# Redes *overlay* peer-to-peer baseadas em SIP

Oscar Bravo
Centro Algoritmi
Universidade do Minho
Email: a48058@alunos.uminho.pt

Maria João Nicolau
Centro Algoritmi
Universidade do Minho
Email: joao@dsi.uminho.pt

António Costa
Centro Algoritmi
Universidade do Minho
Email: costa@di.uminho.pt

***Abstract*—As redes Peer-to-peer (P2P) tem vindo a ganhar popularidade na internet, devido ao aumento do número de utilizadores e de serviços de natureza distribuída, como a partilha de ficheiros, e as chamadas de voz sobre IP (VoIP). A criação deste tipo de redes, baseando-se em soluções abertas como o SIP, pode facilitar a criação de novos tipos de serviços, assim como permitir uma mais fácil integração de diferentes serviços.**

**Neste trabalho foi desenvolvida uma implementação JAVA, capaz de criar redes P2PSIP com um ou dois níveis hierárquicos. A comunicação entre os nós da rede P2P, é feita através de um protocolo totalmente baseado em SIP. Como algoritmos a utilizar pelo *overlay* P2P, foram implementados o algoritmo Chord e EpiChord. Para comprovar o funcionamento da implementação JAVA, foram efectuados testes num ambiente real, recorrendo a uma topologia de rede emulada com o CORE.**

## I. Introdução

O aumento do número de utilizadores na internet, assim como os serviços disponibilizados, faz com que as redes *peer-to-peer*, devido à sua natureza distribuída, assumam um papel cada vez mais importante na Internet. Actualmente existem diversas publicações científicas que abordam os vários aspectos destas redes, existindo inclusive um grupo de trabalho dedicado exclusivamente ao seu estudo: o Internet Research Task Force - Peer-to-Peer Research Group (IRTF-P2PWG) [2].

Nas rede *peer-to-peer*, a forma como os dados são indexados e o modo como os nós (peers) são posicionados na rede (*overlay*), leva a que alguns autores classifiquem as redes P2P de duas formas distintas [2] [3], *overlay* não estruturados e *overlays* estruturados.

Num *overlay* P2P não estruturado (por exemplo o Gnutella [4]), o mecanismo utilizado para a localização de recursos na rede, consiste habitualmente na utilização de técnicas de *flooding* [2], [3], onde basicamente a rede é inundada com mensagens para a localização do recurso pretendido. 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. Por isso actualmente as redes P2P são maioritariamente redes estruturadas.

Os *overlay*s P2P estruturados caracterizam-se pelo facto da sua topologia ser bem definida, na qual os nós são posicionados de uma forma controlada, e os recursos são posicionados de uma forma determinística no *overlay*, tornando mais eficiente a sua localização. Este tipo de *overlay* é por vezes denominado por *overlay* DHT, isto porque, normalmente os *overlays* estruturados recorrem a mecanismos baseados em Distributed Hash Tables (DHTs) para o posicionamento dos recursos no *overlay*. Actualmente, os algoritmos DHT mais populares sao o Chord [5] e o Kademlia [6].

O facto de as redes P2P serem habitualmente associadas à partilha de ficheiros, através de protocolos como Gnutella, BitTorrent, Kazaa, etc, faz com que sejam para muitas empresas da indústria sinónimo de actividades ilegais. Contudo, existem diversas aplicações que beneficiam de um suporte *peer-to-peer*, por exemplo aplicações VoIP (*Voice over Internet Protocol*) como o Skype.

Muitas das soluções existentes para implementar redes *peer-to-peer* são fechadas, existindo, no entanto, um esforço a nível académico para desenvolver redes *peer-to-peer* genéricas, baseadas em soluções abertas. A utilização de redes genéricas tem algumas vantagens, por não estarem limitadas a um tipo de serviço ou aplicação específica, podendo ser utilizadas para implementar diversos serviços ou suportar múltiplas aplicações. O facto de utilizarem soluções abertas, como por exemplo o SIP [7], é também uma vantagem, pois existe um maior conhecimento sobre o funcionamento dessas soluções, o que pode facilitar a sua implementação, assim como permitir que entidades diferentes consigam uma melhor interoperabilidade entre si uma vez que o funcionamento da solução utilizada é bem conhecido.

O SIP (Session Initiation Protocol) é um protocolo de sinalização standard do IETF (Internet Engineering Task Force), que funciona ao nível da camada de aplicação. É bastante utilizado para estabelecer sessões entre um ou vários participantes, por exemplo no estabelecimento de chamadas telefónicas através da internet, distribuição de conteúdos multimédia, conferências, etc.

Existe actualmente um grupo de trabalho do IETF, Peer-to-Peer Session Initiation Protocol (P2PSIP-WG) [8], que se dedica ao estudo e criação de um protocolo P2PSIP. Das varias propostas para protocolos P2PSIP existentes, foram abordadas o dSIP [9], o P2PP [10] e o RELOAD [11]. O protocolo dSIP [9], [12] permite a criação de *overlays* P2P estruturados, recorrendo à utilização exclusiva de mensagens SIP, é por isso, um protocolo totalmente baseado em SIP.

O P2PP [10], é uma outra especificação preliminar para um protocolo P2PSIP, permite que o *overlay* P2P seja criado através de protocolo P2P estruturados ou não estruturados. Para além disso, as mensagens utilizadas por este protocolo não são mensagens SIP, mas sim mensagens binárias. Com este protocolo, o envio de mensagens SIP entre *peers* é conseguido

através do encapsulamento das mensagens SIP nas mensagens binárias do P2PP.
O protocolo RELOAD(REsource LOcation And Discovery) [7], [11], é a mais recente proposta do P2PSIP-WG, contudo, assim como o P2PP, também utiliza um protocolo binário para a construção e gestão do *overlay*, permitindo a utilização do SIP, ou de outros protocolos por cima do protocolo RELOAD.

Neste trabalho foi desenvolvida uma implementação JAVA capaz de criar uma rede *peer-to-peer* estruturada, totalmente baseada em SIP. A comunicação é feita através de um protocolo P2PSIP, tendo para o overlay P2P sido implementados dois algoritmos DHT: o Chord [5] e o EpiChord [13]. Das várias soluções para protocolos P2PSIP estudadas, a especificação preliminar (*draft*) do protocolo dSIP [9] foi a escolhida. O dSIP foi o escolhido por ser das várias propostas, aquela que mais se enquadra com o que era pretendido, nomeadamente o facto de o protocolo ser totalmente baseado em SIP.
Relativamente aos algoritmos DHT implementados, a escolha do Chord deve-se ao facto deste ser um dos algoritmos DHT mais populares, além de ser de implementação obrigatória em todas as soluções para protocolos P2PSIP estudadas. O EpiChord é uma variante do Chord, que segundo os seus autores consegue melhorar o desempenho do *overlay* e por esse motivo também foi implementado.
A implementação Chord desenvolvida baseou-se no trabalho de Cirani et al [19], embora muitos dos componentes tenham sido adaptados e reescritos. A implementação do Epichord foi construída de raíz.

Para além da criação de um *overlay* P2PSIP utilizando o Chord ou EpiChord, a aplicação é capaz de criar um *overlay* com dois níveis hierárquicos. O primeiro nível hierárquico é composto exclusivamente por *peers*, formando estes o *overlay*. Num segundo nível encontram-se os clientes que se ligam a um ou vários *peers*, e utilizam os serviços disponibilizados pelo *overlay*, sem que participem activamente na construção e gestão deste, actuando apenas em seu beneficio. A criação de uma hierarquia deste género visa comprovar que em alguns cenários pode ser melhor para o desempenho do *overlay*, que determinados *peers*, deixem de participar activamente no *overlay*, passando a ser clientes. Por participação activa no *overlay*, compreende-se a troca de mensagens entre *peers* para a gestão e construção do *overlay*, e também armazenamento de informação.

## II. Estado da Arte

### A. Distributed Session Initiation Protocol (dSIP)

O Distributed Session Initiation Protocol (dSIP) é uma especificação preliminar (*draft*) [9] para um protocolo P2PSIP proposto por alguns autores da SIPeerior Technologies e da CISCO. É visto como uma evolução do protocolo SoSIMPLE [14] o qual foi especificado por alguns dos mesmos autores do dSIP. Este protocolo é totalmente baseado em SIP [15], sendo este utilizado para efectuar toda a gestão do *overlay* P2P, como por exemplo para o registo e localização de recursos ou *peers*.

O facto de o SIP ser um protocolo extensível, permitiu a criação de novos cabeçalhos SIP, necessários para o transporte de informação relevante para a gestão do *overlay* P2P. Segundo os autores do dSIP [9], a utilização das mensagens SIP tradicionais, permite a utilização no dSIP, de mecanismos habitualmente utilizados no SIP (STUN [16], TURN [17] e ICE [18]) para ultrapassar os problemas causados por NAT's e firewalls.

*1) Estrutura do Overlay P2P:* O protocolo dSIP foi desenvolvido de modo a ser modular, podendo ser utilizado com múltiplos algoritmos de DHT, sendo necessário suportar pelo menos o Chord.
Relativamente à estrutura do *overlay* P2P, no dSIP os *peers* são organizados no *overlay* de acordo com o algoritmo DHT em utilização. São atribuídos identificadores únicos (Peer-ID e Resource-ID) aos *peers* e aos recursos armazenados no *overlay*, sendo que ambos os identificadores devem pertencer ao mesmo espaço de endereçamento. O calculo dos identificadores únicos pode ser obtido recorrendo a diversos algoritmos, contudo é necessário que seja utilizado o mesmo algoritmo por todos os *peers* do *overlay*. Na implementação base, que utiliza o Chord como DHT o algoritmo utilizado é o SHA-1 com 160 bits.
A forma como os identificadores dos *peers* são obtidos pode variar consoante o nível de segurança que é pretendido. Por exemplo, pode ser obtido aplicando o algoritmo SHA-1 sobre a combinação endereço IP e porta do *peer*, ou recorrendo a uma entidade certificadora responsável por emitir os identificadores.
Através do Peer-ID de um *peer* o algoritmo DHT em utilização, determina a localização do *peer* no *overlay*, assim como os identificadores dos recursos pelos quais o *peer* é responsável. A forma como um *peer* é posicionado no *overlay*, e os recursos pelos quais é responsável depende do algoritmo DHT que o *overlay* utiliza.
O *Resource-ID*, identificador utilizado para identificar recursos armazenados no *overlay*, pode ser obtido através da aplicação de uma função de hash sobre o nome ou sobre um conjunto de palavras que descrevem o recurso. O recurso é armazenado no *peer* que tiver o *Peer-ID* mais próximo do seu *Resource-ID*.
Devido à constante entrada e saída de *peers* numa rede P2P, a informação relativa aos recursos armazenados vai sendo trocada entre estes de forma a que os recursos estejam sempre acessíveis. São implementados mecanismos de redundância de informação para evitar que haja perda de dados, quando por exemplo um *peer* falha antes de transmitir para outro *peer* a informação relativa aos recursos pelos quais era responsável.

A forma exacta como a localização de recursos é feita varia consoante o algoritmo DHT em utilização. De uma forma genérica, para a localização de um determinado recurso, um *peer* deve consultar a sua tabela de encaminhamento, e enviar a mensagem para o *peer* que possuí o *Peer-ID* mais próximo do *Resource-ID* do recurso pretendido. Devendo esse *peer*, dependendo do mecanismo de encaminhamento em utilização, reenviar a mensagem para o *peer* mais próximo do recurso que conhece, ou enviar uma mensagem de resposta contendo a informação relativa ao *peer* mais próximo do recurso que

este conhece.

### B. EpiChord

O EpiChord é um algoritmo de localização de recursos para redes *peer-to-peer* baseadas em *Distributed Hash Tables* (DHTs), desenvolvido por Ben Leong et al. [13].
Uma das principais características que diferencia o EpiChord de outros algoritmos DHT existentes, é a utilização de uma tabela de encaminhamento (denominada pelos autores por *cache*) sem limite máximo de entradas. Outros algoritmos DHT como por exemplo, o Chord possuem um limite máximo de O(log N) entradas.

Devido ao facto de o número de entradas na tabela de encaminhamento poder afectar o número de saltos necessários para localizar um recurso no *overlay*, os autores do EpiChord afirmam que este permite reduzir em média o número de saltos necessários para a localização de um recurso.

*1) Estratégia de Encaminhamento:* Para além da utilização de uma *cache* com tamanho indefinido, o EpiChord implementa uma nova estratégia de encaminhamento denominada por *reactive routing*. Esta estratégia de encaminhamento utiliza as mensagens de localização de recursos para transportar informação útil para a manutenção da cache dos nós do *overlay*. Desta forma, os nós do *overlay*, conseguem manter parte da sua cache actualizada observando apenas o tráfego de localização de recursos que vão recebendo, adicionando informação de encaminhamento às mensagens de resposta que enviam. Contrariamente, outros DHT's como por exemplo o Chord, necessitam de enviar periodicamente mensagens para as várias entradas das suas tabelas de encaminhamento (*finger table* no caso do Chord), de forma a verificar a validade das mesmas. No EpiChord, este comportamento é apenas necessário, se o tráfego na rede for demasiado baixo, pois nesse caso o número de mensagens que cada nó recebe pode não ser suficiente para manter a sua cache minimamente actualizada. Sendo necessário enviar mensagens para apenas algumas entradas da cache, de modo a manter a cache minimamente actualizada.

Com esta estratégia de encaminhamento, a informação contida na cache nem sempre é a mais actualizada quando comparada com outras estratégias de encaminhamento. Pelo que para a localização de recursos é necessário recorrer ao envio de mensagens em paralelo, de modo a minimizar o efeito da existência de entradas inválidas na cache.
O envio de mensagens em paralelo, nem sempre é uma boa solução pelo tráfego extra que gera, contudo como o EpiChord possui uma cache muito grande o número de saltos necessários para localizar o recurso é mais reduzido o que reduz também o número de mensagens de localização a enviar e o respectivo tráfego na rede.
Os autores [13] afirmam que o envio de mensagens em paralelo do EpiChord quando comparado com uma implementação Chord tradicional, com tráfego de localização de recursos semelhante, consegue melhorar em média a performance da localização de recursos tanto em número de saltos como na latência.

*2) Localização de Recursos:* A localização de recursos no EpiChord é feita recorrendo ao envio de mensagens (denominadas por *queries*) em paralelo, sendo que para se iniciar uma localização de um recurso são enviadas **p** mensagens em paralelo, onde **p** é um parâmetro configurável do sistema.
O algoritmo utilizado para determinar o destino de cada uma das **p** mensagens a enviar é simples, é consultada a cache para se obter um conjunto de nós mais próximos do nó responsável pelo recurso a localizar. Exemplo, denominando por **id** o identificador do recurso a localizar, o envio das **p** mensagens é feito da seguinte forma: é enviada uma mensagem para o nó da cache que é o sucessor imediato do **id** a localizar e de seguida, envia-se para os **p-1** nós da cache que antecedem o **id** do recurso. A figura 1 mostra um exemplo do algoritmo descrito anteriormente, neste exemplo um nó **x** pretende localizar o recurso identificador por **id**. Quando um

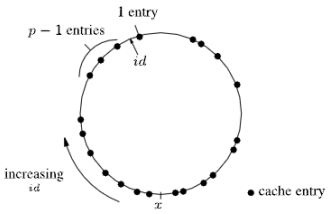


Fig. 1. Escolha do destino das **p** mensagens a enviar [13]

nó **x** inicializa a localização de um recurso identificado por **id** através do envio de **p** mensagens, os nós que receberem mensagens para a localização do recurso, devem enviar uma mensagem de resposta, de acordo com as seguintes condições:

- Se o nó for responsável pelo recurso pretendido, a resposta deve conter o valor associado ao recurso (caso exista), assim como deve também enviar informação de *routing*, contendo os dados relativos ao seu antecessor.
- Se o nó é antecessor de **id** relativamente ao nó **x** a mensagem de resposta deve conter informação sobre o sucessor do nó, assim como os **l**[1] melhores próximos saltos obtidos através da cache, onde **l** é um parâmetro configurável do sistema.
- Se o nó é sucessor de **id** relativamente ao nó **x** a mensagem de resposta deve conter informação sobre o antecessor do nó, assim como os **l** melhores próximos saltos obtidos através da cache.

À medida que as respostas vão sendo recebidas pelo nó **x**, este verifica a informação de *routing* nelas contidas, actualizando a sua cache com essa informação. Para além da actualização da cache, **x** envia novas mensagens de localização em paralelo para os nós contidos nas mensagens de resposta e que se encontram mais próximos do destino comparativamente aos nós que já responderam. Isto é, à medida que os nós vão respondendo, são actualizados dois parâmetros que identificam os dois nó mais próximos (sucessor e antecessor do identificador a localizar) do destino, sendo apenas enviadas novas

[1]Os **l** melhores próximos saltos referem-se ao *peer* que sucede o identificador do recurso, e os **l-1** nós que antecedem o recurso

mensagens se o destinatário estiver mais próximo do destino que o actual melhor sucessor ou antecessor.
O algoritmo de localização utiliza uma estratégia de *routing* iterativo, o que permite que o nó que inicia a localização possa verificar se está a aproximar-se do nó pretendido, assim como manter um histórico das mensagens enviadas. Desta forma é possível detectar casos em que um nó é referenciado múltiplas vezes nas mensagens de resposta de outros nó, evitando-se o envio de mensagens repetidas para o mesmo nó.

## III. Descrição e análise do problema

### A. Descrição geral

A escolha do protocolo dSIP como protocolo P2PSIP a utilizar, deve-se ao facto de esta ser uma solução totalmente baseada em SIP, sendo por isso, das propostas apresentadas, aquela que melhor se adequa aos objectivos deste trabalho.

O facto de o dSIP ser totalmente baseado em SIP é uma vantagem pois o SIP é um protocolo normalizado e bastante utilizado em diversas aplicações, nomeadamente aplicações para efectuar chamadas de voz sobre IP (vOIP). A utilização de um protocolo conhecido, como o SIP, torna mais fácil a implementação de novos serviços assim como pode permitir a interoperabilidade entre diferentes serviços.

Relativamente ao *overlay peer-to-peer*, os *peers* que o formam posicionam-se e estabelecem ligações com outros *peers* de acordo com o algoritmo DHT em utilização no *overlay*.
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 arquitectura 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 do utilizador.
O identificador do recurso (o *hash* da chave) é utilizado pelo algoritmo DHT do *overlay* para definir a localização na qual o recurso deve ser armazenado no *overlay*.

### B. Hierarquia com dois níveis

Num *overlay* P2P tradicional todos os *peers* possuem a mesma importância, devendo participar activamente 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 por outros motivos como por exemplo o desempenho do *overlay*.

Neste trabalho, é abordado o segundo ponto, no qual se pretende estudar qual o impacto que tem no desempenho do *overlay* a existência de *peers* com menores capacidades.
Designam-se por *peers* com menores capacidades, aqueles que possuem algumas das seguintes características:

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

Os *smartphones* são um bom exemplo de um dispositivo móvel capaz de participar num *overlay* P2P mas cuja participação pode ter algum impacto tanto no *overlay* como no próprio dispositivo, devido às possiveis limitações a nível de conectividade e também às características do dispositivo, nomeadamente a bateria.

A existência no *overlay* de *peers* com algumas das características enumeradas como por exemplo, com pouca largura de banda, ou alta probabilidade de perda de pacotes, pode influenciar o desempenho do *overlay*, afectando o tempo para a localização de recursos.

Para minimizar estes problemas, implementamos um *overlay* P2P de dois níveis.
O nível mais baixo da hierarquia, é composto por nós com menores capacidades, designados por clientes. Um cliente não forma nem participa activamente em qualquer *overlay* P2P, podendo contudo, utilizar os serviços disponibilizados pelo *overlay*. Para aceder aos serviços disponibilizados pelo *overlay*, um cliente tem de estabelecer uma ligação com pelo menos um *peer* do *overlay*, enviando para este os seus pedidos para armazenar ou localizar recursos.
Desta forma, um cliente actua sempre para seu próprio benefício, não recebendo mensagens que não são para si, nem tem a responsabilidade de armazenar dados do *overlay*, utilizando um *peer* pertencente ao *overlay* como intermediário para aceder aos serviços que este disponibiliza.

O nível superior da hierarquia, é composto pelos nós(peers) que formam realmente o *overlay* DHT. Estes actuam como *peers* normais, armazenando recursos e encaminhando mensagens entre si. A única alteração que é necessário efectuar 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 para, por exemplo armazenar um recurso no *overlay*, o *peer* que recebeu o pedido do cliente armazena o recurso no *overlay*, como se este fosse seu, não havendo necessidade de efectuar alterações nos algoritmos DHT.

A figura 2 mostra um exemplo da hierarquia descrita.
É possível verificar na figura, que o *overlay* é composto exclusivamente por *peers*, ficando os clientes no nível inferior, acedendo aos serviços do *overlay* através de um ou vários *peers*.

### C. Overlay peer-to-peer

A especificação preliminar do protocolo dSIP [9] não impõe restrições relativamente aos algoritmos DHT que podem ser utilizados pelo *overlay*. Contudo, define que pelo menos o algoritmo Chord deve ser implementado.
Neste trabalho, para além da implementação obrigatória do algoritmo Chord, decidiu-se implementar o EpiChord, que é

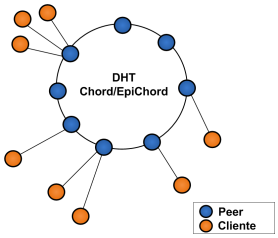


Fig. 2. P2PSIP - Hierarquia de dois níveis

uma variante do algoritmo Chord, e que promete melhorar o desempenho geral do *overlay*.

Com a implementação destes dois algoritmos DHT, pretende-se analisar o desempenho de ambos num cenário real, de modo a verificar se os resultados obtidos estão de acordo com os resultados obtidos através de simulação pelos autores do EpiChord.

Para além da análise ao desempenho dos algoritmos, pretendemos também analisar qual o impacto da utilização de um *overlay* com ou sem hierarquia, analisando alguns parâmetros importantes na localização de recursos, como o número de saltos e o tempo médio da localização.

Na aplicação desenvolvida neste trabalho, ambos os algoritmos DHT utilizam como algoritmo de *hashing* o SHA-1 de 160 bits, utilizado para gerar o *hash* dos identificadores dos *peers* e dos recursos. O uso de identificadores compostos por 160 bits, significa que o anel formado pelos **peers** do *overlay* tem um espaço de endereçamento $2^{160}$ identificadores.

Um outro aspecto importante na implementação dos algoritmos DHT da aplicação é a estratégia de encaminhamento a utilizar. Visto que o algoritmo EpiChord foi desenvolvido para funcionar com uma estratégia de encaminhamento iterativo, decidiu-se que esta deveria ser a estratégia a utilizar na implementação de ambos os algoritmos, Chord e EpiChord. Desta forma, as comparações aos resultados obtidos de ambos os algoritmos DHT tornam-se mais justas.

*1) Implementação Chord:* A implementação do algoritmo Chord neste trabalho foi baseada na especificação preliminar [12] dos mesmos autores de dSIP, na qual se descreve a forma como o algoritmo Chord deve ser implementado num *overlay* P2PSIP dSIP.

S. Cirani et al. autores de uma outra especificação [19] desenvolveram uma aplicação que implementa o protocolo dSIP utilizando o algoritmo Chord e Kademlia. O código fonte da aplicação encontra-se disponível na web [19] e foi em parte utilizado na implementação do Chord neste trabalho. Na implementação deste trabalho, foi aproveitada a forma hierarquia como o algoritmo Chord foi implementado por [19], assim como alguns dos seus componentes, como por exemplo, o componente responsável pela criação das mensagens P2PSIP. Os componentes reutilizados, sofreram todos eles bastantes modificações, de modo a criar um maior nível de abstração entre a camada DHT, e a camada SIP. Para além disso, a stack SIP utilizada na implementação de S. Cirani et al. era a MJSIP, tendo sido alterada neste trabalho para a Jain-SIP.

*2) Implementação EpiChord:* Relativamente à implementação do EpiChord, esta seguiu também a especificação preliminar [12] que descreve a forma como o Chord deve ser implementado no dSIP.

Um dos pontos principais da implementação do EpiChord, é a gestão da cache, pois esta é completamente diferente da gestão da *finger table* do Chord. No EpiChord a cache é preenchida com informação proveniente das mensagens recebidas. É importante detectar e remover entradas mortas, que falharam um determinado número de vezes ou cujo **lifetime** exceda um determinado valor pré-definido. A gestão da cache é bastante importante para o bom funcionamento do algoritmo, pois a localização de recursos baseia-se na informação contida na cache. Para além disso, o facto de não haver um número máximo de entradas na cache torna importante a detecção e remoção de entradas que já tenham expirado, de modo a limitar o número de entradas na cache a cada instante.

Um outro aspecto a ter em conta na implementação deste algoritmo, é a forma como é feita a introdução e localização de recursos no *overlay*. Enquanto que no Chord se recorre a um mecanismo simples de troca de mensagens para inserir ou localizar recursos, no EpiChord o processo é um pouco mais complexo, utilizado-se um mecanismo baseado no envio de mensagens em paralelo.

As diferenças entre os dois algoritmos, levam a que as suas implementações neste trabalho partilhem poucos pontos em comum, levando a que a implementação EpiChord tenha sido criada praticamente de raiz.

### *D. Mensagens SIP utilizadas*

As mensagens utilizadas pelo protocolo dSIP são mensagens SIP tradicionais com a adição de dois novos tipos de cabeçalhos (DHTPeerID e DHT-Link), necessários para o transporte de informação relevante para a gestão do *overlay peer-to-peer*.

Visto que neste trabalho se pretende que exista a possibilidade de haver clientes e *peers*, numa hierarquia de dois níveis, e como o protocolo dSIP especifica apenas as mensagens que os *peers* devem trocar entre si, não prevendo o cenário da existência deste tipo de clientes. Foi necessário especificar os tipos de mensagens que os clientes devem utilizar para comunicar com os *peers*. Para simplificar este processo, decidiu-se criar um novo cabeçalho SIP denominado por 'ClientID', e que é baseado no cabeçalho 'DHT-PeerID'. O novo cabeçalho é utilizado apenas pelos clientes, e tem como objectivo permitir que um *peer* possa identificar a origem de uma mensagem. A existência de um cabeçalho 'ClientID' ou 'DHT-PeerID' numa mensagem, permite a um *peer* identificar facilmente se a mensagem é oriunda de um cliente ou de um *peer*.

Desta forma, clientes e *peers* utilizam ambos os tipos de mensagens definidos pelo protocolo dSIP, onde a única diferença nas mensagens enviadas por clientes ou *peers* é a utilização do novo cabeçalho 'ClientID' ou do cabeçalho 'DHT-PeerID'.

*1) Cabeçalhos das mensagens:* Relativamente aos cabeçalhos das mensagens SIP definidos pelo dSIP, a

implementação dos algoritmos Chord e EpiChord não introduz qualquer alteração nestes, definindo apenas a forma como o cabeçalho *DHT-Link* deve ser constituído.
Na implementação Chord e EpiChord, o cabeçalho *DHT-Link* é utilizado para um *peer* enviar para outros informação relativa ao seu sucessor, predecessor ou elementos da tabela de encaminhamento(*finger table* no Chord, e *cache* no EpiChord).

Segundo [12], na implementação Chord, o parâmetro **link** do cabeçalho deve ter um dos seguintes valores:

- P[N] - Indica que o cabeçalho contêm informação relativa ao predecessor N do *peer*.
- S[N] - Indica que o cabeçalho contêm informação relativa ao sucessor N do *peer*.
- F[N] - Indica que o cabeçalho contêm informação relativa à entrada N da *finger table* do *peer*.

O valor de **N** deve ser um valor positivo, e para links do tipo P ou S indica a sua a profundidade devendo este valor ser igual ou superior a 1. Por exemplo, se o parâmetro link tiver o valor P1, significa que o cabeçalho contem informação sobre o predecessor imediato do *peer*, já no caso de o valor ser por exemplo S5, significa que a informação contida no cabeçalho é relativa ao quinto sucessor do *peer*. A razão pela qual o valor de N deve ser superior a 1, deve-se ao facto de que se o valor for por exemplo S0 ou P0, o cabeçalho conteria informação sobre o próprio *peer*, o que não faz sentido.
Se o parâmetro link for do tipo F, isto é, para cabeçalhos que possuem informação sobre um *peer* da *finger table*, o valor de N indica o índice do *peer* na *finger table*.

A figura 3 mostra um exemplo de um cabeçalho DHT-Link utilizado na implementação Chord do dSIP. Neste exemplo, o cabeçalho possuí informação sobre o sucessor imediato (S1) do *peer* que envia a mensagem. Os atributos relevantes do *peer*, tais como o seu identificador (peer-ID) e o endereço IP são visíveis neste cabeçalho.

**Diferenças na implementação EpiChord**

```
DHT-Link: <sip:peer@192.0.2.1;peer-ID=671a65bf22>
;link=S1;expires=600
```

Fig. 3. Chord - Exemplo de um cabeçalho DHT-Link do protocolo dSIP

O conteúdo do cabeçalho DHT-Link na implementação do algoritmo EpiChord, é ligeiramente diferente nomeadamente nos parâmetros **link** e **expires**.
Visto que o EpiChord não utiliza uma *finger table* mas sim uma *cache* como tabela de encaminhamento, o valor **F[N]** que o parâmetro **link** pode ter, deixa de fazer sentido. Por isso, na implementação do EpiChord, este deixa de poder ter o valor **F[N]** passando a poder ter um novo valor **C**(sem qualquer índice), que indica que o cabeçalho contêm informação relativa a um **link** da cache.

Uma outra diferença na implementação EpiChord, tem a ver com o significado do parâmetro **expires** do cabeçalho DHT-Link. O EpiChord não utiliza o parâmetro **expires** nas entradas da sua cache, utilizando antes o parâmetro **lifetime**, que tem um significado diferente. Para evitar a criação de um novo cabeçalho, no qual a única diferença seria que o parâmetro **expires** passaria a chamar-se **lifetime**, decidiu-se utilizar o mesmo cabeçalho da implementação Chord, onde o parâmetro **expires** contêm o valor do parâmetro **lifetime** do link na cache do *peer*.

## IV. Implementação

Neste capítulo descreve-se a arquitectura e implementação das aplicações desenvolvidas, onde o JAVA foi a linguagem de programação escolhida. Os algoritmos DHT implementados nas aplicações foram o Chord [5] e o EpiChord [13], tendo ambos sido implementados de forma a poderem funcionar num *overlay* com um ou dois níveis hierárquicos.

### A. Arquitectura

A arquitectura das aplicações desenvolvidas foi implementada de modo a que a mesma arquitectura pudesse ser utilizada pelas várias aplicações, permitindo ser adaptada a qualquer tipo de algoritmo DHT que se pretenda implementar. Para tal, o desenho da arquitectura pode ser dividido em várias camadas, posicionadas hierarquicamente.
A figura 4 ilustra as camadas envolvidas na arquitectura das aplicações.

Em termos de desenvolvimento, neste trabalho foram desenvolvidas as camadas P2PSIP e a camada responsável pelo algoritmo DHT a utilizar. As camadas SIP e transporte foram implementadas recorrendo à biblioteca JAIN-SIP [20]. Esta biblioteca oferece um conjunto de API's que permitem facilmente criar uma aplicação capaz de enviar e receber mensagens SIP.

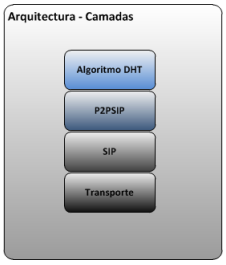


Fig. 4. Arquitectura das aplicações - Camadas

A camada P2PSIP é a responsável por efectuar a comunicação com a stack SIP para o envio e recepção de mensagens, oferecendo às camadas superiores . É composta por diversas classes, onde as mais importantes são a classe *SipLayer*, *ConvertToSIP* e *ConvertToP2PSIP*.

A classe *SipLayer* é a classe principal desta camada pois é a responsável por efectuar o envio de mensagens P2PSIP, assim como receber e enviar para a camada superior, mensagens P2PSIP recebidas. Para o envio de mensagens P2PSIP esta classe disponibiliza os seguintes métodos:

- *public void enviarPedidoP2PSIP(P2PSIPMessage msg, ITransactionListener callback)* - Este método é utilizado para o envio de um pedido P2PSIP, onde como

parâmetros recebe a mensagem P2PSIP a enviar, assim como uma referência para o objecto que deverá ser notificado quando for recebida uma resposta para a mensagem enviada.

- *public void enviarRespostaP2PSIP(P2PSIPMessage msg)* - Este método é utilizado para o envio de uma mensagem de resposta a um pedido P2PSIP recebido, como parâmetros tem apenas a mensagem P2PSIP a enviar como resposta ao pedido.

Ambos os métodos disponíveis para o envio de mensagens P2PSIP recorrem à classe *ConvertToSIP* para efectuarem a conversão do objecto *P2PSIPMessage* (que contêm a informação da mensagem P2PSIP a enviar) num objecto SIP reconhecido pela camada SIP em utilização.
Quando uma mensagem SIP é recebida por esta camada, é verificada se a mensagem é do tipo P2PSIP, analisando os seus cabeçalhos. No caso de a mensagem ser um pedido P2PSIP, é feita a conversão da mensagem SIP para um objecto *P2PSIP* equivalente, sendo este objecto enviado para a camada superior para ser processado. Caso a mensagem recebida seja uma resposta P2PSIP, após esta ser convertida num objecto P2PSIP equivalente, a mensagem é enviada directamente para o objecto que enviou o pedido original, de modo a que este a possa processar.

## V. Testes e Resultados

Os testes efectuados foram realizados com o auxilio do Common Open Research Emulator (CORE) [1].
O CORE é uma ferramenta ‘que permite emular redes, redes essas que podem se ligar a outras redes emuladas e/ou redes reais‘ [1].

Foi criada no CORE numa topologia composta por 3 Routers, 3 Switches e 31 Hosts (peers ou clientes). A figura 5 mostra como os vários componentes da topologia se encontram ligados. Os testes foram efectuados numa máquina com processador Intel Q6600 de 2.40 GHz e com 6GB de memória RAM. Um dos objectivos dos testes efectuados era a validação

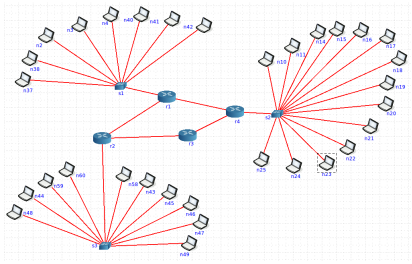

Fig. 5. Topologia utilizada para efectuar os testes

da implementação dos algoritmos Chord e EpiChord, por isso, foi necessário configurar a topologia de modo a que as características das ligações dos *peers* fossem idênticas às características das ligações dos *peers* nos testes efectuados pelos autores do EpiChord.
Para isso, nas ligações entre os Hosts e os Switches da topologia utilizada, foi definido um atraso de 80ms, fazendo com que o RTT(*Round Trip Time*) médio entre *peers* fosse igual ao valor utilizado nos testes efectuados pelos autores do EpiChord, 0.16seg.

Os testes efectuados focam-se no desempenho dos algoritmos em situações em que o volume de tráfego de localização de recursos é muito grande, onde cada *peer* efectua aproximadamente duas localizações por segundo. Por isso, para além das configurações na topologia, a aplicação desenvolvida foi também ela modificada para que cada *peer* efectuasse cerca de dois lookups por segundo. A lista com os recursos que cada *peer* deve tentar localizar, foi criada e armazenada em ficheiros previamente, de forma a que cada *peer* efectuasse sempre a mesma sequência de lookups em cada teste. Assim, os resultados obtidos em testes diferentes, podem ser comparados, uma vez que não há alterações nos lookups a efectuar que possam alterar os resultados.

Nos testes efectuados, alguns *peers* foram denominados como *peers* de *bootstrap*, isto é, todos os *peers* ao serem inicializados recebiam uma lista com os *peers* de *bootstrap*, aos quais se deveriam tentar ligar para se juntarem ao *overlay*. Cada *peer* ao inicializar estabelecia uma ligação com o primeiro *peer* da lista, sendo os outros utilizados caso o *peer* seleccionado não respondesse.
Para evitar que um servidor de *bootstrap* fosse o mais utilizado pelos restantes *peers*, foi utilizada uma técnica de *Round Robin*, sendo rodada a lista que cada *peer* recebe com os *peers* de *bootstrap*, desta forma preveniu-se que um *peer* ficasse inicialmente sobrecarregado devido a ser o primeiro da lista, e todos os outros estarem a tentar juntar-se ao *overlay* através dele.

A inicialização dos *peers* e clientes de cada teste foi efectuada recorrendo a um *script* desenvolvido neste trabalho, o qual inicializa a aplicação JAVA de cada o *peer* ou cliente com os vários parâmetros, de acordo com o teste a efectuar. Em primeiro lugar são inicializados os *peers* de *bootstrap*, de seguida os restantes *peers*. Se o teste utilizar clientes, estes são os últimos a ser inicializados. O endereço IP e porta dos *peers* de *bootstrap* estão pré-configurados no *script* utilizado para a inicialização dos *peers*. As aplicações JAVA inicializadas, recebem vários parâmetros, que permitem entre outras coisas, configurar o algoritmo DHT em utilização. Um desses parâmetros é utilizado para definir quanto tempo a aplicação deve esperar até iniciar os testes de localização de recursos. Visto que os testes que foram efectuados se focam no desempenho dos algoritmos na localização de recursos, o valor desse parâmetro foi definido em 90s. Desta forma quando os testes são inicializados o *overlay* encontra-se estabilizado, sendo por isso os resultados mais fiáveis.

Nos testes efectuados aos algoritmos Chord e EpiChord, para medir o desempenho destes na localização de recursos foram analisados os seguintes parâmetros:

- Tempo - Tempo médio necessário para localizar um recurso no *overlay*
- Número de saltos - Número de saltos necessários para se

localizar um recurso

- Timeouts - Percentagem de timeouts

Uma vez que os testes foram efectuados num ambiente real, não simulado, cada teste foi repetido três vezes. O resultado final de cada teste foi obtido através da média dos resultados das repetições efectuadas. Desta forma, os resultados dos testes são mais fidedignos, pois qualquer variação, numa das repetições, proveniente de um factor externo à aplicação, acaba por ser minimizada com os resultados das outras repetições.

### A. Validação da Implementação

A validação da implementação dos algoritmos Chord e EpiChord, foi conseguida através da realização de testes nos quais os 31 Hosts da topologia, foram utilizados como Peers. A utilização dos 31 Hosts como *peers* deve-se ao facto de que o EpiChord foi testado pelos seus autores num *overlay* composto exclusivamente por *peers*, não havendo o conceito de cliente. Por isso, nos testes efectuados nesta secção todos os 31Hosts da topologia são *peers*.

Para estes testes foram escolhidos três *peers*, um de cada sub-rede de *peers*, como *peers* de bootstrap.

Os parâmetros dos algoritmos Chord e EpiChord utilizados neste teste, foram os mesmos que os autores do EpiChord utilizaram nos seus testes, contudo, alguns parâmetros não puderam ser utilizador por não serem referidos nos testes do EpiChord, e/ou por a escala dos testes ser diferente, como por exemplo o número de Fingers a utilizar no Chord.

No caso do algoritmo Chord, foi definido um minuto (60seg) como o período de estabilização, isto significa que os *peers* verificam a cada minuto o estado do seus sucessores e antecessores directos. Para além do período de estabilização, foi também definido o período de tempo em que as entradas na *Finger table* devem ser verificadas. O valor deste parâmetro não se encontra especificado nos testes efectuados ao EpiChord, pelo que foi definido como 70seg, valor pertencente à gama de valores recomendados pelos autores da implementação Chord do dSIP [12]. O último parâmetro do algoritmo Chord a configurar é o número de entradas na *Finger Table*, sendo que os autores da implementação Chord do dSIP [12] recomendam que para redes pequenas se utilize 16 entradas na tabela, tendo sido este o valor utilizado nos testes efectuados.

Quanto à implementação EpiChord, foi utilizado o mesmo valor do Chord (60seg) para o período de estabilização, este valor, é o valor indicado pelos autores do EpiChord nos testes que estes efectuaram. Nesses mesmos testes, definem também que as entradas na cache devem ter um *lifetime* máximo de 120s, tendo por isso sido este o valor utilizado nos testes efectuados.

Relativamente ao nível de paralelismo na localização de recursos, foram efectuados testes com três níveis diferentes de paralelismo, tendo sido testadas as seguintes combinações: P=1 L=1, P=3 L=3 e P=5 L=3.

Um outro parâmetro, comum a ambos os algoritmos, definido pelos autores do EpiChord nos seus testes, é o valor do *timeout*, tendo estes definido 0.5seg. Contudo, com esse valor de *timeout*, o número de perdas de mensagens era demasiado elevado, pelo que nos testes efectuados o valor definido para o timeout foi de 5seg.

A razão pela qual o valor utilizado pelos autores do EpiChord para o *timeout* parece não funcionar na prática deve-se a diversos factores, como o facto de esse valor ter sido utilizado em simulação e não num ambiente real. O facto de que neste trabalho o EpiChord ser implementado sobre um outro protocolo, o SIP, e que, apesar de a aplicação ter sido desenvolvida para suportar concorrência (*multi-threading*), a própria stack SIP utilizada definia como valor óptimo de *timeout* 32seg. Por isso, neste trabalho o valor de *timeout* utilizado foi de 5 segundos.

Os resultados deste teste, encontram-se na tabela I. Através da analise dos resultados da tabela I, e comparando-os com os resultados obtidos pelos autores do EpiChord nos seus testes (Fig 11 e 12 de [13]) é possível verificar que a implementação dos algoritmos Chord e EpiChord parece estar correcta. Os valores obtidos encontram-se dentro dos valores esperados, mesmo tendo em conta a diferença no número de *peers* utilizado neste teste.

| | **Chord** | **EpiChord (P=1 L=1)** | **EpiChord (P=3 L=3)** | **EpiChord (P=5 L=3)** |
|---|---|---|---|---|
| **Lookups (seg)** | 0,547 | 0,158 | 0,153 | 0,156 |
| **Num Saltos** | 3,122 | 0,932 | 0,899 | 0,897 |
| **Timeouts (%)** | 0,622 | 0,000 | 0,000 | 0,000 |

TABLE I
RESULTADOS DO TESTE DE LOOKUP INTENSIVO

Para além de comprovarem o funcionamento das implementações, estes resultados mostram também que a utilização do EpiChord, permite melhorar o desempenho da localização de recursos, melhorando em cerca de 71% o tempo de localização de recursos, assim como melhora cerca de 70% em média o número de saltos necessários para se localizar um recurso.

### B. Peers com limitações

Para verificar se um *overlay* beneficia da existência de uma hierarquia composta por *peers* e clientes, onde os clientes podem ser *peers* com menores capacidades. Foram efectuados testes onde se introduziram *peers* com limitações ao nível da conectividade. De forma a que a comparação dos resultados fosse mais justa, estes testes foram efectuados nas mesmas condições dos testes efectuados em V-A, onde a única diferença existente tem a ver com as limitações introduzidas em alguns dos *peers* da topologia.

As limitações introduzidas na conectividade de alguns *peers*, foram a alteração da largura de banda passando de 10Mbps para 256Kbps, e também o atraso das suas ligações foi alterado de 80ms para 200ms. Desta forma, o RTT dos *peers* do *overlay* varia entre os 0.16 e os 0.40 seg.

Para verificar o impacto deste tipo de *peers* no *overlay*, foram criados diferentes cenários, onde se foram introduzindo mais *peers* com limitações, de modo a avaliar qual o impacto destes

no desempenho do *overlay* na localização de recursos.
Os cenários criados neste teste foram os seguintes:

- Cenário 1 - Foram colocadas limitações na conectividade de 5 dos 31 Peers, equivalente a 16% dos *peers*.
- Cenário 2 - Foram colocadas limitações na conectividade de 10 dos 31 Peers, equivalente a 32% dos *peers*.
- Cenário 3 - Foram colocadas limitações na conectividade de 15 dos 31 Peers, equivalente a 48% dos *peers*.

A tabela II contêm os resultados obtidos neste teste, para os três cenários descritos. Analisando os dados da tabela II, e

| | **Chord** | **EpiChord (P=1 L=1)** | **EpiChord (P=3 L=3)** | **EpiChord (P=5 L=3)** |
|---|---|---|---|---|
| Cenário 1 - 5 Peers Limitados | | | | |
| **Lookups (seg)** | 0,611 | 0,183 | 0,177 | 0,181 |
| **Num Saltos** | 3,097 | 0,934 | 0,899 | 0,898 |
| **Timeouts (%)** | 0,727 | 0,000 | 0,000 | 0,000 |
| Cenário 2 - 10 Peers Limitados | | | | |
| **Lookups (seg)** | 0,749 | 0,224 | 0,215 | 0,225 |
| **Num Saltos** | 3,118 | 0,934 | 0,899 | 0,897 |
| **Timeouts (%)** | 0,790 | 0,000 | 0,000 | 0,000 |
| Cenário 3 - 15 Peers Limitados | | | | |
| **Lookups (seg)** | 0,872 | 0,266 | 0,256 | 0,271 |
| **Num Saltos** | 3,009 | 0,932 | 0,899 | 0,897 |
| **Timeouts (%)** | 0,906 | 0,000 | 0,000 | 0,000 |

TABLE II
RESULTADOS DO TESTE DE *lookup* INTENSIVO - CENÁRIO 1, 2 E 3

comparando-os com os da tabela I é facilmente perceptível, que a existência de *peers* com limitações na conectividade influência o desempenho do *overlay* na localização de recursos.

Como era expectável, foi no cenário 3 que se verificou o maior aumento no tempo da localização de recursos, pois é neste cenário que se encontra o maior número de *peers* com limitações. O valor médio do tempo de localização de recursos neste cenário, aumentou cerca de 59% no Chord, variando de 67% a 73.7% nas variantes do EpiChord.
Nos restantes cenários é também evidente um aumento no tempo médio da localização de recursos, onde como também era esperado, se vê que o aumento do tempo de localização de recursos está directamente ligado ao aumento do número de *peers* com limitações.

## VI. CONCLUSÃO

Através da análise aos resultados dos testes efectuados, e comparando-os com os resultados obtidos pelos autores do EpiChord, concluímos que os algoritmos implementados se encontram a funcionar correctamente. Os resultados mostram também que o aumento do número de *peers* com limitações no *overlay*, influência o desempenho global do *overlay* na localização de recursos. Tendo sido verificada uma maior influência negativa, no algoritmo EpiChord, os tempos deste algoritmo, apesar de continuarem inferiores ao do algoritmo Chord, sofrem um maior aumento em percentagem.

Para trabalho futuro, e uma vez tendo sido provado o bom funcionamento da aplicação desenvolvida, pretendemos efectuar testes num ambiente real, mas com um número significativamente maior de hosts(*peers* e clientes). Dessa forma, poderemos analisar num cenário mais realista, as vantagens e desvantagens da utilização do SIP para a criação de um *overlay* P2P assim como avaliar melhor o desempenho dos algoritmos Chord e EpiChord num *overlay* com e sem dois níveis hierárquicos.

## REFERENCES


[1] J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim, "CORE: A real-time network emulator," in *IEEE Military Communications Conference*, 2008.

[2] G. Camarillo, "Peer-to-Peers (P2P) architecture: Definition, taxonomies, examples, and applicability," Nov. 2009.

[3] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys and Tutorials*.

[4] T. Klingberg and R. Manfredi, "The gnutella protocol specification v0.4," Jun. 2002.

[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," 2001.

[6] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," *Peer-to-Peer Systems*, p. 53?65, 2002.

[7] C. Jennings, S. Baset, H. Schulzrinne, B. Lowekamp, and E. Rescorla, "A SIP usage for RELOAD," *Internet Draft*, Jul. 2010. [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-sip-05

[8] D. Bryan and B. Rosen, "Peer-to-peer session initiation protocol (p2psip) - work group." [Online]. Available: http://tools.ietf.org/wg/p2psip/

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

[10] H. Schulzrinne, M. Matuszewski, and S. Baset, "Peer-to-Peer protocol (P2PP)," *Internet Draft*, Nov. 2007. [Online]. Available: http://tools.ietf.org/html/draft-baset-p2psip-p2pp-01

[11] C. Jennings, S. Baset, H. Schulzrinne, B. Lowekamp, and E. Rescorla, "REsource LOcation and discovery (RELOAD) base protocol," *Internet Draft*, Oct. 2010. [Online]. Available: http://tools.ietf.org/html/draft-ietf-p2psip-base-13

[12] D. Bryan and M. Zangrilli, "A chord-based DHT for resource lookup in P2PSIP," Feb. 2007. [Online]. Available: http://tools.ietf.org/html/draft-zangrilli-p2psip-dsip-dhtchord-00

[13] B. Leong, B. Liskov, and E. D. Demaine, "Epichord: Parallelizing the chord lookup algorithm with reactive routing state management," *Computer Communications*.

[14] D. Bryan, B. Lowekamp, and C. Jennings, "Sosimple: A Serverless, Standards-based, P2P SIP Communication System," 2005.

[15] G. Camarillo, M. Handley, J. Peterson, J. Rosenberg, A. Johnston, H. Schulzrinne, and R. Sparks, "SIP: session initiation protocol." [Online]. Available: http://tools.ietf.org/html/rfc3261

[16] D. Wing, P. Matthews, R. Mahy, and J. Rosenberg, "Session traversal utilities for NAT (STUN)," Oct. 2008. [Online]. Available: http://tools.ietf.org/html/rfc5389

[17] J. Rosenberg, R. Mahy, and P. Matthews, "Traversal using relays around NAT (TURN): relay extensions to session traversal utilities for NAT (STUN)," Apr. 2010. [Online]. Available: http://tools.ietf.org/html/rfc5766

[18] J. Rosenberg, R. Mahy, and P. Matthews, "Interactive connectivity establishment (ICE): a protocol for network address translator NAT traversal for offer/answer protocols," Apr. 2010. [Online]. Available: http://merlot.tools.ietf.org/pdf/rfc5245.pdf

[19] S. Cirani and L. Veltri, "A kademlia-based DHT for resource lookup in P2PSIP," http://tools.ietf.org/html/draft-cirani-p2psip-dsip-dhtkademlia-00, Oct. 2007. [Online]. Available: http://tools.ietf.org/html/draft-cirani-p2psip-dsip-dhtkademlia-00

[20] M. Ranganathan, P. O'Doherty, and J. van Bemmel, "JAIN-SIP: Stack sip for java." [Online]. Available: http://jsip.java.net/