Universidade Federal de Campina Grande
Departamento de Engenharia Elétrica
Estágio Supervisionado

Implementação de um *User Agent* Residente em um Dispositivo Móvel Embarcado para Redes de Telefonia IP Utilizando o Padrão SIP

Daniel da Costa Uchôa
daniel@dee.ufcg.edu.br

Disciplina:
Estágio Supervisionado

Orientador:
Angelo Perkusich

Campina Grande, Junho de 2005

Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

Conteúdo

Lista de Figuras

Apresentação

Este é um relatório referente às atividades de Estágio Supervisionado desenvolvidas pelo aluno Daniel da Costa Uchôa, graduando em Engenharia Elétrica pela Universidade Federal de Campina Grande. As atividades foram realizadas no período de fevereiro de 2005 à junho de 2005, no Laboratório de Sistemas Embarcados.

Resumo

O protocolo de inicialização de sessão SIP vem sendo cada vez mais utilizado, se tornando um protocolo de sinalização padrão para estabelecimento de comunicação fim-a-fim entre diferentes dispositivos usando diferentes plataformas. Entre tais dispositivos podemos citar PCs, telefones IP, aparelhos celulares, *handhelds* e até TVs digitais. Assim, desenvolvedores de aplicativos deverão estar familiarizados com as especificações deste protocolo, bem como com suas implementações e ferramentas de desenvolvimento, visto que uma comunicação fim-a-fim será um requisito incontestável em novas aplicações. Portanto, este trabalho analisa uma biblioteca de desenvolvimento de terminais SIP para a plataforma Linux chamada de *eXosip2*, que oferece uma API de programação simples, prometendo um desenvolvimento fácil e rápido de aplicativos terminais SIP. No fim, utilizaremos como hospedeiro para o terminal SIP um *handheld*, e o integraremos às outras entidades SIP que constituem uma rede local móvel de voz sobre IP, a qual denominamos de MVoIP.

Capítulo 1

Introdução

A cada ano são introduzidos um grande número de novos dispositivos de comunicação e aplicações em todo o mundo, o que está criado uma miscelânea de aparelhos e meios de comunicação. Ao mesmo tempo, diferentes dispositivo estão convergindo para uma mesma plataforma e oferecendo os mesmos serviços, como o caso dos *smart phones* e aparelhos *handhelds*, que tendem a se tornar um só dispositivo.

Neste contexto de convergência, serviços e aplicações são redefinidos. Aplicações não são mais entidades isoladas trocando informações apenas com a interface do usuário. A próxima geração de aplicações envolverá conectividade fim-a-fim entre terminais IP. Esta capacidade de estabelecimento de conexões fim-a-fim é o ingrediente chave para permitir uma comunicação mais rica.

Surge então uma importante questão: como será possível garantir interoperabilidade entre diferentes dispositivos e aplicações? Uma solução é utilizar o menor número de protocolos de comunicação possível. Desde que estes protocolos deverão ser utilizados para diferentes plataformas, eles precisam ser escaláveis.

O protocolo de inicialização de sessão, o SIP[1], apesar de jovem, está chamando atenção de um grande número de companhias desenvolvedoras neste sentido. O SIP está surgindo como protocolo padrão de comunicação fim-a-fim entre diferentes dispositivos utilizando diversas plataformas. A força do SIP pode ser resumida em três características básicas:

[1]Session Initiation Protocol - definido na RFC 3261 [2].

escalabilidade, extensibilidade e flexibilidade.

O SIP é um protocolo usado para criar, modificar e terminar sessões com um ou mais participantes, além de negociar os codificadores/decodificadores (*codecs*), localização de usuários, redirecionamento de mensagens, entre outras funções [2].

Basicamente, os elementos que constituem o mundo SIP são os *user agents*, os *proxies*, os *registrars*, os servidores de redirecionamento (*redirect servers*), e os *gateways* de sinalização para interoperação com outros protocolos de sinalização.

Neste trabalho, vamos nos concentrar no comportamento dos *user agents*. Os *user agents* (UA) são terminais da Internet que utilizam o SIP para localizar um ao outro e para negociar características de sessão [7]. Usualmente, são aplicativos residentes nos dispositivos do usuário. Estes dispositivos podem ser computadores, aparelhos celulares, *gateways* PSTN[2], *handhelds* (PDAs), e etc.

Utilizaremos como *host* (hospedeiro) para o *user agent* um dispositivo móvel embarcado. Poderemos optar por um *handheld* com o sistema operacional Linux embarcado [9] ou um aparelho celular com o sistema operacional Symbian [8]. Ambos os sistemas operacionais oferecem uma interface de programação (API[3]) que possibilita o desenvolvimento de novas aplicações.

O propósito deste trabalho, portanto, é realizar um levantamento sobre soluções SIP disponíveis para *user agents*, proprietários ou livres, e de ferramentas de desenvolvimento de tais aplicações. Testaremos algumas das implementações encontradas dentro de uma rede local de voz sobre IP que será implementada no Laboratório de Sistemas Embutidos em conjunto com mais outros três alunos: Danilo Freire, José Luís e Olympio Cipriano.

Com a implementação desta rede, será possível o estabelecimento de uma comunicação multimídia fim-a-fim entre um dispositivo móvel embarcado e qualquer outro dispositivo terminal dentro de uma rede de telefonia IP. Os usuários de tais dispositivos terão a possibilidade de serviços como "pressione para falar" (*push-to-talk*), onde o usuário pressiona uma tecla para convidar uma pessoa ou um grupo de usuários de uma lista previamente configurada para abertura de sessão. Os usuários serão tipicamente conectados instantaneamente,

[2] *Public Switched Telephone Network*

[3] *Application Programming Interface*

podendo então conversarem entre si, trocarem mensagens instantâneas, ou compartilharem dados multimídia, como áudio e vídeo em tempo real, realizar videoconferências, etc.

Além de novas possibilidades de comunicação multimídia, um conceito importante será introduzido: a informação sobre a *presença* do indivíduo. Com isso, os usuários poderão saber quando e como contatar seus amigos, mesmo antes de efetuar uma chamada.

Através da presença, os usuários destes dispositivos móveis terão acesso a informações sobre as pessoas que fazem parte do seu ciclo de amizade, incluindo o *status* de conexão/chamada, identidade do usuário, a capacidade dos terminais utilizados, a localização da pessoa e sua disponibilidade para atender ou não chamadas. Assim, os usuários móveis terão a opção de mostrar seu *status* (ocupado, fora da sala de trabalho, fora do escritório, de férias ou de folga, etc.) para usuários escolhidos, sugerindo a forma mais apropriada de meio de comunicação a ser utilizada (*chat*, mensagem instantânea, *e-mail*, etc.).

Capítulo 2

O Protocolo SIP

2.1 O Propósito do SIP

Há muitas aplicações na Internet que necessitam da criação e gerenciamento de uma *sessão*, onde uma sessão é considerada uma troca de dados entre uma associação de participantes [2]. A implementação de tais aplicações se torna complicada devido ao comportamento destes participantes: usuários podem se movimentar entre terminais, podem ser endereçados por múltiplos nomes, e eles pode se comunicar através da utilização de diferentes meios, algumas vezes simultaneamente.

Diversos protocolos para transporte de várias formas de dados de sessão multimídia em tempo real, tal como voz, vídeo, ou mensagens de texto, foram criados. O protocolo de inicialização de sessão SIP[1] trabalha em conjunto com estes protocolos, possibilitando que terminais de Internet se descubram e concordem com uma caracterização de sessão que eles desejem compartilhar.

O SIP é uma ferramenta de propósito geral ágil, capaz de estabelecer, modificar, e finalizar sessões multimídias (conferências). O SIP trabalha independentemente dos protocolos de transporte usados na camada mais a baixo, e sem depender do tipo de sessão que

[1] *Session Initiation Protocol*

está sendo estabelecido.

O SIP é um protocolo de controle de camada de aplicação projetado para fácil implementação, com boa escalabilidade e flexibilidade. Como dito na Seção 2, o SIP foi desenvolvido pela IETF, e sua especificação é disponível sob forma de diversas RFCs[2], sendo a RFC 3261 [2] a mais importante, pois contém a especificação do núcleo do protocolo.

O propósito deste protocolo é apenas fazer com que a comunicação seja possível, mas a comunicação propriamente dita precisa ser alcançada por outros meios. Isto quer dizer que o SIP não é um sistema de comunicação verticalmente integrado, ao invés disto, ele é um componente que pode ser usado com outros protocolos IETF para construir uma arquitetura multimídia completa.

Tipicamente, tais arquiteturas irão incluir protocolos como o RTP[3] [3] para transporte de dados em tempo real e provisionamento de uma realimentação sobre qualidade de serviço, QoS[4], o RTSP[5] [4] para controle de entrega do fluxo (*streaming*) de mídia, o MEGACO[6] para controle dos *gateways* (definido mais a frente) para a rede PSTN, e o SDP[7] para descrever sessões multimídias. Todos estes protocolos fazem parte do ambiente IP, que chamamos de *suite* IP. Portanto, o SIP pode ser usado conjuntamente com outros protocolos com a finalidade de prover serviços completos aos usuários. Entretanto, a funcionalidade básica e operação do SIP não dependem de nenhum destes protocolos.

O SIP dá suporte a cinco propriedades importantes no estabelecimento e finalização de comunicações multimídia:

- **situação do usuário:** determinação do sistema terminal que será usado para comunicação;
- **disponibilidade do usuário:** determinação da disposição da parte chamada para entrar em comunicação;
- **configuração de sessão:** estabelecimento de parâmetros de sessão em ambas as partes

[2] *Request For Comments*
[3] *Real-Time Transport Protocol*
[4] *Quality of Service*
[5] *Real-Time Streaming Protocol*
[6] *Media Gateway Control Protocol*
[7] *Session Description Protocol*

destino e fonte;

- **gerenciamento de sessão:** inclui transferência e finalização de sessões, modificação dos parâmetros de sessão, e a invocação de serviços.

Ao invés de oferecer serviços, o SIP oferece primitivas as quais podem ser utilizadas para a implementação de diferentes serviços. Como exemplo do uso das primitivas do SIP para a implementação de serviços, podemos citar que ele pode localizar um usuário e entregar um objeto opaco (pois para o SIP não interessa o que há no objeto) na localização corrente do usuário desejado (exemplo retirado de [2]). Se esta primitiva for usada para a entrega de um descritor de sessão escrito em SDP, por exemplo, os terminais poderão concordar nos parâmetros da sessão. Se a mesma primitiva for usada para entregar além da descrição SDP também uma foto da pessoa que disca, um serviço de identificador de chamadas poderá ser facilmente implementado. Ou seja, como os exemplos mostram, uma única primitiva é tipicamente usada para prover diferentes tipos de serviços.

O SDP é um protocolo de descrição de sessão utilizado para descrever e codificar características dos participantes da sessão, de forma que todos possam concordar com o *codec* ou o protocolo de transporte utilizados, permitindo que a conversação seja estabelecida.

2.2 Estrutura do Protocolo

2.2.1 Arquitetura da Aplicação

Desde que o SIP é apenas um protocolo de sinalização, ele não se preocupa com a mídia utilizada. Portanto, outros protocolos são necessários. Na Figura 2.1 está ilustrada a pilha de protocolos necessários para uma aplicação SIP.

É importante destacar que o SIP não é entendido como um protocolo de uso geral. Seu propósito é apenas promover a comunicação, mas a comunicação em si precisa ser suportada por outros protocolos da arquitetura IP (o que chamamos de *suite* IP). Para uma comu-

nicação em tempo real seria necessária a utilização do protocolo RTP[8] sobre o UDP[9]sobre IP (ver Figura 2.1). Dessa forma, os protocolos adicionais necessários para a transmissão propriamente dita serão oferecidos pelo núcleo (*kernel*) do sistema operacional escolhido, sobre o qual a aplicação SIP residirá.

Dois importantes protocolos para o SIP são o SDP e o RTP. O SDP é utilizado para ajustar a mídia da sessão, e o RTP é usada para o transporte de mídia.

Um problema que deve-se ter em mente com respeito à transporte é a falta de largura de banda para a transmissão de informação. Portanto, a codificação e decodificação desempenham um importante papel em chamadas VoIP. Para isto, são utilizados os *codecs* de áudio e vídeo. A própria natureza da Internet levanta a questão sobre qualidade de serviço (QoS), por fazer uso de uma conexão orientada à pacotes. Cada pacote segue um caminho independente, podendo chegar ao destino fora de ordem e com atrasos variáveis entre chegadas de cada pacote. SIP e RTP não implementam QoS, e um outro protocolo deve ser usado para isto. Um protocolo para esta finalidade, que utiliza a mesma pilha e trabalha conjuntamente como o RTP é o RTCP[10]. Um outro protocolo para esta finalidade é o RSVP[11].

2.2.2 Arquitetura do Protocolo

O SIP, por sua vez, é também estruturado como um protocolo organizado em camadas, o que significa que seu comportamento é descrito em termos de um conjunto de estágios de processamento idealmente independentes, com apenas um fraco acoplamento entre cada estágio. Estas camadas podem ser visualizadas na Figura 2.2.O comportamento do protocolo é descrito como camadas para o propósito de apresentação, permitindo a descrição de

[8]O RTP (*Real-Time Protocol*) é um protocolo da arquitetura IP usado para *stream* de mídia (áudio e vídeo), onde uma pronta entrega torna-se necessária [3].

[9]O UDP (*User Datagram Protocol*) é a versão datagrama para a camada de transporte da arquitetura IP. Esta camada é projetada de forma a permitir que pares de entidades dos servidores fonte e destino mantenham uma conversação.

[10]O *Real Time Control Protocol* é um protocolo usado para monitorar a QoS e levar informação sobre participantes de uma sessão formada. Pertence à *suite* IP.

[11]*Reservation Protocol*. É utilizado para reservar recursos da rede, para uma conexão de voz sobre IP, por exemplo.

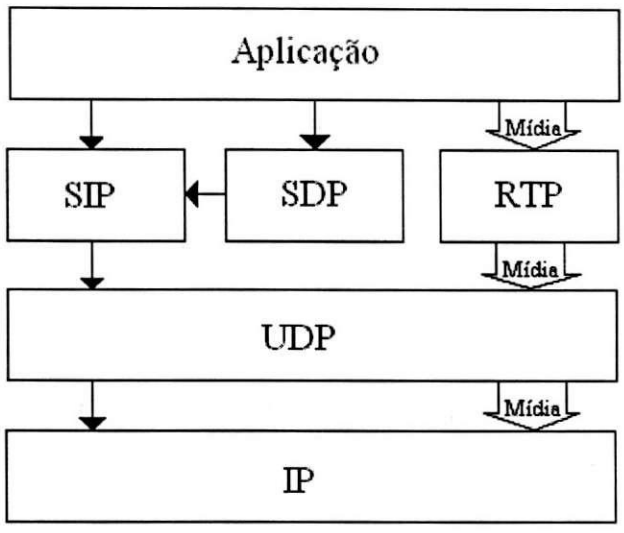

Figura 2.1: Pilha de protocolos de uma aplicação SIP.

funções comuns através de elementos em uma única sessão. Ele não dita nenhuma forma de implementação. Quando dizemos que um elemento contém camadas, entendemos que ele é complacente com um conjunto de regras definidas por aquela camada.

Nem todo elemento especificado pelo protocolo possui todas as camadas. Além disso, os elementos especificados pelo SIP são entidades lógicas, e não físicas.

A camada mais baixa do SIP é sua *camada de sintaxe e codificação.* Sua codificação é especificada utilizando a gramática ABFN[12][6].

A segunda camada é a *camada de transporte.* Ela define como um cliente envia pedidos e recebe respostas, e como um servidor recebe pedidos e envia respostas sobre a rede. Todos os elementos SIP possuem esta camada. Esta camada se preocupa com a determinação da conexão a ser usada para um pedido ou resposta no caso de transporte orientado à conexão. Todo elemento SIP precisa implementar os protocolos de transporte UDP e TCP.

A terceira camada é a *camada de transação.* Transações são componentes fundamentais do SIP. Uma transação é um pedido enviado pelo cliente transação, utilizando a camada de transporte, para um servidor transação, ao longo da qual todas as respostas para aquele

[12] *Augmented Backus-Naur Format*

pedido enviadas a partir do servidor transação voltam ao cliente. A camada de transação trata de retransmissões da camada de aplicação, comparando respostas com pedidos, e fim do tempo de vida das mensagens (*timeouts*) no nível de aplicação.

Qualquer tarefa que um *user agent client* (UAC) pode executar é realizada fazendo-se uso de uma série de transações. *User agents* (UA) possuem a camada de transação assim como os servidores *proxy stateful*. Servidores *stateless proxy* não possuem esta camada.

A camada de transação possui um componente cliente, referenciado com cliente transação (*client transaction*), e um componente servidor, chamado servidor transação (*server transaction*). Cada componente é representado por uma máquina de estados finitos que é construída com a finalidade de processar um pedido em particular.

A camada acima da camada de transação é chamada *camada de transação usuário* (*Transaction User* (TU)). Cada uma das entidades SIP, exceto *stateless proxies*, é um transação usuário.

Quando um TU deseja enviar um pedido, ele cria uma instância transação cliente e passa para ela o pedido, junto com o endereço IP, a porta, e transporte para o qual enviar o pedido. Um TU que cria um transação cliente pode também cancelá-lo. Quando um cliente cancela uma transação, ele pede que o servidor pare processamentos adicionais, retorna para o estado que existia antes da transação ser iniciada, e gera uma resposta de erro específica para àquela transação. Tudo isto é feito com o pedido CANCEL, o qual constitui uma transação própria, mas referencia a transação a ser cancelada.

Os elementos SIP possuem um núcleo que os diferenciam uns dos outros., Núcleos, exceto para *stateless proxies*, são TUs. Enquanto que o comportamento do núcleo de um UAC e UAS depende do método, existem algumas regras comuns à todos os métodos. Para UAC, estas regras governam a construção de pedidos; para UAS, elas governam o processamento de um pedido e a geração de uma resposta.

2.3 Cenários Típicos do SIP

Neste capítulo vamos exemplificar resumidamente três cenários básicos que tipicamente

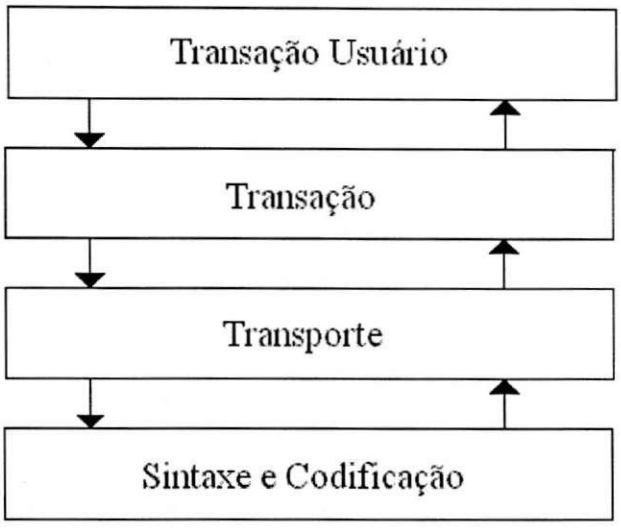

Figura 2.2: Arquitetura do protocolo SIP.

se configuram em um tráfego de mensagens SIP

2.3.1 Registro

Usuários precisam se cadastrar em um servidor *Registrar* para que possam receber chamadas de outros usuários. Um registro compreende um pedido REGISTER seguido de uma resposta 200 OK enviada pelo *registrar* para o *user agent* que originou o pedido informado que o registro foi realizado com sucesso. Geralmente, registros devem ser autorizados, podendo surgir uma resposta 407 se o usuário não enviar as credenciais válidas, indicando que é necessária a autenticação no servidor para este serviço. Um exemplo de registro é ilustrado na Figura 2.3

2.3.2 Convite

Um convite de sessão inicializa-se com um pedido INVITE enviado por um *user agent*

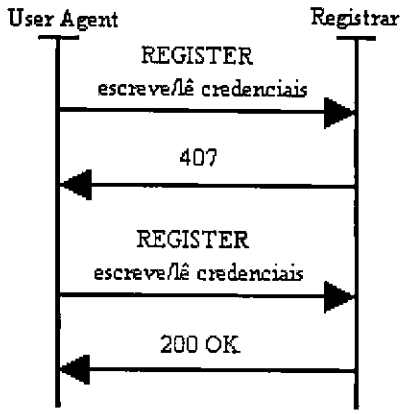

Figura 2.3: Fluxo da mensagem REGISTER.

para um servidor *proxy*. O *proxy* envia imediatamente uma resposta 100 Trying para parar retransmissões e passar adiante o pedido.

Todas as respostas provisionais enviadas pelo destino final da chamada devem ser enviadas de volta para o originador da chamada. Repare a resposta 180 Ringing no fluxo de chamada da Figura 2.4. Este resposta é gerada quando o telefone da parte chamada começa a tocar.

Um 200 OK é gerado uma vez que a parte chamada atende o telefone e é retransmitida pelo *user agent* da parte chamada até que esta receba um confirmação ACK do originador da chamada. A sessão é estabelecida neste ponto, e um fluxo de mídia é estabelecido entre os dois *user agents* (comunicação fim-a-fim).

2.3.3 Finalização de Sessão

A finalização de uma sessão estabelecida é realizada pelo envio do pedido BYE dentro de um diálogo estabelecido pela mensagem INVITE. Mensagens BYE são enviadas diretamente de um *user agent* para outro, ao menos que algum *proxy* no caminho do pedido INVITE tenha indicado que deseja intermediar a conversação fazendo uso do cabeçalho "**Record-Route**", gravando informações de roteamento.

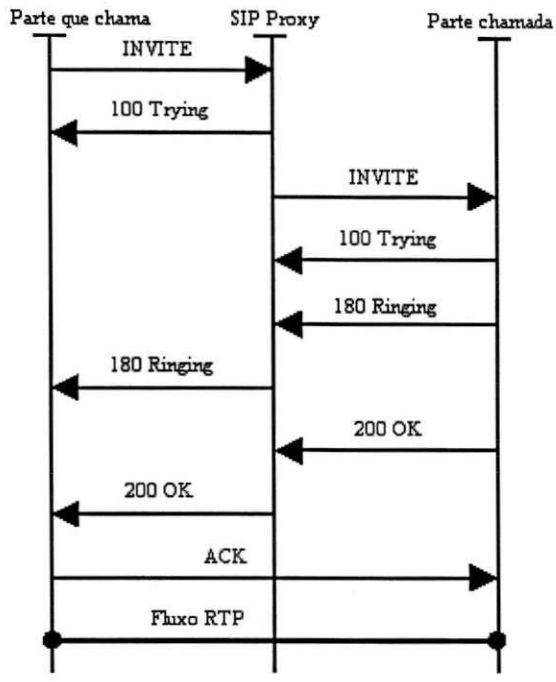

Figura 2.4: Fluxo da mensagem INVITE.

A parte interessada em finalizar a sessão envia um BYE para a outra parte envolvida na sessão. A outra parte envia uma resposta 200 OK para confirmar o BYE e a sessão é terminada (ver Figura 2.5).

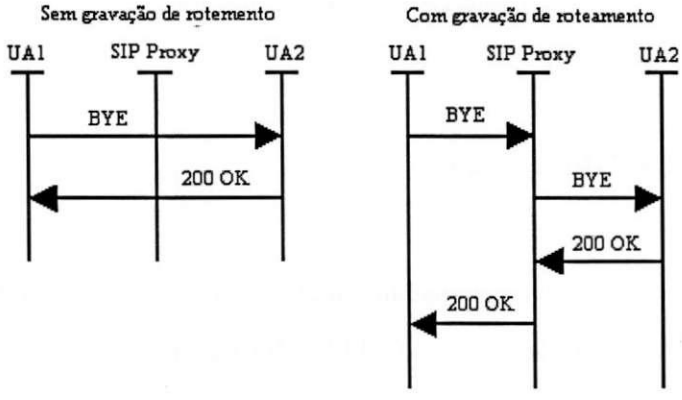

Figura 2.5: Fluxo da mensagem BYE.

Capítulo 3

Bibliotecas de Desenvolvimento

Entre as bibliotecas de código aberto (*open source*) disponíveis como pilha do protocolo SIP para a plataforma Linux encontradas, as mais populares são a *Vocal* e a *oSIP*. Para este trabalho escolhemos a oSIP pelo fato de ser uma implementação fiel do protocolo e possuir características compatíveis com o trabalho proposto.

A oSIP é uma biblioteca de propósito geral, flexível tanto para construção de servidores *proxies* quanto para desenvolvimento de *user agents*. O preço pago por essa flexibilidade é uma enorme complexidade na sua utilização. Dessa forma, optamos pela escolha de uma outra biblioteca que funciona um nível acima da oSIP: é a biblioteca *eXosip*. A eXosip é uma biblioteca específica para desenvolvimento de terminais SIP (*user agents*). Ela abstrai para o usuário a complexidade da oSIP, não sendo mais necessário ao desenvolvedor um conhecimento tão profundo das especificações do protocolo SIP.

3.1 oSIP

A pilha oSIP é uma biblioteca SIP e uma pilha SIP, a qual está disponível como código aberto (*open source*) sob a licensa GPL [11]. Esta pilha provê uma API para criação, fragmentação e modificação de mensagens SIP e SDP. É uma iniciativa de Aymeric Mozard. Ela se oferece como uma biblioteca flexível e auto-suficiente, que pode ser utilizada para a

construção de aplicações SIP cliente, *proxy* e *registrar*.

oSIP é uma implementação do SIP que promete oferecer para desenvolvedores de software multimídia e de telecomunicações uma interface fácil e poderosa para inicializar e controlar sessões SIP na suas aplicações. Por ser pequena em tamanho e código, pode ser implementada tanto em telefones IP como em dispositivos móveis embarcados, sem no entanto ser limitada para agentes finais. oSIP pode também ser utilizada para implementação de SIP *Proxies*.

A biblioteca oSIP é dividida em três partes [11]:

- **Parser**: é um fragmentador para extrair e compor URIs, mensagens e conteúdos que definem a sessão;
- **Finite State Machine (FSM)**: máquina de estados que utiliza o fragmentador para geração de estados. É a camada de transação do SIP, que por sua vez é controlada pelo *user agent* (UA);
- **Transaction Manager**: gerenciador de transação que controla as filas de eventos ou transações na máquina de estados finitos.

Como vantagens, podemos citar sua natureza leve, sua fidelidade às especificações SIP [2] e sua portabilidade. Dizemos que é leve no sentido de que requer muito pouca memória e poder de processamento normal, o que permite ser executada em pequenos sistemas operacionais. Ela pode ser executada no *Familiar*, uma distribuição de Linux Embarcado para *handhelds*. Por ser fiel ao SIP original, torna-se fácil de interpretar e prover documentação. oSIP pode ser embarcada em qualquer plataforma, sendo então portável.

Oferecer uma única API tão flexível ao ponto de poder ser utilizada no desenvolvimento de clientes indo até *proxies*, servidores de conferência e *registrars* a torna tão grande e complexa que atrasa o tempo de entendimento e desenvolvimento e dificulta bastante a criação de uma aplicação SIP mais simples. É necessário uma API mais específica para evitar que programadores percam muito tempo tendo que entender profundamente os pormenores do protocolo SIP.

Outra desvantagem, é que as mensagens SIP não podem ser construídas com um simples pedido ou resposta, tendo que ser montadas manualmente.

Assim, o autor partiu para a tentativa de oferecer uma API específica para desenvolvimento de UA. É a *eXosip2*, que será discutida na próxima seção.

3.2 eXosip2

3.2.1 Informações Gerais

A pilha GNU[1] oSIP é o primeiro passo necessário para implementar aplicações SIP. Como uma camada mais baixa da pilha, não existe limitações na forma de usá-la, porém sua enorme API faz torna esta pilha muito complexa e inadequada para iniciantes e programadores em geral.

A biblioteca *libeXosip*, da pilha *eXosip* (*eXtended oSIP*), é baseada na oSIP, e implementa um pouco mais que as funções SIP, de forma que seu usuário possa construir aplicações SIP rapidamente[2].

libeXosip é uma biblioteca sob a licensa GPL[3] que extende as capacidades da biblioteca oSIP. Ela visa a implementação de uma API de programação de alto nível simples, capaz de controlar o SIP no estabelecimento de sessões e extensões comuns. A pilha *eXosip2* é uma referência à *eXtended eXoSIP*. Ela é uma reformulação da *eXosip* que foi totalmente reescrita.

A página principal do projeto é HTTP://SAVANNAH.GNU.ORG/PROJECTS/EXOSIP . Lá encontra-se um *link* para a área de *download*, porém nenhuma documentação é disponibilizada. Este foi um grande problema encontrado durante este trabalho. Até o momento,

[1]Projeto que culminou na criação de um sistema operacional compatível com o UNIX, mas que não sofresse das restrições de nome e código proprietários. GNU é um acrônimo para *GNU is Not Unix*. Para mais informações: http://www.gnu.org .

[2]Na verdade, esta biblioteca ainda não conseguiu atingir este nível por falta de documentação, tornando o desenvolvimento bastante penoso e demorado.

[3]GPL é um acrônimo para *GNU Public License*, uma licensa que garante que ao adquirir um software, o usuário não terá restrições quanto ao seu uso.

a versão mais nova disponível é a versão 1.9.1, um *pre-release.* Esperamos que quando realmente lançada, seu autor disponibilize uma boa documentação, requisito extremamente necessário para agilizar o processo de desenvolvimento.

A documentação pode ser gerada a partir do código fonte da biblioteca, com o auxílio do programa DOXYGEN. Foi apenas dessa forma que conseguimos obter alguma informação sobre a API e compreender um pouco do seu funcionamento. Esta biblioteca foi desenvolvida pelo mesmo autor da oSIP, Aymeric Mozard.

Como característica, *eXosip2* oferece suporte para:

- **REGISTER**: um cliente pode registrar um ou mais endereços de sua localização;
- **INVITE ou re-INVITE**: é usado para iniciar uma chamada ou modificar uma chamada já existente;
- **OPTIONS**: é um pedido enviado a um servidor para saber as capacidades, sendo que o servidor pode enviar de volta uma lista de métodos. Em alguns casos, o servidor pode também enviar as capacidades de algum usuário requisitado;
- **INFO, UPDATE**: outros métodos dentro de chamadas;
- **REFER**: realiza transferência de chamadas;
- **PRACK**: garante confiabilidade para respostas provisionais;
- **SUBSCRIBE/NOTIFY**: associado ao pacote de eventos SIP;
- **PUBLISH**: publicação de estado de evento ;
- **MESSAGE**: envio de mensagens instantâneas entre *user agents.*

A API *eXosip2* promete ser flexível o bastante para permitir que o programador controle e complete as mensagens SIP. Dessa forma, a pilha é mantida extensível.

Enquanto que oSIP pode ser utilizada para construção de qualquer aplicação SIP, *eXosip2* provê API apenas para controle dos terminais SIP. O autor promete que uma vez finalizada, esta biblioteca fornecerá API para gerenciamento de chamadas e para características de mensagem e presença.

A FSF[4] defende este conceito de software livre, ou seja, software livre de restrições, sujeito a GNU GPL. Assim, segundo esta licensa, embora possa existir um custo envolvido na obtenção do software, dali em diante ele pode ser utilizada de qualquer forma desejada, sendo usualmente distribuído sob a forma de código fonte.

Sendo *eXosip2* um software livre GNU GPL, o programador que construir aplicações SIP com seu auxílio deverá lançar sua aplicação sob a licensa GPL. Em outras palavras, a aplicação pode ser até adquirida mediante algum custo, mas o desenvolvedor precisa disponibilizar de alguma forma o código fonte de sua aplicação, tornando-a código aberto (ıopen source).

Como oSIP é LGPL[5], ou seja, uma GPL menos restrita, o desenvolvedor pode optar por implementar aplicações proprietárias e ligá-las com oSIP.

3.2.2 Instalação

Requisitos

Como o SIP é apenas um protocolo de sinalização, não é de sua responsabilidade a conversação propriamente dita, ou seja, o transporte do fluxo de voz de uma chamada deve ser realizado por outros protocolos adicionais. Como vimos no Capítulo 2.2, utiliza-se a *suite* de protocolos IP para implementar o fluxo de mídia, ou seja, o RTP sobre UDP sobre IP.

Assim, para instalar *eXosip2*, precisa-se antes instalar o *oRTP*, que é uma biblioteca C LPGL implementando o protocolo RTP, definido na RFC 1889[3], como forma de suprir a ausência de implementação deste protocolo por parte do *kernel* 2.4 do Linux, que implementa apenas o UDP e IP necessários. oRTP pode ser baixado gratuitamente em :
http://www.ortp.org .

A oRTP tem como objetivo básico possibilitar o envio e recebimento de pacotes RTP através de redes IP v.4, implementando funcionalidades como identificação da informação

[4] *Free Software Foundation.*
[5] *Lesser GPL*

carregada, checagem de ordem de entrega de pacotes e transporte de informação de sincronismo do codificador/decodificador (*timestamp*). Dessa forma, uma compensação de *jitter* é realizada.

Por sua vez, oRTP tem como pré-requisito a biblioteca *glib*, com versão superior a 2.0.0, que pode ser baixada gratuitamente em **http://www.gtk.org** .

Procedimentos de Instalação

Este programa foi desenvolvido para executar no Linux, mas poderá funcionar em qualquer sistema Unix.

O procedimento de instalação natural seria:

```
$ configure
$ make
$ su -c 'make install'
```

Porém, *eXosip2* vem com uma aplicação demo chamada *josua* (*Jack's open sip User Agent*), que é uma implementação do SIP baseada na oSIP . *josua* é um *user agent* bem primitivo, sendo uma aplicação de console, ou seja, não possui uma interface gráfica. É como se fosse o esqueleto de um *softphone*, servindo como base inicial para outras aplicações mais sofisticadas.

Esta aplicação utiliza uma biblioteca adicional chamada *mediastreamer*, responsável por toda a parte de mídia do *josua* (já que isto não diz respeito ao SIP), ou seja, ela implementa as funções necessárias para montar os pacotes de voz e enviá-los. O *mediastreamer* vem junto com uma aplicação bastante conhecida: o *linphone*, um *softphone* aberto bastante popular. Este *mediastreamer* foi desenvolvido pelo pessoal do *linphone*, sendo a parte essencial deste aplicativo. Porém não encontra-se na web o pacote de instalação do *mediastreamer*, e nem instalando-se o *linphone* e obtendo-se o código fonte do *mediastreamer*, consegue-se instalar o *eXosip2* da forma mostrada acima.

Foram encontrados diversos relatos do mesmo problema em fóruns de discussão na Internet, e mesmo sendo este um problema aparentemente antigo, acontecendo também com o

demo *osipua* do oSIP, e há muito já conhecido pelo autor, nenhuma providência foi tomada. Sendo assim, a única forma de se conseguir compilar e instalar *eXosip2* é desabilitando seu demo, da seguinte forma:

```
$ ./configure --disable-josua
$ make
$ su -c 'make install'
```

3.2.3 Inicialização

Quando utilizando *eXosip2*, nossa primeira tarefa é inicializar o contexto *eXosip2* e a biblioteca *libosip* (fragmentador e máquina de estados). Isto precisa ser realizado antes de qualquer utilização de *eXosip2*. O trecho de código abaixo, retirado do arquivo main_ncurses.c de *josua* nos mostra como realizar esta tarefa:

```
#include <eXosip2/eXosip.h>

int i;

TRACE_INITIALIZE (6, stdout);

i=eXosip_init();
if (i!=0)
  return -1;

i = eXosip_listen_to(IPPROTO_UDP, INADDR_ANY, cfg.port);
if (i!=0)
  {
    eXosip_quit();
    fprintf (stderr, "could not initialize transport layer\n");
```

```
    return -1;
  }
... então precisamos enviar mensagens e esperar por eventos eXosip ...
```

De forma resumida, este código inicializa o *osip trace*, inicializa as pilhas oSIP e eXosip, e abre um *socket* UDP para sinalização. Consulte a API contida na documentação gerada pelo DOXYGEN para entender melhor cada função e seus parâmetros.

Agora precisamos esperar e tratar eventos eXosip. Veja um código para capturar eventos *eXosip_event* da pilha eXosip2:

```
eXosip_event_t *je;
for (;;)
  {
    je = eXosip_event_wait (0, 50);
    eXosip_automatic_action ();
    if (je == NULL)
      break;
    if (je->type == EXOSIP_CALL_NEW)
      {
      ....
      ....
      }
    else if (je->type == EXOSIP_CALL_ACK)
      {
      ....
      ....
      }
    else if (je->type == EXOSIP_CALL_ANSWERED)
      {
      ....
```

```
    ....
    }
  else .....
  ....
  ....
}
```

Será recebido um evento para cada mensagem SIP enviada. Cada evento contém o pedido original da transação afetada e a última resposta que dispara o evento quando disponível.

Assim, pode-se acessar todos os cabeçalhos destas mensagens e armazená-los no seu próprio contexto para outras aplicações ou *displays* gráficos.

Quando se recebe um pedido REFER para transferência de chamada, por exemplo, tipicamente retira-se o cabeçalho "**Refer-To**":

```
osip_header_t *referto_head = NULL;
i = osip_message_header_get_byname (evt->sip,
                          "refer-to", 0, &referto_head);
if (referto_head == NULL || referto_head->hvalue == NULL)
```

Os eventos *eXosip_event* também contém identificadores para chamadas, registros, assinaturas de saída ou chegada, quando aplicável. Tais identificadores são utilizados na API para controlar chamadas, registros, assinaturas de saída ou chegada. Esta API irá construir mensagens padrão com os cabeçalhos SIP usuais e enviar tais mensagens, sem o programador precisar se preocupar com isto.

3.2.4 Como Iniciar, Modificar ou Terminar Chamadas

Iniciar uma Chamada

Para iniciar uma chamada, precisamos de alguns cabeçalhos que serão utilizados por eXosip2 para construir o pedido padrão INVITE:

```
osip_message_t *invite;
  int i;

  i = eXosip_call_build_initial_invite (&invite, "<sip:to@antisip.com>",
                                        "<sip:from@antisip.com>",
                                        NULL, // optional route header
                                        "This is a call for a conversation");
  if (i != 0)
    {
      return -1;
    }

  osip_message_set_supported (invite, "100rel");

  {
    char tmp[4096];
    char localip[128];

    eXosip_guess_localip (AF_INET, localip, 128);
    snprintf (tmp, 4096,
              "v=0\r\n"
              "o=josua 0 0 IN IP4 %s\r\n"
              "s=conversation\r\n"
              "c=IN IP4 %s\r\n"
              "t=0 0\r\n"
              "m=audio %s RTP/AVP 0 8 101\r\n"
              "a=rtpmap:0 PCMU/8000\r\n"
              "a=rtpmap:8 PCMA/8000\r\n"
              "a=rtpmap:101 telephone-event/8000\r\n"
              "a=fmtp:101 0-11\r\n", localip, localip, port);
```

```
  osip_message_set_body (invite, tmp, strlen (tmp));
  osip_message_set_content_type (invite, "application/sdp");
}

eXosip_lock ();
i = eXosip_call_send_initial_invite (invite);
if (i > 0)
  {
    eXosip_call_set_reference (i, reference);
  }
eXosip_unlock ();
return i;
```

O código acima utiliza a função *eXosip_call_build_invite* para construir um pedido SIP padrão INVITE para estabelecimento de uma nova chamada. Note que deve-se inserir um corpo de mensagem SDP informando os parâmetros de áudio utilizado para o fluxo RTP.

O elemento de retorno de *eXosip_call_build_invite* é o identificador da chamada que poderá ser usado para enviar um pedido CANCEL. Nos eventos futuros que não 100 Trying, também será retornado o identificador do diálogo, que também é necessário para controlar o estabelecimento de chamadas.

Responder uma Chamada

Para responder uma chamada, precisa-se enviar uma resposta 180 Ringing ao receber um convite INVITE:

```
eXosip_lock ();
eXosip_call_send_answer (ca->tid, 180, NULL);
eXosip_unlock ();
```

Então, se o usuário desejar receber a chamada, ele precisa enviar um 200 OK e inserir um corpo SDP na sua resposta:

```
osip_message_t *answer = NULL;

  eXosip_lock ();
  i = eXosip_call_build_answer (ca->tid, 200, &answer);
  if (i != 0)
  {
     eXosip_call_send_answer (ca->tid, 400, NULL);
  }
  else
  {
     i = sdp_complete_200ok (ca->did, answer);
     if (i != 0)
     {
        osip_message_free (answer);
        eXosip_call_send_answer (ca->tid, 415, NULL);
     }
     else
        eXosip_call_send_answer (ca->tid, 200, answer);
  }
  eXosip_unlock ();
```

Enviando Outros Pedidos

A API de controle de chamada permite que se envie e receba REFER, UPDATE, INFO, OPTIONS, NOTIFY e INVITEs com uma sessão já estabelecida. Como exemplo, vamos ver um trecho de código referente ao envio de um pedido INFO, usado para enviar um DTMF fora de faixa dentro da camada de sinalização:

```
osip_message_t *info;
  char dtmf_body[1000];
  int i;

  eXosip_lock ();
  i = eXosip_call_build_info (ca->did, &info);
  if (i == 0)
  {
     snprintf (dtmf_body, 999, "Signal=%c\r\nDuration=250\r\n", c);
     osip_message_set_content_type (info, "application/dtmf-relay");
     osip_message_set_body (info, dtmf_body, strlen (dtmf_body));
     i = eXosip_call_send_request (ca->did, info);
  }
  eXosip_unlock ();
```

Capítulo 4

Rede MVoIP

O projeto MVoIP é uma iniciativa para a implementação de uma rede de voz sobre IP (VoIP) no campus da Universidade Federal de Campina Grande, a UFCG. O grupo é formado pelos alunos de graduação em Engenharia Elétrica Danilo Freire, José Luís, Olympio Cipriano e por mim, Daniel Uchôa.

A primeira etapa do projeto consistia de montar uma rede local para VoIP com usuários fixos e móveis. Como usuários fixos, a rede atende a computadores PCs e telefones analógicos. Os dispositivos móveis utilizadas são aparelhos celulares e *handhelds*.

A arquitetura básica da rede local para voz sobre IP é ilustrada na Figura 4.1. A rede MVoIP está ligada a uma rede de telefonia analógica privada, que por sua vez liga-se a rede PSTN através de um PABX.

Na rede IP local temos PCs e telefones IP. Ilustra-se também na figura uma rede sem-fio, onde *handhelds* e aparelhos celulares são conectados utilizando um enlace *Bluetooth* ou *Wi-Fi*.

Os *gateways* são tradutores de protocolos que permitem a interligação das redes. Os protocolos da rede telefônica e da rede sem-fio são traduzidos para o protocolo IP. A rede sem-fio na verdade pode utilizar um enlace que encapsula o protocolo IP, sendo então seus componentes considerados terminas IP.

Os servidores de chamadas redirecionam as chamadas para a própria subrede ou para outra rede local, dependendo do número chamado. Chamadas para a Internet ou para a rede

PSTN são redirecionadas para o *gateway*, que se encarrega de repassá-las para o seu destino.

No padrão SIP, os PCs, telefones IP, *handhelds* ou dispositivos celulares hospedam os *user agents*. Estes são terminais IP que utilizam o protocolo SIP para localizarem uns aos outros e para negociar características de sessão, tal como o *codec* utilizado. O *user agent* é um software que reside no dispositivo terminal de cada usuário como forma de uma aplicação.

Neste estágio de implementação, a rede foi toda construída para a plataforma Linux, utilizando soluções de código aberto disponíveis gratuitamente pela web. A aplicação SIP cliente no aparelho celular ainda não foi desenvolvida, e será construída sob a plataforma Symbian [8].

Os servidores *proxy* e *registrar*, e o *gateway* para a rede de telefonia privada, foram implementados pelo *Asterisk*, um *software open source* para Linux que está se tornando bastante popular [12]. O *gateway* para a rede móvel foi implementado a partir de *scripts shell*.

Minha maior participação neste projeto foi na implementação do cliente SIP em um dispositivo móvel embarcado: um *handheld* da Compaq. O *Familiar*, uma distribuição do Linux para este tipo de dispositivo, foi instalado no aparelho. O *SIP phone* escolhido foi o *Linphone* [13], que tem uma versão para o *Familiar*. Depois de instalado, o *Linphone* foi configurado para se registrar no servidor *Registrar*, e enviar todas as chamadas realizadas no *handheld* para o servidor *Proxy*, ambas entidades lógicas implementadas pelo *Asterisk*. A configuração das tecnologias de comunicação disponíveis no dispositivo, *Bluetooth* e *Wi-Fi*, também foi realizada.

Estas etapas concluídas, o usuário já está apto a realizar chamadas através de seu dispositivo, bastando para isto digitar a URI do usuário que deseja chamar ou apenas escolher um nome da agenda de contatos do aplicativo.

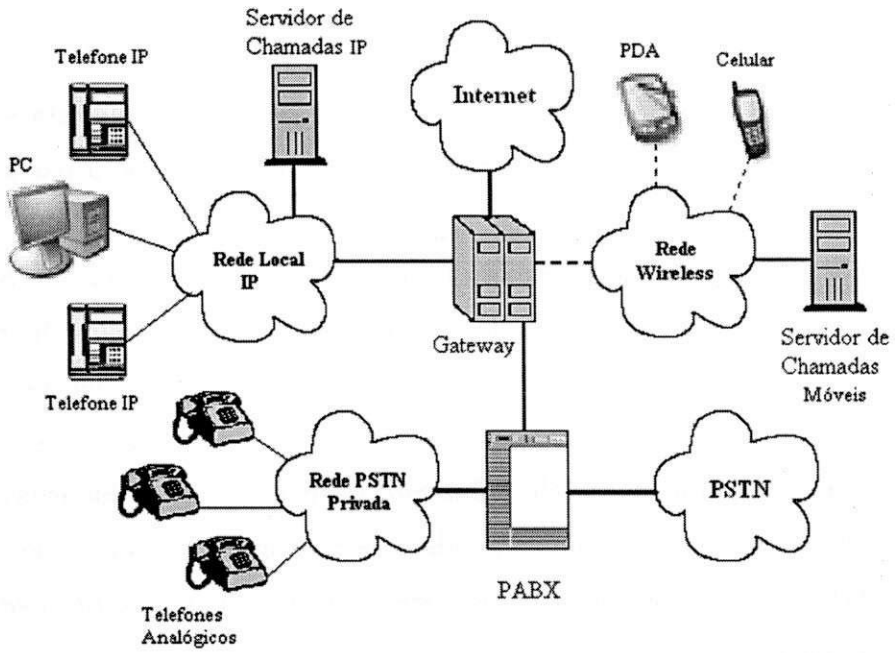

Figura 4.1: Arquitetura de uma rede local para voz sobre IP.

Capítulo 5

Conclusão

O protocolo de inicialização de sessão SIP pode ser utilizado com uma boa opção para prover sinalização para comunicação entre diversos dispositivos usando diferentes plataformas. A portabilidade do SIP faz com que entidades residentes em PCs, telefones IP, aparelhos celulares, *handhelds* e até TVs Digitais, possam estabelecer uma sessão multimídia fim-a-fim.

O sistema operacional Linux também apresenta uma certa flexibilidade, sendo uma opção de plataforma viável para todos estes dispositivos citados. Para nosso dispositivo alvo, o *handheld* iPAQ h3900, uma distribuição Linux chamada *Familiar* está disponível. Estes aparelhos vêm com o *Windows for Pocket PC*, mais conhecido como WinCE, pré-instalado, devendo ser substituído por seu proprietário pelo *Familiar*.

Uma implementação aberta para Linux bastante popular é a biblioteca e pilha oSIP. A pilha oSIP oferece uma API de programação genérica, podendo ser utilizada tanto para desenvolvimento de aplicações mais simples, como terminais SIP, quanto entidades mais sofisticadas e robustas, como o caso de servidores SIP. Esta flexibilidade, porém, tornou-se uma desvantagem, desde que a API teve que ficar bastante grande para atender todos estes propósitos. Isto tornou o aprendizado e desenvolvimento de aplicação simples específicas mais lento e custoso.

Como solução a este problema, o mesmo desenvolvedor de oSIP está trabalhando na tentativa de oferecer aos programadores uma API específica para criação de *user agents*, sendo bem mais simples, o que facilita o entendimento e agiliza o desenvolvimento. Esta

pilha é uma extensão de oSIP, e é chamada de *eXosip*. No entanto, até o presente momento, este objetivo não foi ainda alcançado. Esta pilha peca na falta de documentação e na complexidade inicial de seu uso.

Dado as dificuldades no desenvolvimento de *user agents* acima assinaladas, optou-se pela utilização de uma solução cliente SIP que fosse código aberto (*open source*). Uma aplicação deste tipo que está se tornando bastante popular e que possui uma versão para a distribuição Linux *Familiar*, sendo então totalmente adequada aos nossos propósitos, é o *Linphone*. Este é um *softphone* SIP aparentemente bastante estável, que oferece uma interface gráfica com o usuário bastante agradável. O *Linphone* foi desenvolvido utilizando a API do oSIP.

Para a implementação do *user agent* SIP residente no *handheld* da rede MVoIP foram necessárias três etapas: a instalação da distribuição Linux *Familiar*, produzida pela Compaq para dispositivos *handhelds*; a instalação da versão do *Linphone* para o *Familiar*; e a etapa de configuração. Esta última consistiu de dois passos: configuração das tecnologias de comunicação *Bluetooth* e *Wi-Fi*, e configuração do *Linphone* para se registrar no servidor *Registrar*, e enviar todas as chamadas realizadas no *handheld* para o servidor *Proxy*. O dispositivo móvel embarcado foi então incorporado à rede MVoIP e testado. Tudo funcionou corretamente.

Bibliografia

[1] RFC 971: "*IP: Internet Protocol*", Internet Engineering Task Force's - IETF, Network Working Group, disponível em http://www.ietf.org.html .

[2] RFC 3261: "*SIP: Session Initiation Protocol*", Internet Engineering Task Force's - IETF, Network Working Group, disponível em http://www.ietf.org.html .

[3] RFC 1889: "*RTP: A Transport Protocol for Real-Time Applications*", Internet Engineering Task Force's - IETF, Network Working Group, disponível em http://www.ietf.org.html .

[4] RFC 2326: "*RTSP: Real-Time Streaming Protocol (RTSP)*", Internet Engineering Task Force's - IETF, Network Working Group, disponível em http://www.ietf.org.html .

[5] RFC 3015: "*Megaco Protocol Version 1.0*", Internet Engineering Task Force's - IETF, Network Working Group, disponível em http://www.ietf.org.html .

[6] RFC 2234: "*Augmented BNF for Syntax Specifications: ABNF*", Internet Engineering Task Force's - IETF, Network Working Group, disponível em http://www.ietf.org.html .

[7] TERENA (2004): "*IP Telephony Cookbook*", TERENA Report.

[8] Edwards, L., Barker, R. (2004) : "*Nokia Mobile Developer Series - Developing Series 60 Applications: A Guide for Symbian OS C++ Developers*", Addison-Wesley, New York, USA.

[9] Hollabaugh, C. (2002): "*Embedded Linux*", Addison-Wesley, New York, USA.

[10] Tanenbaum, A. S.(1996): "*Computer Networks*", Prentice-Hall, Inc., New Jersey, USA.

[11] http://www.gnu.org/software/osip/doc/osip-0.7.x.html .

[12] http://www.asterisk.org .

[13] http://www.linphone.org/doc/us/manual/index.html .