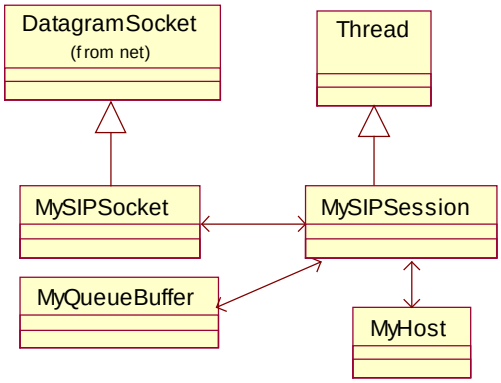

Figura 3.4 Diagrama de Classes SIP

Classe	Descrição
Datagram Socket	Classe Java padrão que implementa o protocolo UDP
Thread	Classe Java padrão que implementa *threads*.
MySIPSocket.java	Classe que recebe e envia mensagens SIP. Foi criada a partir da Datagram Socket para enviar mensagens UDP [WIL01]. Ela possui uma interface com a classe padrão *Thread* para trabalhar em "*background*" esperando os pacotes SIP chegarem.
MySIPSession.Java	Classe que controla uma chamada SIP e monta as mensagens. Ela é uma classe herdada de *Thread* para ficar verificando continuamente o buffer a espera de novas mensagens SIP. Esta classe envia e recebe mensagens do tipo INVITE, ACK, BYE, OK, RINGING e CANCEL
MyQueueBuffer	Classe que um implementa uma fila como buffer para mensagens UDP.
MyHost	Classe encarregada de armazenar os dados das máquinas envolvidas na troca de mensagens. Os dados são endereço IP, porta usada para trocar mensagens SIP e mensagens RTP, tipo de *payload* e outros.

Tabela 3-1 Classes SIP

3.2.2 Digitalização de voz e envio/recebimento de pacotes RTP

A figura 3.5 mostra a arquitetura do módulo de envio e recebimento de pacotes RTP contendo amostras de voz:

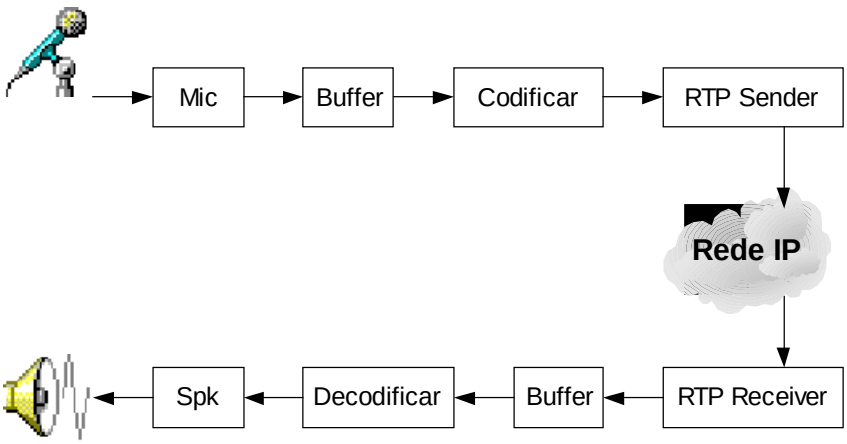

Figura 3.5 Digitalização de voz e envio/recebimento de pacotes RTP

a) A voz recebida pelo microfone é codificada e armazenada em um buffer.
b) A voz codificada é dividida em pacotes RTP e enviada pela rede IP para um destinatário.
c) Ao chegar ao destino, os pacotes RTP recebidos são armazenados em um buffer para evitar perda.
d) Os pacotes são então decodificados e passados para a caixa de som.

O processo todo é iniciado pelo recebimento de mensagens SIP INVITE no módulo de sinalização e é finalizado pelo recebimento de uma mensagem SIP BYE ou SIP CANCEL.

Um dos objetivos da aplicação é verificar a perda de pacotes de voz. Essa verificação é feita através da transmissão de um arquivo de voz pré-gravado (Arquivo de Voz 1) pela aplicação. No destinatário, os pacotes recebidos são montados e gravados em disco (Arquivo de Voz 2). As diferenças encontradas entre os arquivos mostram a perda entre os arquivos.

O problema foi especificado usando as seguintes classes:

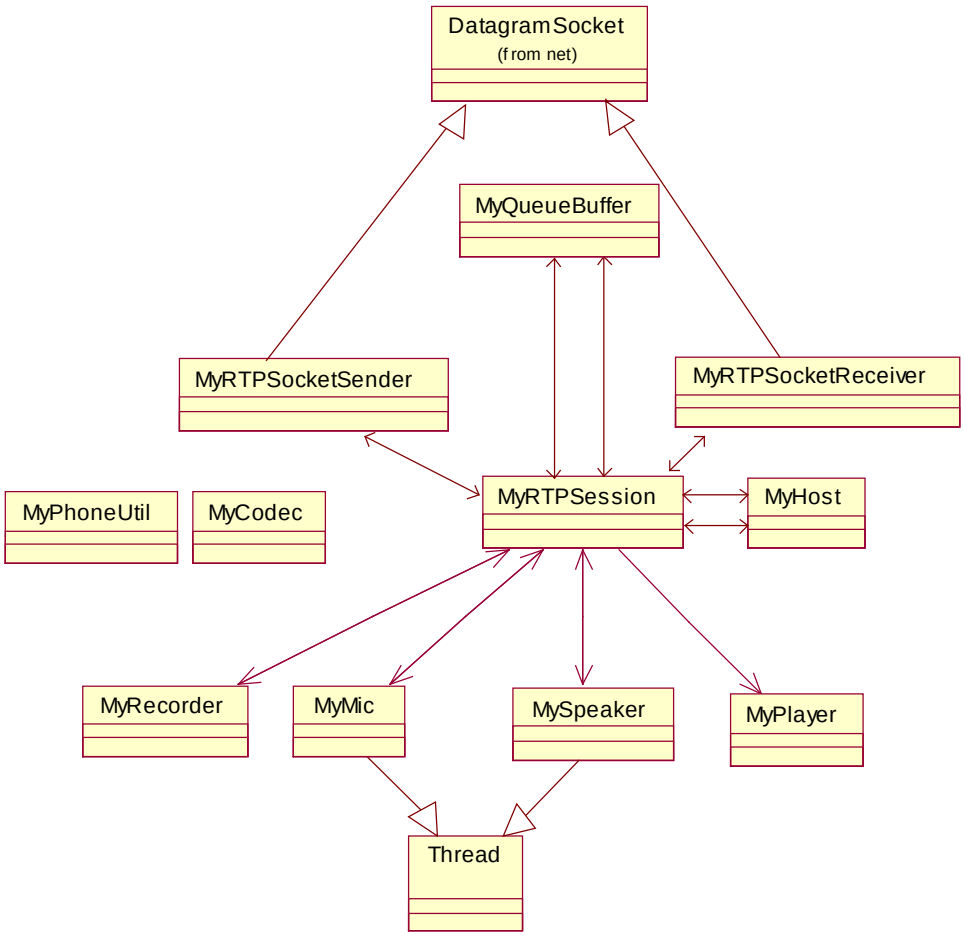

Figura 3.6 Classes RTP e de digitalização de voz

Classe	**Descrição**
Datagram Socket	Classe Java padrão que codifica o protocolo UDP
Thread	Classe Java padrão que implementa a utilização de *threads*
MyQueueBuffer	Classe que um implementa uma fila como buffer para mensagens UDP. Existe um buffer para as mensagens que serão enviadas (buffer de envio) e outro para as mensagens recebidas (buffer de recebimento).
MyPhoneUtil	Classe que usada como biblioteca de constantes e funções de conversão binária de inteiros. Baseada nas funções encontradas em [JAVAa], [HIL].
MyCodec	Classe que converte PCM 16bits mono 8Hz em M-law e A-law. Os algoritmos de M-law w A-law foram encontrados em [FAQ] e [WOO],e convertidos para Java usando as técnicas encontradas em [DAV97] e [LIN00].
MyHost	Classe encarregada de armazenar os dados das máquinas envolvidas na troca de mensagens. Os dados são endereço IP, porta usada para trocar mensagens SIP e mensagens RTP, tipo de *payload* e outros.
MyRecorder	Classe que monta os pacotes de voz recebidos e os transforma em um arquivo .AU.
MyMic	Classe thread que captura os pacotes de voz vindos do microfone em formato PCM 16bits 8Hz Mono Big-endian, codifica-os em um formato pré-determinado os armazena no buffer de envio.
MySpeaker	Classe thread que retira os pacotes de voz codificados do buffer, transforma-os para o formato PCM 16bits 8Hz Mono Big-endian e os passa para a caixa de som.
MyPlayer	Classe que lê um arquivo .AU, transforma-os em pacotes de voz que serão posteriormente enviados.

MyRTPSocketSender	Classe que lê os pacotes do buffer de envio, encapsula-os dentro de um pacote RTP e os envia como um datagrama UDP.
MyRTPSocketReceiver	Classe que recebe os pacotes RTP, separar deles os pacotes de voz e coloca estes dentro do buffer de recebimento.
MyRTPSession	Classe que gerencia a troca de mídia, definindo os destinatários, portas RTP e controlando o recebimento e envio de pacotes.

Tabela 3-2 Classes RTP e de digitalização de voz

3.2.3 Interface com o usuário

A interface permite:

- Iniciar uma chamada com INVITE
- Ouvir o “telefone tocando” quando receber um RINGING
- Aceitar uma chamada, “atendendo o telefone”.
- Finalizar uma chamada com BYE ou CANCEL
- Visualizar o *log* das mensagens SIP trocadas
- Selecionar o tipo de *codec* que será usado nas chamadas
- Selecionar para enviar o arquivo .AU ao invés de trocar mensagens de voz.
- Selecionar para gravar em formato .AU as mensagens de voz recebidas

As classes envolvidas são:

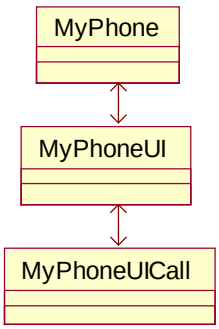

Figura 3.7 Classes de interface com o usuário

Classe	**Descrição**
MyPhoneUI	Tela principal
MyPhoneUICall	Tela onde o usuário digita o endereço do destinatário da chamada. Quando o botão ok é pressionado a chamada é iniciada
MyPhone	Classe de controle que faz o intermédio entre a interface e os outros módulos.

As figuras abaixo mostram as telas implementadas:

Figura 3.8 `MyPhoneUI.Java`

Figura 3.9 MyPhobeUI.java mostrando como selecionar o codificador

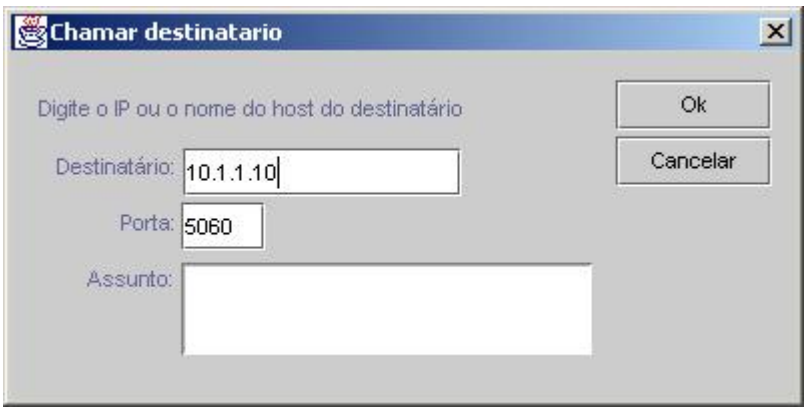

Figura 3.10 `MyPhoneUICall.java`

3.2.4 Modelo de classes

O modelo geral das classes é mostrado na Figura 3.8:

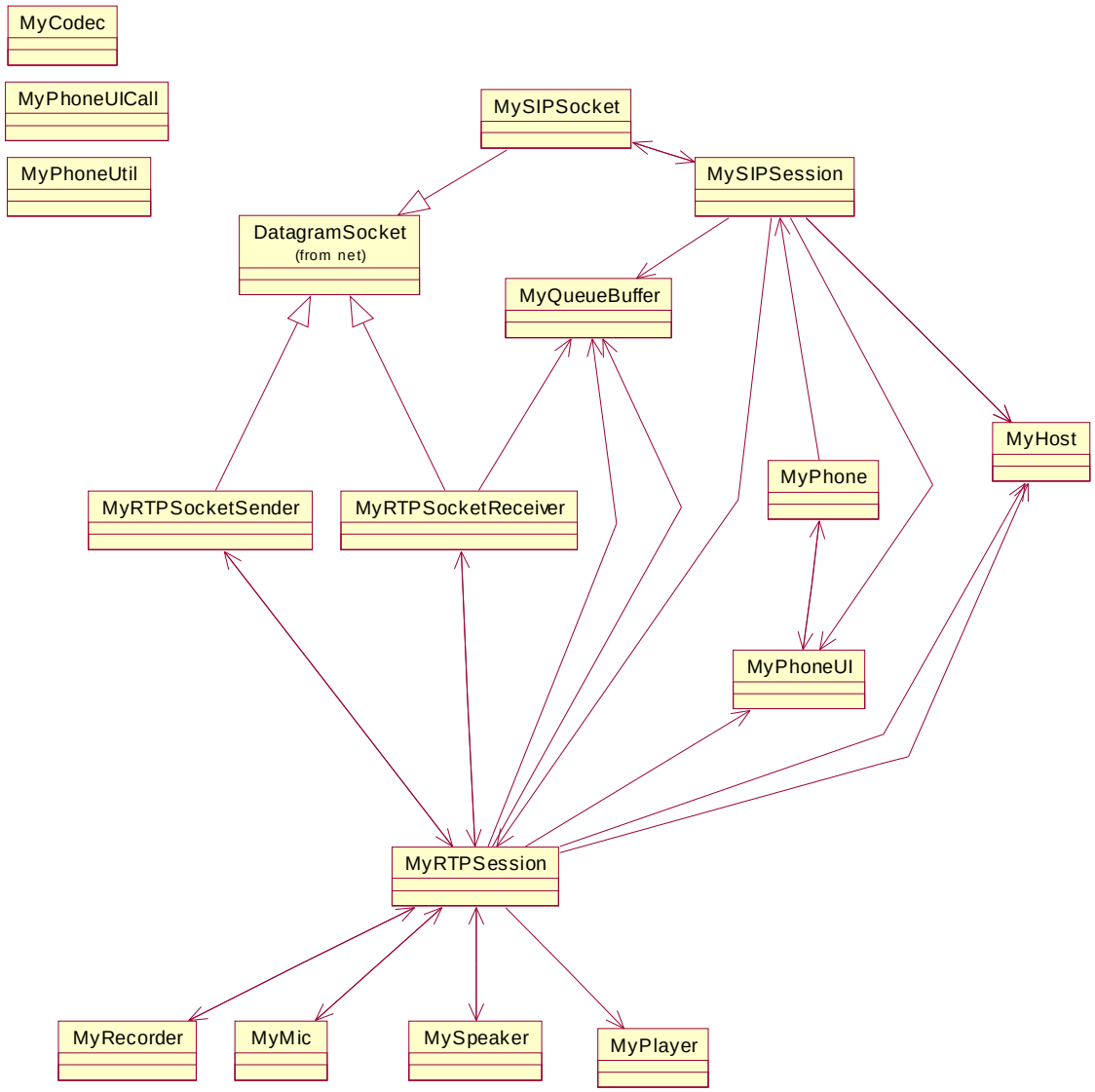

Figura 3.11 Modelo de classes

3.2.5 Funcionamento da aplicação

Apresentaremos, a seguir detalhes do funcionamento da aplicação, explicandp como são enviadas e recebidas mensagens SIP e iniciando e finalizando uma sessão RTP.

3.2.5.1 Enviando uma mensagem SIP

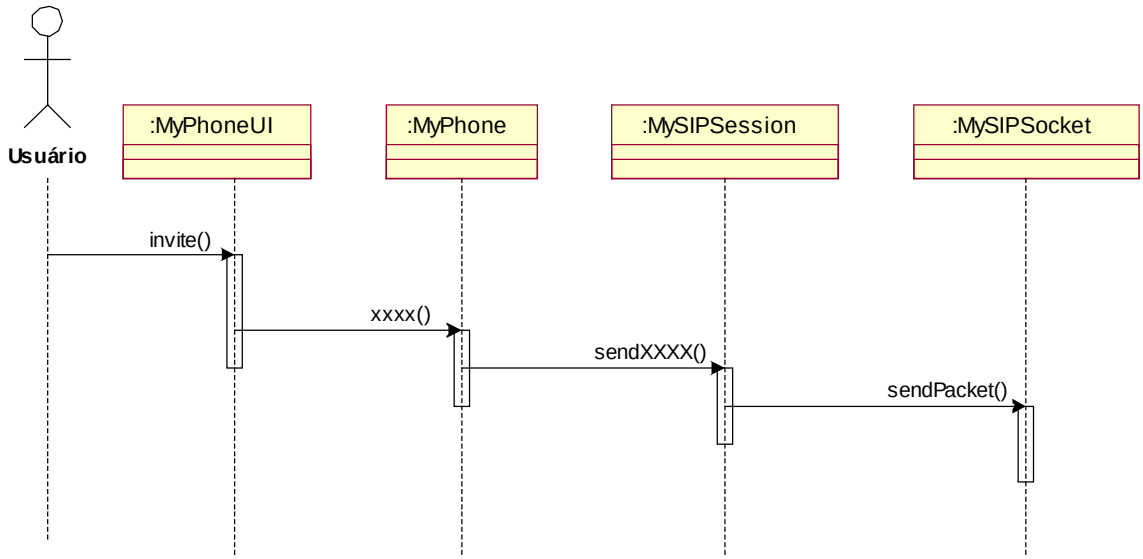

Figura 3.12 Enviando uma solicitação SIP

Na interface, de posse do endereço do destinatário, o usuário escolhe a solicitação (INVITE, BYE ou CANCEL). Com esse dados MySIPSession monta a mensagem SIP correspondente a solicitação repassa para MySIPSocket envia-la via UDP.

3.2.5.2 Recebendo uma mensagem SIP

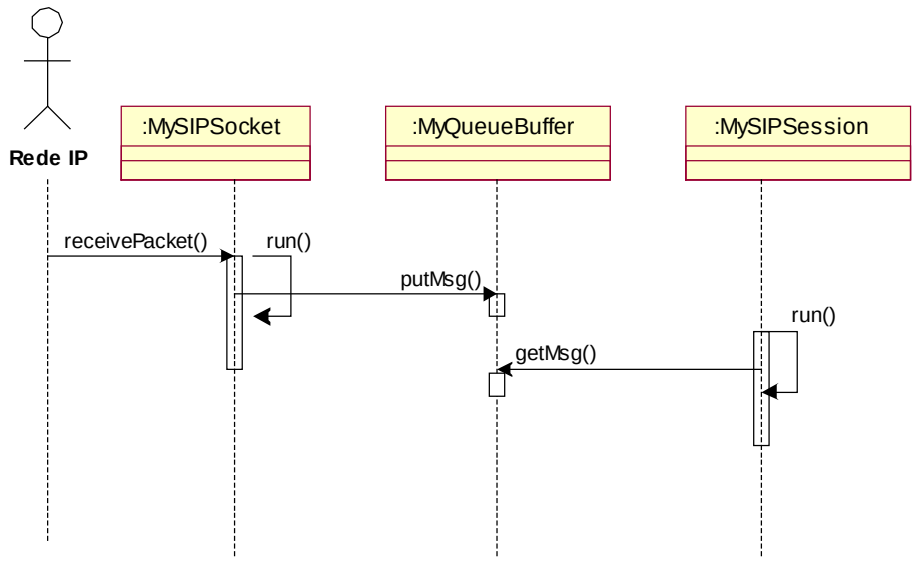

Figura 3.13 Recebendo uma solicitação

MySIPSocket fica “ouvindo” a porta selecionada a espera de mensagens. Quando uma mensagem chega, ela é colocada dentro do buffer de recebimento através do método *putMsg()*.A thread SIPSession retira a mensagem da fila (*getMsg()*)e a processa. Esse processamento pode ser abrir/finalizar uma sessão RTP.

3.2.5.3 Iniciando uma sessão RTP

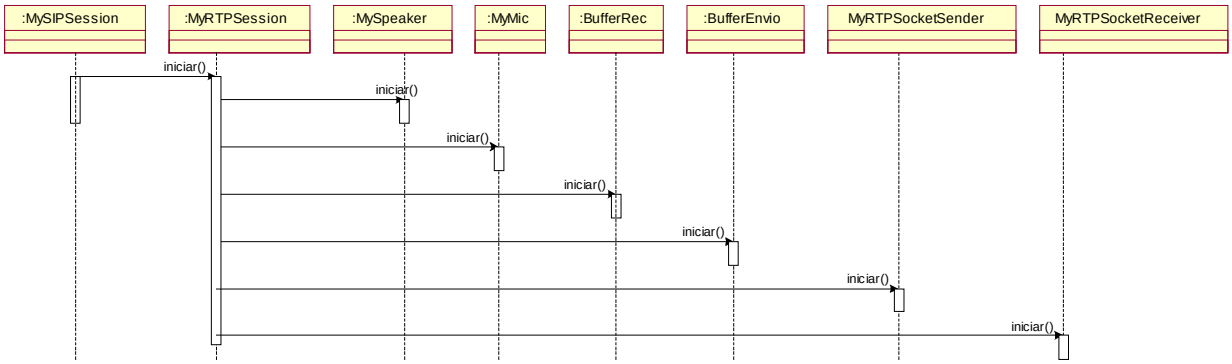

Figura 3.14 Iniciar sessão RTP

Após concluído o processo de sinalização para estabelecimento da chamada, MySIPSession inicia a sessão RTP (MyRTPSession), passando para ela os parâmetros definidos no corpo SDP das mensagens SIP. MyRTPSession inicia então os buffers de recebimento e envio (BufferRec e BufferEnvio respectivamente) e as threads MyMic para capturar a voz via microfone e MySpeaker para enviar os pacotes de voz recebidos para a caixa de som. Se for escolhida a opção de gravar em arquivo, MyRecorder será ativado ao invés do MySpeaker. Se for escolhida a opção de envio de arquivo .AU, MyPlayer será ativado ao invés de MyMic. Os sockets de envio e recebimentos (MyRTPSocketSender e MyRTPSocketReceiver) são iniciados.

3.2.5.4 Finalizando uma sessão RTP

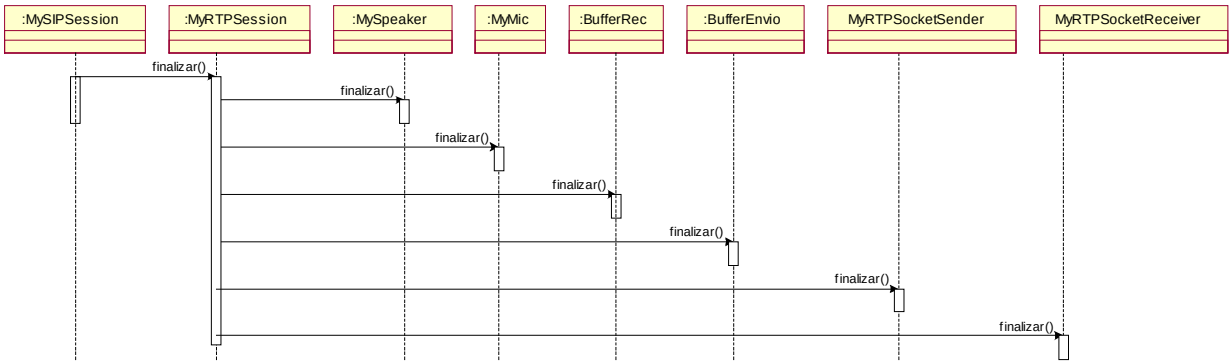

Figura 3.15 Finalizar sessão RTP

Após concluído o processo de sinalização para finalização da chamada, MySIPSession termina a sessão RTP (MyRTPSession) e esta termina todos os objetos ligados a ela.

4 Análise de Desempenho

Este capítulo apresenta resultados de um estudo experimental da utilização da aplicação VoIP desenvolvida em relação a diferentes topologias de rede e diferentes sistemas operacionais. O objetivo do experimento é verificar a utilização do protocolo SIP e também de alguns fatores que afetam a qualidade de voz como o tempo de atraso na chegada dos pacotes e o *jitter*.

4.1 Topologia do experimento

O estudo consistiu na definição de um ambiente de teste que simulasse a utilização real da aplicação. A Figura 4.1 mostra o ambiente de teste:

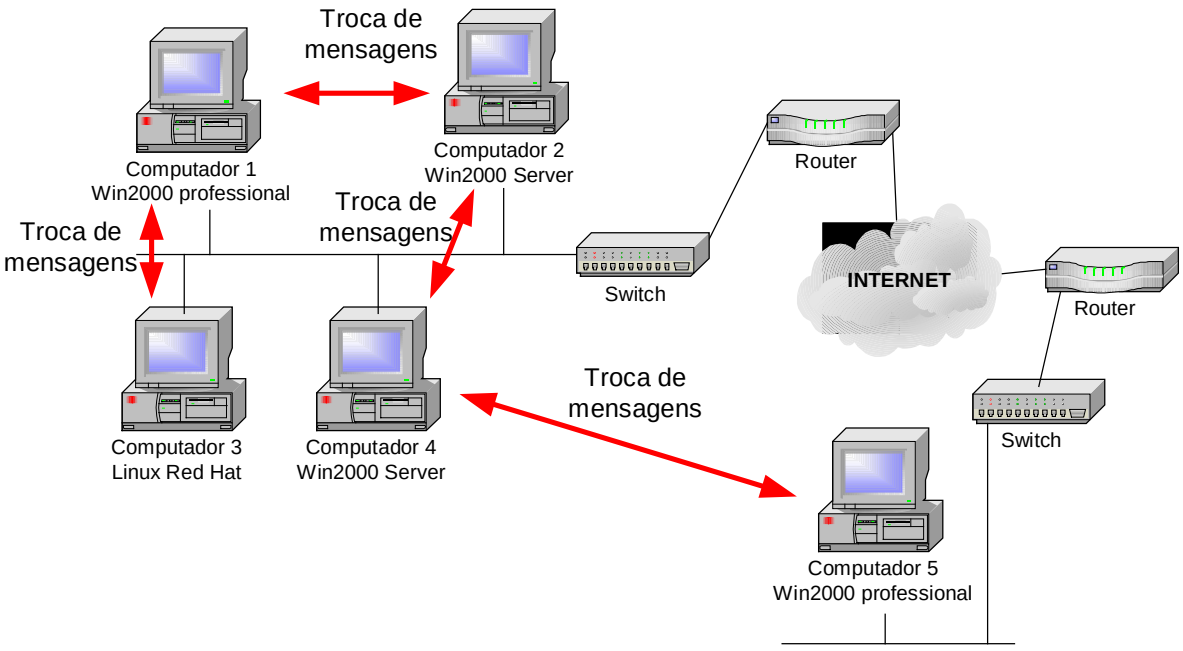

Figura 4.1 Ambiente experimental

Para esse ambiente foram definidos os seguintes cenários:

Cenário	**Descrição do cenário**	**Experimento**	**Descrição**
Cenário1 Rede Local	Rede Local Ethernet de 10Mbit/s com micros ligados por hubs Ethernet 10/100Mbit/s	Experimento 1	Somente tráfego de voz
		Experimento 2	Tráfego de voz com tráfego aleatório de dados
		Experimento 3	Tráfego de voz com trafego de dados contínuo e uniforme
		Experimento 4	Tráfego de voz com trafego de dados a 30% da capacidade da rede
Cenário 2 Internet	Link de modem 33.600bps	Experimento 5	Tráfego de voz com tráfego aleatório de dados
		Experimento 6	Tráfego de voz com tráfego contínuo e uniforme

Tabela 4-1 Cenários experimentais

Os horários dos computadores são sincronizados para facilitar a captura de dados sobre o atraso na chegada de pacotes. O protocolo SIP não é afetado por essa sincronização. Em todos os experimentos foi usada seqüência SIP descrita na Figura 4.2:

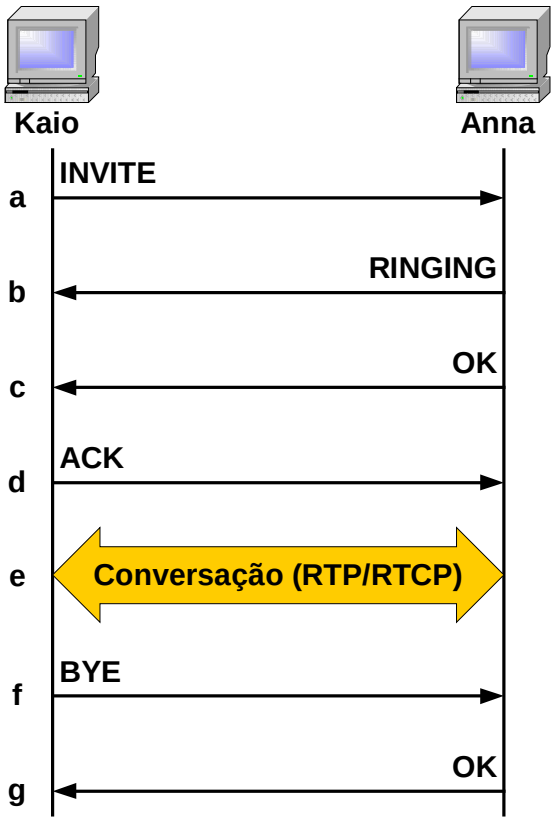

Figura 4.2 Seqüência SIP usada nos experimentos

4.2 Coletando dados de atraso, perda de pacotes e *jitter*

Para facilitar a coleta de dados sobre atrasos e perdas de pacotes e *jitter*, foi adotada a estratégia de enviar um arquivo pré-gravado de voz `VOZ_FEMININA.AU` (Figura 4.3) com a ferramenta *Sound Fourge 5*b no formato PCM 16Bits 8MHz. Esse é o formato de som gerado pela aplicação antes dele ser codificado para *m-Law* ou *A-law*. A Figura 4.4 mostra mais propriedades do arquivo.

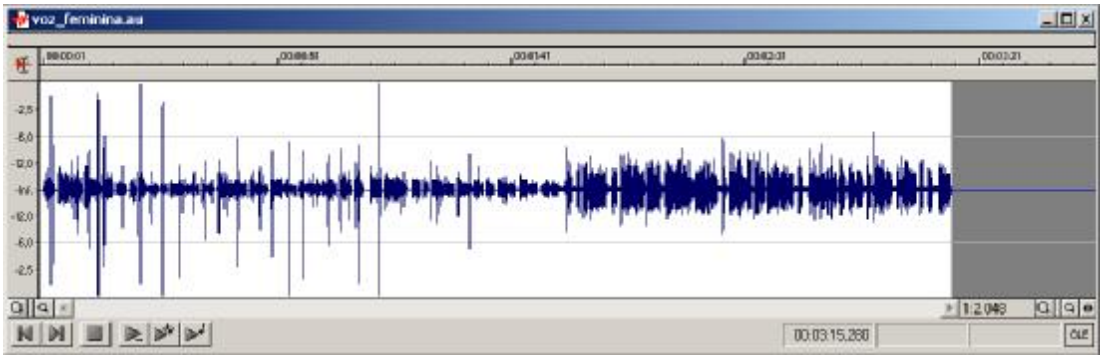

Figura 4.3 Voz feminina.au

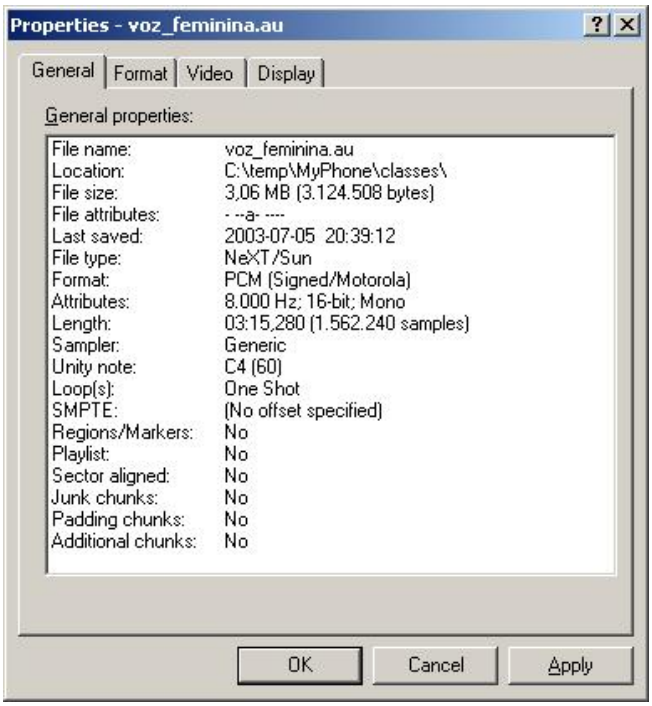

Figura 4.4 Propriedades do arquivo voz_feminina.au

Este arquivo possui 3min e 15,280s de voz gravada, o que corresponde a 195280ms de voz.. O arquivo é dividido em amostras de 320bytes e codificado para G.711 m-Law de 8bits e 8KHz, gerando amostras de 160bytes (que correspondem a 20ms de voz) em um total de 9764 pacotes. Esses pacotes são enviados segundo o processo descrito na Figura 4.4. Cada amostra é encapsulada dentro de um pacote RTP. Para fins de geração de log, foi acrescentado ao cabeçalho RTP o campo DTHR_ENVIO (8bytes) para armazenar a data e hora envio do pacote. Cada pacote RTP fica então com 180bytes. Os pacotes são enviados pela rede IP do computador REMETENTE ao computador DESTINATÁRIO. No computador DESTINATÁRIO é gerado o log de cada pacote que chega. Esse log é formado com os dados do cabeçalho RTP, o campo de DTHR_ENVIO e também com a data e hora de chegada. Os dados desse log são usados para gerar os dados para análise.

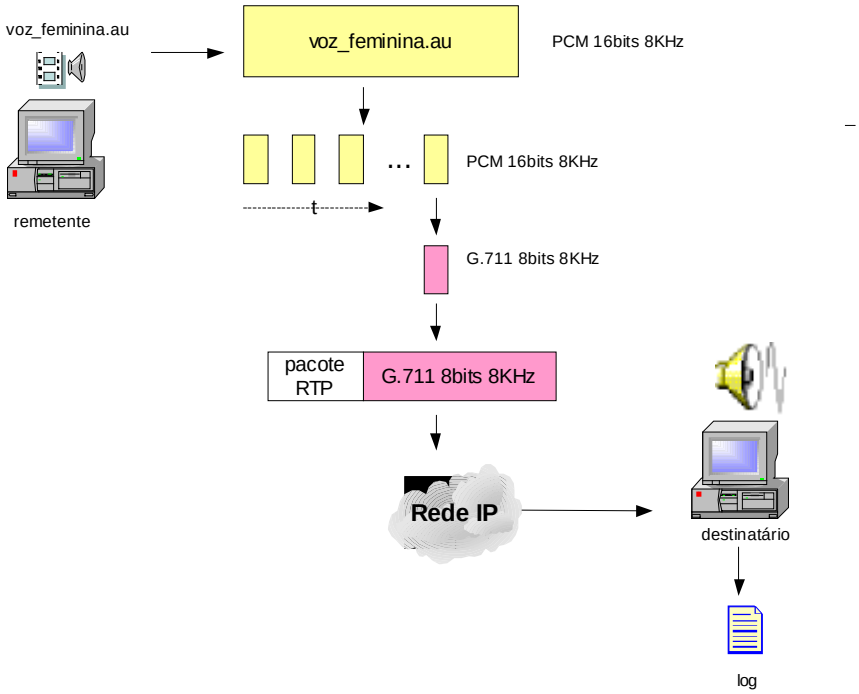

Figura 4.5 Processo do experimento

4.3 Experimento 1

Neste experimento existia somente o tráfego de voz na rede. O resultado do experimento mostrou que não houve perda de pacotes, pois os 9674 pacotes RTP chegaram ao destino. A troca de mensagens SIP funcionou perfeitamente não havendo qualquer perda de pacotes. Foi detectado um atraso ocorrido na chegada de um pacote mostrado pela descontinuidade na Figura 4.6. A variação do tempo de chegadas inter-pacotes (*jitter*) mostrado na Figura 4.7 mostra que quase todos pacotes chegaram dentro de 0 a 15ms. Isso indica um *jitter* muito pequeno.

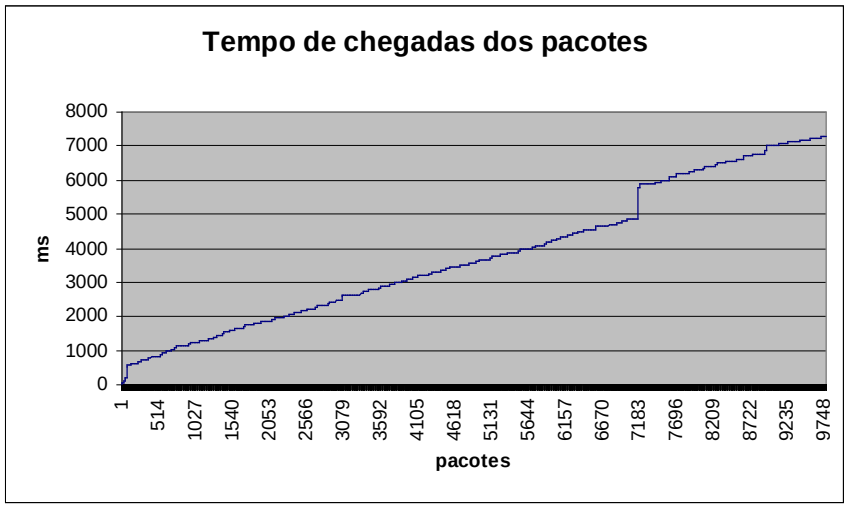

Figura 4.6 Experimento 1 - Tempo de chegada dos pacotes

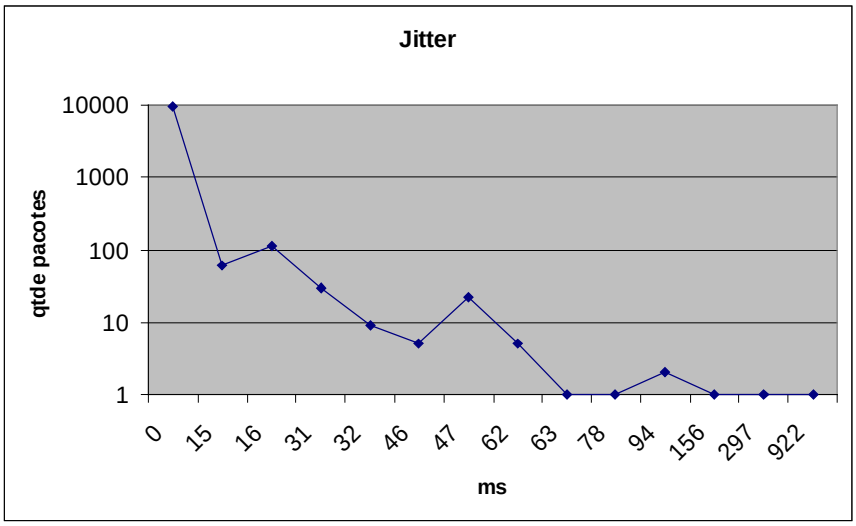

Figura 4.7 Experimento1 - Jitter

4.4 Experimento 2

Neste experimento foram utilizados surtos aleatórios de dados para verificar sua interferência na qualidade de voz. O resultado do experimento mostrou que não houve perda de pacotes de som ou SIP. Foi detectado um atraso anômalo ocorrido na chegada de alguns pacotes mostrados pelas descontinuidades no gráfico da Figura 4.8. Os surtos de dados não influenciaram a qualidade de voz, pois os pacotes chegaram em tempo hábil e com *jitter* insignificante.

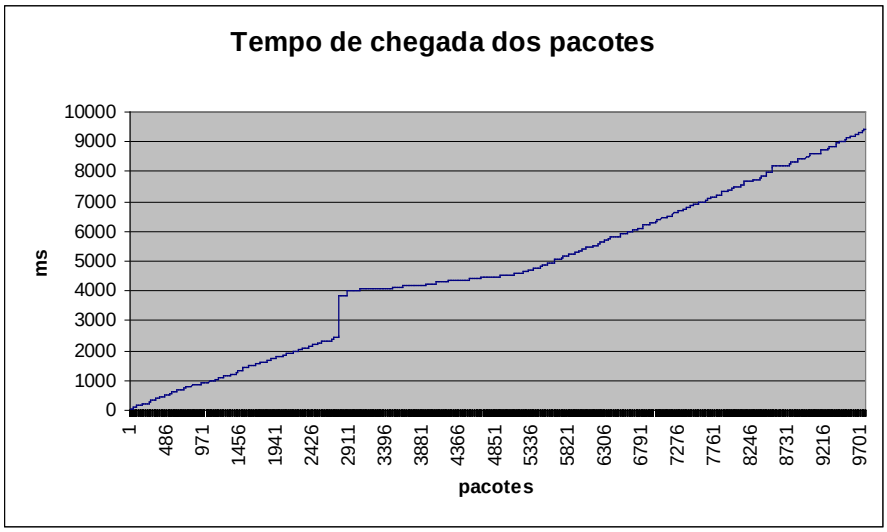

Figura 4.8 Experimento 2 - Tempo de chegada de pacotes

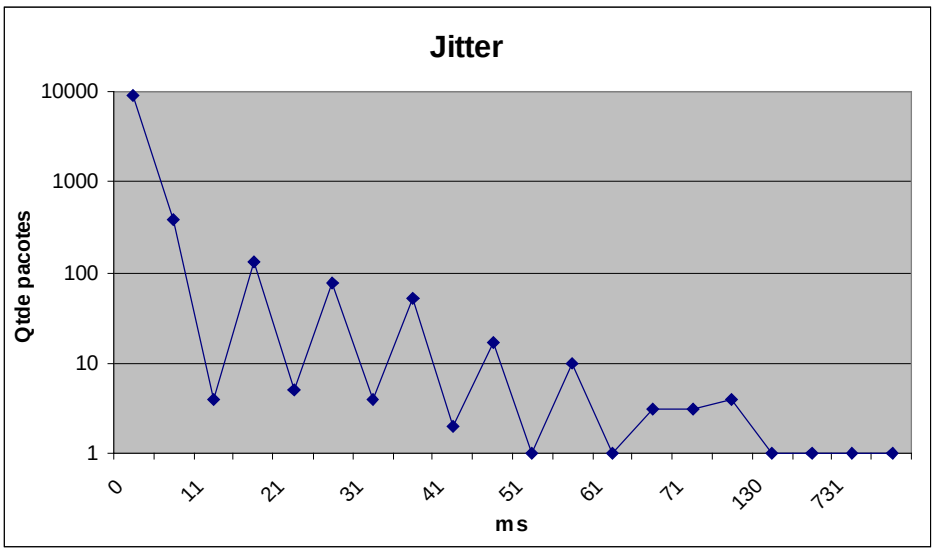

Figura 4.9 Experimento 2 – Jitter

4.5 Experimento 3

Para este experimento foi usando um fluxo contínuo de dados (um outro computador fazia FTP). Neste experimento também não houve perda de pacotes tanto SIP quanto RTP. Novamente foi detectado um atraso perceptível no meio da conversação mostrado tanto pelo gráfico de tempo de chegada (Figura 4.10). Esse atraso causou um "corte" na conversação. Houve um aumento no *jitter*, pois o tempo de chegada inter-pacotes passou a variar como mostra a Figura 4.11. Ainda assim a qualidade da conversação não foi afetada.

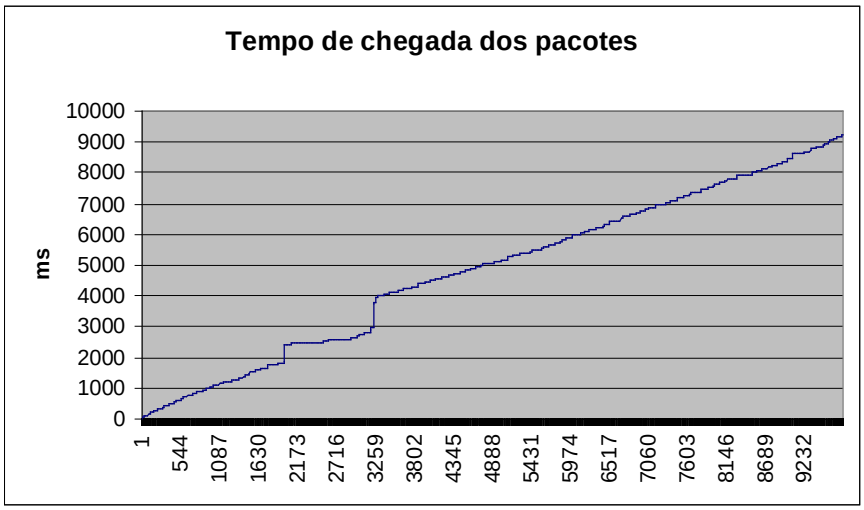

Figura 4.10 Experimento 3 - Tempo de chagada dos pacotes

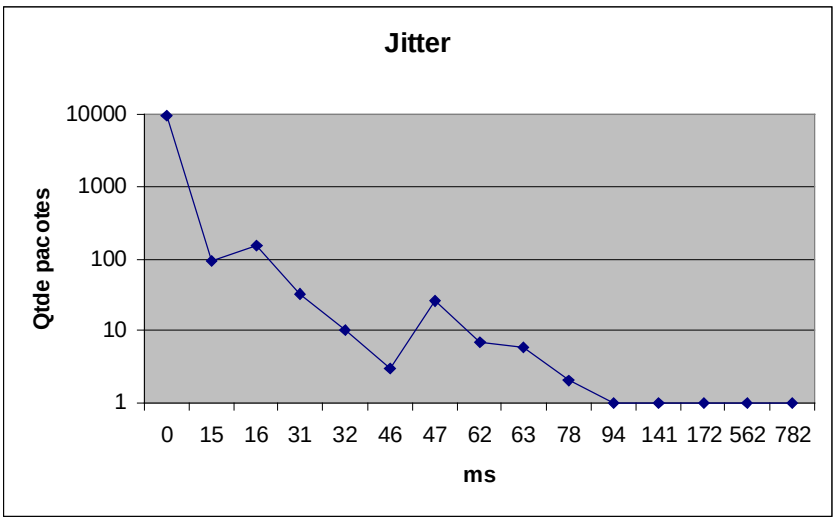

Figura 4.11 Experimento 3 - Jitter

4.6 Experimento 4

Neste experimento a carga de utilização da rede foi elevada para 30% através do uso de ferramentas e aplicativos, simulando um tráfego pesado de dados. O tempo de chegada dos pacotes foi aumentando e ainda houve uma anomalia. Isso pode ser observado na Figura 4.12. O jitter foi pequeno (Figura 4.13) e insuficiente para afetar a qualidade de voz. Não houve perdas de pacotes RTP ou SIP.

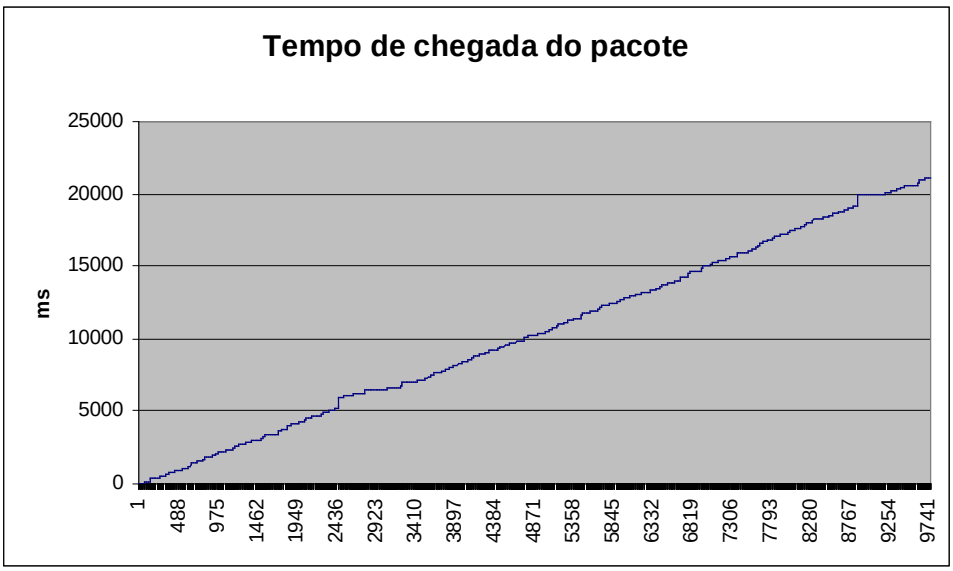

Figura 4.12 Experimento 4 - Tempo de chagada dos pacotes

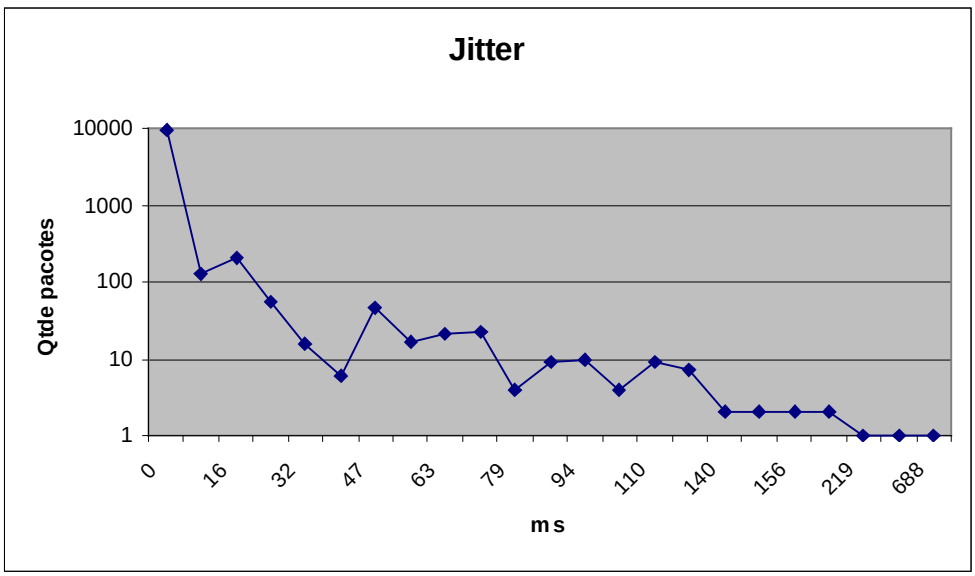

Figura 4.13 Experimento 4 - Jitter

4.7 Experimento 5

Este experimento simula a utilização da aplicação na Internet apenas fazendo a conversação. O computador DESTINATÁRIO está ligado a Internet através de um link de 2Mbits e o REMETENTE através de um link de modem de 33.600Kbps. Os pacotes foram chegando ao destino de maneira linear (Figura 4.14), mas com um atraso considerável, o que afetou muito a qualidade de voz. A conversação ficou muito entrecortada. Além disso, o envio demorou muito mais que o tempo de voz gravado (aproximadamente 3min e 15s).

Figura 4.14 Experimento 5 - Tempo de Chegada dos pacotes

Os pacotes eram recebidos em surtos de 10 em 10 unidades. Entre esses 10 pacotes, o próximo chegava com um atraso grande (aproximadamente 2s). O gráfico de tempo de chegada de inter-pacotes (Figura 4.15), mostra que houve muito variação (*jitter*). Quase metade dos pacotes chegou com tempo superior a 300ms, atraso intolerável para uma conversação via Internet.

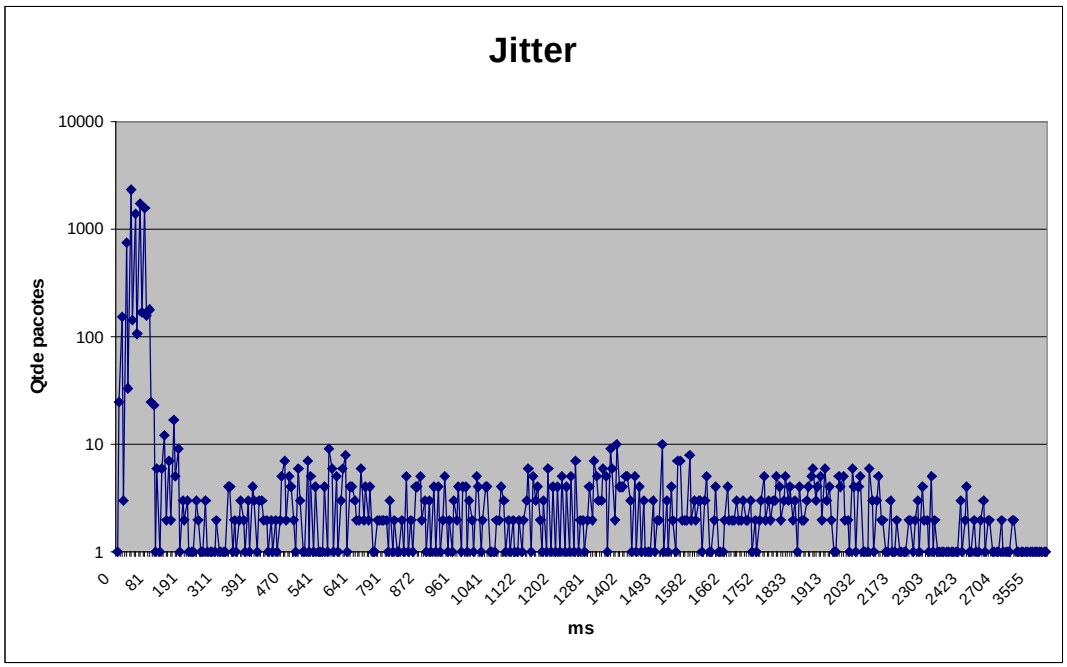

Figura 4.15 Experimento 5 - Jitter

4.8 Experimento 6

Este experimento simula a utilização da aplicação na Internet fazendo a conversação e trocando dados de maneira contínua (utilizando FTP). O computador DESTINATÁRIO está ligado a Internet através de um link de 2Mbits e o REMETENTE através de um link de modem de 33.600Kbps. O atraso foi muito grande e deixou a conversação impraticável, demorando mais de 15mim para terminar o que levaria 3min e 15s.

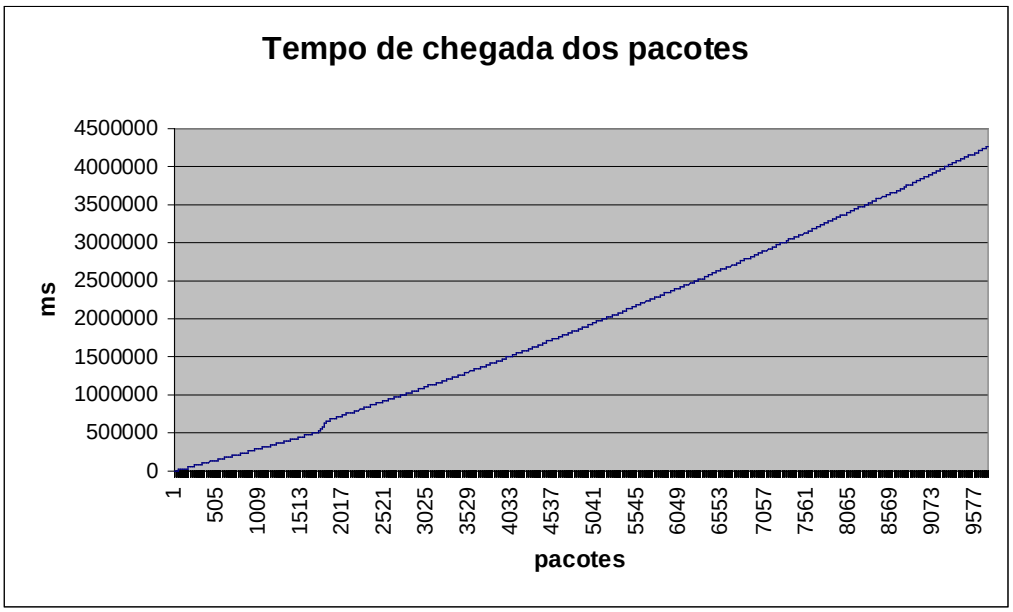

Figura 4.16 Experimento 6 - Tempo de chegada dos pacotes

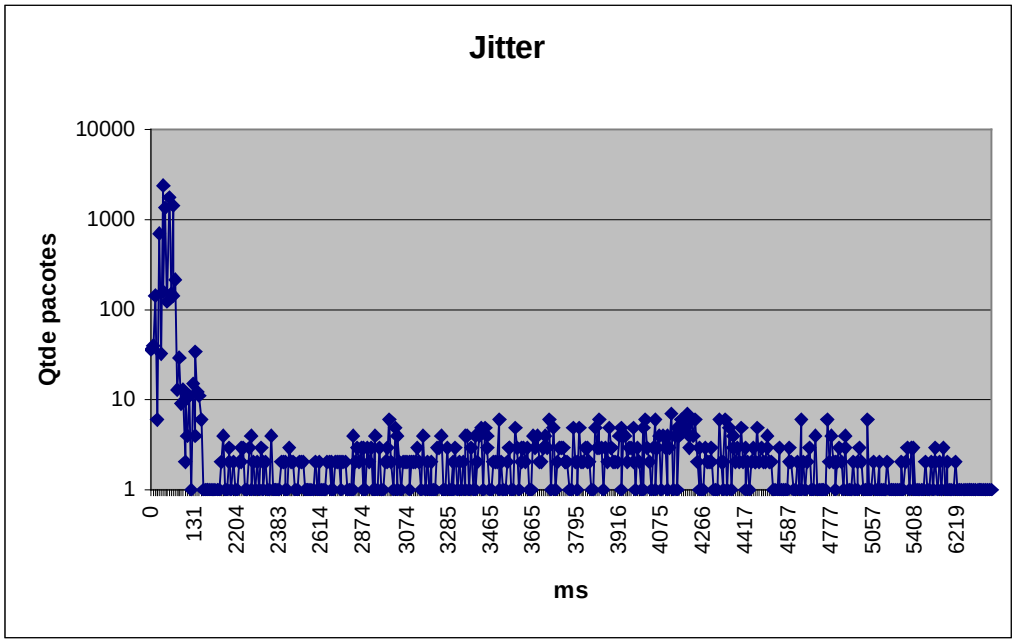

Figura 4.17 Experimento 6 – Jitter

O *jitter* também atrapalhou a conversação, pois a Figura 4.17 mostra que houve um salto no tempo de chegada inter-pacotes de 131ms para quase 2000ms. O resto dos pacotes chegou com muita variação.

4.9 Comparação dos experimentos do cenário 1

A Figura 4.18 mostra que quando não há tráfego na rede os pacotes chegam muito mais rápidos. Os experimentos 1 a 3 mostram tempos de chegadas quase iguais variando apenas pela carga na rede. O experimento 4 mostra um tempo de chegada de quase o dobro dos demais. Mesmo assim, a variação de carga na rede, não afetou a qualidade da conversação.

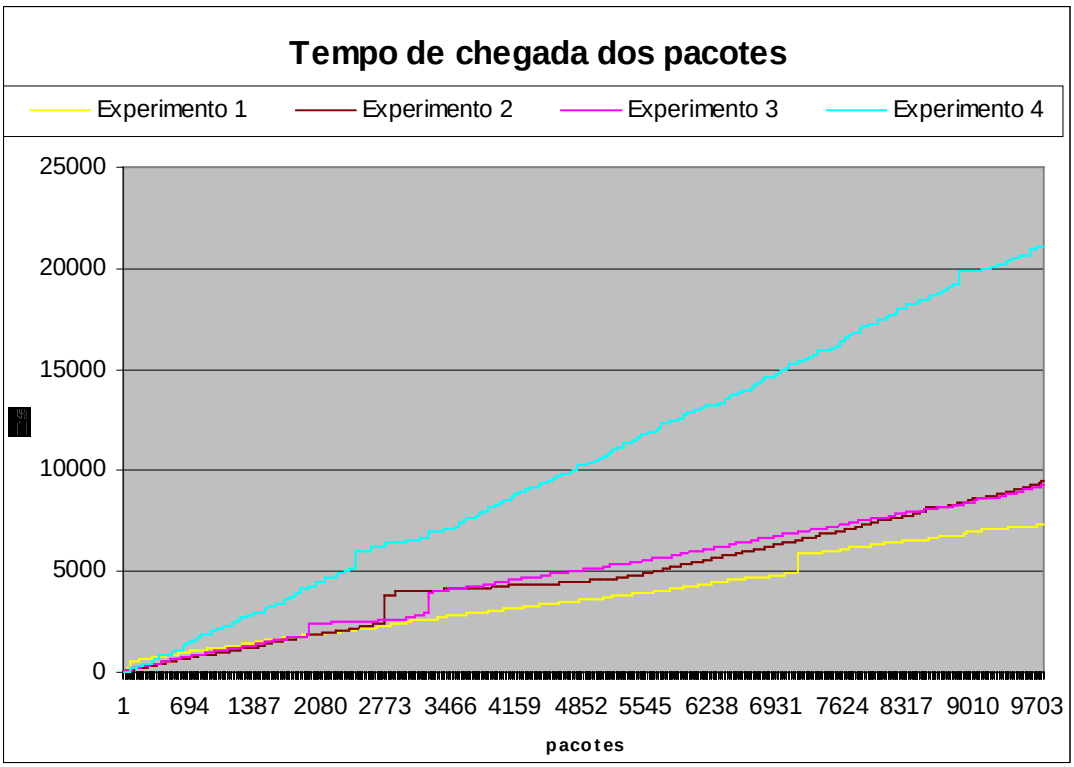

Figura 4.18 Comparação do tempo de chegados dos pacotes

O tempo de chegada inter-pacotes dos 4 experimentos (Figura 4.19) mostra que quase todos os pacotes chegaram com tempo inferior a 200ms, portanto não afetando a qualidade da conversação. Em todos os experimentos não houve perda de pacotes.

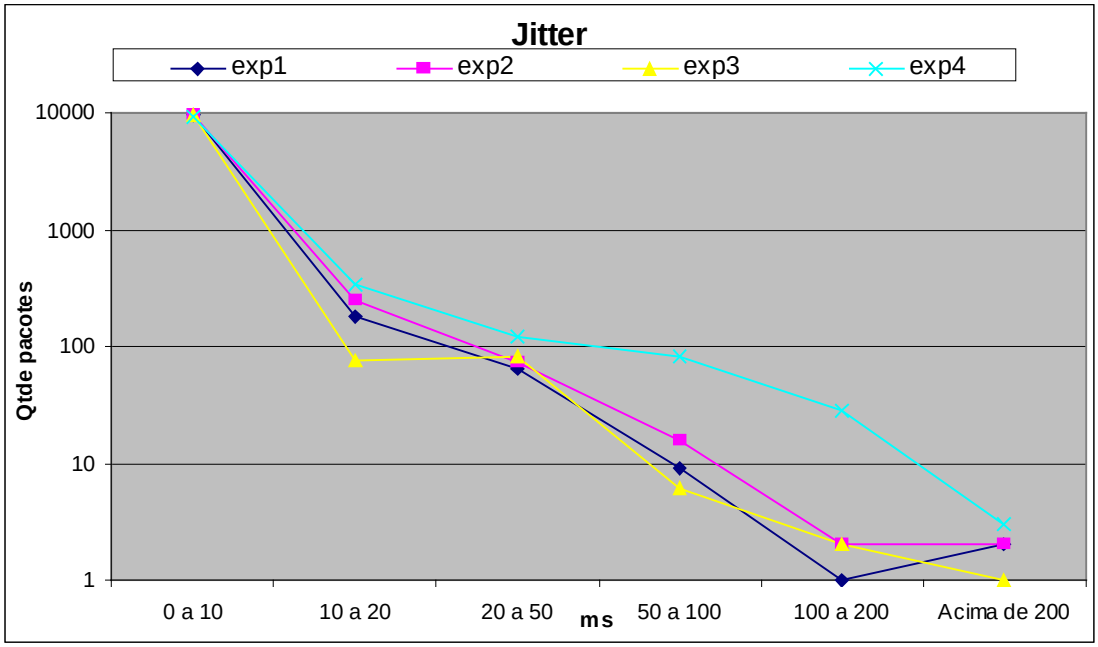

Figura 4.19 Comparação do jitter entre os experimentos

Nos quatro experimentos a seqüência de mensagens SIP ocorreu com sucesso. Não houve atrasos na entrega de pacotes SIP/UDP. Em relação aos pacotes RTP, em todos os experimentos apareceram atrasos em determinados pontos da conversação. Este atraso ocorria devido a uma falha na sincronização do buffer com o socket de recebimento,. Essa falha já foi consertada no código-fonte constante no anexo. Mesmo com essa falta de sincronização, o atraso dos pacotes foi mínimo e dentro dos padrões aceitáveis para Telefonia IP que é de até 300ms. Os experimentos mostram que é perfeitamente viável a utilização da aplicação dentro de uma rede local.

4.10 Comparação dos experimentos do cenário 2

O experimento 6 alerta que quando utilizado juntamente com troca de dados, o atraso torna a comunicação de voz inviável pois o tempo de chegada foi mais que o dobro do experimento 5 (Figura 4.20). Esse atraso tornou uma conversa de 3min e 15s em outra não inteligível de mais de 15 minutos.

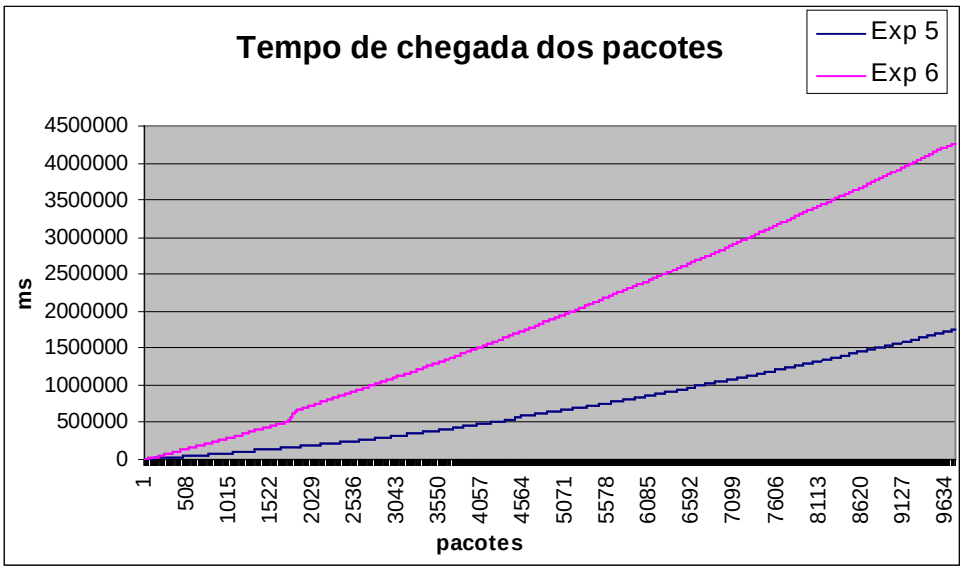

Figura 4.20 Comparação entre os tempos de chegadas dos experimento 5 e 6

A comparação entre os *jitter* dos experimentos (Figura 4.21) mostra valores parecidos até os 2000ms. A partir dai, há uma grande variação (foi o ponto de pico na transferência FTP no experimento 6).

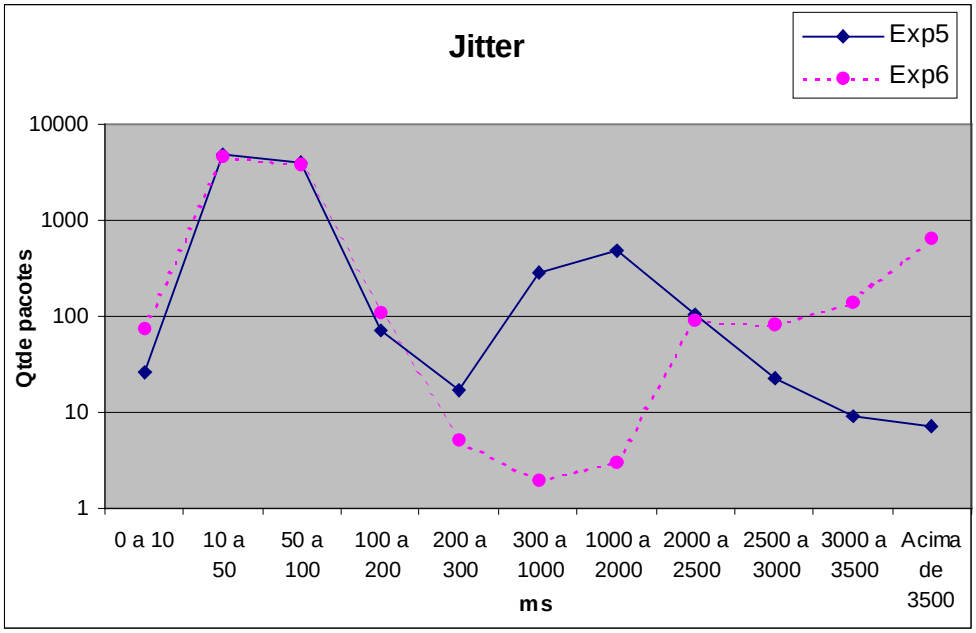

Figura 4.21 Comparação do jitter entre os experimentos 5 e 6

Nos dois experimentos a seqüência de mensagens SIP ocorreu com sucesso. Não houve atrasos na entrega de pacotes SIP/UDP. Em relação aos pacotes RTP, a perda de pacotes foi mínima (apenas um pacote no experimento 5 e onze pacotes no experimento 6). Isso comprova que a perda de pacotes abaixo de 5% não afeta a conversação. A qualidade de voz é afetada pelo atraso e jitter. Em ambos os experimentos, esses fatores deixaram inviáveis as conversações em uma conexão de 33.600Kbps usando o G.711 PCM m-law, pois este precisa de um fluxo constante de 64Kbps. É necessário utilizar codecs mais modernos como o G.729 ou G.723.1. Mesmo assim, foi comprovado que a aplicação consegue trocar dados de voz sem perda de pacotes..

5 Conclusão

Existem muitas coisas que devem ser melhoradas para que a Telefonia IP possa competir de igual para igual com a Telefonia convencional. Em 1999, a IETF publicou a RFC 2543 que definia o Session Initiation Protocol (SIP) como modelo de sinalização para Telefonia IP. Ele foi desenvolvido pensando em escalabilidade, reuso e interoperabilidade. Hoje, com algumas dezenas de implementações, ele está se tornando um protocolo maduro. As previsões indicam que a partir de 2004, haverá um grande crescimento na adoção da Telefonia IP. SIP provavelmente deverá ser o protocolo que facilitará o crescimento. Portanto, conhecer o protocolo e desenvolver aplicações com ele tem grande futuro.

Este trabalho mostra que um USER AGENT SIP pode ser implementado usando Java. Apesar da linguagem ser lenta para aplicações desktop, pode-se encontrar maneiras de otimizar o código ou mesmo redistribuir os serviços entre as classes. Uma grande dificuldade foi encontrar código-fonte dos codificadores mais modernos, visto que eles estão protegidos por patentes.

Existem muitos trabalhos futuros que podem utilizar/continuar esta aplicação:

- Implementar um SIP PROXY;
- Implementar novos codificadores de voz mais avançados (G.729, GSM e outros) para serem usados na aplicação ao invés do G.711;
- Alterar a aplicação para J2EE, criando um serviço de voz via web com inúmeras aplicações.
- Desenvolver o algoritmo adapitativo mostrado em [NOB01];

6 Referências Bibliográficas

[ALE98] ALENCAR, M. "*Telefonia Digital*", Editora Érica, 1998.

[BRA02] BRADNER, S., "*Internet Telephony: Progress Along the Road*", IEEE Internet Computing, http://computer.org/Internet/, Junho 2002

[CAM02] CAMARILLO, G.; "*SIP Demystified*", Mc Graw-Hill, 2002

[COL01] COLLINS, D.; "*Carrier Grade Voice Over IP*", Mc Graw-Hill, 2001

[COM98] COMER, D. E.; "*Interligação em Rede com TCP/IP Volume I*" , 5ª Tiragem da Tradução da Terceira Edição, Editora Campus, 1998

[CRO98] CROWCROFT, J. ; et al, "*Internetwork Multimedia*", UCL Press , 1998

[DAL98] DALGIC, I.; Fang, H.; "*A Comparison of H.323 and SIP for IP Telephony Signaling*", 1998

[DAV97] DAVIS, M., "*Porting C++ to Java*", http://review.software.ibm.com/developer/library/porting

[DET02] DETTMER, R,"*The convent phone*", IEE Review , Janeiro 2002

[FAQ] Frequently Asked Questions about DSP, USENET, www.google.com.br

[FIN02] FINEBERG, Victoria,"*A Practical Architecture for Implementing End-to-End QoS in an IP Network*", IEEE Communications Magazine, Janeiro 2002

[HAL00] HALADYNA, M., "*Voice over IP Protocols, Standards and Components*"**,** 2000

[HAN98] HANDLEY, M.; Jacobson, V.; "*SDP: Session Description Protocol*", RFC 2327, Abril 1998

[HAN99] HANDLEY, M.; Schulzrinne, H.; Scholler, E.; H., Rosemberg, J; "*SIP: Session Initiation Protocol*", RFC 2543, Março 1999

[HER02] HERSENT, O.; Guide, D.; Petit, J.; "*Telefonia IP*", Addison Wesley, 2002

[HIL] HILFINGER, P., "*Numbers*", Universidade da Califórnia

[ITU] Recomendations G.729, "*Coding of speech at 8kbits/s using conjugate structure-algebric code excited linear prediction*"

[ITUa] Recomendations G.722, "*Description of the digital test sequences for the verification of the G.722 64kbits/s SB-ADPCM 7KHz coded*"

[ITUb] Recomendations G.726, "*40, 32, 24, 16 kbits/s Adaptive Diffrencial Pulse Code Modulatin (ADPCM)*"

[ITUc] Recomendations G.728, "*Coding Speech at 16kbits/s using low-delay code excited linear prediction*"

[ITUd] Recomendations G.728, "*G.723.1 Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s*"

[JAN02] JANSEN, J.; et al, "*Assessing Voice Quality in Packet Based Telephony*", IEEE Internet Computing, Junho 2002

[JAVA] Java Home Page, http://java.sun.com

[JAVAa] Java Sound API Programmer's Guide, http://java.sun.com

[LEI99] LEINONEN, J.,"*Real-Time Voice over Packet-Switched Networks*", Seminar on Real-Time and Embedded Systems, Novembro 1999

[LIND00] LINDLEY, C.; "*Digital Audio with Java*", Prentice Hall, 2000

[MAC00] MICHAEL, B.; "*SIP Rules!";* Computer Telephony, 2000

[MEL002] MELVIN, Hugh; Murphy, L.; "*Time Synchronization for VoIP Quality of Service*", IEEE Internet Computing, http://computer.org/Internet/, Junho 2002

[MUR] MURHAMMER, M; et al, "*TCP/IP Tutorial and Technical Overview*", http://www.redbooks.ibm.com

[NOB01] NOBREGA, O; "*Um algoritmo adaptativo de Transmissão para serviços de Voz sobre Redes IP*", Dissertação de Mestrado, UFPE, 2001

[POL99] POLYZOIS, C; et al, "*From POTS to PANS: A commentary on evolution to Internet Telephony*", IEEE Network, Junho 1999

[ROS] ROSENBERG, J., "A Tutorial on SIP", www.dynamicsoft.com

[ROS00] ROSENBERG, J.; Shockley , R., "*Session Initiation Protocol (SIP): A key Component for Internet Telephony*", www.computertelephony.com, Junho 2000

[RTP] RTP Site http://www.cs.columbia.edu/~hgs/rtp/

[SCH00] SCHULZRINNE, H.; "*Session Initiation Protocol (SIP)",* Columbia

University, http://www.cs.columbia.edu/~hgs, 2000

[SCH02] SCHULZRINNE, H.; , Arabshian, K, "Providing Emergency Services in Internet Telephony", IEEE Internet Computing, http://computer.org/Internet/, Junho 2002

[SCH96] SCHULZRINNE, H. et al; "*RTP: A Transport Protocol for Real-Time Applications*", RFC 1889, Janeiro 1996

[SCH96a] SCHULZRINNE, H.; "*RTP Profile for Audio e Video Conferences with Minimal Control*", RFC 1890, Janeiro 1996

[SCH98] SCHULZRINNE, H., Rosemberg, J.; "*Internet Telephony: Architecture and Protocols an IETF Perspective*", Julho 1998

[SCH98a] SCHULZRINNE, H.; Rosemberg, J.; "*A Comparison of SIP and H.323 for Internet Telephony*", Network and Operating System Support for Digital Audio and Video (NOSSDAV), Cambridge, 1998

[SCH98b] SCHULZRINNE, H.; Rosenberg, J., "*Signaling for Internet Telephony*", Columbia University, 1998

[SCH99a] SCHULZRINNE, H.; "*Requirements for SIP Servers and User Agents (V2.0)*", Columbia University, 1999

[SIP] SIP home page http://www.cs.columbia.edu/~hgs/sip/

[SIS] SISALEM D.; Kuthan, J., "Understand SIP", http://www.fokus.gmd.de/mobis/siptutorial/

[SPA] SPANIAS, A., "*Speech Coding: A tutorial review*", Proceedings of IEEE Vol 82 Número 10, Outubro 1994

[STEEB] STEEB, W,"*Mathematical Tools in Signal Processing with C++ and Java Simulations*", International School for Scientific Computing

[TAN97] TANENBAUM, A.; "*Redes de Computadores*", 7ª Tiragem da Tradução da Terceira Edição, Editora Campus, 1997

[VAR02] VARSHNEY, Upkar et al; "*Voice Over IP*", Comunicatins of the ACM, Janeiro 2002.

[VOV] "*Voice Over IP Protocols: an overview*", www.vovida.org

[WIL01] WILLIAMS, A., "*Java 2 Network Protocols Black Book*", Coriolis, 2001

[WOO] WOODARD J., "*Speech Coding*", http://www-mobile.ecs.soton.ac.uk/speech_codecs/

7 Apêndice

7.1 Código-fonte

7.1.1 MyCodec.java

```
/**
 * MyCodec.java
 * @author Jucimar Jr
 * Usada para converter pcm para µ-law e A-law e vice-versa
 *
 * @todo verificar A-law
 * @todo implementar g.723.1
 * @todo implementar GGM 0610
 *
 */

public class MyCodec {

	/**
	 * converte de pcm16, para um determinado codec
	 */
	public static byte[] convertFromPCM16(byte[] pcmData, int tipoCodec) {

		switch (tipoCodec) {
			case MyRTPSession.ULAW:
				return pcm16ToUlaw(pcmData);
			case MyRTPSession.ALAW:
				return pcm16ToAlaw(pcmData);
			case MyRTPSession.PCM16:
				return pcmData;
			default:
				return pcmData;
		}
	}

	/**
	 * converte de um determinado codec pra pcm
	 */
	public static byte[] convertToPCM16(byte[] somCodificado, int tipoCodec) {

		switch (tipoCodec) {
			case MyRTPSession.ULAW:
				return ulawToPcm16(somCodificado);
			case MyRTPSession.ALAW:
				return alawToPcm16(somCodificado);
			case MyRTPSession.PCM16:
				return somCodificado;
			default:
				return somCodificado;
		}
	}

	/**
	 * converte de pcm16 para µ-law 8bits
	 * baseado no código em C disponível em
	 * http://svr-www.eng.cam.ac.uk/comp.speech/Section2/Q2.7.html
	 * em 10-jun-2002
	 */
	public static byte[] pcm16ToUlaw (byte[] pcmData) {

		boolean ZEROTRAP = true;
		short BIAS       = 0x84; // pronuncia vâias
		int CLIP         = 32635;

		int exp_lut[] =
```

```
          { 0,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3,
            4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
            5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
            5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
            6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
            6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
            6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
            6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7};

          int  sign;
          int  exponent;
          int  mantissa;
          int  sample;
          int  ulawByte;
          int  qtdeUlawByte = pcmData.length / 2;

          byte[] ulawData = new byte [qtdeUlawByte];

          for (int i = 0; i < pcmData.length ; i = i + 2) {

                    sample = ( pcmData[i] << 8 ) | ( pcmData[i+1] & 0xFF );

                    sign = (sample >> 8) & 0x80;           /* set aside the sign */
                    if (sign != 0) sample = -sample;                /* get magnitude */
                    if (sample > CLIP) sample = CLIP;               /* clip the magnitude */

                    /* Convert from 16 bit linear to ulaw. */
                    sample = sample + BIAS;
                    exponent = exp_lut[(sample >> 7) & 0xFF];
                    mantissa = (sample >> (exponent + 3)) & 0x0F;
                    ulawByte = ~(sign | (exponent << 4) | mantissa);
                    if (ZEROTRAP)
                              if (ulawByte == 0) ulawByte = 0x02;    /* optional CCITT trap */

                    ulawData[i/2] = (byte) ulawByte;

          } // if

          return ulawData;
}


/**
 * converte de µ-law 8bits para pcm16
 * baseado no código em C disponível em
 * http://svr-www.eng.cam.ac.uk/comp.speech/Section2/Q2.7.html
 * em 10-jun-2002
 */
public static byte[] ulawToPcm16 (byte[] ulawData) {

          int exp_lut[] = {0,132,396,924,1980,4092,8316,16764};
          int sign;
          int exponent;
          int mantissa;
          int sample;
          int ulawByte;
          int qtdePcmByte = ulawData.length * 2;

          byte[] pcmData = new byte [qtdePcmByte];

          for (int i = 0; i < ulawData.length; i++) {

                    ulawByte = (int) ulawData[i];
                    ulawByte = ~ulawByte;

                    sign = (ulawByte & 0x80);
                    exponent = (ulawByte >> 4) & 0x07;
                    mantissa = ulawByte & 0x0F;
                    sample = exp_lut[exponent] + (mantissa << (exponent + 3));
                    if (sign != 0) sample = -sample;

                    pcmData[2*i] = (byte) (sample >> 8); // +SIG
                    pcmData[2*i+1] = (byte) (sample & 0xFF);; //-SIG
          }

          return(pcmData);
}

/**
 * converte de pcm16 para A-law 8bits
 * baseado no código em C disponível em
 * g711.txt da Sun Microsystems
 */
public static byte[] pcm16ToAlaw (byte[] pcmData) {
          short seg_end[] = {0xFF, 0x1FF, 0x3FF, 0x7FF,
                                                  0xFFF, 0x1FFF, 0x3FFF, 0x7FFF};

          byte SEG_SHIFT  = 4;  /* Left shift for segment number. */
          byte QUANT_MASK = 0xF;  /* Quantization field mask. */
          byte mask;
          byte seg = 8;
          byte aval;
```

```
                short pcmSample;
                byte  alawByte;

                int  qtdeAlawByte = pcmData.length / 2;

                byte[] alawData = new byte [qtdeAlawByte];

                for (int i = 0; i < pcmData.length ; i = i + 2) {

                        // transforma bytes em short
                        pcmSample = (short) ( ( pcmData[i] << 8 ) | ( pcmData[i+1] & 0xFF ) );

                        if (pcmSample >= 0) {
                                mask = (byte) 0xD5;  /* sign (7th) bit = 1 */
                        } else {
                                mask = 0x55;  /* sign bit = 0 */
                                pcmSample = (byte) (-pcmSample - 8);
                        }

                        /* Convert the scaled magnitude to segment number. */
                        for (byte j = 0; j < 8; j++) {
                                if (pcmSample <= seg_end[j]) {
                                        seg = j;
                                        break;
                                }
                        }

                        if (seg >= 8)  /* out of range, return maximum value. */
                                alawByte = (byte) (0x7F ^ mask);
                        else {
                                aval = (byte) (seg << SEG_SHIFT);
                                if (seg < 2)
                                        aval |= (pcmSample >> 4) & QUANT_MASK;
                                else
                                        aval |= (pcmSample >> (seg + 3)) & QUANT_MASK;
                                        alawByte = (byte) (aval ^ mask);
                        }

                        alawData[i/2] = alawByte;
                }

                return alawData;
        }


        /**
         * converte de A-law 8bits para pcm16
         * baseado no código em C disponível em
         * g711.txt da Sun Microsystems
         */
        public static byte[] alawToPcm16(byte[] alawData) {

                short t;
                short seg;
                short alawByte;
                short pcmSample;
                byte  SEG_SHIFT  = 4;  /* Left shift for segment number. */
                byte  QUANT_MASK = 0xF;  /* Quantization field mask. */
                byte  SEG_MASK   = 0x70;  /* Segment field mask. */
                short SIGN_BIT   = 0x80;

                int qtdePcmByte = alawData.length * 2;

                byte[] pcmData = new byte [qtdePcmByte];

                for (int i = 0; i < alawData.length; i++) {

                        alawByte = (short) alawData[i];

                        alawByte ^= 0x55;

                        t = (short) ((alawByte & QUANT_MASK) << 4);
                        seg = (short) ((alawByte & SEG_MASK) >> SEG_SHIFT);
                        switch (seg) {
                                case 0:
                                        t += 8;
                                        break;
                                case 1:
                                        t += 0x108;
                                        break;
                                default:
                                        t += 0x108;
                                        t <<= seg - 1;
                        }

                        pcmSample = (short) ( (alawByte & SIGN_BIT) == 0 ? t : -t);

                        pcmData[2*i] = (byte) (pcmSample >> 8); // +SIG
                        pcmData[2*i+1] = (byte) (pcmSample & 0xFF);; //-SIG
                }

                return pcmData;
        }
}
```

7.1.2 MyHost.Java

```
import java.net.*;

public class MyHost {

          public String      userName;
          public String      hostName;
          public InetAddress hostAddress;
          public int         sipPort;
          public int         rtpPort;
          public int         rtpPayloadType;
          public int         rtpLineBufferSize;
          public int         rtpPacketSize;
          public String      sipUrl;

          public MyHost( String usuario, int portaSip , int portaRtp, int codec, boolean local ) {

                    try {

                              userName       = usuario;
                              sipPort        = portaSip;

                              rtpPayloadType = codec;

                              if (local) {
                                        hostName    = InetAddress.getLocalHost().getHostName();
                                        hostAddress = InetAddress.getLocalHost();
                                        // gera o nr da porta RTP. sempre tem q ser par
                                        rtpPort        = 50000 + ( (int) ( Math.random() * 10000 ) );
                                        if ((rtpPort % 2) != 0) rtpPort++;

                              } else {
                                        hostName    = InetAddress.getByName(usuario).getHostName();
                                        hostAddress = InetAddress.getByName(usuario);
                                        rtpPort = 0;
                              }

                              userName = hostName;

                              sipUrl = userName + "@" + hostName;

                              if (sipPort != 5060) sipUrl += ":" + sipPort;

                    } catch (Exception e) {}
          }

          public String print(String titulo) {
                    return
                    "******************\n" +
                    titulo.toUpperCase() +
                    "\n******************\n" +
                    "sipUrl: " +  sipUrl + "\n" +
                    "userName: " + userName + "\n" +
                    "hostName: " + hostName + "\n" +
                    "sipPort: "  + sipPort + "\n" +
                    "rtpPort: "  + rtpPort + "\n" +
                    "rtpPayloadType: " + rtpPayloadType + "\n" +
                    "rtpLineBufferSize: "  + rtpLineBufferSize  + "\n" +
                    "rtpPacketSize: "  + rtpPacketSize  + "\n";
          }
}
```

7.1.3 MyMic.Java

```
/**
 * MyMic.java
 *
 * Controla a captura de som a partir do microfone
 *
 * @author Jucimar Jr
 *
 */

import    javax.sound.sampled.*;

public class MyMic extends Thread {

          private TargetDataLine line;
          private MyRTPSession   rtpSession;
          private boolean        isRunning = true;

          /**
           * usa a sessao rtp e o
           * codificador que será utilizado
           */

          public MyMic(MyRTPSession rtp) throws Exception {
                    FloatControl          gain;
                    AudioFormat               audioFormat;
                    AudioFileFormat.Type  targetType;
                    AudioInputStream      audioInputStream;

                    rtpSession = rtp;

                    // define o formato como PCM 16bits 8KHz Mono Big-endian
                    // e captura dados de audio
                    audioFormat = new AudioFormat(8000.0F, 16, 1, true, true);

                    DataLine.Info info =
                              new DataLine.Info(
                                        TargetDataLine.class,
                                        audioFormat
                              );

                    line = (TargetDataLine) AudioSystem.getLine(info);
                    line.open(audioFormat);
                    audioInputStream = new AudioInputStream(line);

                    // define o volume do microfone pro maximo possível
                    // deixando o usuário baixar pela cx de som do micro
                    // na maioria dos casos nao funciona pois nao dah pra aumentar o volume
                    // do microfone ..mas quem sabe em algum consiga.. :-)
                    if (line.isControlSupported(FloatControl.Type.MASTER_GAIN)) {
                       gain = (FloatControl) line.getControl(FloatControl.Type.MASTER_GAIN);
                       gain.setValue(gain.getMaximum());
                    }
          }

          /**
           * Inicia a linha e a thread
           */
          public void start() {
                    line.start();
                    super.start();
          }

          /**
           * fecha a linha e a thread
           */
          public void close() {
                    line.stop();
                    line.flush();
                    line.close();

                    //esse boolean eh pra fazer sair do run() da thread
                    isRunning = false;
          }

          /**
           * o som capturado pelo microfone eh colocado na fila de msgs para ser
           * capturado pelo MyRTPSocketSender e enviado ao destinatario
           */
          public void run() {

                    try {
                              // define o buffer de captura de som de acordo com o codificador
                              byte[] soundPacket =
                                        new byte[rtpSession.rtpDestinatario.rtpLineBufferSize];

                              while (isRunning) {

                                        // le um "pedaço" do som e preenche soundPacket
                                        rtpSession.setFeedback(5,"MICING");
                                        line.read(soundPacket, 0, soundPacket.length );
                                        rtpSession.setFeedback(6,"MICING");

                                        // ja entra na fila codificado o som codificado
                                        // se for u-law ou A-law pega um array de 320 e
                                        // transforma em um de 160
                                        rtpSession.queueSender.putMsg(
```

```
                                        MyCodec.convertFromPCM16(
                                                soundPacket,
                                                rtpSession.rtpDestinatario.rtpPayloadType
                                        )
                                );
                        }
                } catch (Exception e) {
                        rtpSession.setFeedback(666,"MyMic.run():\n" + e);
                }
        }

}
```

7.1.4 MyPhone.Java

```
/**
 * MyPhone.java
 *
 * @author Jucimar Jr
 *
 * Classe de interface entre a tela e a sessao sip
 *
 */

public class MyPhone {

          MySIPSession  sipSession;
          MyPhoneUI     phoneUI;

          public MyPhone(MyPhoneUI ui) {
                    phoneUI = ui;
          }

          /**
           * inicia a sessao sip
           */
          public String init (int sipPort, int rtpPort, int codec ) throws Exception {
                    MyHost remetente = new MyHost("",sipPort,rtpPort,codec,true);
                    sipSession = new MySIPSession(phoneUI, remetente);
                    return remetente.sipUrl;
          }

          /**
           * convida um destinatario para uma conversaçao
           * fazendo uma chamada
           */
          public void invite(String username, int sipPort) throws Exception {
                    MyHost destinatario = new MyHost(username,sipPort,0,0, false);
                    sipSession.setDestinatario(destinatario);
                    sipSession.sendINVITE();
          }

          /**
           * envia uma msg ok, qdo o remetente atende o telefone
           */
          public void ok() {
                    sipSession.sendOK();
          }

          /**
           * envia uma msg bye qdo o remetente desliga uma chamada
           */
          public void bye() {
                    sipSession.sendBYE();
          }

          /**
           * envia uma msg cancel qdo o remetente cancela uma transaçao
           */
          public void cancel() {
                    sipSession.sendCANCEL();
          }

          /**
           * altera o codec do remetente pra sempre ficar de acordo com o que foi
           * escolhido na UI
           */
          public void setRemeteteCodec( int codec ) {
                    sipSession.setPayloadType(codec);
          }

          /**
           * retorna o destinatario da mensagem
           */
          public MyHost getDestinatario() {
                    return sipSession.getDestinatario();
          }

}
```

7.1.5 MyPhoneUI.java

```
import java.awt.*;
import javax.swing.*;
import java.io.*;
import javax.sound.sampled.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.border.*;

public class MyPhoneUI extends JFrame {//implements Runnable {


          MyPhone    phone;

          public static final String VERSAO = "MyPhone 0.2002.06.14";
          private static final int FORM_LARGURA = 250;
          private static final int FORM_ALTURA  = 260;
          private static final int FORM_X  = 100;
          private static final int FORM_Y  = 100;


          Clip som = null;

          int feedbackTipo = 999;
          String feedbackMsg = "";
          boolean isRinging = false;

          JTextField txtPortaSIP = new JTextField();
          JLabel jLabel4 = new JLabel();
          ButtonGroup buttonGroup1 = new ButtonGroup();
          JCheckBox chkMic = new JCheckBox();
          JCheckBox chkSpeaker = new JCheckBox();
          JPanel panTelefone = new JPanel();
          JLabel jLabel11 = new JLabel();
          JButton btnAtender = new JButton();
          JButton btnDesligar = new JButton();
          JLabel lblSpeaker = new JLabel();
          JLabel lblMic = new JLabel();
          JButton btnChamar = new JButton();
          JButton btnIniciar = new JButton();
          JPanel panConfiguracao = new JPanel();
          JTabbedPane jTabbedPane1 = new JTabbedPane();
          JCheckBox chkPlayer = new JCheckBox();
          JComboBox arquivoCombo = new JComboBox();
          JCheckBox chkRecorder = new JCheckBox();
          JComboBox cmbCodec = new JComboBox();
          JLabel jLabel13 = new JLabel();
          JScrollPane pan1 = new JScrollPane();
          Border border1;
          Border border2;
          JTextArea lblFeedback = new JTextArea();
          JScrollPane jScrollPane1 = new JScrollPane();
          JTextArea txtLog = new JTextArea();



          public MyPhoneUI() {
                    try {
                              jbInit();
                    }
                    catch(Exception e) {
                              e.printStackTrace();
                    }
          }
          public static void main(String[] args) {

                    try {

                              MyPhoneUI phoneUI = new MyPhoneUI();

                              phoneUI.setSize(FORM_LARGURA,FORM_ALTURA);
                              phoneUI.setLocation(FORM_X,FORM_Y);
                              phoneUI.setResizable(false);
                              phoneUI.cmbCodec.addItem("µ-Law 8-bits 8KHz");
                              phoneUI.cmbCodec.addItem("PCM 16-bits 8KHz");
                              phoneUI.cmbCodec.addItem("A-Law 8-bits 8KHz");
                              phoneUI.cmbCodec.addItem("GSM Full Rate");

                              phoneUI.arquivoCombo.addItem("voz_feminina.au");
                              phoneUI.arquivoCombo.addItem("voz_feminina2.au");
                              phoneUI.arquivoCombo.addItem("voz_masculina.au");
                              phoneUI.arquivoCombo.addItem("voz_masculina2.au");

                              phoneUI.setFeedback(-1,"");

                              phoneUI.show();

                              phoneUI.setTitle(VERSAO);

                    } catch (Exception e) {e.toString();}



          }
          private void jbInit() throws Exception {
```

```
                    border1 = BorderFactory.createBevelBorder(BevelBorder.LOWERED,Color.white,Color.white,new Color(148, 145,
140),new Color(103, 101, 98));
                    border2 = BorderFactory.createBevelBorder(BevelBorder.LOWERED,Color.white,Color.white,new Color(134, 134,
134),new Color(93, 93, 93));
                    this.getContentPane().setLayout(null);

                    txtPortaSIP.setFont(new java.awt.Font("SansSerif", 0, 11));
                    txtPortaSIP.setText("5060");
                    txtPortaSIP.setBounds(new Rectangle(77, 9, 46, 20));
                    jLabel4.setFont(new java.awt.Font("SansSerif", 0, 11));
                    jLabel4.setForeground(Color.white);
                    jLabel4.setHorizontalAlignment(SwingConstants.RIGHT);
                    jLabel4.setText("Minha Porta:");
                    jLabel4.setBounds(new Rectangle(7, 11, 67, 17));
                    chkMic.setFont(new java.awt.Font("SansSerif", 0, 11));
                    chkMic.setSelected(true);
                    chkMic.setText("Habilitar microfone");
                    chkMic.setBounds(new Rectangle(61, 35, 128, 20));
                    chkSpeaker.setBounds(new Rectangle(61, 53, 118, 20));
                    chkSpeaker.setFont(new java.awt.Font("SansSerif", 0, 11));
                    chkSpeaker.setToolTipText("");
                    chkSpeaker.setSelected(true);
                    chkSpeaker.setText("Habilitar speaker");
                    panTelefone.setLayout(null);
                    jLabel11.setBounds(new Rectangle(4, 120, 51, 17));
                    jLabel11.setText("Arquivo:");
                    jLabel11.setFont(new java.awt.Font("SansSerif", 0, 11));
                    jLabel11.setHorizontalAlignment(SwingConstants.RIGHT);
                    btnAtender.setBackground(Color.gray);
                    btnAtender.setFont(new java.awt.Font("SansSerif", 0, 10));
                    btnAtender.setText("Atender");
                    btnAtender.setBounds(new Rectangle(5, 183, 76, 18));
                    btnAtender.addActionListener(new java.awt.event.ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                                        btnAtender_actionPerformed(e);
                              }
                    });
                    btnDesligar.setBounds(new Rectangle(159, 183, 76, 18));
                    btnDesligar.addActionListener(new java.awt.event.ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                                        btnDesligar_actionPerformed(e);
                              }
                    });
                    btnDesligar.setBackground(Color.gray);
                    btnDesligar.setFont(new java.awt.Font("SansSerif", 0, 10));
                    btnDesligar.setText("Desligar");
                    lblSpeaker.setBounds(new Rectangle(207, 8, 24, 10));
                    lblSpeaker.setHorizontalAlignment(SwingConstants.CENTER);
                    lblSpeaker.setOpaque(true);
                    lblSpeaker.setBorder(border2);
                    lblSpeaker.setForeground(Color.white);
                    lblSpeaker.setBackground(Color.black);
                    lblSpeaker.setFont(new java.awt.Font("SansSerif", 0, 11));


                    lblMic.setBackground(Color.black);
                    lblMic.setFont(new java.awt.Font("SansSerif", 0, 11));
                    lblMic.setForeground(Color.white);
                    lblMic.setBorder(border2);
                    lblMic.setOpaque(true);
                    lblMic.setHorizontalAlignment(SwingConstants.CENTER);
                    lblMic.setBounds(new Rectangle(207, 18, 24, 10));

                    btnChamar.setBounds(new Rectangle(82, 183, 76, 18));
                    btnChamar.addActionListener(new java.awt.event.ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                                        btnChamar_actionPerformed(e);
                              }
                    });
                    btnChamar.setBackground(Color.gray);
                    btnChamar.setFont(new java.awt.Font("SansSerif", 0, 10));
                    btnChamar.setText("Ligar");



                    btnIniciar.setBackground(Color.gray);
                    btnIniciar.setFont(new java.awt.Font("SansSerif", 0, 10));
                    btnIniciar.setText("Iniciar");
                    btnIniciar.setBounds(new Rectangle(127, 10, 72, 18));
                    btnIniciar.addActionListener(new java.awt.event.ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                                        btnIniciar_actionPerformed(e);
                              }
                    });
                    panConfiguracao.setLayout(null);
                    jTabbedPane1.setBackground(Color.black);
                    jTabbedPane1.setFont(new java.awt.Font("Dialog", 0, 11));
                    jTabbedPane1.setForeground(Color.white);
                    jTabbedPane1.setBounds(new Rectangle(1, 0, 242, 230));


                    this.addWindowListener(new java.awt.event.WindowAdapter() {
                              public void windowClosing(WindowEvent e) {
                                        this_windowClosing(e);
                              }
                    });
                    chkPlayer.setBounds(new Rectangle(61, 71, 169, 20));
                    chkPlayer.setFont(new java.awt.Font("SansSerif", 0, 11));
                    chkPlayer.setText("Habilitar player");
                    arquivoCombo.setBounds(new Rectangle(61, 115, 141, 25));
                    arquivoCombo.setFont(new java.awt.Font("SansSerif", 0, 11));
```

```
          arquivoCombo.setSelectedItem(this);
          chkRecorder.setFont(new java.awt.Font("SansSerif", 0, 11));
          chkRecorder.setText("Habilitar recorder");
          chkRecorder.setBounds(new Rectangle(61, 92, 169, 20));

          cmbCodec.setFont(new java.awt.Font("SansSerif", 0, 11));
          cmbCodec.setSelectedItem(this);
          cmbCodec.setBounds(new Rectangle(60, 10, 141, 24));
          jLabel13.setFont(new java.awt.Font("SansSerif", 0, 11));
          jLabel13.setHorizontalAlignment(SwingConstants.RIGHT);
          jLabel13.setText("Codec:");
          jLabel13.setBounds(new Rectangle(12, 12, 44, 17));
          pan1.getViewport().setBackground(Color.black);
          pan1.setBorder(border1);
          pan1.setBounds(new Rectangle(5, 34, 228, 144));
          lblFeedback.setBackground(Color.black);
          lblFeedback.setFont(new java.awt.Font("Monospaced", 0, 11));
          lblFeedback.setText("jTextArea1");
          panTelefone.setBackground(Color.darkGray);
          this.getContentPane().setBackground(Color.black);
          panTelefone.add(btnAtender, null);
          panTelefone.add(pan1, null);
          panTelefone.add(jLabel4, null);
          panTelefone.add(txtPortaSIP, null);
          panTelefone.add(lblMic, null);
          panTelefone.add(btnIniciar, null);
          panTelefone.add(lblSpeaker, null);
          panTelefone.add(btnChamar, null);
          panTelefone.add(btnDesligar, null);

          pan1.getViewport().add(lblFeedback, null);

          panConfiguracao.add(cmbCodec, null);
          panConfiguracao.add(jLabel11, null);
          panConfiguracao.add(arquivoCombo, null);
          panConfiguracao.add(chkRecorder, null);
          panConfiguracao.add(chkPlayer, null);
          panConfiguracao.add(chkSpeaker, null);
          panConfiguracao.add(chkMic, null);
          panConfiguracao.add(jLabel13, null);

          jScrollPane1.getViewport().add(txtLog, null);

          this.getContentPane().add(jTabbedPane1, null);

          jTabbedPane1.add(panTelefone, "Telefone");
          jTabbedPane1.add(panConfiguracao, "Configuração");
          jTabbedPane1.add(jScrollPane1, "Log");
}

private void iniciar() {

          try {
                    phone = new MyPhone(this);

                    setTitle(
                              phone.init(
                                        Integer.parseInt(txtPortaSIP.getText()),
                                        0,
                                        getCodec()
                              )
                    );
          } catch (Exception e) {
                    setFeedback(666,"MyPhoneUI.iniciar():\n" + e);
          }
}

private void sair() {
          hide();
          System.exit(0);
}


private int getCodec() {
          switch ( cmbCodec.getSelectedIndex() ) {
                    case 0:
                              return MyRTPSession.ULAW;
                    case 1:
                              return MyRTPSession.PCM16;
                    case 2:
                              return MyRTPSession.ALAW;
                    default:
                              return MyRTPSession.PCM16;
          }
}


void btnSair_actionPerformed(ActionEvent e) {
          sair();
}

void btnIniciar_actionPerformed(ActionEvent e) {

          if ( phone == null ) iniciar();
}

void this_windowClosing(WindowEvent e) {
          sair();
}

void btnChamar_actionPerformed(ActionEvent e) {
```

```
          closeSound();
          invite();
}

void invite() {

          try {

                    if ( phone == null ) iniciar();

                    phone.setRemetenteCodec( getCodec() );

                    MyPhoneUICall callUI = new MyPhoneUICall(this,"Chamar destinatario",true);

                    callUI.setSize(400,200);
                    callUI.setLocation(100,100);

                    callUI.show();

                    if (callUI.isOkPressed) {
                              phone.invite(
                                        callUI.txtDestinatarioHostName.getText(),
                                        Integer.parseInt(callUI.txtDestinatarioPortaSIP.getText())
                              );

                    }
          } catch (Exception e) {
                    setFeedback(666,"MyPhoneUI.invite():\n" + e);
          }
}

void btnAtender_actionPerformed(ActionEvent e) {
          closeSound();
          phone.ok();
}

void btnDesligar_actionPerformed(ActionEvent e) {
          phone.bye();
          setFeedback(-1,"Bye");
}

public int getDevice() {

          int r = 3;

          if (chkMic.isSelected()) r = 1;
          if (chkSpeaker.isSelected()) r = 2;

          if (chkMic.isSelected() && chkSpeaker.isSelected()) r = 3;

          if (chkPlayer.isSelected()) r = 4;
          if (chkRecorder.isSelected()) r = 5;

          return r;
}

public String getFile() { return (String) arquivoCombo.getSelectedItem(); }

public void setFeedback(int tipo, String msg) {

          feedbackTipo = tipo;
          feedbackMsg  = msg;

          switch (tipo) {

                              case -1:
                                        lblFeedback.setBackground(Color.black);
                                        lblFeedback.setForeground(Color.green);
                                        lblSpeaker.setBackground(Color.black);
                                        lblSpeaker.setForeground(Color.green);
                                        lblMic.setBackground(Color.black);
                                        lblMic.setForeground(Color.green);
                                        lblFeedback.setText(msg);
                                        break;

                              case 1:
                                        lblFeedback.setBackground(Color.black);
                                        lblFeedback.setForeground(Color.green);
                                        lblFeedback.setText(msg);
                                        break;

                              case 2:
                                        playSound("ringin.au");
                                        lblFeedback.setBackground(Color.black);
                                        lblFeedback.setForeground(Color.green);
                                        lblFeedback.setText(msg);
                                        break;

                              case 3:
                                        lblFeedback.setBackground(Color.black);
                                        lblFeedback.setForeground(Color.green);
                                        lblFeedback.setText(msg);
                                        break;

                              case 4:
                                        lblFeedback.setText("");
                                        break;

                              case 5:
                                        lblMic.setBackground(Color.black);
                                        lblMic.setForeground(Color.green);
```

```
                                        break;

                                case 6:
                                        lblMic.setBackground(Color.green);
                                        lblMic.setForeground(Color.black);
                                        break;

                                case 7:
                                        lblSpeaker.setBackground(Color.black);
                                        lblSpeaker.setForeground(Color.green);
                                        break;

                                case 8:
                                        lblSpeaker.setBackground(Color.green);
                                        lblSpeaker.setForeground(Color.black);
                                        break;

                                case 2723:
                                        lblFeedback.append(msg + "\n");
                                        break;

                                case 1606:
                                        txtLog.append(msg+"\n");
                                        break;

                                case 666:
                                        JOptionPane.showMessageDialog(
                                                this,
                                                msg,
                                                "Erro",
                                                JOptionPane.ERROR_MESSAGE);
                                                txtLog.append(msg+"\n");
                                        break;

                                default:
                        }

        }


        private void playSound(String fileName){

                FloatControl     gainControl;
                AudioInputStream audioInputStream = null;

                closeSound();

                try       {
                        audioInputStream = AudioSystem.getAudioInputStream(getClass().getResource(fileName));
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }


                if (audioInputStream != null) {
                        AudioFormat         format = audioInputStream.getFormat();
                        DataLine.Info       info = new DataLine.Info(Clip.class, format);
                        try       {
                                som = (Clip) AudioSystem.getLine(info);

                                if (som.isControlSupported(FloatControl.Type.MASTER_GAIN/*VOLUME*/)) {
                                        gainControl = (FloatControl) som.getControl(FloatControl.Type.MASTER_GAIN);
                                        gainControl.setValue( gainControl.getMaximum() );
                                }

                                if (som.isControlSupported(FloatControl.Type.VOLUME/*VOLUME*/)) {
                                        gainControl = (FloatControl) som.getControl(FloatControl.Type.VOLUME);
                                        gainControl.setValue( gainControl.getMaximum() );
                                }

                        som.open(audioInputStream);

                        }
                        catch (Exception e)
                        {
                                e.printStackTrace();
                        }

                        som.loop(0);
                }

        }

        private void closeSound() {
                if (som != null) {
                        som.stop();
                        som.flush();
                        som.close();
                        som = null;
                }
        }

        void repeatFeedback() {
                setFeedback(feedbackTipo,feedbackMsg);
        }


        void btnCancelar_actionPerformed(ActionEvent e) {
                phone.cancel();
```

```
        }
}
```

7.1.6 MyPhoneUICall.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;

/**
 * Title:        My Phone
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:      Mestrado em Redes de Computadores
 * @author Jucimar Maia da Silva Jr
 * @version 1.0
 */

public class MyPhoneUICall extends JDialog {

          JTextField txtDestinatarioHostName = new JTextField();
          JLabel jLabel12 = new JLabel();
          JTextField txtDestinatarioPortaSIP = new JTextField();
          JLabel jLabel8 = new JLabel();
          JLabel jLabel13 = new JLabel();
          JButton okButton = new JButton();
          JButton cancelButton = new JButton();

          boolean isOkPressed = false;
          JTextArea memoAssunto = new JTextArea();
          Border border1;
          JLabel jLabel9 = new JLabel();

          public MyPhoneUICall(Frame frame, String title, boolean modal) {
                    super(frame, title, modal);
                    try {
                              jbInit();
                              pack();
                    }
                    catch(Exception ex) {
                              ex.printStackTrace();
                    }
          }

          public MyPhoneUICall() {
                    this(null, "", false);
          }
          void jbInit() throws Exception {
                    border1 = BorderFactory.createBevelBorder(BevelBorder.LOWERED,Color.white,Color.white,new Color(178, 178,
178),new Color(124, 124, 124));
                    this.getContentPane().setLayout(null);
                    txtDestinatarioHostName.setFont(new java.awt.Font("SansSerif", 0, 11));
                    txtDestinatarioHostName.setToolTipText("");
                    txtDestinatarioHostName.setText("10.1.1.10");
                    txtDestinatarioHostName.setBounds(new Rectangle(86, 48, 139, 24));
                    jLabel12.setHorizontalAlignment(SwingConstants.RIGHT);
                    jLabel12.setFont(new java.awt.Font("SansSerif", 0, 11));
                    jLabel12.setToolTipText("");
                    jLabel12.setText("Destinatário:");
                    jLabel12.setBounds(new Rectangle(8, 49, 75, 17));
                    txtDestinatarioPortaSIP.setFont(new java.awt.Font("SansSerif", 0, 11));
                    txtDestinatarioPortaSIP.setText("5060");
                    txtDestinatarioPortaSIP.setBounds(new Rectangle(85, 75, 42, 24));
                    jLabel8.setBounds(new Rectangle(24, 77, 59, 17));
                    jLabel8.setText("Porta:");
                    jLabel8.setFont(new java.awt.Font("SansSerif", 0, 11));
                    jLabel8.setToolTipText("");
                    jLabel8.setHorizontalAlignment(SwingConstants.RIGHT);
                    jLabel13.setBounds(new Rectangle(12, 15, 214, 27));
                    jLabel13.setText("Digite o IP ou o nome do host do destinatário");
                    jLabel13.setToolTipText("");
                    jLabel13.setFont(new java.awt.Font("SansSerif", 0, 11));
                    jLabel13.setHorizontalAlignment(SwingConstants.RIGHT);
                    okButton.setFont(new java.awt.Font("SansSerif", 0, 11));
                    okButton.setText("Ok");
                    okButton.setBounds(new Rectangle(302, 14, 78, 25));
                    okButton.addActionListener(new java.awt.event.ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                                        okButton_actionPerformed(e);
                              }
                    });
                    cancelButton.setBounds(new Rectangle(302, 42, 78, 25));
                    cancelButton.addActionListener(new java.awt.event.ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                                        cancelButton_actionPerformed(e);
                              }
                    });
                    cancelButton.setText("Cancelar");
                    cancelButton.setFont(new java.awt.Font("SansSerif", 0, 11));
                    cancelButton.setToolTipText("");
                    memoAssunto.setBorder(border1);
                    memoAssunto.setBounds(new Rectangle(85, 105, 205, 46));
                    jLabel9.setHorizontalAlignment(SwingConstants.RIGHT);
                    jLabel9.setToolTipText("");
                    jLabel9.setFont(new java.awt.Font("SansSerif", 0, 11));
                    jLabel9.setText("Assunto:");
                    jLabel9.setBounds(new Rectangle(23, 105, 59, 17));
                    this.getContentPane().add(txtDestinatarioHostName, null);
                    this.getContentPane().add(jLabel12, null);
                    this.getContentPane().add(jLabel8, null);
```

```
                this.getContentPane().add(txtDestinatarioPortaSIP, null);
                this.getContentPane().add(jLabel13, null);
                this.getContentPane().add(memoAssunto, null);
                this.getContentPane().add(jLabel9, null);
                this.getContentPane().add(cancelButton, null);
                this.getContentPane().add(okButton, null);


        }

        void cancelButton_actionPerformed(ActionEvent e) {
                hide();
        }

        void okButton_actionPerformed(ActionEvent e) {
                isOkPressed = true;
                hide();
        }
}
```

7.1.7 MyPhoneUtil.java

```
/**
 * MyPhoneUtil.java
 *
 * @author Jucimar Jr
 *
 * Classe usada para armazenar constantes e funçoes usadas em todos as outras
 * classes
 *
 */

import java.math.BigInteger;

public class MyPhoneUtil {

	/**
	 * converte um vetor de bytes big-endian em um numero inteiro de 32bits
	 */
	public static long byteToLong64(byte[] byteArray, int initIndex ) {

		//array para formar o long
		// tmpByteArray[0] indica o sinal q deve ser sempre +
		byte[] tmpByteArray = new byte[9];

		//torna o sinal positivo;
		tmpByteArray[0] = 0;

		tmpByteArray[1] = byteArray[initIndex];
		tmpByteArray[2] = byteArray[initIndex+1];
		tmpByteArray[3] = byteArray[initIndex+2];
		tmpByteArray[4] = byteArray[initIndex+3];
		tmpByteArray[5] = byteArray[initIndex+4];
		tmpByteArray[6] = byteArray[initIndex+5];
		tmpByteArray[7] = byteArray[initIndex+6];
		tmpByteArray[8] = byteArray[initIndex+7];

		return new BigInteger(tmpByteArray).longValue();

		/*
		return ((byteArray[initIndex]<<56)
		 | ((byteArray[initIndex+1] & 0xFF)<<48)
		 | ((byteArray[initIndex+2] & 0xFF)<<40)
		 | ((byteArray[initIndex+3] & 0xFF)<<32)
		 | ((byteArray[initIndex+4] & 0xFF)<<24)
		 | ((byteArray[initIndex+5] & 0xFF)<<16)
		 | ((byteArray[initIndex+6] & 0xFF)<<8)
		 |  (byteArray[initIndex+7] & 0xFF)
		);
		*/
	}

	/**
	 * converte um vetor de bytes big-endian em um numero inteiro de 32bits
	 */
	public static long byte4ToLong(byte[] byteArray, int initIndex ) {

		//array para formar o long
		// tmpByteArray[0] indica o sinal q deve ser sempre +
		byte[] tmpByteArray = new byte[5];

		//torna o sinal positivo;
		tmpByteArray[0] = 0;

		tmpByteArray[1] = byteArray[initIndex];
		tmpByteArray[2] = byteArray[initIndex+1];
		tmpByteArray[3] = byteArray[initIndex+2];
		tmpByteArray[4] = byteArray[initIndex+3];

		return new BigInteger(tmpByteArray).longValue();
	}

	/**
	 * converte um vetor de bytes big-endian em um numero inteiro de 32bits
	 */
	public static int byteToInt32(byte[] byteArray, int initIndex ) {

		return ((byteArray[initIndex]<<24) |
				((byteArray[initIndex+1] & 0xFF)<<16) |
				((byteArray[initIndex+2] & 0xFF)<<8) |
					(byteArray[initIndex+3] & 0xFF));
	}

	/**
	 * converte um vetor de bytes big-endian em um numero inteiro de 16bits
	 */
	public static int byteToInt16(byte[] byteArray, int initIndex ) {
		return ( byteArray[initIndex]<<8 ) | ( byteArray[initIndex+1] & 0xFF );
	}

	/**
	 * converte um vetor de bytes big-endian em um numero short de 16bits
	 */
	public static int byteToShort16(byte[] byteArray, int initIndex ) {
		return (short) byteToInt16(byteArray, initIndex);
	}

	/**
```

```
	 * converte um numero long em um vetor de bytes
	 */
	public static byte[] longToByte4(long longInt, byte[] byteArray, int initIndex){
		byteArray[initIndex]   = (byte) ((longInt & 0xFF000000L)>> 24);
		byteArray[initIndex+1] = (byte) ((longInt & 0xFF0000L)  >> 16);
		byteArray[initIndex+2] = (byte) ((longInt & 0xFF00L)    >> 8);
		byteArray[initIndex+3] = (byte) (longInt & 0xFFL);

		return byteArray;
	}

	/**
	 * converte um numero long de 64bits big-endian em um vetor[8] de bytes
	 */
	public static byte[] long64ToByte(long l, byte[] byteArray, int initIndex){

		byteArray[initIndex]   = (byte) (l >> 56);
		byteArray[initIndex+1] = (byte) ((l >>> 48) & 0xFF);
		byteArray[initIndex+2] = (byte) ((l >>> 40) & 0xFF);
		byteArray[initIndex+3] = (byte) ((l >>> 32) & 0xFF);
		byteArray[initIndex+4] = (byte) ((l >>> 24) & 0xFF);
		byteArray[initIndex+5] = (byte) ((l >>> 16) & 0xFF);
		byteArray[initIndex+6] = (byte) ((l >>> 8) & 0xFF);
		byteArray[initIndex+7] = (byte) (l & 0xFF);

		return byteArray;
	}

	/**
	 * converte um numero inteiro de 32bits big-endian em um vetor de bytes
	 */
	public static byte[] int32ToByte(int int32, byte[] byteArray, int initIndex){
		byteArray[initIndex]   = (byte) (int32 >> 24);
		byteArray[initIndex+1] = (byte) ((int32 >>> 16) & 0xFF);
		byteArray[initIndex+2] = (byte) ((int32 >>> 8) & 0xFF);
		byteArray[initIndex+3] = (byte) (int32 & 0xFF);

		return byteArray;
	}

	/**
	 * converte um numero inteiro de 16bits em um vetor de bytes
	 */
	public static byte[] int16ToByte(int int16, byte[] byteArray, int initIndex){
		byteArray[initIndex]   = (byte) (int16 >> 8); // +SIG
		byteArray[initIndex+1] = (byte) (int16 & 0xFF);; //-SIG

		return byteArray;
	}

	/**
	 * converte um numero short de 16bits em um vetor de bytes
	 */
	public static byte[] short16ToByte(short short16, byte[] byteArray, int initIndex){
		return int16ToByte( short16, byteArray, initIndex);
	}
}
```

7.1.8 MyPlayer.java

```
/**
 * MyPlayer.java
 *
 * @author Jucimar Jr
 *
 * Lê um arquivo no formato como PCM 16bits 8KHz Mono Big-endian e coloca
 * pacote por pacote dentro da fila de mensagens para serenviado pela rede
 *
 */

import    javax.sound.sampled.*;

public class MyPlayer {

          /**
           * recebe a sessao rtp + o nome do arquivo a ser lido
           */
          public MyPlayer(MyRTPSession rtp, String filename) {

                    try {

                              AudioInputStream audioInputStream;
                              int eof = 0;
                              int i = 0;
                              byte[] soundPacket =
                                        new byte[rtp.rtpDestinatario.rtpLineBufferSize];

                              // abre o arquivo baseado na url dele
                              audioInputStream =
                                        AudioSystem.getAudioInputStream(
                                        getClass().getResource(filename)
                              );

                              // enqto nao for fim de arquivo (eof = -1) vai lendo do arquivo e
                              // colocando na fila de msgs. os pacotes da fila ja estao
                              // codificados
                              while ( eof != -1 ) {
                                        eof = audioInputStream.read(soundPacket, 0, soundPacket.length);

                                        // nao insere o pacote -1 pro arquivo ficar igual
                                        if (eof != -1) {
                                                  rtp.queueSender.putMsg(
                                                            MyCodec.convertFromPCM16(
                                                                      soundPacket,
                                                                      rtp.rtpDestinatario.rtpPayloadType
                                                            )
                                                  );
                                        }
                              }
                    } catch (Exception e) {
                              rtp.setFeedback(666,"player():\n" + e);
                    }
          }
}
```

7.1.9 MyQueueBuffer.java

```
import java.util.Vector;

public class MyQueueBuffer extends Vector {

          public MyQueueBuffer () {
                    super();
          }

          public synchronized Object getMsg() throws Exception {
                    while (isEmpty()) {
                              wait();
                    }
                    Object msg = firstElement();
                    removeElement( msg );
                    return msg;
          }

          public synchronized void putMsg( Object msg ) throws Exception {
                    addElement(msg);
                    notify();
          }

}
```

7.1.10 MyRecorder.java

```
/**
 * MyRecorder.java
 *
 * @author Jucimar Jr
 *
 * Grava um arquivo no formato como PCM 16bits 8KHz Mono Big-endian com os
 * pacotes de som capturados. Os pacotes sao capturados da filae escritos em
 * um ByteArrayOutputStream. Desse outputStream eles sao colocados em um
 * ByteArrayInputStream e entao gravados como um arquivo de som .AU
 *
 * A ideia inicial de colocar uma thread q ficasse lendo continuamente a fila e
 * preenchendo o outputStream não funcionou muito bem. os pacotes sumiam do nada
 *
 * agora espero q tdos os pacotes sejam enviados e colocados na fila e só entao,
 * ao finalizar a chamada, os pacotes são gravados em arquivo.
 *
 */

import    java.io.*;
import    javax.sound.sampled.*;

public class MyRecorder {

          private MyRTPSession rtpSession;
          private boolean               isRunning = true;
          private String       fileName;

          /**
           * é necessário passar como parametros a fila de mgs, o nome do arquivo
           * que será criado e o tipo de codificador que será utilizado
           */
          public MyRecorder(MyRTPSession rtp, String filename) {
                    fileName   = filename;
                    rtpSession = rtp;
          }

          /**
           * captura os pacotes da fila
           * grava eles numa outputstream
           * passa eles pra uma inputstream
           * usa o audiosystem e essa inputstream para gravar o arquivo .au
           */
          public void saveFile() {

                    try {

                              ByteArrayOutputStream out = new ByteArrayOutputStream();

                              int i = 0;

                              //enqto tiver msgs na fila, vai gravado uma a uma no outputstream
                              while ( ! rtpSession.queueReceiver.isEmpty() ) {

                                        // pedaco de som capturado da fila
                                        byte[] soundPacket =
                                                  MyCodec.convertToPCM16(
                                                            (byte[]) rtpSession.queueReceiver.getMsg(),
                                                             rtpSession.rtpRemetente.rtpPayloadType
                                                  );

                                        out.write(soundPacket,0, soundPacket.length );
                              }

                              // fecha o array
                              out.close();

                              // passa o array da output pra inputstream
                              byte[] soundArray = out.toByteArray();
                              ByteArrayInputStream in = new ByteArrayInputStream(soundArray);

                              // define o formato e prepara o sistema de som
                              // pra gravação do arquivo .au PCM 16bits 8KHz Mono Big-endian
                              AudioFormat audioFormat = new AudioFormat(8000.0F, 16, 1, true, true);
                              AudioFileFormat.Type fileType = AudioFileFormat.Type.AU;

                              //cria a audioinputstream
                              AudioInputStream audioInputStream =
                                        new AudioInputStream(
                                                  in,
                                                  audioFormat,
                                                  soundArray.length / audioFormat.getFrameSize()
                                        );

                              //CRIA O arquivo no disco
                              AudioSystem.write(audioInputStream, fileType, new File(fileName));
                    } catch (Exception e) {
                              rtpSession.setFeedback(666,"recorder.saveFile():\n" + e);
                    }
          }
}
```

7.1.11 MyRTPSession.java

```
/**
 * MyRTPSession.java
 * @author Jucimar Jr
 *
 * formacao do cabeçalho rtp tirado da rfc 1889
 * Os 1os 12 bytes sao obrigatórios
 *

          0                   1                   2                   3
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |V=2|P|X|  CC   |M|     PT      |       sequence number         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           timestamp                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           synchronization source (SSRC) identifier            |
   +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
   |            contributing source (CSRC) identifiers             |
   |                               ....                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 - byte 1

  0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|1 0|0|0|0 0 0 0| = -128
+-+-+-+-+-+-+-+-+


- byte 2

+-+-+-+-+-+-+-+-+
|0|0 0 0 0 0 0 0| = 0 para PCMU
+-+-+-+-+-+-+-+-+

+-+-+-+-+-+-+-+-+
|0|0 0 0 1 0 0 0| = 8 para PCMA
+-+-+-+-+-+-+-+-+

+-+-+-+-+-+-+-+-+
|0|0 0 0 0 0 1 1| = 0 para GSM
+-+-+-+-+-+-+-+-+

- byte 3,4 - transformar um inteiro em  2 bytes
- byte 5,6,7 e 8 - transformar um inteiro em  4 bytes
- byte 9,10,11 e 12 - transformar um inteiro em  4 bytes

entao:

byte[] rtpHeader = {
          -128,
          codecByte,
          seqNumberByte1,
          seqNumberByte2,
          timestampByte1,
          timestampByte2,
          timestampByte3,
          timestampByte4,
          ssrcByte1,
          ssrcByte2,
          ssrcByte3,
          ssrcByte4}
*/

import java.util.*;
import java.io.*;

public class MyRTPSession {

          public static final int ULAW  = 0;
          public static final int ALAW  = 8;
          public static final int GSM   = 3;
          public static final int PCM16 = 96;

          public static final int RTP_HEADER_SIZE = 12;
          public static final int PKT_REPORT_SIZE = 22;


          // os pacotes estao com o tamanho dobrado
          public static final int ULAW_PCT_SIZE  = 160;
          public static final int ALAW_PCT_SIZE  = 160;
          public static final int GSM_PCT_SIZE   = 160;
          public static final int PCM16_PCT_SIZE = 8000;

          // os pacotes estao com o tamanho dobrado
          public static final int ULAW_LINE_BUFFER_SIZE  = 320;
          public static final int ALAW_LINE_BUFFER_SIZE  = 320;
          public static final int GSM_LINE_BUFFER_SIZE   = 320;
          public static final int PCM16_LINE_BUFFER_SIZE = 8000;

          MyMic      mic;
          MySpeaker  speaker;
          MyRecorder recorder;
          MyPlayer   player;

          MyHost rtpDestinatario;
```

```
MyHost rtpRemetente;

MyQueueBuffer queueSender;
MyQueueBuffer queueReceiver;

MyRTPSocketSender   socketSender;
MyRTPSocketReceiver socketReceiver;

MyPhoneUI phoneUI;

ByteArrayOutputStream outSender;
ByteArrayOutputStream outReceiver;

int rtpRemetentePayloadType;
int rtpRemetenteLineBufferSize;
int rtpRemetentePacketSize;

int rtpDestinatarioPayloadType;
int rtpDestinatarioLineBufferSize;
int rtpDestinatarioPacketSize;

int rtpSSRC;
int rtpTimestamp = 0;
int rtpSeqNumber = 0;

int rtpQtdTimestamp = 0;
int rtpQtdSeqNumber = 0;

int pctReceived = 1;
int pctSent = 1;

/**
 * passa a tela, remetente e destinatario
 */
public MyRTPSession(MyHost rm, MyHost dt, MyPhoneUI ui) {

          try {

                    phoneUI = ui;

                    rtpRemetente    = rm;
                    rtpDestinatario = dt;

                    rtpSSRC = (int) ( Math.random() * 100000);

                    rtpRemetente.rtpPacketSize =
                              setPacketSize(rtpRemetente.rtpPayloadType);

                    rtpRemetente.rtpLineBufferSize =
                              setLineBufferSize(rtpRemetente.rtpPayloadType);

                    rtpDestinatario.rtpPacketSize =
                              setPacketSize(rtpDestinatario.rtpPayloadType);

                    rtpDestinatario.rtpLineBufferSize =
                              setLineBufferSize(rtpDestinatario.rtpPayloadType);

                    System.out.println(
                              rtpRemetente.print("remetente") +
                              rtpDestinatario.print("destinatario")
                    );


                    // seleciona o device escolhido na tela

                    int dev = phoneUI.getDevice();

                    switch (dev) {
                              case 1:
                                        startMic();
                                        break;
                              case 2:
                                        startSpeaker();
                                        break;
                              case 3:
                                        startMic();
                                        startSpeaker();
                                        break;
                              case 4:
                                        startPlayer();
                                        break;
                              case 5:
                                        startRecorder();
                                        break;
                    }
          } catch (Exception e) {
                    setFeedback(666,"RTPSession:\n" + e);
          }
}

/**
 * inicializa o microfone
 */
private void startMic() throws Exception {

          queueSender  = new MyQueueBuffer();
          socketSender = new MyRTPSocketSender(this);

          if ( mic==null ) mic = new MyMic(this);

          mic.start();
```

```
          System.out.println("ABRIR MICROFONE");
}

/**
 * Inicializa o playback
 */
private void startSpeaker() throws Exception {
          queueReceiver = new MyQueueBuffer();
          socketReceiver = new MyRTPSocketReceiver(this);
          socketReceiver.setReceiveBufferSize(2147483647);
          if ( speaker == null ) speaker = new MySpeaker(this);

          speaker.start();

          System.out.println("ABRIR CX DE SOM");
}

/**
 * inicializa o gravador de arquivos
 */
private void startRecorder() throws Exception {

          if ( recorder == null ) {

                    outReceiver = new ByteArrayOutputStream();
                    queueReceiver  = new MyQueueBuffer();
                    socketReceiver = new MyRTPSocketReceiver(this);
                    socketReceiver.setReceiveBufferSize(2147483647);


                    recorder = new MyRecorder(this,
                              rtpDestinatario.hostName + "_" +
                              rtpDestinatario.rtpPort  + "_to_" +
                              rtpRemetente.hostName + "_" +
                              rtpRemetente.rtpPort  + "_at_" +
                              new Date().getTime() + "_" +
                              phoneUI.getFile()
                    );
          }
}

/**
 * inicializa o player
 */
private void startPlayer() throws Exception {

          queueSender  = new MyQueueBuffer();
          socketSender = new MyRTPSocketSender(this);
          socketSender.setSendBufferSize(2147483647);

          if ( player == null ) player = new MyPlayer(this, phoneUI.getFile());
}

/**
 * fecha o microfone
 */
private void stopMic() throws Exception {
          if ( mic != null ) {
                    stopSocketSender();
                    mic.close();
          }
}

/**
 * fecha o playback
 */
private void stopSpeaker() throws Exception {
          if ( speaker != null ) {
                    stopSocketReceiver();
                    speaker.close();
          }
}

/**
 * fecha o gravador de arquivos
 */

private void stopRecorder() throws Exception {
          if ( recorder != null ) {
                    recorder.saveFile();
                    recorder = null;
                    stopSocketReceiver();
          }
}

/**
 * fecha o playback de arquivos
 */
private void stopPlayer() throws Exception {
          if ( player != null ) {
                    player = null;
                    stopSocketSender();
          }
}

private void stopSocketSender() throws Exception {
          socketSender.isRunning = false;
          socketSender.disconnect();
          socketSender.close();
          socketSender = null;
```

```
}

private void stopSocketReceiver() throws Exception {
          socketReceiver.isRunning = false;
          socketReceiver.close();
          socketReceiver = null;
}

/**
 * fecha td
 */
public void close() {

          try {
                    stopMic();
                    stopSpeaker();
                    stopRecorder();
                    stopPlayer();
                    writeReportFiles();

          } catch (Exception e) {
                    setFeedback(666,"MyRTPSession.close():\n" + e.toString());
          }
}

/**
 * insere o cabeçalho udp no pacote de som
 */

public byte[] insertRtpHeader(byte[] soundPacket)  throws Exception{

/*
          // teste de sincronização JJR
          byte[] soundPacket = (byte[]) queueSender.getMsg();
*/

          // o cabeçalho rtp tem 12 bytes obrigatorios
          byte[] rtpHeader  = new byte[RTP_HEADER_SIZE];

          // cria o pacote rtp com o tamanho do cabeçalho rtp + o tamanho do
          // pacote de som
          byte[] rtpPacket = new byte[rtpHeader.length + soundPacket.length];

          // se chegou ao tamanho max aceito no pacote, zera,
          // incrementa o contador de "zeradas". senao incrementa
          // de um para cada pacote enviado
          rtpSeqNumber++;
          if (rtpSeqNumber > 65535) {
                    rtpSeqNumber = 0;
                    rtpQtdSeqNumber++;
          }

          // incrementa o timestamp de acordo com o tipo de codec e o
          // tamanho do pct usado
          // se chegar ao max zera e incrementa o contador de voltas
          rtpTimestamp = rtpTimestamp + rtpRemetente.rtpPacketSize;
          if (rtpTimestamp > 2147483647) {
                    rtpTimestamp = 0;
                    rtpQtdTimestamp++;
          }

          // V + P + X + CC
          rtpHeader[0] = -128;

          // M + PT
          rtpHeader[1] = (byte) rtpRemetente.rtpPayloadType;

          // sequence number
          MyPhoneUtil.int16ToByte(rtpSeqNumber,rtpHeader,2);

          //timestamp
          MyPhoneUtil.int32ToByte(rtpTimestamp,rtpHeader,4);

          //ssrc
          MyPhoneUtil.int32ToByte(rtpSSRC,rtpHeader,8);

          // copia header pra dentro do rtpPayload
          System.arraycopy(rtpHeader, 0, rtpPacket, 0, rtpHeader.length);

          // copia o som codificado pra dentro do payload
          System.arraycopy(soundPacket, 0, rtpPacket, 12, soundPacket.length);

          System.out.println("Spct#:" + pctSent++ +
                    "\tpct lenght:" + rtpPacket.length +
                    "\tssrc:"      + rtpSSRC +
                    "\tSeqnum:"    + rtpSeqNumber +
                    "\tTimestamp:" + rtpTimestamp
          );
          return rtpPacket;
}

/**
 * remove o cabeçalho RTP para o pacote de som ser inserido na fila
 * de mensagens
 */
public byte[] removeRtpHeader(byte[] rtpPacket) throws Exception{

          byte[] soundPacket = new byte[rtpPacket.length - RTP_HEADER_SIZE];
          byte[] rtpPacketReport = new byte[PKT_REPORT_SIZE];

          //captura o seqnumber
```

```
		int rtpRecSeqNumber = MyPhoneUtil.byteToInt16(rtpPacket,2);

		//captura o timestamp
		int rtpRecTimestamp = MyPhoneUtil.byteToInt32(rtpPacket,4);

		//captura o ssrc
		int rtpRecSSRC = MyPhoneUtil.byteToInt32(rtpPacket,8);


		// copia o payload do pacote pra dentro do array
		System.arraycopy(
			rtpPacket,
			RTP_HEADER_SIZE,
			soundPacket,
			0,
			soundPacket.length
		);

		//contador de pacotes - 4 bytes
		MyPhoneUtil.int32ToByte(pctReceived,rtpPacketReport,0);

		// + 10 bytes
		//copia o cabeçalho do pacote rtp pra dentro de um array
		System.arraycopy(rtpPacket, 2, rtpPacketReport, 4, RTP_HEADER_SIZE-2);

		// acrescenta a data/hora de recebimento
		long datelong = new  Date().getTime();

		// + 8 bytes
		MyPhoneUtil.long64ToByte(datelong,rtpPacketReport,14);

		long datelong2 =  MyPhoneUtil.byteToLong64(rtpPacketReport,14);

		if (recorder != null)
			outReceiver.write(rtpPacketReport,0,rtpPacketReport.length);



		System.out.println("Rpct#:" + pctReceived++ +
			"\tpct lenght:" + rtpPacket.length +
			"\tssrc:" + rtpRecSSRC +
			"\tSeqnum:" + rtpRecSeqNumber +
			"\tTimestamp:" + rtpRecTimestamp +
			"\tDatelong:"  + datelong +
			"\tDatelong2:" + datelong2
		);

		return soundPacket;
	}


	private void writeReportFiles() throws IOException {

		if (outReceiver != null) {
			outReceiver.close();

			FileWriter file =
				new FileWriter (
					"rr_" +
					rtpDestinatario.hostName + "_" +
					rtpDestinatario.rtpPort + "_" +
					new Date().getTime() + "_" +
					phoneUI.getFile() + ".txt"
				);

			byte[] rrArray = outReceiver.toByteArray();

			int arrayLength = rrArray.length-1;

			for (int i = 0; i <= arrayLength; i= i + PKT_REPORT_SIZE) {

				//captura contador
				int rtpCont = MyPhoneUtil.byteToInt32(rrArray, i);

				//captura o seqnumber
				int rtpRecSeqNumber = MyPhoneUtil.byteToInt16(rrArray, i + 4);

				//captura o timestamp
				int rtpRecTimestamp = MyPhoneUtil.byteToInt32(rrArray, i + 6);

				//captura o ssrc
				int rtpRecSSRC = MyPhoneUtil.byteToInt32(rrArray, i + 10);

				//captura timestamp de chegada
				long rtpRecTime = MyPhoneUtil.byteToLong64(rrArray,i + 14);

				file.write(
					i + 1 + "\t" +
					rtpCont + "\t" +
					rtpRecSSRC + "\t" +
					rtpRecSeqNumber + "\t" +
					rtpRecTimestamp + "\t" +
					rtpRecTime + "\r\n"
				);
			}

			file.close();

		};

	}
```

```
	/**
	 * pega o tamanho padrão de pacotes de acordo com o codificador
	 */
	public int setPacketSize(int codec) {
		switch (codec) {
			case ULAW:
				return ULAW_PCT_SIZE;
			case ALAW:
				return ALAW_PCT_SIZE;
			case GSM:
				return GSM_PCT_SIZE;
			case PCM16:
				return PCM16_PCT_SIZE;
			default:
				return PCM16_PCT_SIZE;
		}
	}

	public int setLineBufferSize(int codec) {
		switch (codec) {
			case ULAW:
				return ULAW_LINE_BUFFER_SIZE;
			case ALAW:
				return ALAW_LINE_BUFFER_SIZE;
			case GSM:
				return GSM_LINE_BUFFER_SIZE;
			case PCM16:
				return PCM16_LINE_BUFFER_SIZE;
			default:
				return PCM16_LINE_BUFFER_SIZE;
		}
	}

	/**
	 * passa o feedback para a interface gráfica
	 */
	public void setFeedback(int tipo, String msg) {
		phoneUI.setFeedback(tipo,msg);
	}
}
```

7.1.12 MyRTPSocketReceiver.java

```
/**
 * MyRTPSocketReceiver.java
 * @author Jucimar Jr
 *
 * Socket UDP para recebimento de pacotes RTP
 *
 * Cada pacote udp contem o cabeçalho rtp + os samples de som
 */

import java.net.*;
import java.util.*;
import java.io.*;

public class MyRTPSocketReceiver extends DatagramSocket implements Runnable{

          private MyQueueBuffer queueReceiver; // buffer da cx de som
          public  boolean       isRunning = true;
          private MyRTPSession  rtpSession;

          /**
           * inicia o socket + a thread
           */
          public MyRTPSocketReceiver(MyRTPSession rtp)
          throws SocketException {
                    super(rtp.rtpRemetente.rtpPort);
                    rtpSession = rtp;
                    new Thread(this).start();
          }

          /**
           * recebe pacotes udp e retira o cabeçalho rtp para o som ser processado
           */
          protected void receivePacket() throws Exception {

                    // captura o tamanho do pacote de acordo com o codificador usado pelo
                    // destinatario
                    int packetSize = rtpSession.rtpRemetente.rtpPacketSize;

                    // cria o buffer com o tamnho do pacote + o tamanho do
                    // cabeçalho rtp
                    byte[] rtpPacket = new byte[packetSize + MyRTPSession.RTP_HEADER_SIZE];

                    DatagramPacket udpPacket = new DatagramPacket(
                              rtpPacket,
                              rtpPacket.length
                    );

                    //recebe o pacote e ..
                    receive(udpPacket);

                    //retira o cabeçalho rtp
                    byte[] soundPacket = rtpSession.removeRtpHeader(rtpPacket);

                    // poe o pacote de som na fial de mensagens
                    rtpSession.queueReceiver.putMsg( soundPacket );
          }

          /**
           * fica rodando esperando a chegada de algum pacote udp
           * fica manda o feedback de recepçao pra interface gráfica como se fosse
           * um LED
           */
          public void run() {

                    try {
                              while (isRunning) { receivePacket();}
                              this.close();
                    } catch (SocketException se) {
                    } catch (Exception e) {
                              rtpSession.setFeedback(666,"MyRTPSocketReceiver.run():\n" + e);
                    }
          }
}
```

7.1.13 MyRTPSender.java

```
/**
 * MyRTPSocketSender.java
 * @author Jucimar Jr
 *
 * Socket UDP para envio de pacotes RTP
 *
 * Cada pacote udp contem o cabeçalho rtp + os samples de som
 * @todo colocar um capturar de erros q manda as coisas pra interface grafica
 */

import java.net.*;
import java.util.*;
import java.io.*;

public class MyRTPSocketSender extends DatagramSocket implements Runnable{

	public  boolean isRunning = true;

	MyRTPSession rtpSession;

	/**
	 * inicia o socket e a thread
	 */
	public MyRTPSocketSender(MyRTPSession rtp)
	throws SocketException {
		super();
		//setSendBufferSize(65536);
		new Thread(this).start();
		rtpSession = rtp;

		// conecta com o servidor pra ficar mais rápido
		connect(
			rtpSession.rtpDestinatario.hostAddress,
			rtpSession.rtpDestinatario.rtpPort
		);
	}

	/**
	 * envia um pacote
	 */
	protected void sendPacket() throws Exception {

		//captura o pacote de som da fila
		byte[] soundPacket = (byte[]) rtpSession.queueSender.getMsg();

		// insere o cabeçalho rtp no pacote de som
		byte[] rtpPacket   = rtpSession.insertRtpHeader(soundPacket);

		//empacota o pacote rtp dentro de um udp
		DatagramPacket udpPacket = new DatagramPacket(
			rtpPacket,
			rtpPacket.length,
			rtpSession.rtpDestinatario.hostAddress,
			rtpSession.rtpDestinatario.rtpPort
		);

		// envia pela Internet
		send(udpPacket);
   }


   /**
	* fica enviando os pacotes udp enqto o socket estiver rodando
	* sempre enviar fica mando o feedback pra a interface de usuário
	*/
   public void run() {

		try {
			while (isRunning) { sendPacket(); }
			this.close();
		} catch (SocketException se) {
		} catch (Exception e) {
			rtpSession.setFeedback(666,"MyRTPSocketSender.run():\n" + e);
		}
   }
}
```

7.1.14 MySIPSession.java

```
/**
 * MySIPSession.java
 * @author Jucimar Jr
 *
 * envia , gera e parsa as msg SIP
 * ele é uma thread q fica recebendo mgs da fila e enviando pelo socket
 *
 */

import java.io.*;
import java.util.*;

public class MySIPSession extends Thread {

          final static int SIP_TIMEOUT = 10000;
          final static int SIP_PORT    = 5060;
          final static String CR       = "\r\n";

          // msgs SIP
          final static int NONE     = 0;
          final static int INVITE   = 1;
          final static int ACK      = 2;
          final static int BYE      = 3;
          final static int CANCEL   = 4;
          final static int REGISTER = 5;
          final static int TRYING   = 100;
          final static int RINGING  = 180;
          final static int CALL_FORWARDING = 181;
          final static int QUEUED   = 182;
          final static int OK       = 200;
          final static int OK_IN    = 201;
          final static int BAD_RESQUEST     = 400;
          final static int UNAUTHORIZED     = 401;
          final static int PAYMENT_REQUIRED = 402;
          final static int FORBIDDEN        = 403;
          final static int NOT_FOUND        = 404;
          final static int TRANSACTION_CANCELED = 487;
          final static int SERVER_FAILURE       = 500;
          final static int BUSY                 = 600;

          final static int TALKING  = 999;
          final static int TAM_MAX = 65536;
          private boolean DEBUG = true;


          MySIPSocket   sipSocket;
          MyQueueBuffer sipQueue;
          MyHost        sipRemetente;
          MyHost        sipDestinatario;

          MyRTPSession rtpSession;

          MyPhoneUI phoneUI;

          // campos sip
          String sipUrl;
          String sipCallID;
          String sipFrom;
          String sipTo;
          String sipCSeq_metodo;
          int    sipCSeq_cont = 1;
          String sipContentType;
          int    sipContentLength;
          String sipContact;
          String sipSubject;

          String sdp;
          String sdpV;
          String sdpO;
          String sdpC;
          String sdpT;
          String sdpM;
          String sdpA;
          String sdpS;


          // posicao atual da sessao
          int sipStatus = NONE;

          /**
           * cria a sessao indicando o remetente e a tela
           * start a thread
           */
          public MySIPSession (MyPhoneUI ui, MyHost remetente) throws Exception {

                    sipRemetente = remetente;
                    sipQueue  = new MyQueueBuffer();
                    sipSocket = new MySIPSocket(this);
                    phoneUI   = ui;
                    new Thread(this).start();
          }

          /**
           * passa o feedback da sessao sip para a tela
           */
          public void setFeedback(int tipo, String msg) {
```

```
          phoneUI.setFeedback(tipo,msg);
}

/**
 * altera o destinatario
 */
public void setDestinatario(MyHost destinatario) {
          if (destinatario == null) {
                    sipDestinatario = null;
          } else {
                    sipDestinatario = destinatario;
          }
}

/**
 * retorna o destinatario
 */
public MyHost getDestinatario() {
          return sipDestinatario;
}

/**
 * retorna o destinatario
 */
public MyHost getRemetente() {
          return sipRemetente;
}


/**
 * altera o tipo de payload
 */
public void setPayloadType(int codec) {
          sipRemetente.rtpPayloadType = codec;
}


/**
 * Abre a sessao RTP
 */
private void openRtpSession() {
          try {
                    // seta o timeout do socket sip para "eterno"
                    // pois durante a conversa rtp nao tem msgs sip
                    sipSocket.setSoTimeout(0);

                    rtpSession = new MyRTPSession (
                              sipRemetente,
                              sipDestinatario,
                              phoneUI
                    );

                    sipStatus = TALKING;
                    setFeedback(3,"TALKING\n" + getDestinatario().sipUrl);

          } catch (Exception e) {
                    setFeedback(666,"openRtpSession():\n" + e);
          }
}

/**
 * Fecha a sessaoRTP
 */
private void closeRtpSession() {

          try {
                    // fecha a sessao rtp
                    if ( rtpSession != null) {
                              rtpSession.close();
                              rtpSession = null;
                    }

                    // apaga o destinatario, pois numa proxima sessao ele pode mudar
                    sipDestinatario = null;

                    // deixa a sessao sip em estado de espera por uma nova conexao
                    sipStatus = NONE;

                    setFeedback(-1,"BYE");

                    //seta o timeout da sessao para eterno, pois provavelmente foi modi
                    // fica do por um BYE ou CANCEL
                    sipSocket.setSoTimeout(0);

          } catch (Exception e) {
                    setFeedback(666,"closeRtpSession():\n" + e);
          }
}

/**
 * processa as msgs sip q chegam via socket
 */
public void run() {
          try {
                    while ( true ) {
                              // pega as mensagens da fila sip
                              // e verifica se elas ta ok
                              String msg = (String) sipQueue.getMsg();

                              int sipMsgType = parse(msg);
```

```
                                switch (sipMsgType) {
                                        case INVITE:
                                                receiveINVITE(msg);
                                                break;
                                        case OK:
                                                receiveOK(msg);
                                                break;
                                        case ACK:
                                                receiveACK();
                                                break;
                                        case BYE:
                                                receiveBYE();
                                                break;
                                        case CANCEL:
                                                receiveCANCEL();
                                                break;
                                        case TRANSACTION_CANCELED:
                                                receiveTRANSACION_CANCELED();
                                                break;
                                        case RINGING:
                                                receiveRINGING();
                                                break;
                                } // switch
                        }// while

                } catch (Exception e) {
                        setFeedback(666,"MySIPSession.run():\n" + e);
                }

        }

        /**
         * envia uma msg invite
         */
        public void sendINVITE () {
                if ( sipStatus == NONE) {
                        sipStatus = INVITE;
                        sipSocket.sendPacket(createINVITE() );
                        try {
                                sipSocket.setSoTimeout(SIP_TIMEOUT);
                        } catch (Exception e) {};
                }
        }

        /**
         * envia uma msg bye
         */
        public void sendBYE() {
                if ( sipStatus == TALKING ) {
                        sipStatus = BYE;
                        sipSocket.sendPacket(createBYE() );
                        try {
                                sipSocket.setSoTimeout(SIP_TIMEOUT);
                        } catch (Exception e) {};
                }
        }

        /**
         * envia uma msg cancel
         */
        public void sendCANCEL() {
                if ( ( sipStatus == INVITE ) ||
                           ( sipStatus == BYE )
                        ) {
                        sipStatus = CANCEL;
                        sipSocket.sendPacket(createCANCEL());
                }

                if ( sipStatus == RINGING ) {
                        sipStatus = NONE;
                        sipSocket.sendPacket(createCANCEL());
                }

                closeRtpSession();
        }

        /**
         * envia uma msg ok
         */
        public void sendOK () {
                if ( sipStatus == RINGING ) {
                        sipStatus = OK;
                        //jjr 14/6 abri aki pra testar
                        //openRtpSession();
                        sipSocket.sendPacket(createOK());

                }
        }

        /**
         * envia um ok interno
         * @todo adequar o ok interno ao metodo q o chamou INVITE, CANCEL, BYE
         */
        private void sendOKinterno () {
                sipStatus = OK;
                sipSocket.sendPacket(createOK());
        }

        /**
         * envia um ack
         * @todo adequar o ack interno ao metodo q o chamou INVITE, CANCEL, BYE
```

```
 */
private void sendACK () {
         sipSocket.sendPacket(createACK() );
}

/**
 * envia um ringing
 */
private void sendRINGING() {
         sipSocket.sendPacket(createRINGING());
}

/**
 * envia uma msg transacao cancelada
 */
private void sendTRANSACTION_CANCELED () {
         sipSocket.sendPacket(createTRANSACTION_CANCELED());
}


/**
 * trata o recebimento de msg INVITE
 * faz alguma coisa apenas se o sessao estiver em estado de espera
 */
private void receiveINVITE(String msg) {

         if (sipStatus == NONE) {

                  extractFields(msg);
                  sdp = extractSDP(msg);
                  sipContentLength = sdp.length();
                  sipCSeq_metodo   = "INVITE";
                  sipCSeq_cont     = 1;

                  sendRINGING();
                  sipStatus = RINGING;
                  setFeedback(2,"RINGING\n" + getDestinatario().sipUrl);
         }
}

/**
 * trata o recebimento de msg OK
 * faz alguma coisa apenas se o sessao estiver em estado de espera
 */
private void receiveOK(String msg) {
         switch (sipStatus) {
                  // Se recebeu um OK depois de send um INVITE entao
                  // envia um ack de resposta e abre a sessao rtp
                  case INVITE:
                           sendACK();
                           sdp = extractSDP(msg);
                           openRtpSession();
                           break;

                  // Se recebeu um OK depois de send um BYE entao
                  // fech a sessao rtp
                  case BYE:
                           closeRtpSession();
                           break;

                  // se recebeu um ok depois de send um cancel, fica esperando uma
                  // msg "transacao cancelada"
                  case CANCEL:
                           sipStatus  = TRANSACTION_CANCELED;
                           break;
         }
}

/**
 * trata o recebimento de msg ack
 */
private void receiveACK() {
         switch (sipStatus) {
                  // Se recebeu um ack depois de send um ok entao
                  // abre a sessao rtp
                  case OK:
                           // jjr14.6 desmarcaou aki pra testar
                           openRtpSession();
                           break;

                  // Se recebeu um ACK depois de send um ter recebido um cancel,
                  // ficando em estado de transacao cancelada
                  // deixa a sessao em modo de espera novamente
                  case TRANSACTION_CANCELED:
                           sipStatus = NONE;
                           break;
         }
}

/**
 * trata o recebimento de msg bye
 * @todo ao send o bye eh precisa reconstruir os campos to, from e etc
 * @todo modificar o ok interno pra se adequar ao bye, cancel e etc
 */
 private void receiveBYE() {
         // se a transação tiver em talking envia um ok interno e modificado
         if (sipStatus == TALKING ) {
                  sendOKinterno();
                  closeRtpSession();
         }
 }
```

```
/**
* trata o recebimento de msg cancel
* @todo nao seria melhor deixar logo a sessao em modo de espera??
* @todo cancel tb nao modifica os campos??
*/
private void receiveCANCEL() {
        if (( sipStatus == INVITE  ) ||
                ( sipStatus == BYE     ) ||
                ( sipStatus == RINGING ) ||
                ( sipStatus == TRYING  )
        ) {
                sendOKinterno();
                sendTRANSACTION_CANCELED();
                sipStatus = TRANSACTION_CANCELED;
        }
}

/**
* trata o recebimento de msg transacao cancelada
* @todo nao seria melhor deixar logo a sessao em modo de espera??
*/
private void receiveTRANSACION_CANCELED() {
        if (sipStatus == TRANSACTION_CANCELED ) {
                sendACK();
                sipStatus = NONE;
        }
}

/**
* trata o recebimento de msg ringing
* só faz o feedback pra tela
*/
private void receiveRINGING() {
        if (sipStatus == INVITE) {
                setFeedback(1,"CALLING\n" + getDestinatario().sipUrl);
        }
}

/**
 * Cria os pedidos INVITE, CANCEL, BYE ...
 */
private String createRequest (String request) {

        String msg;

        sipCSeq_metodo = request;

        msg = sipCSeq_metodo + " sip:" +
                sipTo.substring(1, sipTo.length() - 1) +
                " SIP/2.0" + CR;

        msg += "From: " + sipFrom + CR;
        msg += "To: " + sipTo + CR;
        msg += "Call-ID: " + sipCallID + CR;
        msg += "CSeq: " + sipCSeq_cont + " " +  sipCSeq_metodo + CR;
        msg += "Subject: " + sipSubject + CR;
        msg += "Content-Type: " + sipContentType + CR;
        msg += "Content-Length: "+ sipContentLength + CR;
        msg += CR + sdp + CR + CR;

        return msg;
}

/**
 * cria as respostas ok, ack ...
 */
private String createResponse(String response) {

        String corpo = response + CR +
                "From: " + sipFrom + CR +
                "To: " + sipTo + CR +
                "Call-ID: " + sipCallID + CR +
                "CSeq: " + sipCSeq_cont + " " + sipCSeq_metodo + CR + CR;

        return corpo;
}

/**
 * cria uma msg invite
 */
private String createINVITE () {

        sipStatus = INVITE;

        //nr aleatorio de call id
        sipCallID = ( (int) ( Math.random() * new Date().getTime() ) ) +
                                "@" + getRemetente().hostName;

        sipFrom   = "<" + getRemetente().userName +
                                "@" + getRemetente().hostName + ">";

        if (getRemetente().sipPort != SIP_PORT)
                sipFrom += ":" + getRemetente().sipPort;

        sipTo     = "<" + getDestinatario().userName +
                                "@" + getDestinatario().hostName + ">";

        if (getDestinatario().sipPort != SIP_PORT)
                sipFrom += ":" + getDestinatario().sipPort;
```

```
          sipCSeq_metodo  = "INVITE";
          sipCSeq_cont     = 1;
          sipSubject       = "Ligação IP";
          sipContentType   = "application/sdp";
          sdp = createSDP();
          sipContentLength = sdp.length();
          sipContact       = sipTo;

          return createRequest("INVITE");
}

/**
 * cria a msg bye. o bye modifica os campos internos
 */
private String createBYE() {

          sipFrom   = "<" + getRemetente().sipUrl + ">";

          sipTo     = "<" + getDestinatario().sipUrl + ">";

          sipCSeq_metodo   = "BYE";
          sipSubject       = "Ligação IP";
          sipContentType   = "application/sdp";
          sdp = "";//createSDP();
          sipContentLength = sdp.length();
          sipContact       = sipTo;
          sipCSeq_cont++;


          return createRequest ("BYE");
}


/**
 * criaçao de diversos pedidos e respostas
 */
private String createOK() {
          if (sipStatus == OK) {
                    sdp = createSDP();
                    return createResponse("SIP/2.0 200 OK") + sdp + CR + CR;
          } else {
                    return createResponse("SIP/2.0 200 OK");
          }
}

private String createACK()    { return createRequest ("ACK");}
private String createCANCEL() { return createRequest ("CANCEL");}
private String createRINGING() {return createResponse("SIP/2.0 180 RINGING");}
private String createTRYING()  {return createResponse("SIP/2.0 100 TRYING");}
private String createTRANSACTION_CANCELED() {
          return createResponse("SIP/2.0 487 TRANSACTION CANCELED");
}


/**
 * cria campo sdp
 * @todo precisa de dados de payload. verificar sdp
 */
private String createSDP() {

          int i = (int) new Date().getTime();

          sdpV = "v=0" + CR;

          sdpO = "o=" + getRemetente().userName + " " + i +
                       " IN IP4 " + getRemetente().hostAddress.getHostAddress() + CR;

          sdpS = "s=" + sipSubject + CR;

          sdpC = "c=IN IP4 " + getRemetente().hostAddress.getHostAddress() + CR;

          sdpT = "t=0 0" + CR;

          sdpM = "m=audio " + getRemetente().rtpPort +
                       " RTP/AVP " + getRemetente().rtpPayloadType + CR;

          switch (getRemetente().rtpPayloadType) {
                    case (MyRTPSession.ULAW):
                              sdpA = "a=rtpmap 0 PCMU/8000" + CR;
                              break;
                    case (MyRTPSession.ALAW):
                              sdpA = "a=rtpmap 8 PCMA/8000" + CR;
                              break;
                    case (MyRTPSession.GSM):
                              sdpA = "a=rtpmap 3 GSM/8000" + CR;
                              break;
                    case (MyRTPSession.PCM16):
                              sdpA = "a=rtpmap 96 PCM16/8000" + CR;
                              break;
          }

          return sdpV + sdpO + sdpS + sdpC + sdpT + sdpM + sdpA;
}

/**
 * verifica o tipo de linha e testa a integridade da msg
 */
private int parse(String msg){

          if ( msg.indexOf("INVITE") == 0 ) return INVITE;
          if ( msg.indexOf("ACK")    == 0 ) return ACK;
```

```
		if ( msg.indexOf("BYE")    == 0 ) return BYE;
		if ( msg.indexOf("CANCEL") == 0 ) return CANCEL;

		if ( msg.indexOf("SIP/2.0 200") == 0 ) return OK;
		if ( msg.indexOf("SIP/2.0 100") == 0 ) return TRYING;
		if ( msg.indexOf("SIP/2.0 180") == 0 ) return RINGING;
		if ( msg.indexOf("SIP/2.0 487") == 0 ) return TRANSACTION_CANCELED;
		if ( msg.indexOf("SIP/2.0 5")   == 0 ) return SERVER_FAILURE;

		return BAD_RESQUEST;
	}


	private void extractFields(String msg) {

		StringTokenizer msgTokens = new StringTokenizer(msg,CR);

		if (msgTokens.hasMoreTokens() ) {

			String startLine = msgTokens.nextToken();

			if ((startLine.indexOf("INVITE") == 0) ||
				( startLine.indexOf("BYE") == 0)) {

				StringTokenizer lineTokens = new StringTokenizer(startLine," ");

				while (lineTokens.hasMoreTokens() ) {
					sipCSeq_metodo = lineTokens.nextToken();
					sipUrl = lineTokens.nextToken();
					lineTokens.nextToken();
				}
			}

			if ( sipCSeq_metodo.equals("INVITE") ) {

				while (msgTokens.hasMoreTokens() ) {

					String line = msgTokens.nextToken();

					if ( line.indexOf("To:") == 0 ) {
						sipTo = line.substring(5,line.length()-1).trim();
					}

					if ( line.indexOf("From:") == 0 ) {
						sipFrom = line.substring(7,line.length()-1).trim();
					}

					if ( line.indexOf("Call-ID:") == 0 ) {
						sipCallID = line.substring(8,line.length()).trim();
					}

					if ( line.indexOf("Content-Type:") == 0 ) {
						sipContentType = line.substring(13,line.length()).trim();
					}

					if ( line.indexOf("Subject:") == 0 ) {
						sipSubject = line.substring(8,line.length()).trim();
					}

					if ( line.indexOf("Contact:") == 0 ) {
						sipContact = line.substring(8,line.length()).trim();
					}

				}

			}
		}

	}


	private String extractSDP(String msg) {

		StringTokenizer msgTokens = new StringTokenizer(msg,CR);

		if (msgTokens.hasMoreTokens() ) {

			while (msgTokens.hasMoreTokens() ) {

				String line = msgTokens.nextToken();

				if ( line.indexOf("v=") == 0 ) {
					sdpV = line;
				}

				if ( line.indexOf("o=") == 0 ) {
					sdpO = line;
				}

				if ( line.indexOf("s=") == 0 ) {
					sdpS = line;
				}

				if ( line.indexOf("c=") == 0 ) {
					sdpC = line;
				}

				if ( line.indexOf("t=") == 0 ) {
					sdpT = line;
				}
```

```
                                        if ( line.indexOf("m=") == 0 ) {
                                                sdpM = line;

                                                StringTokenizer mTokens = new StringTokenizer(sdpM," ");

                                                while (mTokens.hasMoreTokens() ) {
                                                        // pula m=audio
                                                        mTokens.nextToken();

                                                        //captura porta rtp do destinatario
                                                        getDestinatario().rtpPort =
                                                                Integer.parseInt(mTokens.nextToken());


                                                        // pula RTP/AVP
                                                        mTokens.nextToken();

                                                        //captura o tipo de  do destinatario
                                                        // por enqto aceita apenas 1 payload
                                                        // tem q poder escolher entre varios e
                                                        // se nenhum agradar manda uma mgs NAO ACEITANDO
                                                        getDestinatario().rtpPayloadType =
                                                                Integer.parseInt(mTokens.nextToken());

                                                        getRemetente().rtpPayloadType =
                                                                getDestinatario().rtpPayloadType;

                                                        while ( mTokens.hasMoreTokens() ) {
                                                                mTokens.nextToken();
                                                        }
                                                }
                                        }

                                        if ( line.indexOf("a=") == 0 ) {
                                                sdpA = line;
                                        }
                                }

                        }

                        return sdpV + sdpO + sdpS + sdpC + sdpT + sdpM + sdpA;
                }

}
```

7.1.15 MySIPSocket.java

```
/**
 * MySIPSocket.java
 * @author Jucimar Jr
 *
 * socket udp para recepçao de pacotes sip
 */

import java.net.*;
import java.util.*;
import java.io.*;

public class MySIPSocket extends DatagramSocket implements Runnable{

	final static int MAX_PCT_SIZE = 1500;
	MySIPSession sipSession;

	// armazena a ultima msg enviada
	// usado em caso de retransmissao
	private String lastSipMessage;

	/**
	 * abre o socket usando a sessao sip
	 * inicia a thread
	 */
	public MySIPSocket(MySIPSession sip) throws SocketException{
		super(sip.getRemetente().sipPort);
		sipSession = sip;
		new Thread(this).start();
	}

	/**
	 * recebe os pacotes
	 * @todo enviar msg de ocu
	 */
	private void receivePacket() throws Exception {

		try {
			byte[] sipPayload = new byte[MAX_PCT_SIZE];

			DatagramPacket sipPacket =
				new DatagramPacket(
				sipPayload,
				sipPayload.length
			);

			// recebe o pacote sip
			receive( sipPacket );

			// se o destinatario for nulo cria o usu
			if ( sipSession.getDestinatario() == null ) {

				MyHost destinatario = new MyHost (
					sipPacket.getAddress().getHostName(),
					sipPacket.getPort(),
					0,
					0,
					false
				);

				// altera o destinatario da sessao sip
				sipSession.setDestinatario(destinatario);
			}

			// coloca a msg na fila
			sipSession.sipQueue.putMsg(new String(sipPacket.getData()).trim());

			sipSession.setFeedback(1606, "rec:\n" + new String(sipPacket.getData()).trim() + "\n");

		// se der timeout reenvia a ultima msg
		} catch (InterruptedIOException e) {
			sendPacket(lastSipMessage);
		}
	}


	/**
	 * envia pacotes sip
	 */
	protected void sendPacket(String sipMsg) {

		try {
			if (sipSession.getDestinatario() != null) {
				// cria o pacote sip
				DatagramPacket sipPacket = new DatagramPacket (
					sipMsg.getBytes(),
					sipMsg.length(),
					sipSession.getDestinatario().hostAddress,
					sipSession.getDestinatario().sipPort
				);

				send(sipPacket);

				// captura a msg enviada para possiveis reenvios
				lastSipMessage = sipMsg;

				sipSession.setFeedback(1606,"snd:\n" + new String(sipPacket.getData()).trim()+ "\n");
```

```
                        }
                } catch (IOException ioe) {
                } catch (Exception e) {
                        sipSession.setFeedback(666,"MySIPSocket.sendPacket()\n:" + e);
                }

        }

        /**
         * fica ouvindo a porta sip a espera de pacotes
         */
        public void run() {
                try {
                        while (true) { receivePacket();        }
                } catch (Exception e) {
                        sipSession.setFeedback(666,"MySocket.run()\n:" + e);
                }
        }
}
```

7.1.16 MySpeaker.java

```
/**
 * MySpeaker.java
 *
 * Toca o som capturado pelo socket na cx de som
 *
 * @author Jucimar Jr
 *
 */
import    javax.sound.sampled.*;

public class MySpeaker extends Thread {

          private SourceDataLine line;
          private MyRTPSession   rtpSession;
          private boolean                 isRunning = true;

          /**
           * passa a sessao rtp como parametro
           */
          public MySpeaker(MyRTPSession rtp) throws Exception {

                    AudioFormat           audioFormat;
                    FloatControl   gain;

                    rtpSession = rtp;

                    // define o formato como PCM 16bits 8KHz Mono Big-endian
                    // e captura dados de audio
                    audioFormat = new AudioFormat(8000.0F, 16, 1, true, true);

                    DataLine.Info info =
                              new DataLine.Info(
                                        SourceDataLine.class,
                                        audioFormat
                              );

                    line = (SourceDataLine) AudioSystem.getLine(info);
                    line.open(audioFormat);

                    // define o volume do speaker pro maximo possível
                    // deixando o usuário baixar pela cx de som do micro
                    if (line.isControlSupported(FloatControl.Type.MASTER_GAIN/*VOLUME*/)) {
                       gain = (FloatControl) line.getControl(FloatControl.Type.MASTER_GAIN);
                       gain.setValue(gain.getMaximum());
                    }
          }

          // inicia a linha e a thread
          public void start() {
                    line.start();
                    super.start();
          }

          /**
           * fecha a linha e dah uma lipeza geral nela
           * seta isRunning para false pra sair da thread
           */
          public void close() {
                    line.stop();
                    line.flush();
                    line.close();

                    isRunning = false;
          }

          /**
           * a line captura os pedaços de som q tao na fila de msgs e toca eles
           * na cx de som com volume max
           */
          public void run() {
                    try {
                              while ( isRunning ) {
                                        //tira os pedacos de som da fila e converte eles pra PCM
                                        byte[] soundPacket =
                                                  MyCodec.convertToPCM16(
                                                            (byte[]) rtpSession.queueReceiver.getMsg(),
                                                            rtpSession.rtpRemetente.rtpPayloadType
                                                  );

                                        rtpSession.setFeedback(7,"SPEAKING");
                                        line.write(soundPacket, 0, soundPacket.length);
                                        rtpSession.setFeedback(8,"SPEAKING");
                              }
                    } catch (Exception e) {
                              rtpSession.setFeedback(666,"MySpeaker.run():\n" + e);
                    }
          }
}
```