Universidade do Minho
Escola de Engenharia

António Daniel da Mota Lopes

VoIP em Redes Peer-to-Peer

Outubro de 2014

António Daniel da Mota Lopes VoIP em Redes Peer-to-Peer

UMinho | 2014

Universidade do Minho
Escola de Engenharia

António Daniel da Mota Lopes

VoIP em Redes Peer-to-Peer

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia de Comunicacões

Trabalho efetuado sob a orientação de
Professora Doutora Maria João Nicolau
Professor Doutor António Costa

Outubro de 2014

DECLARAÇÃO

António Daniel da Mota Lopes

Endereço electrónico : a.lopes90@gmail.com Telefone :927712353

Número do Bilhete de Identidade: 13776062

Título dissertação :

VoIP em Redes Peer-to-peer

Orientador : António Luís Duarte Costa
Co-orientadora:

Maria João Mesquita Rodrigues da Cunha Nicolau Pinto

Ano de conclusão: 2014

Ciclo de Estudos Integrados Conducentes ao Grau de Mestre em Engenharia de Comunicações

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, ___/___/_____

Assinatura: _____

Agradecimentos

Gostaria de agradecer aos meus orientadores, Professora Doutora Maria João Nicolau e Professor Doutor António Costa, pela dedicação, apoio e paciência durante a realização desta dissertação.

Aos meus pais e irmãos pelo apoio. Aos meus amigos que me acompanharam, apoiaram, ensinaram e fortaleceram durante o meu percurso académico, nomeadamente o Bruno Lopes, João Antunes, Luís Cunha, Nuno Correia, Nuno Pereira, Pedro Castro, Pedro Cunha e Phillipe Costa. Um especial obrigado ao Luís Cunha por todo o apoio e amizade que me proporcionou neste meu último ano académico.

Resumo

A utilização da rede IP para transportar voz (VoIP) trouxe vantagens face à rede pública de comutação de circuitos fundamentalmente porque oferece potencialmente uma qualidade superior de chamada, maior largura de banda com menores custos e melhor utilização de recursos. Contudo, a arquitetura do VoIP utiliza servidores centrais para armazenar a informação dos utilizadores e a sua respetiva localização, o que torna o sistema vulnerável a ataques de negação de serviço (DoS) e traz problemas de escala. A utilização de redes peer-to-peer pode contribuir para minimizar estes problemas, já que estas redes, pela sua natureza distribuída, permitem armazenar a informação de utilizadores e respetivas localizações em vários nós, o que torna possível que, em vez dos pedidos serem todos canalizados para um mesmo servidor, sejam distribuídos por vários servidores.

Os *peers* que constituem redes *peer-to-peer*, além de armazenarem a informação de forma distribuída, podem trazer outras vantagens, como por exemplo, o encaminhamento de dados entre nós da rede. Com vista a melhorar o desempenho, pensou-se em tirar partido do encaminhamento na rede *peer-to-peer*. Além de permitir contornar Firewalls e/ou NATs, permite que a *media* seja encaminhada por caminhos alternativos, obtendo melhor qualidade de serviço que nem sempre o caminho definido pela rede permite oferecer.

Neste trabalho foi desenvolvida uma aplicação VoIP, em JAVA, capaz de se integrar e tirar partido de uma rede peer-to-peer, totalmente baseada em SIP, que foi desenvolvida num trabalho anterior. O facto do protocolo adotado para a criação e manutenção da rede ser o SIP, facilitou a integração entre as duas componentes, uma vez que o uso do SIP é utilizado em várias operações de sinalização requeridas pelo VoIP.

Neste sentido foram propostas alterações na arquitetura, bem como um protocolo de reencaminhamento automático das chamadas pela rede peer-to-peer, com um número ajustável de saltos, de modo a mostrar melhorias em termos de desempenho global da rede. A implementação JAVA foi testada em ambiente emulado com o emulador CORE, com uma topologia e vários cenários de teste, que permitiram comprovar que as alterações propostas permitem efetivamente acomodar mais chamadas com os mesmos recursos.

Abstract

Voice over Internet Protocol (VoIP) potentially offers a several advantages compared to the PSTN (Public Switched Telephone Network) fundamentally, a higher call quality, greater bandwidth at lower cost and better infrastructure utilization. However, the architecture of VoIP uses central servers to store user information and their respective location, which makes the system vulnerable to DoS(Denial of Service) attacks and causes problems of scalability. The use of peer-to-peer networks can minimize these problems, because their distributed nature, can store the information of users and respective locations on multiple nodes, which makes possible, instead of the requests are all received by a single server, be distributed across multiple servers.

The peers in the peer-to-peer network, besides store the information, also can provide other advantages, such as the relay of voice packets between network nodes. To improve performance, we thought taking advantage of relay in peer-to-peer network. Besides allowing bypass firewalls and/or NAT, allows voice data to be forwarded for alternative paths, obtaining better quality of service when the direct path of the networt can't offer that quality.

In this master thesis we developed a JAVA VoIP application, able to integrate and take advantage of a peer-to-peer network, entirely based on SIP, which was developed in a previous work. The fact the protocol adopted for the creation and maintenance of peer-to-peer network be the SIP, facilitated the integration between the two components, since the use of SIP is used in various signaling operations required by VoIP.

In this way, we proposed architectural changes in the peer-to-peer network, and a protocol for automatic call forwarding by peer-to-peer nodes, with an adjustable number of hops, in order to show improvements in terms of global network performance. The JAVA implementation has been tested on a emulated environment with CORE emulator, with a topology and various test scenarios, which can prove the proposed alteration permit effectively accommodate more calls with the same resources.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

API	**A**pplication **P**rogramming **I**nterface
CORE	**c**ommon **O**pen **R**esearch **E**mulator
DHT	**D**istributed **h**ash **T**able
dSIP	**d**istributed **S**ession **I**nitiation **P**rptocol
IETF	**I**nternet **E**ngineering **T**ask **F**orce
IP	**I**nternet **P**rotocol
NAT	**N**etwork **A**ddress **T**ranslation
OWD	**O**ne **W**ay **D**elay
P2P	**P**eer-**T**o-**P**eer
P2PSIP	**P**eer-**T**o-**P**eer **S**ession **I**nitiation **P**rotocol
P2P	**P**ulse **C**ode **M**odulation
PSTN	**P**ublic **S**witched **T**elephone **N**etwork
PSTN	**P**ublic **S**witched **T**elephone **N**etwork
RELOAD	**RE**source **LO**cation **A**nd **D**iscovery
RTCP	**R**eal-Time **T**ransport **C**ontrol **P**rotocol
RTP	**R**eal-Time **T**ransport **P**rotocol
RTT	**R**ound **T**rip **T**ime
SIP	**S**ession **I**nitiation **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
UA	**U**ser **A**gent
UAC	**U**ser **A**gent **C**lient
UAS	**U**ser **A**gent **S**erver
UDP	**U**ser **D**atagram **P**rotocol
URI	**U**niform **R**esource **I**dentifier
VoIP	**V**oice **o**ver **I**nternet **P**rotocol

Capítulo 1

Introdução

1.1 Enquadramento

Com o aumento do número de chamadas telefónicas e devido à globalização da Internet começou-se a equacionar a sua utilização para transportar voz, como alternativa às tradicionais chamadas efetuadas pela rede pública de comutação de circuitos (PSTN). Surgiram então as comunicações que usam a rede IP para transportar voz que são designadas por comunicações VoIP (*Voice over Internet Protocol*) [1]. Esta solução oferece várias vantagens face à PSTN, nomeadamente:

- Infra-estrutura comum com a Internet, o que permite que com apenas uma infra-estrutura se tenha acesso aos dois serviços (internet e telefone);
- Qualidade de chamadas superior visto que na PSTN é permitido um único codec, o G.711[2], que possui uma codificação PCM com ritmo de 64kbps, enquanto que o VoIP permite um leque alargado de *codecs*, e estes permitem a negociação de uma comunicação com qualidade superior;
- Flexibilidade, isto é, como os pacotes VoIP são apenas outra forma de dados, os sistemas VoIP podem oferecer serviços que integram voz, vídeo, mensagens de texto e disponibilidade do utilizador;
- "Chamadas Grátis" visto que a contabilização dos dados coincide com o tráfego gasto. Se o utilizador tiver associado um contrato com tráfego ilimitado então todo o tráfego, seja ele qual for, não tem custos adicionais.

Um exemplo de aplicações VOIP atualmente existentes são o *Skype* e o *Viber*.

O VoIP pode ser desenvolvido de acordo com o paradigma "Cliente-Servidor", no entanto esta alternativa tem alguns problemas de escalabilidade. O grande número de utilizadores pode provocar congestionamento e sobrecarga, tanto na rede como nos próprios servidores o que pode conduzir à instabilidade e falta de fiabilidade do sistema. Um outro problema desta abordagem é o facto de existir um único ponto de falha: o servidor. Se este falhar, todo o sistema vai abaixo, por se tratar de um sistema centralizado. Uma melhor solução passa pela utilização de um sistema completamente distribuído para a implementação do VoIP, sendo uma das possibilidades a utilização de redes *peer-to-peer*.

As redes Peer-to-Peer (P2P) [3] são redes de overlay que podem ser estruturadas ou não estruturadas e que permitem localizar e aceder a recursos de forma distribuída e escalável, tendo por isso sido ao longo dos anos usadas para distribuição de conteúdos. Revelam-se bastante atrativas como suporte a comunicações de áudio e vídeo entre dois ou mais interlocutores. Nesse contexto a rede P2P forma-se com o intuito de fornecer simplesmente o mapeamento entre o URL do destinatário da chamada e a sua localização efetiva. O IETF (*Internet Engineering Task Force*) tem neste momento um grupo de trabalho, denominado por *Internet Engineering Task Force - P2PSIP Working Group* (IETF-P2PSIP WG) [4], a desenvolver propostas com vista à normalização das redes P2P com sinalização SIP para uso em aplicações VoIP.

O SIP (*Session Initiation Protocol*) [5] é um protocolo de sinalização standard do IETF que funciona ao nível da camada de aplicação e é bastante utilizado para estabelecer sessões entre um ou vários participantes. Em trabalhos desenvolvidos anteriormente foi avaliada a utilização do protocolo SIP para suporte ao VOIP em redes P2P. Nesses trabalhos, o protocolo SIP era usado como suporte a toda a comunicação, quer na troca de mensagens entre os nós para formar o overlay, quer no registo e na localização de terminais VoIP. Este trabalho de dissertação usa como ponto de partida esse trabalho inicial [6], fazendo-o evoluir em várias direções:

- Estudar um outro protocolo (que introduza um menor *overhead*) para criar a rede peer-to-peer;
- Estudar mecanismos que tirem o máximo partido das vantagens que o *overlay* oferece, nomeadamente o encaminhamento da *media*;
- Implementar e avaliar a utilização de uma aplicação VoIP por cima da rede peer-to-peer

1.2 Objetivos

Os objetivos desta dissertação podem ser sintetizados nos seguintes pontos:

- Estudar as atuais propostas VoIP em redes P2P;
- Estudar o protocolo que foi normalizado pelo IETF para a criação e manutenção do *overlay*;
- Desenhar uma solução de encaminhamento de dados efetuado pelos *peers* do *overlay*;
- Implementar a solução proposta;
- Implementar uma aplicação VoIP que permita testar o encaminhamento de dados;
- Obter resultados experimentais das implementações que provem a sua validade.

1.3 Estrutura da dissertação

No capitulo 1 foi feita uma introdução ao tema desta dissertação, descrevendo a motivação e principais objetivos a alcançar.

No capitulo 2 descreve-se o funcionamento geral do VoIP bem como os protocolos de sinalização mais utilizados, nomeadamente o SIP e o H.323. Além disso, também é descrito como é feita a transmissão de dados em tempo real e os protocolos envolvidos.

No capítulo 3 é apresentada a definição de rede *peer-to-peer* bem como as suas características gerais, objetivos, vantagens, estrutura e o protocolo normalizado pelo IETF, denominado por RELOAD, para a criação e manutenção de um *overlay*. Ainda neste capitulo, são apresentados vários projetos realizados e relacionados com VoIP em redes *peer-to-peer*

No capítulo 4 é feita a descrição e análise do problema, na qual é apresentado a rede *overlay* base, quais as suas funcionalidades e as alterações necessárias que foram efetuadas para suportar a sinalização das chamadas VoIP. É Também feita uma descrição do algoritmo de encaminhamento de dados concebido e implementado.

No capítulo 5 é apresentada a implementação da aplicação VoIP bem como do algoritmo de encaminhamento de dados, de forma detalhada, nas quais são apresentas as arquiteturas gerais, a forma como se interligam entre eles e com o *overlay*, e as várias

componentes desenvolvidas e quais os seus propósitos. Além disso, são apresentadas as APIs adotadas e sua devida justificação.

No capítulo 6 apresenta-se os testes e resultados experimentais realizados à aplicação VoIP desenvolvida, apresentando a ferramenta nos quais foram efetuados e descrevendo os vários ambientes de testes. Os vários cenários de teste foram desenhados de forma a comprovar as funcionalidades da aplicação desenvolvida e para obter resultados de forma a debater o seu desempenho.

Por ultimo, o capitulo 7 apresenta as conclusões desta dissertação, os objetivos cumpridos, as possíveis modificações a efetuar ao *software* desenvolvido e propostas de trabalho futuro.

Capítulo 2

VoIP

Voz sobre IP (VoIP), é um termo utilizado para caracterizar o serviço que consiste em transmitir informação de voz através do protocolo IP (*Internet Protocol*). De uma forma geral, isto significa enviar informação de voz em formato digital dentro de pacotes de dados ao invés da utilização do tradicional protocolo de comutação de circuitos utilizado há décadas pelas companhias telefónicas.

A maior vantagem da tecnologia VoIP é a possibilidade da redução dos custos de utilização dos serviços telefónicos comuns, principalmente quando as redes de dados instaladas também passam a transmitir voz.

Para se utilizar VoIP, o primeiro passo é a conversão dos sinais de voz analógicos para sinais digitais, de forma que a informação possa ser transmitido através de uma rede IP. Este processo é realizado por *codecs*, que podem ser implementados por *software* ou *hardware*. Mas estes *codecs* não realizam somente a conversão analógico-digital, eles são responsáveis também pela compressão dos sinais digitais, para que estes possam passar na rede de forma mais rápida e eficiente, mas isto também não é o bastante.

No VoIP os dados podem apresentar ecos, atrasos e *jitter* quando os pacotes não chegam na ordem correta ou sofrem atrasos. Com a combinação dos avanços obtidos na garantia de qualidade de serviço, onde a *media* tem prioridade sobre os restantes dados, e o avanço nos algoritmos de compressão, tornaram o VoIP numa realidade hoje em dia [7].

De forma a ilustrar o funcionamento geral do VoIP recorremos a um exemplo que está ilustrado na figura 2.1.

A Alice e o Bob têm de estar registados num servidor onde tem armazenada a sua localização, por exemplo, alice@atlanta.com, onde o que se segue depois do "@" é o servidor onde está a sua conta, neste caso em "atlanta.com".

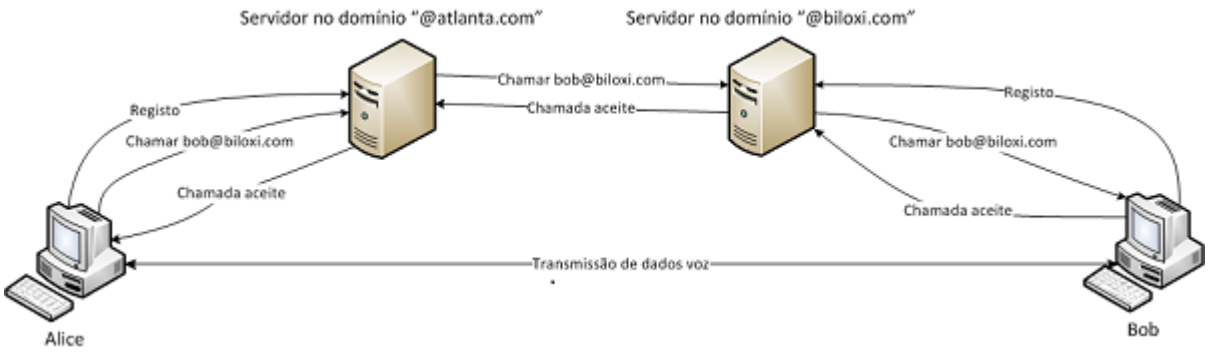


FIGURA 2.1: Exemplo geral do funcionamento do VoIP

Quando Alice e Bob querem comunicar um com o outro, devem estabelecer ligação através dos seus servidores. Então, Alice contacta o seu servidor "*atlanta.com*"e pedir para encaminhar um "pedido de chamada"para Bob. "*atlanta.com*"encaminha o pedido para "*biloxi.com*"e este reencaminha para Bob.

Quando Bob recebe o pedido de chamada manda a resposta na direção oposta através de "*biloxi.com*", "*atlanta.com*"e eventualmente para Alice e se a chamada for aceite, são estabelecidos vários parâmetros, como por exemplo, os *codecs* e o IP e porta que vão ser utilizados.

Alice indica quais os *codecs* que o equipamento suporta para a comunicação assim como Bob. Se Alice anunciar G.729 e G.711, Bob pode usar qualquer um dos *codecs* e pode comutar entre eles durante a sessão. No entanto, se Bob comunicar apenas o G.729, Alice só pode usar, única e exclusivamente esse.

Após esta negociação entre os dois utilizadores, como o uso de servidores como intermediários pode ser lento e ineficiente, é feita uma ligação direta entre eles para a transmissão de dados.

Resumindo, o tráfego de sinalização é transmitido através dos servidores, a *media* é transmitida diretamente entre os dois pontos.

2.1 Transporte dos Dados

Embora a transmissão possa ser "barata", em algumas partes do mundo bem como em muitas empresas privadas, a transmissão tem um custo de tal forma elevado que é necessário um esforço para usar a largura de banda disponível de forma eficiente. Começou-se então a usar *codecs* para comprimir os dados. Outra técnica para aumentar a eficiência da largura de banda é fazer a deteção de voz e anular o silêncio. A qualidade

de voz pode ser mantida ao usar supressão de silencio, se o *codec* recetor inserir cuidadosamente um ruído de conforto em cada período de tempo de silencio. Por exemplo, o Anexo B da recomendação ITU-T G.729 define um robusto detetor de atividade de voz que mede as mudanças, ao longo do tempo, do ruído de fundo e envia, a uma baixa taxa de transmissão, informação suficiente para o recetor gerar "ruído de conforto"que é a característica do ruído de fundo do telefone origem [1].

A codificação e formação de cada *media* resulta em atrasos maiores que aqueles obtidos em comutação de circuitos. As taxas de transmissão variam entre e 5Kbps a 64kbps. Geralmente quanto mais baixa a taxa de transmissão mais complexa é a codificação. Para o IPv4 o cabeçalho RTP/UDP/IP é 40bytes. Um *payload* de 40bytes significaria 50% de eficiência do *payload*. A 64kbps, demoraria apenas 5ms para acumular 40bytes, mas a 8kbps iria demorar 40ms para os acumular. Um atraso de 40ms na formação de cada *media* é significativo e muitos sistemas VoIP usam taxas de transmissão de 16kbps (20ms para acumular os 40 bytes), apesar da baixa eficiência quando se utilizam *codecs* de baixa taxa de transmissão. Para uma conversação continua, a largura de banda requerida (BW), em kbps, é obtida através do tamanho do cabeçalho H(em bits), a taxa de transmissão do *codec* R(em kbps) e o tempo de formação do *payload*(em milisegundos):

$$BW = R + \frac{H}{S} \tag{2.1}$$

Existem algoritmos de compressão que melhoram a eficiência do *payload*. Os 40 *bytes* do cabeçalho RTP/UDP/IP podem ser comprimidos para 2-7 bytes. Tipicamente, cabeçalhos comprimidos são 4 bytes, incluindo 2 bytes para o *checksum*. Quanto menor a largura de banda, maior o atraso na formação da *media* e maior a complexidade da codificação. [1]

Usando o mesmo exemplo da Alice e do Bob, depois de estabelecidos os parâmetros de configuração, enviar e receber dados é conceptualmente simples: consiste em dividir os dados em pequenos fragmentos e envia-los para o outro lado.

Infelizmente, o TCP (assim como outros protocolos de *stream* sequenciais) não são protocolos otimizados para a transmissão de dados em tempo real devido à sua sequenciação e confirmação obrigatória dos dados. Por exemplo, se por algum motivo, a rede perde pacotes ou se os reordena em outra sequência, a aplicação vai detetar anomalias e pedir a retransmissão de dados. Este comportamento pode criar atrasos inaceitáveis.

Na maioria dos ambientes VoIP, os dados são transmitidos em cima do protocolo UDP [8] [1], que fornece um serviço não fiável e receção desordenada dos datagramas. Os dados são

também encapsulados com o cabeçalho RTP(*Real-Time Transport Protocol*) que fornece a informação necessária para o recetor fazer uma correta reprodução dos dados recebidos. Mais importante, o cabeçalho contem o número da sequência e o tempo de ocorrência, que permite ao recetor reconstruir a *stream* na sequência correta, mesmo na ocorrência de perdas ou no caso dos datagramas chegarem desordenados [9].

Pelo facto do RTP não ser um protocolo fiável, pacotes perdidos ou corrompidos não são retransmitidos, implicando interrupções ou lacunas na *stream*. O recetor pode tratar isto detetando as falhas e encobrindo os erros, por exemplo, reproduzindo a amostra anterior novamente. No entanto, pacotes reordenados ou atrasados não criam necessariamente falhas. As implementações de reordenação de pacotes são complicadas mas a ideia básica é simples: em vez de reproduzir as amostras à medida que chegam, armazenam-se num *buffer* os pacotes recebidos. O pacote a ser reproduzido está sempre um pouco atrás do pacote recebido mais recentemente. Se um pacote está atrasado ou é reordenado, desta forma ainda existe um certo intervalo de tempo em que pode ser recebido sem causar uma falha na reprodução da *stream* pois o recetor irá encaixa-lo no seu lugar dentro do *buffer*. Infelizmente qualquer *buffer* causa latência, isto significa que as implementações têm de acordar um tamanho de *buffer* (e, portanto, a latência dos dados) conforme a possibilidade de haver pacotes reordenados ou perdidos que causem falhas na receção

2.2 Protocolos de Sinalização de Chamada

É chamada sinalização de chamada o processo envolvido no estabelecimento de chamada entre dois utilizadores. Para isso foram desenvolvidos vários protocolos, entre eles os usados no VoIP são o H.323, SIP, MGCP e Megaco/H.248. O H.323 e o SIP são protocolos de inicialização e controlo ligações fim-a-fim, enquanto que o MGCP e o Megaco são protocolos de inicialização e controlo em sistemas com a filosofia *Master-Slave*. O H.323 e Megaco são tipicamente utilizados para vídeo-conferências e chamadas telefónicas, porém são baseados num paradigma orientado à conexão. Em seguida é dada uma breve explicação do protocolo H.323 e do SIP, visto que são os que mais se identificam no seio desta dissertação.

2.2.1 Protocolo H.323

Sendo o protocolo H.323 pioneiro no VoIP [10], tem o objetivo de especificar sistemas VoIP e que não garantem Qualidade de Serviço (QoS). Além disso, estabelece padrões para codificação e descodificação de fluxos da *media* e vídeo, garantindo que produtos

baseados no padrão H.323 de um fabricante é compatível com produtos H.323 de outros fabricantes.

Este protocolo envolve várias entidades nas quais se destacam os terminais, os *gateways* e os *gatekeepers* e que podem ser integradas em computadores pessoais, *routers* ou dispositivos *stand-alone*. A função de um *gateway* é permitir a conversão(digital-analógico e vice versa) e *transcoding* (conversão entre formatos) dos dados entre um dispositivo H.323 e um não H.323, como por exemplo, conversão, transmissão e procedimentos de sinalização entre um telefone ligado à rede de comutação de circuitos(PSTN) e uma rede IP. Além disso, faz também compressão dos dados e é capaz de gerar e detetar sinais DTMF (*Dual Tone Multiple Frequency*).

A entidade denominada como terminal divide-se em três universos:

- Uma entidade responsável pela sinalização e que faz o controlo da chamada através dos protocolos H.225.0 e do H.245;
- Uma camada H.225.0 que formata o áudio transmitido e coloca as *streams* em mensagens, recupera os fluxos de áudio das mensagens que tenham sido recebidas da interface de rede e executa a ordenação, sequenciação, deteção e correção de erros se apropriado;
- Um *transcoder* e também um compressor de dados.

A entidade *gatekeeper* tem como caracteriza-se por controlar a admissão de chamadas e por efetuar a tradução de endereços. Então, vários *gatekeepers* podem-se comunicar uns com os outros para coordenar os seus serviços de controlo e além disso, as redes com *gateways* devem (mas não é obrigatório) ter *gatekeepers* para traduzir endereços E.164(tradicional número de telefone) em endereços de transporte, isto é, endereço IP e número de porta. Os *gatekeeper* são logicamente separados de todas as outras entidades, mas fisicamente, podem coexistir com um terminal, um *gateway* ou um *proxy*. Quando está presente numa rede VoIP, o *gatekeeper* fornece as seguintes funções:

- Tradução de endereços: o *gatekeeper* traduz números de telefone em endereços IP, através de tabelas de tradução que são atualizadas usando, por exemplo, mensagens de registo.
- Controlo de admissão: autoriza o acesso à rede usando mensagens H.225. O critério de admissão pode incluir autorização da chamada, largura de banda ou outras políticas

- Controlo de largura de banda: controla a quantidade da largura de banda que o terminal pode usar
- Gestão de zonas: um terminal pode apenas registar-se num *gatekeeper* de cada vez

Todas as funções descritas acima são fornecidas aos terminais e *gateways* registados. Um *gatekeeper* pode ou não participar na sinalização de controlo da chamada.

O H.323 tem ainda dois canais dedicados que são:

- *Registo, Admissão e estado do canal:* o canal RAS(*Registration, Admission and Status*) transporta mensagens no processo de registo de um terminal num *gatekeeper* que associa o numero telefónico num IP e porta que vai ser usado para sinalização de chamadas. O canal RAS é também usado para transmissão e admissão, alteração de largura de banda, estado e troca de mensagens entre os terminais e o *gatekeeper*. H.225.0 recomenda *timeouts* e enviar novamente as mensagens RAS, uma vez que estas são transmitidas pelo protocolo UDP que é um protocolo não fiável;
- *Canal de sinalização de chamada*: transporta mensagens de controlo H.225.0 usando o TCP, tornando-o num canal fiável. Os terminais e *gatekeepers* usam mensagens Q.931 (com TCP) para sinalização da chamada. No caso em que as redes não têm *gatekeepers* os terminais enviam a sinalização da chamada diretamente para o terminal pretendido, usando o "*Call Signaling Address*", senão a primeira mensagem de admissão é enviada, através do canal RAS, para o *gatekeeper* no qual este lhe indica se deve mandar as mensagens de sinalização diretamente para o outro terminal ou se devem ser encaminhadas por ele.

Na figura 2.2 está ilustrada o estabelecimento de uma chamada entre dois terminais onde não existe nenhum *gatekeeper* intermédio.

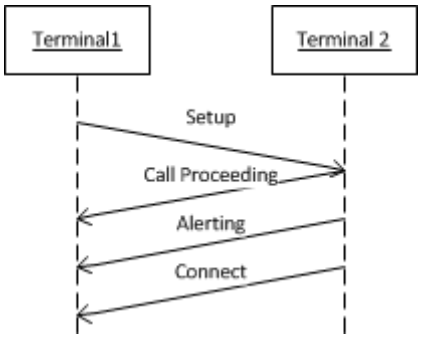


FIGURA 2.2: Estabelecimento de uma chamada sem *Gatekeeper*

O terminal 1 envia uma mensagem de *setup* para o terminal 2 no qual responde com uma mensagem de *Call Proceeding*, *Alerting* e por fim com a mensagem *connect* que contém o endereço para o canal no qual se vai efetuar o controlo da chamada.

Na figura 2.3 está ilustrado o caso em que existe um *gatekeeper* na sinalização da chamada.

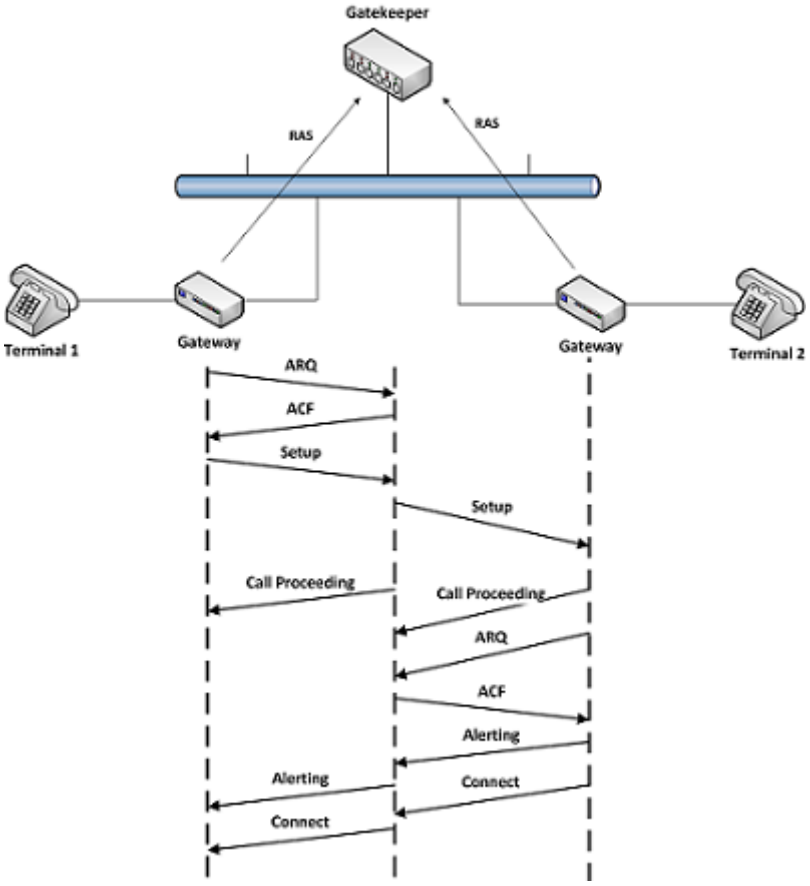


FIGURA 2.3: Estabelecimento de uma chamada com *Gatekeeper*

O primeiro passo que o *gateway* conectado ao terminal 1 faz é enviar ao *gatekeeper* uma mensagem de admissão (ARQ) no qual responde com uma confirmação (ACF). Depois, tal como no exemplo anterior, é enviada uma mensagem de *setup* que é encaminhada para o *gateway* em que o terminal 2 está ligado. Nesse *gateway* é também feito o processo de admissão e posteriormente, o *gateway* responde com um *Alerting* seguidamente com o *Connect*.

Na figura 2.4 encontra-se uma figura com a arquitetura geral deste protocolo e a forma como as diferentes entidades se ligam entre elas.

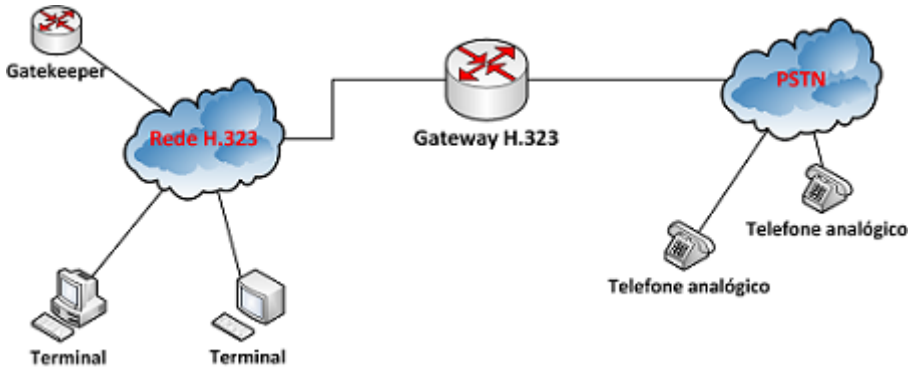


FIGURA 2.4: Arquitetura geral H.323

2.2.2 Protocolo SIP

O *Session Initiation Protocol* (SIP) é um protocolo de sinalização da camada de aplicação, normalizado pelo IETF. É utilizado para estabelecer, modificar ou terminar sessões entre um ou vários participantes [5]. Atualmente é um dos protocolos mais utilizados para a implementação de serviços de voz sobre IP (VoIP) na Internet e também para o estabelecimento de conferências multimédia. O SIP não especifica quais os tipos de sessões para os quais pode ser utilizado, sendo apenas responsável pela sua gestão. Para tal, o SIP oferece funcionalidades que permitem o registo, autenticação e localização de utilizadores. Permite também que os terminais de uma sessão negoceiem entre si os tipos de dados multimédia a utilizar, assim como permite a alteração de parâmetros de uma sessão já estabelecida enquanto esta decorre.

O protocolo foi desenvolvido utilizando um modelo de pedido e resposta idêntico ao utilizado no protocolo HTTP, onde cada mensagem enviada invoca uma determinada ação do lado do servidor, gerando este pelo menos uma mensagem de resposta.

Componentes Principais

É designado por *User Agent*(UA) SIP qualquer dispositivo terminal que execute uma aplicação baseada em SIP, sendo que um UA pode ser um cliente ou um servidor. O cliente, *User Agent Client* (UAC) gera pedidos, para por exemplo estabelecer uma sessão com um outro UA, enquanto que o servidor *User Agent Server* (UAS) recebe e responde a pedidos. O UAC para além de gerar pedidos, é também responsável por processar as mensagens de resposta ao pedido efetuado.

Para além do UA, o protocolo SIP define outros componentes importantes, tais como:

- Servidor *Proxy* - É um servidor intermediário, que tem como tarefa principal encaminhar as mensagens para o destino, ou para um outro servidor mais próximo

do destinatário da mensagem. Para além do encaminhamento, um *Proxy* pode ser utilizado para aplicar políticas, como por exemplo, verificar se o utilizador pode efetuar a chamada.

- Servidor de Redirecionamento - É um servidor (UAS) que recebe pedidos para a localização de um UA, recorrendo a mensagens de redirecionamento, com URI's alternativos que devem ser contactados para a localização do UA.
- Servidor de Registo - É um servidor responsável por aceitar mensagens de registo, do tipo REGISTER, guardando a informação que recebe no serviço de localização do domínio a que pertence.
- Servidor de Localização - O servidor de localização contêm uma lista na qual aos endereços (AoR) são associados zero ou mais endereços de contacto. Este servidor é utilizado pelos servidores de redirecionamento ou *proxies* para obter informação sobre a localização dos UA.
- *Back-to-back User Agent* (B2BUA) - *Um back-to-back user agent* (B2BUA) é uma entidade lógica que atua como um cliente (UAC) e como um servidor (UAS) em simultâneo. Ao receber pedidos, este atua como um servidor (UAS) de modo a processar o pedido recebido assim como para gerar a mensagem de resposta. Atua como cliente (UAC) quando necessita de gerar pedidos. Contrariamente a um servidor *proxy*, um B2BUA participa ativamente na troca de mensagens dos diálogos que estabeleceu.

Os componentes descritos anteriormente são componentes lógicos, sendo possível que uma aplicação combine um ou vários destes componentes. É habitual combinar os servidores de localização, de registo e de *proxy* numa única aplicação SIP.

A figura 2.5 representa um exemplo que ilustra as entidades SIP envolvidas no registo de um utilizador e na sua localização.

Neste exemplo, o utilizador carol envia um pedido de registo para o servidor de registo do seu domínio (chicago.com), informando-o que se encontra disponível para contacto no endereço cube2214a.chicago.com. O servidor de registo valida o pedido e armazena a informação no servidor de localização. De seguida, o utilizador bob decide contactar o utilizador carol. Para tal, e uma vez que o bob não possuí informação sobre o endereço no qual o utilizador carol está localizado, conhecendo apenas o seu endereço SIP (carol@ chicago.com), necessita de enviar a sua mensagem do tipo *INVITE* para o servidor *proxy* do domínio do utilizador carol. O servidor *proxy* por sua vez, contacta o servidor de localização do seu domínio, de modo a obter um endereço no qual o destinatário pode ser contactado. Neste caso, recebe como resposta o endereço de contacto que o utilizador

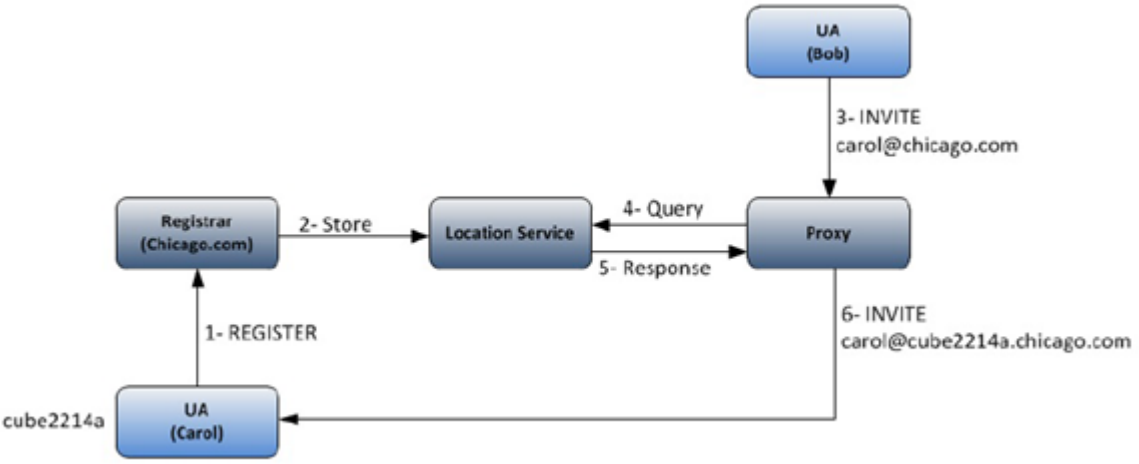


FIGURA 2.5: Exemplo de registo e localização de um UA

carol tinha registado anteriormente quando efetuou o seu registo. Uma vez possuindo um endereço no qual o destinatário pode ser contactado, o servidor de *proxy*, reenvia a mensagem do tipo INVITE para o endereço de contacto do destinatário.

As mensagens SIP são complicadas mas a informação mais importante contida no INVITE é:

- O URI do destinatário (carol@chicago.com)
- O *call-ID* que identifica a chamada
- Os parâmetros de configuração da chamada. Por exemplo, Bob pode indicar que quer estabelecer a chamada apenas com um canal de áudio. Esta troca de parâmetros é efetuada através do protocolo SDP (*Session Description Protocol*) [5]. O SDP contém o endereço IP no qual Carol vai estar a receber os dados, como por exemplo, 192.0.0.2 na porta 31152 e os *codecs* que irá usar (G.729 ou G.711).

Quando um deles desligar o telefone é enviada uma mensagem *BYE* para o servidor indicar o fim de conexão.

Eventos SIP

Existem vários tipos de eventos que são referentes ao pedido efetuado pelo utilizador. As mensagens referentes a pedidos, diferenciam-se pelo facto de que a sua primeira linha contêm informação relativamente ao pedido a efetuar, estando definidos na especificação do protocolo os seguintes tipos de pedidos:

- REGISTER - Este tipo de pedido é utilizado para um utilizador registar informação relativamente à forma como pode ser contactado.
- INVITE, ACK e CANCEL - Estes tipos de pedidos são utilizados para estabelecer sessões entre utilizadores SIP.
- BYE - Utilizado para terminar sessões SIP.
- OPTIONS - Serve para obter informações sobre as funcionalidades disponíveis de um servidor

Como resposta a esses eventos foram especificados seis tipos de classes, onde cada uma delas representa um conjunto de respostas do mesmo tipo. O primeiro digito representa a classe associada ao código, então as classes são:

- Provisório (1xx) - O pedido foi recebido, mas ainda não há uma resposta;
- Sucesso (2xx) - O pedido enviado foi recebido, processado e aceite;
- Redirecionamento (3xx) - O pedido tem de ser redirecionado;
- Erro no Cliente (4xx) - O servidor não consegue processar a mensagem, esta pode ter erros sintáticos ou o servidor não pode satisfazer os requisitos;
- Erro no Servidor (5xx) - O servidor não consegue processar a mensagem, apesar de não ter detetado qualquer erro na mensagem;
- Falha Geral (6xx) - O pedido não pode ser satisfeito em nenhum servidor.

Capítulo 3

Redes Peer-To-Peer

As redes P2P caracterizam-se por serem distribuídas. Os nós (*peers*) que a constituem formam uma rede sobreposta designada de rede *overlay* (ou apenas *overlay*) na qual todos eles devem cooperar entre si. Os peers do overlay devem disponibilizar parte dos seus recursos, como poder de processamento e/ou armazenamento, de forma a que em conjunto possam fornecer um determinado serviço. Cada peer desempenha funções de cliente e de servidor, contrariamente ao modelo tradicional cliente-servidor em que os servidores são utilizados para disponibilizarem serviços e onde os clientes atuam apenas como consumidores, atuando apenas para seu próprio benefício. Um peer de uma rede p2p deve atuar das duas formas: como cliente quando pretende usufruir de uma funcionalidade oferecida pelo overlay, e também como servidor, quando necessita de suportar operações para o benefício da rede, por exemplo, no encaminhamento de mensagens entre os nós, ou no armazenamento de informação do *overlay*.

Apesar das funcionalidades que uma rede P2P deve implementar dependerem do tipo de serviço para o qual a rede será utilizada, existe pelo menos uma funcionalidade ou mecanismo, comum a qualquer tipo de serviço, que é a localização de nós da rede (*peer discovery*). É necessário que exista um mecanismo para a localização de nós na rede, de modo a que um novo peer possa localizar um ou vários peers pertencentes à rede para que ele se possa juntar à rede P2P. Existem diversos mecanismos para a localização de peers, sendo que habitualmente são utilizados mecanismos centralizados. Uma das técnicas mais utilizadas recorre a servidores de arranque denominados de *bootstrap servers* cuja função é disponibilizar um conjunto de endereços IP pertencentes a nós do *overlay* que podem estar disponíveis para receber o novo nó que deseja juntar-se ao *overlay*.

Para além da localização de *peers* na rede, uma rede P2P implementa algumas das seguintes funcionalidades [11]:

- Indexação de dados - os recursos de dados armazenados no *overlay*, devem ser indexados de alguma forma, devendo existir um mecanismo responsável pela indexação dos recursos de dados da rede;
- Armazenamento de dados - funções de armazenamento, assim como de obtenção de dados guardados no *overlay*;
- Processamentos de dados - dependendo do tipo de serviço os *peers* do *overlay* podem colaborar no processamento de dados;
- Encaminhamento de Mensagens - responsável pelo encaminhamento de mensagens entre os nós da rede.

Apesar de uma rede P2P se caracterizar pela sua natureza distribuída, algumas das suas funcionalidades podem ser implementadas recorrendo a soluções centralizadas, como por exemplo a indexação dos conteúdos de dados. Contudo, a utilização de servidores centralizados pode trazer alguns problemas, por exemplo problemas de escala e tolerância a falhas, o que dependendo do tipo de aplicação que se pretende poderá ser um problema.

Uma rede P2P pode ser classificada pela forma como os dados armazenados são indexados [3], podendo estes ser indexados de uma forma centralizada, localmente, ou de uma forma distribuída. Numa estratégia de indexação centralizada, um servidor central possui referências para todos os conteúdos de dados disponibilizados por todos os nós do *overlay*. Com uma estratégia de indexação local, cada peer referencia apenas os seus conteúdos de dados, não possuindo qualquer conhecimento sobre os conteúdos de outros *peers*. Esta estratégia é utilizada por diversos protocolos P2P.Numa estratégia distribuída, as referências para os conteúdos de dados estão distribuídas em vários *peers* do *overlay*. Exemplos de protocolos P2P que utilizam esta estratégia são os protocolos baseados em DHTs (*Distributed Hash Table*.

A forma como os dados são indexados e o modo como os *peers* são posicionados no *overlay*, leva a que as redes de *overlay* sejam classificadas de duas formas distintas, não estruturadas e estruturadas [3] [11].

3.1 *Overlays* não estruturados

Neste tipo de rede os *peers* são posicionados de uma forma aleatória no *overlay*, formando uma topologia que habitualmente tem um ou dois níveis hierárquicos [11]. Numa topologia plana, sem hierarquia, os *peers* que formam o *overlay* estabelecem ligações com outros *peers* de uma forma aleatória, tendo todos eles igual importância na rede e desempenhando as mesmas funções. Numa topologia com uma hierarquia de dois níveis, existe

uma distinção entre os *peers* do *overlay*, existindo dois tipos de *peers*, denominados de *super-peers* e *peers*.

Um *super-peer* é eleito pela rede, normalmente pelo facto de ter mais recursos e/ou por estar mais tempo disponível comparativamente com a maioria dos restantes *peers*, podendo obviamente o processo de eleição derivar de outros parâmetros que não os aqui referidos. Neste tipo de *overlay* hierárquico com *super-peers*, a topologia da rede é dividida em dois níveis. Um nível no qual os *peers* possuem ligações apenas com *super-peers*, estabelecendo uma ou mais ligações, e um outro nível no qual os *super-peers* estabelecem ligações entre si, formando um *overlay* no qual só os *super-peers* participam.

Uma vez que pelas suas características os *super-peers* possuírem uma maior importância na rede, são responsáveis pelo encaminhamento de mensagens no overlay, atuando como *proxies* em favor dos *peers* normais, devem encaminhar para outros *super-peers* as mensagens enviadas pelos *peers* normais. Para além do encaminhamento de mensagens, os *super-peers* mantêm um índice no qual estão referenciados os recursos de dados partilhados pelos *peers* com os quais possuem uma ligação.

Numa rede P2P não estruturada, o mecanismo utilizado para a localização de recursos na rede, consiste habitualmente na utilização de técnicas de inundação (*flooding*) [3] [11]. De acordo com estas técnicas a rede é inundada com mensagens para a localização do recurso pretendido (*queries*), existindo um parâmetro (TTL(*Time To Live*)) que especifica o número máximo de saltos que uma mensagem pode dar. Desta forma limita-se a propagação excessiva de mensagens na rede. Este mecanismo de localização de recursos tem alguns problemas devido à grande quantidade de tráfego que pode ser gerado. Além disso, não permite garantir que um recurso que exista na rede seja encontrado, pois o recurso pode estar num *peer* ao qual as mensagens com a *query* não chegam devido ao número de saltos ter atingido o valor máximo. O *flooding* é um mecanismo considerado eficiente para a localização de recursos populares na rede, estando mais sujeito a falhar quando a popularidade do recurso pretendida é baixa. Este mecanismo quando utilizado numa topologia composta por *peers* e *super-peers*, é normalmente utilizado apenas no *overlay* formado pelos *super-peers*, uma vez que cada *super-peer* possui conhecimento sobre os recursos partilhados por cada um dos *peers* aos quais está ligado, o que faz com que não haja necessidade de os *peers* normais participarem neste processo.

3.2 *Overlays* estruturados

Uma rede P2P estruturada caracteriza-se pelo facto de a sua topologia ser bem definida, na qual os *peers* são posicionados de uma forma controlada num anel lógico, e os recursos

são posicionados de uma forma determinística no *overlay* tornando mais eficiente a sua localização.

Este tipo de rede recorre a mecanismos baseados em DHTs para o posicionamento dos recursos no *overlay*. Numa DHT são atribuídos identificadores únicos aos *peers* da rede (NodeIDs), identificadores esses que podem ser utilizados para o *overlay* decidir onde o *peer* deverá ficar posicionado, assim como os *peers* com os quais este deve estabelecer ligações.

A figura 3.1 mostra uma topologia em anel, representando uma DHT, na qual é possível verificar os identificadores de cada *peer*, e os identificadores de recursos que cada peer é responsável por gerir.

Os recursos de dados são também identificados através de identificadores únicos denominados por chaves (*keys*), sendo que esses identificadores podem ser obtidos através de técnicas de *hashing*. Através do identificador de um recurso, o algoritmo DHT determina qual o *peer* que deve ficar responsável pelo seu armazenamento, escolhendo o *peer* com o identificador mais próximo do identificador do recurso a armazenar. De modo a tornar o sistema tolerante a falhas, alguns protocolos DHT utilizam técnicas de replicação de conteúdos nas quais os recursos são armazenados nos N peers com *NodeID* mais próximo da chave do recurso a armazenar.

Para efetuar a localização de um determinado recurso na rede, cada *peer* possui uma tabela com os endereços IP e os respetivos identificadores (*NodeID*) de alguns peers vizinhos, devendo enviar uma mensagem de *query* para o *peer* da sua tabela que possui o identificador mais próximo da chave do recurso a localizar. Este processo é repetido até que o *peer* responsável pelo recurso pretendido é localizado. Em teoria, a localização de recursos em DHT requer em média O(log N) saltos, onde N é o número de peers do overlay P2P.

3.3 Trabalhos em curso no IETF

O grupo de trabalho P2PSIP [4] foi criado pelo IETF com o objetivo de padronizar os protocolos necessários para se criar redes *peer-to-peer*. Esta necessidade de criar um protocolo padrão pretende resolver as incompatibilidades atualmente existentes entre protocolos utilizados por diferentes fabricantes. Existem vários *drafts* ativos, nomeadamente o protocolo RELOAD, propostas especificas para a descoberta de recursos e serviços na rede, a DHT utilizada, e várias extensões ao protocolo RELOAD que lhe permitem funções adicionais. Recentemente, em janeiro de 2014, o *draft* do RELOAD passou a RFC (RFC 6940) [12].

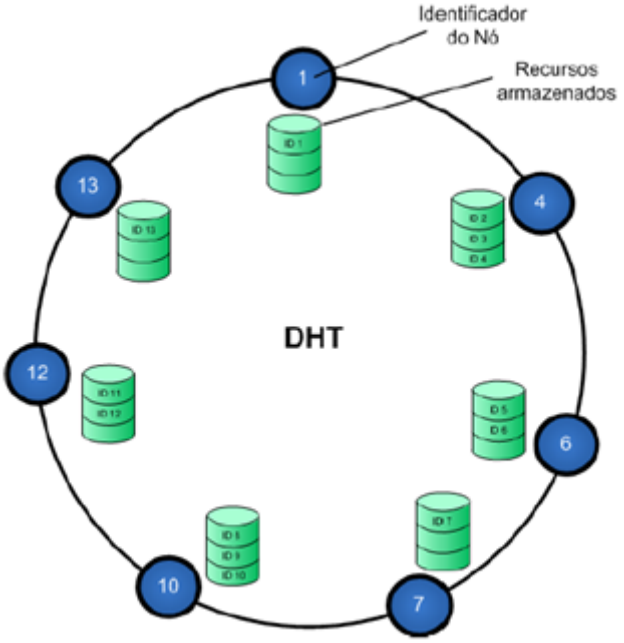


FIGURA 3.1: Exemplo de rede de *overlay* estruturado

Protocolo RELOAD

O *Resource LOcation And Discovery* (RELOAD) é a proposta do IETF para a criação de um protocolo de sinalização flexível para ser utilizado com outros protocolos, desde que sejam semelhantes ao SIP, como por exemplo, o P2PSIP. O grupo de trabalho responsável por esta especificação é composto por alguns dos autores de propostas P2PSIP, como por exemplo o dSIP [13] e o P2PP [14]. Contrariamente a outras propostas, como o dSIP, este protocolo não utiliza mensagens SIP na gestão do *overlay*, utilizando um novo protocolo. O protocolo utilizado na gestão do *overlay*, é um protocolo binário, desenvolvido para ser mais leve, permitindo melhorar o desempenho global, uma vez que o tamanho das mensagens é reduzido quando comparado com mensagens de texto como as SIP, o que reduz o tráfego no *overlay* [15].

O RELOAD garante as seguintes funcionalidades, consideradas essenciais para um protocolo peer-to-peer na Internet:

- Segurança: como a rede *peer-to-peer* pode ser formada por *peers* que originalmente não possuem nenhuma relação de confiança, o RELOAD define a existência de um servidor central para providenciar as credenciais necessárias para cada *peer*, assim cada operação que for feita na rede poderá usar mecanismos de segurança, como autenticação e outros;
- Flexibilidade: o RELOAD foi desenhado para suporta várias aplicações diferentes, principalmente comunicações multimédia que utilizam o SIP. A flexibilidade advém

de ser possível que cada aplicação crie seus próprios tipos de dados e regras para o seu uso;

- NAT: o RELOAD está preparado para funcionar em ambientes onde muitos, ou mesmo todos, os nós estão por trás de *firewalls* a usar NAT. O protocolo inclui diversos mecanismos de NAT-T, nomeadamente o ICE(*Interactive Connectivity Establishment*) [16], o TURN(*Transversal Using Relays around NAT*) [17] e o STUN(*Session Transversal Utilities for NAT*) [18].

- Desempenho do Encaminhamento: o RELOAD foi definido com a preocupação em otimizar o encaminhamento, pois este é um ponto importante numa rede p2p, já que podem ser introduzidos atrasos de encaminhamento em cada nó, quer devido à largura de banda disponível, quer pela capacidade de processamento do *hardware*. Nesse sentido, as mensagens foram definidas com um cabeçalho simples de modo a minimizar os recursos necessários para o seu encaminhamento.

- Algoritmo de Overlay: o RELOAD define uma interface abstrata para permitir que seja simples adaptar outros algoritmos de encaminhamento no *overlay*, seja estruturada (DHT) ou não estruturada. Além disso, a implementação do Chord [19] é obrigatória, a fim de garantir a interoperabilidade.

- Suporte para Clientes: os clientes RELOAD diferem de *peers* RELOAD pelo facto que os clientes não armazenam informação nem participam ativamente para no *overay*, isto é, apenas tiram proveito das suas potencialidades, através dos *peers*, para localizar utilizadores, recursos, armazenar informação e para contactar outros *peers* e/ou clientes.

Arquitectura

A arquitetura do protocolo RELOAD divide-se em três camadas distintas, camada de aplicação, camada peer-to-peer ou camada RELOAD, e a camada de transporte .

A camada RELOAD permite a utilização de diferentes protocolos ao nível da aplicação, e é responsável pela criação e gestão do *overlay*, permitindo também que o *overlay* possa utilizar qualquer algoritmo DHT. Contudo, a implementação do algoritmo Chord é obrigatória, pois este é o algoritmo DHT utilizado por omissão. Esta camada é composta pelos seguintes componentes, tal como ilustrada na figura 3.2:

- Message Transport - responsável pelo envio e recepção de mensagens.

- Storage - responsável pelo armazenamento de informação no overlay.

- Topology Plugin - responsável por implementar o algoritmo DHT a utilizar no overlay.
- Forwarding and Link ranagement - Armazena e implementa a tabela de encaminhamento, fornecendo serviços de encaminhamento de pacotes entre os nós.

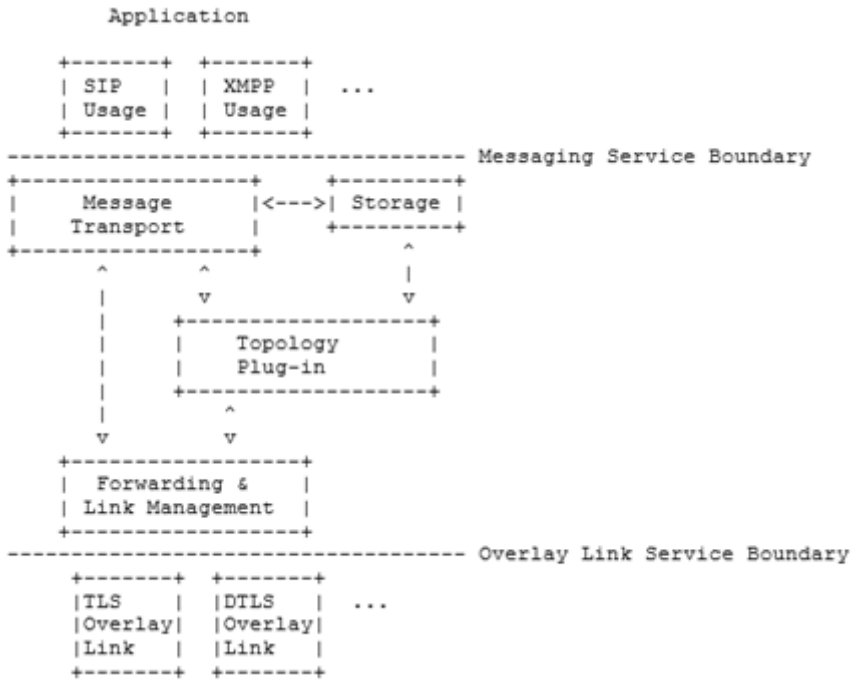


FIGURA 3.2: Arquitectura do Protocolo RELOAD [12]

De acordo com a arquitetura representada na figura 3.2, uma aplicação VoIP, que utilize SIP pode utilizar o RELOAD para localizar outros *peers* no *overlay*, podendo também (opcionalmente) utilizar o *overlay* para estabelecer ligações entre os *peers* que se encontram protegidos por *firewalls* ou NATs, tirando partido dos mecanismos que o RELOAD implementa para ultrapassar esses problemas, nomeadamente o ICE, STUN e TURN, já referidos.

Segurança

Um protocolo peer-to-peer deve ter uma preocupação grande com a segurança para poder ter uma utilização mais alargada. Este é um dos aspetos positivos do RELOAD. Ele define a existência de um certificado para cada nó, que permite ao *peer* criar uma relação de confiança com os demais *peers* da rede [12]. São dois os modelos de geração de certificados utilizados. O primeiro é baseado num processo centralizado, que reserva um identificador único para cada nó, associando-lhe um par de chaves (publica e privada) e o respetivo certificado de chave publica. Neste caso, é necessário haver um servidor

central, que seja confiável e possa garantir a autenticidade dos certificados. A segunda maneira, indicada para redes fechadas onde a segurança não é uma questão principal, é baseada em certificados auto-assinados. Neste modo, o próprio nó gera o seu *Node-ID*, nome de utilizador, as chaves e o seu próprio certificado. O certificado também é utilizado para verificar autenticidade das mensagens que passam no *overlay*. Além de certificados, o RELOAD permite que o controlo de acesso seja feito com chave pré-partilhada utilizando o TLS-PSK (*Transport Layer Security – Pre-Shared Key*) [20] e TLS-SRP (*Transport Layer Security – Secure Remote Password*) [21].

O RELOAD garante a segurança nas comunicações em três níveis:

- Conexão: é garantido pelo uso do TLS (*Transport Layer Security*) ou DTLS (*Datagram Transport Layer Security*);
- Mensagem: toda mensagens devem ser assinadas digitalmente;
- Objeto: todos os objetos armazenados devem ser assinados.

Com esses níveis de segurança, o RELOAD permite aos peers verificarem a origem e integridade dos dados recebidos. A confidencialidade das informações não é considerada requisito.

Tipos de nós

Além dos *peers*, que participam ativamente no *overlay*, o RELOAD, define nós "*clientes*", que se conectam a rede *peer-to-peer* através de um *peer* e não participam do encaminhando das mensagens pela rede, nem fazem armazenamento de dados [22]. Estes nós clientes têm as mesmas responsabilidades quanto à segurança que os *peers*, necessitando também de certificados para fazerem parte da rede. Para facilitar a implementação, o protocolo que o cliente utiliza para se comunicar com o *peer* é o mesmo utilizado pelos peers para a comunicação no *overlay*. Existem alguns motivos que podem impedir um cliente de atuar na rede como *peer*, tais como:

- O cliente não possui conectividade suficiente, por exemplo, por estar atrás de um NAT ou não ter largura de banda suficiente.
- O cliente não possui recursos suficientes, como processamento, espaço de armazenamento ou até bateria.
- Pode haver algum outro critério específico do algoritmo utilizado, como por exemplo, limite para o número de peers.

Localização e Junção ao Overlay

Antes de um *peer* se juntar a uma rede P2PSIP, primeiramente deve obter um *Node-ID*, informação de configuração e opcionalmente um conjunto de credenciais. O *Node-ID* é um identificador único que identificará o *peer* dentro do *overlay*, enquanto que as credenciais servem para dar autorização ao *peer* para se juntar ao *overlay*.

O protocolo RELOAD especifica o formato da informação de configuração e como esta pode ser obtida, assim como o *Node-ID* e as credenciais e como esta pode ser obtida através de um servidor de registo *offline*.

Uma vez obtida a informação de configuração, o RELOAD especifica os mecanismos que permitem ao *peer* obter o endereço de "*multicast-bootstrap*"no ficheiro de configuração e localizar o "*bootstrap peer*"mandando uma mensagem em *multicast* para aquele endereço de grupo. Adicionalmente, o *peer* pode ter armazenado *peers* com quem já comunicou anteriormente e usar esses como "*bootstrap peers*", ou pode ainda obter o endereço do "*bootstrap peer*"por qualquer outro mecanismo.

A função do "*bootstrap peer*"é simplesmente encaminhar o *peer* que se quer juntar ao *overlay* para um *peer*, denominado por "*admitting peer*", que o vai ajudar a juntar-se à rede. O "*admitting peer*"pode ser o vizinho do novo *peer* ou o próprio "*bootstrap peer*"de modo a estabelecer as ligações apropriadas que o tornem completamente funcional dentro da rede. Os detalhes de como este processo decorre depende da DHT usada no *overlay*.

Nas várias etapas deste processo, podem ser pedidas ao novo *peer* as suas credenciais para verificar a sua autorização de junção ao *overlay*. Similarmente, os vários *peers* contactados podem ter de apresentar as suas credencias para que o novo *peer* possa verificar que se está a juntar ao *overlay* correto [15].

Estabelecimento de Sessões Multimédia

Cada *peer* possui uma base de dados interna que tem como função armazenar recursos, que lhe são atribuídas pelo algoritmo DHT, dos contactos dos utilizadores ou qualquer outro serviço. Então, tendo como exemplo a figura 3.3, supondo que o utilizador com *username* 'A' acabou de se juntar ao *overlay*, o algoritmo DHT definiu que o respetivo *Resource Record* (username do utilizador e respetivo identificador) deve ser armazenado no *peer X*. Para isso é usada uma mensagem do RELOAD denominada por *store* na qual transporta o *Resource Record* até ao *peer* destino. Quando o utilizador "B"deseja contactar "A", vai procurar no *overlay* o *Resource Record* de A usando a mensagem RELOAD denominada por *fetch*. O *peer* que contem o *Resource Record* do utilizador "A"responde utilizando uma mensagem denominada por *fetch-resp* com a respetiva informação solicitada. No *peer* que se encontra o utilizador "B", após obter o *Resource*

Record do destinatário, faz a sinalização da chamada diretamente para o *peer* no qual se encontra o utilizador B, através de mensagens SIP [23]. As mensagens entre os *peers* A, B e X podem viajar através de *peers* intermédios ou até mesmo através de NAT.

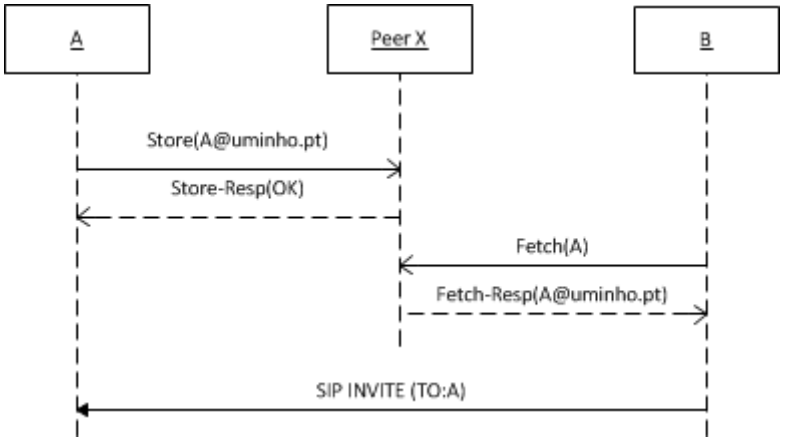


FIGURA 3.3: Exemplo de Registo e Procura de um Utilizador

Resumindo, O RELOAD é usado unicamente para descoberta de *peers*, após isso as mensagens trocadas entre os *peers* são enviadas usando o protocolo SIP.

Estrutura das Mensagens RELOAD As mensagens RELOAD estão estruturadas em 3 campos: "*Forwarding header*, obrigatório em todas as mensagens, "*Message Content*"e "*Security block*". Em cada mensagem, difere o conteúdo e o propósito, mas as partes que as constituem são sempre estas. A estrutura foi desenvolvida para satisfazer os requisitos de um protocolo do tipo pergunta/resposta. Destina-se a ser extensível, utilizando campos tais como o comprimento, o tipo e o valor para facilitar o processo de adição de novos campos. A interoperabilidade é também um fator chave, sendo por isso obrigatório a implementação de algumas partes da estrutura. A figura 3.4 ilustra as três partes da estrutura já mencionada.

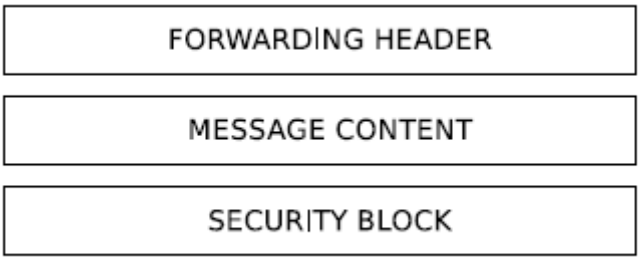


FIGURA 3.4: Estrutura das Mensagens RELOAD

O "*Forwarding Header*"é usado para o encaminhamento das mensagens dentro do *overlay* e também permite que os *peers* as identifiquem como mensagens RELOAD. O "*Message Content*"é indiferente na perspetiva da camada de encaminhamento mas é interpretado por camadas superiores. No "*Security Block*"é onde se podem encontrar os certificados e as assinaturas nas mensagens. As assinaturas também são incluídas em algumas mensagens usadas para armazenar informação. Podem ser processadas sem fazer o *parsing* do conteúdo das mensagens.

3.4 VoIP em redes P2P

Seguidamente serão apresentados três trabalhos desenvolvidos relacionados ao tema VoIP em Redes P2P. O primeiro trata-se de uma implementação de um *software* VoIP e testado em *Bare* PCs e PCs comuns em ligações ponto-a-ponto. O segundo trata-se de comunicação VoIP entre dispositivos móveis dentro de uma rede *peer-to-peer* local. Por último é apresentado uma proposta de um algoritmo para encontrar *peers* intermediários pertencentes a um *pverlay*, capazes de efetuar encaminhamento de chamadas VoIP.

3.4.1 A *peer-to-peer Bare* PC VoIP *application*

Em [24] os autores apresentam os *Bare* PC num ambiente ponto-a-ponto. *Bare* PC são dispositivos em que não existe qualquer Sistema Operativo e as aplicações operam diretamente com o hardware. Estes são de particular interesse para a comunicação segura, fiável e eficiente comunicação ponto-a-ponto. Neste artigo os autores começam por apresentar a arquitetura dos Bare PC e as suas vantagens em utiliza-los em comunicações p2p. Em seguida apresentam uma aplicação VoIP, e presentam os resultados obtidos nas várias experiências realizadas, incluindo perda de pacotes, atraso, *jitter*, e MOS (*Mean Opinion Score*).

Na computação em Bare PCs, as aplicações interagem diretamente com o hardware, evitando o Sistema Operativo como *middleware*. A sua arquitetura está ilustrada na figura 3.5. As convencionais camadas de computação são mapeadas num *Application Object* e este executa diretamente no hardware sem mais nenhuma intervenção de software ou *firmware*.

Na figura 3.6 os autores ilustram os elementos envolvidos na emissão/receção de áudio na rede. O som é captado através de um microfone e é digitalizado com um ADC PCM de 16 bits e armazenado num *buffer*. Depois é amostrado t ms de voz do *buffer* que resulta 8t bytes de dados depois da compressão usando o codec G.711. Seguidamente

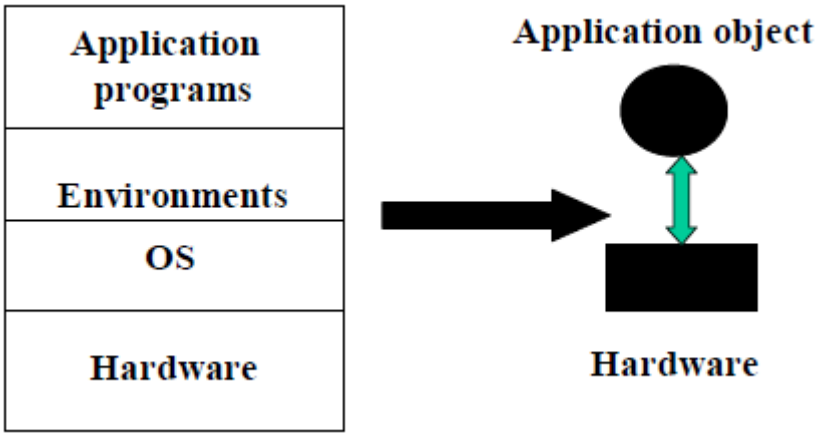


FIGURA 3.5: Arquitetura Geral dos Bare PC [24]

são adicionados os cabeçalhos RTP, UDP e IP à amostra retirada e envia-se para o *send buffer*. Feito isto a placa de rede envia o pacote de voz para a rede. Inversamente, um pacote de voz é armazenado no *receive buffer* pela placa de rede, são tratados os cabeçalhos RTP, UDP, IP e a *media* são enviados para o *jitter buffer*. Após a descodificação efetuada pelo "*G.711 Decoder*", os dados são armazenados no *speaker buffer* e posteriormente reproduzidos.

Os autores referem ainda que não implementaram mecanismos de supressão de silencio nem "disfarce"de perda de pacotes que resultaria num aumento de performance. Também, uma vez que os seus interesses eram numa comunicação peer-to-peer, não usaram o SIP para gestão da conexão.

Para testarem a performance obtida pela aplicação VoIP os autores realizaram várias experiências no seu laboratório usando uma simples e isolada rede de teste. Para obtenção e medição foi usado um *hub* (em vez de um *switch*) para ligar os vários Bare PC e um *sniffer* passivo para monitorizar as chamadas. Cada Bare PC era um 2.4GHz Dell Optiplex GX260 com 512Mb de memória. Foi também usada uma ferramenta para calcular o MOS. Começaram por transferir a *media* entre 2 Bare PC com a aplicação VoIP e mediram a perda de pacotes, *delay* e o *jitter* para pacotes de voz de tamanho desde 10ms até 120ms. Como era o único tráfego a passar pela rede não houve perda de pacotes, tal como era esperado. Os resultados obtidos do *delay* e do *jitter* ficaram dentro do intervalo dado como aceitável para transmissão de voz. Depois foi substituído um dos Bare PC por um PC com Sistema Operativo Windows e voltaram a efetuar o mesmo teste. Neste teste o PC com Windows permite apenas um pacote de tamanho máximo até 60ms. Para chamadas numa só direção o intervalo máximo está muito próximo do tamanho dos dados, daí poder ser observada uma relação linear.Se for considerado

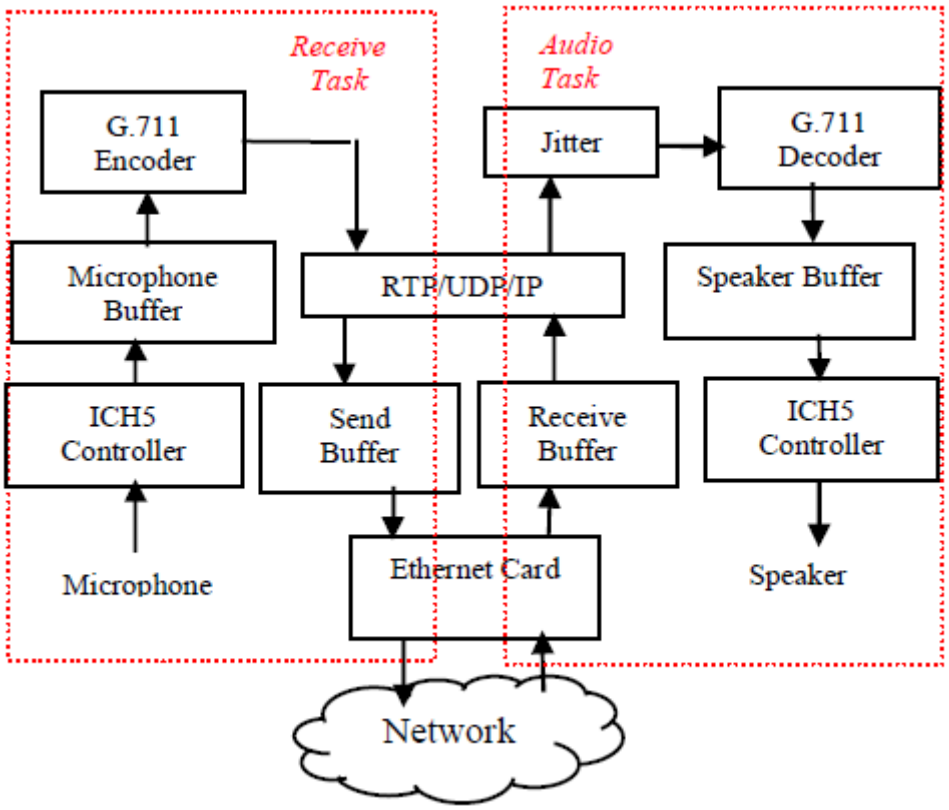


FIGURA 3.6: Componentes da Aplicação VoIP [24]

tráfego em ambas as direções verifica-se que a variação dos *Bare* PC é menor que a do cliente Windows.

Segundo os autores, os Bare PC são vantajosos devido à sua simplicidade e eficiência. Podem ser usados em comunicações ponto-a-ponto sem qualquer dependência de nenhum Sistema Operativo no qual os testes realizados o provaram. Ainda há trabalho a desenvolver pela frente, como por exemplo, questões de segurança e desenvolvimento de mecanismos para atravessar NATs e *firewalls*.

3.4.2 *Mobile* P2PSIP

Em [25] os autores apresentam uma rede peer-to-peer que permitem estabelecimento se sessões multimédia de dispositivos móveis usando o SIP. Nesta arquitetura não existem qualquer tipo de servidores centrais e o ambiente de teste consiste no estabelecimento de chamadas VoIP dos dispositivos através da rede. Os autores começam por apresentar a arquitetura do sistema, os conceitos gerais do SIP e como é feita a descoberta de recursos dentro do *overlay*. Seguidamente apresentam alguns testes efetuados.

Esta arquitetura permite estabelecimento de sessões multimédia entre membros do *overlay* usando o SIP. A arquitetura está representada na figura 3.7.

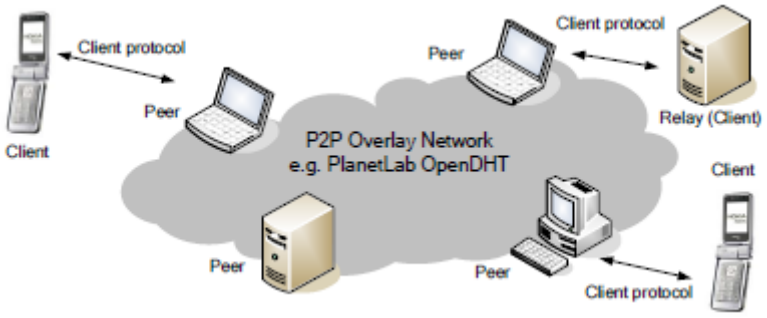


FIGURA 3.7: Arquitetura Geral do *overlay* [25]

Nesta arquitetura existem dois tipos de entidades que são os *peers* tal e qual como noutra rede p2p comum, com as mesmas funções e os clientes que se conectam aos *peers* para obter ou adicionar informação à rede. o SIP UA pode ser visto como um *peer* ou cliente e o *software* que os autores utilizaram deve estar instalado nestes dispositivos. Igualmente a outras implementações p2p, os dados são armazenados em DHT's e também são usados mecanismos de NAT-Transversal.

Para efetuar os testes foram utilizados dois Nokia E61 e a DHT utilizada foi a *openDHT* da *PlanetLab*. Os autores mediram a performance do sistema em dois cenários, o primeiro com ligação à rede por 3G e o outro por WLAN. O acesso por WLAN consistiu num *access point* que estava ligado diretamente à Internet. O serviço 3G foi fornecido por um operador Finlandês chamado Elisa. Para obterem os dados, usaram um *software* que tem a capacidade de medir os atrasos no registo, descoberta de endereços e estabelecimento de chamadas. Para contabilizar o total de dados transmitidos na rede foi usado o *Ethereal* que é um *sniffer* de rede tal como sucessor o *Wireshark*. As medições foram feitas através de 21 amostras obtidas de cada um dos parâmetros.

Os autores concluem, baseado nos testes efetuados, tanto no registo como no estabelecimento de chamadas que o valor médio do atraso é bastante superior quando usado como ligação à rede o 3G. Verificam ainda que quando usada a rede 3G que existem picos nos atrasos e que apesar de não encontrarem uma explicação clara, apontam para duas explicações, o mecanismo de reconfiguração da DHT ou congestão na rede 3G.

3.4.3 *ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP*

Em [26] é apresentada uma solução para encaminhamento da *media* que garanta a qualidade requerida nas comunicações VoIP. O primeiro passo foi comprovarem que o encaminhamento de dados por nós intermédios é benéfico face ao encaminhamento direto entre os pontos fim a fim. Seguidamente, os autores estudaram como o *skype* faz o encaminhamento da *media* e como é feita a escolha dos nós intermédios, no qual detetaram várias falhas no algoritmo de escolha de nós intermédios. Com base no algoritmo de encaminhamento do *skype* os autores decidiram propor um mecanismo capaz de melhorar a qualidade das chamadas VoIP e a escalabilidade do sistema.

Segundo os autores, o VoIP tem de cumprir certos requerimentos para ter qualidade da chamada. O primeiro requerimento apresentado é o MOS (*Mean Opinion Score*) que é uma métrica usada para avaliar a "qualidade do discurso"e que o seu valor varia entre 1 e 5. Neste contexto, o valor definido para o MOS é de 3.6, então, um MOS abaixo de 3.6 significa uma má qualidade de chamada. Outro fator é o OWD (*One Way Delay*), que o ITU (*International Telecommunication Union*) definiu, não deve exceder os 150ms.

De forma a compararem o *routing* direto com o *routing* em nós intermédios, com o objetivo de verificar qual o melhor a cumprir a qualidade requerida para o VoIP, arquitetaram um ambiente de teste que consistiu em obter 269413 IPs, coletados do *Gnutella*, e um grande número de entradas das tabelas de *routing* BGP, obtidas através do *RouteViews* (*www.routeviews.org*) e do *RIPERIS (http://www.ripe.net/data-tools/stats/ris/ris-raw-data)*. Dessas entradas obtidas das tabelas de *routing* BGP, os autores organizaram os IPs por prefixos e deduziram as conexões entre os vários Sistemas Autónomos. IPs com o mesmo prefixo foram agrupados no mesmo *cluster*. Ao cruzar os IPs obtidos do *Gnutella* com as entradas obtidas das tabelas de *routing*, dos 269413 endereços IP, 103625 coincidiram com 7171 prefixos e pertencentes a 1461 Sistemas Autónomos. Aleatoriamente, foi eleito um *host* de cada Sistema Autónomo como seu representante e foram feitas medições do RTT entre eles, diretamente e com encaminhamento feito em um nó intermédio, usando uma ferramenta denominada por *King*. Com encaminhamento direto, das 10^5 medições efetuadas, 10^3 verificou-se que o RTT é superior a 300ms e em 10^4 o RTT é superior a 200ms. Nas restantes medições foram obtidos RTTs menores que 200ms. Com encaminhamento num nó intermédio, 60 por cento dos valores obtido são menores que os valores obtidos no encaminhamento direto, nos quais a maioria teve um RTT inferior a 100ms. Assim, concluíram que o encaminhamento com um nó intermédio é benéfico.

Os autores para testar o funcionamento do *skype* usaram uma ferramenta chamada *WimDump* para obter a *media* transmitida durante 14 sessões. Adicionalmente implementaram uma aplicação para analisar os dados obtidos. Após a analise concluíram que o *skype* tem várias limitações, nomeadamente:

- Elevada latência devido à inapropriada escolha dos nós intermédios - em duas das sessões verificaram que o RTT é superior a 350ms, então, significa que a qualidade dessas duas sessões é insatisfatória. Além disso, em uma outra sessão o RTT foi sensivelmente 250ms (que é inferior ao máximo definido) mas mesmo assim procurou e encontrou novos caminhos com RTT inferior, mas não fez uso deles. Concluíram ainda que o *skype* faz *probes* a nós sem nenhum conhecimento base da ligação, o que causa um grande número de *probes* na rede desnecessariamente;

- Múltiplos *probes* para nós pertencentes ao mesmo Sistema Autónomo - para cada sessão os autores usaram o *traceroute* num dos *hosts* da extremidade da ligação e verificaram que o *skype* não considera a topologia interna do Sistema Autónomo. Uma vez que um *peer* pertencente a um determinado Sistema Autónomo não consegue cumprir o requerimento de OWD inferior a 150ms, um outro *peer* pertencente ao mesmo Sistema Autónomo também não o vai cumprir uma vez que estes partilham a mesma ligação física. Nestes casos, efetuar *probes* para mais que um *peer* dentro de um Sistema Autónomo é desnecessário e provoca mais tráfego nas ligações;

- Tempo de estabilização de nós intermédios elevado - verificou-se que o *skype* faz demasiados *probes* para encontrar os nós intermédios ideais, o que provoca um tráfego elevado e grandes atrasos. Nos testes efetuados verificaram numa sessões que esta demorou 329 segundos (mais de 5 minutos) até estabilizar o melhor caminho. O tempo de estabilização pode ser elevado devido ao facto das várias trocas de nós intermédios ao inicio de cada sessão;

- Probes desnecessários - além das razões já mencionadas, ainda se verificou que em cada uma das sessões foram feitos mais de 20 *probes* até encontrar e estabilizar o caminho ideal. Ainda, em duas das sessões foram atingidos os máximos de 37 e 59 *probes* nos quais os seus RTTs foram de 355ms e 238ms, respetivamente. Mesmo após a estabilização, os testes mostraram que na maioria das 14 sessões, quando estas estabilizaram, continuaram a efetuar *probes*. Isto significa que as condições da rede estão em constantes mudanças.

Identificadas as limitações do *skype* os autores propuseram uma solução denominada por *ASAP* e que tem como característica principal os nós terem conhecimentos do seu

Sistema Autónomo e dos Sistemas Autónomos mais próximos. As entidades constituintes do ASAP são:

- *Bootstrap* - são *hosts* com um grande poder de processamento e que devem estar sempre ligados. Esta entidade tem a função de manter a informação sobre as topologias dos Sistemas Autónomos atualizadas, ter uma tabela com os prefixos dos IPs que pertencem a cada um dos Sistemas Autónomos e uma tabela com os prefixos dos IPs pertencentes a um determinado Sistema Autónomo, sendo que para identificar um Sistema Autónomo é utilizado um número (ASN). Quando recebe um pedido de um novo *peer* para se juntar à rede, o *bootstrap* extrai-lhe o prefixo do endereço IP , e é-lhe devolvido a informação de qual *cluster* pertence e qual o endereço IP do *peer* responsável por aquele *cluster*. O *bootstrap* também tem como função difundir a topologia pelos representantes dos Sistemas Autónomos quando solicitado. Além disso, no caso de algum problema com os representantes, o *bootstrap* tem como função eleger um ou vários novos representantes;

- Representantes - este tipo de *peers* devem ter um bom poder de processamento e que possua uma ligação com largura de banda elevada dento do Sistema Autónomo a que pertence. Esta entidade tem como funções saber quais os endereços IPs de todos os *peers* pertencentes ao seu *cluster*, periodicamente contactar o *bootstrap* a fim de obter a topologia do Sistema Autónomo atualizada, periodicamente executar uma função que tem como objetivo atualizar quais os *clusters* mais próximos e, por último, processar pedidos provenientes de *clusters* vizinhos;

  End hosts - *hosts* que efetuam e recebem chamadas. Estes têm como dever obter a informação para se juntar à rede, tornarem-se representantes do *cluster* a que pertence, se necessário, periodicamente informar o representante das suas capacidades (largura de banda, poder de processamento, tempo *online* e outra informação relevante) e por último, quando inicia uma chamada, executar uma função que tem como objetivo encontrar o melhor caminho para a *media* transitar.

Na figura 3.8 estão representadas as várias entidades e respetivas interações.

Como se pode verificar na imagem, estão representadas as trocas de mensagens desde a junção à rede do *host* h1 até à chamada entre o *host* h1 e o *host* h2. Quando o *host* h1 presente no *cluster* A se quer juntar à rede, envia um *join* ao *bootstrap* no qual lhe devolve o endereço IP do respetivo representante. Depois, contacta o representante e este devolve-lhe a lista de *clusters* mais próximos. Quando h1 inicia uma chamada VoIP com o *host h2*, presente no *cluster* B, h1 faz a medição do RTT da ligação direta entre eles e se verificar que é superior a 300ms, h1 contacta h2 a fim de saber quais os *clusters* mais

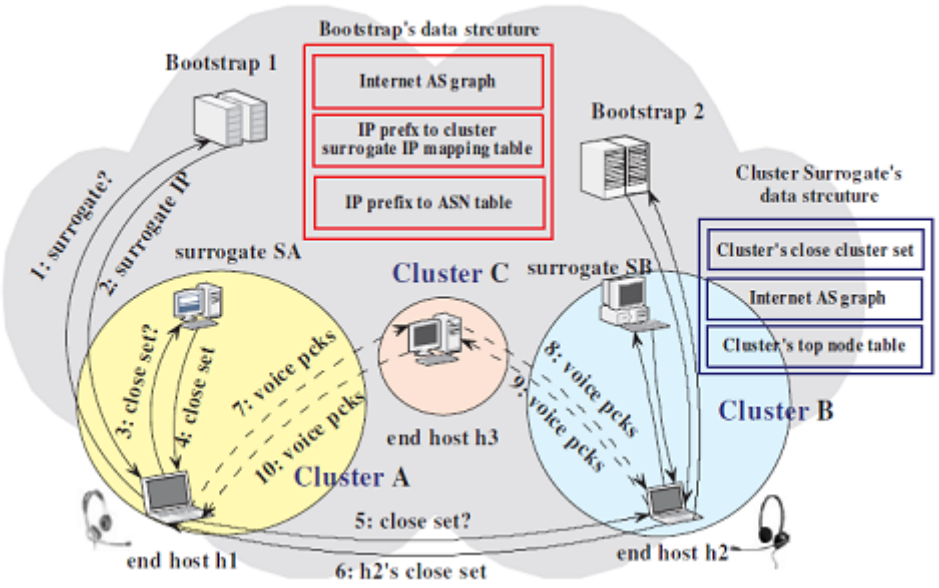


FIGURA 3.8: Arquitetura geral do ASAP [26]

próximos do *cluster* B. Após cruzar os *clusters* presentes na lista enviada por h2 com os *clusters* presentes na lista local, é feita uma seleção do melhor *cluster* para encaminhar os dados, neste caso é o *cluster* C. Então, a *media* passam a ser encaminhados por um *peer* intermédio que nesta situação é o h3.

Os autores após implementarem o ASAP arquitetaram um ambiente de teste de forma a comparar o desempenho do ASAP com outros mecanismos de procura de nós intermédios que efetuam o encaminhamento da *media*. Então, com as mesmas entradas das tabelas de *routing* BGP já referidas, foram obtidos 20955 Sistemas Autónomos, 56907 ligações entre eles e foram coletados 100000 pares de *peers* do *Gnutella* que representaram 100000 chamadas VoIP, das quais 1000 sessões tiveram na sua ligação direta um RTT superior a 300ms. Os métodos de procura de nós intermédios a que se comparou o ASAP foram denominados por DEDI que usa nós intermédios pré-definidos, RAND que escolhe aleatoriamente um *peer*, MIX que é uma combinação do DEDI e do RAND, e por último o OPT que escolhe sempre o *peer* intermédio com ligações de menor RTT. Esta escolha é feita "*offline*", isto é, já conhece todos os RTTs de todas as ligações. Com os resultados obtidos verificou-se que usando o DEDI, RAND e o MIX todas as sessões encontraram mais de 500 caminhos alternativos enquanto que com o ASAP, 90 por cento das sessões conseguiram encontrar mais de 10^4 caminhos alternativos. Quanto ao RTT verificou-se que os resultados obtidos pelo ASAP estão muito próximos do OPT (que são os caminhos ótimos) e nos quais os RTTs são inferiores a 115ms, enquanto que no DEDI, RAND e MIX mais de 5 por cento das sessões tem, como melhor RTT, valores acima de 1 segundo.

Os autores concluem que o desempenho do ASAP se mostrou altamente eficiente e escalável, no qual o seu desempenho se encontra muito perto de um algoritmo de seleção ótima.

Capítulo 4

Aplicação VoIP na Rede P2P

Um dos principais objetivos deste trabalho é o desenvolvimento e avaliação de uma aplicação VoIP que utilize uma rede P2P baseada totalmente em SIP que foi desenvolvida num trabalho anterior [6]. Assim, neste capitulo, começa-se por descrever a rede P2P e depois as principais decisões tomadas e respetiva justificação para o desenvolvimento da aplicação VoIP, que se dá pelo nome de P2PSIPVoIP.

4.1 Rede *overlay* base

Para formar o *overlay* base, cada nó usa um protocolo especifico para se juntar aos outros nós. O protocolo P2PSIP usado na rede é o dSIP [13] devido a ser um protocolo totalmente baseado no SIP e o que mais se identifica com o trabalho. O facto do dSIP ser totalmente baseado em SIP é vantajoso pois o SIP é um protocolo simples, normalizado e com um bom desempenho. Além disso, o facto de ser um protocolo suportado por um grande número de dispositivos, permite a reutilização de *stacks* SIP implementadas, minimizando assim o número de protocolos que um peer precisa suportar. Um outro aspeto positivo do SIP é que sendo tráfego conhecido, pode atravessar as *firewalls* sem ser bloqueado, tal como o HTTP.

Uma vez formado o overlay, este é utilizado para armazenar e localizar recursos, oferecendo de uma forma distribuída, os serviços que habitualmente um Servidor de Registo e Localização oferece numa arquitetura SIP tradicional. Os recursos armazenados pelo overlay são compostos por um par chave/valor, onde a chave é o identificador do recurso. No contexto deste trabalho, a chave de um recurso é o endereço SIP do utilizador, e o valor é o endereço IP e porta de contacto.

O protocolo dSIP impõe restrições aos algoritmos DHT e define que pelo menos o algoritmo *Chord* [19] deve ser implementado. Contudo, na rede desenvolvida, para além da implementação obrigatória do Chord também está implementado o *EpiChord* [27], que é uma variante do *Chord* que pretende melhorar o desempenho geral do *overlay*. Na rede desenvolvida, ambos os algoritmos DHT utilizam como algoritmo de *hashing* o SHA-1 de 160 bits para gerar o *hash* dos identificadores dos peers e dos recursos.

Nesta arquitetura foi implementada e considerada uma hierarquia em dois níveis, isto é, com dois tipo de *peers*. Num *overlay* P2P tradicional não hierarquizado, todos os *peers* possuem a mesma importância, participando ativamente nas tarefas de encaminhamento, armazenamento e localização de recursos disponibilizadas pelo *overlay*. Contudo, há casos em que atribuir a todos os *peers* a mesma importância pode não ser o mais desejável, quer por questões de segurança, quer pelo desempenho do overlay. Neste caso foi criada uma hierarquia com dois níveis em que no nível inferior se encontram os *peers* com menos capacidades e no nível superior os *peers* com mais capacidades. *Peers* com poucas capacidades têm algumas das seguintes características:

- Ligação à rede de menor qualidade (tráfego limitado, menor largura de banda, ou ligação com maior probabilidade de perda de pacotes);
- Pouco tempo de permanência no *overlay*, podendo gerar muito tráfego de manutenção do *overlay* (transferência de recursos, atualizações das tabelas..);
- Recursos limitados em termos de *hardware* (CPU, memória, bateria, etc)

Estes *peers* com poucas capacidades foram designados por *Clientes* já que não fazem parte efetivamente do *overlay*, ou seja, não participam em qualquer função de encaminhamento ou armazenamento de recursos, contudo, podem utilizar os serviços disponibilizados por ele. Para isso basta ligarem-se a pelo menos um *peer* que integra o *overlay*, enviando para este os seus pedidos para armazenar ou localizar recursos. Desta forma, um cliente atua sempre para seu próprio beneficio, não recebendo mensagens que não são para si, nem tem a responsabilidade de armazenar dados, utilizando um *peer* pertencente ao *overlay* como intermediário para aceder aos serviços que este disponibiliza. Este é o nível inferior da hierarquia.

O nível superior da hierarquia é constituído pelos dispositivos com mais capacidades e que são chamados de *peers*. São estes que formam efetivamente o *overlay*. Estes atuam como *peers* normais, armazenando recursos e encaminhando mensagens entre si. A única alteração que é necessário efetuar nos *peers*, é adicionar suporte para clientes. Isto é, permitir que os *peers* possam receber do exterior do *overlay*, mensagens para localizar ou armazenar recursos no *overlay*. Sempre que é recebido de um cliente um pedido

este executa-o como se fosse seu, não havendo necessidade de efetuar alterações nos algoritmos DHT. Na rede implementada, um dispositivo que se conecte como cliente, será sempre cliente e um *peer* será sempre um *peer*, isto é, não há promoção/despromoção dos dispositivos na hierarquia do *overlay*. A promoção e despromoção dinâmica está planeada para trabalho futuro.

Quanto ao tipo de mensagens utilizadas pelo dSIP são mensagens SIP tradicionais com a adição de dois novos cabeçalhos necessários para o transporte de informação relevante para a gestão do *overlay*. Visto que existe uma hierarquia de dois níveis foi necessário especificar os tipos de mensagens que os clientes devem utilizar para comunicar com os *peers*, isto porque no dSIP não existe distinção entre os vários *peers*. Para simplificar o processo de comunicação entre os níveis da hierarquia, foi adicionado um cabeçalho identificador do Cliente e este baseia-se no cabeçalho identificador do *peer*. Desta forma os *peers* conseguem identificar o Cliente que originou uma determinada mensagem.

Na rede desenvolvida todos os pedidos efetuados são do tipo *REGISTER* e as respostas são dadas através de códigos com significados diferentes: *REDIRECT*, quando uma mensagem deve ser redirecionada para outro peer, *NOT FOUND*, quando o peer ou recurso não foi encontrado, e *OK* quando o pedido foi efetuado com sucesso. Existem vários tipos de mensagens P2PSIP, onde cada tipo de mensagem se diferencia nos cabeçalhos. Assim sendo os tipos de mensagens utilizadas na comunicação entre peers e cliente-peers são as seguintes:

- Peer Register - registo de um *peer* no *overlay*
- Peer Query - procura de um *peer* no *overlay*
- Peer Leave - saída de um *peer* do *overlay*
- Resource Register - registo de um recurso no *overlay*
- Resource Query - procura de um recurso no *overlay*
- Resource Remove - remoção de um recurso do *overlay*
- Client Register - registo de um cliente no *overlay*
- Client Leave - saída de um cliente do *overlay*
- Client Resource Register - registo de um recurso proveniente de um cliente no *overlay*
- Client Resource Remove - remoção de um recurso proveniente de um cliente do *overlay*

- Client Resource Query - pedido de um cliente para procura de um recurso no *overlay*

Estas mensagens possuem formatos idênticos, tendo que o que as distingue é um parâmetro que determina qual o seu tipo.

4.2 Aplicação VoIP

Antes da implementação da aplicação VoIP foi necessário adicionar funcionalidades ao *overlay* para dar suporte à sinalização da chamada entre *peers*. Para isso foram adicionados três novos tipos de mensagens:

- Peer Invite - envio de um INVITE para o *peer* a contactar
- Peer Ack - envio do ACK após receber confirmação positiva do *peer* a contactar.
- Peer Bye - mensagem de sinalização de fim de chamada originado por qualquer um dos *peers* da chamada.

Após adicionar estas funcionalidades foram estudadas três alternativas na implementação do P2PSIPVoIP. Numa primeira implementação foi utilizado o *overlay* apenas para a descoberta de recursos, isto é, apenas para saber qual o IP e porta no qual o agente SIP destinatário se encontra. A sinalização da chamada e transmissão dos dados são efetuados diretamente pela rede física (fora do *overlay*). A vantagem desta solução é o facto dos *peers* não precisarem de nenhum algoritmo de procura de um *peer*, ou vários, que lhe reencaminhem os dados, o que torna a aplicação mais leve. As desvantagens é a dificuldade em atravessar NATs e *firewalls*, o fraco desempenho quando há um elevado tráfego na ligação direta[1], não tirando assim partido do *overlay*.

Na figura 4.1 está ilustrado um exemplo desta solução, onde o *host1* comunica com o *host2*. Os *hosts* (*host* 1 e *host* 2) começam por se registar no *overlay*. O registo destes no *overlay* está ilustrado como um *Put*, no qual leva o URI e IP-porta no qual o seu agente SIP está ativo. Posteriormente, o *host1* faz um *Get* para saber qual o IP-porta do agente SIP do destinatário, dado o seu URI. Após obter a resposta trata de enviar um *INVITE* para o *host2* que este aceita e a transmissão de dados é feita diretamente entre eles. Na figura é o *host1* que origina o *BYE* que indica fim de chamada mas pode ser qualquer um dos *peers* a faze-lo.

[1] "ligação direta"significa que a *media* transita por fora do *overlay*

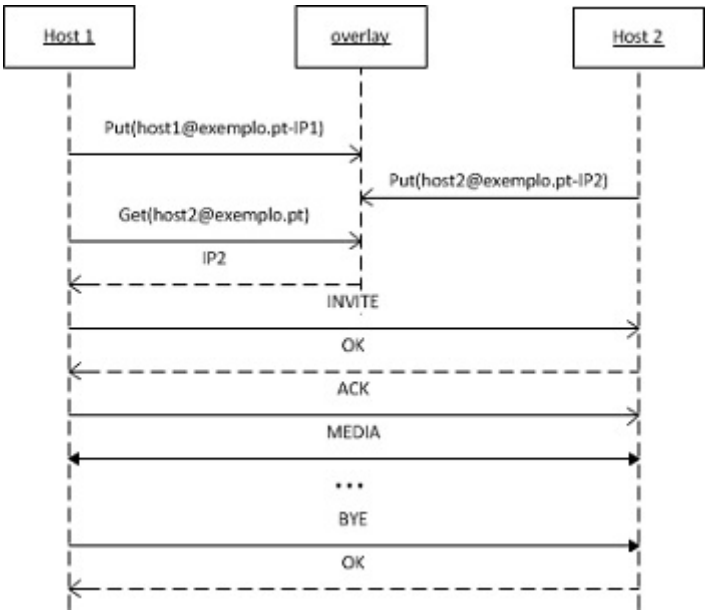


FIGURA 4.1: Exemplo do funcionamento da aplicação VoIP sem reencaminhamento

Na segunda implementação, o *overlay* não foi apenas utilizado para descoberta de recursos mas também para reencaminhar a *media* por um *peer* intermédio. Nesta solução o *peer* emissor deve ter a noção se a ligação direta está congestionada ou não e tomar a decisão por onde envia os dados. O *peer* intermédio vai apenas reencaminhar os dados de uma forma simples para o destinatário como vai ser explicado mais à frente. A vantagem desta solução é o facto de encontrar um caminho alternativo para a transmissão dos dados quando a ligação direta está congestionada e o facto de aproveitar o *overlay* para atravessar NATs e *firewalls*. Em contra ponto, a aplicação tem como processamento adicional descobrir um *peer* intermédio e tomar a decisão se deve mandar os dados diretamente para o recetor ou para o *peer* intermédio. Na figura 4.2 está ilustrado um diagrama de sequência que exemplifica esta implementação.

O registo dos *peers* está ilustrado como um *Put*, o qual contem o URI e IP-porta do agente SIP de cada um. Após se registarem, o *host1* faz um *Get* com o URI do *host2* com o objetivo de descobrir qual o IP-porta para enviar o *INVITE*, que segue pela ligação direta tal como na implementação anterior. Após o *host2* aceitar a chamada e admitindo que a ligação direta está congestionada, os dados são encaminhados por um *peer* denominado por *host3*. Este *peer* intermédio está ilustrado na figura como sendo o mesmo encaminhador em ambas as direções mas cada um dos *hosts* poder usar um *peer* diferente como intermediário. No exemplo é o *host1* que envia a mensagem *BYE* para sinalizar o fim de chamada mas pode ser qualquer um dos *peers* a faze-lo.

A terceira implementação é muito semelhante à segunda, com a diferença que nesta o número máximo de *peers* intermédios a ser utilizado é definido pelo utilizador. Esta

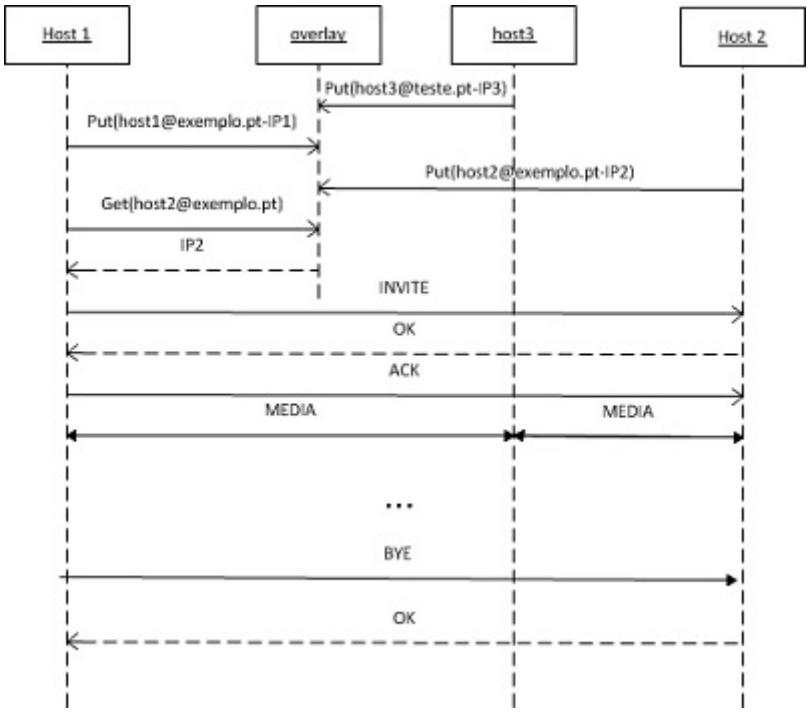


FIGURA 4.2: Exemplo do funcionamento da aplicação VoIP com um *peer* intermédio

solução não restringe o algoritmo a um só salto, tornando assim maiores as hipóteses dos dados chegarem ao recetor quando existe um grande congestionamento na ligação direta, bem como nas ligações vizinhas. A desvantagem desta solução é o facto de obrigar os *peers* intermédios a manterem em *cache* uma tabela de encaminhamento tal como vai ser explicado mais à frente. Além disso, a procura de *peers* para reencaminhar os dados vai gerar mais tráfego no *overlay*. Na figura 4.3 está ilustrado um diagrama de sequência que exemplifica esta implementação.

O registo dos *peers* está ilustrado como um *Put* tal como nos exemplos anteriores. Após se registarem, o *host1* faz um *Get* com o URI do *host2* com o objetivo de descobrir qual o IP-porta. Nesta solução a sinalização da chamada também é feita pelo caminho direto entre ambos os *hosts*. Após o *host2* aceitar a chamada e admitindo que a ligação direta está congestionada, os dados são encaminhados, neste caso, por dois *peers* que são o *host3* e o *host4*. Estes *peers* intermédios estão ilustrados na figura como sendo os mesmos encaminhadores em ambas as direções mas não implica que tem de ser obrigatoriamente os mesmos, podem ser estes dois numa das direções e quaisquer outros não representados na figura na direção oposta. No exemplo é o *host1* que envia a mensagem *BYE* para sinalizar o fim de chamada mas pode ser qualquer um dos *peers* a faze-lo.

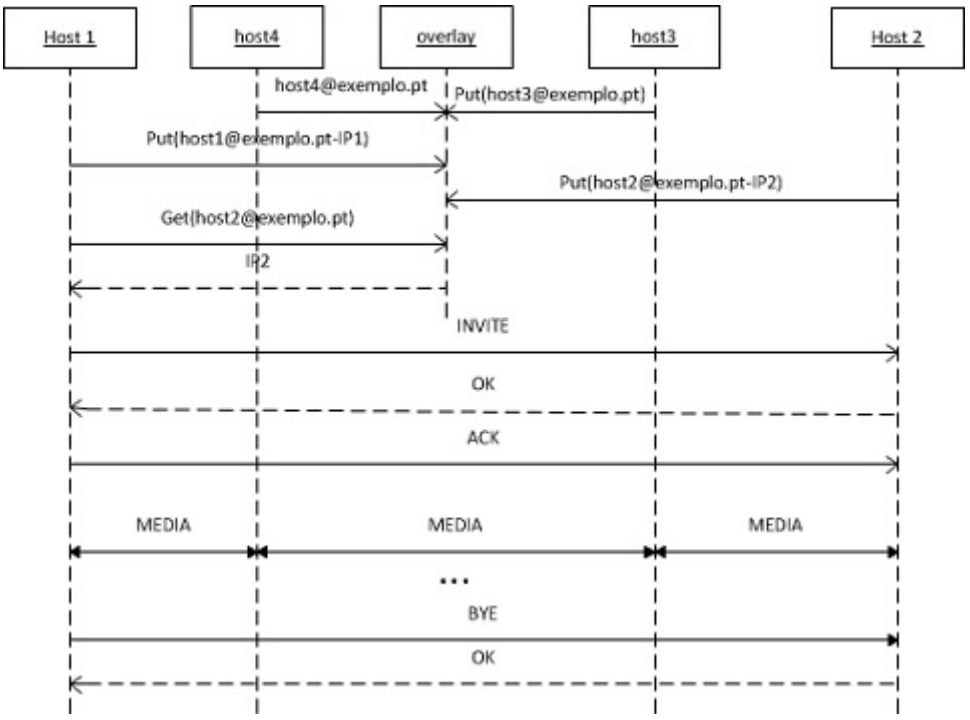


FIGURA 4.3: Exemplo do funcionamento da aplicação VoIP com n *peers* intermédios

4.3 Encaminhamento de dados

Como já referido o encaminhamento de dados pelos *peers* do *overlay* tem vantagens tais como atravessar NATs e *firewalls*, oportunidade de caminhos alternativos desde o emissor até ao recetor e um melhor desempenho. A ideia geral é o *peer* emissor sabendo que a ligação direta está congestionada, poder optar por caminhos alternativos através de um ou mais *peers* que façam chegar os dados até ao recetor. No entanto, existem vários requisitos mínimos que têm de ser cumpridos para ter um QoS mínimo. A titulo de exemplo refira-se o atraso na ligação fim-a-fim entre o emissor-recetor não exceder os 150ms [26]. Na figura 4.4 está ilustrado um exemplo de uma topologia de rede em que os dispositivos sombreados a cinzento pertencem a um *overlay* (não sendo a sua estrutura importante para este caso).

Supondo que há uma chamada de voz a decorrer entre o dispositivo n5 e n11, no protocolo de encaminhamento da rede física os dados são enviados pelo menor número de saltos até ao destino, neste caso, r2-r4 e r4-r2. Tal como representado na figura, o facto do OWD(*One Way Delay*) entre r2-r4 e vice-versa ser superior a 150ms leva a que se deva procurar um caminho alternativo para os dados transitarem.

A fim de tornar possível o encaminhamento de dados, é necessário a adição de novas

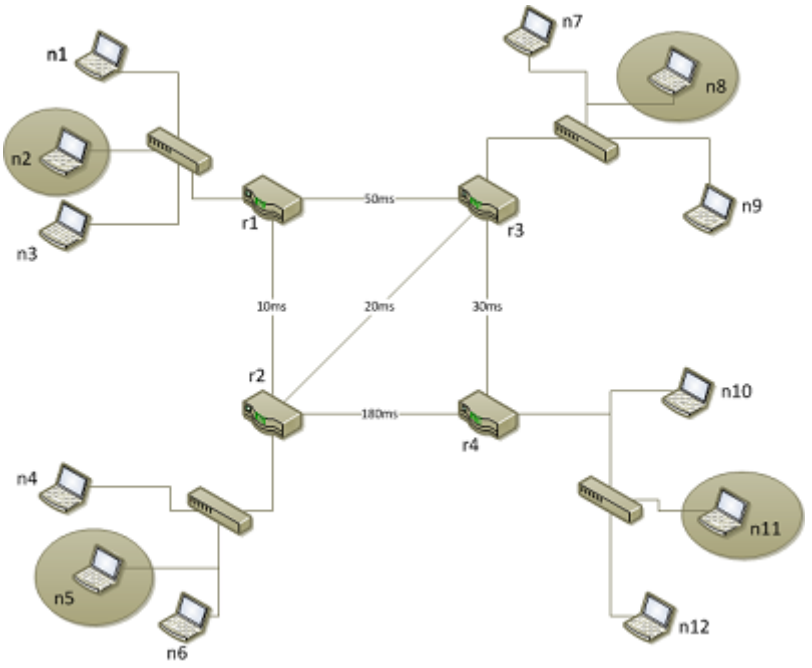


FIGURA 4.4: Topologia de Rede Exemplo

componentes, nomeadamente a interação com a *cache* DHT utilizada, medição da qualidade da chamada, criação de novas tabelas auxiliares e a sua atualização, criação de um algoritmo de encaminhamento e criação mensagens que mantenham atualizadas as tabelas e em complemento ao algoritmo de encaminhamento. O objetivo e enquadramento destas novas componentes é apresentado de seguida.

4.3.1 Interação com a *cache* DHT

A *cache* DHT é uma tabela mantida em cada *peer* que forma o *overlay* e que contém uma série de *peers* conhecidos a fim de tornar o tempo de procura de recursos o menor possível [6]. Esta *cache* é dependente do algoritmo DHT em utilização (*Chord* ou *Epichord*). Como estão presentes nesta tabela uma série de *peers* conhecidos, a solução concebida utiliza esta *cache* para obter uma lista de *peers* vizinhos que sejam potenciais encaminhadores de *media*.

No contexto desta solução, cada *peer* deve ter uma lista com *peers* vizinhos e potenciais reencaminhadores, sendo que é com auxilio desta *cache* que os IPs dos vizinhos são obtidos.

4.3.2 Tabelas utilizadas

Cada *peer* possui uma lista de vizinhos que são potenciais reencaminhadores de mensagens sendo que a métrica para os validar e considerar é o RTT (*Round Trip Time*) que o *peer* tem para cada um deles. Em função disso foi criada uma tabela denominada como "tabela de distancias", onde se armazenam o endereço IP de um determinado *peer* vizinho e respetivo RTT.

Devido à necessidade da constante atualização de tais valores, foi criado um mecanismo de refrescamento que será devidamente discutido mais à frente, que requer o uso de uma tabela auxiliar. A esta tabela foi dado o nome de "tabela de pedido de pings"ou apenas "tabela de pings". Os valores que esta armazena são o endereço IP destino e o *timestamp* inicial.

Quando um *peer* recebe alguma mensagem sonda para o avaliar como potencial reencaminhador, este precisa de saber a qual endereço IP deve responder consoante o endereço IP destino. Com tal objetivo foi criada uma "tabela de respostas"que contem o IP destino pedido e o respetivo endereço do *peer* que lhe fez o pedido.

Por ultimo, todos os *peers* possuem uma tabela para saber qual o próximo endereço IP a quem deve enviar os dados, que é denominada de "tabela de encaminhamento". Os campos que esta tabela possui são o IP destino, endereço IP para qual são encaminhados os dados e o RTT total desde o emissor até ao recetor

Na figura 4.5 estão representadas as tabelas auxiliares e necessárias que dão suporte ao reencaminhamento dos dados pelo *overlay.*

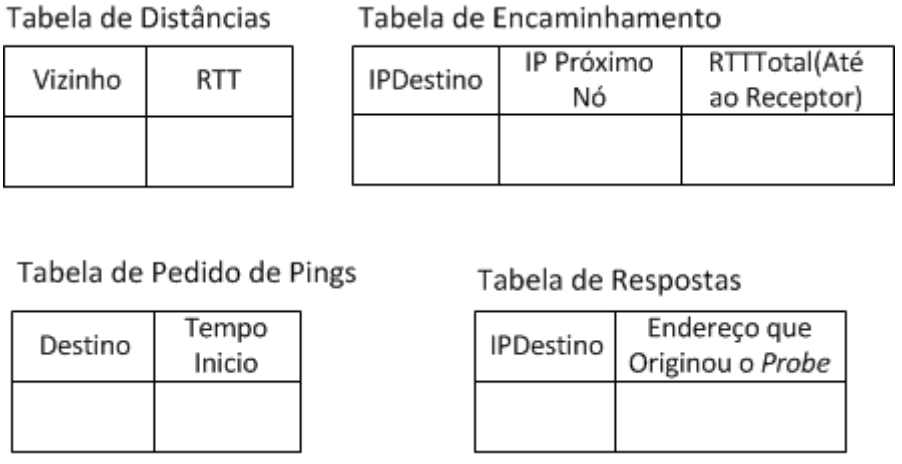


FIGURA 4.5: Tabelas presentes nos *peers*

4.3.3 Tipo de mensagens

Para dar suporte às funcionalidades do algoritmo de encaminhamento, fazer a medição da qualidade da chamada e manter a tabela de distâncias atualizada em cada um dos *peers* houve a necessidade de criar vários tipos de mensagens devidamente identificadas para cada um dos propósitos.

A atualização do RTT para cada um dos *peers* vizinhos contidos na tabela de distancia é feita através da implementação de um mecanismo de *pings*. Assim sendo, para lhe dar suporte foram criados dois tipos de datagramas que estão representados nas figuras 4.6 e 4.7. O formato destes datagramas bem como o funcionamento do mecanismo de medição do RTT entre determinados *peers* será apresentado e detalhado numa secção mais à frente.

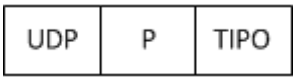


FIGURA 4.6: Datagrama *ping*

FIGURA 4.7: Datagrama *echo*

O reencaminhamento de dados é necessário quando detetado uma má qualidade de chamada. O parâmetro usado para determinar a qualidade da chamada foi o OWD, sendo a medição feita com auxilio à *media* transmitida, portanto é o recetor que faz este cálculo. Deste modo, foi necessário incluir nos datagramas um cabeçalho adicional chamado *Timestamp* no qual vai contido o tempo em que o datagrama foi enviado pelo emissor, permitindo assim calcular o OWD, subtraindo o tempo de chegada do datagrama ao *timestamp* lá incluído. Além disso, para o reencaminhamento nos *peers* intermédios estes precisam saber qual o endereço do destinatário para enviar os dados para o próximo nó. Para resolver este problema foi criado um campo na mensagem chamado 'IPDestino' o qual contem o endereço IP do *peer* final. O identificador deste tipo de datagramas é o 'V' de *voice* e todo ele está ilustrado na figura 4.8.

FIGURA 4.8: Datagrama *voice*

É o recetor que faz a medição do OWD com auxilio do *timestamp* contido na *media* e caso este seja superior ao valor de referencia deve alertar o emissor de alguma forma.

Consequentemente, foi construída uma mensagem no qual o seu identificador é o 'W' de *warning* e que é o único campo deste datagrama, tal como se pode verificar na figura 4.9.

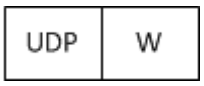


FIGURA 4.9: Datagrama *warning*

Quando o *peer* emissor recebe uma mensagem de *warning* significa que o recetor verificou que a qualidade da chamada não é suficiente, então deve procurar um caminho alternativo que satisfaça a condição em que o OWD seja inferior a 150ms. Assim sendo, o algoritmo de encaminhamento é despoletado e este tenta encontrar um vizinho por onde reencaminhar os dados, sendo necessário enviar-lhes uma mensagem para tal. Com este objetivo foi criada uma mensagem na qual contém o endereço IP destino do *peer*, o RTT acumulado até ao momento e o número de saltos máximo que esta mensagem pode efetuar. A utilização destes campos vai ser debatida posteriormente quando se falar do algoritmo. Estas mensagens chama-se *probes* e a sua estrutura está ilustrada na figura 4.10.

FIGURA 4.10: Datagrama Probe

No ultimo cenário, caso um *peer* que tenha recebido um *probe* e consiga retransmitir os dados até ao recetor com um OWD total inferior ao valor referência, deve notificar o *peer* originador do pedido. Houve portanto a necessidade de criar um tipo de mensagem de resposta. No primeiro campo deste datagrama é incluído o identificador que neste caso é o 'R' de *response*, no segundo campo vai o endereço IP do destinatário final e por ultimo o RTT total desde o *peer* emissor até ao *peer* recetor O formato desde datagrama está ilustrado na figura 4.11.

FIGURA 4.11: Datagrama Response

4.3.4 Algoritmo de Encaminhamento

O algoritmo é executado apenas após receção de um datagrama "*warning*". O primeiro passo do algoritmo é o envio de *probes* para os vizinhos contidos na tabela de distância

para quais o valor do RTT seja menor que 300ms (dobro do OWD máximo referenciado). Os *peers* vizinhos, ao receber mensagens de *probe*, verificam qual o seu RTT para o IP destino e, se esse RTT adicionado com o RTT acumulado incluído no *probe* for inferior a 300ms, este passa a ser candidato a próximo nó e notifica o emissor enviando uma mensagem do tipo *response*. Caso contrário repete o processo e envia *probes* para os seus vizinhos.

Usando como exemplo a figura 4.4, como o OWD entre r2-r4 é superior a 150ms, o dispositivo n5 e n11 vão procurar um caminho alternativo para transmitir os dados. Supondo que o n11 conhece n8 como seu vizinho, manda-lhe um *probe* para testar a sua ligação direta a n5, com um RTT acumulado 60ms. Em n8, uma vez recebido o *probe*, este verifica que o RTT até n5 é de 40ms. Como a soma desse RTT com o RTT acumulado é de 100ms que é inferior ao RTT referência de 300ms. n8 notifica n11 que é um possível próximo nó intermédio. Contudo não precisa de adicionar uma nova entrada à tabela de encaminhamento visto que o endereço IP do destinatário final coincide com o endereço IP do próximo nó. Nesses casos os *peers* intermédios não precisam de atualizar a tabela de encaminhamento. Este exemplo ilustra a segunda implementação do VoIP discutida anteriormente em que existe apenas um *peer* intermédio. Este cenário em que os *peers* não precisam de nenhuma tabela de encaminhamento acontece se e só se o número máximo de vizinhos que os *probes* atinjam seja exatamente igual a um.

Supondo agora que o RTT entre r2-r3 é de 250ms e que n8 conhece n2 como seu vizinho. o *peer* n8 quando faz a soma dos 250ms com o acumulado de 60ms obtém 310ms, que não obedece à condição. Por outro lado verifica que o seu RTT até n2 é de 100ms e que este RTT mais o RTT acumulado é de 160ms, logo envia um *probe* a n2 nas mesmas condições que n11 enviou a n8, diferindo no RTT acumulado que agora é 160ms. Após o *probe* ser recebido em n2, este verifica que o seu RTT até n5 é 20ms e que somando com os 160ms acumulados faz um total de 180ms. Como o valor obtido está dentro da condição estabelecida, n2 adiciona uma nova entrada à sua tabela de encaminhamento e envia uma resposta para n8, que por sua vez repete o mesmo procedimento. Nesta situação, os dados de n11 para n5 vão por n11-n8-n2-n5. Em n5 o algoritmo é aplicado de forma idêntica. Este cenário exemplifica a terceira implementação do VoIP discutida anteriormente.

4.3.5 Algoritmo de medição do RTT

Em face do algoritmo de encaminhamento apresentado surgiu a necessidade de fazer a medição do RTT em duas situações distintas. Uma delas é a necessidade da constante

atualização da tabela de distâncias e para isso foi criada uma tabela e dois tipos de mensagens que auxiliam este cálculo tal como apresentado anteriormente. Então, o primeiro passo é enviar uma mensagem *ping* para todos os *peers* contidos na lista de vizinhos, sequencialmente, armazenando na tabela de *pings* qual o seu endereço IP e respetivo *timestamps* do instante em que essa mensagem foi enviada. Os vizinhos, quando recebem um *ping*, devem criar uma mensagem de *echo*, adicionar-lhe o *timestamp* correspondente ao tempo da sua criação e envia-la para o endereço que originou o pedido. Quando recebida uma mensagem de *echo* o *peer* verifica qual o IP que originou esse datagrama e vai à tabela de *pings* procurar uma correspondência e caso esse endereço coincida com alguma entrada, é extraído o *timestamp* da mensagem e subtraído ao *timestamp* correspondente no endereço IP coincidente. Esta solução tem como característica principal não esperar qualquer resposta proveniente de cada um dos vizinhos, uma vez que estes podem ou não responder ou então demorar demasiado tempo a faze-lo, o que teria um impacto negativo, como por exemplo a existência de esperas intermináveis. Deste modo, uma vez que o mecanismo foi arquitetado desta forma foi denominado de *stateless pings*.

A outra situação em que precisamos efetuar a medição do RTT é quando um *peer* recebe uma mensagem *probe* e este tem de saber qual a latência até ao *peer* destino requerido Neste cenário a tabela de *pings* não é utilizada uma vez que o algoritmo consistem em mandar um *ping*, armazenar localmente qual o *timestamp*, esperar pela mensagem de *echo*, retirar-lhe o *timestamp* e subtraí-lo ao guardado anteriormente. Este processo é feito n vezes para obter uma maior precisão. A este tipo de serviço chamamos-lhe *stateful pings* uma vez que o processo espera efetivamente as mensagens de *echo*.

A distinção entre os *stateless pings* e os *stateful pings* é efetuada na mensagem *ping* no campo "Tipo", sendo necessária para a implementação dos mecanismos apresentados.

Decidiu-se efetuar a medição do RTT desta forma em vez de, por exemplo, utilizar uma biblioteca *ICMP* porque normalmente não é fácil a integração ICMP, em qualquer linguagem de programação, e além disso também porque os *peers* poderiam ou não aceitar este tipo de pedidos. O tipo de mensagens de *ping* e *echo* também poderiam ser estruturadas à custa do SIP tal como foram criadas as mensagens para a originar e manter o *overlay*. No entanto, dado que as mensagens SIP são muito maiores, gerando mais *overhead* no *overlay*, optou-se por criar mensagens de estrutura mais compacta para minimizar a probabilidade de congestionamento do *overlay*.

Capítulo 5

Implementação

Neste capítulo descreve-se a arquitetura e implementações desenvolvidas, onde o JAVA foi a linguagem de programação adotada. O capítulo começa por apresentar a arquitetura do *software* desenvolvido bem como as camadas que o constituem e debatidas as bibliotecas analisadas e justificação da adoção das preferidas. Posteriormente serão apresentadas as implementações das várias camadas constituintes do P2PSIPVoIP.

5.1 Arquitetura

A arquitetura desenvolvida foi projetada de modo a que a mesma arquitetura fosse facilmente utilizável por várias aplicações e não apenas o VoIP. Deste modo, o sistema foi estruturado em dois grandes blocos em que um é o *overlay*, as suas funcionalidades e interação com o utilizador. Um outro que é a comunicação, transmissão e reencaminhamento de dados entre *peers*. Posto isto, o desenho da arquitetura foi dividida em várias camadas, hierarquicamente posicionadas, em ambos os casos. Na figura 5.1 estão ilustradas todas as camadas constituintes do *overlay* e da aplicação VoIP desenvolvida.

5.1.1 Camadas referentes ao *overlay*

Camada Aplicação

Camada de interação com o utilizador. Nesta camada foi onde se implementou o *interface* gráfico que interage com o utilizador. Então, através desta camada o utilizador pode configurar os seus parâmetros, como por exemplo, qual o seu *username*, qual DHT utilizar ou a qual *overlay* se quer juntar. Além disso, é aqui que se torna possível efetuar e receber chamadas.

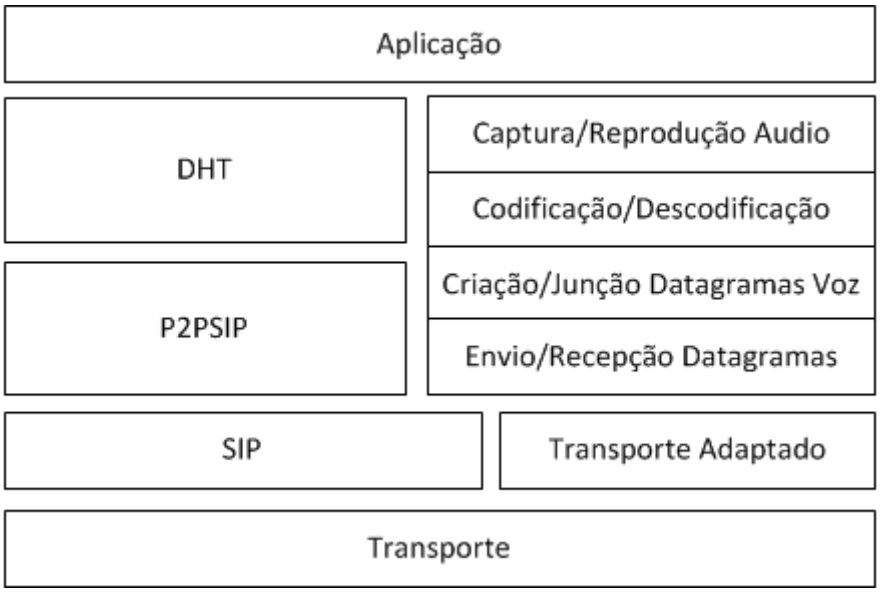


FIGURA 5.1: Arquitetura das aplicações - camadas

Camada DHT

Esta camada é responsável pela implementação do algoritmo P2P utilizado para criar e manter o *overlay*. Portanto é aqui onde as mensagens P2PSIP são construídas, calculadas as distancias entre dois identificadores, gestão de eventos e gestão de recursos. Além disso, oferece funções à camada de *Aplicação* para efetuar a procura de recursos no *overlay* bem como a entregadas respetivas respostas.

Camada P2PSIP

Esta camada é responsável por efetuar a ligação entre as camadas superiores e a camada SIP a utilizar. Deste modo, a utilização de diferentes bibliotecas SIP depende apenas de modificações ao nível desta camada. A separação em diferentes camadas permitiu que a implementação das camadas superiores pudessem ser feitas independentemente da biblioteca SIP a utilizar.

Camada SIP

É nesta camada onde a biblioteca SIP utilizada disponibiliza métodos à camada P2PSIP para a conversão de mensagens P2PSIP-SIP e SIP-P2PSIP. Além disso, disponibiliza outros métodos auxiliares necessários para o *overlay*, como por exemplo, temporização de eventos.

Camada de Transporte

Esta camada, é responsável pelo encapsulamento das mensagens SIP a fim de as transmitir na rede IP. Além de fazer o encapsulamento também armazena uma espécie de identificador para quando receber uma resposta saber a que pedido se refere. Esse identificador, dependendo do tipo de mensagens, também pode ser usado para gerar uma resposta automaticamente.

5.1.2 Camadas referentes à chamada VoIP e encaminhamento de dados

As camadas constituintes deste bloco que contribuem para os seus objetivos e bom funcionamento estão ilustradas na figura 5.2.

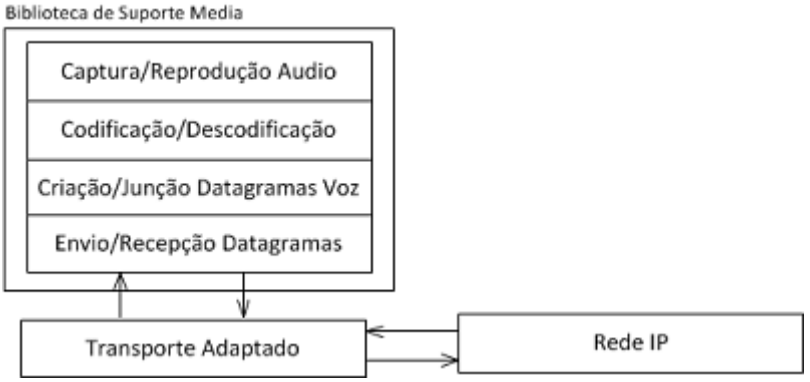


FIGURA 5.2: Camadas referentes à chamada e encaminhamento de dados

Captura/reprodução áudio

Nesta camada faz-se a captura e reprodução do áudio que para isso, utiliza o microfone e os altifalantes do dispositivo.

Codificação/descodificação

Nesta camada os dados são codificados ou descodificados de acordo com o *codec* suportado pelo dispositivo e selecionado no estabelecimento da chamada, com o objetivo de serem transmitidos para o *overlay* ou para serem reproduzidos, dependendo da situação.

Criação/junção de datagramas

Esta camada é a responsável por abrir os canais RTP. Além disso, é aqui que são adicionados os cabeçalhos RTP aos dados codificados na camada superior, isto no caso de transmissão. No caso de receção, é aqui onde os cabeçalhos RTP são analisados e tratados os devidos datagramas no seu respetivo tempo.

Envio/receção de datagramas

É nesta camada onde são abertos os canais UDP, para transmissão na rede dos datagramas criados pela camada superior e pela receção dos dados provenientes da rede IP.

Estas quatro camadas apresentadas são as integrantes da biblioteca escolhida. No entanto, devido ao reencaminhamento de dados e para não modificar os mecanismos internos da biblioteca, foi adicionada uma nova camada denominada de *transporte adaptado* e na camada *envio/receção de datagramas* em vez de transmitir e receber os dados diretamente do meio físico foi forçada uma passagem pelo *transporte adaptado*.

Transporte adaptado

Nesta camada é onde se envia, recebe e reencaminha a *media* na rede IP. Além disso, são recebidas mensagens de outros tipos, como por exemplo *probes*, *pings* ou *warnings* que conforme o seu identificador são realizadas determinadas ações. Também é nesta camada que é calculado o OWD quando recebida a *media*, e ainda, caso esse valor seja superior ao valor referência, a criação e envio de mensagens *warning* também é efetuado nesta camada.

5.1.3 Bibliotecas selecionadas

Para implementação das camadas apresentadas é necessário a utilização de uma API que suporte o SIP e uma que permita a captura/reprodução de áudio, codificação/descodificação, criação e transmissão dos dados no meio físico. A biblioteca SIP escolhida e utilizada foi o JAIN-SIP [28], uma vez que foi esta a utilizada no trabalho anterior [6], mantendo assim a coerência.

Para a captura/reprodução de áudio, codificação/descodificação, criação e transmissão dos dados no meio físico foram analisados e debatidos três potencias *softwares* que são o *JOpenPhone* [29], *Peers* [30] e *MjSIP* [31]. Segue-se uma tabela com as suas principais características:

	Linguagem	Open Source	Última atualização	Ajuda/ Suporte	Em Atualização	Fácil Suporte
JOpenPhone	Java	Sim	04-05-2005	Não	Não	Não
Peers	Java	Sim	07-09-2011	Sim	Sim	Não
MjSIP	Java	Sim	22-04-2012	Sim	Sim	Sim

Tabela 5.1: Tabela de análise às várias APIs

Como podemos verificar na tabela, a linguagem de programação utilizada para a implementação destes *softphones* foi o *java* e os seus códigos-fonte estão disponíveis para qualquer utilizador. Quanto ao *JOpenPhone*, este já não é suportado nem atualizado desde 2005 e além disso os seus blocos para a comunicação não são fáceis de integrar na nossa aplicação. O *softphone Peers* foi atualizado a 01-05-2014, portanto, ainda este ano, porém, o seu código fonte não está disponível para download, logo, a versão analisada foi a versão de 07-09-2011, daí estar na tabela esta data como sendo a de última atualização. Comparando com o JOpenPhone, este é mais atual e além disso é um *software* ainda em desenvolvimento e com suporte. Por ultimo temos o *MjSIP* no qual ainda está em atualização e com suporte, sendo que a sua última versão foi disponibilizada em 22-04-2012, então, dos três *softwares* é o mais atual. Em comparação com o *Peers*, o *MjSIP* tem uma fácil integração, dos blocos que dão suporte à comunicação à nossa implementação, enquanto que para integrar os módulos do *Peers* não é trivial. Posto isto e com a análise feita, os módulos utilizados para a captura/reprodução de áudio, codificação/descodificação, criação e transmissão dos dados na rede IP são os do *MjSIP* e que serão devidamente assinalados mais à frente.

5.2 Implementação das camadas referentes ao *overlay*

Em termos de desenvolvimento, neste trabalho foi implementada a camada de *Aplicação* e tal como referido no capítulo anterior, antes de implementar a aplicação VoIP foi necessário preparar a rede de forma a que a chamada fosse sinalizada. Consequentemente, foram adicionados novos tipos de mensagens e complementadas as camadas *DHT* e *P2PSIP*, incluindo-lhes classes e métodos tal como se descreve de seguida.

5.2.1 CamadaP2PSIP

Esta camada é responsável por efetuar a ligação entre a camada DHT e a *stack* SIP utilizada. Para criar um certo nível de abstração com as camadas superiores, esta camada recorre a um conjunto de classes e interfaces, das quais se destacam, as classes *SipLayer*, *ConvertToSIP*, *ConvertToP2PSIP* e as interfaces *IP2PSIPResponseListener*, *IP2PSIPRequestListener* e *IMessagesStatsListener* do quais as suas interações estão ilustradas na figura 5.3.

A classe *SipLayer* é a classe principal desta camada pois é responsável por comunicar diretamente com a *stack* SIP, para o envio e receção de mensagens. Ao ser inicializada a classe *SipLayer*, é necessário passar uma referência para um objeto que implemente

FIGURA 5.3: Entidades principais da camada *P2PSIP*

a interface *IP2PSIPRequestListener*. Essa referência é necessária para que a camada *P2PSIP* possa notificar a camada *DHT* da chegada de novas mensagens.

Envio de mensagens

Para o envio das mensagens P2PSIP para sinalização da chamada foram disponibilizados os seguintes métodos:

- *public void enviarPedidoP2P(P2PSIPMessage msg, IP2PSIPResponseListner callback)* - este método é utilizado para enviar um pedido P2PSIP. Como parâmetros recebe uma mensagem P2PSIP a enviar, assim como uma instância para o objeto que deverá ser notificado quando for recebida uma resposta para a mensagem enviada.
- *public void enviarRespostaP2P(P2PSIPMessage msg)* - este método é utilizado para enviar um pedido P2PSIP e como parâmetro recebe apenas a mensagem P2PSIP que contém a resposta.

A interface *IP2PSIPResponseListener* é utilizada para que após o envio de um pedido P2PSIP, a sua resposta possa ser entregue a um objeto na camada superior. Esta interface, define os seguintes métodos abstratos:

- *public void onTransFailureResponse(P2PSIPMessage msg)* - executado quando o código de resposta da mensagem SIP tem o valor superior a 299. O único parâmetro que recebe é a mensagem P2P que contém a resposta.
- *public void onTransProvisionalResponse(P2PSIPMessage msg)* - executado quando o código de resposta da mensagem SIP tem um valor entre 100 e 199. O único parâmetro que recebe é a mensagem P2P que contém a resposta.
- *public void onTransSuccessResponse(P2PSIPMessage msg)* - executado quando o código de resposta da mensagem SIP tem um valor entre 200 e 299 . O único parâmetro que recebe é a mensagem P2P que contém a resposta.

- public void onTransTimeout() - executado quando o envio de uma mensagem P2PSIP resulta em *timeout*.

A utilização de uma interface para notificar a camada superior da chegada de uma resposta, permite uma melhor distinção entre as camadas. Desta forma, no envio de um pedido P2PSIP, esta camada requer apenas a instância de um objeto do *listener*, que pode ser de qualquer tipo, necessitando apenas de implementar a interface especificada. A camada P2PSIP é apenas responsável por notificar o objeto em questão da ocorrência de um desses eventos. A implementação dos métodos definidos por esta interface depende apenas e é da responsabilidade da camada superior.

Os métodos disponíveis para o envio de mensagens P2PSIP recorrem à classe *ConvertToSIP* para efetuar a conversão do objeto *P2PSIPMessage* (que contem a informação da mensagem P2PSIP a enviar) num objeto SIP reconhecido pela biblioteca utilizada, neste caso o JAIN-SIP.

Receção de mensagens

A receção de mensagens SIP é feita de diferentes formas, consoante a mensagem recebida seja um pedido, ou uma resposta a um pedido enviado anteriormente. Em ambos os casos, é verificado se a mensagem recebida, é uma mensagem válida. Esta verificação é feita através da análise dos cabeçalhos da mensagem recebida que após a sua validação é feita a conversão da mensagem SIP para P2PSIP recorrendo à classe *ConvertToP2PSIP*.

No caso de a mensagem recebida ser um pedido, esta é entregue na camada *DHT*, notificando-a através da interface *IP2PSIPRequestListener* de modo a ser processada. Caso seja uma resposta a um pedido enviado anteriormente, o código da resposta da mensagem SIP é analisado e consoante o seu valor, a notificação enviada para a camada *DHT* difere. Esta notificação é feita através da interface *IP2PSIPResponseListener* do objeto de *callback* referenciado no envio do pedido efetuado.

5.2.2 Camada DHT

A base desta camada é formada por um conjunto de classes e interfaces, que podem ser expandidas por cada implementação de um algoritmo DHT a utilizar. Foi criada na classe *Peer* (que representa um *peer* genérico) a implementação de um conjunto de métodos comuns a qualquer *peer* independentemente da DHT utilizada, nomeadamente os métodos *put, get, invite, ack* e *bye*. Na figura 5.4 estão ilustradas as principais classes desta camada.

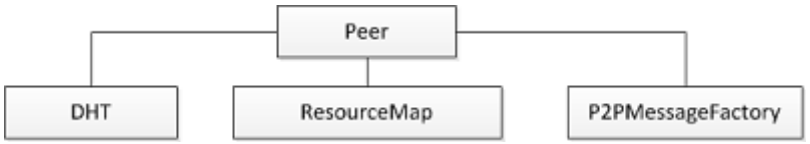


FIGURA 5.4: Entidades principais da camada *DHT*

A classe DHT desta camada, é uma classe genérica, com métodos abstratos e que foi expandida na implementação de algoritmos DHT. O objetivo é permitir que todos os algoritmos DHT utilizem classes derivadas desta para representar informação relativa ao algoritmo DHT. A informação contida nesta classe identifica o nome do algoritmo DHT, assim como o nome do algoritmo utilizado para calcular o hash dos identificadores.

A gestão de recursos é algo comum a qualquer *peer* pertencente a um *overlay*, independentemente do algoritmo em utilização. Por isso, na camada *DHT* foram implementadas as classes *Resource* e *ResourceMap* para a gestão de recursos. A classe *Resource* é utilizada para guardar informação relativa a um recurso, nomeadamente o identificador, nome, valor associado ao recurso e o tempo ao fim do qual este expira.

A criação de mensagens P2PSIP, é uma funcionalidade comum independente da DHT utilizada, logo, e para evitar a repetição de código, foi criada a classe *P2PMessageFactory*. Esta classe possui um conjunto de métodos que permitem a criação de diferentes tipos de mensagem P2PSIP de sinalização da chamada. Assim, foram adicionados métodos para a criação dos diferente tipos de mensagens de sinalização, nomeadamente o *INVITE*, *ACK* e *BYE*. Seguem-se os métodos implementados e propósito de cada um deles:

- *constructInviteRequest(String destination, String username)* - este método é responsável pela construção da mensagem P2PSIP contendo o INVITE para o URL do utilizador que se quer contactar. Recebe como parâmetros o IP e porta contidos numa *String* e o *username* do utilizador a contactar.
- *ConstructInviteRequestResponse(int code, P2PSIPMessage req)* - este método gera uma resposta a um INVITE recebido de um determinado utilizador. Recebe como parâmetros o código de resposta e a mensagem p2psip que contém os dados referentes ao INVITE.
- *constructAck(String destination, String username)* - este método é o responsável por sinalizar que recebeu uma confirmação positiva ao INVITE efetuado. Recebe no primeiro parâmetro o IP e porta e o *username* de quem tem a conexão estabelecida no segundo parâmetro.

- *constructBye(String destination, String username)* - este método é o responsável por sinalizar o fim de chamada de uma das partes envolvidas na chamada. Recebe no primeiro parâmetro o IP e porta e o *username* com quem tem a conexão estabelecida no segundo parâmetro.

- *constructByeResponse(int code, P2PSIPMessage req)* - este método gera a resposta ao BYE proveniente do parceiro de sessão. Recebe como parâmetros o código de resposta e a mensagem P2PSIP que contém os dados referentes ao BYE.

5.2.3 Camada Aplicação

Esta camada é a responsável pela interação com o utilizador em que este fornece a informações necessárias para se juntar ao *overlay*, efetuar, receber chamadas e definir qual DHT utilizar (útil no ambiente de teste). Para isso foram desenvolvidas várias classes: *login*, *principal*, *IncomingCall* e *MakeCall*. Na figura 5.5 estão ilustradas as classes e a interação entre elas.

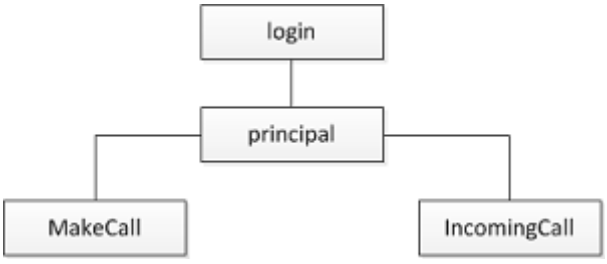


FIGURA 5.5: Entidades principais da camada *Aplicação*

A classe *login* é a primeira interação com o utilizador em que este deve fornecer qual o seu *username* que será o seu identificador dentro de um determinado domínio, qual o seu IP e porta no qual o agente SIP vai estar a executar, qual a DHT (Chord ou EpiChord) e qual o *bootstrap peer* que o vai ajudar a entrar no *overlay*. Após se juntar à rede a classe que irá interagir com o utilizador é a denominada por *principal* na qual poderá efetuar ou receber chamadas.

Esta camada está posicionada acima da camada DHT. Quando o utilizador quer efetuar uma chamada notifica a camada DHT com o URI que quer contactar e esta trata de o notificar quando obtida uma resposta. Se o URI for obtido, após pesquisa no *overlay*, é gerada uma mensagem de INVITE para o tal destino. Esta operação é feita na classe *MakeCall*. Quando recebida uma chamada, a camada *DHT* notifica a camada *Aplicação* e esta notifica o utilizador através da classe *IncomingCall*. O utilizador pode aceitar ou rejeitar a chamada e conforme a operação é gerada uma resposta que é devolvida para quem o pretende contactar. O código para resposta quando a chamada é

aceite é o 200 e 486 no caso da chamada ser rejeitada ou se já existe uma comunicação ativa com outro *peer*. É também nesta camada que é feita a captura/reprodução de áudio, codificação/descodificação, criação e transmissão dos dados no meio físico após dois utilizadores iniciarem uma sessão. Para tal, foram incluídos sete *packages* do *MjSIP* que são o *org.zoolu.net, org.zoolu.sdp, org.zoolu.sound, org.zoolu.tools, org.zoolu.ua, voip.local.media e voio.local.net* os quais se sofrerem alterações ou atualizações futuras para os incluir nesta implementação bastará apenas incluí-los, sem qualquer modificação adicional.

Após a chamada ser aceite, tanto no emissor como no recetor, é chamado o método *startMediaSessions()* que inicia o sistema de captura/reprodução, codificação/descodificação, criação e transmissão dos dados no meio físico conforme os parâmetros (IP e porta RTP, *codecs* suportados, etc) acordados no estabelecimento da chamada através do protocolo SDP. Aquando algum dos *peers* pretender terminar a comunicação, notifica a camada DHT e esta gera a mensagem de BYE. Quando obtida a resposta, esta camada é notificada e ambos os *peers* utilizam o método *stopMediaSessions()* a fim de fechar os vários *sockets* utilizados. A classe denominada de*principal* é apresentada de novo ao utilizador, para possibilitar novas interações.

5.3 Implementação da camada de transporte adaptada

Anteriormente foi apresentada a organização das camadas referentes à chamada e encaminhamento de dados. Consequentemente a implementar o algoritmo de encaminhamento apresentado no capítulo anterior foi necessário acrescentar uma camada na API que dá suporte à comunicação. Essa camada situa-se entre a camada de transporte do *MjSIP* e a rede IP. A esta camada chamamos de *transporte adaptado* e tem como função receber datagramas de voz do *MJSip* e enviá-los para o próximo nó conforme a tabela de encaminhamento. Além disso também recebe datagramas da rede IP tratando-os conforme o seu tipo.

Tipos de mensagens

Para dar suporte às funcionalidades do algoritmo de encaminhamento, fazer a medição da qualidade da chamada e manter a tabela de distancias atualizada em cada um dos *peers*, criaram-se vários tipos de mensagens devidamente identificadas para cada um dos propósitos, sendo elas:

- *warning* - contem apenas um *byte* com um 'W' que o identifica, é gerado pelo recetor e tem como propósito alertar o emissor que o atraso fim-a-fim da ligação excedeu os 150ms.

- *probe* - o primeiro campo é o seu identificador 'P' com tamanho de um *byte*, o segundo campo 'N' tem o tamanho de um *byte* e tem como função indicar a profundidade máxima que um *peer* pode fazer *probe*, 4 *bytes* para o endereço IP destino e por ultimo 2 *bytes* para o RTT acumulado. Estes datagramas tem o propósito de pedir a um *peer* vizinho para verificar qual o RTT da sua conexão direta para um determinado destino.

- *response* - o primeiro *byte* é o identificador 'R', 4 *bytes* para o endereço IP destino pedido e 2 *bytes* para o RTT Total desde o emissor até ao recetor. Este tipo de datagrama tem como objetivo responder ao *peer* que fez um *probe* caso uma série condições tenham sido cumpridas e que vão ser devidamente apresentadas e explicadas posteriormente.

- *ping* - o primeiro *byte* é o identificador 'P' e o segundo *byte* é o tipo de ping, isto é, *stateless* ou *statefull*. Este datagrama tem como função fazer um pedido de uma mensagem de *echo* a um determinado *peer* de forma a calcular o RTT.

- *echo* - o primeiro campo tem o tamanho de um *byte* no qual se encontra o identificador 'E' e o segundo campo '*Timestamp*' tem o tamanho de 8 *bytes* a fim de auxiliar o cálculo do RTT no recetor desta mensagem.

- *voice* - o primeiro *byte* é o identificador "V", o segundo campo tem o tamanho de 4 *bytes* no qual se encontra o IPDestino, o terceiro campo contém o *timestamp* [1] em que o datagrama de voz foi enviado no emissor(permite calcular o OWD no recetor) e tem o tamanho de 8 *bytes*. Os últimos dois campos deste datagrama tem o tamanho de 332 *bytes* (dependendo do *codec* utilizado, neste caso foi o G.711) e contêm a *media* com os respetivos cabeçalhos RTP.

5.3.1 Reencaminhamento de dados

Cada *peer* tem presentes várias tabelas, nomeadamente, uma tabela de encaminhamento para saber qual o próximo nó para um determinado endereço e uma tabela de distâncias que contem uma lista de *peers* vizinhos e o respetivo RTT até eles. Além disso, tem uma tabela de *pings* que é necessária para auxiliar o cálculo do RTT até os vizinhos. Esta tabela armazena o *timestamp* inicial e o endereço destino. Cada *peer* tem também

[1] embora o *timestamp* esteja no RTP, este é um *timestamp* relativo que e necessário para o processamento RTP, o qual não é o pretendido para o calculo do OWD

uma tabela de respostas para saber a que *peer* deve responder quando recebe um *probe* ou aquando de uma resposta de um outro *peer*.

De modo a ser possível o reencaminhamento de dados foram implementadas as classes *ExternalRelay*, *InternalRelay*, *melhorCaminho*, *StatefulPing*, *fazping*, *StatelessPing* e *ReceivePingResponse* nas quais as suas interações estão ilustradas na figura 5.6.

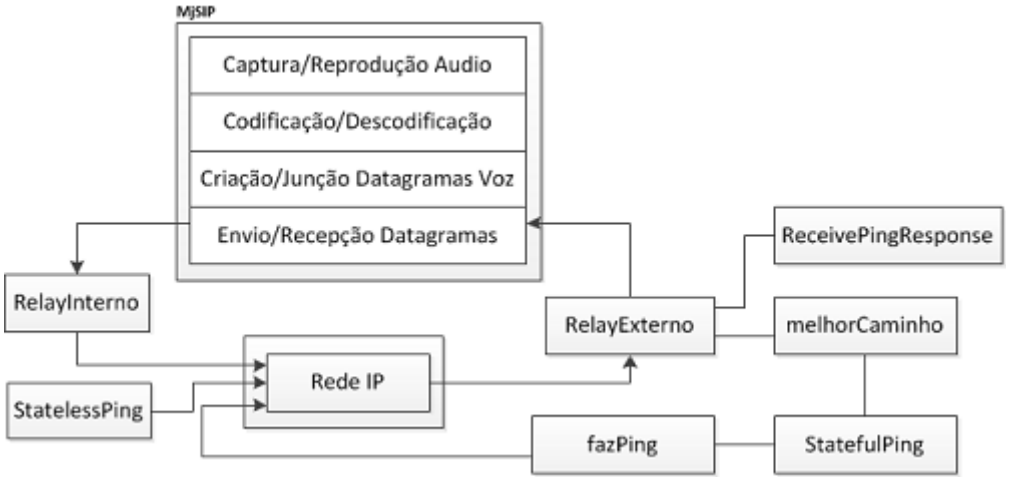


FIGURA 5.6: Entidades principais na implementação do reencaminhamento de dados

Receção de dados - *ExternalRelay*

Esta classe é uma *thread* invocada pela classe *principal* quando iniciada e tem como função receber datagramas da rede IP e tratá-los conforme o seu tipo. Para se ligar à rede IP é criado um *socket* na porta 50001 na qual capta os vários tipos de mensagens. Quando recebida uma mensagem, é analisado o primeiro *byte* que é onde se situa o seu identificador e caso este seja do tipo 'P', verifica se é do tipo *ping* ou do tipo *probe*. Se a mensagem for do tipo *probe*, é criada uma nova instância da classe *melhorCaminho* (que está preparada para tratar este tipo de mensagens) e iniciado o algoritmo de verificação de potencial encaminhador, senão verifica no segundo *byte* da mensagem se é um *ping* do tipo *stateless* ou *statefull*, constrói a mensagem de *echo* e envia-a para o endereço IP que originou a mensagem *ping*. No ponto de vista do *ExternalRelay* a única diferença entre ser um *ping* do tipo *stateless* ou *statefull* é o número da porta para qual a mensagem *echo* é enviada (porta 50001 para *stateless pings* e 50003 para *stateful pings*). Nesta classe quando são recebidos *echos* é criada uma nova instância da classe *ReceivePingResponse* passando-lhe o datagrama recebido como argumento.

Quando recebida uma mensagem do tipo *warning* significa que o recetor detetou que o OWD quebrou a barreira de 150ms. Então deve ser encontrado um caminho alternativo

para a *media* transitar. Para isso, é criada uma nova instância da classe *melhorCaminho*, que é a classe responsável por tratar este tipo de mensagens, passando-lhe como argumento a mensagem recebida.

Caso a mensagem seja do tipo *response*, retira-se o endereço IP e RTT total lá contidos e verifica-se se aquele endereço já existe na tabela de encaminhamento. Se existir, verifica-se se o RTT total recebido é inferior ao RTT contido na tabela e caso seja, o próximo nó passa a ser o *peer* cujo endereço IP é o que originou a mensagem *response*. Se o endereço IP contido na mensagem não existir, é adicionada uma nova entrada na tabela com esse endereço, tendo como próximo nó o endereço do *peer* que originou a mensagem de *response* e com o RTT total extraído da mensagem.

A última verificação feita é se o datagrama recebido é do tipo *voice* e se entrar nesta condição, a próxima ação é extrair o endereço IP lá contido e comparar com o endereço local do *peer*. Caso não coincidam significa que não é para ele a mensagem, verifica-se na tabela de encaminhamento se é conhecido o próximo nó para esse endereço, reencaminhando a mensagem. Caso o endereço IP coincida com o endereço local, é extraído o *timestamp* da mensagem, subtrai-se o *timestamp* atual ao obtido da mensagem e compara-se se o valor obtido é inferior a 150ms e caso esta condição se verifique retiram-se os cabeçalhos do algoritmo de reencaminhamento e passa-se o datagrama RTP para a camada superior para posterior reprodução. Caso contrário é construída uma mensagem de *warning* e enviada para o endereço IP no qual se encontra o *peer* que estabeleceu a chamada. Para melhor compreensão foi criado um fluxograma geral e que está ilustrado na figura 5.7. O cálculo do OWD foi efetuado desta forma em vez de usar as potencialidades que o RTCP oferece [9] devido à biblioteca adotada não implementar este protocolo, logo, esta foi uma forma simples de contornar o problema. Requer porém um sincronismo de relógio em todos os *peers*.

Procura de caminho alternativo - *melhorCaminho*

A classe *melhorCaminho* tem como função tratar mensagens do tipo *warning* ou *probe* a fim de encontrar caminhos alternativos (no caso de receber um *warning*) ou testar se é um potencial encaminhador para um determinado endereço IP destino. Então, esta classe é criada quando a classe *ExternalRelay* recebe um destes dois tipos de mensagens. O algoritmo começa por verificar o primeiro *byte* da mensagem a fim de saber se é do tipo *warning* ou do tipo *probe*. Se for do tipo *warning*, significa que o recetor detetou uma má qualidade de chamada, então, o emissor para se certificar faz uma medição do RTT utilizando os *stateful pings*, que foram implementados na classe *StatefulPing*, e caso o RTT exceda os 300ms é chamado o método *ProbeVizinhos*. No caso de ser um *probe*, é extraído o endereço para qual deva testar a ligação, qual o RTT acumulado e

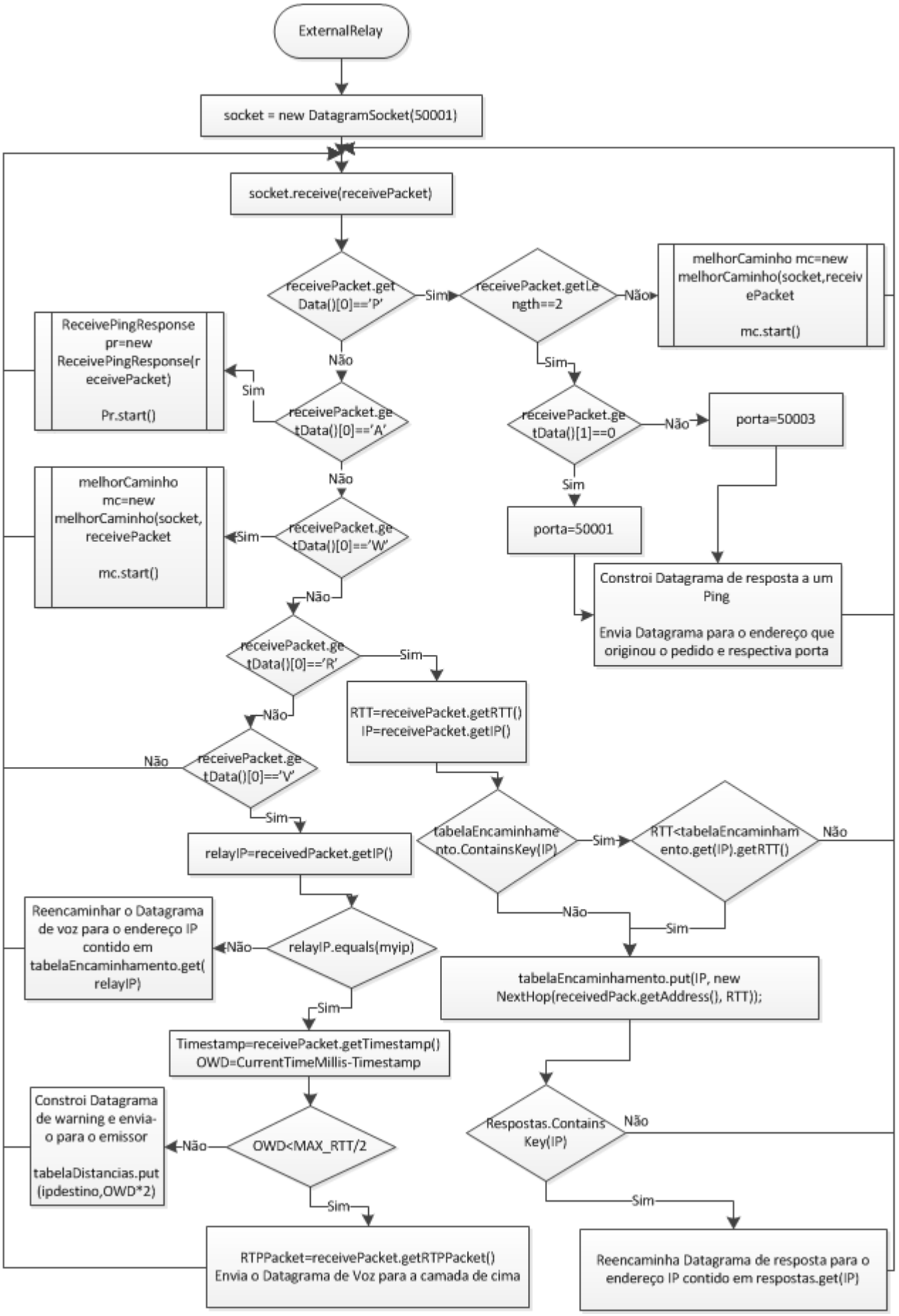


FIGURA 5.7: Fluxograma da classe *ExternalRelay*

qual o número de saltos possíveis. O primeiro passo é verificar se o endereço IP destino é um vizinho conhecido e caso seja não é necessário voltar medir a ligação uma vez que essa informação é constantemente atualizada. Caso não se encontre na tabela de vizinhos, faz-se uma medição da ligação com auxilio da classe *StatefulPing*. Somando o RTT obtido com o RTT acumulado, se este for inferior a 300ms, significa que é um potencial reencaminhador para a *media*, então, adiciona uma nova entrada à tabela de encaminhamento e notifica o *peer* que lhe enviou a mensagem através do método *RespostaProbe*. Caso o RTT total exceda os 300ms, envia mensagens de *probe* para vizinhos através do método *probeVizinho*. De forma a entender melhor o algoritmo, na figura 5.8 está ilustrado o fluxograma com o seu funcionamento.

Os métodos *probeVizinhos* e *respostaProbe* foram implementados como auxilio, de forma a não existir código repetido dentro da classe. Segue-se o objetivo de cada um deles, respetivas implementações e funcionamento.

- *public void probeVizinhos(int RTTAcumulado, InetAddress ipdestino, int saltos)* - este método tem como função construir mensagens do tipo *probe* com os devidos campos e enviá-la para vizinhos que são vistos como potenciais encaminhadores da *media*. Recebe como argumento o RTT acumulado até ao momento, o endereço IP destino e o número de saltos que a mensagem *probe* pode ser propagada. O método começa por confirmar que o número de saltos é superior a zero. Depois, são escolhidos aleatoriamente cinco vizinhos (definimos um máximo de cinco vizinhos para não gerar demasiados *probes*), para os quais a mensagem *probe* será enviada. Permite-se assim que para cada *probe* que se tenha de efetuar não sejam sempre os mesmos cinco vizinhos os escolhidos. Para gerar os números aleatórios foi utilizada uma classe nativa do *java* denominada por *Random*, em que cada número gerado tem de ser diferente dos anteriores obtidos, como a classe *Random* não tem essa funcionalidade implementada foi criada uma lista de forma a poder fazer a comparação entre o número gerado e os números já obtidos. Quando obtido o número, compara-se na tabela de distâncias qual o correspondente, verifica-se se o somatório do RTT acumulado com o RTT até ao vizinho é inferior a 300ms e caso seja, a mensagem *probe* é enviada para ele, pois, sendo este somatório inferior a 300ms significa que o vizinho ainda tem a hipótese de atingir o destino com o RTT dentro do limite definido. Na figura 5.9 está ilustrado o fluxograma com o funcionamento deste método.

- *private void respostaProbe(int rttTotal, InetAddress ipdestino)* - este método tem como função construir a mensagem do tipo *response* com os devidos campos e enviá-la para o endereço IP contido na tabela de respostas correspondente a um determinado endereço IP destino. O seu fluxograma ilustra-se na figura 5.10

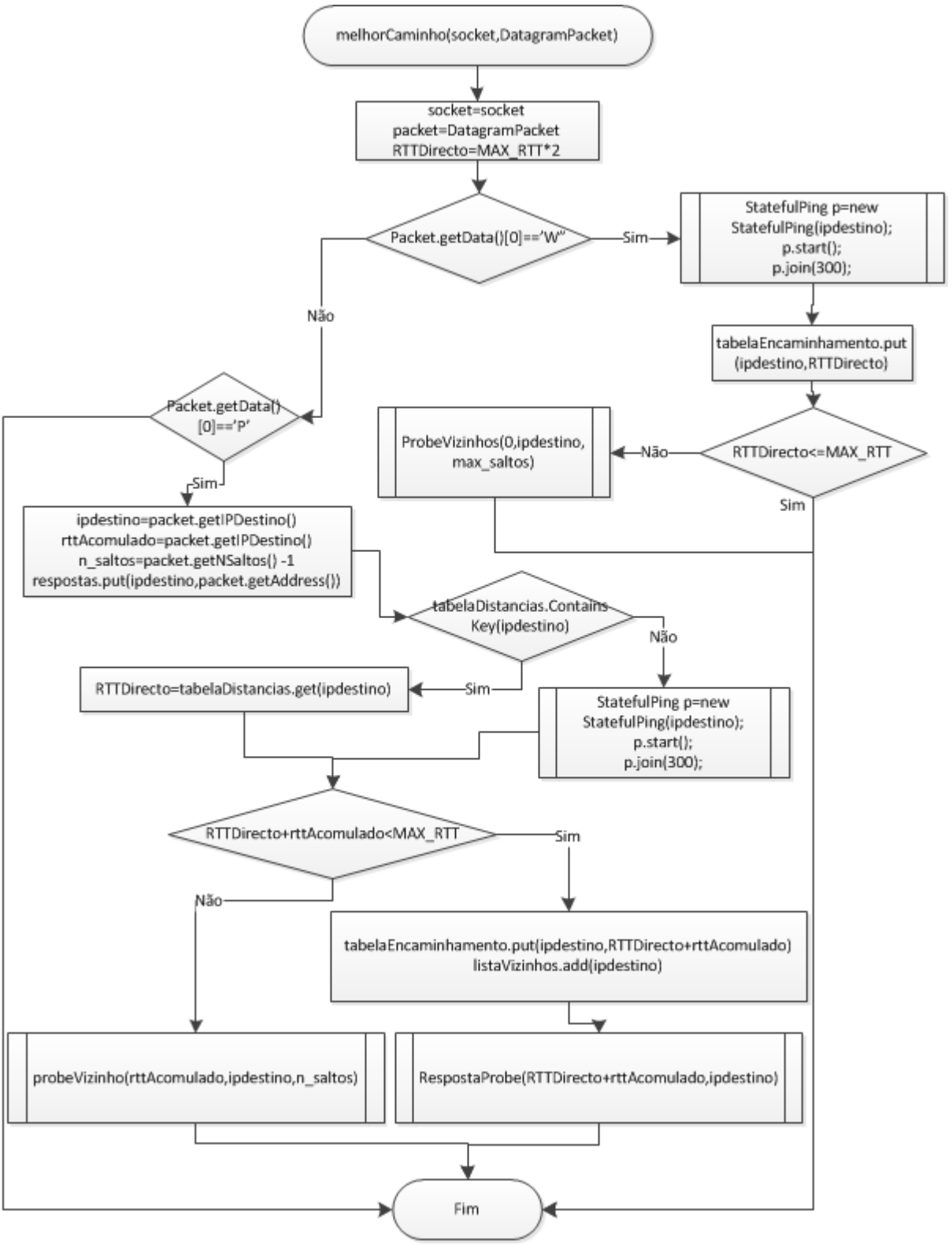


FIGURA 5.8: Fluxograma da classe *melhorCaminho*

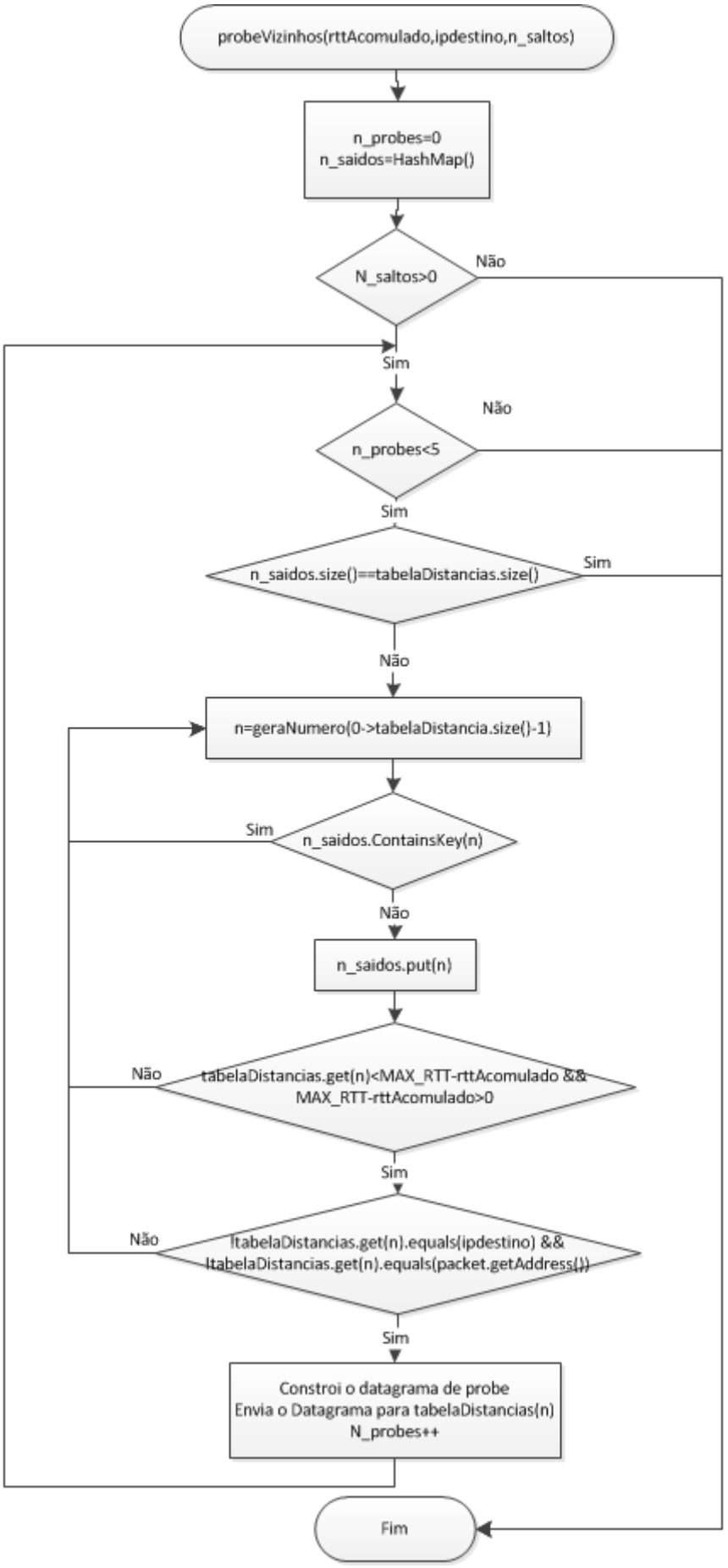


FIGURA 5.9: Fluxograma do método *probeVizinhos* presente na classe *melhorCaminho*

FIGURA 5.10: Fluxograma do método *respostaProbe* presente na classe *melhorCaminho*

Medição do RTT - *StatelessPing* e *StatefulPing*

Para efetuar a medição do RTT de uma determinada ligação foram criados dois tipos de pings, tal como anteriormente explicado. Foram então criadas duas classes para tal, denominadas *StatelessPing* e *StatefulPing*.

A classe *StatelessPing* tem como objetivo enviar datagramas do tipo *ping* periodicamente para os *peers* vizinhos. Esta classe é uma *thread* instanciada pela classe *principal* quando iniciada, portanto está sempre em funcionamento, mesmo quando o utilizador não se encontra numa chamada. A *thread* começa por abrir um *socket* na porta 50002 para o envio das mensagens, depois percorre a lista de vizinhos e para cada um deles cria a mensagem *ping* como sendo do tipo zero. Isto porque as mensagens de *echo* para este tipo de *pings* têm de ir para a porta 50001, uma vez que é onde se encontra a *thread ExternalRelay* que instancia a classe que calcula o RTT com base no *echo* recebido. Após enviar a mensagem, guarda na tabela de *pings* qual endereço IP para quem enviou e qual o *timestamp* atual. Na figura 5.11 está ilustrado o fluxograma com o seu funcionamento.

Tal como referido, a classe que trata das mensagens *echo* para este tipo de *pings* é instanciada pela classe *ExternalRelay* e é denominada por *ReceivePingResponse*. Esta classe é uma *thread* e por cada *echo* recebido é instanciada esta classe, portanto uma *thread* por cada *echo*. A classe começa por verificar se o endereço IP que originou a mensagem *ping* existe na tabela de *pings* e caso exista, para obter o RTT, subtrai o *timestamp* contido na mensagem *echo* ao *timestamp* armazenado na tabela de *pings* (assim obtém o OWD) e multiplica por dois. Obtido o RTT, atualiza na tabela de distâncias o RTT para aquele vizinho e remove da tabela de *pings* a entrada correspondente ao pedido que foi tratado. Na figura 5.12 está ilustrado o fluxograma com o seu funcionamento.

A classe *StatefulPing* é uma *thread* instanciada pela classe *melhorCaminho* quando esta precisa de saber qual o RTT de uma ligação instantaneamente. De forma a ser possível que mais que um *StatefulPing* efetue pedidos *ping* em simultâneo num só *socket*, foi

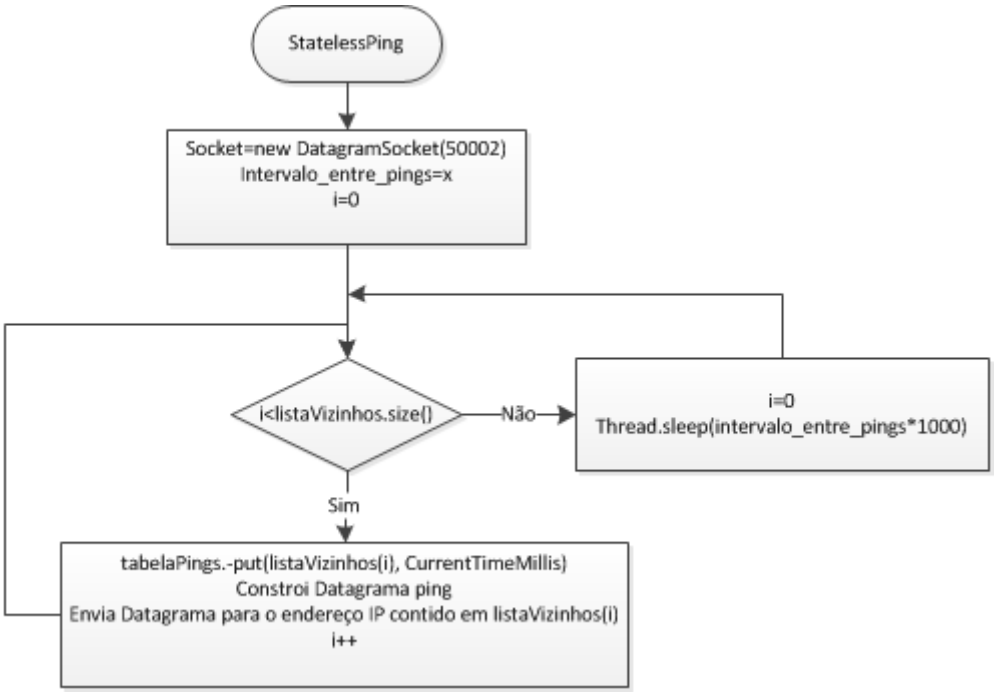


FIGURA 5.11: Fluxograma da classe *StatelessPing*

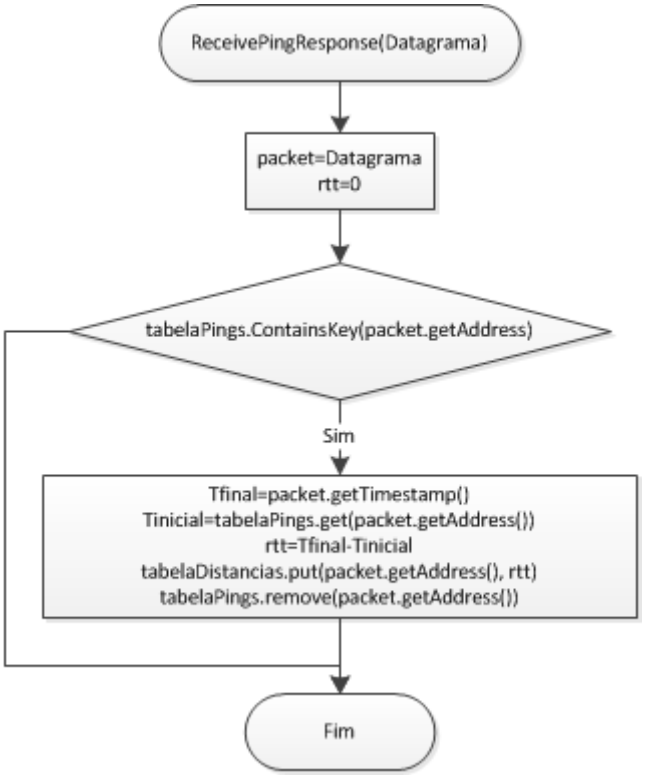


FIGURA 5.12: Fluxograma da classe *ReceivePingResponse*

necessário implementar uma classe denominada por *fazping* na qual está à escuta na porta 50003 e todos os pedidos de *ping* provenientes das várias *threads* são feitos de forma sincronizada, sequencialmente. Tal é necessário para não tornar o comportamento do sistema imprevisível. Como objetivo geral, a classe *fazping* tem de enviar um *ping*, esperar a respetiva mensagem de *echo*, calcular o RTT, repetir o processo n vezes e fazer a média do RTT tendo em conta o número de amostras obtidas, assim se obtém uma medição com maior precisão. Para melhor compreensão, nas figuras 5.13 e 5.14 estão ilustrados os fluxogramas da sua implementação.

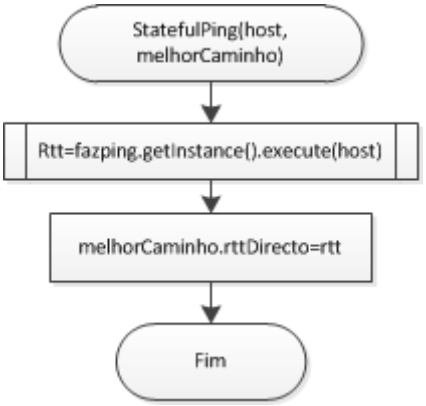


FIGURA 5.13: Fluxograma da classe *StatefulPing*

Envio de *media* - *InternalRelay*

Esta classe é uma *thread* instanciada quando é feita o estabelecimento de uma chamada e tem como objetivo receber a *media* provenientes do *MjSIP*, acrescentar-lhe os cabeçalhos necessários para o reencaminhamento de dados e cálculo do OWD no recetor, e enviar para o próximo *peer* de acordo com a tabela de encaminhamento. Para isso, a classe começa por abrir um *socket* na porta 50000 no qual fica à espera de receber *media*, depois adiciona-lhe o endereço IP do *peer* a que se destina a mensagem e o *timestamp* atual, procura na tabela de encaminhamento qual o próximo nó e caso o RTT total contido na tabela seja inferior a 300ms a mensagem é enviada, caso contrário e até que o RTT seja modificado mais nenhum datagrama será enviado. Na figura 5.15 está ilustrado o fluxograma com o seu funcionamento.

Considerações finais

Existem parâmetros que é necessário ajustar e que só após com a devida avaliação se poderão definir de forma correta, como por exemplo, o número de *pings*, tempos de espera e número de saltos.

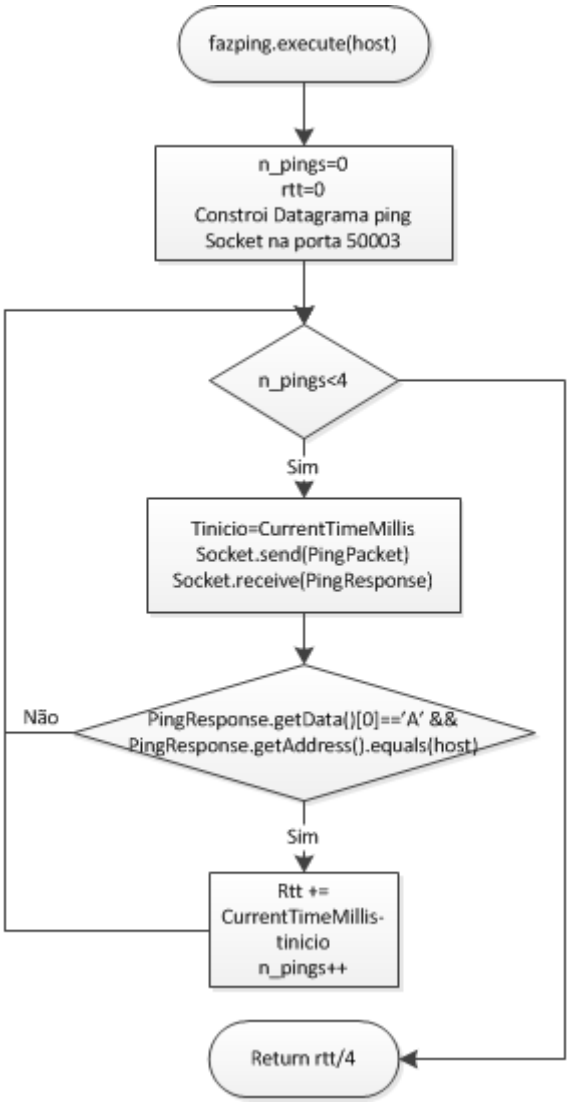


FIGURA 5.14: Fluxograma da classe *fazping*

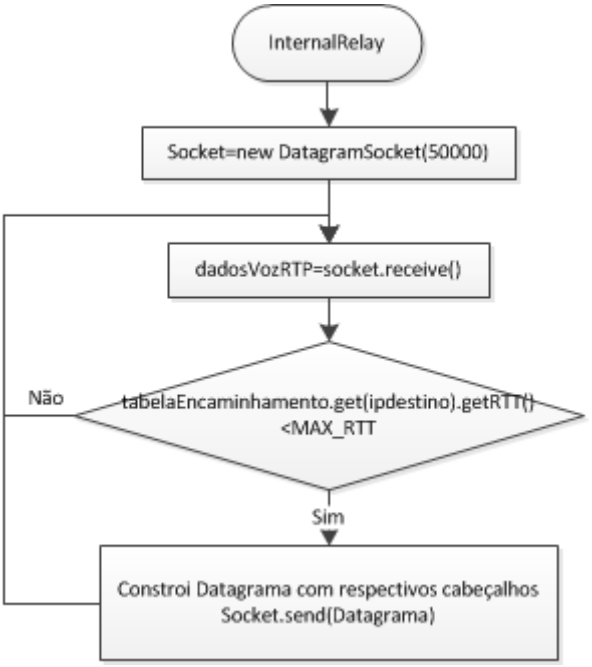


FIGURA 5.15: Fluxograma da classe *InternalRelay*

Capítulo 6

Testes e Resultados

Os testes efetuados focam-se no desempenho e validação do algoritmo de encaminhamento da *media* em situações que o volume de tráfego entre dois pontos é muito grande. Os testes ao *software* foram efetuados numa rede emulada, onde foram desenhados cenários reais, com testes de carga muito exigentes.

6.1 Ambiente de teste

Os testes foram feitos numa máquina na qual se utilizou o Common Open Research Emulator (CORE) [32] que é uma ferramenta que permite emular redes, redes essas se que podem ligar a outras redes emuladas e/ou redes reais. Os testes efetuados no CORE, serviram para obter os parâmetros das métricas consideradas e validar a implementação do algoritmo e das restantes componentes desenvolvidas. Foi criada uma topologia composta por 4 *Routers*, 4 *Switches* e 30 *Hosts*. A figura 6.1 mostra como os vários componentes se encontram ligados.

Os testes efetuados ao *software* dividem-se em dois universos distintos, testes de funcionalidade e testes de desempenho. Nos testes de funcionalidade, foram arquitetados os seguintes três cenários:

- Chamada direta - foi simulada uma chamada entre dois *hosts* pertencentes à rede P2P, em que um envia o *INVITE* e outro aceita a chamada automaticamente, com o objetivo de testar a comunicação.
- Chamada com encaminhamento da *media* num *peer* intermédio - foram simuladas cinco chamadas em simultâneo entre os *peers* ligados ao *router* n1 e os *peers* ligados ao *router* n2, em que a largura de banda da ligação direta entre n1 e n2 foi de

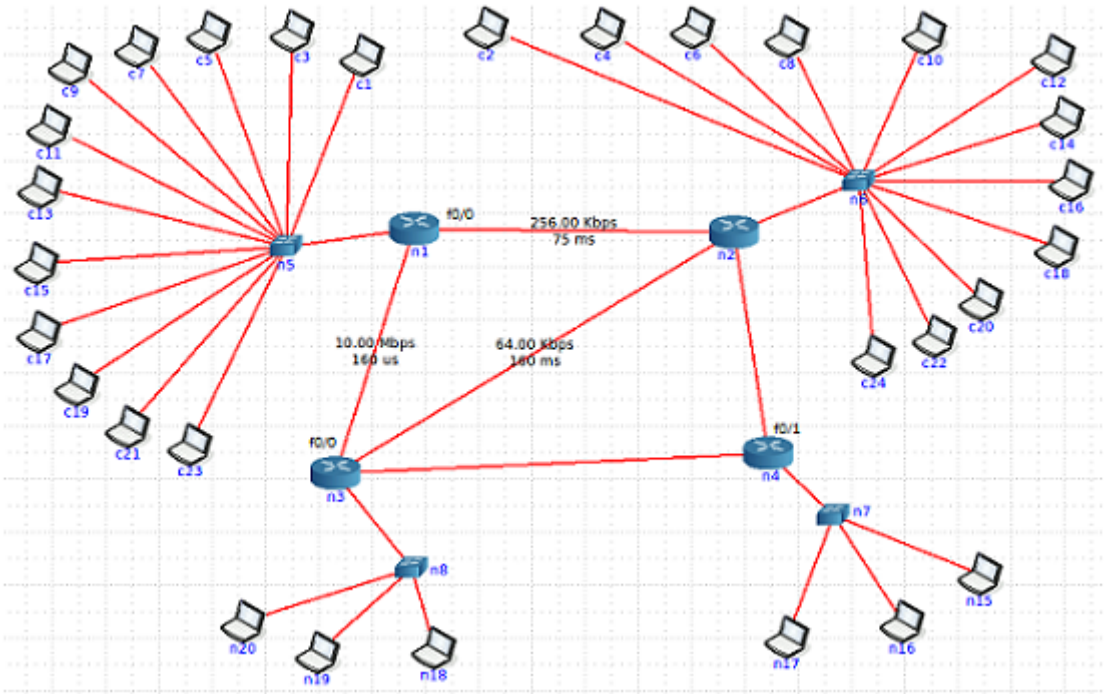


FIGURA 6.1: Topologia Teste

256kbps com um atraso de propagação (*delay* 75ms) resultando num RTT de 150ms. As restantes ligações entre os *routers* tiveram uma largura de banda de 10Mbps com um atraso de propagação (160μs) resultando num RTT de 320μs.

- Chamada com encaminhamento da *media* em n *peers* intermédios - foram simuladas cinco chamadas entre os *peers* ligados ao *router* n1 e os *peers* ligados ao *router* n2 em simultâneo, em que a largura de banda da ligação direta entre n1 e n2 foi de 256kbps com um atraso de propagação (*delay* 75ms) resultando num RTT de 150ms e a ligação entre n2 e n3 uma largura de banda de 64kbps com um atraso de propagação (*delay* 160ms) resultando num RTT de 320ms. Este cenário de teste está representado na figura 6.1.

Nos testes de desempenho, foram obtidos valores para as métricas consideradas em três diferentes cenários:

- Chamadas diretas - foram feitas tentativas de até 12 chamadas em simultâneo entre os *peers* ligados ao *router* n1 e os *peers* ligados ao *router* n2, em que a largura de banda da ligação direta entre n1 e n2 foi de 256kbps com um atraso de propagação (*delay* 75ms) resultando num RTT de 150ms e com largura de banda de 512kbps com atraso de propagação (*delay* 50ms) resultando num RTT de 100ms.

- Chamada com encaminhamento da *media* num *peer* intermédio - foram feitas até 12 chamadas em simultâneo entre os *peers* ligados ao *router* n1 e os *peers* ligados ao *router* n2, em que a largura de banda da ligação direta entre n1 e n2 foi de 256kbps com um atraso de propagação (*delay* 75ms) resultando num RTT de 150ms e com largura de banda de 512kbps com atraso de propagação (*delay* 50ms) resultando num RTT de 100ms. As restantes ligações entre os *routers* tiveram uma largura de banda de 10Mbps com um atraso de propagação (160μs) resultando num RTT de 320μs.

- Chamada com encaminhamento da *media* em n *peers* intermédios - foram feitas até 12 chamadas em simultâneo entre os *peers* ligados ao *router* n1 e os *peers* ligados ao *router* n2, em que a largura de banda da ligação direta entre n1 e n2 foi de 256kbps com um atraso de propagação (*delay* 75ms) resultando num RTT de 150ms e com largura de banda de 512kbps com atraso de propagação (*delay* 50ms) resultando num RTT de 100ms. A ligação entre n2 e n3 teve uma largura de banda de 64kbps com um atraso de propagação (*delay* 160ms) resultando num RTT de 320ms.

Para melhor compreensão segue-se uma tabela detalhada com as características das ligações para cada um dos testes efetuados:

	L.B. n1-n2	RTT n1-n2	L.B. n2-n3	RTT n2-n3	L.B. n3-n4-n2	Número de chamadas(simultâneo)
Directo (funcionalidade)	ilimitada	zero	-	-	-	1
Directo (desempenho)	256kbps 512kbps	150ms 100ms	-	-	-	Máximo permitido pela L.B
Um peer intermédio N peers intermédios (funcionalidade)	256kbps 256kbps	150ms 150ms	10Mbps 64kbps	320μs 320ms	- ilimitado	5 5
Um peer intermédio (desempenho)	256kbps 512kbps	150ms 100ms	10Mbps 10Mbps	320μs 320μs	- -	Aumento sucessivo de 1 a 12 chamadas
N peers intermédios (funcionalidade)	256kbps 512kbps	150ms 100ms	64kbps 64kbps	320ms 320ms	ilimitado ilimitado	Aumento sucessivo de 1 a 12 chamadas

Tabela 6.1: Tabela com vários testes efetuados

Para a execução da aplicação desenvolvida nesta topologia, uma vez que as chamadas tinham de ser executadas numa linha de comandos e de forma automática a aplicação gráfica com interação com o utilizador teve de ser modificada para receber os parâmetros de configuração como argumentos e sem interação gráfica. A inicialização dos *peers* e das chamadas VoIP em cada um dos testes foi efetuada recorrendo a um *script* desenvolvido no qual inicia a aplicação JAVA em cada um dos *peers* com os vários parâmetros de acordo com o *host* onde vai ser executado e com o modo que deve iniciar (direto, um *peer*

intermédio ou n *peers* intermédios). As aplicações JAVA recebem vários parâmetros que permitem configurar os vizinhos, ficheiros de saída, URI, *peer* a contactar entre outros.

Os testes foram efetuados numa máquina com um processador *intel i7* 1.6GHz com 6Gb de RAM, o que limitou a 12 o número máximo de chamadas em simultâneo (testado na prática, por aumento sucessivo do número de chamadas até deixar de funcionar).

6.2 Métricas

Os parâmetros analisados foram o número total de datagramas voz recebidos, o total de datagramas com OWD menor que 150ms, o RTT médio referente ao número de datagramas recebidos e o RTT médio referente ao número de datagramas com OWD menor que 150ms. Todos eles foram medidos no recetor. O número de datagramas recebidos com OWD menor que 150ms auxiliaram o cálculo da taxa de transferência e, bem como os restantes parâmetros, para analisar e comparar o desempenho da rede nos cenários de teste.

6.3 Testes de funcionalidade

Para formar o *overlay* foi escolhido como *bootstrap peer* o *host* c1. Seguem-se os testes efetuados assim como os resultados obtidos e respetiva justificação.

Chamada direta

No teste mais básico efetuado, foi simulada uma chamada entre os *hosts* c1 e c2 e capturados os datagramas à saída da *interface* f0/0 do *router* n1. Na figura 6.2 está ilustrado o diagrama de I/O da interface. Como se pode verificar pelo gráfico, a largura de banda que uma chamada *full-duplex* ocupa é sensivelmente 150kbps. Isto deve-se ao facto do codec utilizado ser o G.711 que utiliza uma frequência de amostragem de 8kHz e cada amostra é de 8 bits. Isto significa que o *bitrate* é de 64kbps (constante). Em cada datagrama transmitido vão contidos 320 bytes (2560 bits) da *media*, o que implica que a cada segundo devem ser transmitidos 25 datagramas para totalizar os 64000 bits. Cada um deles inclui os cabeçalhos das várias camadas em que 12 bytes são para o RTP, 13 bytes para o cabeçalho extra necessário no algoritmo de encaminhamento, 8 bytes UDP, 20 bytes IP e 14 bytes da camada 2 que faz um total de 67 bytes adicionais (17 por cento de *overhead*). Adicionando estes 67 bytes aos 320 bytes de voz faz um total de 387 bytes, temos que o *bitrate* ideal para uma chamada é 77.4kbps. Como se trata de uma comunicação bidirecional a largura de banda ocupada é o dobro, ou seja, 154.8kbps.

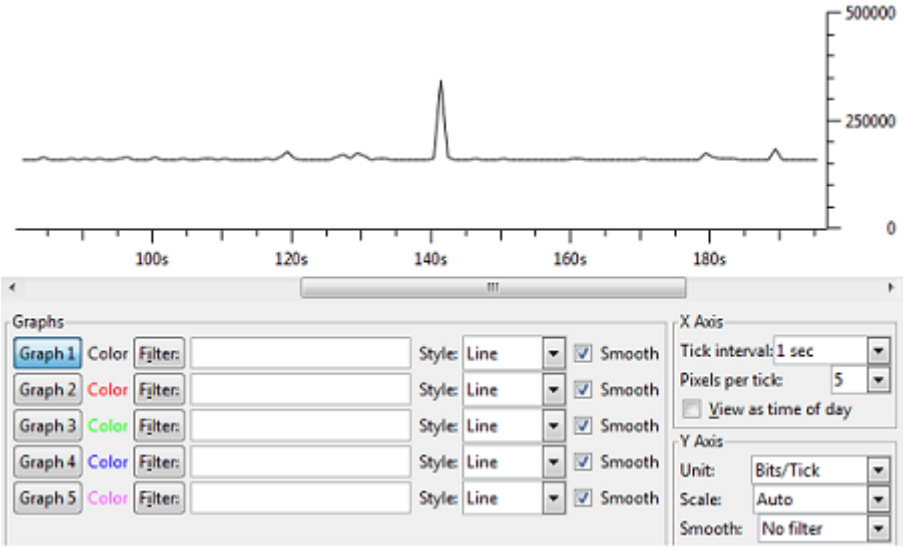


FIGURA 6.2: Simulação de uma chamada

Chamada com um *peer* intermédio

Para verificar o funcionamento do algoritmo de encaminhamento em um *peer* intermédio foi utilizado o *wireshark* para capturar os pacotes no *router* n1 na *interface* f0/0 e no *router* n3 na mesma *interface*. As figuras 6.3 e 6.4 representam o cenário em que são feitas 5 chamadas em simultâneo.

Na figura 6.3 está representado o diagrama de I/O da interface f0/0 do *router* n1. Como se pode verificar o valor médio é sensivelmente 150kbps que é a largura de banda ocupada por uma chamada *full-duplex*. Isto significa que na ligação direta entre n1-n2 está a passar o tráfego de uma chamada e as restantes chamadas estão a ser reencaminhadas por um *peer* intermédio. Neste caso o *peer* é o *host* n19 que está ligado ao *router* n3 e cujo diagrama de I/O está representado na figura 6.4. A largura de banda ocupada pelas 4 chamadas reencaminhadas é de 619.2kbps(154.8*4) tal como se verifica na figura em que o seu valor médio é um pouco superior a 600kbps.

Chamada com n *peers* intermédios

Para verificar o funcionamento do algoritmo de encaminhamento com n *peers* intermédios foram capturados os pacotes no *router* n1 na *interface* f0/0, no *router* n3 na mesma *interface* e no *router* n4 na *interface* f0/1. Nas figuras 6.5, 6.6 e 6.7 estão os gráficos de I/O obtidos na simulação. Como a ligação entre o *router* n3 e o *router* n2 tem uma largura de banda de 64kbps com RTT de 320ms que não cumpre os requisitos mínimos para uma chamada, a *media* foi reencaminhada por um *peer* com ligação ao *router* n4 e que neste caso foi o *host* n16. Como se pode verificar na figura 6.5, a ligação

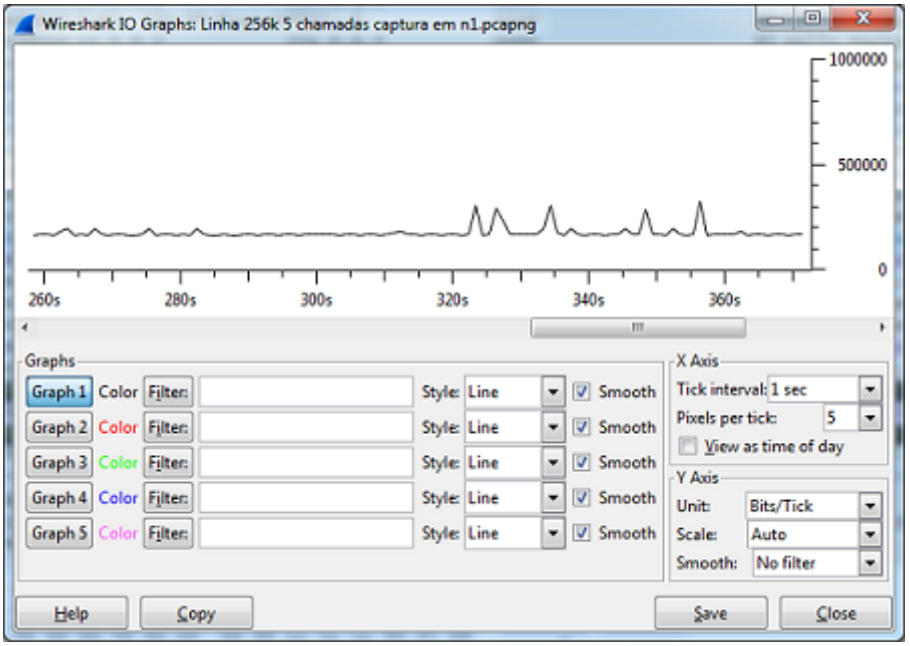


FIGURA 6.3: Captura em n1 na *interface* f0/0 dos pacotes na simulação de 5 chamadas em simultâneo - um *peer* intermédio (uma chamada)

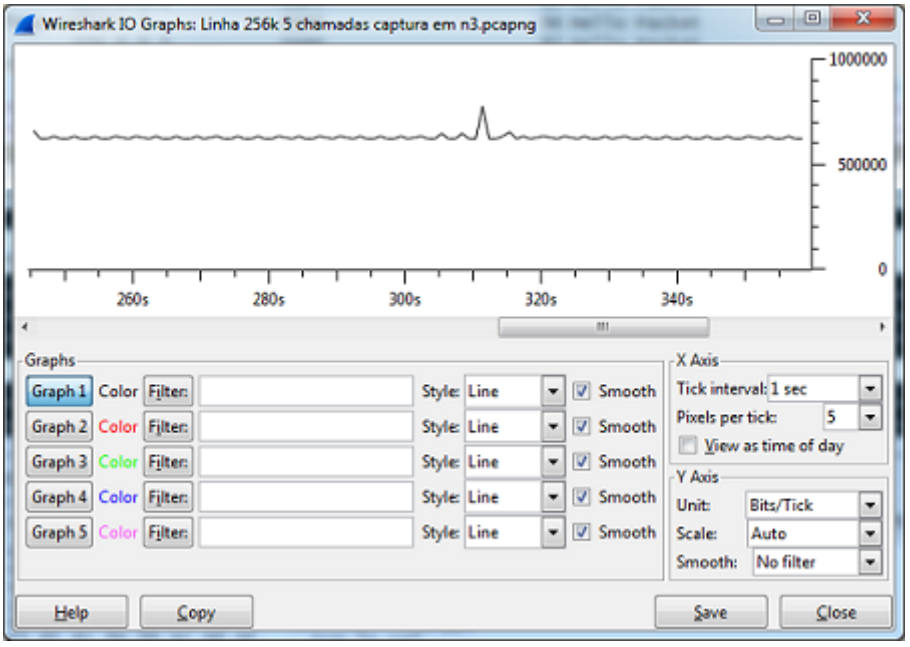


FIGURA 6.4: Captura em n3 na *interface* f0/0 dos pacotes na simulação de 5 chamadas em simultâneo - um *peer* intermédio (quatro chamadas)

direta transporta os dados de uma das 5 chamadas efetuadas. As restantes 4 estão a ser encaminhadas pelo *router* n3 e n4, tal como se verifica nas figuras 6.6 e 6.7.

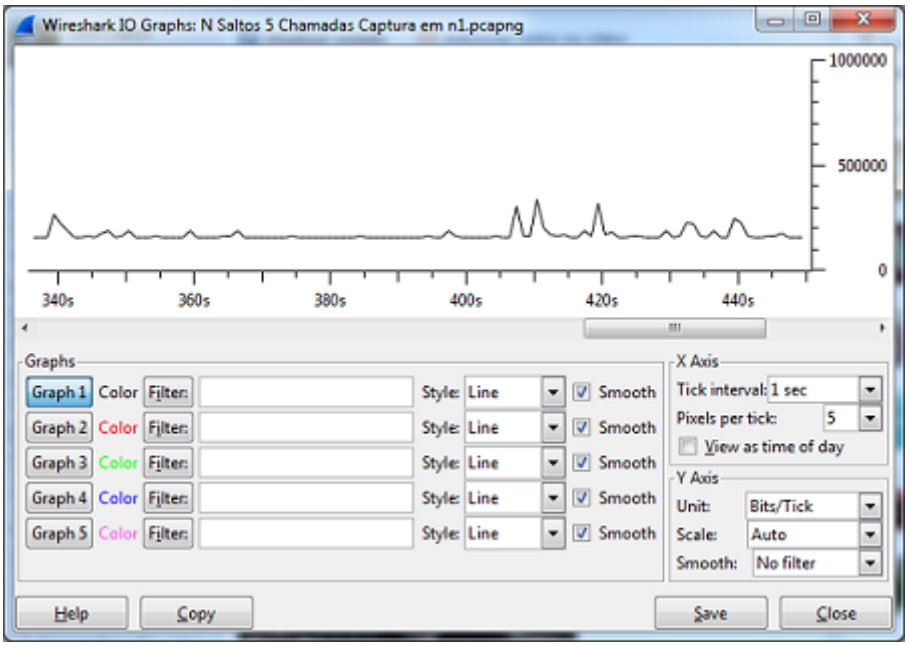


FIGURA 6.5: Captura em n1 na *interface* f0/0 dos pacotes na simulação de 5 chamadas em simultâneo - n *peers* intermédios (uma chamada)

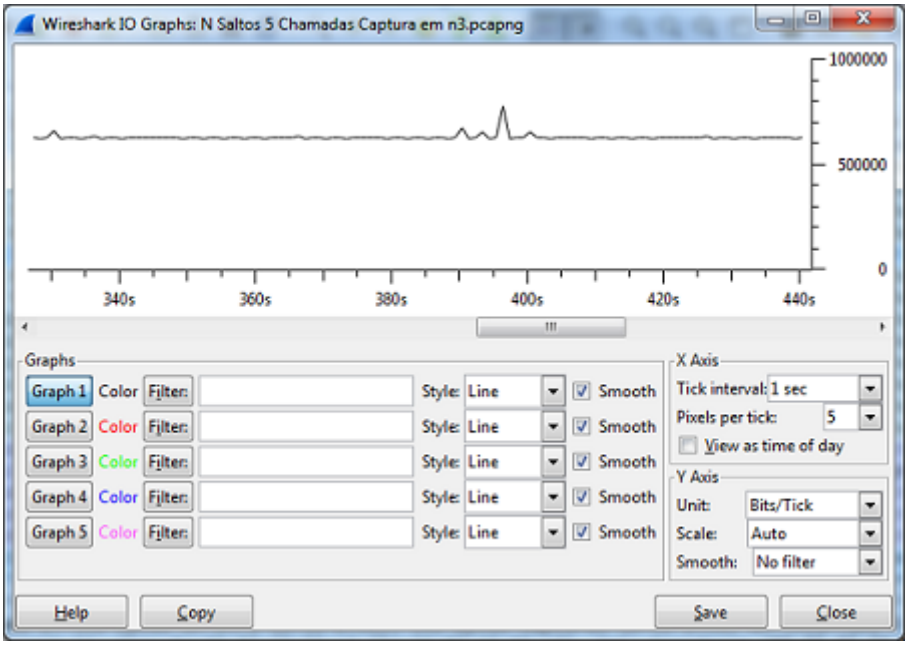


FIGURA 6.6: Captura em n3 na *interface* f0/0 dos pacotes na simulação de 5 chamadas em simultâneo - n *peers* intermédios (quatro chamadas)

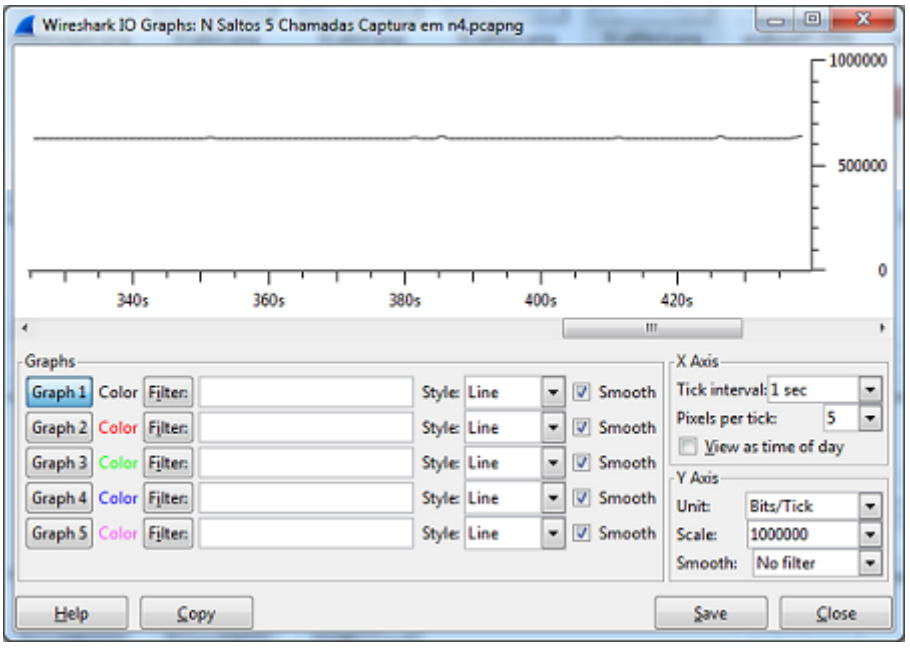


FIGURA 6.7: Captura em n4 na interface f0/1 dos pacotes na simulação de 5 chamadas em simultâneo - n *peers* intermédios (quatro chamadas)

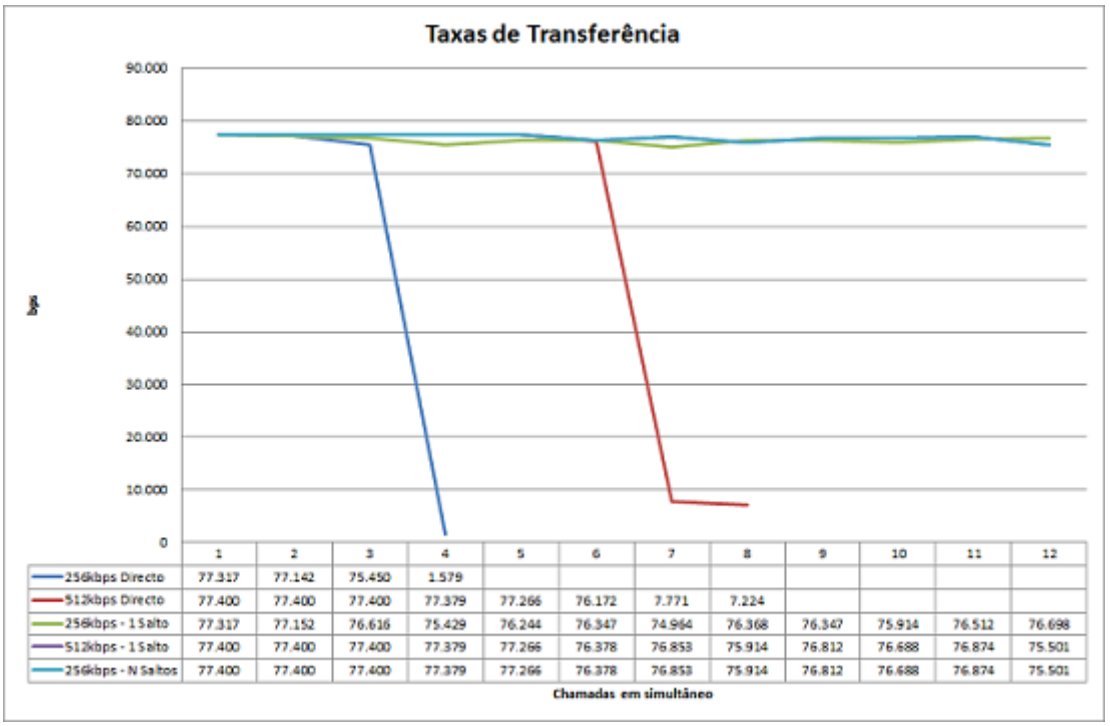
6.4 Testes de desempenho

Após verificar o funcionamento do algoritmo de reencaminhamento foram obtidos os parâmetros para análise e comparação de resultados. Na figura 6.8 está representado o gráfico com a taxa de transferência média para cada uma das chamadas em simultâneo. Para obtenção dos resultados foi efetuado o calculo

$$TT = \frac{br}{t} \tag{6.1}$$

em que "br"é a média dos bits recebidos durante as chamadas efetuadas e o "t"é o tempo de simulação, em segundos. Para obtenção da média do número de bits foram acumulados os números totais de datagramas de voz recebidos com OWD menor que 150ms (visto que são estes que interessam) em cada uma das chamadas e feita uma média entre eles. Após obter esta média, multiplica-se por 3096 (número de bits contido por cada datagrama). O tempo de simulação em todos os testes foi de 5 minutos, portanto, 300 segundos. A taxa de transferência ideal para cada uma das direções da chamada é de 77.4kbps tal como demonstrado nos cálculos acima.

Ainda no gráfico presente na figura 6.8 pode-se verificar que existe um número de chamadas limitado quando não existe reencaminhamento da *media* e que depende dos parâmetros definidos para a ligação. Neste caso e numa ligação com 256kbps com RTT



	1	2	3	4	5	6	7	8	9	10	11	12
256kbps Directo	77.317	77.142	75.450	1.579								
512kbps Directo	77.400	77.400	77.400	77.379	77.266	76.172	7.771	7.224				
256kbps - 1 Salto	77.317	77.152	76.616	75.429	76.244	76.347	74.964	76.368	76.347	75.914	76.512	76.698
512kbps - 1 Salto	77.400	77.400	77.400	77.379	77.266	76.378	76.853	75.914	76.812	76.688	76.874	75.501
256kbps - N Saltos	77.400	77.400	77.400	77.379	77.266	76.378	76.853	75.914	76.812	76.688	76.874	75.501

FIGURA 6.8: Taxas de Transferência

de 150ms o número máximo de chamadas em simultâneo admissíveis é 3. Com 4 chamadas a taxa de transmissão é tão baixa que torna a comunicação sem QoS mínimo. Na tentativa de testar 5 chamadas em simultâneo neste cenário, o congestionamento na rede não permitiu que estas se estabelecessem, isto é, as mensagens SIP perderam-se no processo de sinalização e nem todas foram estabelecidas. No caso em que a ligação tem uma largura de banda de 512kbps com um RTT de 100ms verifica-se que a ligação permite até 6 chamadas em simultâneo. Com 7 e 8 chamadas ainda se consegue estabelecer a ligação mas devido ao enorme congestionamento existem grandes perdas da *media* e que se refletem na taxa de transmissão.

Em relação ao RTT médio, no caso em que não existe reencaminhamento, também há um ponto de rutura no mesmo intervalo de tempo que na taxa de transmissão. No caso em que a ligação tem uma largura de banda de 256kbps a partir da terceira chamada o RTT atinge um valor superior a 12 segundos. Aquando duma largura de banda de 512kbps, com 7 chamadas atinge um valor superior a 6 segundos e com 8 chamadas um superior a 7 segundos. Estes valores estão ilustrados no gráfico da figura 6.9.

No caso em que existe reencaminhamento o RTT médio varia sensivelmente entre 20ms e 200ms. Analisando o gráfico ilustrado na figura 6.10, no cenário em que a largura de banda da ligação direta entre os *routers* n1-n2 é de 256kbps, o RTT diverge muito na situação de reencaminhamento com um ou n *peers* intermédios. Esta situação deve-se

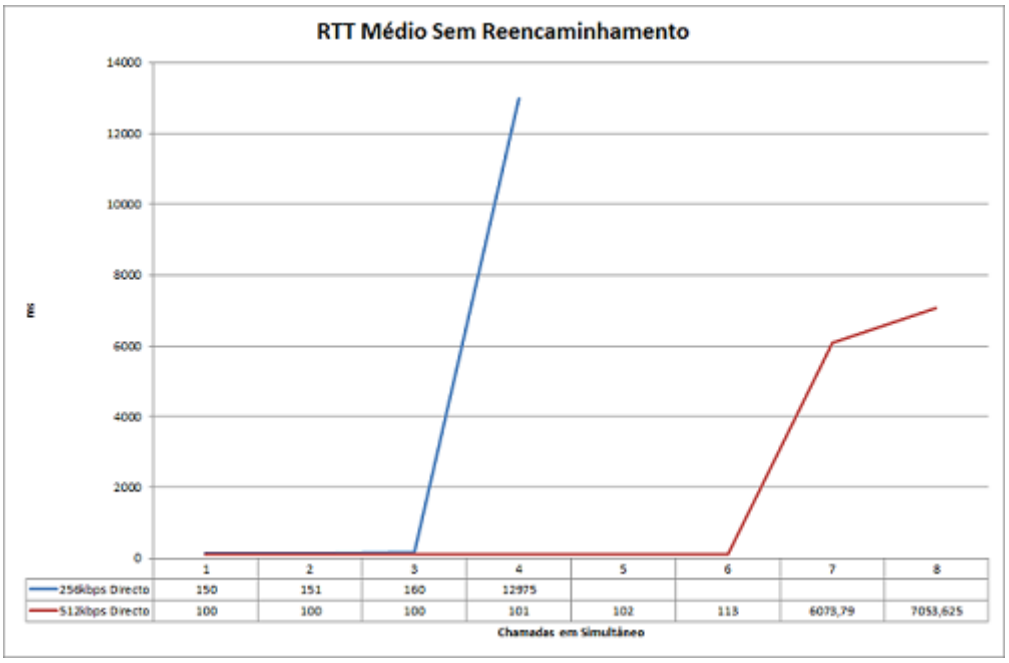


FIGURA 6.9: RTT Médio Sem Reencaminhamento

ao facto do algoritmo de reencaminhamento demorar mais ou menos tempo a estabilizar e encontrar um novo caminho, isto é, enquanto a *media* é transmitida por onde existe muito tráfego acumula-se, um elevado RTT que se reflete na média final. Este tempo de estabilização também se reflete no número de pacotes perdidos e que estão ilustrados no gráfico da figura 6.11. Um outro fator que influencia o RTT é se o algoritmo desvia todas as chamadas pelas ligações onde o RTT é muito baixo ou se mantém chamadas pela ligação direta onde o RTT base neste caso é de 150ms. Comparando com a situação em que não existe reencaminhamento, a ligação direta aguenta com um máximo de 2 chamadas em simultâneo sendo que que com a terceira já existe muitas perdas de dados, enquanto que com o algoritmo de reencaminhamento, a partir da terceira chamada, os dados já começam a ser reencaminhados por outros *peers* tornando assim uma comunicação mais estável. No caso em que a ligação tem uma largura de banda de 512kbps, o RTT também depende do tempo em que o algoritmo demora a estabilizar e do número de chamadas que este consegue manter na ligação direta. Até à quinta chamada em simultâneo o RTT médio é de 100ms porque todas elas vão pela ligação direta. A partir daí o algoritmo de reencaminhamento é ativado e dependendo dos casos o RTT pode diminuir ou aumentar mas sempre dentro do mínimo requerido para uma comunicação VoIP.

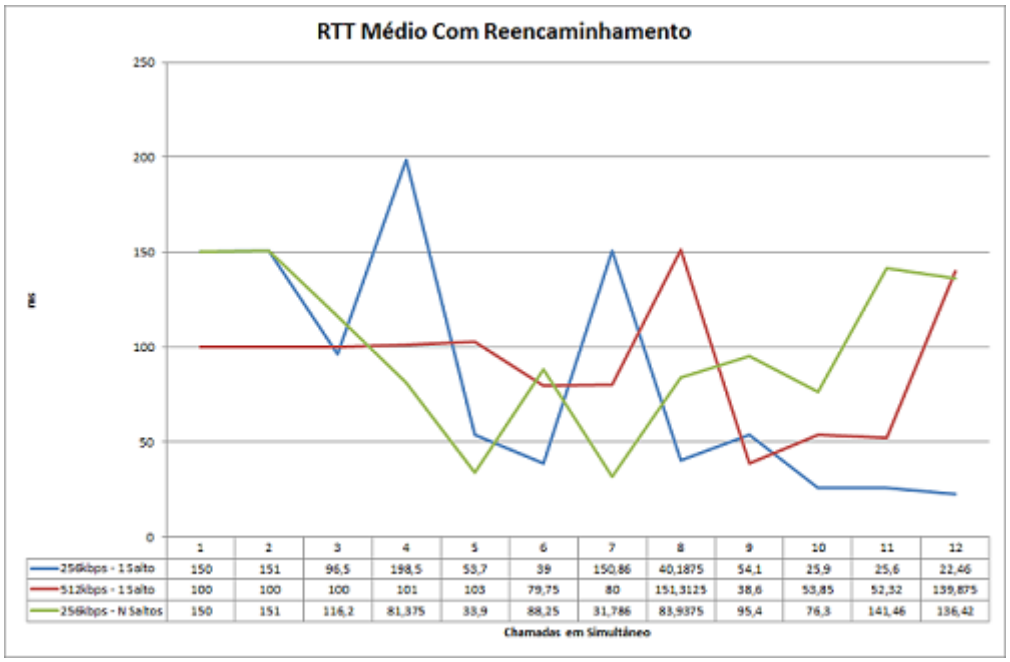


FIGURA 6.10: RTT médio com reencaminhamento

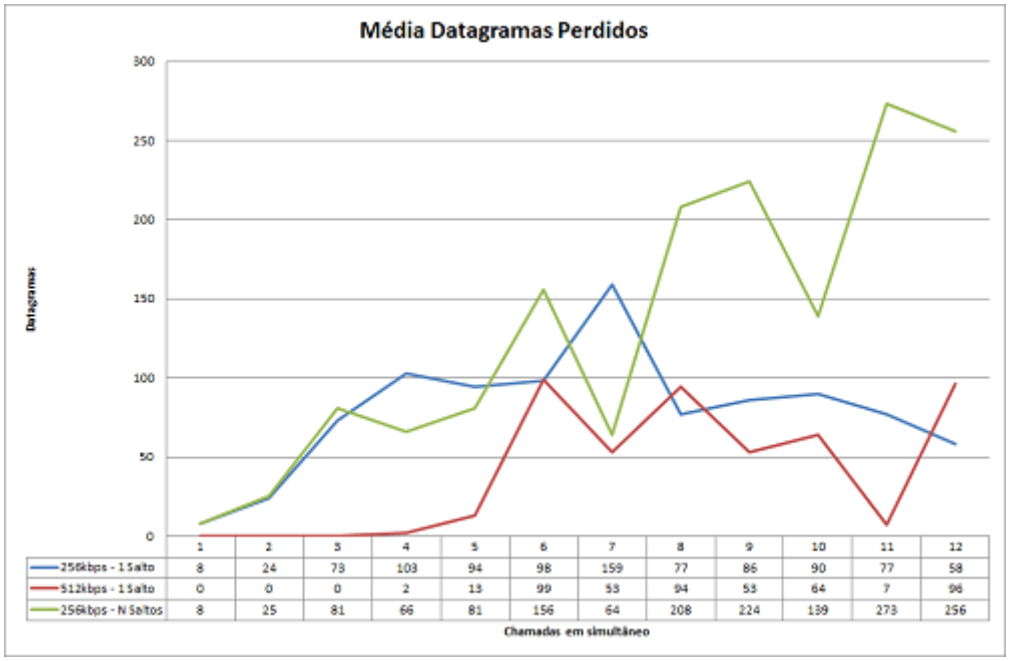


FIGURA 6.11: Média de datagramas perdidos por chamadas em simultâneo

Capítulo 7

Conclusões

Nesta dissertação foi feito um estudo do VoIP, tendo sido estudada a sua arquitetura, funcionamento e protocolos de sinalização mais usados, nomeadamente o SIP e o H.323. O protocolo H.323 foi pioneiro no VoIP e tem o objetivo de especificar sistemas VoIP e que não garantem Qualidade de Serviço (QoS). Além disso, estabelece padrões para codificação e descodificação de fluxos da *media*, garantindo que produtos baseados no padrão H.323 de um fabricante é compatível com produtos H.323 de outros fabricantes.

O SIP (*Session Initiation Protocol*) é um protocolo de texto simples, bastante maduro, pois o seu funcionamento e desempenho é já bem conhecido, para além disso, é um protocolo extensível. O principal objetivo do SIP é o estabelecimento de sessões, permitindo também a negociação de parâmetros quer no estabelecimento de uma sessão como numa sessão já estabelecida, enquadrando-se por isso, perfeitamente no tipo de protocolo necessário para comunicações VoIP.

Foi feito um estudo das redes *peer-to-peer*, tendo sido feito um estudo da forma como se constroem e como os *peers* se posicionam no *overlay*, formando redes estruturadas ou não estruturadas.

Uma rede P2P estruturada caracteriza-se pelo facto da sua topologia ser bem definida, na qual os *peers* são posicionados de uma forma controlada num anel lógico, e os recursos são posicionados de uma forma determinística no *overlay* tornando mais eficiente a sua localização. Um exemplo de uma rede estruturada são as redes que utilizam o protocolo *Chord* que é um protocolo muito usado em redes P2P e que se baseia em DHTs para o posicionamento dos recursos no *overlay*. Numa DHT são atribuídos identificadores únicos aos *peers* da rede (NodeIDs), identificadores esses que podem ser utilizados para o *overlay* decidir onde o *peer* deverá ficar posicionado, assim como os *peers* com os quais

este deve estabelecer ligações. No *Chord*, os *peers* possuem uma tabela de encaminhamento na qual tem a informação sobre os seus vizinhos diretos e uma *finger table*, que é constituída por um conjunto de entradas com informação sobre peers posicionados numa determinada posição do *overlay*.

Nas redes não estruturadas, os *peers* são posicionados de uma forma aleatória no *overlay*, um exemplo é o *Gnutella* que é um *software* de partilha de ficheiros em redes *peer-to-peer* e foi um dos primeiros a surgir. O seu funcionamento baseia-se em criar redes P2P onde a forma como os *peers* estabelecem ligações entre si no *overlay* não é definida, e a localização de recursos é feita através de técnicas que inundam a rede com mensagens de localização (*flooding*).

Um dos objetivos desta dissertação era estudar o protocolo, para criação e manutenção de um *overlay*, normalizado pelo IETF. Por isso, foi estudado o protocolo denominado por RELOAD (*Resource LOcation And Discovery*) no qual o seu RFC foi publicado em janeiro de 2014. O RELOAD caracteriza-se por ser um protocolo binário, desenvolvido para ser mais leve, permitindo melhorar o desempenho global, uma vez que o tamanho das mensagens é reduzido quando comparado com mensagens de texto como as SIP, o que reduz substancialmente o tráfego no *overlay*.

Nesta dissertação também foram apresentados vários trabalhos estudados e implementados no tema "VoIP em redes *peer-to-peer*"do qual se destacou o ASAP (*AS-Aware Peer-Relay Protocol*) e que serviu de base para o nosso desenho e implementação do algoritmo de encaminhamento da *media*. Teoricamente o funcionamento do ASAP é simples e consiste em identificar e formar grupos com os *peers* que o prefixo do endereço IP sejam iguais (significa que estão próximos uns dos outros e partilham as mesmas ligações) e formar *clusters* com os grupos formados. Para cada um dos *clusters* é atribuído um representante que tem como função manter uma lista com *clusters* vizinhos atualizada, entre outras. Quando um novo *peer* se quer juntar à rede contacta um *host* de auxilio, denominado por *bootstrap*, no qual verifica o endereço IP do novo *peer*, identifica o seu prefixo e devolve-lhe a que *cluster* pertence e qual o endereço IP do seu representante. Quando dois nós da rede se encontram na comunicação e detetam que a qualidade da chamada não é suficiente trocam entre si a lista de *clusters* vizinhos e selecionam qual o melhor para encaminhar os dados.

Para além de estudar o protocolo que foi normalizado pelo IETF para a criação e manutenção do *overlay*, nesta dissertação, foi proposto um protocolo de reencaminhamento automático das chamadas pela rede *peer-to-peer*, com um número ajustável de saltos. Cada *peer* possui uma lista com potenciais *peers* que façam reencaminhamento de chamadas. Quando a qualidade da chamada fim-a-fim não é suficiente, os emissores mandam

mensagens sonda (*probes*) para os seus vizinhos com o objetivo de encontrar um caminho alternativo para a *media* transitar. O algoritmo de encaminhamento permite ainda que caso os vizinhos não consigam atingir o recetor, façam *probes* para os seus vizinhos, aumentando assim as hipóteses de encontrar um caminho alternativo.

Na implementação JAVA desenvolvida no âmbito desta dissertação, foram implementados a aplicação VoIP na rede *peer-to-peer* existente e o algoritmo de encaminhamento proposto, efetuado pelos *peers* que constituem o *overlay*. Além disso, foram adicionadas novas funcionalidades à rede *peer-to-peer* que deram suporte à sinalização da chamada VoIP e foi implementada uma interface de interação com o utilizador que permite efetuar ou receber chamadas.

Através da análise e testes efetuados concluímos que as implementações efetuadas se encontram a funcionar corretamente. Os resultados mostram também que com o reencaminhamento de dados, permite acomodar mais chamadas com os mesmos recursos, com a qualidade de serviço requerida.

Como trabalho futuro, uma vez tendo sido provado o bom funcionamento das implementações desenvolvidas, poderia serem efetuados testes de desempenho a uma escala muito maior, uma vez que estes foram muito limitados devido aos recursos que tínhamos.

Seria também interessante implementar o protocolo proposto pelo IETF, denominado por RELOAD, para criação e manutenção do *overay*. Visto que este se trata de um protocolo binário, devia-se comparar quais as vantagens e desvantagens face ao SIP, e qual o impacto que teria no desempenho do *overlay*.

Na rede *peer-to-peer* existente seria interessante implementar um mecanismo que dinamicamente promovesse clientes a *peers* e que despromovesse *peers* a clientes. Os parâmetros a ter em consideração para a promoção e despromoção teriam obviamente que ser bem analisados, de forma a prevenir que este mecanismo tivesse um efeito negativo no desempenho na rede.

Uma outra forma de evoluir este trabalho seria implementar segurança e autenticação, tanto na rede *peer-to-peer* bem como no VoIP, isto é, prevenir que *peers* não identificados ou não autorizados se juntassem ao *overlay* ou prevenir escutas às chamadas. Esta questão da segurança dos dados e autenticação está-se a tornar cada vez mais importante.

Apesar dos testes efetuados ao protocolo de reencaminhamento de dados serem positivos, este poderia ser evoluído em vários aspetos. Cada *peer* não tem em consideração as ligações físicas partilhadas com outros *peers*, isto é, *peers* pertencentes à mesma rede podem trocar *probes* entre sí, o que é desnecessário e causa um aumento de tráfego na rede. Um outro aspeto é o facto de vários *peers* poderem receber vários *probes* de

peers distintos, para o mesmo destino, isto é, *probes* duplicados, o que tem um impacto negativo no desempenho da rede uma vez que gera mais tráfego.

Referências

[1] Theo and Eric Rescorla Skype. An Introduction to standards-based VOIP. 2010.

[2] A. Sollaud. RTP Payload Format for ITU-T Recommendation G.711.1. *RFC 5391*, pages 1–14, 2008. URL http://tools.ietf.org/pdf/rfc5391.pdf.

[3] Rüdiger Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. *Peer-to-Peer Computing, IEEE International*, pages 2–3, 2001. URL http://www.computer.org/csdl/proceedings/p2p/2001/1503/00/15030101.pdf.

[4] IETF-WG. Peer-to-Peer Session Initiation Protocol Working Group. URL https://tools.ietf.org/wg/p2psip/.

[5] G. Camarillo A. Johnston J. Peterson R. Sparks M. Handley J. Rosenberg, H. Schulzrinne and E. Schooler. SIP: Session Initiation Protocol. *RFC3261*, pages 1–269, 2002. URL http://tools.ietf.org/pdf/rfc3261.pdf.

[6] Escola De Engenharia. Oscar Raul Vilela Bravo Redes overlay peer-to-peer baseadas em SIP.

[7] Henning Schulzrinne and Jonathan Rosenberg. Internet Telephony: architecture and protocols – an IETF perspective. *Computer Networks*, 31(3):237–255, February 1999. ISSN 13891286. doi: 10.1016/S0169-7552(98)00265-7. URL http://linkinghub.elsevier.com/retrieve/pii/S0169755298002657.

[8] B Goode. Voice over internet protocol (VoIP). *Proceedings of the IEEE*, 90(9):1495–1517, 2002. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1041060.

[9] Tommi Koistinen. Protocol overview: RTP and RTCP. *Nokia Telecommunications*, pages 1–8, 2000. URL https://www.netlab.tkk.fi/opetus/s38130/k99/presentations/4.pdf.

[10] Henning Schulzrinne and Jonathan Rosenberg. A Comparison of SIP and H. 323 for Internet Telephony. *Proc. International Workshop*, pages 1–4. URL http://www.cs.columbia.edu/~hgs/papers/Schu9807_Comparison.pdf.

[11] Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma and Steven Lim. A survey and comparition os peer-to-peer overlay network schemes. 7: 72–93, 2005. URL https://www.cl.cam.ac.uk/research/dtg/www/files/publications/public/mp431/ieee-survey.pdf.

[12] E. Rescorla S. Baset C. Jennings, B. Lowekamp and H. Schulzrinne. REsource LOcation And Discovery (RELOAD) Base Protocol. *RFC6940*, pages 1–19, Jan. 2014. URL https://tools.ietf.org/pdf/rfc6940.pdf.

[13] D. Bryan. "dSIP: a P2P approach to SIP registration and resource location draft standard. *Internet Draft*, 2007. URL http://tools.ietf.org/html/draft-bryan-p2psip-dsip-00.

[14] S. Baset, H. Schulzrinne and M. Matuszewski. Peer-to-Peer Protocol (P2PP). pages 1–95, 2007. URL http://tools.ietf.org/pdf/draft-baset-p2psip-p2pp-01.pdf.

[15] E. Rescorla S. Baset H. Schulzrinne C. Jennings, B. Lowekamp and T. Schmidt. A SIP Usage for RELOAD draft-ietf-p2psip-sip-13. *Internet draft*, pages 1–19, July 2014. URL https://tools.ietf.org/html/draft-ietf-p2psip-sip-13.

[16] R. Mahy J. Rosenberg and P. Matthews. Interactive connectivity establishment (ICE): a protocol for network address translator NAT traversal for offer/answer. *RFC5245*, pages 1–117, 2010. URL http://tools.ietf.org/pdf/rfc5245.pdf.

[17] P. Matthews R. Mahy and J. Rosenberg. Traversal Using Relays around NAT (TURN). *RFC5766*, pages 1–67, 2010. URL http://tools.ietf.org/pdf/rfc5766.pdf.

[18] P. Matthews J. Rosenberg, R. Mahy and D. Wing. Session Traversal Utilities for NAT (STUN). *RFC5389*, pages 1–51, 2008. URL http://tools.ietf.org/pdf/rfc5389.pdf.

[19] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan Ý. Chord : A Scalable Peer-to-peer Lookup Service for Internet. pages 149–160, 2001. URL http://crystal.uta.edu/~mcguigan/cse6350/papers/Chord.pdf.

[20] U. Blumenthal and P.Goel. Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS). *RFC4785*, pages 1–5, 2007. URL http://tools.ietf.org/pdf/rfc4785.pdf.

[21] D.Taylor, T. Wu T. Perrin, and N. Mavrogiannopoulos. Using the Secure Remote Password (SRP) Protocol for TLS Authentication. *RFC5054*, pages 1–24, 2007. URL http://tools.ietf.org/pdf/rfc5054.pdf.

[22] E. Rescorla S. Baset C. Jennings, B. Lowekamp and H. Schulzrinne. Concepts and Terminology for Peer to Peer SIP draft-ietf-p2psip-concepts-06l. *Internet Draft*, pages 1–19, June 2014. URL https://tools.ietf.org/pdf/draft-ietf-p2psip-concepts-06.pdf.

[23] G. Camarillo J. Maenpaa. Service Discovery Usage for RELOAD. *RFC7374*, pages 1–17. ISSN 2070-1721. URL https://tools.ietf.org/html/rfc7374.

[24] Gholam H. Khaksari, Alexander L. Wijesinha, Ramesh K. Karne, Long He, and Sandeep Girumala. A Peer-to-Peer Bare PC VoIP Application. *2007 4th IEEE Consumer Communications and Networking Conference*, pages 803–807, January 2007. doi: 10.1109/CCNC.2007.163. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4199251.

[25] Marcin Matuszewski and Esko Kokkonen. Mobile P2PSIP - Peer-to-Peer SIP Communication in Mobile Communities. *2008 5th IEEE Consumer Communications and Networking Conference*, pages 1159–1165, 2008. doi: 10.1109/ccnc08.2007.260. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4446558.

[26] Shansi Ren, Lei Guo, and Xiaodong Zhang. ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP. *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pages 70–70, 2006. doi: 10.1109/ICDCS.2006.18. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1648857.

[27] Ben Leong, Barbara Liskov, and Erik D. Demaine. EpiChord: Parallelizing the Chord lookup algorithm with reactive routing state management. *Computer Communications*, 29(9):1243–1259, May 2006. ISSN 01403664. doi: 10.1016/j.comcom.2005.10.002. URL https://www.comp.nus.edu/~bleong/publications/techreport-epichord.pdf.

[28] P. O'Doherty M. Ranganathan and J. van Bemmel. JAIN-SIP: Stack sip for java. URL http://jsip.java.net/.

[29] Jorge Tena Raydelto Hernández. JOpenPhone. 2005. URL http://jopenphone.sourceforge.net.

[30] Yohann Martineau. Peers. 2014. URL peers.sourceforge.net.

[31] DIE University of Roma "Tor Vergata" Dpt. of Information Engineering at University of Parma. MjSIP. 2014. URL http://www.mjsip.org.

[32] Network Technology research group. "Common Open Research Emulator (CORE). 2014. URL http://www.nrl.navy.mil/itd/ncs/products/core.