Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Implementação de um Sistema de Iniciação de Sessão Multimídia para a Plataforma Linux

Autor: Francisco Helder Candido dos Santos Filho
Orientador: Prof. Dr. Luís Geraldo P. Meloni

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: **Engenharia Elétrica**.

Banca Examinadora

Prof. Luís Geraldo P. Meloni, Dr. (Orientador) ..... FEEC/Unicamp
Prof. Dalton Soares Arantes, Dr. ..... FEEC/Unicamp
Prof. Leonardo de S. Mendes, Dr. ..... FEEC/Unicamp
Paulo Batista Lopes, Ph.D. ..... Freescale

Campinas, SP
Julho/2005

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

Sa59i

Santos Filho, Francisco Helder Candido dos
Implementação de um sistema de iniciação de sessão multimídia para a plataforma Linux/ Francisco Helder Candido dos Santos Filho. – Campinas, SP:[s.n.], 2005.

Orientador: Luís Geraldo Pedroso Meloni.
Tese (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1.Linux (Sistema operacional de computador) 2. sistema multimídia. 3. TCP/IP (Protocolo de rede de computação). 4. Redes de computação - Protocolos.I. Meloni, Luís Geraldo Pedroso II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título

Resumo

Esta dissertação trata da modelagem e desenvolvimento de um sistema de voz sobre IP (VoIP) para multiplataforma, utilizando o protocolo SIP para o estabelecimento de sessão. O protocolo de iniciação de sessão (SIP), utilizado no desenvolvimento deste sistema, é um protocolo de sinalização e controle de chamadas entre dois ou mais participantes, que tem ganhado uma grande aceitação por parte da comunidade de telecomunicações. O sistema RT-DSPhone foi desenvolvido para o Linux e o μClinux, que é um sistema operacional baseado no Linux para dispositivo embarcado. O sistema aqui idealizado RT-DSPhone, foi desenvolvido usando-se a linguagem C, juntamente com as APIs da biblioteca oSIP e ferramentas para desenvolvimento em sistemas embarcados, a fim de possibilitar a criação de um sistema de comunicação flexível e robusto. Também é descrito a arquitetura e o funcionamento do protocolo SIP, utilizado para o desenvolvimento da sinalização do RT-DSPhone. É ainda detalhado o desenvolvimento do sistema que permite a criação, controle e finalização de sessões multimídia entre dois participantes através do protocolo SIP. Por fim, são realizados testes de modo a se avaliar a capacidade de interoperabilidade e análise de desempenho do sistema implementado.

Palavras-chave: VoIP, SIP, Linux, μClinux, sistema embarcado.

Abstract

This dissertation deals with modeling and development of Voice over IP (VoIP) system for multiplatform, using the SIP protocol to accomplish the session. The Session Initiation Protocol (SIP), used in the development of this system, is a protocol for creating, modifying and terminating calls between two or more participants, and it has received great support of the communications community. The RT-DSPhone system was developed for Linux and μClinux, which is a operating system for embedded system. The RT-DSPhone system was developed using the C language, the APIs of oSIP library, and tools for embedded systems, for the creation of a robust and flexible communication system. It is also described the architecture and operation of the SIP protocol used in the development of the RT-DSPhone signaling. The development of a system that allows the creation, control and finalization of multimedia sessions between two users through the SIP protocol is also detailed. Finally, some tests are carried out in order to evaluate the interoperability capacity of the developed system.

Keywords: VoIP, SIP, Linux, μClinux, embedded system.

Aos meus pais, irmão, avó e tios

Agradecimentos

Agradeço a Deus em quem confio e está sempre presente em minha vida, guiando meu caminho nesta longa estrada da vida, pois sem Ele não teria superado mais esse obstáculo.

Aos meus pais, pela educação, pelo amor, carinho e apoio incondicional depositados em mim.

Às minhas irmãs, que sempre torceram pelo meu sucesso profissional, em especial a minha irmã Cely e seu marido Daniel, pois sempre me apoiaram e incentivaram.

Agradeço ao meu orientador Professor Meloni, pelo apoio e confiança.

Aos demais colegas de pós-graduação Rodrigo, Lívio, Euler, Paulo, Sérgio, Marcia, Gilmar, Márzio, Jaqueline, Fábio, em especial ao amigo Glauco pelo apoio e ao amigo Helcio pelas críticas e sugest ões. Companheiros de todos os momentos e que contribuíram de forma tão especial, cada um a seu modo, para a conclusão deste trabalho.

Aos meus colegas de trabalho e, principalmente, aos amigos Schammass, Denise e Henrique pelas críticas e sugestões. Especialmente aos colegas de trabalho Paulo Luz e Júlio Costa pelas sugestões e ensinamentos.

Finalmente, agradeço a Motorola e em particular ao Fábio e ao Paulo, por terem cedido seus recursos a realização deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

API	-	Application Programming Interface
CRLF	-	Carriage-Return Line-Feed
DNS	-	Domain Name System
GPL	-	The GNU General Public License
HTTP	-	Hyper Text Transfer Protocol
ISDN	-	Integrated Services Digital Network
IETF	-	Internet Engineering Task Force
ITU	-	International Telecommunication Union
LAN	-	Local Area Network
MCU	-	Multipoint Control Unit
MMU	-	Memory Management Unit
PDU	-	Protocol Data Unit
POSIX	-	Portable Operating System Interface for Unix
PSTN	-	Public Switched Telephone Network
QoS	-	Quality of Service
RFC	-	Request For Comment
RTCP	-	Real-Time Control Protocol
RSTP	-	Real-Time Stream Protocol
RTP	-	Real-Time Transport Protocol
SDL	-	Specification and Description Language

SDP - Session Description Protocol

SIP - Session Initiation Protocol

SMTP - Simple Mail Transfer Protocol

SP - Simple Space

TCP - Transmission Control Protocol

TFTP - Trivial File Transfer Protocol

ToS - Type of Service

TTL - Time To Live

TPC - Technical Program Committee

TU - Transaction User

UA - User Agent

UAC - User Agent Client

UAS - User Agent Server

UDP - User Datagram Protocol

URI - Uniform Resource Identifier

URL - Uniform Resource Locator

VoIP - Voice over IP

Capítulo 1

Introdução

Nos últimos anos vários estudos vêm sendo realizados no tema de multimídia para Internet, buscando a transmissão de áudio, vídeo e dados. Esse tema tem despertado o interesse de profissionais da área de redes de computadores e dos usuários em geral, principalmente em relação a serviços ou transmissão de voz sobre IP, ou de forma mais restrita, de telefone IP, que assume maior importância nesse cenário.

Com o crescimento da Internet, verificou-se uma expansão na utilização de computadores e dispositivos embarcados ligados à rede, para explorar a convergência de serviços como áudio e vídeo juntamente com o tráfego de dados. Estes serviços têm como objetivo poupar tempo e encurtar distância, bem como aumentar a eficiência na utilização de recursos.

Os sistemas informatizados estão crescendo mais e mais, e dentre estes sistemas, os embarcados vêm ganhando uma popularidade imensa, estando cada vez mais presentes no dia a dia das pessoas, na forma de produtos fixos ou móveis de uso pessoal ou de consumo e que apresentam freqüentemente, entre outras características, alguma forma de comunicação de rede e um elevado grau de sofisticação em relação às tarefas que executam (dispositivos inteligentes). É inevitável que o número de dispositivos embarcados irá continuar a crescer rapidamente [Barr, 1999]. Uma outra característica importante dos sistemas embarcados, em particular os de uso pessoal ou de consumo, é que muitos estão sendo projetados para apresentar funcionalidade variada, em contraste com a visão tradicional de tais sistemas.

Atualmente, é inegável a grande participação do sistema operacional Linux em vários segmentos. Inicialmente voltado para servidores, cada vez mais o Linux vem se popularizando como opção efetiva para *Desktops*, PDAs, dentre outras. A popularização do Linux em sistemas embarcados (ou sistemas embutidos) está também ligada ao aparecimento de uma variante do núcleo denominada de μClinux (pronuncia-se micro-Clinux).

Durante anos, companhias e organizações utilizam um conjunto limitado de serviços de comuni-

cações, como o telefone e o fax, suportado pela rede telefônica tradicional e uma rede de pacotes para transporte de dados. Atualmente têm a oportunidade de utilizar a rede IP como única infra-estrutura para as comunicações, permitindo a integração de novos serviços que possibilitam novas formas de comunicação e a redução de custos, tornando-as mais competitivas.

A convergência de serviços tem sido tentada por várias vezes na história das comunicações, mas apenas a Internet oferece a primeira expectativa real na unificação de todos os serviços de comunicações numa única rede e num único sistema terminal [Schulzrinne and Rosenberg, 2000].

Ao contrário da telefonia tradicional, as comunicações IP permitem além do transporte de voz, a integração de vídeo, dados e de novos serviços como *chat*, mensagens instantâneas e *Web*, numa única infra-estrutura e num único serviço.

Atualmente existem dois protocolos para as comunicações de sinais multimídia em tempo real, o SIP da *Internet Engineering Task Force* (IETF) e o H.323 da *International Telecommunications Union* (ITU). Estes dois protocolos são utilizados para encaminhamento, sinalização e controle de chamadas e outros serviços suplementares. O H.323 é um protocolo já estabelecido e largamente utilizado devido principalmente a ter dado provas da sua capacidade e à interoperabilidade com a rede telefônica pública comutada. O SIP é um protocolo recente que promete escalabilidade, flexibilidade e facilidade na criação de serviços. Embora não sejam interoperáveis estes dois protocolos têm seguido uma tendência de aperfeiçoamento do seu funcionamento, onde se observa o aumento de suas semelhanças cada vez que é publicada uma nova versão.

1.1 Objetivo deste Trabalho

O objetivo deste trabalho foi o estudo e desenvolvimento de um sistema VoIP leve e flexível, que rode no sistema operacional Linux e μClinux. O sistema disponibiliza comunicação entre dois computadores, ou dois dispositivos embarcados, ou um dispositivo embarcado e um computador, utilizando o protocolo de sinalização SIP para o estabelecimento e controle das sessões dos vários meios de informações. Para atingir esse objetivo foi necessário realizar um conjunto auxiliar de tarefas, tais como, estudo geral desta tecnologia, análise do dispositivo embarcado adequado para a aplicação, preparo de um ambiente de compilação da placa alvo e geração do *kernel* do μClinux juntamente com a aplicação SIP desenvolvida.

Após a criação do RT-DSPhone foram realizados vários testes para analisar o nível de interoperabilidade dos sistemas, os quais incluiram também a execução de ensaios com outras aplicações SIP e testes de desempenho.

1.2 Estrutura da Dissertação

Esta dissertação está estruturada em oito capítulos. A apartir desta introdução, no segundo capítulo é feita uma especificação e descrição do protocolo SIP. São descritas as funcionalidades do protocolo, seus componentes, mensagens e os requisitos necessários ao nível da sinalização. É feita também uma descrição do protocolo SDP, usado pelo protocolo SIP para negociar o tipo, entre outras características, do codec a ser utilizado na comunicação.

No terceiro capítulo é apresentada uma visão geral do protocolo H.323, apresentando-se as suas características principais e sua sinalização. Esse capítulo tem como objetivo principal a comparação com o protocolo SIP.

No quarto capítulo é analisada a qualidade de serviços no sistema de voz sobre IP, suas principais características e algumas técnicas importantes para garantir a qualidade de serviço na Internet.

No quinto capítulo são apresentadas as ferramentas utilizadas e as fases seguidas para gerar e executar o sistema operacional μClinux no dispositivo embarcado utilizado neste trabalho, como também a seqüência seguida para execução da aplicação no μClinux.

O sexto capítulo abrange todo o processo de implementação do serviço de VoIP, sendo apresentadas as funcionalidades encontradas no sistema RT-DSPhone, como também uma descrição e especificação da arquitetura do sistema. Para isso foi utilizada a linguagem conhecida por SDL [Ellsberger et al., 1997], que é uma ferramenta usada para especificar e descrever sistemas de comunicações.

No sétimo capítulo são apresentados testes, realizados de modo a avaliar o nível de interoperabilidade através de critérios de avaliação definidos pelo *Technical Program Committee* (TPC) e também testes de comunicação com outros softwares baseados no SIP.

Finalmente, no oitavo capítulo são apresentadas as conclusões relativas aos objetivos propostos, e discutidas as perspectivas de trabalhos futuros.

Capítulo 2

Protocolo de Iniciação de Sessão - SIP

O Protocolo de Iniciação de Sessão (SIP) é um protocolo de sinalização para iniciar, manter e terminar uma sessão de áudio, vídeo ou texto entre dois usuários finais, utilizando o Protocolo de Descrição de Sessão (SDP) para negociar essa sessão multimídia. A Sessão SIP envolve dois ou mais participantes, e pode usar comunicação *unicast* ou *multicast*. O SIP foi desenvolvido por um grupo de trabalho da *Internet Engineering Task Force* (IETF) e foi publicado primeiramente na RFC 2543 [Rosemberg et al., 1999] e atualizado na RFC 3162 [Rosemberg and Schulzrinne, 2002]. O SIP é um protocolo baseado no protocolo base do serviço de e-mail *Simple Mail Transfer Protocol* (SMTP) RFC 821 [Postel, 1982] e no protocolo base da Web *Hyper Text Transfer Protocol* (HTTP) RFC 2616.

O SIP é um protocolo de texto que reutiliza várias propriedades do HTTP, baseando-se no modelo cliente/servidor, onde o cliente faz o pedido e o servidor retorna uma resposta ao pedido do cliente. O SIP pode ser estendido para acomodar características e serviços, tais como serviços de controle de chamadas, mobilidade, interoperabilidade com sistemas de telefonia existente e mais. O SIP tem uma infraestrutura de rede, no qual pode habilitar a criação de várias funcionalidades, como registro de usuário, convite para sessão e outros tipos de pedidos [CIS, 2000] [Radvision, 2001].

2.1 Funcionalidades do SIP

SIP é um protocolo de controle atuando na camada de aplicação, que trata sessão multimídia (conferências) como chamadas telefônicas pela Internet. O SIP permite o convite de participantes para sessões existentes, como conferências *multicast*, suportando cinco modos de estabelecimento e término de comunicações multimídia:

- **Localização de Usuário**: determinação do usuário final para o estabelecimento da comunicação;

- **Disponibilidade do Usuário**: determina a disponibilidade do participante da sessão em juntar-se à comunicação;
- **Capacidade do Usuário:** determinação do meio e parâmetros a serem usados;
- **Configurações da Sessão:** determina o estabelecimento dos parâmetros da sessão para ambos os participantes;
- **Gerenciamento de Sessão:** envolve a transferência e término da sessão, modificando os parâmetros e invocando serviços.

O SIP é um componente que pode ser usado com outros tipos de protocolos da IETF para construir uma arquitetura multimídia completa, incluindo entre estes protocolos o *Real-Time Transport Protocol* (RTP) RFC 1889 [Schulzrinne et al., 1996] para transporte de dados em tempo real, o *Real-Time Stream Protocol* (RTSP) RFC 2326 [Schulzrinne et al., 1998] para controlar a entrega de pacotes de media e o *Session Description Protocol* RFC 2327 [Handley and Jacobson, 1998] para descrição de sessão multimedia, mostrando que o SIP pode ser usado em conjunto com outros protocolos, oferecendo um completo serviço para os usuários.

2.2 Estrutura do Protocolo SIP

O SIP é um protocolo estruturado em camadas, o que significa que seu comportamento é descrito em termos de um conjunto de estágios independentes, que trabalham em conjunto para prover suas funcionalidades. O protocolo SIP é subdividido em três camadas básicas: camada de transporte (*Transport Layer*), camada de transação (*Transaction Layer*) e camada de transação do usuário (*Transaction User Layer*).

A camada inferior do protocolo SIP é a camada de transporte. Ela define como o cliente envia pedidos e recebe respostas e como o servidor recebe pedidos e envia respostas sobre a rede. A camada de transporte é responsável pelo gerenciamento de conexão para protocolos de transportes como TCP, UDP e estouro de tempo de transação. Estas conexões são indexadas por um conjunto formado por endereço, porta e protocolo de transporte. Todo elemento SIP pode implementar conexão tanto UDP, quanto TCP.

A segunda camada é a camada de transação. Esta camada é um componente fundamental do protocolo SIP. Uma transação é um pedido enviado por uma transação do cliente para uma transação do servidor e também as respostas para aquele pedido enviado de uma transação do servidor para o cliente. A camada de transação trata as retransmissões das mensagens, verificando as repostas para os pedidos e tratando o estouro do tempo de transação.

A camada superior do protocolo SIP é chamada de transação do usuário (TU). Esta camada é

a responsável pela interação com o usuário, quando em TU se quer enviar um pedido, cria-se uma instância da transação do usuário e passa-se o pedido com o endereço IP, porta e transporte para que seja enviado, podendo-se também cancela-lo.

Uma rede SIP é composta de quatro tipos de entidades lógicas. Cada entidade tem funções específicas e pode participar de uma comunicação SIP como um cliente (iniciando pedido), ou como um servidor (respondendo o pedido). Um dispositivo físico pode ter mais que uma entidade lógica. A seguir definimos quatro entidades lógicas SIP:

***User Agent*:** No SIP, um *User Agent* (UA) é a entidade final. *User Agents* iniciam e terminam sessões pelas trocas de mensagens de pedidos e respostas. RFC 3162 [Rosemberg and Schulzrinne, 2002] define o *User Agent* como uma aplicação, que contém tanto o *User Agent Client*, como o *User Agent Server*, como mostrado abaixo:

- ***User Agent Client* (UAC)** - Uma aplicação cliente que inicia um pedido SIP;
- ***User Agent Server* (UAS)** - Uma aplicação servidor que recebe o pedido SIP e retorna uma resposta.

***proxy Server*:** Um *proxy* é uma entidade que atua tanto como servidor ou como cliente para o propósito de intermediar os dois usuários finais. O *proxy* pode passar adiante um pedido sem nenhuma mudança para seu destino final ou pode alterar alguns parâmetros antes de passar o pedido. Além disso, pode até mesmo decidir enviar uma resposta gerada localmente [Radvision, 2002].

***Redirect Server*:** Um servidor de redirecionamento aceita o pedido SIP, mapeia o endereço SIP da chamada ou o mais novo endereço e o retorna para o cliente. Diferentemente do *proxy*, o servidor de redirecionamento não passa o pedido para outros servidores [Hyun et al., 2002].

***Registrar*:** Um registrador é um servidor que aceita pedido de REGISTER. Os registradores são necessários para manter-se informado da localização atual de um usuário. O endereço IP de um usuário pode mudar sob várias circunstâncias, conexão por meio de um provedor de Internet que fornece endereços dinâmicos ou um usuário móvel. Para ser capaz de alcançar esse usuário a partir de seu endereço SIP, uma entidade na rede SIP precisa manter o mapeamento entre endereços SIP e endereços IP.

2.3 Mensagens SIP

Mensagens SIP são codificadas usando a sintaxe da mensagem HTTP/1.1. O conjunto de caracteres é ISO 10646 com codificação UTF-8 [Yergeau, 1998]. Uma mensagem SIP é tanto um pedido de um cliente para um servidor, quando uma resposta de um servidor para um cliente. Os dois tipos de mensagens consistem de uma linha inicial, um ou mais campos de cabeçalho, uma linha vazia

indicando o fim dos campos de cabeçalho e um campo mensagem de corpo, que é opcional, como mostrado na Figura 2.1.

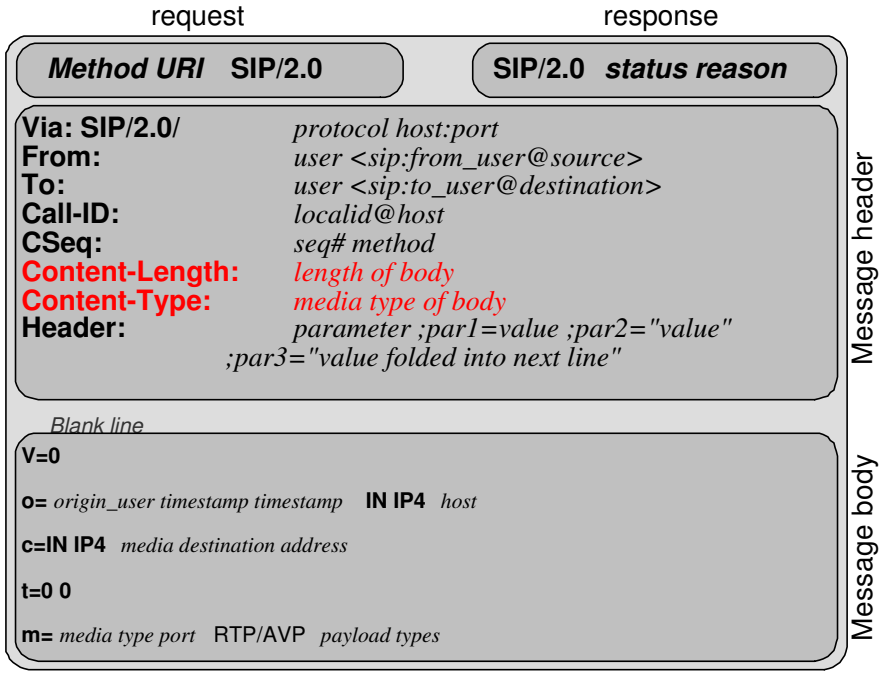

Figura 2.1: Formato da resposta/mensagem SIP.

A linha inicial, cada linha de campo de cabeçalho e a linha vazia pode ser terminada por uma seqüência de *Carriage-Return Line-Feed*(CRLF).

Alguns campos do cabeçalho estão presentes tanto nos pedidos quanto nas respostas. Esses campos são partes do 'cabeçalho geral':

- **CALL-ID**: O parâmetro CALLID serve para identificar os pedidos com as respostas correspondentes. A primeira parte do CALLID deve ser um padrão único dentro de cada computador e a última parte, um nome de domínio ou o endereço IP de uma máquina, tornando-o globalmente único. Um novo CALLID deve ser gerado para cada chamada;
- **Cseq**: Cada pedido tem que ter um campo de cabeçalho Cseq, composto por um número (sem sinal) seqüencial e pelo nome do método. Dentro de uma chamada, o número seqüencial é incrementado a cada novo pedido e começa com um novo valor aleatório;
- **From**: Este campo deve estar presente em todos os pedidos e repostas. tal campo contém um nome opcional (helder) e o endereço SIP (sip:user@143.106.50.220) do originador do pedido;

- **TO**: Esse campo deve estar presente em todos os pedidos e respostas, indicando o destino pretendido de um determinado pedido. tal campo é composto de um nome (filho), seguido pelo endereço SIP (sip:user@143.106.50.218), sendo simplesmente copiado na resposta;
- **Via**:O campo Via é usado para gravar a rota de um pedido, de maneira a permitir que servidores SIP intermediários retransmitam as respostas ao longo do mesmo caminho. Para conseguir isso, cada *proxy* adiciona um novo campo Via com o seu próprio endereço à lista de campos Via existentes;
- **Content-Type**: permite descrever o tipo do corpo da mensagem. No exemplo, o corpo da mensagem contém a descrição da sessão, usando o protocolo IETF SDP [Handley and Jacobson, 1998];
- **Content-lenght**: Contém o número de octetos do corpo da mensagem.

2.3.1 Pedidos

Pedidos SIP são diferenciados das respostas SIP por ter um *Request-Line*, ao invés de um *start-line*. Um *Request-Line* composto do nome do método, um *Request-URI* e a versão do protocolo separados por um caracter de espaço simples (SP). O *Request-Line* termina com um CRLF. O CR ou LF não é permitido, exceto o CRLF no final de linha, também nenhum espaço em branco linear é permitido em algum dos elementos.

Request-Line = Método SP Request-URI SP SIP-Version CRLF

Figura 2.2: Formato de uma mensagem SIP.

Os métodos são classificados em seis modos diferentes, dependendo do tipo de pedido que foi gerado, como apresentado abaixo:

- **INVITE**:O pedido INVITE é usado para iniciar uma sessão;
- **OPTIONS**: Um cliente envia um pedido OPTION ao servidor para saber suas capacidades. O servidor envia de volta uma lista com os métodos que suporta;
- **REGISTER**: Clientes devem registrar sua localização atual (um ou mais endereços) com o pedido REGISTER. Um servidor SIP capaz de aceitar uma mensagem REGISTER é chamado *registrar*;
- **CANCEL**: Um pedido CANCEL pode ser enviado para interromper um pedido que foi enviado anteriormente , enquanto o servidor ainda não tiver enviado uma resposta final;

- **BYE**: Um pedido BYE é enviado pelo usuário de origem ou pelo usuário de destino para terminar uma chamada;
- **ACK**:Um pedido ACK é enviado pelo usuário para confirmar que recebeu uma resposta final do servidor, ou do usuário final, como um 200 OK.

2.3.2 Respostas

Respostas SIP são diferenciadas dos pedidos por terem um *Status-Line*, ao invés de um *start-line*. O *Status-Line* é composto da versão do protocolo, seguido de um número de *Status-Code* e sua frase textual associada, com cada elemento separado por um caracter de espaço simples (SP).

Dependendo do código de *status*, pode haver campos de cabeçalho adicionais e parte dos dados de resposta podem estar vazios, ou ela pode conter dados SDP ou texto explicativo. Até agora foram definidas seis categorias de códigos de *status*, dependendo do primeiro dígito. O primeiro dígito do *Status-Code* define a classe da resposta. Os últimos dois dígitos não têm nenhuma regra específica. Por esse motivo, qualquer resposta com status de 100 a 199 refere-se a uma resposta "1xx *response*", assim como qualquer resposta com status de 200 a 299 refere-se a uma "2xx *response*", como definido abaixo:
1xx: Provisória - pedido recebido, continuando a processar o pedido.

- 100 Tentando;
- 180 Chamando;
- 181 A chamada está sendo retransmitida;
- 182 Coloca na fila.

2xx: Sucesso - a ação foi recebida, entendida e aceita com sucesso.

- 200 OK.

3xx: Redirecionamento - Uma ação adicional deve ser tomada para completar o pedido.

- 300 Múltiplas escolhas;
- 301 Movido permanentemente;
- 302 Movido temporariamente;
- 380 Serviço alternativo.

4xx: Erro de Cliente - O pedido contém sintaxe inválida ou não pode ser efetuado neste servidor.

- 400 Pedido inválido;

- 401 Não autorizado;
- 402 Necessário pagamento;
- 403 Proibido.

5xx: Erro de servidor.

- 500 Erro interno ao servidor;
- 501 Não implementado;
- 502 Gateway inválido;
- 503 Serviço não disponível;
- 504 Tempo esgotado no gateway;
- 505 Versão SIP não suportada.

6xx: Falha global.

- 600 Ocupado em todos os lugares;
- 603 Declínio;
- 604 Não existe em lugar nenhum;
- 606 Não aceitável.

2.4 Comportamento Geral do Terminal SIP

Um *User Agent* representa um sistema final SIP, contendo um *User Agent Client* (UAC), que gera pedidos e um *User Agent Server* (UAS), que gera respostas para os pedidos realizados. Um UAC é capaz de gerar um pedido baseado em um estímulo, gerado por uma ação externa, como o clique de um botão. Um UAS é capaz de receber um pedido e gerar uma resposta baseada na entrada de um usuário, uma ação externa, dando como resultado a execução de um programa, ou alguma outra reação.

Quando um UAC envia um pedido, o pedido passa por um certo número de servidores *proxy*, que repassam o pedido, até chegar ao destino final representado pelo UAS. Quando o UAS gera uma resposta para aquele pedido, esta é encaminhada para o UAC e o pedido é tratado dependendo de dois fatores: primeiro, se o pedido representa ou não um diálogo e segundo, se está baseado no método do pedido.

2.4.1 Gerando o Pedido

Um pedido SIP válido gerado por um UAC deve conter no mínimo os seguintes campos de cabeçalhos: To, From, Cseq, Call-ID, Max-Forwards e Via; todos esses campos de cabeçalhos são necessários em toda mensagem SIP. Estes seis campos de cabeçalhos são fundamentais para a construção de uma mensagem SIP, provendo muitas das mensagens críticas para o roteamento dos serviços, incluindo o endereçamento de mensagens, o roteamento de respostas, limitação na propagação de mensagens, ordenamento de mensagens e a identificação única da transação. Estes campos de cabeçalhos têm um certo grau de importância, como o método, o Request-URL e a versão SIP contidos da primeira linha da mensagem.

2.4.2 Enviando o Pedido

O destino para o pedido é calculado. Caso não exista uma política local de especificação, então o destino deve ser determinado aplicando o procedimento DNS descrito em [Handley and Jacobson, 1998] como segue. Se o primeiro elemento no conjunto de rota indica uma rota estrita, o procedimento deve ser aplicado para o **Request-URI** do pedido, caso contrário o procedimento é aplicado para o primeiro valor de campo de cabeçalho de rota no pedido, ou para o **Request-URI** do pedido, se o campo de cabeçalho de rota não está presente. Este procedimento gera um conjunto ordenado de endereço, porta e transportes para tentativas de conexão. Independente de qual URI é usado como entrada para o procedimento de [Rosemberg et al., 1999], se o **Request-URI** especifica um recurso SIPS (que é um SIP seguro), o UAC deve seguir o procedimento de [Rosemberg et al., 1999] como se a entrada fosse um URI SIPS.

Políticas locais devem especificar um conjunto alternativo de destino para tentativas de conexão. Se o **Request-URI** contém um URI SIPS, algum destino alternativo deve ser contatado com TLS. Não existem restrições no destino alternativo, se o pedido não contém campo de cabeçalho de rota. Isto provê uma alternativa simples para um conjunto de rota pré-existente como o caminho para especificar um *proxy* fora da rota. Portanto, aproximação para configurar um *proxy* fora da rota não é recomendada.

O UAC deve seguir o procedimento definido em [Handley and Jacobson, 1998] para elementos completos, tentando-se cada endereço até um servidor ser contatado. Cada tentativa constitui uma nova transação, e portanto, carrega diferentes campos de cabeçalho **Via** com um novo parâmetro *branch*.

2.4.3 Processando uma Resposta

As respostas são processadas primeiro pela camada de transporte e então passadas para cima para a camada de transação. A camada de transação faz seu processamento e passa a resposta para cima na **TU**. A maioria das respostas processadas na **TU** é um método específico. Portanto, existem alguns comportamentos gerais independentes do método.

2.5 Protocolo de Descrição de Sessão - SDP

O SDP é um protocolo de descrição de sessão para multimídia. O propósito do SDP é carregar informações a respeito do modo de sessão multimídia que os participantes devem negociar para realizar uma comunicação. O SDP é primariamente usado em uma interconexão, porém sendo um protocolo geral, este pode descrever conferências em outro ambiente de rede.

Uma sessão multimídia, para esse propósito é definida como um conjunto de fluxo de mídia que existe em uma certa duração de tempo; um fluxo de mídia pode ser de muitos-para-muitos. O tempo da sessão não precisa ser contínuo. O protocolo SDP inclui em sua mensagem campos que descrevem:

- Nome e propósito da sessão;
- Tempo(s) que a sessão está ativa;
- Informações para receber a mídia (endereço, portas, formatos, etc);
- Informações a respeito do comprimento da banda a ser usado pela conferência;
- Informações de contato da pessoa responsável pela sessão.

Em geral o protocolo SDP contém informações suficientes para que os usuários sejam capazes de entrar em uma sessão e informar os recursos a serem usados pelos participantes da sessão.

As informações de mídia para o protocolo SDP, incluem:

- O tipo de mídia (audio, vídeo, etc);
- O protocolo de transporte [Schulzrinne, 1996](RTP/UDP/IP, H.320, etc);
- O formato de mídia (H.261 vídeo, MPEG vídeo, etc).

Para uma sessão multicast, são incluídos os seguintes:

- Endereço multicast para a mídia;
- porta para transporte da mídia.

Este endereço e porta são endereço de destino e porta de destino do fluxo *multicast*, tanto para transmissão, quanto para recepção ou ambos.

Para uma sessão *unicast* IP, são incluídos os seguintes campos:

- Endereço remoto para mídia;
- porta de transporte para a mídia.

A semântica para este endereço e porta depende da mídia e protocolo usado. Como padrão, o endereço e porta remotos são usados para receber dados. Portanto, algumas mídias podem decidir usar este protocolo para estabelecer um controle de canal para fluxo de mídia atual.

O SIP usa o protocolo de descrição de sessão - SDP para sessões multimídia. O SDP é inteiramente textual usando o conjunto de caracter ISO 10646 no código UTF-8. A forma textual em oposição a codificação binária, como um ASN/1 ou XDR, foi escolhida para aumentar a portabilidade, sendo possível incluir uma variedade de transportes a serem usadas e permitindo maior flexibilidade, pois as ferramentas baseadas em texto podem ser usadas para gerar e processar a descrição de sessão. Uma descrição de sessão SDP consiste de um número de linhas de texto da forma:

<type>=<value>

onde <type> é exatamente um caractere diferenciável entre maiúscula e minúscula e <value> é uma string de texto estruturado, no qual o formato depende do <type>. Espaço em branco não é permitido nos dois lados do sinal '='.

O principal objetivo do SDP é a negociação do codec entre os participantes de uma sessão multimídia. Isso acontece quando o SIP está realizando uma sinalização para a iniciação de uma sessão multimídia. A descrição de sessão é estruturada em uma seção que se aplica à sessão toda (começando com v=...) e várias seções de mídia (cada uma começando com m=...). Os parâmetros nas seções de mídia podem suprimir os parâmetros padrões da seção do nível de sessão. Os campos da mensagem **SDP** são divididos da seguinte maneira:

Descritores de Sessão

v= versão do protocolo;
o= identificação do requisitante da sessão;
s= nome da sessão;
i= informação sobre a sessão;
u= página da Internet que contenha a descrição da sessão;
e= endereço de e-mail do requisitante;

p= número do telefone do requisitante;

c= informações sobre a conexão;

b= largura de banda exigida pela sessão;

z= ajuste do relógio entre as localidades;

k= chave criptográfica;

a= atributos da sessão.

Descritores de Tempo de Sessão

t= tempo em que a sessão vai estar ativa;

r= número de repetições do tempo em que a sessão pode estar ativa.

Descrição da Mídia

m= nome da mídia e endereço de transporte;

i= título da mídia;

c= informações sobre a conexão;

b= informações sobre a largura de banda;

k= chave criptográfica;

a= atributos da mídia;

Apesar do protocolo suportar todos esses campos, somente cinco desses campos são obrigatórios na mensagem SIP.

2.6 Funcionamento Geral do Protocolo SIP

O SIP é um protocolo que permite muitas funcionalidades para tratamento de chamadas. Serão apresentadas algumas funcionalidades do protocolo SIP, como o *proxy*, o registro, localização e mobilidade de um usuário e o estabelecimento de uma sessão SIP.

2.6.1 Registrando um Usuário SIP

O servidor de registro, consiste de um servidor que aceita um pedido de registrar e guarda as informações do terminal que envia o pedido para um servidor de localização, na sua área de domínio de rede. O pedido REGISTER é gerado por um usuário com objetivo de estabelecer ou remover um mapeamento do endereço SIP, no qual pode ser contactado.

Quando um servidor de registro recebe um pedido REGISTER, os seguintes passos acontecem:

1. O usuário gera uma requisição REGISTER, o servidor confere o URL do pedido, para determinar se esse pedido tem ligação com um domínio em sua base de dados;

2. O servidor extrai o endereço do campo do cabeçalho **To**. Se o endereço de registro não for um endereço válido, o servidor envia uma resposta 404 (não encontrado);
3. O servidor checa se o pedido contém o campo **Contact** e um campo **Expires**;
4. O servidor agora processa cada endereço de contato e determina o intervalo de encerramento do registro;
5. O servidor retorna uma resposta 200 (OK). A resposta pode conter valor para o campo **Contact** enumerando todas as ligações correntes. Cada valor **Contact** atribui um parâmetro "expires", indicando o tempo de expiração do registro escolhido pelo servidor.

Na Figura 2.3 tem-se um exemplo de um terminal se registrando no servidor de registro de domínio:

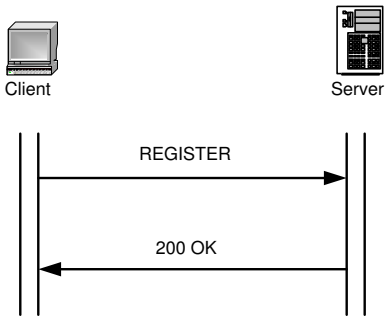

Figura 2.3: Serviço de registrar um terminal.

2.6.2 Localização e Mobilidade do Usuário

O SIP define um modo de localizar um usuário final a partir de seu nome, que é chamada URL SIP. Isso é feito do seguinte modo:

- Primeiro a URL SIP permite ao originador da chamada localizar um servidor SIP, o qual será o destino da mensagem INVITE inicial. O servidor SIP pode ser o destino final da chamada, ou deve saber o endereço de transporte do ponto de destino da chamada;
- Se o servidor SIP não for o destino final da chamada, este vai redirecionar o pedido INVITE para o usuário final da chamada. Isso pode ser feito instruindo-se o usuário inicial da chamada a enviar um novo pedido de INVITE a uma outra localização usando a resposta 302 (*moved temporarily*) ou repassando a mensagem INVITE ao endereço de transporte apropriado.

Apontar para um servidor SIP em vez de apontar para o destino diretamente permite que este se mude, além de permitir algum esquema de *cache*. Se o endereço do ponto de destino da chamada estiver armazenado diretamente no DNS, poderá haver muitos problemas com a *cache* do DNS.

Normalmente todos os registros DNS podem ser armazenados em *cache* pelo resolvedor de DNS. O registro no *cache* expira após TTL (*Time To Live*) segundos. O valor de TTL é armazenado no registro DNS. Portanto, quando o terminal mudar de lugar, o originador da chamada ainda poderia ter um endereço errado no cache do resolvedor DNS e a chamada falharia. A única solução é colocar o valor de TTL em zero e atualizar o registro primário de DNS quando o terminal se mudar.

2.6.3 Estabelecendo uma Sessão SIP

Quando um terminal SIP deseja iniciar uma sessão (de áudio, vídeo ou texto), ele gera um pedido INVITE. O pedido é enviado para um servidor ou direto para o destinatário do pedido para o estabelecimento da sessão. Esse pedido é passado pelo *proxy*, eventualmente é entregue para um ou mais UAS, que podem potencialmente aceitar o pedido de INVITE. O UAS deverá enviar uma mensagem para o originador do pedido, informando que o pedido de INVITE foi aceito, que é representado pelo envio de uma resposta 2xx. Se o pedido de INVITE não for aceito, uma resposta 3xx, 4xx, 5xx ou 6xx é enviada, dependendo do motivo pelo qual foi rejeitada. Antes de se enviar uma resposta final, o UAS pode enviar uma resposta provisória (1xx) para o originador do pedido, informando que a chamada está em progresso.

Depois de receber possivelmente uma ou mais respostas provisórias, o UAC irá receber uma ou mais respostas finais 2xx ou resposta final diferente de 2xx. Um vez recebida a resposta final, o UAC deve enviar um ACK para toda resposta recebida. O procedimento para enviar um ACK depende do tipo de resposta. Para uma resposta final de 300 à 699, o ACK processado é feito na camada de transação e seguido de um conjunto de regras [Rosemberg and Schulzrinne, 2002]. Para as repostas 2xx, o ACK é gerado pelo núcleo do UAC.

Uma resposta 2xx para um INVITE estabelece uma sessão, que cria um diálogo entre o UA que criou o pedido de INVITE e o UA que gerou a resposta. Portanto, quando múltiplas respostas 2xx são recebidas de diferentes UAs remotos, para cada resposta 2xx estabelece-se um diálogo diferente. Todos esses diálogos são partes de uma mesma chamada. Esta seção mostra em detalhes o estabelecimento de uma sessão usando o pedido INVITE.

A primeira etapa consiste em abrir uma conexão de sinalização entre os pontos de origem e destino da sessão. O originador da chamada SIP pode usar sinalização UDP ou TCP - a sintaxe das mensagens é independente do protocolo de transporte usado. Quando se usa o TCP, a mesma conexão pode ser usada para todos os pedidos e respostas SIP (não para os dados de mídia) ou uma nova conexão TCP pode ser usada para cada transação. Se o UDP for usado, então o endereço e a porta a serem usados para as respostas à pedidos SIP estarão contidos no campo de cabeçalho **Via** do pedido SIP. Se nenhuma porta for especificada no endereço, a conexão é feita com a porta 5060 tanto para o TCP quanto para o UDP.

A Figura 2.4 mostra um exemplo típico de uma troca de mensagem SIP entre dois usuários, usr1 e usr2. Neste cenário, usr1 usa uma aplicação em seu PC para chamar Usr2, que usa uma aplicação em seu sistema embarcado, por exemplo um PDA (*Personal Digital Assistant*). Também são mostrados dois servidores *proxy* que atuam entre os dois pontos finais, para facilitar o estabelecimento da sessão.

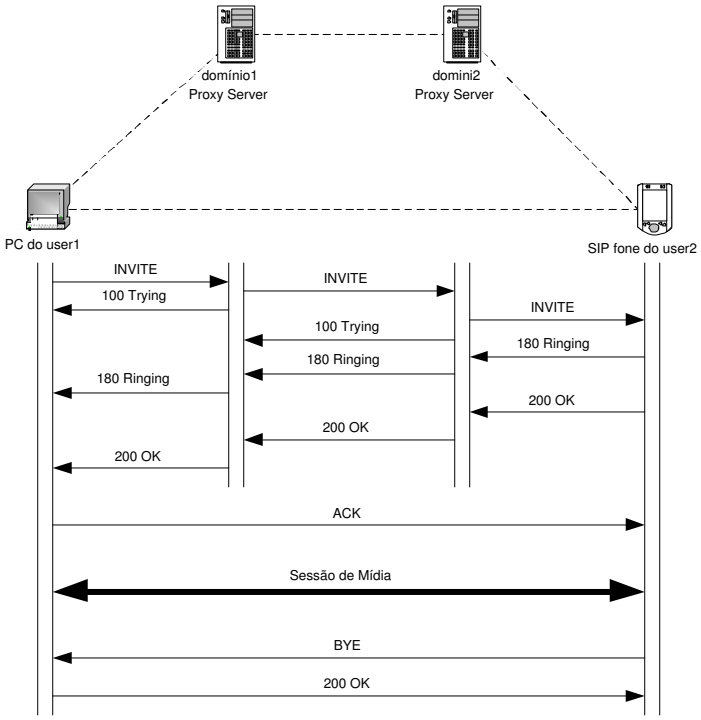

Figura 2.4: Cenário completo de uma sessão SIP.

O usuário1 (usr1) inicia uma chamada para usr2 usando sua identidade SIP, um tipo de *Uniform Resource Identifier*(URI) chamado de SIP-URI, que tem um formato similar ao endereço de email, no qual tipicamente contém um username e um *hostname*. Neste caso o endereço SIP é na forma de sip:usr2@dominio2 , onde o domínio2 é o domínio do serviço SIP, mas poderia ser um endereço IP do destino final, no qual a mensagem não passaria pelo servidor e iria direto para o usuário final. O usuário (usr1) também tem um SIP-URI na forma de sip:usr1@dominio1 . O usr2 pode teclar o URI de usr1, clicando no hyperlink ou entrando em um livro de endereço. O SIP pode prover serviço com segurança, em que partes da mensagem é criptografada, essa técnica é chamada SIPS-URI.

O SIP é baseado no modelo de transação pedido/resposta como o HTTP, onde cada transação consiste em um pedido que envolve um método ou função. Neste exemplo a transação inicia com o usr1 enviando um pedido INVITE endereçado para o URI de usr2. O pedido INVITE contém um certo número de campos de cabeçalhos. O método INVITE pode ser visto na Figura 2.5.

Como o Usr1 não conhece a localização do Usr2 ou o servidor de domínio do usr2, então o usr1

```
INVITE sip:usr2@dominio2 SIP/2.0
Via: SIP/2.0/UDP dominio1:5060;branch=z9hG4knib74u;
received=143.106.50.180
Max-Forwards: 70
To: usr2 <sip:usr2@dominio2>
From: usr1 <sip:usr1@dominio1>;tag=487jni9
Call-ID: j94nbgh46yu48nbfi
CSeq: 7982 INVITE
Contact: <sip:usr1@domino1>
Content-Type: application/sdp
Content-Length: 142

v=0
o=usr1 165468471 216549871332 IN IP4 143.106.50.220
e=usr1@domino1
c=IN IP4 143.106.50.220/7000
t=0 0
m=audio 7000 udp 0
```

Figura 2.5: Formato da mensagem INVITE.

envia o INVITE para o servidor SIP que está no seu domínio (dominio1). O endereço de domínio (dominio1) do servidor SIP pode ser configurado no Usr1 ou pode ser buscado via DHCP.

O servidor de domínio (dominio1) é um tipo de servidor SIP conhecido como servidor *proxy*. O servidor *proxy* recebe um pedido SIP e o direciona para o destinatário ou o próximo servidor. Neste exemplo, o servidor *proxy* recebe o pedido INVITE e envia uma reposta 100 (*Trying*) de volta para o usr1. O servidor *proxy* de domínio (dominio1) localiza o servidor de domínio (dominio2), possivelmente pela ação particular de um DNS (*Domain Name Service*), que encontra o servidor de domínio (dominio2). Como resultado, obtém-se um endereço IP do servidor *proxy* de domínio (dominio2), mas antes de enviar para o servidor do usr2, o servidor de domínio (dominio1) adiciona um valor de campo de cabeçalho Via, que contém o seu próprio endereço, isso irá definir a trajetória seguida pela mensagem SIP. O servidor *proxy* de domínio (dominio2) recebe o pedido INVITE e responde com uma resposta 100 (*Trying*) para o servidor *proxy* de domínio (dominio1), indicando que o pedido INVITE foi recebido e que está processando o pedido. O servidor *proxy* de domínio (dominio2) adiciona um valor no campo cabeçalho Via com seu próprio endereço e encaminha a mensagem para o usr2.

O usr2 recebe o INVITE e alerta o recebimento de uma chamada do usr1, acionando um sinal sonoro para Usuário. O Usr2 indica este sinal de chamada, com uma resposta 180 *Ringing*, que é roteada de volta através dos dois *proxy*s, em um caminho reverso, usando os parâmetros do campo de cabeçalho Via. Cada *proxy* usa o campo cabeçalho Via para determinar onde deve enviar a resposta e remove seu campo de cabeçalho Via do topo.

Quando o usr1 recebe a resposta 180 *Ringing*, ele passa a informação para usuário, que usa um retorno de áudio, como analogia a um tom de chamada de um telefone, para indicar que o Usr2 foi informado da chamada.

Neste exemplo o usr2 decide aceitar a chamada, quando indica o seu estado como livre para receber chamadas, neste momento o usr2 envia uma resposta de 200 (OK) para indicar que a chamada foi aceita. O 200 (OK) contém um corpo de mensagem com a descrição SDP, para o tipo de sessão que o usr2 irá estabelecer com usr1, nesse momento inicia a negociação do tipo de mídia, a qual irá ser usada na sessão. Essa negociação básica da capacidade de mídia é baseado no modelo simples de oferecimento/resposta da negociação SDP. Se usr2 não quiser responder a chamada de usr1, ele altera seu estado gerando uma resposta, que pode ser ocupado. A mensagem de resposta pode ser como apresentado na Figura 2.6.

```
SIP/2.0 200 Ok
Via: SIP/2.0/UDP usr2.dominio2:5060;branch=z9hG4kvoin4604;
received=143.106.50.218
Via: SIP/2.0/UDP dominio2:5060;branch=z9hG4k70bneoijb94;
received=143.106.50.169
Via: SIP/2.0/UDP dominio1:5060;branch=z9hG4knib74u;
received=143.106.50.180
Max-Forwards: 70
To: usr2 <sip:usr2@domnio2>;tag=bniehb496cv
From: usr1 <sip:usr1@domnio1>;tag=487jni9
Call-ID: j94nbgh46yu48nbfi
CSeq: 7982 INVITE
Contact: <sip:usr2@domnio2>
Content-Type: application/sdp
Content-Length: 142

v=0
o=usr2 165468471 216549871332 IN IP4 143.106.50.218
s=audio call
e=usr2@dominio2
c=IN IP4 143.106.56.218/7000
t=0 0
m=audio 7000 udp 0
```

Figura 2.6: Formato da resposta para o pedido INVITE.

Neste caso a resposta 200 (OK) é roteada através de dois *proxies* até chegar no usuário final, que então encerra o sinal de "chamado" e indica que o pedido foi respondido e aceito. Finalmente o usr1 envia uma mensagem ACK (*acknowledgement*), o ACK é enviado diretamente do usr1 para o usr2, saltando os dois *proxies*. Isto ocorre porque o usr1 conhece o endereço do usr2 através do campo cabeçalho Contact. Durante a sessão, tanto o usr1 como usr2 podem decidir mudar as características do meio. Isto é possível reenviando-se um INVITE contendo uma nova descrição de sessão, onde a

outra parte da comunicação envia um 200 (OK) aceitando a mudança. O encerramento de uma sessão pode ser feito por qualquer um dos participantes, que gera uma mensagem BYE. Neste exemplo, o usr2 decide encerrar a sessão. Este BYE é enviado diretamente para o usr1, saltando novamente os *proxies*. O usr1 confirma o recebimento do BYE com uma resposta 200 (OK), que termina a sessão e a transação BYE.

2.6.4 Encerrando uma Sessão SIP

Nesta sessão descreve-se o procedimento para encerrar uma sessão estabelecida pelo protocolo SIP. Quando uma sessão é iniciada com um INVITE, cada resposta 1xx ou 2xx de um UAS distinto dá início a um diálogo, se a troca de mensagens for completada, então será criada uma sessão. Como resultado, cada sessão é "associada" com um único diálogo, resultando em sua criação. Se um INVITE inicial gera uma resposta final diferente de 2xx, então toda sessão é terminada e todo o diálogo que for criado pelas respostas para o pedido é encerrado. O pedido BYE é usado para encerrar uma sessão específica, como mostrado na Figura 2.7. Quando um BYE é recebido em um diálogo, a sessão que está associada àquele diálogo é encerrada. Um UA não pode enviar um BYE fora de um diálogo. O UA originador pode enviar um BYE tanto para um diálogo confirmado, quanto para um inicial. Portanto, o UA de destino não pode enviar um BYE em um diálogo confirmado até que ele tenha recebido um ACK para sua resposta 2xx ou o tempo de espera tenha encerrado. Se nenhuma extensão SIP tenha definido outro estado na camada de aplicação associado com o diálogo, o BYE também deve ser terminado.

```
BYE sip:usr2@dominio2 SIP/2.0
Via: SIP/2.0/UDP dominio1:5060;branch=z9hG4knib74u;
received=143.106.50.220
Max-Forwards: 70
To: usr2 <sip:usr2@domnio2>;tag=bniehb496cv
From: usr1 <sip:usr1@domnio1>;tag=487jni9
Call-ID: j94nbgh46yu48nbfi
CSeq: 7982 INVITE
Contact: <sip:usr2@domnio2>
Content-Type: application/sdp
Content-Length: 142
```

Figura 2.7: Formato do encerramento da sessão.

Quando o usuário recebe um pedido de encerramento de sessão, ele retorna uma reposta 220 OK, que confirma o encerramento da sessão, como mostrado na Figura 2.8.

```
SIP/2.0 200 Ok
Via: SIP/2.0/UDP dominio1:5060;branch=z9hG4knib74u;
Max-Forwards: 70
To: usr2 <sip:usr2@domnio2>;tag=bniehb496cv
From: usr1 <sip:usr1@domnio1>;tag=487jni9
Call-ID: j94nbgh46yu48nbfi
CSeq: 7982 INVITE
Content-Length: 0
```

Figura 2.8: Formato da confirmação do fim da sessão.

2.7 Conclusão do SIP em relação ao H.323

A primeira coisa que impressiona quando se lê o documento SIP é sua simplicidade. O SIP faz em uma transação o que o H.323 versão 1 fez em quatro ou cinco trocas de mensagens, cada uma delas especificada em documentos ITU-T diferentes. Essa diferença resultou em um tempo de configuração menor para os pontos finais SIP. O uso de URLs como identificadores é poderoso, pois a diferença entre o email H.323 (helder@dominio.com) e uma URL SIP (sip:helder@dominio.com) é que um servidor SIP pode redirecionar uma chamada para servidores não SIP de maneira bastante flexível.

A codificação de texto é uma característica atrativa no SIP, pois é simples e pode ser depurada facilmente usando-se simples *sniffers* de rede e faz com que problemas de interoperabilidade sejam detectados 'visualmente'. O único problema pode ser no desempenho e tamanho, pois a codificação de texto ocupa muito espaço, diferente do protocolo H.323 que usa uma codificação feita em binários, tornando-se mais fácil de codificar e decodificar.

Por esses motivos que o protocolo SIP tem sido escolho por um número maior de pesquisadores. Nesse trabalho foi escolhida a utilização do protocolo SIP, pela facilidade de implementação do código para sistema embarcado, sua flexibilidade junto aos servidores de email e os protocolos existentes do HTTP, dentre as inúmeras vantagens citadas anteriormente.

Capítulo 3

Protocolo de sinalização H.323

O Protocolo H.323 é formado por um conjunto de recomendações definidas pelo *International Telecommunications Union* (ITU), que normatiza os sistemas de comunicações de pacotes [telecommunication union, 2001b]. As outras recomendações que compõem o protocolo são: H.245 *Control Protocol for Multimedia Communication* [telecommunication union, 2000], H.225.0 *Call Signalling Protocols and Media Stream Packetization for Packet-based Multimedia Communication Systems* [telecommunication union, 2001a] e Q.931 *User-network Interface Layer 3 Specification for Basic Call Control* [telecommunication union, 1993].

O H.323 não é um protocolo individual, mas sim um conjunto completo de protocolos integrados verticalmente, que definem terminais, equipamentos e serviços para comunicações multimídia sobre redes de dados como a Internet. A recomendação H.323 está limitada à definição da sinalização, controle de fluxo, formato de pacotes e normas de compressão dos meios. As especificações sobre captação dos meios, como os formatos de captação de áudio e vídeo, ou aplicações de dados estão fora do âmbito da recomendação H.323. Neste capítulo é apresentada uma visão geral do protocolo H.323, apresentando-se as suas características principais e sua sinalização, tendo como objetivo principal a comparação com o protocolo SIP.

3.1 Recomendação H.323

A primeira versão da recomendação H.323 foi publicada pelo ITU-T em 1996. Tinha como objetivo inicial os serviços de comunicação de áudio e vídeo sobre LANs sem garantia de qualidade de serviço (QoS). Foi no entanto criticada pelo seu baixo desempenho e problemas de compatibilidade entre diferentes fabricantes, levando estes a adicionar as suas próprias extensões de forma proprietária. A segunda versão, publicada em Janeiro de 1998, dispunha de mecanismos de conexão rápida, resolvendo o problema na demora no estabelecimento da chamada, onde novos recursos eliminaram

a necessidade de extensões proprietárias, assim como novos protocolos [Packetizer, 2002a]. A terceira versão, que foi aprovada em Setembro de 1999, usou de melhoramentos modestos em relação à recomendação H.323v2, introduzindo poucas alterações no documento base. No entanto a recomendação H.323 evoluiu substancialmente através da adição de novos anexos ao H.323 e ao H.225.0, melhorando a arquitetura do H.323 [Packetizer, 2002b]. A última versão (H.323v4), aprovada em Novembro de 2000, inclui melhorias em várias áreas importantes, tais como na escalabilidade, flexibilidade e segurança. Novas características foram adicionadas nos Gateways e *Multipoint Control Unit* (MCU), de modo a facilitar a inclusão de novas soluções adequadas às necessidades crescentes do mercado [Packetizer, 2002c].

A Recomendação H.323 define um conjunto de identidades num sistema H.323, estando incluídas o terminal H.323, *Gatekeeper*, *Multipoint Controller* (MC), *Multipoint Processor* (MP), *Multipoint Control Unit*(MCU) e *Gateway*.

3.2 Componentes do Protocolo H.323

O protocolo H.323 especifica quatro tipos de componentes, sendo que prover serviços de comunicação ponto-a-ponto, ou ponto-a-multiponto em uma rede de comutação de pacotes, que é função do terminal H.323; além disso temos o *Gateway*, o *Gatekeeper* e unidade de controle muiltiponto.

3.2.1 Terminal H.323

Usado para comunicação multimídia em tempo real, um terminal H.323 pode ser tanto uma máquina PC ou um dispositivo embarcado, rodando um protocolo H.323 e uma aplicação multimídia. Este suporta tanto comunicação de áudio e pode opcionalmente suportar tanto comunicação de vídeo quanto de dados. Um terminal H.323 estabelece um canal de comunicação bidirecional com outro terminal H.323, com um *gatekeeper*, com um *gateway* ou com uma unidade de controle multiponto. A comunicação entre dois terminais consiste de mensagens de controle, fluxos multimídia e dados. A Figura 3.1 mostra os elementos internos de um terminal H.323.

- **Codec de vídeo** - Suportando obrigatoriamente a recomendação H.261 [telecommunication union, 2000] e opcionalmente a H.263, referente a codificação de sinal de vídeo;
- **Codec de áudio** - Suportando obrigatoriamente a recomendação G.711 [telecommunication union, 1998] e opcionalmente a recomendação G.723 e G.729 [et al, 1997];
- **Atraso de recepção** - Responsável pelo controle do *jitter* nas recepções de audio e/ou vídeo;

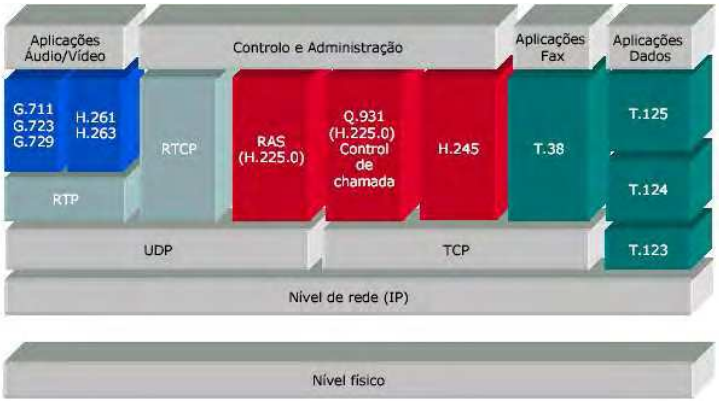

Figura 3.1: Terminal H.323.

- **Canal de dados** - Suporta um ou mais canais de dados, uni ou bidirecionais. O T.12x é usado para a interoperabilidade entre os terminais H.323 ou outro tipo suportado pelos ambientes externos ligados por *gateway*;
- **Controle H.245** - Responsável pelo manuseio de mensagens de controle fim-a-fim, que governam a operação da entidade H.323, no tocante ao modo de operação do receptor para o transmissor e vice-versa, abertura e fechamento de canais lógicos para troca de informações de transmissor para um ou mais receptores, requisição de preferência de modo de operação (*unicast* ou *multcast*), mensagens de controle de fluxo, comandos gerais e indicações;
- **Controle RAS** - Controle das mensagens de sinalização para registro, admissão, troca de banda, pedido de informação sobre a situação atual e desvinculação com o *Getakeeper*;
- **Controle de chamada** - Usado para o estabelecimento de conexão entre dois terminais H.323;
- **H.225** - estabelece o formato das mensagens trocadas pelos canais lógicos de áudio, vídeo, dados ou controle.

3.2.2 Gateway

Converte, apropriadamente, diferentes formatos de mensagens (por exemplo: H.225.0 / H.221) e procedimentos de comunicação (por exemplo: H.245 / H.242). A conversão entre diferentes formatos de áudio, vídeo ou dados também pode ser feito pelo *gateway*. Em geral, o objetivo do *gateway* é refletir as características de um terminal da rede local para um outro terminal da rede comutada por circuitos, e vice-versa.

3.2.3 Gatekeeper

É um elemento opcional no sistema H.323, provendo serviços de controle de chamadas para os terminais. Este possui como funções obrigatórias:

- **tradução de endereços** - permite o uso de apelido em lugar dos endereços de transporte;
- **Controle de admissão** - Autoriza o acesso de recursos usando por base critérios como, permissão de acesso em período predeterminado, banda disponível;
- **Controle de admissão** - Rege os pedidos de troca de banda em uso;
- **gerência de zona** - Provê as funções acima para terminais, MCUs - *Multipoint Control Units* e *gateway* nele registrados.

Opcionalmente, o *gatekeeper* pode apresentar outras funções, tais como:

- **Controle de sinalização** - Pode controlar todo processo de sinalização de chamadas entre terminais, ou deixar que os próprios terminais tratem da sinalização;
- **Autorização de chamadas** - Através da sinalização prevista na H.225, o Gatekeeper pode rejeitar chamadas de um terminal sem autorização adequada;
- **gerência de banda** - controla o número de acessos simultâneos de terminais H.323 a LAN, podendo rejeitar uma chamada por falta de banda disponível;
- **gerência de chamadas** - Controla terminais que possuem chamadas estabelecidas, para o caso de uma nova chamada a um terminal ocupado ser informado aos módulos de direito.

Um conceito importante, introduzido pela segunda versão do H.323 é o de zona, que será apresentado na próxima sessão.

3.2.4 Unidade de Controle de Multiponto

O MCU é um Elemento para suporte a conferência com três ou mais pontos, podendo funcionar em três modos distintos:

- **Modo Centralizado:** - A comunicação entre a MCU e os terminais ou *gateway* é *unicast*. Dados, vídeo e controle passam obrigatoriamente pela MCU;
- **Mode Descentralizado** - Os terminais trocam informações de controle, e opcionalmente de dados, de forma centralizada com a MCU, mas trocam áudio e vídeo entre si por *multcast*;
- **Modo Híbrido** - A comunicação de dados e controle sempre se dá de forma centralizada com a MCU. Contudo, podemos ter áudio também centralizado e *multicast* de vídeo ou vice-versa.

3.3 Zona H.323

Uma Zona H.323 é uma coleção de todos os terminais, *gateways* e MCUs gerenciados todos por um único *gatekeeper*, como na Figura 3.2. Uma zona inclui no mínimo um terminal H.323, podendo incluir emphgateways ou MCUs. Uma zona tem somente um Gatekeeper. Uma zona pode ser independente da topologia da rede e pode participar de múltiplos segmentos de rede que são conectados através de roteadores e outros componentes.

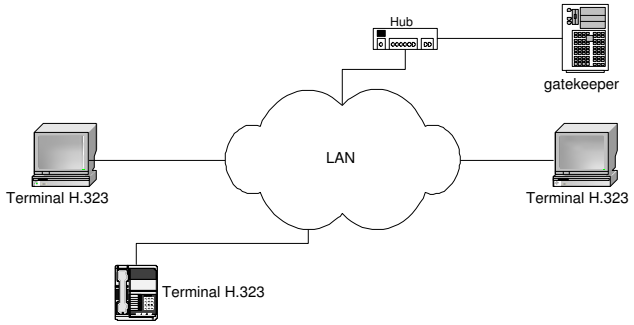

Figura 3.2: Cenário da sinalização H.323.

3.4 Estabelecendo uma Conexão

Na sinalização H.323 consideremos que dois usuários conectados a dois terminais IP distintos gostariam de estabelecer uma chamada de voz. Do ponto de vista de uma rede de comutação de circuitos, os usuários teriam a mesma experiência que teriam se estivessem usando dois terminais analógicos conectados às suas respectivas centrais, desejando ter uma conversação.

3.4.1 Primeira fase: iniciando a chamada

O H.323 usa um subconjunto do protocolo Q.931 [telecommunication union, 1993], utilizado em ISDN (*Integrated Services Digital Network*), de mensagens de sinalização para o controle de chamada na interface usuário-rede. As seguintes mensagens fazem parte do núcleo do H.323 e devem ser suportadas por todos os terminais:

- *Setup*;
- *Alerting*;
- *Connect*;

- *Release Complete*;
- *Status Facility*.

Outras mensagens, como *Call Proceeding*,*Status*, *status Enquiry*, são opcionais. Com relação aos serviços suplementares, somente a mensagem *Facility* é suportada; todas as outras, como *Hold*, *Retrieve*, *Suspend* etc. são proibidas (a mesma funcionalidade é fornecida pelo H.450).

Como mostra a Figura 3.3, o terminal A deseja fazer uma ligação para o terminal B, conhecendo o endereço IP de B. O terminal A envia ao terminal B uma mensagem de *Setup* na porta conhecida do canal de sinalização de chamadas usando uma conexão TCP. Essa mensagem é definida no H.225.0 e contém os seguintes elementos de informação, que tomamos emprestado do Q.931:

- Um campo discriminador de protocolo, com o valor 08H (define isso como uma mensagem de controle de chamada usuário-rede);
- Um valor de referência de chamada localmente único (CVR), de 2 octetos, escolhido pelo lado originado, que será copiado em todas as mensagens posteriores relativas a essa chamada;
- Uma capacidade de transporte, um campo complexo que pode indicar, entre outras possibilidades, se a chamada terá apenas áudio ou áudio e vídeo. *gateways* ISDN podem colocar nesse campo alguns elementos copiados da mensagem *Setup*;
- Um número de chamado e subordens, que devem ser usados quando o endereço for um número E.164. Quando fixado em 1001 (plano de numeração privado), significa que o endereço chamado será encontrado no elemento de informação usuário-usuário da mensagem *Setup*. Se o terminal A conhece o terminal B por meio de seu endereço IP, o plano de numeração será colocado em 1001;
- Um número chamador e sub-endereço, que podem estar presentes se o terminal chamador tiver um número E.164.
- Uma PDU H.323 de usuário, que encapsula a maior parte da informação estendida necessária ao H.323. Nesse caso, trata-se de um elemento de informação *Setup* que contém:
  - um identificador de protocolo (que indica a versão do H.225 em uso);
  - um endereço H.245 opcional, caso o transmissor concorde em receber mensagens H.245 antes de se conectar;
  - um campo de endereço da fonte relacionando os *aliases* do transmissor, como o endereço helder@unicamp.br (no caso de o transmissor ter apenas um número E.164, esse campo é o *calling party field* do Q.931);
  - um campo de informação da fonte pode ser usado pelo terminal chamado para determinar a natureza do equipamento de chamada (MCU, *gateway*, etc.);

- um conjunto de endereços de destino. Vários tipos são definidos no H.323v2 tais como caracteres com número de telefones normal, outro como url-ID, ou endereço de e-mail;
- um identificador de chamada (CALLID), que é determinado por A e deve ser único em nível global, e não somente em nível local.

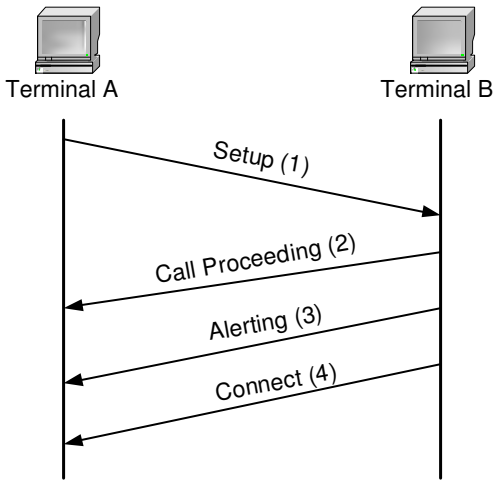

Figura 3.3: Inicializando uma chamada H.323.

As mensagens de Release Complete, Alerting, Connect ou Call Proceeding devem ser enviadas pelo terminal B imediatamente após o recebimento de uma mensagem Setup. Uma delas deve ser recebida pelo terminal A antes que o temporizador de *setup* expire (em geral, após quatro minutos). O terminal B envia uma mensagem Call Proceeding, indicando que recebeu a mensagem e que está processando. Após Alerting ter sido enviada, para informar ao terminal A o estabelecimento da conexão, o usuário tem até três minutos para aceitar ou recusar a chamada.

Assim que o terminal B aceita a chamada, seu terminal envia uma mensagem Connect com:

- Discriminador de protocolo Q.931, a mesma referência de chamada e o tipo de mensagem 07h;
- Na PDU H.323 existe agora um elemento de informação Connect usuário-usuário, com:

  - identificador de protocolo;
  - endereço IP que B quer que A use para abrir a conexão TCP H.245;
  - a informação de destino, que permite que A saiba se está conectado a um *gateway* ou não;
  - identificador de chamada copiado da mensagem Setup.

3.4.2 Segunda fase: estabelecendo o canal de controle

Negociação de Capacidade

O controle da chamada e as mensagens de troca de capacidades são enviadas na segunda conexão TCP, que o terminal chamador estabelece com uma porta dinâmica no terminal chamado, como mostrado na Figura 3.4. As mensagens são definidas no H.245. O terminal chamador abre esse canal de controle H.245 imediatamente após receber uma mensagem *Alerting*, *Call Proceeding* ou *Connect*, não importando qual delas especifica em primeiro lugar o endereço de transporte H.245 a ser usado. É usada uma conexão TCP que deve ser mantida ao longo de toda a chamada. De maneira alternativa, o terminal chamado poderia ter estabelecido esse canal se o terminal chamador tivesse indicado um endereço de transporte H.245 na mensagem *Setup*. O canal de controle H.245 é o único para cada chamada entre dois terminais, mesmo se vários fluxos de mídia estiverem estabelecidos para envio de áudio, vídeo ou dados. Esse canal também é conhecido como canal lógico 0.

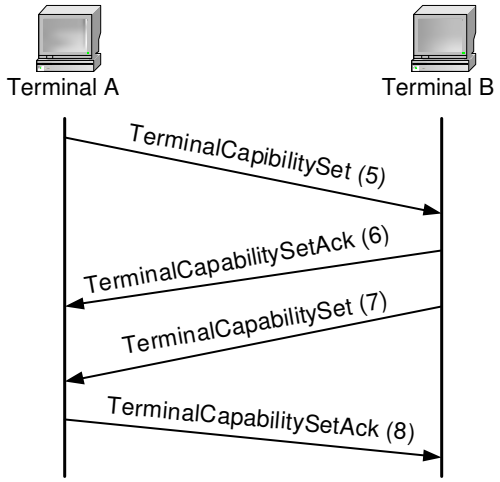

Figura 3.4: Estabelecendo o canal de controle.

A primeira mensagem trocada através do canal de controle é *TerminalCapabilitySet*, que transporta os seguintes elementos de informação:

- Um número de sequência;
- Uma tabela, que é uma lista ordenada de configurações de codecs que o terminal suporta. Cada configuração de codec é uma estrutura na qual o terminal pode expressar os codecs que podem ser usados simultaneamente e alternativamente;
- Descritores de capacidade.

Os terminais enviam essa mensagem entre si e acusam recebimento com uma mensagem *TerminalCapabilitySetAck*.

3.4.3 Terceira fase: início da chamada

Nesta fase o terminal A e o terminal B precisam abrir canais de mídia para voz e possivelmente para vídeo e dados, como mostrado na Figura 3.5. Os dados para esses canais de mídia serão transportados em vários canais lógicos, que são unidirecionais exceto no caso dos canais de dados T.120. Para abrir um canal de voz até B, A envia uma mensagem H.245 *OpenLogicalChannel* que contém o número que será dado ao canal lógico, além de outros parâmetros como o tipo dos dados que serão transportados (exemplo: áudio G.711). No caso de som ou vídeo (que será transportado sobre RTP), a mensagem *OpenLogicalChannel* menciona também o endereço UDP e a porta para a qual B deve enviar os RR (*Receiver Reports*) RTCP, o tipo do *payload* RTP e a capacidade onde o transmissor especifica quantos quadros de codec são colocados em cada pacote RTP.

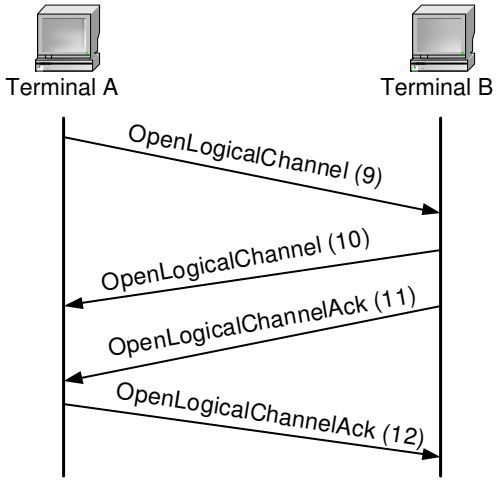

Figura 3.5: Início da chamada.

O terminal B envia uma mensagem *OpenLogicalChannelAck* referente a esse canal lógico tão logo esteja pronto para receber dados de A. Essa mensagem contém o número da porta UDP para a qual A deve enviar os dados RTP e a porta UDP para a qual A deve enviar os dados RTCP. Enquanto isso, o terminal B também pode ter aberto um canal lógico com A seguindo o mesmo procedimento.

3.4.4 Quarta fase: serviços da chamada

Nesta fase os terminais A e B permitem a conversação e a visualização dos usuários caso eles também tenham aberto canal lógico de vídeo, como mostrado na Figura 3.6. Os dados da mídia são enviados

em pacotes RTP. Os pacotes RTCP enviados por A são usados para permitir que B sincronize múltiplos fluxos RTP e também podem ser usados por B para estimar a taxa esperada de dados RTP e para estimar a distância ao transmissor. Os pacotes RTCP enviados por B permitem que A meça a qualidade de serviço da rede entre A e B: as mensagens RTCP contêm a fração de pacotes que forem perdidos desde o último relatório RTP, a perda cumulativa de pacotes, o *jitter* entre chegadas e o mais alto número de seqüência recebida. Os terminais H.323 devem responder ao aumento da perda de pacotes reduzindo a taxa de envio dos mesmos.

O H.323 manda usar apenas um par de portas RTP/RTCP para cada sessão. Pode haver três sessões principais entre os terminais H.323: a sessão de áudio (sessão com ID 1), a sessão de vídeo (sessão com ID 2) e a sessão de dados (sessão com ID 3), mas nada no padrão impede que um terminal abra mais sessões. Para cada sessão deve haver apenas uma porta RTCP usada, isto é, se houver simultaneamente um fluxo RTP de A para B e de B para A, o transmissor RTCP e os relatórios RTP para ambos os fluxos vão usar a mesma porta UDP.

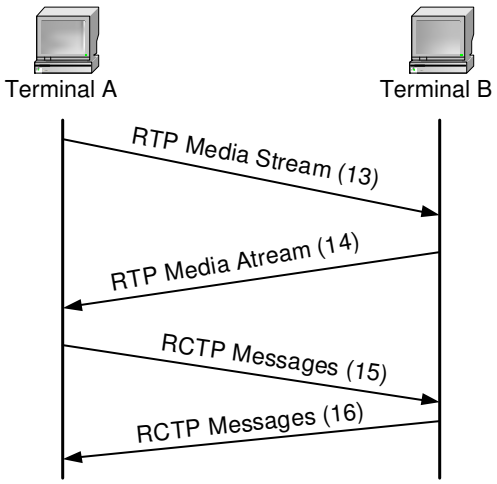

Figura 3.6: Diálogo durante a chamada.

3.4.5 Enfim: encerrando uma chamada

Finalizar uma chamada H.323 não é tão simples. Se for o terminal A quem vai finalizar a chamada, o terminal A deve enviar uma mensagem H.245 *CloseLogicalChannel* para cada canal lógico que A abriu. O terminal B acusa o recebimento dessas mensagens com uma mensagem *CloseLogicalChannelAck*. Depois de todos os canais lógicos terem sido fechados, A envia uma mensagem H.245 *endSessionCommand*, espera até que tenha recebido a mesma mensagem de B e fecha o canal de controle H.245. Finalmente, A e B devem enviar uma mensagem H.225 *ReleaseComplete* através do

canal de sinalização de chamada se ainda estiver aberto; esse canal é então fechado, assim como a chamada, como mostrado na Figura 3.7.

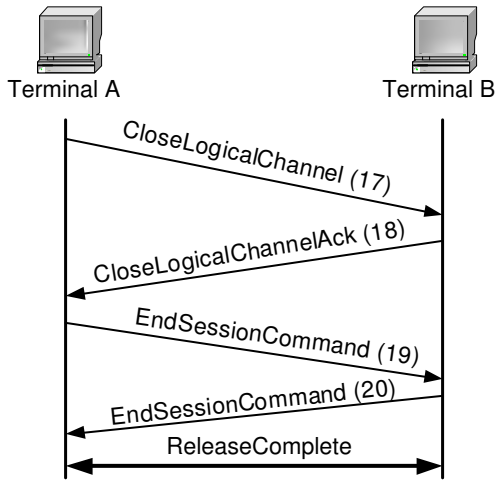

Figura 3.7: Encerramento de chamada.

Capítulo 4

Qualidade de Serviço em Voz sobre IP

No contexto da Internet, qualidade de serviço (QoS: *Quality of Service*) refere-se a capacidade de rede em prover melhor serviço para um tráfego para várias mídias. Intuitivamente, qualidade de serviço expressa, em última análise, o grau de satisfação do usuário com os serviços buscados na rede.

Inicialmente o protocolo IP, como outras tecnologias de redes de pacotes, foi construído para transportar dados, mas não voz ou vídeo. Até então, o requisito 'qualidade de serviço' era a garantia de integridade dos dados. Ou seja, os dados não deveriam ser danificados ou perdidos. Hoje, com o desenvolvimento da tecnologia de redes, tornou-se possível o transporte de sinais multimídia (áudio, voz, entre outros) em tempo real sobre uma rede IP. Para esse novo de tipo de tráfego, no entanto, é fundamental a caracterização e o controle das flutuações de tráfego na rede, que é conhecido como *jitter*. O controle dessas flutuações é especialmente importante em aplicações em tempo real que precisam manter um tamanho de *buffer* de pior caso para garantir a entrega oportuna dos pacotes [Schröder and Sherif, 1997]. A taxa de perda de pacotes também é um parâmetro importante para QoS em rede de pacotes.

4.1 Fatores que Influenciam na Qualidade de Serviço

Vários são os fatores que influenciam na qualidade de transmissão de voz em uma rede IP, entre eles podem-se citar:

- Atraso ou latência;
- *Jitter*
- Perda de Pacotes;
- Banda;
- Eco.

4.1.1 Atraso ou latência

O tempo necessário para que um pacote de áudio gerado na origem chegue até seu destino não deve ultrapassar um patamar adequado, sob a pena de degradar a qualidade da aplicação. O patamar ideal depende fundamentalmente de três aspectos: o tipo de interatividade entre os usuários da aplicação, o nível de exigência dos usuários da aplicação e o quanto se está disposto em gastar para viabilizar uma solução que garanta atrasos pequenos. Dependendo da aplicação em questão, o grau de interação entre os usuários é grande ou não.

A latência refere-se o tempo total medido, desde o momento que o locutor emite um som, até que o ouvinte receba o sinal. Um valor de latência baixo torna mais interativa a conversação. Em uma rede PSTN, a latência para uma chamada fim-a-fim é abaixo de 150ms.

Na implementação de VoIP usado para reduzir custos, estudos sugerem que usuários irão tolerar valores de latência de no máximo 200 ms. A recomendação G.114 da ITU de 1996 define que o limite de tempo para transmissões fim-a-fim é de:

- Abaixo de 150 ms: aceitável para maioria das aplicações dos usuários;
- de 150 até 400 ms: aceitável, porém o administrador deve enviar em consideração o tempo de transmissão e seu impacto na qualidade das aplicações de usuário;
- acima de 400 ms: não aceitável para o propósito de rede em geral.

4.1.2 Jitter

Um grande problema enfrentado na transmissão VoIP é a característica aleatória do atraso da rede IP. Em uma conexão típica, os pacotes passam por vários roteadores até atingir o destino. Os pacotes não são transmitidos pelo mesmo caminho, podendo alguns pacotes levar mais tempo que outros. Cada roteador acrescenta um atraso variado, dependendo do número de pacotes que estão no *buffer*. A flutuação do instante da chegada dos pacotes ao redor do valor médio da latência é chamado de jitter.

Na transmissão de voz sobre IP os datagramas podem tomar caminhos diferentes na rede, resultando em diferentes tempos de propagação, ou se pode ter congestionamentos momentâneos que obriguem a maiores retardos. Para contornar este problema, são usados *buffers* nas entradas dos equipamentos decodificadores, afim de guardar alguns pacotes como “reserva” em uma fila, que é servida de forma constante. Mesmo que alguns pacotes sofram uma demora maior que a normal para chegada (dentro de certos limites), os pacotes no *buffer* são enviados para o decodificador com uma taxa constantes. A quantidade ótima de pacotes armazenados nestes *buffers* depende do tamanho dos pacotes de voz, da taxa de transmissão, do atraso médio da rede e como não poderia deixar de ser, da exigência da qualidade da voz requerida.

4.1.3 Banda

Cada tipo de codificador de voz necessita de uma banda mínima para transmissão pelo canal. A fim de minimizar o requisito de banda, as técnicas mais empregadas, além da compressão inerente do codificador, são de supressão de silêncio na conversação e a de compressão de cabeçalhos dos pacotes IP.

Supressão de Silêncio

Ao longo de uma conversação existem vários períodos de silêncio. Valendo-se disto, as implementações dos codificadores de voz podem reduzir a banda consumida. A técnica de supressão de silêncio suprime a transferência de pausas, respirações e outros períodos de silêncio podendo chegar a 50-60% do tempo de uma chamada, economizando largura de banda de transmissão. Como a falta de pacotes é considerada silêncio absoluto é necessário uma função para adicionar um "ruído de conforto" na transmissão.

Compressão do Cabeçalho IP

As soluções de áudio sobre IP utilizam, além deste protocolo, o UDP (*User Datagram Protocol*) e o RTP (*Real Time Transport Protocol*), como protocolos de transporte. A soma dos cabeçalhos dos três protocolos resulta em 40 bytes (20 para o IP, 8 para o UDP e 12 para o RTP). Tomando como exemplo uma implementação de G.729 com um pacote formado por dois quadros do codec ou G.723.1 a 5,3Kbps, com um pacote formado por um quadro do codec, em ambos os casos teremos 20 bytes de informação a ser transmitida. Em vista destes dados, fica evidente o despropósito na distribuição de bytes úteis e de controle. Assim, uma técnica empregada é descrita na RFC 2508 [Casner and Jacobson, 1999] onde a maioria dos pacotes terão seus cabeçalhos comprimidos para 2 ou 4 bytes, dependendo do uso de *checksum* pelo UDP ou não. A idéia básica é que após a transmissão do primeiro pacote sem compressão de cabeçalho, vários campos dos pacotes seguintes podem ser suprimidos ou variam de forma conhecida, sendo "remontados" no destino.

4.1.4 Perda de Pacotes

A rede IP não garante a entrega dos pacotes e também não foi projetada para transmitir voz, podendo os pacotes de voz serem descartados como se fossem pacotes de dados, dentro de um tráfego congestionado. A perda de pacotes é portanto inevitável, podendo influenciar significativamente a qualidade de serviço de voz sobre IP.

As redes IP não podem assegurar que todos os pacotes serão entregues, muito menos na ordem

correta de envio. Alguns pacotes podem ser perdidos durantes as transmissões quando a rede estiver congestionada. A interpolação entre pacotes é uma solução encontrada para solucionar o problema de perda de pacotes, uma vez que a qualidade de voz é sensível ao tempo de entrega dos pacotes. Perdas de pacote maior que 10% geralmente não são toleráveis.

4.1.5 Eco

O eco é uma percepção que o locutor tem de sua própria voz, porém atrasada. O eco pode ser causado de duas maneiras: pode ser um eco elétrico, causado pelos cirtuitos do percurso, ou pode ser um eco acústico, por vibrações acústicas no ambiente. Se o eco do locutor for refletido duas vezes, ele também poderá afetar o lado do ouvinte. Neste caso, o ouvinte escuta a voz do chamador duas vezes, chegam primeiro um sinal alto e depois um sinal atenuado e muito atrasado.

4.2 Técnicas de Qualidade de Serviços

A largura de banda é o limitador da capacidade de transmissão de dados na rede, e uma largura de banda inadequada pode causar deste um atraso do pacote quanto uma perda. Como o tráfego na rede é irregular e a rede não é orientada à conexão, é necessário definir características para reservar recursos com a finalidade de conferir tratamento adequado para pacotes que fluem através da mesma.

Existem várias técnicas de priorização de pacotes que podem ser usadas. Entre elas estão COS, TOS, DiffServ e IntServ.

4.2.1 Classes de Serviços (COS)

A idéia é prover uma divisão dos diferentes fluxos de redes em um número reduzido de “classes de serviços”. Essas classes descrevem valores que podem ser usados para identificar uma determinada prioridade. Para isso é necessário que pacotes carreguem consigo as classes à que pertencem.

Essas classes de serviços podem ser implementadas de duas maneiras, nos quais a cada classe é atribuída uma prioridade, relativa às demais classes:

- Pacotes com prioridades maiores possuem tratamento privilegiado em relação aos de prioridades menores;
- Quando um congestionamento ocorre, os pacotes de prioridades menores são descartados em favor aos pacotes de prioridades maior.

4.2.2 Tipos de Classes (TOS)

O protocolo IP versão 4 (IPV4), reserva oito bits de seu cabeçalho para um campo chamado Tipos de Serviços (TOS), como mostrado na Figura 4.1. De acordo com a RFC 791, este campo possui quatro sub-campos que são:

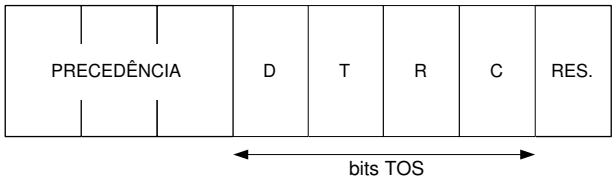

Figura 4.1: Sub-campos do TOS (Types Of Service).

1. Os três primeiros bits representam o tipo de datagrama, sendo o indicativo de prioridade;
2. O quarto bit (bit D: *delay*), se ativo, solicita um roteamento com baixo atraso;
3. O quinto bit (bit T: *throughput*), se ativo, solicita um roteamento com alto fluxo;
4. O sexto bit (bit R: *reliable*), se ativo, solicita um roteamento com alta confiabilidade;
5. Os dois últimos bits foram reservados para expansões futuras.

Na RFC 1349 redefine-se os campos acrescentando mais um bit (bit C: cost) aos bits D, T e R. Estes bits são denominados de bits TOS e possuem o significado como apresentado na tabela 4.1.

TOS	Descrição
1000	minimize o atraso de propagação (bit D)
0100	maximize fluxo (bit T)
0010	maximize confiabilidade (bit R)
0001	minimize custo monetário (bit C)
0000	serviço normal (melhor esforço)

Tabela 4.1: Descrição dos bits TOS

Os bits TOS são utilizados para fornecer informações para os roteadores e também podem ser usados por *switches* de nível 3.

4.2.3 Serviço Diferenciados (DiffServ)

A arquitetura de Serviços Diferenciados (DiffServ), que foi proposto pelo IETF, possui uma alta escalabilidade a custa de uma forte agregação de fluxos em umas poucas classes de serviços. O

DiffServ emprega classificação como mecanismo de QoS e utiliza o campo TOS do datagrama IPv4.

Segundo a RFC 2475, o DiffServ é um refinamento do esquema de prioridades relativas, cuja principal desvantagem é assegurar desempenho às aplicações, apenas em termos relativos. O esquema de prioridade relativa garante que uma aplicação gerando tráfego com uma determinada prioridade, terá desempenho melhor que outra de menor prioridade, isso dependendo da carga da rede. O DiffServ emprega um mecanismo de priorização denominado "comportamento por *hop*" (*Per-Hop Behavior*-PHB), que define comportamentos padrões para o encaminhamento de datagramas. Caso todos os roteadores de um determinado caminho tenham DiffServ implementados, o encaminhamento de um datagrama marcado com um determinado PHB terá um perfil de encaminhamento uniforme.

A definição de PHBs é bastante flexível e podem ser caracterizadas por funções tais como:

- recursos (banda, *buffer*);
- parâmetros de QoS (atraso, *jitter*, taxas de perdas);
- presença de outros PHBs.

4.2.4 Serviços Integrados (IntServ)

A técnica de serviços integrados é focada no fluxo individual de pacotes, entre os mesmos pontos de origem e destino. Nesta abordagem cada fluxo pode requisitar níveis diferentes de serviços à rede, sendo nitidamente quantificados como banda mínima necessária ou tolerante a um atraso máximo definido. Os quatro componentes básicos do IntServ são:

- **Unidade de controle de admissão** - que identifica se a rede pode suprir o serviço requisitado;
- **Unidade de classificação** - inspeciona os campos dos pacotes para determinar suas classes e níveis de serviços definidos;
- **Unidade de *schedule*** - aplica um ou mais mecanismos de gerência de tráfego para garantir que o pacote seja transmitido à rede, a tempo de satisfazer a banda e atraso adequados ao tipo de fluxos;
- **Protocolo de reserva de recursos RSVP** - que é responsável pela sinalização dos níveis de requisitos aos nós da rede.

O Protocolo RSVP

Definido na RFC 2205, o protocolo RSVP (*Resource Reservation Protocol*) não é um protocolo de roteamento, mas trabalha junto com o protocolo de roteamento, desempenhando a função de sinalizar aos nós a necessidade de reserva de recursos na rede, permitindo que os pacotes possam trafegar

conforme qualidade de serviço definida.

Suas principais funcionalidades são:

- suporte a *unicast* e *multicast*;
- reserva de recursos da rede somente no sentido exigido;
- transparência quanto ao tipo de dados;
- atualização periódica quanto à capacidade e topologia da rede.

O RSVP trabalha com quatro tipos básicos de mensagens, que são:

- ***reservation-request messages*** - enviada pelo receptor ou transmissor, percorrendo o caminho indicado pela *path message* e solicitando reserva dos recursos necessários;
- ***path messages*** - enviado pelo transmissor para o receptor, que percorre todos os roteadores indicados como sendo o melhor caminho pelo protocolo de roteamento. Esta mensagem contém informações sobre o caminho, com objetivo de informar ao receptor os recursos;
- ***error and confirmation messages*** - existem três tipos diferentes de mensagens:
  a ***path-error messages*** - quando ocorre um erro no registro de caminho de uma *path message*, esta mensagem é retornada ao transmissor;
  b ***reservation-request error messages*** - informa ao receptor erro de admissão, banda indisponível, serviços não suportados ou caminho ambíguo, em retorno a um *reservation-request message*;
  c ***reservation-request acknowledgment messages*** - enviada ao receptor como confirmação de recebimento da *reservation-request message*.
- ***teardown messages*** - desfaz o caminho de reserva de recursos sem esperar por um *timeout*.

4.3 Mecanismo para atingir Qualidade de Serviço

4.3.1 Canceladores de Eco

O cancelador de eco constrói uma estimativa de eco para removê-la do sinal que chega. O eco é modelado com uma estimativa de sinais, semelhantes ao sinal de entrada, porém atrasados e com amplitude menor (uma convolução do sinal de entrada, como a da resposta impulsiva do canal).

Os canceladores VoIP existentes comercialmente cancelam apenas o eco da terminação mais próxima. Para um cancelamento efetivo, é necessária a existência de canceladores em todos os *gateways* VoIP da rede. Não há como garantir que todos os equipamentos utilizados na rede IP tenham um

cancelador de eco. Caso não tiverem, não adianta um alto investimento em um cancelador na rede local, se o eco vem remotamente na rede IP [Rachid and Meloni, 2004].

Basicamente os canceladores de eco elétricos são filtros adaptativos (FIR: resposta finita ao impulso) colocados na rede, principalmente em centrais de comutação internacionais.

O desempenho de um cancelador de eco envolve parâmetros, tais como a melhoria da perda por retorno de eco que é dado pela redução do nível de eco entre as portas de entrada e saída, e o tamanho da janela que modela a resposta ao impulso.

4.3.2 Reconstrução de pacotes

As perdas de pacotes são ocasionadas, principalmente pelos congestionamentos momentâneos encontrados na Internet, os quais provocam perdas de até 50% em redes internacionais e 12% em redes nacionais [Lin, 1998]. Assim, técnicas como reconstrução de pacotes perdidos auxiliam na melhoria da qualidade de serviço.

Esta técnica usa as amostras adjacentes à perdida para realizar a estimativa de acordo com algum critério pré-estabelecido. Estas são principalmente utilizadas em codificadores em forma de onda [Lin, 1998].

Um método típico de interpolação é estimar as amostras perdidas como a média temporal das amostras vizinhas, como mostrado na Figura 4.2. Este critério é bastante razoável quando as amostras do sinal de voz são correlacionadas, mas o resultados são ruins quando as amostras do sinal são descorrelacionadas.

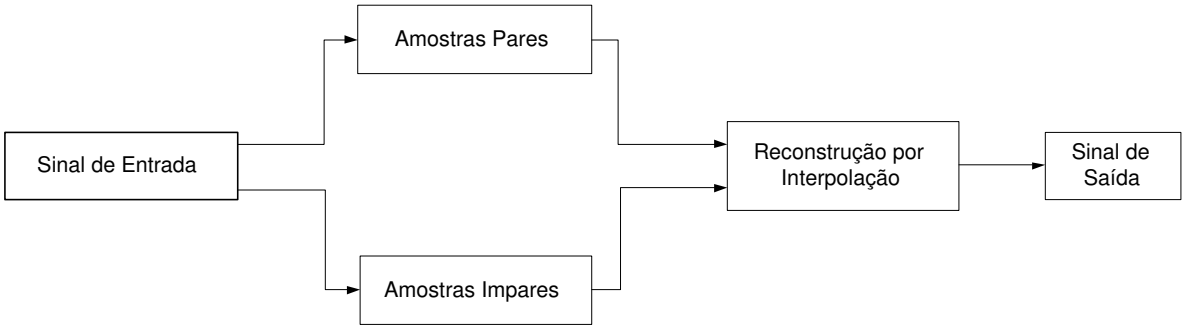

Figura 4.2: Interpolação por média temporal.

A utilização de filtros adaptativos, tais como o LMS [Lin, 1998] para a interpolação e estimativas das amostras perdidas, apresentam resultados melhores que os alcançados com a média temporal.

Outro método de reconstrução de pacotes é usando interpolação de parâmetros de codificadores da família CELP [Marques and Meloni, 2004].

4.3.3 Controle do Buffer de Jitter

Os efeitos do *jitter* ou variações de atraso em VoIP podem ser eliminados ou reduzidos pela utilizando de *buffers* na recepção denomidados de *buffer* de *jitter*. O *buffer* de *jitter* armazena temporariamente os pacotes de voz recebidos, introduzindo um atraso adicional antes de enviá-los ao decodificadores, de modo a igualar o atraso total sofrido por todos os pacotes. Cita-se como exemplo, um sistema de controle do *buffer* de *jitter* introduz atrasos adicionais de modo que todo pacote recebido tenha um atraso total de 120 ms. Se um pacote sofreu um atraso de 100ms na rede, ele sofre um atraso adicional de 20 ms, e se outro pacote sofreu um atraso de 90 ms na rede, ele sofre um atraso adicional de 30 ms.

A escolha do atraso do *buffer* de *jitter* é crítica. Para que o *buffer* de *jitter* seja capaz de eliminar completamente o efeito do *jitter*, sua janela de atraso deve ser igual à diferença entre o máximo e mínimo atraso na rede. Por exemplo, se o mínimo atraso na rede é de 70 ms e o máximo é de 130 ms, com um atraso nominal de 100 ms, um *buffer* de *jitter* com janela de atraso de 60 ms (os atrasos introduzidos pelo *buffer* são de 0 e 60 ms) é capaz de eliminar completamente o *jitter*, fazendo com que todos os pacotes recebidos tenham atrasos de 130 ms.

Se a variação do atraso na rede for muito grande, caso da Internet por exemplo, a tentativa de se eliminar o *jitter* aumentando-se a janela do *buffer* de *jitter* leva a pacotes sem *jitter* mas com atrasos totais inaceitavelmente altos, e portanto inúteis. Uma solução simples é fixar o tamanho do *buffer* de *jitter* num valor adequado. Esta solução no entanto pode levar à perdas de pacotes causada por atrasos excessivos na rede ou por *overflow* do *buffer* de *jitter*. Para tornar a operação do sistema mais eficiente é desejável que o tamanho da janela de tempo do *buffer* de *jitter* seja adaptativo. Ou seja, a janela de tempo do *buffer* aumenta (até um certo limite) ou diminua acompanhando a variação do *jitter* na rede. A característica de adaptação do *buffer* de *jitter* é conseguida basicamente através da medição e comparação contínua do atraso dos pacotes que chegam (atraso instantâneo) com uma referência (atraso de referência). Esta referência é continuamente atualizada com base em uma média móvel do atraso dos pacotes recebidos em um determinado período de tempo, podendo-se empregar maior peso para os pacotes recebidos mais recentemente. Com isso consegue-se um ajuste dinâmico do tamanho da janela de tempo do *buffer* de *jitter* ao longo do tempo, de modo que o atraso instantâneo nunca se afaste muito do atraso de referência [André and Cunha, 1999]. Em geral, a escolha do tamanho da janela é uma solução de compromisso entre atraso do pacote e taxa de perda de pacotes. Janelas maiores resultam em maiores atrasos e menores perdas de pacotes, enquanto janelas menores resulta em atrasos menores e maiores perdas de pacotes.

Capítulo 5

Sistema Operacional μClinux

O μClinux ou micro-controle Linux é um sistema operacional baseado no sistema Linux, que foi desenvolvido para sistemas embarcados. A limitação de memória é o principal fator para o baixo custo destes dispositivos, onde o preço de componentes é crucial para empresas desse ramo[Collins, 2000]. O μClinux é uma solução gratuita, pois pertence a comunidade de software livre e surgiu da necessidade de se rodar um sistema estável e robusto nesses processadores.

O μClinux possui basicamente quase toda funcionalidade do Linux, sendo encontrado nos dois ramos principais do *kernel*, 2.0 e 2.4 [John, 2002]. A parte ligada ao gerenciamento de memória foi reescrita, além de várias outras pequenas modificações no *kernel*. A Interface do Programa de Aplicação (do inglês Application Program Interface - API) fornecida pelo μClinux é praticamente idêntica à do Linux, permitindo que várias aplicações Linux possam rodar sem problemas no μClinux. Foi necessário criar também uma nova biblioteca C (libc). A primeira biblioteca C criada foi denominada de uC-libc, mas devido a sua falta de compatibilidade com outras implementações de bibliotecas C acabou gerando um trabalho derivado que originou uma nova biblioteca, denominada de uClibc. A uClibc, além de manter a compatibilidade com outras implementações de bibliotecas C, está também preparada para ser compilada para arquiteturas com ou sem MMU (*Memory Management Unit*).

5.1 Arquitetura do μClinux

O μClinux é a forma mais popular do Linux embarcado [Networks, 2002]. Atualmente o *kernel* suporta uma grande variedade de plataformas, entre elas está o processador PowerQUICC II utilizado neste trabalho; citando-se o Axis ETRAX, ARM, ATARI 68k, etc. A única diferença entre o Linux e o μClinux é o suporte para processador sem MMU e o fato dele ser projetado para um tamanho de memória bastante reduzido. O μClinux também tem suporte para redes de computadores incluindo uma completa pilha TCP/IP. Este sistema operacional suporta uma grande variedade de

sistema de arquivos, tais como EXT2, SAMBA, NFS (*Network File Systems*), entre outros. Tem-se tentado desenvolver um μClinux o mais compatível possível com o Linux padrão, fazendo com que as aplicações desenvolvidas dentro do ambiente Linux possam ser suportadas em ambiente μClinux, geralmente com poucas mudanças.

Processadores com MMU, como o Pentium e PowerPC, têm embutido no seu microcódigo o suporte ao gerenciamento de memória, que permite que os processos rodem em um espaço de endereçamento virtual, deixando para o sistema operacional o controle e mapeamento entre os endereçamentos virtuais e reais do sistema. Isto gera um sistema mais seguro, em que um processo não consegue invadir a área de memória do outro e a comunicação entre eles só pode ser feita através de funções do sistema operacional. Já os processos baseados em μClinux compartilham o mesmo espaço real de endereçamento, uma vez que não existe este mapeamento entre endereçamento real e virtual, permitindo a utilização de microprocecssadores sem MMU (chamados de NOMMU no *kernel*), gerando um sistema final com custo reduzido.

A principal vantagem do *kernel* do μClinux em comparação ao *kernel* do Linux é o tamanho (conhecido como o Linux dos pequenos). Quando o *kernel* é compilado com as opções básicas, tais como suporte para o processador, interpretador de comandos (console) e ferramentas de rede, o seu tamanho é da ordem de 600 kB (quilo bytes). Para esse tamanho é requerido 1 MB de memória livre para o μClinux ser executado. O arquivo imagem, que é composto do *kernel* do μClinux, sistema de arquivos e aplicação VoIP, é gerado num formato especial de compactação, tendo o tamanho variando de 500 até 900 kB. Para esse trabalho, o *kernel* foi compilado com suportes necessários para trabalhar em rede, incluindo alguns protocolos tais como protocolo TCP/IP.

Na fase de desenvolvimento do sistema foi gerada uma imagem somente com o *kernel* do μClinux, sendo montado um sistema de arquivos via rede, mais conhecido por NFS, em um servidor remoto Linux. Esse ambiente foi utilizado para realização de testes do sistema, evitando o trabalho e o tempo de gerar e gravar uma nova imagem na placa a cada mudança na implementação VoIP.

O *kernel* do μClinux pode ser encontrado na Internet gratuitamente, na comunidade de software livre, assim como todas as ferramentas e pacotes do seu ambiente de compilação. O pacote de distribuição inclui também algumas bibliotecas, licenciada por LGPL [Nikkannen, 2003]. As ferramentas de geração do *kernel* podem ser obtidos na página oficial do μClinux [uClinux's Home Page, 2002b].

5.1.1 Bibliotecas μClinux

O μClinux usa uma versão simplificada da biblioteca C padrão, que foi desenvolvida para ser utilizada no μClinux. Todas as mudanças no desenvolvimento da biblioteca uClibc foram feitas com o objetivo de projetar uma biblioteca C otimizada. O μClinux também provê as APIs para a biblioteca padrão libC do Linux, na qual se pode migrar as aplicações de sistemas operacionais baseados em POSIX

para o μClinux. Isso tudo é obtido livremente dentro da licença LPGL [uClinux's Home Page, 2002a].

Dentro do ambiente de desenvolvimento pode-se escolher dentre duas bibliotecas, a uC-libc e uClibc dependendo da necessidade da aplicação. Neste trabalho foi escolhida a biblioteca uClibc, pois é uma biblioteca que mantem uma compatibilidade com as implementações da biblioteca C padrão, sendo ainda uma biblioteca mais estável que a uC-libc.

O pacote μClinux inclui também um conjunto de outras bibliotecas que podem ser requeridas por uma aplicação. Algumas bibliotecas podem ser portadas para o μClinux, sendo escolhidas na geração do *kernel*. Essas bibliotecas são mostradas na Tabela 5.1:

Biblioteca	Característica
libgmp	GNU biblioteca para precisão aritmética.
libg	Inclui gtermcap, que é usado para enviar string de controle para o terminal.
libpcap	Prove uma interface independente do sistema para capturar pacotes.
zlib	Biblioteca de compressão de dados.

Tabela 5.1: Bibliotecas portadas para o μClinux.

A distribuição tem também um conjunto de bibliotecas padrões 5.2, que são sempre geradas na compilação.

Biblioteca	Descrição
libatm	Software experimental para ATM (*Asynchronous Transfer Mode*).
libnet	API de rede genérico, que provê acesso à vários protocolos.
libam	Biblioteca de Interface para os módulos de autenticação para o Linux.
libpng	Biblioteca de referência gráfica de rede requerida pela zlib.

Tabela 5.2: Bibliotecas padrões do μClinux.

5.1.2 Dependência do Hardware dentro do μClinux

Normalmente, um sistema embarcado é uma parte de um conjunto maior de um sistema, tal como um computador de bordo de um carro, ou parte do controle de sinalização de um sistema de telecomunicações.

O μClinux vem aumentando o suporte para diferentes tipos de processadores. No *kernel* do μClinux, a separação entre dependência e independência do hardware no código fonte é feita de forma bem clara.

O dispositivo escolhido para este trabalho foi o MPC8265 da família MPC8260 PowerQUICC II,

que é usado em sistemas de telecomunicações em geral. O dispositivo é suportado pelo sistema operacional μClinux, que roda a aplicação de VoIP desenvolvida neste trabalho, sendo possível comunicar-se com um computador pessoal, rodando a mesma aplicação.

5.2 Ferramentas de Desenvolvimento

Para compilar e executar uma aplicação no μClinux, pode-se escolher uma compilação direta no μClinux executado na placa, ou fazer uma compilação cruzada. A compilação cruzada é representada por um conjunto de ferramentas de desenvolvimento instalado em uma máquina pessoal. Estas ferramentas são usadas para fazer uma compilação do *kernel* e da aplicação para a plataforma.

A versão da ferramenta de desenvolvimento usada no trabalho é a powerpc-linux-20010315, que pode gerar vários diferentes tipos de imagens. A ferramenta é baseada no gcc-2.95.3 e inclui alguns componentes básicos como mostrado na Tabela 5.3.

Ferramenta	**Descrição**
binutils	Coleção de ferramentas binárias(ld, as, etc.) Baseada no binutils-2.12 da GNU
powerpc-linux-gcc/g++	Compilador C/C++. Baseado no gcc-2.95.3
elf2flt	Um conversor de elf para flat
genromfs	Ferramenta para criar imagens romfs. Baseada no projeto romfs, genromfs-0.5.1

Tabela 5.3: Ferramentas para Desenvolvimento Cruzado.

Como padrão, as ferramentas de desenvolvimento são sempre instaladas dentro de um diretório chamado `powerpc-linux` e no seguinte caminho `/usr/local/`, onde é instalado e configurado todas as ferramentas necessárias para gerar o *kernel* do μClinux e a aplicação.

O μClinux requer dois tipos de depurador, um para o código fonte do *kernel* e outro para as aplicações. O μClinux provê a habilidade de depurar, usando a ferramenta conhecida como GDB, que permite depurar programas escritos em C/C++ e outras linguagens.

5.3 Configurando e Gerando o μClinux

Compilar um *kernel* do μClinux é o mesmo que compilar o *kernel* de um Linux padrão. A seguir são demonstrados os passos realizados na compilação do μClinux para plataforma PowerQUICC II da família MPC8260 da Motorola®. Para o processo de geração e compilação da imagem foram

utilizadas as ferramentas de desenvolvimento conhecidas como *toolchain*. Essas ferramentas incluem os compiladores cruzados especifico para a plataforma PowerPC e todas as bibliotecas necessárias para a geração do μClinux e da aplicação VoIP.

Para a configuração básica do μClinux, utiliza-se o comando "make xconfig" que abre uma janela principal com um menu base de opções, como mostrado na Figura 5.1.

Figura 5.1: Janela de configuração básica.

Clicando na opção Target Platform Selection na janela da Figura 5.1, uma segunda janela abre com opções de plataforma, como mostrado na Figura 5.2. Na opção Vendor/Product a placa MPC8265 da Motorola é selecionada. Na opção kernel Version é escolhida a versão 2.4 do linux e em Libc Version é escolhida a biblioteca uClibc. Por fim, é selecionada a opção Customize Vendor/User Settings para poder incluir a aplicação de VoIP desenvolvida neste trabalho, como visto na Figura 5.2.

Figura 5.2: Janela de configuração da plataforma.

Depois de escolhido as opções da plataforma alvo, clica-se no botão Main Menu (veja Figura 5.2), que faz o sistema voltar para a janela da Figura 5.1. Para salvar as configurações feitas, deve-se clicar no botão Save and Exit (veja Figura 5.1), que em seguida irá exibir uma janela com

opções de programas à serem adicionados à memória (veja Figura 5.3). Nesta Janela (veja Figura 5.3) clica-se no botão `Miscellaneous Applications`, que abrirá uma nova Janela com variedades de aplicações a serem adicionadas no sistema, como mostra a Figura 5.4. Neste caso é escolhido a opção "yes" do item `RT-DSPhone`, que é a aplicação VoIP desenvolvida neste trabalho.

Figura 5.3: Janela de configuração de aplicações para μClinux.

Figura 5.4: Janela de configuração de aplicações variadas.

Em seguida são geradas as imagens com o comando "make all". Dependendo do tipo de ambiente que será montado na placa, uma imagem deverá ser escolhida, como mostra a Tabela 5.4. Neste trabalho usou-se a imagem `linux.bin` na fase de testes e a imagem `imagem.bin` na fase final da implementação.

Como no Linux, o μClinux também prover um mecanismo para desfazer tudo o que foi feito no sistema. Isto é feito através do comando "make clean", que irá remover os diretórios espelhos e

Imagem	Descrição
image.bin	imagem binária com *kernel*, sistema de arquivos e aplicação
image.elf*	imagem elf com *kernel*, sistema de arquivos e aplicação
linux.bin*	imagem binária só com o *kernel*
romfs.img	imagem do sistema de arquivos

Tabela 5.4: Imagens geradas na compilação do μClinux.

imagens. A opção mais completa é o comando "make distclean", que remove todos os arquivos de configuração criados, juntamente com as configurações básicas de plataforma e bibliotecas. Depois deste comando o sistema retorna ao estado inicial e todas as configurações deverão ser feitas novamente.

As instruções básicas para o μClinux suportar uma placa específica é feita através da inclução de um conjunto de arquivos como mostrado na Tabela 5.5. Os arquivos contêm as configurações básicas de arquitetura do dispositivo alvo e são incluídos no diretório vendors/PRODUTO/MODELO do ambiente μClinux.

Arquivo	**Descrição**
config.linux-2.4	Inclui a configuração padrão do *kernel*.
config.arch	Arquitetura configurada especificamente para compilar a imagem.
config.modules	Será incluído os módulos de configurações se módulos habilitados.
config.uClibc	Usado para selecionar as configurações básicas para a biblioteca uClibc.
Makefile	Constrói o sistema. Inclui as instruções para construir romfs e imagem. Este arquivo irá definir a estrutura de arquivos que irá ser construído no romfs e instala alguns arquivos para a estrutura de arquivos, como rc e inittab.

Tabela 5.5: Arquivos Base de Configuração do Dispositivo.

O ambiente também prover alguns comandos, que podem ser usados para compilar partes específicas do ambiente, como mostra na Tabela 5.6:

5.4 Carregando o Sistema Operacional no Dispositivo

A forma de armazenamento da imagem gerada é feita diretamente na memória. Um caso mais simples de execução do *kernel* é quando a posição inicial do *kernel* é carregado em um endereço da memória que representa o endereço inicial para o processador. Com este tipo de configuração, o *kernel* do μClinux está preparado para ser descomprimido, iniciar as configurações e montar o sistema de

Comando	Descrição
make user_only	Examina através da árvore de diretório dos usuários e faz o que é preciso.
make romfs	Faz somente o sistema de arquivos rom
make image	Roda os genromfs e cria as imagens.
make linux	Compila o *kernel* do Linux.
make lib_only	Compila somente as bibliotecas.
make user_clean	Limpa os diretórios do usuário, removendo todos os arquivos objeto.

Tabela 5.6: Comandos Extras de Compilação.

arquivos.

A opção mais segura e flexível é iniciar a placa com um pequeno código residente chamado de *bootloader*, que serve para iniciar o *offset* da memória. O *bootloader* trata as configurações iniciais da placa, tais como a iniciação básica do hardware, e entre outras tarefas o carregamento da imagem do μClinux para a placa. O *bootloader* utilizado neste trabalho foi o U-Boot (*Universal Boot Loader*), que é usado na plataforma PowerQUICC II. O U-Boot inicia as configurações básicas do processador, tais como reconhecimento dos registradores, mapeamento dos endereços de memória e etc. Depois de iniciar a placa, a imagem pode ser carregada através da interface serial ou Ethernet, utilizando o protocolo TFTP (*Trivial File Transfer Protocol*).

No μClinux o sistema de arquivos e o *kernel* podem ser compilados em partes separadas. Na distribuição padrão do Linux, o sistema de arquivos é embutido como parte do *kernel*. Esta configuração requer um *bootloader* sofisticado, o qual faz um tratamento especial de cada parte na memória. O μClinux permite a partição do sistema de arquivos e *kernel*, sendo posicionados na memória em diferentes seguimentos.

O arquivo imagem que é carregado para memória está comprimido. Quando o comando go PC (onde PC é o endereço inicial da imagem) é executado, a primeira fase é descomprimir o *kernel* e o sistema de arquivos e em seguida é iniciar o sistema operacional. Isto requer que o *bootloader* usado seja capaz de tratar a descompressão de arquivos. O tamanho das imagens geradas que podem ser carregadas em Flash ou RAM são apresentadas na Tabela 5.7.

Imagem	Tamanho (MB)
image.bin	2
image.elf*	6.2
linux.bin*	0.86
romfs.img	1.14

Tabela 5.7: Tamanho das imagens geradas.

A partição da memória dentro do μClinux garante que o *bootloader* e algumas outras configurações estejam no início e que o restante da memória seja reservado para a imagem do μCLinux. Uma partição típica de memória pode ser vista na Figura 5.5:

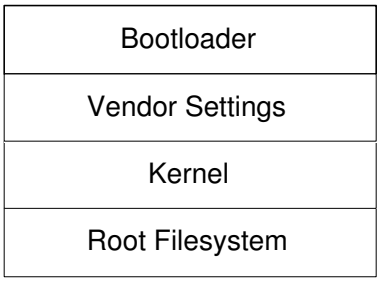

Figura 5.5: Típica partição da memória na plataforma para o μClinux.

O processo de iniciação do μLinux na placa é semelhante a iniciação do Linux em uma máquina pessoal. Hoje a atualização e o processo de desenvolvimento do *kernel* do μClinux estão ocorrendo rapidamente, assim como o número de placas que são suportadas pelo o sistema operacional μClinux.

O *kernel* do Linux já provê suporte para software em tempo real, com características *Preemptive*, sendo que o *kernel* do μClinux também tem algum enfoque para o suporte em tempo real. Existem dois tipos de enfoque para o suporte em tempo real no μClinux, *Real-Time Linux* (RTLinux) e *Real-Time Application interface* (RTAI).

RTLinux - É um projeto baseado no princípio do *kernel* duplo, em que um pequeno *Real-Time kernel* (RTKernel) coexiste com um sistema operacional de propósito geral como o Linux. O RTLinux foi desenvolvido no Instituto de Tecnologia do Novo México. Enquanto uma versão código aberto do RTLunix ainda está disponível, muito dos trabalhos está sendo desenvolvido numa versão proprietária chamada RTLinux/Pro oferecida pela FSM Labs.

RTAI - É uma melhoria do RTLinux, desenvolvido pelo Instituto Politécnico de Milão. O RTAI usa abstração de interrupção para adicionar características determinísticas de *real-time* para o *kernel* do Linux. Este consiste basicamente de um despachante de interrupções: RTAI captura principalmente as interrupções de periféricos e se necessário re-encaminha para o Linux. O RTAI não é uma modificação intrusa do *kernel*; usando o conceito de HAL (*hardware abstraction layer*) para pegar informações do Linux e capturar algumas funções fundamentais. O HAL prover poucas dependências para o *kernel* do Linux.

Capítulo 6

O Sistema SIP Implementado

Com o aumento dos serviços de comunicações para Internet e o crescente uso de dispositivos embarcados nos meios de comunicações, há grande interesse da comunidade acadêmica e empresarial no estudo de sistema VoIP. Uma das principais características no desenvolvimento de um sistema VoIP para comunicação pela Internet é a interoperabilidade entre os sistemas, pois possibilita a comunicação entre os diferentes tipos de sistemas implementados. Neste trabalho houve uma preocupação em garantir um certo nível de interoperabilidade com outros softwares, como a flexibilidade de utilização do sistema em várias plataformas. Para isso, foram utilizados protocolos normatizados pelo IETF, disponibilizados na Internet em seu *site* oficial e a utilização de uma biblioteca chamada oSIP (open SIP) [GNU, 2002], desenvolvida pela GNU software livre.

O sistema foi desenvolvido em linguagem ANSI C padrão e foram utilizandos as ferramentas de compilação gcc (GNU) e o compilador cruzado powerpc-linux-gcc gerado para o processador PowerQUICC II (Motorola). Esse Processador é amplamente utilizado nos sistemas de comunicações atuais. A escolha do ambiente Linux deveu-se a grande robustez desse sistema operacional e a sua crescente utilização no mundo das comunicações e sistemas embarcados. Além disso, o Linux é um sistema operacional aberto e livre, para o qual as ferramentas e softwares livres para o desenvolvimento do trabalho são fáceis de se encontrar.

6.1 Biblioteca oSIP Utilizada na Implementação

A pilha oSIP é uma ferramenta desenvolvida pela GNU, para prover suporte no desenvolvimento de aplicações SIP, sendo implementada somente nas camadas mais baixas do protocolo SIP, a serem descritos na subsecção 6.1.2. A biblioteca oSIP foi desenvolvida em C padrão e tem tamanho e código pequenos, sendo possível o uso no desenvolvimento de soft-phone IP, bem como software para SIP embarcado. Portando, oSIP não é destinado somente para uma implementação particular, podendo

ser configurado para aplicações SIP ponto-a-ponto, *proxy* e algum tipo mais específico de *User Agent*, tal como B2BUA (*Back-to-Back User Agent*).

6.1.1 Nível de Transação do oSIP

O protocolo SIP é dividido em camadas; a camada mais baixa é de sintaxe e codificação das mensagens. A segunda camada é a camada de transporte, que define como um cliente envia pedidos e recebe respostas, e como um servidor recebe pedidos e envia respostas. A terceira camada é a camada de transação, que é responsável por enviar pedidos pelo *Client Transaction* (usando a camada de transporte) para um *Server Transaction*. A camada de transação trata as retransmissões, associando as repostas aos pedidos e disparo de *timeouts*. A camada acima da camada de transação é chamada *Transaction User* (TU), onde cada entidade SIP é uma transação do usuário. Quando a TU quer enviar um pedido, ele cria uma instância da transação cliente (CT) e passa o pedido junto com o endereço IP do destinatário, porta e tipo de transporte a ser usado na conexão.

Toda transação SIP tem um lado cliente e um lado servidor, também conhecido como transação cliente e transação servidor. A entidade lógica *Client Transaction* é responsável por receber pedidos da TU e entregar esses pedidos (usando a camada de transporte) para uma *Server Transaction*, que recebe e repassa as repostas para a TU. O objetivo de uma transação servidor é receber pedidos da camada de transporte e repassar para TU. A ST deve também tratar os *timeouts*, assim como receber uma reposta da TU e passar para a camada de transporte para ser transmitida pela rede.

6.1.2 Características da Biblioteca oSIP

A principal característica implementada no oSIP é um SIP *parser*. O SIP parser é capaz de fazer a análise das mensagens de pedidos e respostas SIP e SDP. Atualmente a biblioteca oSIP é capaz de analisar um conjunto de cabeçalhos SIP como *Via*, *CALL-ID*, *To*, *From*, *Contact*, *CSeq*, *Content-Type*. Todos os outros cabeçalhos SIP são gravados como *strings*.

A característica mais interessante e complexa implementada pela pilha oSIP é representada pelas quatro máquinas de estados que são aplicadas para diferentes transações definida pela rfc3261 [Rosemberg et al., 1999]. O protocolo SIP define quatro máquinas de estados, como listado abaixo:

- ICT : Invite Client Transaction
- NICT: Non Invite Client Transaction
- IST : Invite Server Transaction
- NIST: Non Invite Server Transaction

Em seguida é descrito o funcionamento das máquinas de estados *Invite Client Transaction* (ICT) e *Invite Server Transaction* (IST), explicando os processos de envio e recepção de um INVITE por parte do UAC e UAS, respectivamente. Devido ao funcionamento das máquinas de estados *Non Invite Client Transaction* (NICT) e *Non Invite Server Transaction* (NIST) serem similares as máquinas de estados ICT e IST explicadas nos dois pontos seguintes, estas são apresentadas apenas no Apêndice B.

Diagrama de Estado para um ICT

A transação INVITE consiste de uma sinalização de três vias, onde uma transação cliente envia um INVITE, uma transação servidor responde e em seguida uma transação cliente envia um ACK. Para transações não confiáveis (e.g. usando UDP), a transação cliente retransmite pedidos em um intervalo que inicia com o valor de T_1 segundos e dobra a cada nova retransmissão. T_1 é um valor estimado pelo *round-trip time* (RTT), e seu valor padrão é 500 ms.

O diagrama de estado para uma *Invite Client Transaction* é mostrado na Figura 6.1. Quando a TU passa um INVITE para uma transação cliente, o diagrama de estado entra no estado *Calling* e passa o INVITE para a camada de transporte para ser transmitido. Se estiver utilizando um protocolo não confiável, a transação cliente ativa o timer A com o valor de T_1. Se o timer A disparar, a transação cliente deve retransmitir o pedido INVITE, passando-o para a camada de transporte e reiniciando o timer A com valor 2*T_1. Depois de entrar no estado *Calling*, o timer B é ativado com o valor de 64*T_1, que funciona como um *timeout* para a recepção de reposta. Caso esse *timeout* dispare ou ocorra um erro de transporte, a máquina passa para o estado *Terminated* e a TU é avisado do acontecimento.

Se a transação cliente recebe uma resposta provisória (1xx) enquanto estava no estado *Calling*, esta transita para o estado *Proceeding* e passa a resposta para TU. No estado *Proceeding* a transação cliente não deve mais enviar pedido INVITE e todas as respostas provisórias devem ser passadas para TU.

Quando ambos os estado *Calling* e *Proceeding* receberem uma resposta entre 300-699, a máquina de estados comuta para o estado *Completed*. A transação cliente deve passar a resposta para TU e gerar uma reposta ACK para a camada de transporte para a transmissão. O ACK deve ser enviado para o mesmo endereço, porta e tipo de transporte enviado no pedido inicial. A transação cliente deve ativar o timer D com valor pelo menos de 32 segundos para transporte não confiável e zero para transporte confiável. Qualquer retransmissão de reposta final recebida enquanto no estado *Completed* deve causar um envio de um ACK.

Se o timer D dispara enquanto a transação cliente está no estado *Completed*, a transação cliente passa para o estado *Terminated* e deve informar o *timeout* para TU .

Quando ambos os estados *Calling* e *Proceeding* receberem uma reposta 2xx, a transação cliente

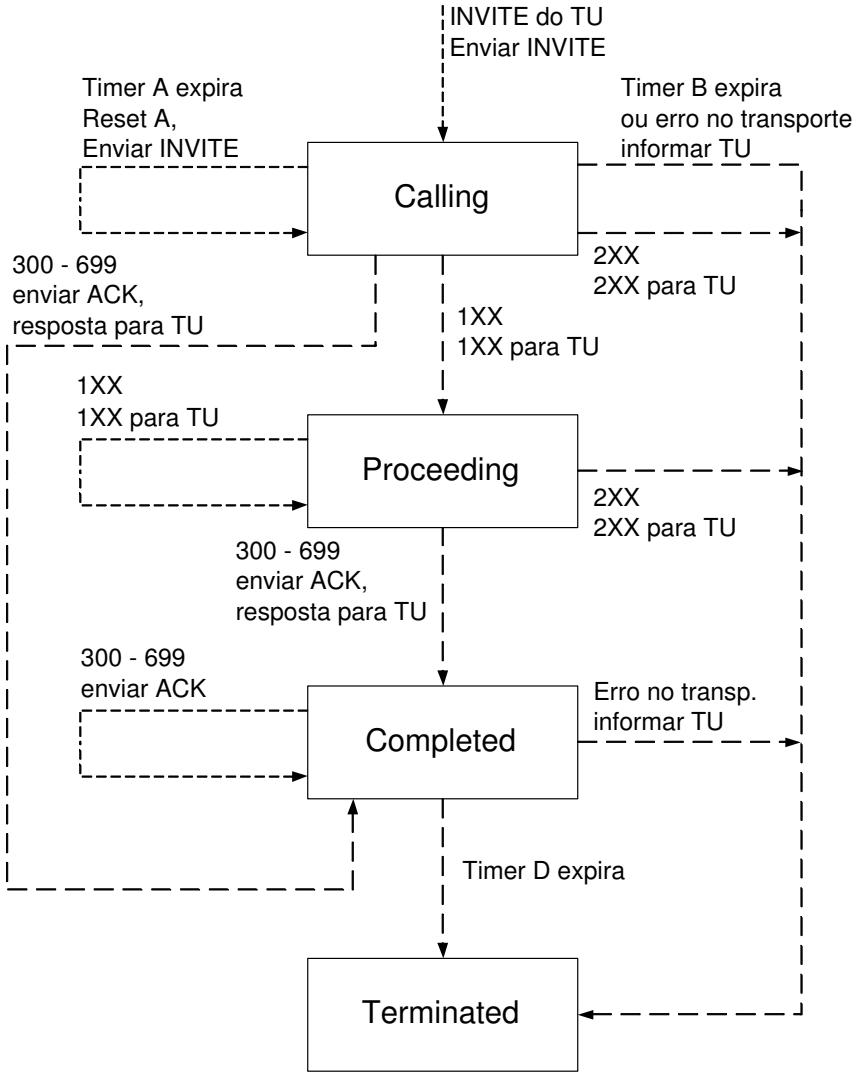

Figura 6.1: Invite Client Transaction.

deve passar para o estado *Terminated* e a resposta deve ser passada para TU, estabelecendo neste caso um diálogo entre o UAC e UAS.

Quando a transação cliente entra no estado *Terminated* , a *Client Transaction* deve ser destruída. Para o UAC, a recepção de uma reposta final deve gerar o envio de um ACK, que é passado para camada de transporte para ser transmitido pela rede. Após o envio do ACK, a aplicação está pronta para realizar uma comunicação.

Diagrama de Estado para um IST

Uma transação servidor é criada quando um pedido INVITE é recebido, como mostrado no diagrama de estado da Figura 6.2. Quando uma transação servidor recebe um pedido INVITE, esta entra no estado *Proceeding*. A transação servidor deve gerar uma resposta 100 TRYING, se a TU não gerar uma resposta final dentro de 200 milisegundos. Enquanto a transação servidor estiver no estado *Proceeding*, todas as respostas provisórias recebidas da TU devem ser passadas para camada de transporte e transmitidas pela rede. Se for recebida uma retransmissão de um INVITE durante esse estado, a transação servidor deve transmitir a última resposta provisória recebida da TU para essa retransmissão.

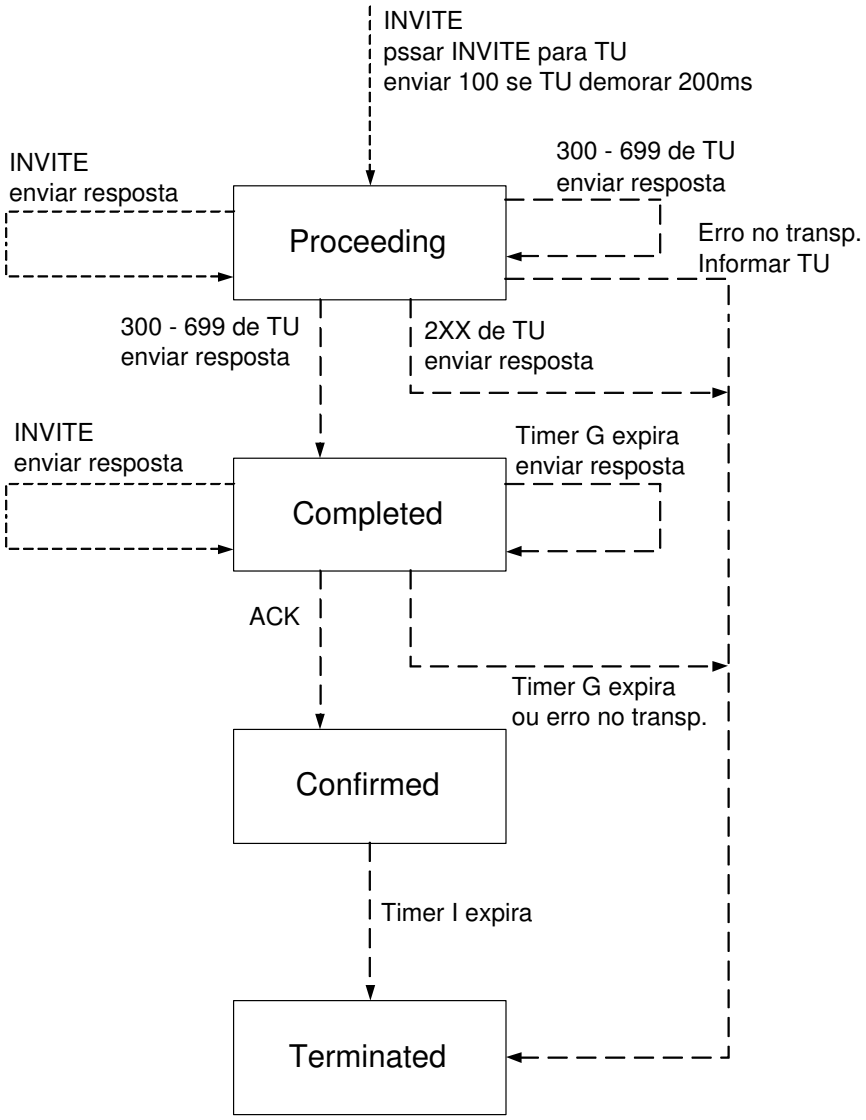

Figura 6.2: Invite Server Transaction.

Se durante o estado *Proceeding*, a TU passa uma reposta 2xx para a transação servidor, esta deve passar essa resposta para a camada de transporte para transmissão e deve comutar para o estado *Terminated*. A resposta 2xx não é retransmitida pela transação servidor e a retransmissão de resposta 2xx deve ser tratada pela TU.

No estado *Proceeding*, se a transação servidor receber uma resposta entre 300-699 da TU, esta deve ser passada para camada de transporte para transmissão e a transação servidor deve entrar no estado *Completed*. Para transporte não confiável, um *timer* G deve ser ativado com o valor T_1 segundos.

Quando no estado *Completed*, o *timer* H é ativado com um valor de 64*T_1 independente do tipo de transporte. Este *timer* determina o tempo que uma transação cliente irá continuar retransmitindo um pedido. Cada vez que o timer G dispara, a resposta é retransmitida e em seguida o timer G é iniciado com o valor mim(2*T_1,T_2), onde o valor de T_2 é 4 segundos. Se durante o estado *Completed* é recebido um ACK, a transação servidor deve passar para estado *Confirmed* e os *timers* devem ser desativados. Se ainda no estado *Completed*, o timer H disparar, isso implica que houve um erro de transporte, a TU deve ser informada que uma falha de transação ocorreu e a transação servidor passa para o estado *Terminated*.

O propósito do estado *Confirmed* é absorver alguma mensagem adicional ACK que chegar. Neste estado, o *timer* I é iniciado com um valor T_4 (5 segundos) para transporte não confiável e zero segundos para transporte confiável. Uma vez que o *timer* I dispare, a transação servidor deve passar para o estado *Terminated*. Quando a transação está no estado *Terminated*, a transação servidor deve ser destruída, como na transação cliente. Isto é preciso para assegurar a confiabilidade da resposta 2xx no envio de um INVITE.

6.2 Descrição do Sistema SIP Implementado

O sistema SIP implementado é composto de uma entidade Phone-Cliente, responsável por enviar um pedido SIP e uma entidade Phone-Servidor, responsável por responder esse pedido. Para a especificação e descrição do sistema RT-DSPhone desenvolvido neste trabalho, foi utilizado uma linguagem chamada SDL (*Specification and Description Language*). SDL é uma linguagem padrão para especificar e descrever sistemas de comunicações. A ferramenta SDL foi desenvolvida pela CCITT (*Comité Consultatif International Téléphonique*), hoje ITU-T e a recomendação Z.100 do ITU-T [Ellsberger et al., 1997].

O sistema foi implementado utilizando-se a ferramenta de desenvolvimento Cinderella [Page, 2002], que usa a linguagem SDL para o desenvolvimento de um sistema. Escolheu-se essa ferramenta devido à qualidade oferecida pela mesma e ao fato de ser uma ferramenta para especificar sistemas de telecomunicações interativos e gerar códigos. Foi feita uma descrição e especificação

do sistema RT-DSPhone no software Cinderella, como descrito nas secções seguintes. Também se fez uso do Cinderella para realizar simulações e testes do sistema desenvolvido. Por fim foi usado o Cinderella para gerar um pseudocódigo e este foi portado para a linguagem C.

6.2.1 Sistema Atuando Como Cliente

Um sistema cliente é uma entidade lógica que cria um novo pedido e usa a máquina de estados apresentada nesta seção para enviá-lo. Esta característica acontece somente enquanto estiver ocorrendo a transação. Em outras palavras, se o sistema inicia um pedido, este atua como cliente enquanto durar essa transação. A Figura 6.3 mostra a máquina de estados do sistema realizando transação como um cliente no envio de um pedido SIP.

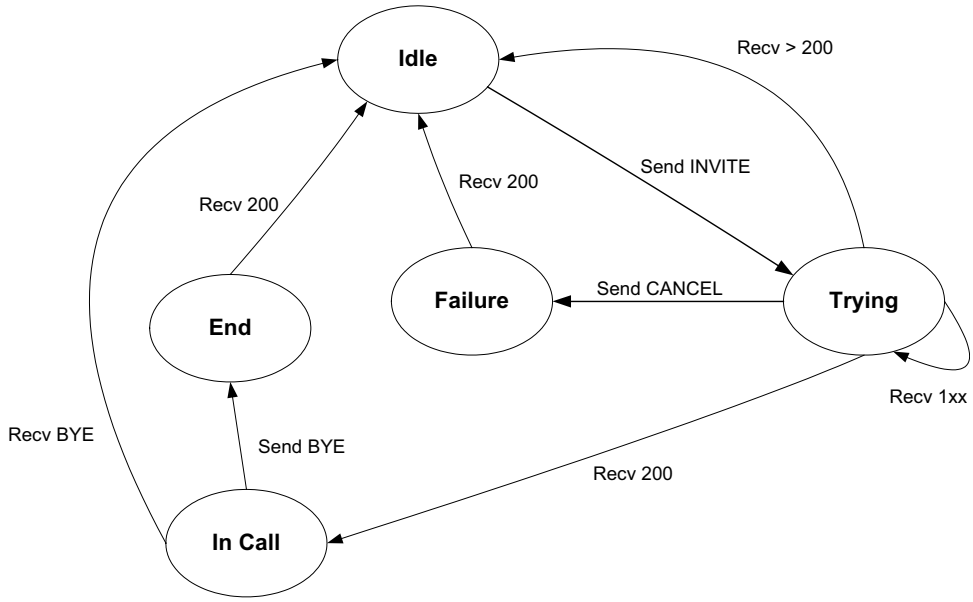

Figura 6.3: Máquina de estados finita para um sistema cliente.

Sistema Phone Cliente

O sistema **Phone_Cliente** consiste de dois blocos chamados de **UAC** e **Transport_Layer**, como mostra a Figura 6.4. O bloco **UAC** é responsável pela interface com o usuário e a camada de transporte, por onde vai ser enviada e recebida uma mensagem SIP. Os sinais **Tentando**, **Conectado**, **Ausente**, **Encerrar** e **Sair** são enviados para o usuário pelo bloco **UAC**, através do canal **InpUser**, para mostrar o estado em que se encontra a sinalização. O sinal **Option** recebe do usuário, via canal **InpUser**, uma opção definida pelo tipo **escolha**, representado pelos caracteres (i=INVITE), (b=BYE) e (q=Sair). O

bloco **Transport_Layer** recebe do bloco **UAC**, via canal **sndT**, uma mensagem de pedido SIP representada pelo sinal **pedido**, convertendo essa mensagem para o formato **string**, repesentada pelo sinal **msg**, e envia pela rede via canal **sndNet**. O bloco **Transport_Layer** é responsável por receber uma mensagem definida pelo sinal **buf**, via canal **rcvNet**. Essa mensagem é convertida para uma mensagem de resposta SIP e passada para o bloco **UAC**. Para essas transações é usado o protocolo UDP.

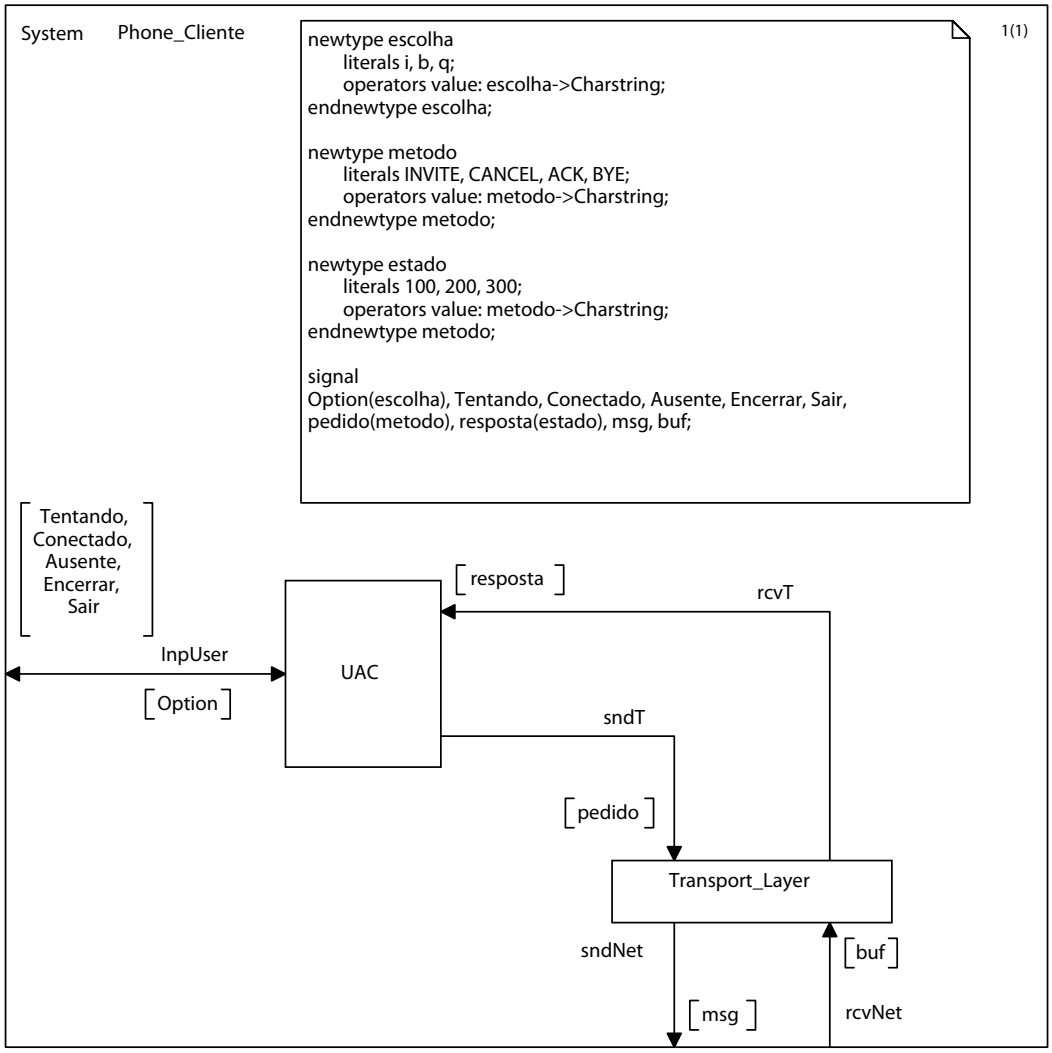

Figura 6.4: Diagrama de bloco do sistema Phone Cliente.

Foram definidos tipos para as mensagens de pedido SIP, representados pelos sinais **INVITE**, **CANCEL**, **ACK** e **BYE**. Para as mensagens de resposta SIP, são definidos tipos representados pelos sinais literais **100**, **200**, **300**.

Bloco *User Agent Client*

A descrição do bloco **UAC** é mostrado na Figura 6.5. Esse bloco contém dois processos, o processo **TU** e o processo **Client_Transaction**, que se comunicam através da rotina **Sync**. O processo **TU** comunica com o canal **InpUser**, via rotina **InpEsc**. Já o processo **Client_Transaction** comunica com o canal **rcvT** e **sndT** através das rotinas **rcvCT** e **sndCT** respectivamente.

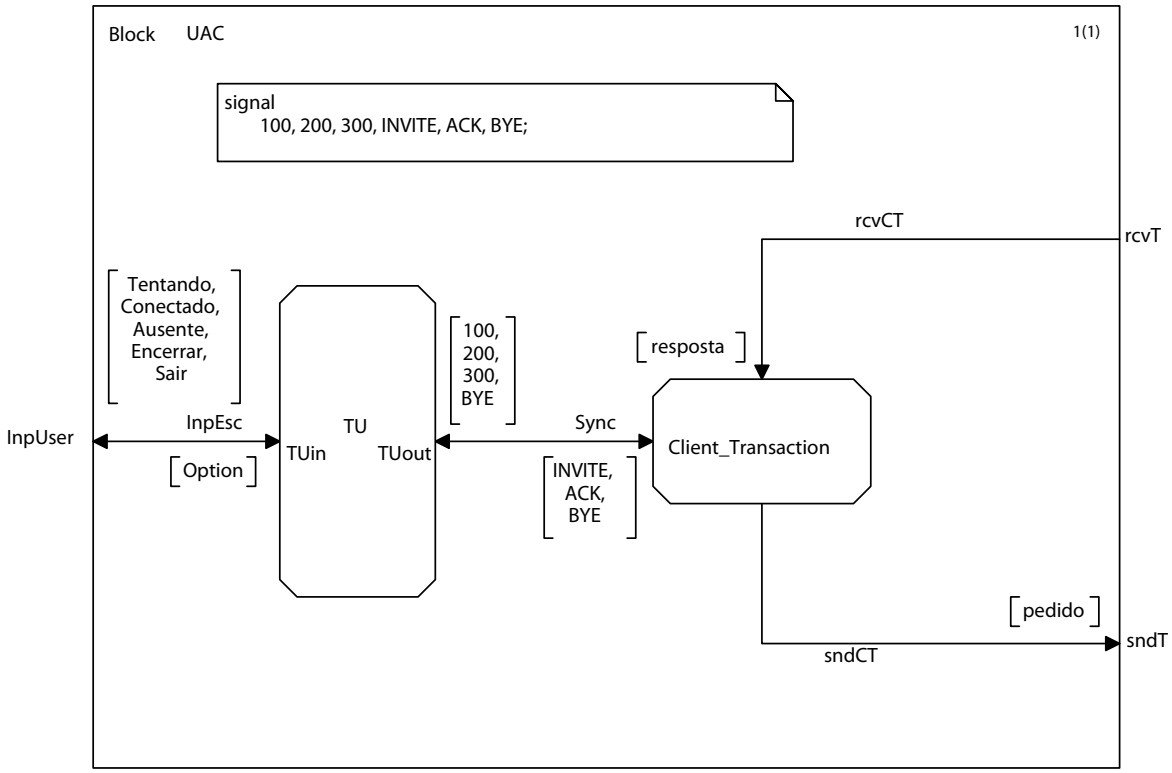

Figura 6.5: Diagrama de bloco UAC.

- **Processo TU**

Para rodar o programa RT-DSPhone é preciso estar no console do Linux e digitar o seguinte comando `siphone -t sip:user@143.106.50.218`. Então o processo **TU** irá iniciar todas as funções do protocolo SIP e receber o endereço SIP do destinatário, depois entra no estado **Idle**, que significa que o sistema está pronto, como mostrado na Figura 6.6.

O usuário inicia uma sessão digitando a opção "i" que será recebido pelo sinal **Option**. Depois de receber o sinal "i", o processo **TU** irá montar a mensagem de pedido SIP, onde os principais campos são o endereço do destinatário e o método da mensagem SIP. Então o processo **TU** envia o sinal **INVITE** para o processo **Client_Transaction** e passa para o estado **Trying**, onde aguarda por uma resposta do processo **Client_Transaction**. Se o processo **TU** receber do processo **Client_Transaction**

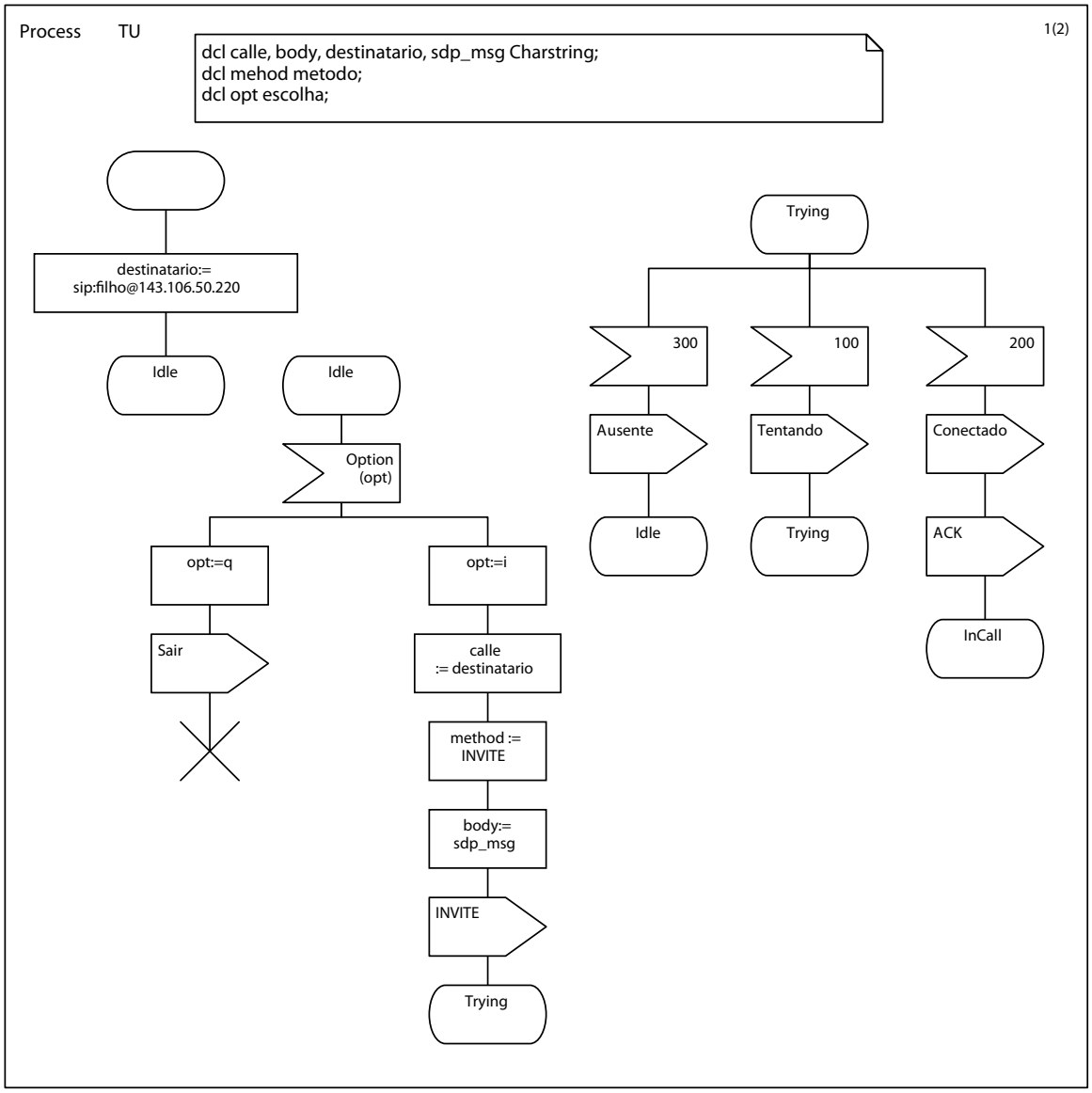

Figura 6.6: Descrição do Processo TU.

uma reposta **300**, significando que o destinatário está ausente, o processo **TU** envia um sinal **Ausente** para o usuário e entra no estado **Idle**. Se receber uma reposta **100**, significando que o destinatário esta processando a resposta, o processo **TU** envia um sinal **Tentando** para o usuário e continua no estado **Trying**. Se ele recebe a resposta **200**, significando que o destinatário aceitou o pedido **INVITE**, o processo **TU** envia um sinal **Conectado** para o usuário e uma mensagem **ACK** direto para a camada de transporte, entrando em seguida no estado **Incall**. O estado **Incall** significa que o sistema já está pronto para abrir um canal de comunicação entre os dois usuários. Qualquer um dos dois usuários pode encerrar a comunicação digitando a opção “b”, que é recebida pelo processo **TU** e envia um

sinal **BYE** para o processo **Client_Transaction**. Então o processo **TU** entra no estado **End** e fica aguardando por um sinal **200** do **Client_Transaction**. Depois de receber o sinal **200**, o **TU** envia um sinal **Encerrar** para o usuário, informando que a comunicação está encerrada. Isto é mostrado no diagrama da Figura 6.7.

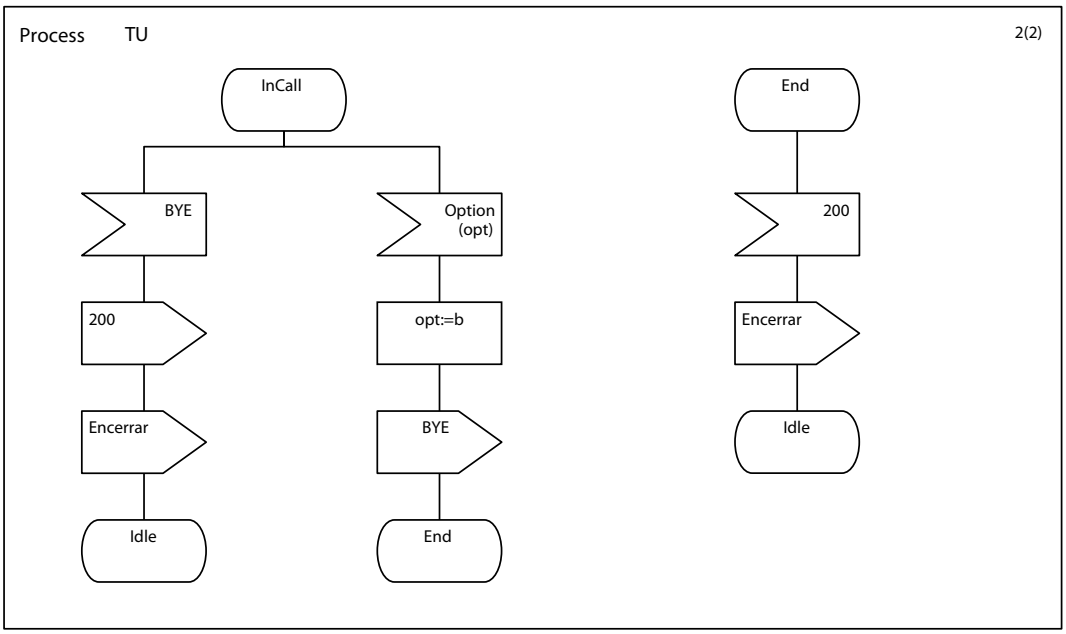

Figura 6.7: Descrição do processo TU.

- **Processo Client_Transaction**

O processo **Client_Transaction** mostrado nas Figuras 6.8 e 6.9 é uma especificação e descrição da máquina de estados ICT, apresentado na seção 6.1.

Bloco *Transport Layer*

O bloco **Transport_Layer** é responsável pela transmissão de pedidos e respostas sobre a rede IP. A camada de transporte é responsável pelo gerenciamento de conexões para protocolo de transporte orientado ou não a conexão como TCP, SCTP, ou UDP. Essas conexões são indexadas por um conjunto formado pelo endereço IP, a porta e o protocolo de transporte. O protocolo de transporte utilizado na implementação foi o UDP, sendo a solução mais utilizada, pois não tem mecanismo de retransmissão, evitando portanto o problema de atraso na funcionalidade de reenvio. Porém o UDP não garante a entrega dos pacotes e nem que estes vão chegar ao destino na ordem de saída. No diagrama mostrado na Figura 6.10, o bloco **Transport_Layer** é formado pelos processos **udp_tl** e **udp_send**. O Processo

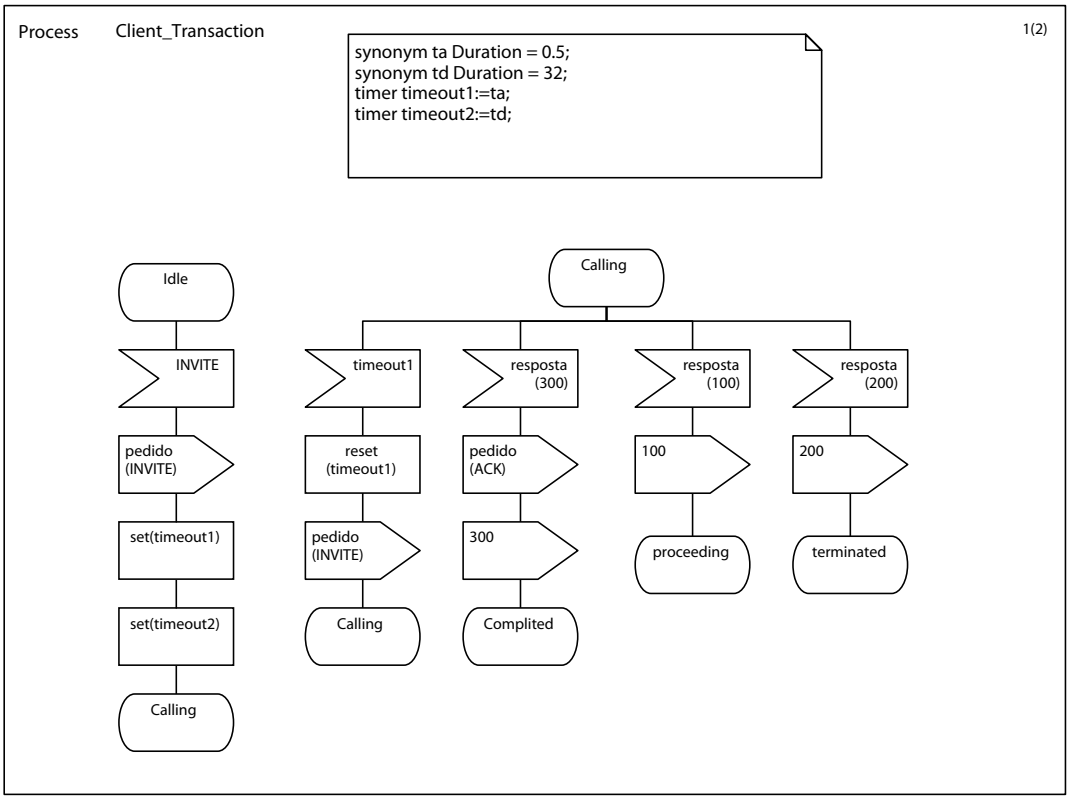

Figura 6.8: Diagrama do Processo Client_Transaction.

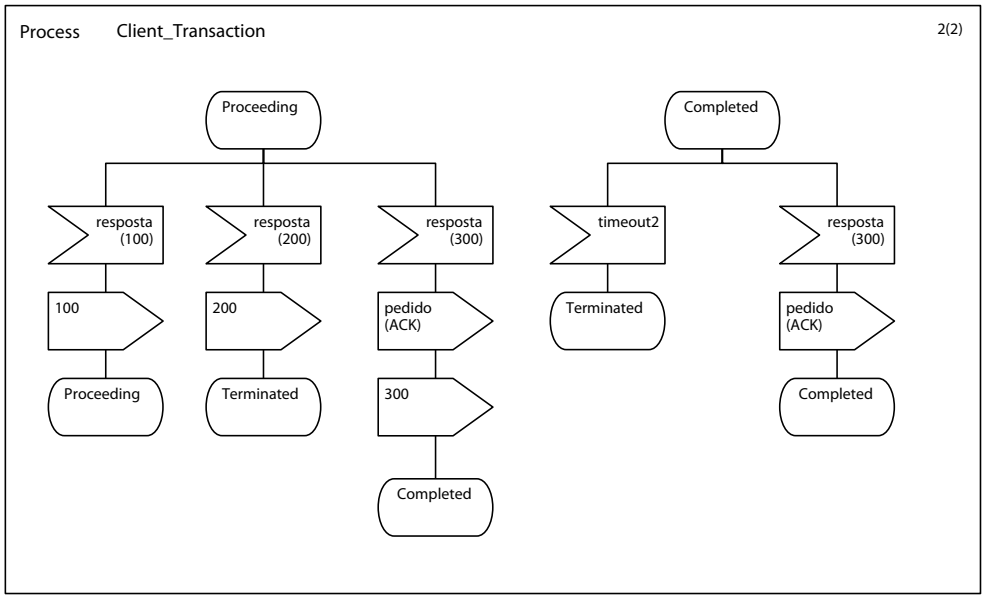

Figura 6.9: Diagrama do Processo Client_Transaction.

udp_tl recebe uma mensagem do canal **rcvNet**, pela rotina **rcv_socket**, e a envia para o canal **rcvT** pela rotina **rcv**. O processo **udp_send** recebe uma mensagem de pedido SIP do canal **sndT**, via rotina **snd**, e a entrega ao canal **sndNet** pela rotina **snd_socket**.

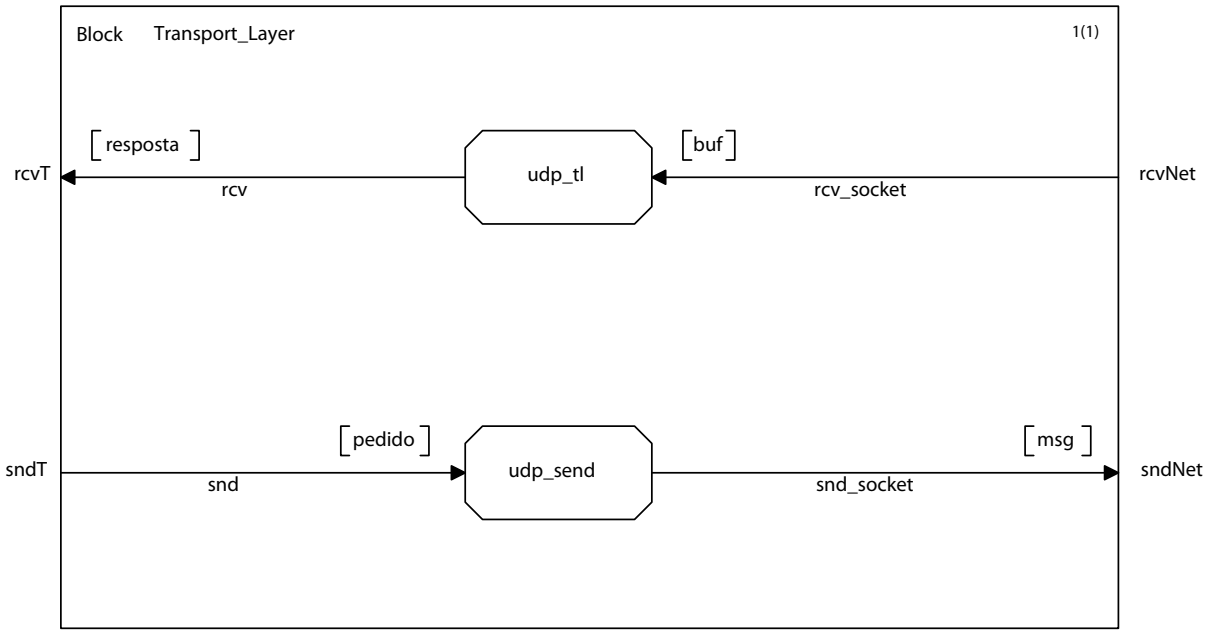

Figura 6.10: Diagrama do bloco Transport_Layer.

- **Processo udp_tl**

Quando o sistema é iniciado, o processo **udp_tl** cria um *thread* que abre um *socket* UDP e ficará sempre ouvindo mensagens na porta 5060 (porta *default*, definida pela RFC 3162). Quando uma mensagem chega nesta porta é armazenada em um buffer e convertida para o formato da mensagem SIP, sendo depois analisada pela API `osip_parse` da biblioteca oSIP. Essa API descobre qual tipo de reposta representa essa mensagem, passando para o **Client_Transaction** essa reposta para ser analisada. Como mostra a Figura 6.11.

- **Processo udp_send**

Quando o bloco **Client_Transaction** for enviar um pedido SIP, este passada a mensagem para o processo **udp_send**, que converte a mensagem SIP para uma mensagem do tipo *string*, através da API `msg_2char` da biblioteca oSIP. O processo **udp_send** envia esta mensagem pela rede, como mostrado na Figura 6.12.

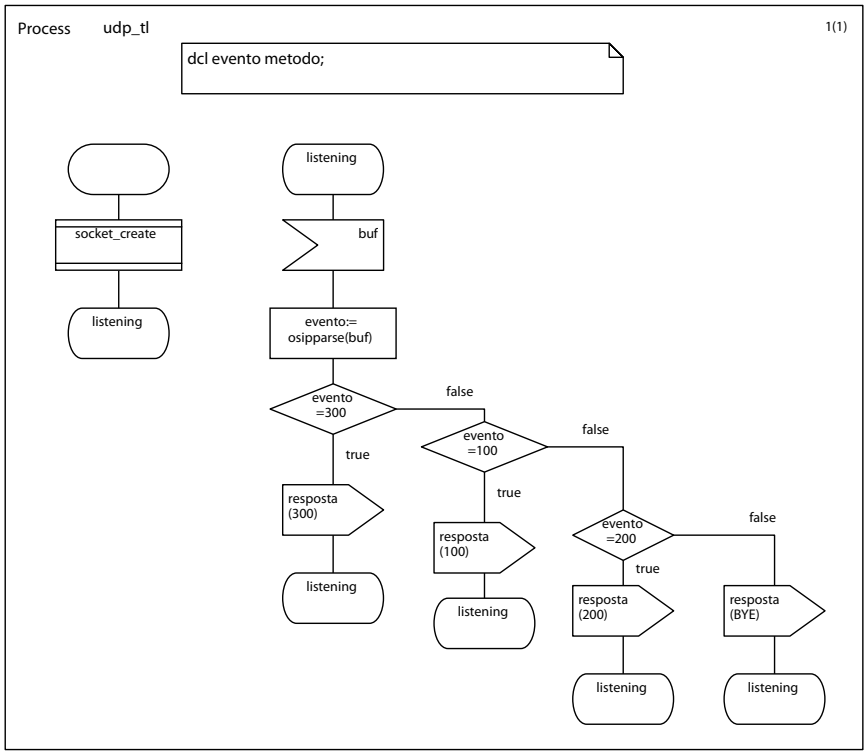

Figura 6.11: Diagrama do Processo udp_tl.

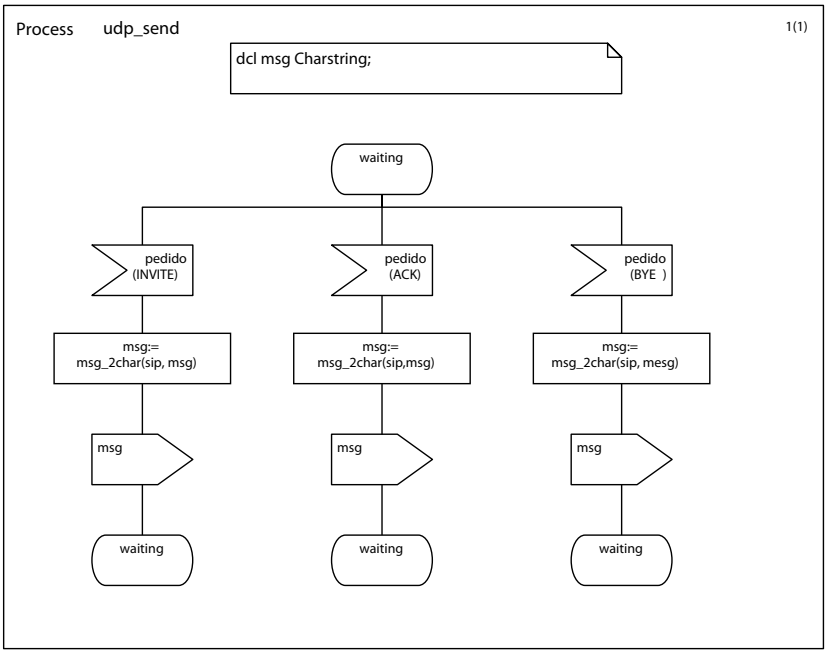

Figura 6.12: Diagrama do processo udp_send.

6.2.2 Sistema Atuando Como Servidor

Um sistema servidor é uma entidade lógica que gera uma resposta para um pedido SIP. Esta entidade lógica pode aceitar, rejeitar ou redirecionar o pedido. Esta característica acontece somente enquanto estiver ocorrendo a transação. Isto significa que, se o sistema receber um pedido, este atua como servidor enquanto durar essa transação. A Figura 6.13 mostra a máquina de estados do sistema realizando transação como servidor na geração de uma resposta SIP.

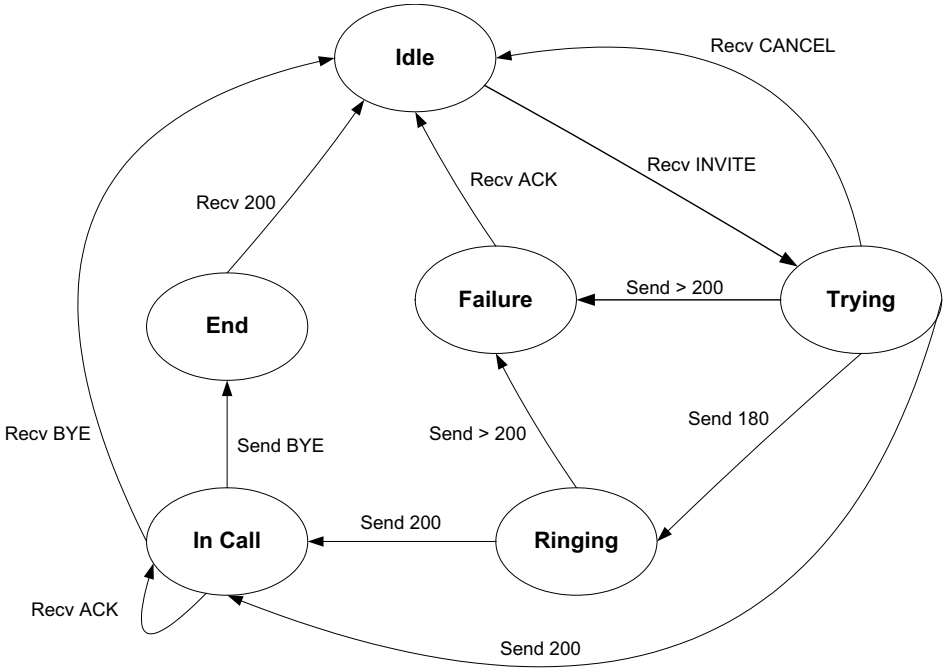

Figura 6.13: Máquina de estados finita para um sistema servidor.

Sistema Phone Servidor

O sistema **Phone_Servidor** consiste de dois blocos chamados **UAS** e **Transport_Layer**, como mostra a Figura 6.14. O bloco **UAS** é responsável pela comunicação entre o usuário e a camada de transporte, por onde serão recebidas e enviadas as mensagens SIP. Os sinais **Tentando**, **Conectado**, **Ausente**, **Encerrar** e **Sair** são enviados para o usuário pelo **UAS**, através do canal **InpUser** para mostrar o estado em que se encontra a sinalização. O sinal **Option** recebe do usuário, via canal **InpUser**, uma opção definida pelo tipo **escolha**, representada pelos caracteres (b=BYE), (q=Sair) e (s=Estado).

O bloco **Transport_Layer** recebe do **UAS** uma mensagem de resposta SIP, via canal **sndT** representada pelo sinal **resposta**, convertendo-a para uma mensagem padrão, representada pelo sinal **msg** e enviando-a pela rede, via o canal **sndNet**. O bloco **Transport_Layer** recebe uma mensagem **buf**

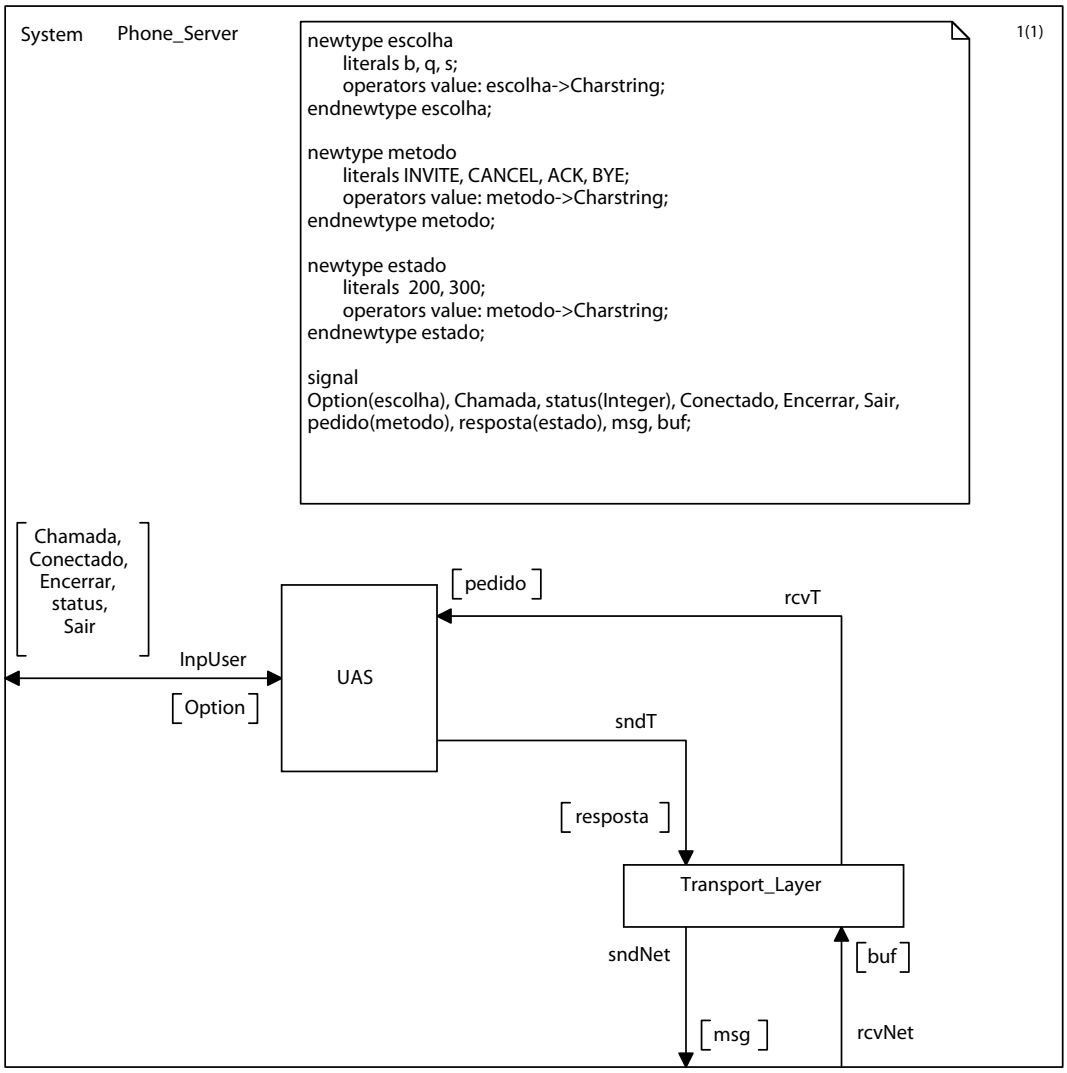

Figura 6.14: Diagrama do sistema Phone_Servidor.

padrão da rede, via canal **rcvNet**, convertendo-a para uma mensagem de pedido SIP e a passando-a para o bloco **UAS**. Para essas transações é usado o protocolo UDP.

São definidos tipos para as mensagens de reposta SIP, representados pelos sinais literais **100**, **200**, e **300**. Para as mensagens de pedido SIP, são definidos tipos representados pelos sinais **INVITE**, **CANCEL**, **ACK** e **BYE**.

Bloco *User Agent Server*

A descrição do bloco **UAS** é mostrado na Figura 6.15. Este contém dois processos, o processo **TU** e o processo **Server_Transaction**. Os processos se comunicam através da rotina **Sync**, e o processo

TU comunica com o canal **InpUser**, via rotina **InpEsc**. Já o processo **Server_Transaction** comunica com o canal **rcvT** e **sndT**, através das rotinas **rcvST** e **sndST** respectivamente.

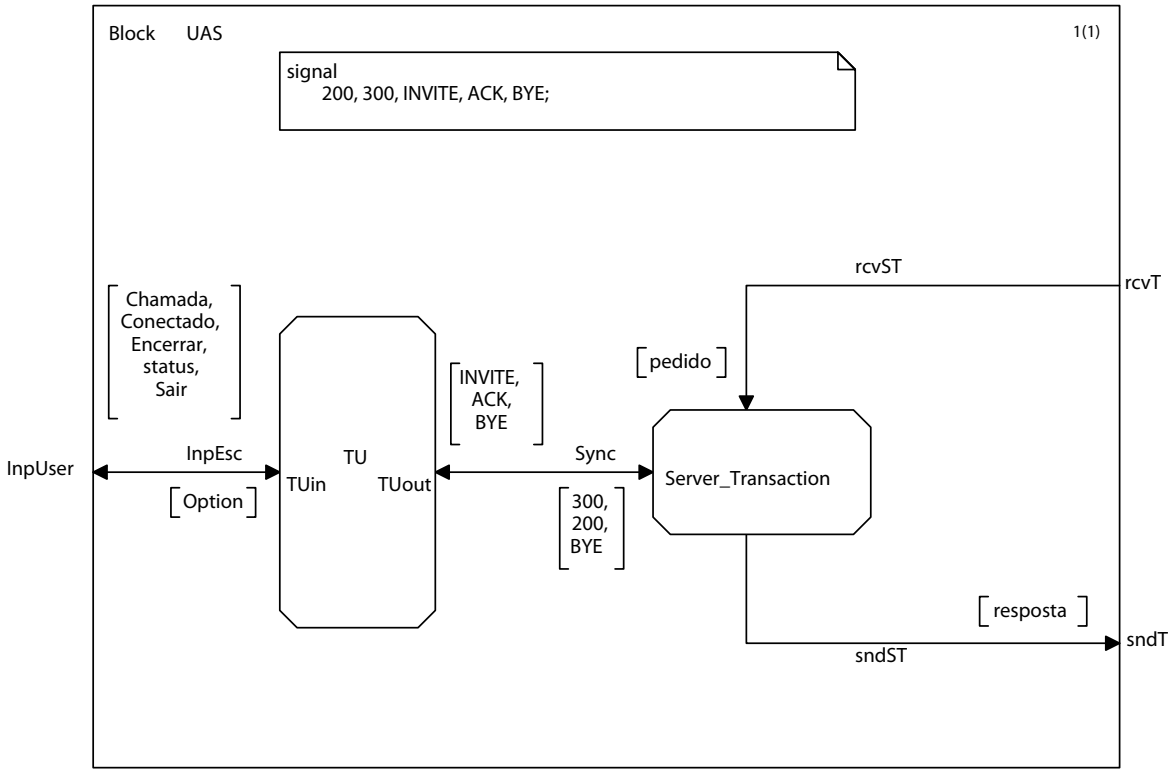

Figura 6.15: Diagrama do bloco UAS.

- **Processo TU**

O processo **TU** irá iniciar todas as funções do protocolo SIP, receber o endereço SIP do destinatário e entrar no estado **Idle**, como mostra a Figura 6.16.

O usuário pode sair do sistema digitando a opção "q", como pode mudar o estado atual digitando a opção "s". Se o usuário escolher a opção "s", este deve entrar com o valor 200, significando que está disponível para comunicação, ou com o valor 300, significando que está indisponível para comunicação.

Se o processo **TU** receber um sinal **INVITE** do processo **Server_Transaction**, este entra no estado **Trying**. Neste estado, a reposta para mensagem recebida é tratada verificando o conteúdo da variável **valor**. Se for igual a **300**, significa que o usuário está ausente, então o processo **TU** passa para o processo **Server_Transaction** o sinal **300** e entra no estado **Idle**. Se for igual a **200**, significa que usuário está disponível para comunicação, passando para o usuário o sinal **chamada** e para o processo **Server_Transaction** o sinal **200**, entrando em seguida no estado **inCall**. Quando no estado

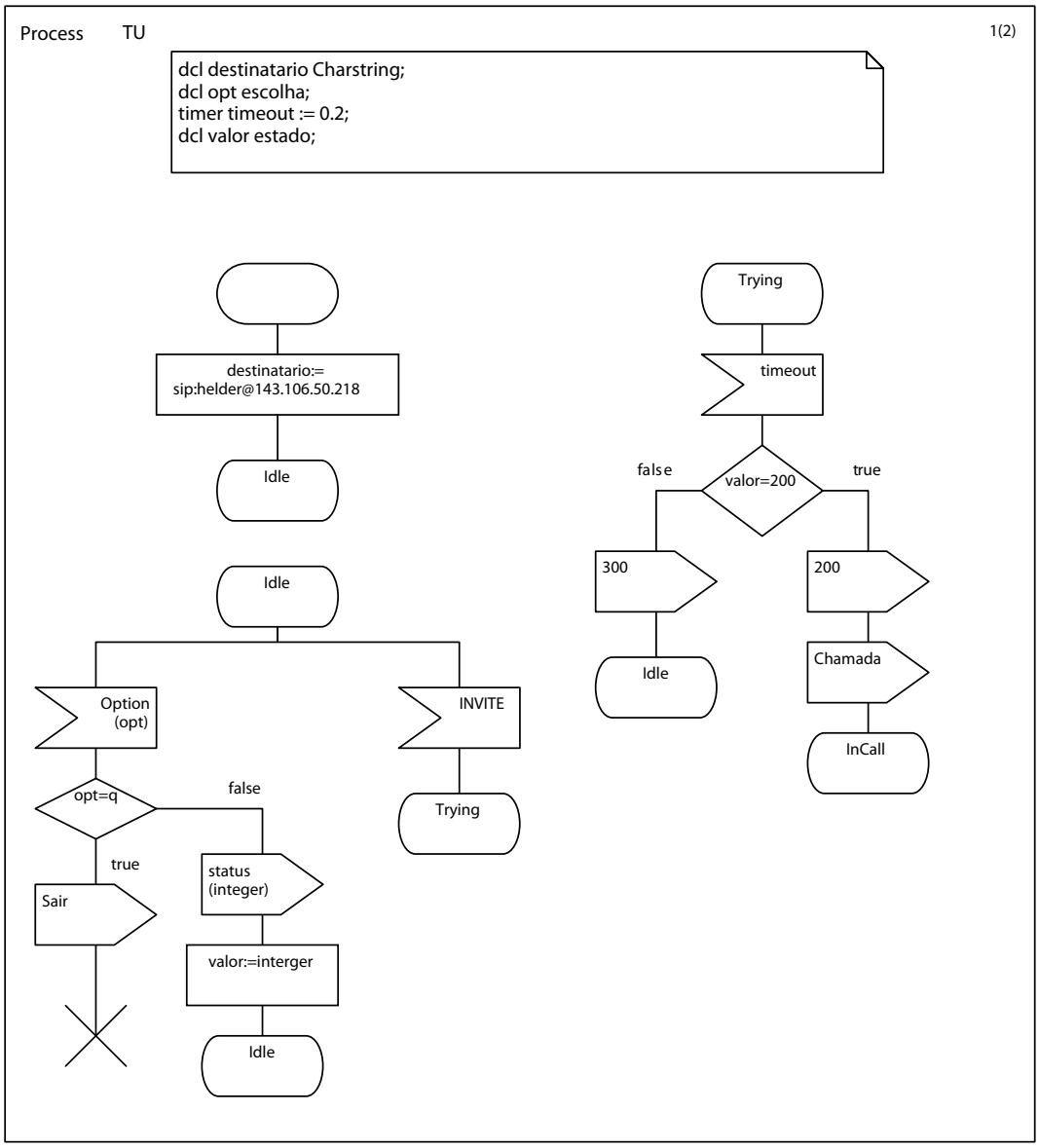

Figura 6.16: Descrição do processo TU.

inCall receber um sinal **ACK**, isto significa que a comunicação entre os dois usuários está aberta, permanecendo no estado **inCall**. Se receber um sinal **BYE**, significa que o outro lado da comunicação deseja encerrar a sessão. Ele passa um sinal **200** para **Server_Transaction** e um sinal **Encerrar** para o usuário, entrando no estado **Idle**. Se o usuário quiser encerrar a conexão, este entra com a opção (b=BYE), fazendo com que o processo **TU** passe um sinal **BYE** para o **Server_Transaction** e entre no estado **End**. Quando o processo **TU** recebe um sinal **200**, este passa para o usuário um sinal **Encerrar** e entra no estado **Idle**, como mostra na Figura 6.17.

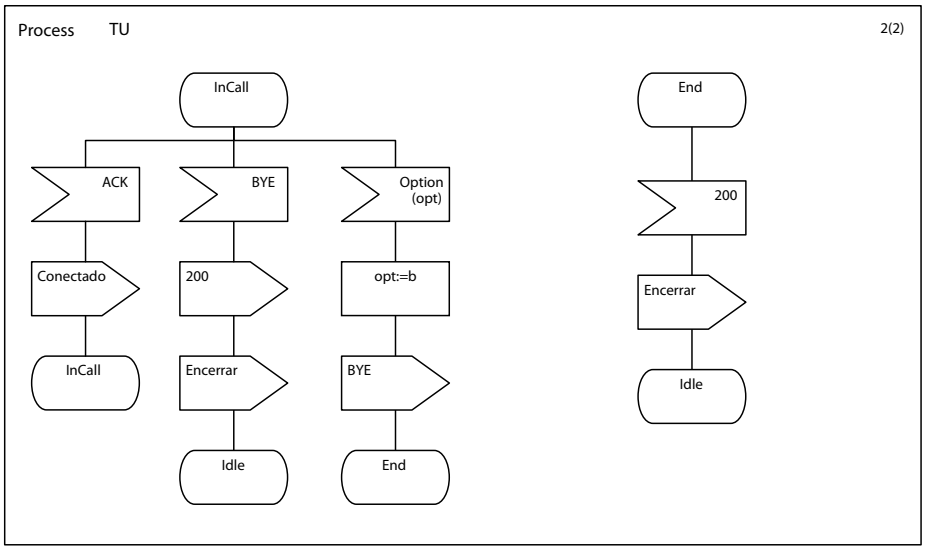

Figura 6.17: Descrição do processo TU.

- **Processo Server_Transaction**

O processo **Server_Transaction** mostrado nas Figuras 6.18 e 6.19 respectivamente é uma especificação e descrição da máquina de estados IST apresentada na secção 6.1.

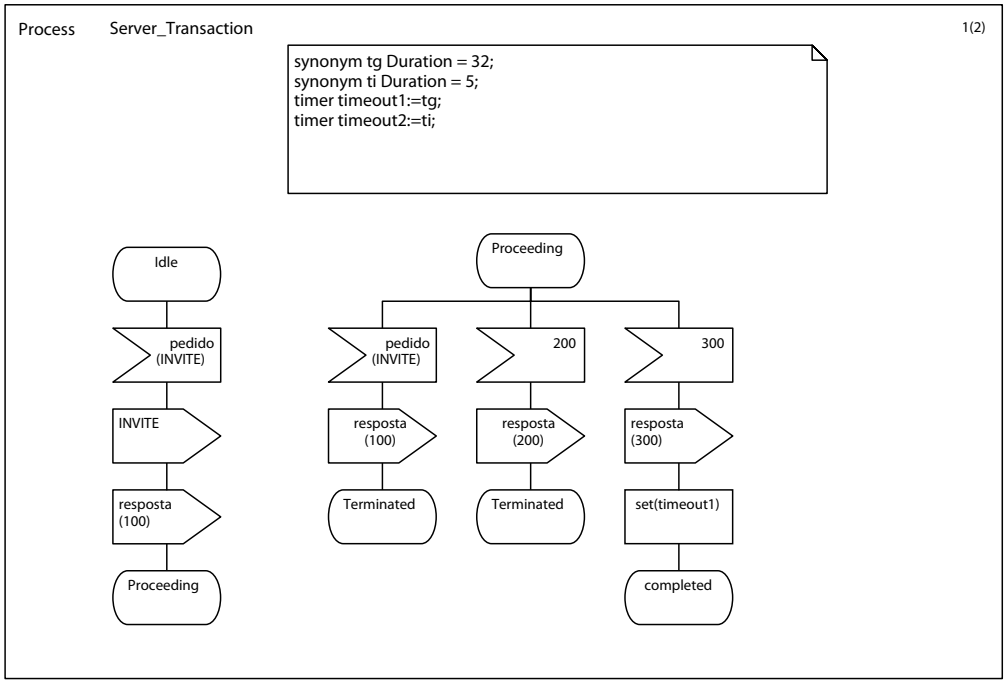

Figura 6.18: Diagrama do processo Server_Transaction.

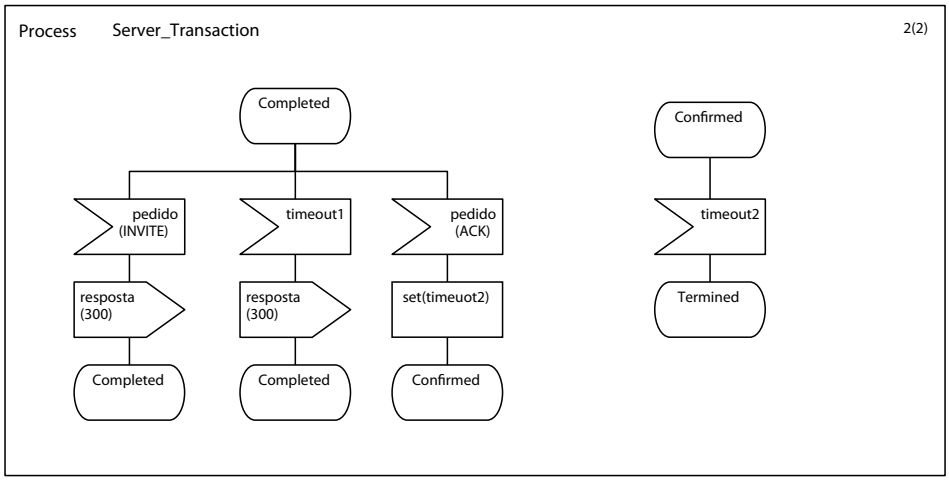

Figura 6.19: Diagrama do processo Server_Transaction.

Bloco *Transport Layer*

O diagrama mostrado na Figura 6.20, representa o processo **Transport_Layer** sendo formado pelo bloco **udp_tl**, que recebe uma mensagem de pedido do canal **rcvNet**, pela rotina **rcv_socket**, e envia para o canal **rcvT**, pela rotina **rcv**. O processo **udp_send** recebe uma mensagem resposta SIP do canal **sndT**, via rotina **snd** e entrega ao canal **sndNet**, pela rotina **snd_socket**.

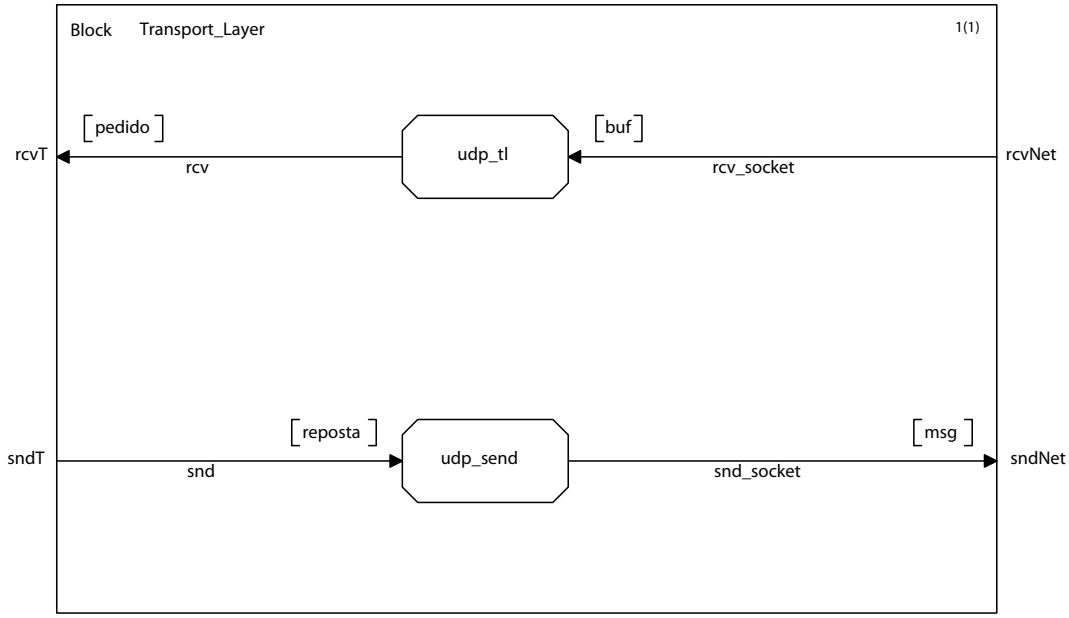

Figura 6.20: Diagrama do bloco Transport_Layer.

- **Processo udp_tl**

Quando o sistema é iniciado, o processo **udp_tl** cria um *thread* que abre um *socket* UDP que ficará sempre observando mensagens na porta 5060. Quando uma mensagem chega nesta porta, ela é armazenada em um *buffer* e convertida para o formato da mensagem SIP, sendo depois analisada pela API `osipparse` da biblioteca oSIP. Essa API descobre qual tipo de pedido representa essa mensagem, passando para o **Server_Transaction** esse pedido para ser tratada, como mostra a Figura 6.21.

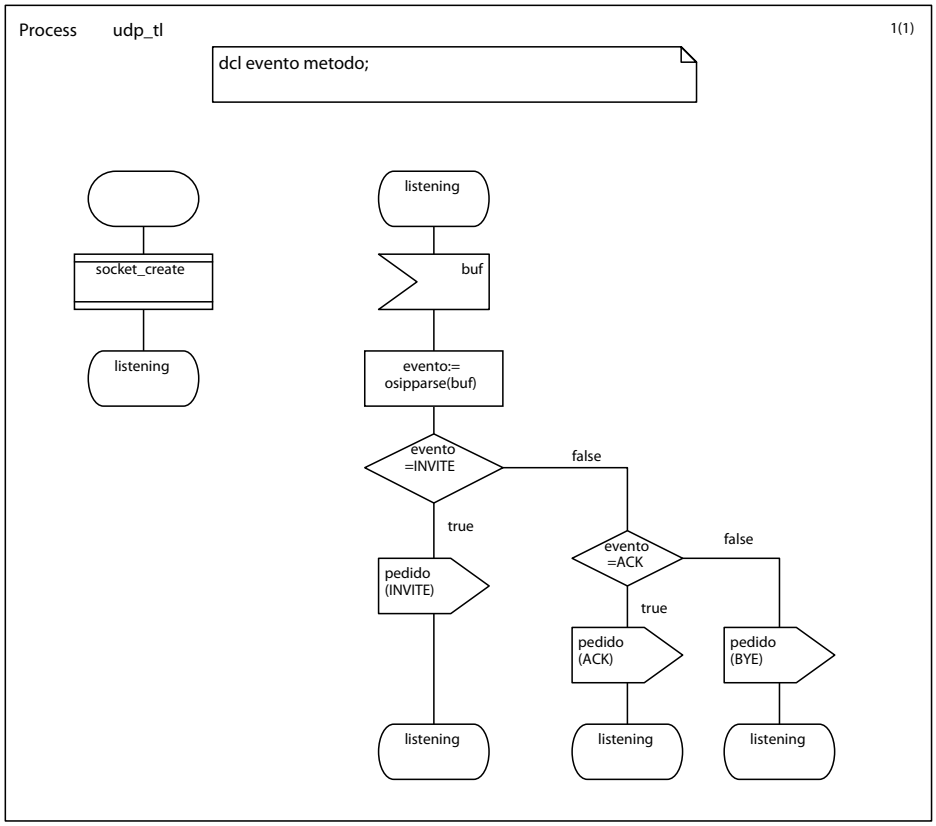

Figura 6.21: Diagrama do processo udp_tl.

- **Processo udp_send**

Quando o **Server_Transaction** quer enviar uma resposta SIP, ele passa a mensagem para o processo **udp_send**. Este processo converte a mensagem SIP para um formato *string*, através da API `msg_2char` da biblioteca oSIP. Após tratar essa mensagem, esta é enviada pela rede, como mostra a Figura 6.22.

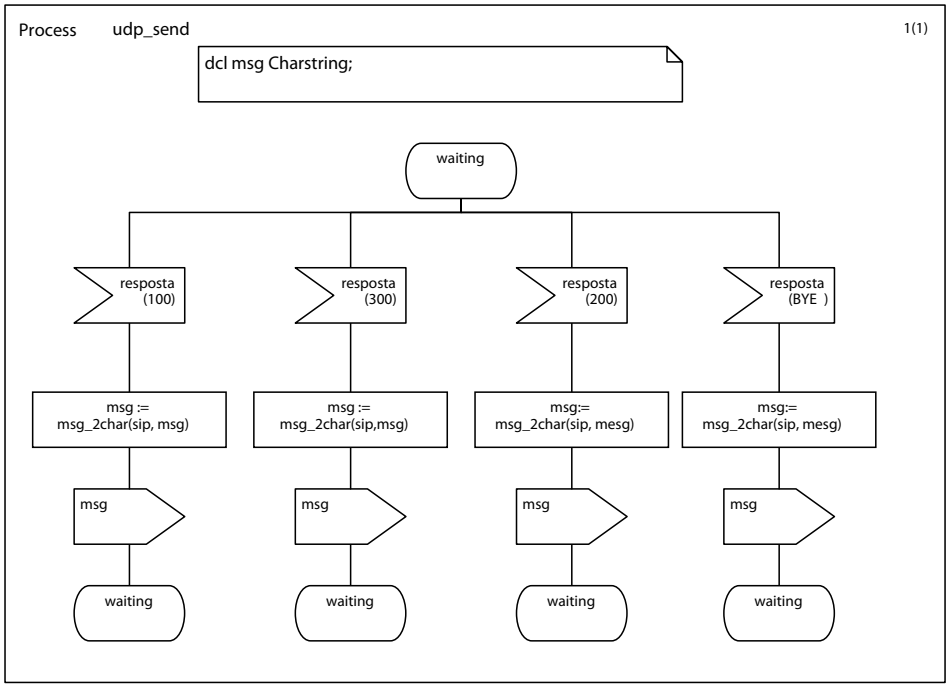

Figura 6.22: Diagrama do processo udp_send.

Capítulo 7

Resultados e Testes

Após a implementação do sistema RT-DSPhone do laboratório RT$\frac{M}{}$DSP, foram realizados testes de interoperabilidade. O primeiro teste é de serviços suportados pelo sistema desenvolvido, que é definido através de critérios de avaliação especificado pelo *Technical Program Committee* (TPC) do IETF. O segundo teste consiste em verificar a interoperabilidade do RT-DSPhone com outros softwares desenvolvidos por outros implementadores, que utilizam o protocolo SIP para realizar a sinalização de chamadas. Esses testes têm como intuito avaliar o nível de interoperabilidade do sistema desenvolvido, garantindo assim um certo nível de confiabilidade [Int, 2000].

7.1 Classificação do Nível de Interoperabilidade

O *Technical Program Committe* do IETF é responsável por organizar a parte técnica dos eventos de teste de interoperabilidade SIP. Estes eventos são encontros de grupos de desenvolvedores SIP que se reúnem de quatro em quatro meses, com o objetivo de realizarem a testes de interoperabilidade das suas implementações com as de outros grupos.

Este comitê definiu uma classificação de interoperabilidade das implementações que consiste em classificar as entidades SIP (UAs e *proxies*) em implementações básicas, intermediárias e avançadas de acordo com o cumprimento de uma lista de características necessárias para cada um dos níveis. Uma implementação satisfaz um determinado nível se cumprir o limiar mínimo de 80% dos critérios desse nível. Nas tabelas 7.1, 7.2, 7.3 [Schulzrinne, 2002] pode-se ver os critérios para os vários níveis e resultados da aplicação dos testes ao RT-DSPhone.

Essas tabelas descrevem a classificação de implementação para a entidade SIP *User Agent* (UA) desenvolvida neste trabalho. Este cenário é baseado no sistema fim-a-fim, testando o alto nível de funcionalidade entre as entidades SIP envolvidas. Essas tabelas tiveram suas últimas atualizações em Maio de 2005, baseadas na RFC3261.

Nível	Característica	Suporte
Básico	Envia INVITE sobre UDP	Sim
	Recebe INVITE sobre UDP	Sim
	Gera ACK apropriadamente	Sim
	Aceita e rejeita Chamada	Sim
	SDP com uma única linha m e c, e um único codificador	Sim
	Os cabeçalhos To, From, Call-ID, CSeq, Via, Content-Lenght, Content-Type tratados convenientimente	Sim
	Gera tags no campo To	Sim
	Termina chamadas com BYE sobre UDP	Sim
	Recebe BYE sobre UDP	Sim
	Suporta cabeçalho de forma compacta	Sim
	Rejeita pedidos desconhecidos com a resposta 501	Não (a)
	Envia/recebe streams, com a possibiklidade de não enviar pacotes RTCP	Sim

Tabela 7.1: Critérios de avaliação da interoperabilidade SIP - nível básico.

Nível	Característica	Suporte
http://www.uol.com.br/ Intermediário	Suporte dos cabeçalhos Require e *proxy*-Require	Sim
	Retransmite pacotes perdidos para INVITE e BYE	Sim
	Tem em consideração o cabeçalho Contact no INVITE e a resposta 2xx a um INVITE	Sim
	Processa o CANCEL para o INVITE	Sim
	Autenticação para o Registo utilizando o método Basic	(b)
	Permite o redireccionamento para páginas Web ou e-mail	Não
	Recebe texto ou HTML em respostas 3xx ou 4xx	Não
	Suporta o cabeçalho Accept sem SDP	Sim
	Suporta o registo com períodos de refrescamento para endereços unicast, tendo em consideração o cabeçalho	Não
	Expires da resposta REGISTER	Não
	Suporta o redirecionamento	Não
	Suporta múltiplos codificadores na linha 'm'	Sim
	Múltiplas linhas 'm' são processadas correctamente	Sim
	Tipos de meios desconhecidos e encontrados na linha são tratados convenientemente.	Não
	Tanto o nome do domínio como o endereço IP encontrados no campo 'c' são aceitos	Sim
	Gera pacotes RTCP	Não
	Responde ao pedido OPTIONS	Sim
	Suporta URLs não-SIP no Register	Não
	Copia um Record-Route da resposta para a Route	Não
	Consegue obter registos atuais	Não

Tabela 7.2: Critérios de avaliação da interoperabilidade SIP - nível intermediário.

Nível	Característica	Suporte
	Tenta automaticamente vários redireccionamentos	Não
	Gera pedidos REGISTER multicast	Não
	Suporta re-INVITE: suspende um stream	Sim
	Suporta re-INVITE: retoma um stream	Não
	Suporta re-INVITE: desactiva um único stream	Sim
	Suporta re-INVITE: altera codificadores	Sim
Avançado	Suporta re-INVITE: adiciona um stream	Sim
	Suporta re-INVITE: altera o endereço do meio para um endereço ou porta diferente (mobilidade)	Não
	Suporta o cabeçalho Expires do INVITE	Sim
	Registo de um terceiro elemento	Não
	Gera pedidos com URLs do tipo tel: e entrega-os a um servidor próprio	Não
	Processa múltiplas respostas MIME	Não

Tabela 7.3: Critérios de avaliação da interoperabilidade SIP - nível avançado.

Ao analisar a tabela 7.1 na coluna "Suporte", aparece a única resposta negativa desta tabela, representada pela opção (a) não sendo implementada por indisponibilidade de recursos. Já na tabela 7.2 na coluna "Suporte" o item assinalado com a opção (b), que se verifica pelo fato da RFC 3261 definir que o método de autenticação Básica foi abandonado, utilizando-se apenas o método *Digest* (que é um método de autenticação SIP).

Utilizando este critério de avaliação, verifica-se que o RT-DSPhone satisfaz em 92% os critérios básicos de interoperabilidade. Já no nível intermediário, obedece a 50% dos critérios, não atingindo o limiar de 80% necessário. O mesmo acontece para o nível avançado, que tem uma taxa um pouco menor que o nível intermediário, obedecendo a 42% dos critérios, também não satisfazendo o limiar de 80% para o nível avançado. O motivo do sistema não atingir o nível esperado nas Tabelas 7.2 e 7.3, vem do fato de se ter dado ênfase à sinalização ponto-a-ponto e não sendo testadas as características de sinalização como o *proxy*. Neste caso, as sinalizações com o *proxy* e o próprio *proxy* não foram implementados devido à sua abrangência.

7.2 Análise de Desempenho do Sistema RT-DSPhone

Nesta seção foi realizado uma análise de desempenho do sistema SIP implementado, para testar o sistema *User Agent* SIP desenvolvido. Para a realização desses testes, foi utilizada uma ferramenta conhecida por SIPp, que é um sistema de código aberto desenvolvido para teste de desempenho e geração de tráfego. O SIPp inclui alguns cenários básicos do *User Agent* do SipStone [Schulzrinne et al., 2002] (UAC e UAS), que estabelece e realiza múltiplas chamadas com os métodos INVITE e

BYE. O SIPp é caracterizado por uma janela dinâmica das estatísticas dos testes em andamento (taxa de chamada, *round trip delay* e estatísticas de mensagens), TCP e UDP sobre múltiplos *sockets* ou multiplexado com gerenciamento de retransmissão e ajuste dinâmico da taxa de chamadas. Quando otimizado para teste de tráfego, *stress* e desempenho, o SIPp pode ser usado para iniciar e encerrar uma simples chamada, provendo um resultado de passou/falhou.

O SIPp foi utilizado para testar o desempenho do sistema *User Agent* do RT-DSPhone. Para os testes com o *User Agent Server* do RT-DSPhone, foi utilizado uma taxa de transmissão de 2.365 cps (chamadas por segundo), um total de 135 chamadas INVITE geradas, sendo obtido um resultado de 135 sinalizações realizadas com sucesso, como mostrado nas Figuras 7.1, 7.2 e 7.3 respectivamente. Os resultados dos testes com o *User Agent Client* do RT-DSPhone é apresentado no Apêndice C.

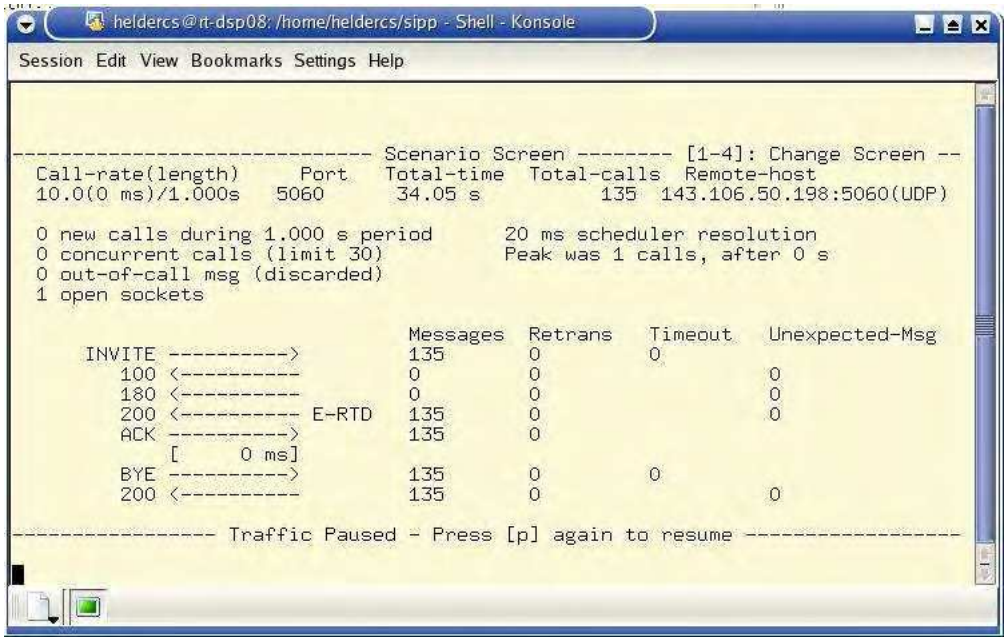

Figura 7.1: Análise de desempenho do RT-DSPhone atuando como Servidor.

7.3 Interoperabilidade com outras Aplicações

O segundo critério de teste consiste em avaliar ao nível de sinalização do RT-DSPhone com alguns softwares SIP, disponibilizados na Internet gratuitamente, ou em versão gratuita temporariamente. Entre esses softwares estão o Kphone que é um *User Agent* SIP para Linux, o *User Agent* Ubiquity que atua como “soft phone”, o Linphone que é um web fone e etc. Para ver mais detalhes desses testes consulte o apêndice D.

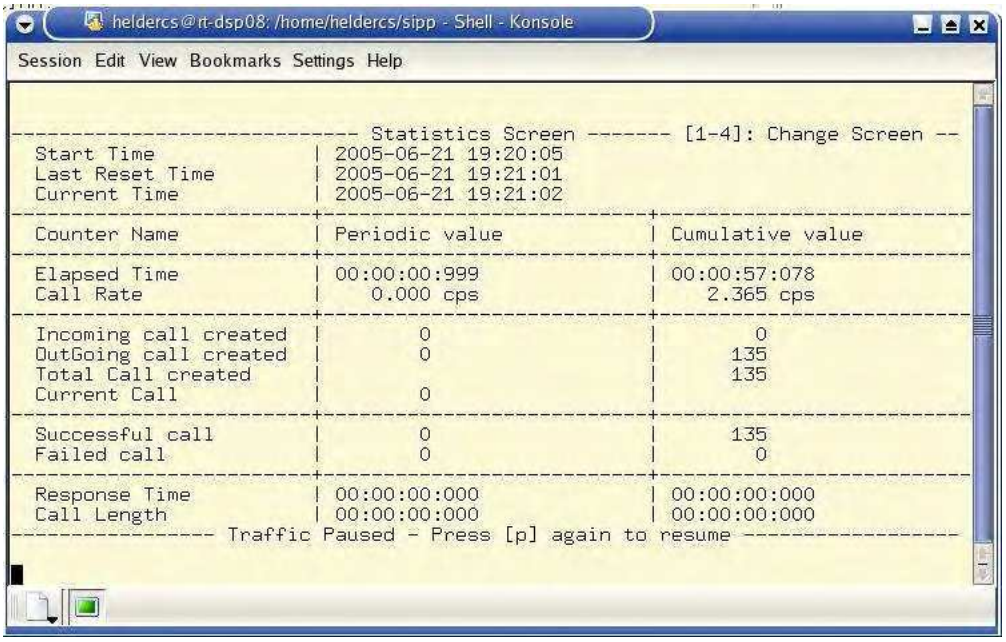

Figura 7.2: Análise de desempenho do RT-DSPhone atuando como servidor.

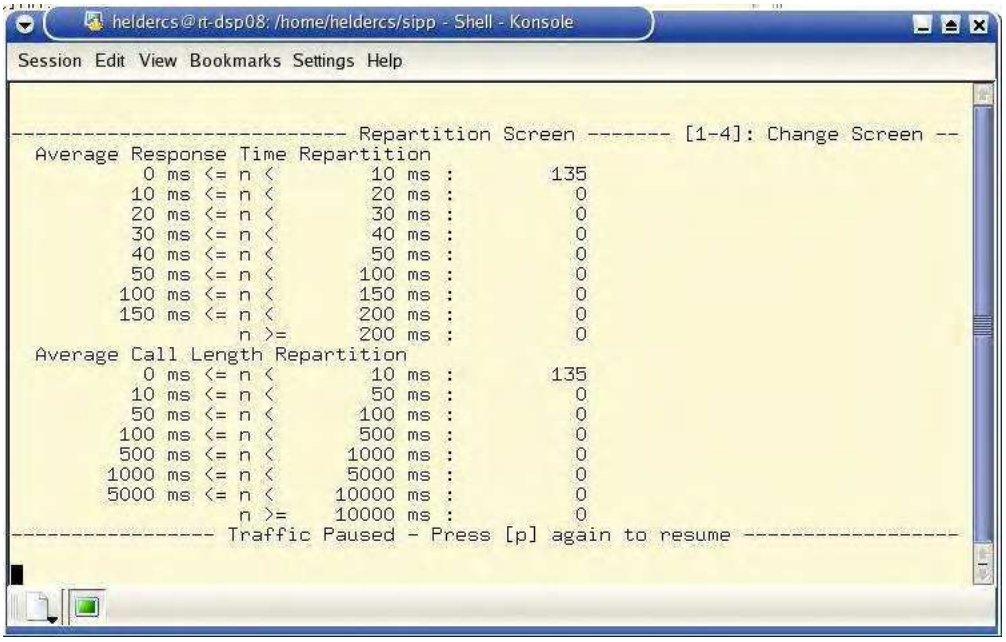

Figura 7.3: Análise de desempenho do RT-DSPhone atuando como servidor.

7.3.1 Características das Aplicações

Nesta seção são apresentadas algumas das características dos *User Agents* utilizados na realização dos testes. Este cenário envolve um computador executando um UA e um dispositivo embarcado

PowerQUICC II da Motorola executando um UA. Através do computador é feito um acesso remoto, via telnet, para a placa de desenvolvimento com o objetivo de mostrar os dois sistemas realizando a sinalização SIP. O ambiente para a realização dos testes é apresentado na Figura 7.4.

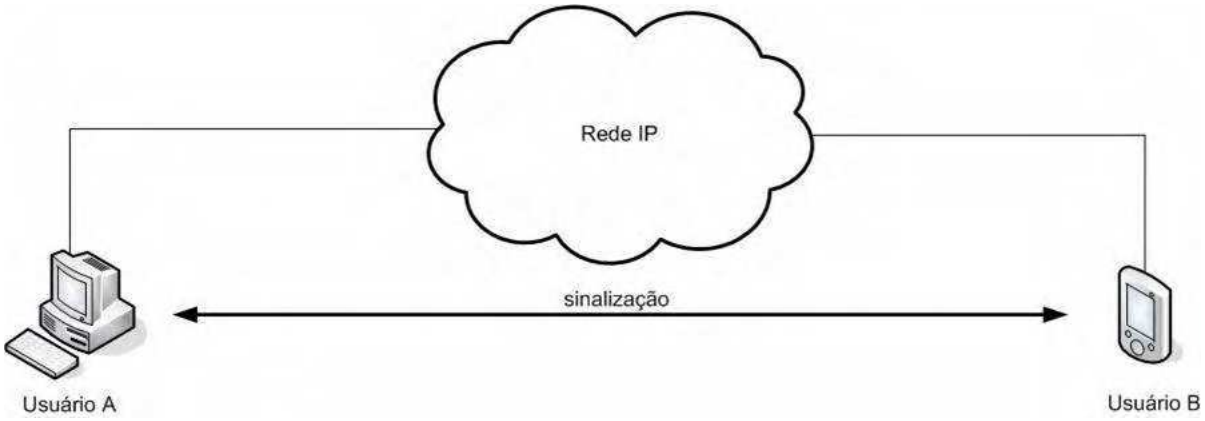

Figura 7.4: Cenário para realização dos testes.

A descrição da seqüência de sinalização realizada entre os sistemas, utilizando o protocolo UDP, é apresentada abaixo:

1. Usuário A chama usuário B:
   - Usuário A envia INVITE para usuário B.
   - Usuário B envia uma mensagem 100 TRYING e 180 RINGING (opcional).
   - Usuário B envia uma mensagem 200 OK para usuário A.
   - usuário A envia um ACK para usuário B.
2. Usuários terminando chamada:
   - usuário A envia um BYE para usuário B.
   - usuário B envia um BYE para usuário A.
3. Usuário B chama usuário A:
   - Usuário B envia INVITE para usuário A.
   - Usuário A envia uma mensagem 100 TRYING e 180 RINGING (opcional).
   - Usuário A envia uma mensagem 200 OK para usuário B.
   - usuário B envia um ACK para usuário A.
4. Usuários terminando chamada:

- usuário B envia um BYE para usuário A.
- usuário A envia um BYE para usuário B.

Nos testes realizados se obteve uma completa troca de sinalização SIP entre os softwares de outros fabricantes e o RT-DSPhone, mas não sendo realizado os testes de transferência de áudio em tempo real, devido a falta de recursos disponíveis na placa alvo para essa finalidade.

7.4 Resultados

O RT-DSPhone faz a implementação SIP básica, sendo capaz de realizar sinalizações ponto-a-ponto entre usuários finais. Para as configurações de áudio, necessita-se a inclusão de codeces, embora vários estejam disponíveis no acervo do laboratório RT$\frac{M}{}$DSP, não foi possível a realização destes, já que a placa não tem capacidade de PDS. Embora a placa possua conversores AD e DA, não foi implementado o codec simples G.711, devido essa característica fugir do escopo deste trabalho.

Com pequenas mudanças no makefile do ambiente de compilação dos fontes do sistema RT-DSPhone, pode-se gerar uma aplicação RT-DSPhone para uma variedade de plataformas, pois tanto a linguagem C padrão, quanto as APIs da biblioteca oSIP utilizadas no desenvolvimento desse sistema, suportam uma grande variedade de sistemas operacionais, tais como o Linux, MacOS, OpenBsd, Solaris, Windows e WinCE. Algumas medidas (conhecidas como medidas de *footprint*) foram realizadas durante a execução da aplicação VoIP no sistema embarcado PowerQUICC II. Essas medidas incluem o tamanho do μClinux e da aplicação, como o percentual de memória dinâmica ocupada pelo μClinux e pela aplicação, como mostrado na Tabela 7.4.

Tipo	**Tamanho (MB)**	**%Mem**
image.bin	2	0.18
image.elf*	6.2	0.1
linux.bin*	0.86	0.1
RT-DSPhone	0.02	0.08

Tabela 7.4: Medidas obtidas no dispositivo.

Capítulo 8

Conclusões

O grande crescimento do número de usuários da Internet, com o desenvolvimento contínuo da microeletrônica, nos faz pensar cada vez mais em utilizar a estrutura de rede da Internet para transmitir diversas fomar de comunicações, inclusive a voz. No entanto, utilizar a estrutura IP para transmitir voz, não é tarefa fácil. Afinal, são mídias sensíveis ao atraso e o protocolo IP não garante que os dados transmitidos cheguem ao seu destino na mesma ordem em que foram enviados e dentro de um intervalo regular de tempo, pois o protocolo não estabelece uma conexão dedicada para enviar os dados. Entretanto, a transmissão de voz sobre IP possibilita o barateamento das comunicações, pois pode evitar custos de ligações interurbanas e internacionais, além de gerar um maior potencial de comunicação entre os usuários da rede.

A quantidade de sistemas embarcados tem aumentado de forma exponencial, rumo a uma gigantesca integração entre produtos de diversos fabricantes, dispositivos, tecnologias e redes de comunicação. Em breve, poderemos assistir televisão em nossos celulares e teremos também em nossas residências telefones IP fazendo ligações interurbanas com o custo de uma ligação local, além de televisores que se conectam à Internet.

A recente atualização do protocolo SIP (RFC 3261), utilizado para implementar a sinalização do RT-DSPhone, veio esclarecer alguns detalhes em relação à versão inicial (RFC 2543) e melhorar o desempenho no que diz respeito à segurança. Segundo diversos especialistas, essas alterações e com a sua crescente divulgação, está levando este protocolo a ter uma maior aceitação por parte de um elevado número de empresas.

O SIP foi também escolhido pelo 3rd *Generation Partnership Project* (3GPP), para estabelecer sessões multimídia na rede UMTS (R5), e por grandes operadoras, como a WorldCom e AOL, tornando-se a principal alternativa à recomendação H.323.

Embora a qualidade de serviço seja um aspecto importante, e por isso foi revista no Capítulo 4 desta dissertação , resolveu-se em não implamentar soluções de qualidade de serviço no sistema

RT-DSPhone que garantam a qualidade de serviço associadas à transmissão de dados sobre redes IP. Convém entretanto mencionar que vários trabalhos de QoS foram desenvolvidos no laboratório RT$\frac{M}{}$DSP [Rachid and Meloni, 2004] [Marques and Meloni, 2004] [Barbosa and Meloni, 2002].

8.1 Trabalho Desenvolvido

O presente trabalho teve como objetivo o estudo e desenvolvimento de um sistema que permitisse a comunicação entre dois participantes, que poderiam estar tanto em um computador quanto em um dispositivo embarcado. Para o estabelecimento de uma sessão foi necessário o estudo intensivo dos protocolos de sinalizações mais utilizados na tecnologia VoIP, com destaque para o protocolo SIP. Foi feito um estudo dos dispositivos mais utilizados com processadores de comunicação, dando destaque para o dispositivo PowerQUICC II da Motorola. Deve-se destacar a grande dificuldade encontrada em gerar ferramentas para o desenvolvimento do sistema operacional μClinux e da aplicação VoIP para o sistema embarcado.

Foi feita também uma pesquisa de software auxiliar para o suporte da sinalização SIP, mais especificamente a biblioteca oSIP, que ajudasse a construir aplicações SIP e aplicações com suporte multimídia. Em seguida foram detalhados os requisitos de sinalização que o sistema RT-DSPhone deveria suportar.

Em relação ao processo de implementação do serviço anteriormente definido, deve-se destacar o estudo dos níveis de transação definido na RFC 3261. Entre os vários problemas inerentes ao funcionamento desta aplicação, salienta-se a geração de ferramentas para o desenvolvimento da aplicação no dispositivo embarcado PowerQUICC II, tais como a geração de *cross-compiler* para o mesmo e a geração do *kernel* do μClinux para a placa.

Finalmente, foram realizados testes de modo a se avaliar o nível de interoperabilidade através de critérios de avaliação definidos pelo *Technical Program Committee* da IETF e testes de interoperabilidade com diversos softwares comerciais disponíveis na Internet.

As principais contribuições feitas neste trabalho, além de uma ampla revisão bibliográfica, consiste em tecer conclusões sobre o estado atual desta tecnologia e a sua importância dentro do setor das telecomunicações, bem como o desenvolvimento de um sistema SIP capaz de comunicar com outras aplicações com a vantagem de poder ser executado em múltiplas plataformas.

8.2 Sugestões para Trabalhos Futuros

O trabalho efetuado possibilitou a criação de uma base do protocolo SIP para desenvolvimento de VoIP em diversas plataformas de *hardware*. Podemos observar uma série de trabalhos relacionados

ao tema que poderiam ser explorados:

- Utilização de ferramenta de apoio em sistema de educação a distância, fornecendo uma solução de baixo custo para os usuários de tais sistemas, a fim de garantir uma forma de comunicação com professores e monitores das disciplinas ministradas a distância.
- Existe também uma grande expectativa relativa a esta tecnologia, com o aparecimento da terceira geração de telecomunicações móveis (3G), prevendo nos próximos anos a oferta de serviços voz e vídeo em pacotes, e o aparecimento de um grande número de dispositivos ligados à rede IP.
- A interoperação SIP/H.323 constituindo um interessante ramo para estudos, bem com a tradução SIP/SS7.
- Desenvolvimento gráfico do sistema RT-DSPhone para um dispositivo embarcado, utilizando por exemplo, a ferramenta QTEmbedded da TrollTech.
- A integração da sinalização SIP com plataformas de *gateways* em linha de pesquisa VoIP do RT-DSP.

Referências Bibliográficas

Tribuzi André and Oliveira Morgado Cunha. Voz sobre ip. tutorial, CBA - Soluções em Informática, Dezembro 1999.

Lucas M. Barbosa and Luís G. Meloni. Algorítmos de busca em codificadores acelp. Tese de mestrado, Universidade Estadual de Campinas, 2002.

Michael Barr. *Programming embedded systems*. Oreilly, 1999.

S. Casner and V. Jacobson. Compressing ip/udp/rtp headers for low-speed serial links. Rfc 2508, IETF, Fevereiro 1999.

Overview of the session initiation protocol. CISCO, Janeiro 2000.

Chritopher Michael Collins. An evaluation of embedded system behavior using full-system software emulation. *Department of Eletrical Engineering and Computer Engineering*, Janeiro 2000.

J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL - formal object-oriented language for communication systems*. Teclogic, 1997.

R. Salami et al. Itu-t g.729 annex a: reduced complexity 8 kb/s cs-acelp codec for digital simultaneous voice and data. Technical report, IEEE Communication Mag., Setembro 1997.

MPC862 PowerQUICC family users manual. Freescale Semiconductor, Maio 2003.

Libosip reference manual. GNU, Maio 2002.

M. Handley and V. Jacobson. Sdp: session decription protocol. Request for comments 2327, Internet Engineering Task Force, Abril 1998.

W. Hyun, M. Huh, S. Kang, and D. Young Kim. An implemantetion of sip servers for internet telephony. *Tutorial*, Fevereiro 2002.

SIP SIG activity group SIP interoperability scenarios test plan, Junho 2000. International Multimedia Teleconferencing Consortium.

Drabik John. uclinux: World‘s most popular embedded linux distro? *LinuxDevices.com*, Novembro 2002.

D. Lin. Real-time voice transmmissions over the internet. M.sc. thesis, Dept. of Eletrical Engineering, Univ. of Illinois, Dezembro 1998.

P. Henrique Marques and Luís G. Meloni. Melhoria da qualidade de fala através da interpolação de parâmetros do codec gsm-amr em redes de pacotes. Tese de mestrado, Universidade Estadual de Campinas, Outubro 2004.

Arcturus Networks. uclinux white paper overview. Tutorial, Emdoor company, November 2002.

Kimmo Nikkannen. Uclinux as an embedded solution. Bachalor's thesis, turku Polytechnic, January 2003.

Packetizer. H.323 version 2 - overview. Página na internet, Packetizer, Outubro 2002a. http://www.packetizer.com/iptel/h323/whatsnew_v2.html .

Packetizer. H.323 version 3 - overview. Página na internet, Packetizer, Outubro 2002b. http://www.packetizer.com/iptel/h323/whatsnew_v3.html .

Packetizer. H.323 version 4 - overview. Página na internet, Packetizer, Outubro 2002c. http://www.packetizer.com/iptel/h323/whatsnew_v4.html .

Cinderella's Home Page. Cinderella sdl. Página na internet, Cinderella Software Company, 2002. http://www.cinderella.dk .

J. Postel. Simple mail transfer protocol. Request for comments 821, Internet Engineering Task Force, Agosto 1982.

Euler Rachid and Luís G. Meloni. Cancelamento de eco am telefonia ip. *Unicamp*, Outubro 2004.

Radvision. Sip: Protocol overview. Technical report, Radvision Company, 2001.

Radvision. Understanding sip severs. Technical report, Radvision Company, 2002.

J. Rosemberg and H. Schulzrinne. Sip: session initiation protocol. Request for comments 2543, Internet Engineering Task Force, Outubro 2002.

J. Rosemberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: session initiation protocol. Request for comments 3261, Internet Engineering Task Force, Outubro 1999.

G. Schröder and M. H. Sherif. The road to g.729: Itu 8-kb/s speech coding algorithm with wireline quality. *IEEE Communication Mag.*, setembro 1997.

H. Schulzrinne. Rtp profile for audio and video conferences with minimal control. Request for comments 1890, Internet Engineering Task Force, Janeiro 1996.

H. Schulzrinne. Classification for sip interoperability test event. Página na internet, University, Agosto 2002. http://www.cs.columbia.edu/sip/sipit/classification.html .

H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: a transport protocol for real-time applications. Request for comments 1889, Internet Engineering Task Force, Janeiro 1996.

H. Schulzrinne, S. Narayanam, and J. Lennox. Sipstone benchmarking sip server performance. *Columbia University*, Abril 2002.

H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (rstp). Request for comments 2326, Internet Engineering Task Force, Abril 1998.

H. Schulzrinne and J. Rosenberg. Internet telephony. Technical report, Internet Engineering Task Force, Junho 2000.

International telecommunication union. Isdn user-network interface layer 3 specification for basic call control. Recomendação q.931, ITU-T, 1993.

International telecommunication union. Pulso code modulation (pcm) of voice frequencies. Recomendação g.711, ITU-T, 1998.

International telecommunication union. Control protocol for mulltimedia communication. Recomendação h.245, ITU-T, Março 2000.

International telecommunication union. Call signalling protocols and media stream packetization for packet-based multimedia communication systems. Recomendação h.225.0, ITU-T, Novembro 2001a.

International telecommunication union. Packet-based mulltimedia communication systems. Recomendação h.323, ITU-T, Novembro 2001b.

uClinux's Home Page. uclib - a c lbrary for embedded system. Página na internet, Networks Arcturus, Novembro 2002a. http://www.uclib.org/ .

uClinux's Home Page. uclinux - uclinux operating system. Página na internet, Networks Arcturus, 2002b. http://www.uclinux.org/ .

F. Yergeau. Utf-8, a transformation format of iso 10646. Request for comments 2279, Internet Engineering Task Force, Janeiro 1998.

Apêndice A

Sistema Embutido PowerQUICC II

Um sistema embarcado é um sistema computacional embutido em um sistema maior e programado para realizar uma tarefa específica. Podemos encontrar um sistema embarcado em relógios, celulares, equipamentos de rede como roteadores e *gateways*, calculadoras, televisores, DVDs, vídeo game, etc.

A quantidade de dispositivos embarcados disponíveis para o desenvolvimento de sistema de telecomunicações é enorme, dentre elas temos:

- Dragonball da Motorola;
- PowerQUICC da Motorola;
- WinPath da wintegra;
- Inteli960;
- entre outros.

O PowerQUICC II é um versátil processador desenvolvido especificamente para área de comunicações, onde o PowerQUICC II integra um chip de alta performance, baseado na arquitetura PowerPC e derivado da família Motorola MPC860 *QuadIntegrated communications Contrroller* (PowerQUICC) [Fre, 2003]. O PowerQUICC é uma unidade de integração de sistema muito flexível e com muitos controladores de comunicações que podem ser utilizados por aplicações em geral, particularmente em telecomunicações e sistemas de rede.

Para o desenvolvimento deste trabalho foi usada uma placa de desenvolvimento da Motorola, com funcionalidades específicas para a área de telecomunicações. Na Figura A.1 é apresentado o diagrama de bloco da placa de desenvolvimento.

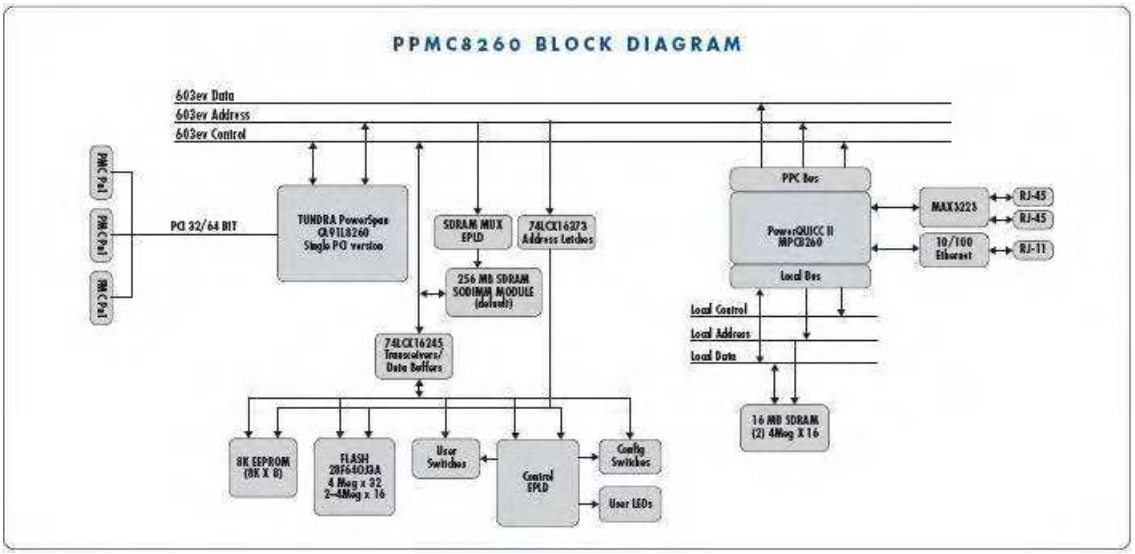

Figura A.1: Diagrama de blocos da placa de desenvolvimento.

Apêndice B

Diagramas de Estados

Diagrama de estado para uma transação cliente diferente de um INVITE:

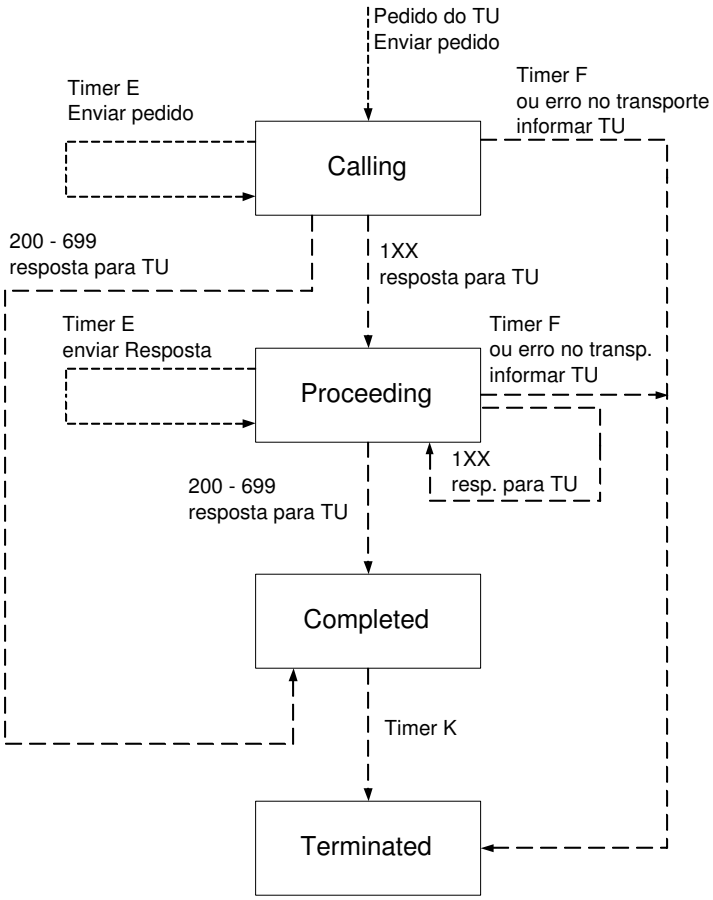

Figura B.1: Non-INVITE Client Transaction.

Diagrama de estado para uma transação Servidor diferente de um INVITE:

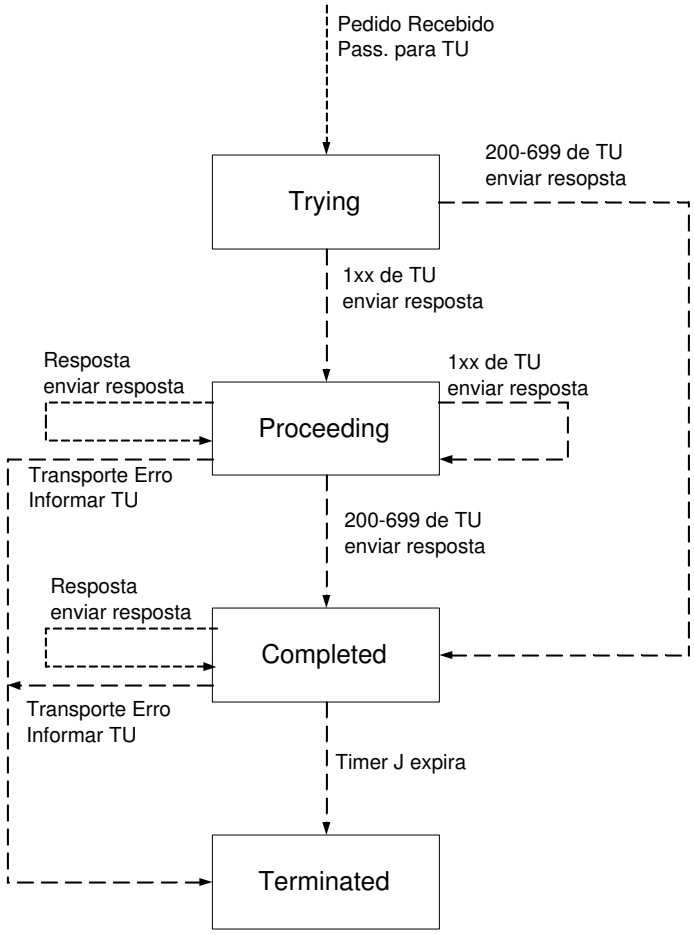

Figura B.2: Non-INVITE Server Transaction.

Apêndice C

Análise de Desempenho do Sistema RT-DSPhone Cliente

Foi obtido um resultado de 100 sinalizações realizadas com sucesso, como mostrados nas Figuras C.1, C.2 e C.3 respectivamente.

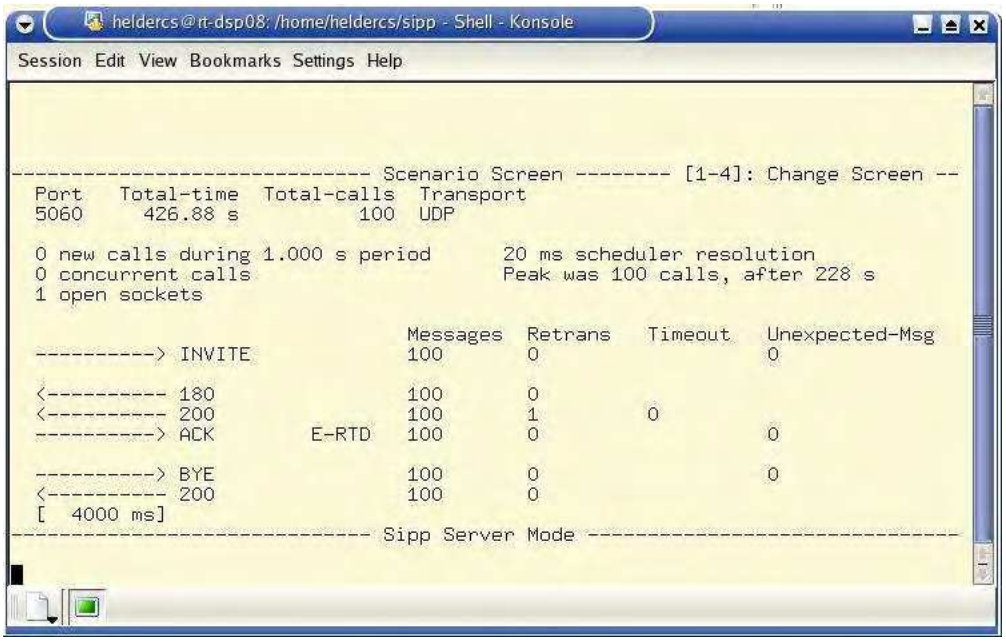

Figura C.1: Cenário da análise de desempenho do RT-DSPhone Cliente.

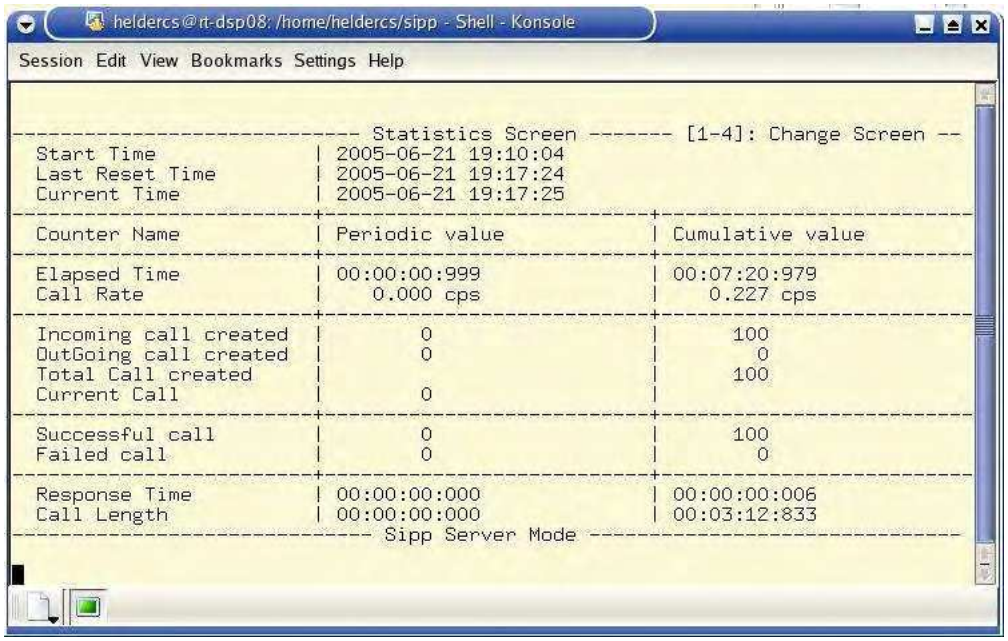

Figura C.2: Cenário da análise de desempenho do RT-DSPhone Cliente.

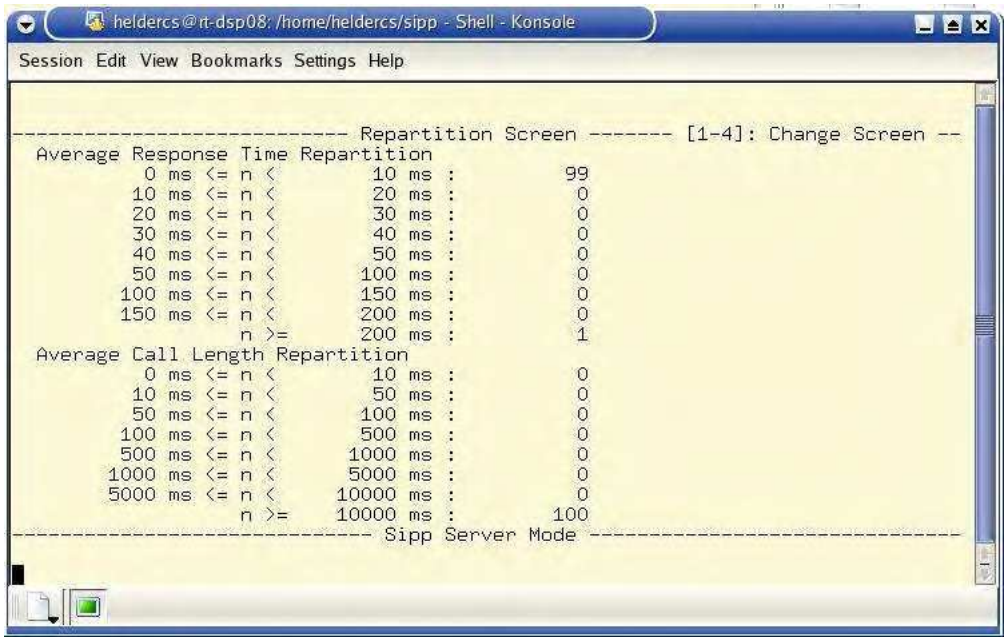

Figura C.3: Cenário da análise de desempenho do RT-DSPhone Servidor.

Apêndice D

Interoperabilidade com outros Softwares

D.0.1 Sistema Kphone

O Kphone é um *User Agent* SIP para Linux, no qual se pode iniciar uma conexão VoIP sobre Internet. Ele suporta chamadas de áudio, mensagens instantâneas, e vídeos entre dois computadores. A Figura D.1 ilustra o estabelecimento de uma chamada entre o RT-DSPhone e o Kphone.

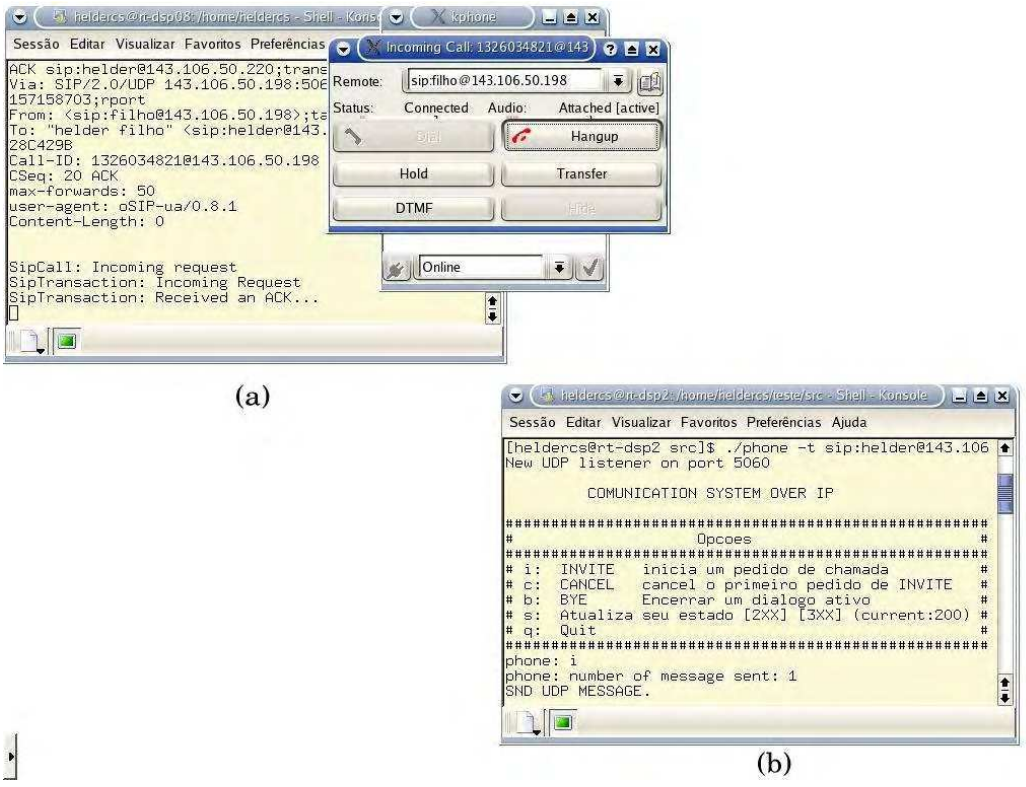

Figura D.1: Estabelecimento de uma Sessão entre a)Kphone e b)RT-DSPhone.

D.0.2 Sistema Ubiquity

As propriedades da Ubiquity Software Corporation, definidos segundo a RFC 2543, exemplificam uma implementação baseada na JAIN SIP lite para Windows. O *User Agent* Ubiquity atua como um "soft fone", permitindo realizar chamadas de PC-para-fone usando a Internet e sendo utilizado para os desenvolvedores testarem seus produtos. A Figura D.2 ilustra o estabelecimento de uma sessão entre o RT-DSPhone e o Ubiquity.

Figura D.2: Estabelecimento de uma Sessão entre a)Ubiquity e b)RT-DSPhone.

D.0.3 Sistema Linphone

O Linphone é um phone web, que foi desenvolvido por um grupo de desenvolvedores franceses e tem as seguintes funcionalidades:

- Trabalha com o Gnome Desktop dentro do Linux.
- Inclui uma larga variedade de codecs (G711-lei-u, G711-lei-a, LPC10-15, GSM, and SPEEX).
- É um software livre, disponível por GPL (*General Public Licence*).

O Linphone é compatível com alguns sistemas SIP desenvolvidos, tais como:

- O softphone eStara (software comercial para Windows).
- O Pingtel phone.
- O Hotsip, um software gratuito para windows.

- O Vocal, uma pilha SIP código aberta, da Vovida.

A Figura D.3 ilustra o estabelecimento de uma chamada entre o RT-DSPhone e o Linphone.

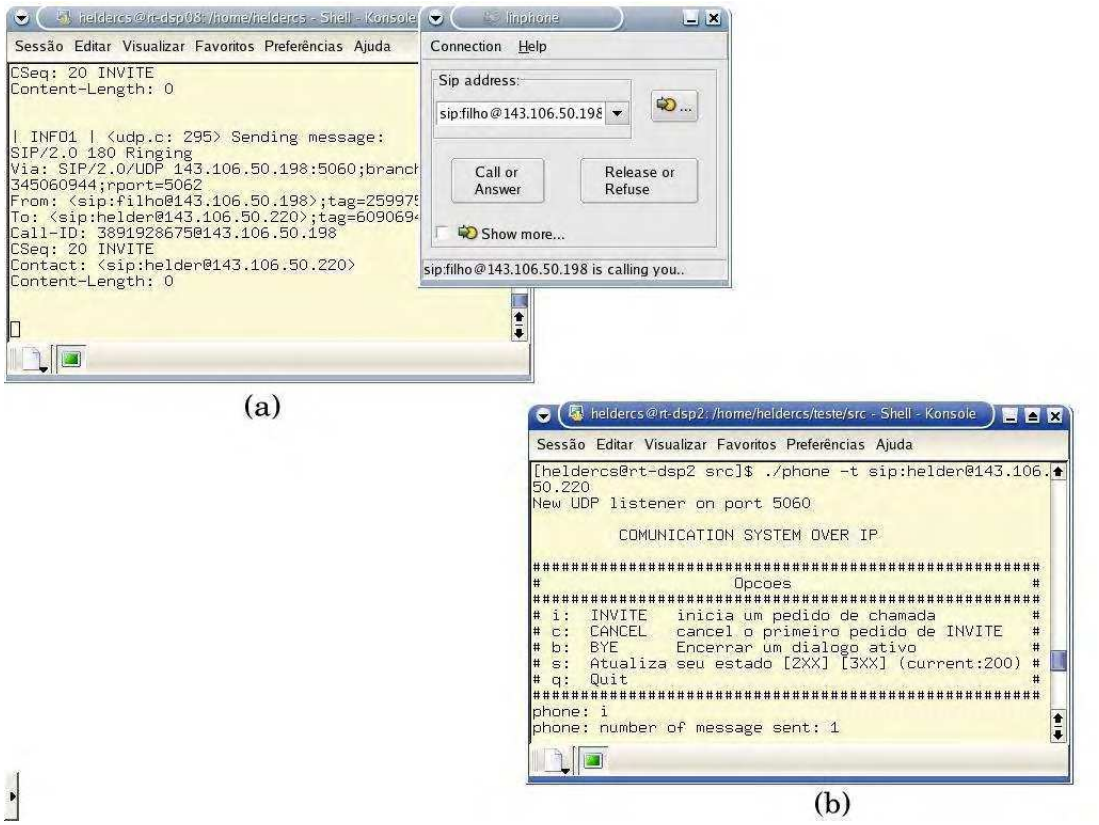

Figura D.3: Estabelecimento de uma Sessão entre a)Linphone e b)RT-DSPhone