Peer to Peer Communication over the Internet
An Open Approach

Swapnil Dinkar Pundkar

swapnil@iitg.ernet.in

Senior Undergraduate

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati, INDIA

Developed at Telecom Italia, Turin, Italy during MAY – JULY 2007

Under the guidance of

Mr. Enrico Marroco

Table of Contents

1 Introduction

Voice over Internet Protocol (VoIP) [25] is a technology that allows making voice calls using a broadband Internet connection instead of a regular (or analog) phone line.

VoIP is an example of Real Time Communication (RTC) where RTC is the communication made in the real time i.e., with no delay, is completed before the deadline is met and the deadline is event specific. Since VoIP communication is carried over the internet, VoIP services convert the voice into a digital signal that travels over the Internet and the call is automatically routed to the VoIP Phone, regardless of where you are connected to the internet. Some VoIP providers offer their services for free, normally only for calls to other subscribers to the service, normally charged calls can also be made from a VoIP phone to public switched telephone network (PSTN) and vice-versa. VoIP (IP Telephony) is more beneficial than traditional telephony because it provides features like presence and instant messaging along with making cheaper voice calls. The presence feature enables the user to find, retrieve and subscribe to changes in the presence information (i.e. online or offline) of other users. Instant messaging is means of sending short text messages that are delivered immediately to other online users. Although Voice over IP services offer important revenue-generating opportunities, as well as many technical challenges in providing high-quality services. The widespread use of IP telephony presents many problems to the developers such as routing VoIP traffic through Network Address Translators (NATs) [22] and Firewalls

Structured P2P (peer-to-peer) technologies using DHT (Distributed Hash Table) are cheaper alternative than age-old centralized client server system. Network Address Translation (NAT) causes difficulties for peer-to-peer (P2P) communication, since the peers involved may not be reachable at any globally valid IP address. Reliability and security issues cannot be overlooked.

1.1 Objective of the Thesis

This document elaborated at Telecom Italia Labs, Turin, Italy from May to July 2007 focuses on key issues in development of XPP – Extensible Peer Protocol designed to be used in peer to peer services to provide internet telephony services. This document intends to give a brief overview of the SIPDHT open source project describing the history, motivation and high level abstract details of the project. This document does not describe the implementation details and specifications of the protocol and is recommended to have a brief overview of the project.

The following section describes the two popularly used VoIP implementations Skype and SIP. It is then followed by introduction to the P2PSIP workgroup. The next section elaborates on the SIPDHT2 project describing the underlying DHT and the protocol developed. The document then describes the software developed.

1.2 Overview of the VoIP software

This section describes the most used VoIP implementations highlighting the main characteristics and technical features.

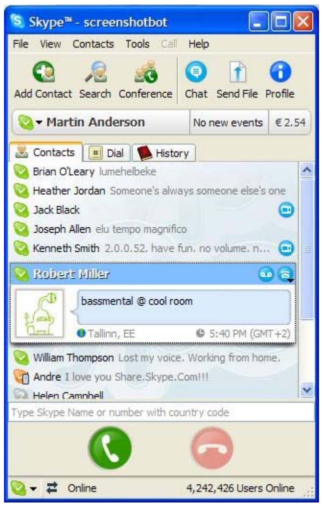
1.2.1 Skype

Skype [10] is first software to implement VoIP in a completely decentralized distributed fashion. The study of Skype is important because it is able to work across firewalls and NATs with least amount of infrastructure as it uses the Skype peers to route the calls.

Skype is a peer-to-peer VOIP application providing services over the Internet like placing voice calls, audio conferencing, instant messaging, buddy lists etc and is developed by the organization that created Kazza [11] and was acquired by eBay in Sep, 2005. Skype is able to work across firewalls and NATs and has better voice quality than other existing VOIP clients. Skype also offers paid services that allow Skype users to initiate and receive calls via regular telephone numbers through VoIP-PSTN gateways.

Skype uses an peer-to-peer network, which contains two types of nodes, Ordinary hosts and Super nodes. An ordinary host is a Skype application, which can be used to place voice calls and send text messages. A super node is an ordinary host’s endpoint on the Skype overlay i.e., it is used by the ordinary host to connect to the overlay. Any node with a public IP address having sufficient CPU, memory, and network bandwidth is a candidate to become a super node. Skype makes use of supernodes to keep track of normal peers or clients. A client registers on a supernode and can connect to the overlay only through the supernode. An ordinary host (client) must register itself on a Skype login server by providing necessary authentication details. The Skype login server stores the usernames and passwords and ensures that they are unique across the Skype namespace. The Skype Login server is not a part of the overlay. There are also SkypeIN [15] and SkypeOUT [14] servers, which provide PSTN to PC and PC to PSTN bridging respectively.

To determine the type of firewall and NAT it is behind, a Skype node uses a variant of STUN [17] protocol and this information is refreshed periodically. Skype uses wideband codecs, which allows it to maintain reasonable call quality at an available bandwidth of 32 kb/s. It uses TCP for signaling, and both UDP and TCP for transporting media traffic. When either the Skype users trying to communicate are behind a firewall or a NAT the information is relayed by a third Skype node to the individual nodes. This helps specially in case of a conference in which the third node can be used to broadcast the information to the other nodes. Figure 1 is an example of the Skype screenshot: the main view gives a quick overview of the user’s contacts and their status (i.e., offline or not). It also tells if there are any missed calls, voicemails or messages, and how much Skype Credit the user has left.



Martin Anderson | No new events | € 2.54

Brian O'Leary lumehelbeke
Heather Jordan Someone's always someone else's one
Jack Black
Joseph Allen elu tempo magnifico
Kenneth Smith 2.0.0.52. have fun. no volume. n...
Robert Miller
bassmental @ cool room
Tallinn, EE 5:40 PM (GMT+2)
William Thompson Lost my voice. Working from home.
Andre I love you Share.Skype.Com!!!
Helen Campbell

Online 4,242,426 Users Online

Figure 1 Example of Skype screenshot

From a technical point of view, the popularity of Skype is mainly due its features described in the following sections

A. Robustness towards NATs and Firewall

A Skype client sends the control traffic (requests) over the super node overlay peer-to-peer network. If the request is accepted a direct connection is established between the peers. If one of the peers is behind a firewall or NAT, Skype uses connection reversal technique in which the peer behind the NAT initiates a TCP/UDP session regardless of which node initialized the VOIP or a File Transfer request. If both the peers are behind the NAT or firewall Skype uses STUN [17] like NAT traversal to establish a direct connection. If the direct connection fails, Skype uses a TURN [19] like approach where media session is relayed by publicly reachable supernode.

B. An effective login method

A Skype client with a public IP address and the one behind a port restricted NAT take about 3-7 seconds to complete the login procedure and the one behind a UDP-restricted firewall took about 35 seconds. For immediate subsequent logins it takes about 5-10 seconds [16]. Skype claims to have implemented a ‘3G P2P’ or ‘Global Index’ [18] technology, which guarantees to find a user if that user has logged in the Skype network in the last 72 hours. The Skype Client performs user information caching at intermediate nodes. When a call is put on hold a Skype client sends UDP packets three times a second to ensure that the UDP bindings with the NAT box are alive.

C. Security

Skype uses Advanced Encryption Standard (AES). Skype uses 256-bit encryption, which has a total of 1.1 x 10e77 possible keys, in order to encrypt the data in each Skype call. Skype uses 1024 bit RSA to negotiate symmetric AES keys. The Skype server at login using 1536 or 2048-bit RSA certificates certify user public keys.

Skype relies heavily on its super node overlay network to relay the messages. Skype doesn’t allow the user machine to prevent itself from becoming a Super node. But a node cannot become a super node if has limited bandwidth. So if every Skype user puts a limitation on its application bandwidth the Skype network will possibly collapse. Skype is a proprietary solution for today’s growing use of Internet Telephony.

1.2.2 Session Initiation Protocol (SIP)

SIP [7] is developed by Internet Engineering Task Force (IETF) [27] for VoIP [28] and other text and multimedia sessions like instant messaging, online games, videos etc. It is an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants such as multicast conferences and it works independently of the underlying protocol and the session that is established. SIP works in concert with several other protocols like Real Time Transport Protocols (RTP) [38] and Message Session Relay Protocol (MSRP) [36] and is involved in the signaling portion of a communication session i.e. call setup and tear down. SIP itself doesn’t know anything about the details of the session. SIP just conveys information about the protocol used to describe the session. SIP is usually used with Session Description Protocol (SDP) [37], which is used to describe the parameters of the session to be established.

A. Architecture

The SIP architecture consists of following elements.

- SIP User Agent: Caller Application that initiates and sends SIP requests.
- SIP Proxy: Helps to route requests to the user's current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users.
- SIP Registrar Servers: Accept the registration request from SIP user agents and store their registration. Pass on the registration information to Location server when asked for.
- SIP Location Servers: Provide information about the possible location of the SIP User Agent
- SIP Redirect Servers: Provides the information about the next hop the client should make and then the client contacts the next hop server or proxy directly.
- SIP Gateways: Used as translators between SIP user agents and other terminal types. For example, translation of a call from IP telephony to PSTN.

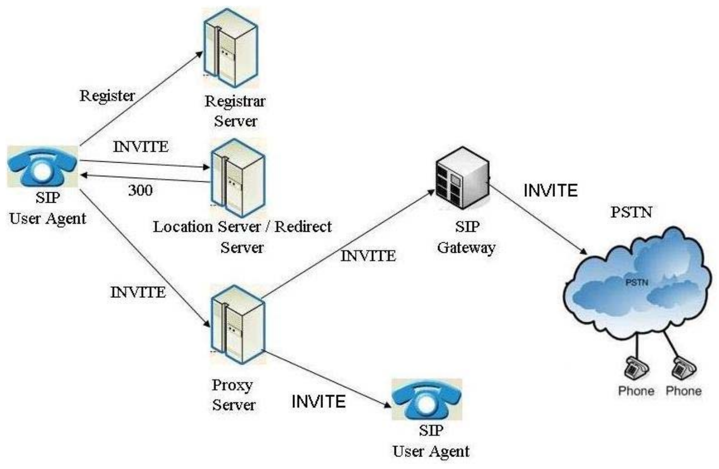


Figure 2 SIP Architecture

B. How does SIP work?

SIP users are identified using a SIP address, which is very similar to the email address. If the user's e-mail ID is bob@sip.com then its SIP address would be sip:bob@sip.com which is also called Universal Resource Identifier (URI). The user first registers on a registrar server, which can also be integrated into a proxy server. SIP works on the request response paradigm. To initiate a session the caller (SIP User Agent) sends an INVITE request to the SIP Server (Proxy or Redirect). The SIP server is then responsible for searching the addresses of the clients on which the other SIP user agent has registered and then sends out one or more than one INVITE requests to different addresses. The SIP proxy may forward the request to other SIP proxy or redirect server. The called party on receiving the INVITE request sends back a response, which is forwarded back through the exact reverse path followed by the request. The response contains the call parameters whether to accept or reject the call. SIP provides many response types like accept, reject, busy, and hang. If accepted the session is now active. SIP can also be used to modify the session i.e. adding or removing things like audio streams, codecs, hold and mute the call. Finally SIP is used to terminate the session.

SIP is not NAT and firewall friendly. SIP uses tools like STUN [17] and ICE [31] to establish sessions through NATs and firewalls.

C. Basic SIP Call Flow

Consider the scenario, which contains SIP user agents A and B and SIP Proxies 1 and 2. User Agent A calls the User Agent B using the proxies. User agent A sends an INVITE to Proxy 1, which forwards it to Proxy 2, which forwards it to User agent B. The initial INVITE does not contain the Authorization credentials that Proxy 1 requires and so an authorization response is sent containing the challenge information. After the call setup the voice call begins using RTP. User agent B ends the session by sending a BYE, which is sent on the same path followed by the request.

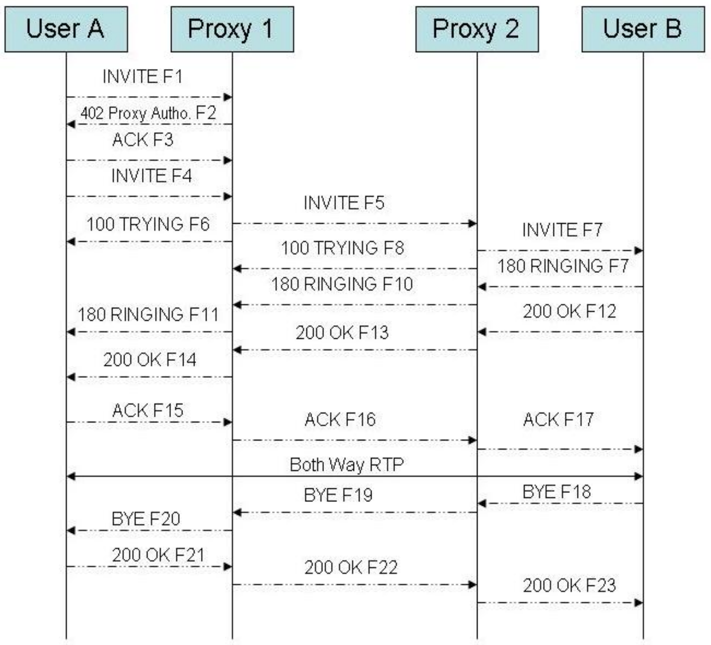


Figure 3 Basic Call Flow in SIP

1.3 P2PSIP

The P2P nature of Skye allows it to cut costs on infrastructure management and provide cheap call relates to the users. But Skype uses proprietary protocols which cannot be used by other organizations to develop applications. SIP on the other hand is an open source project. The peer-to-peer SIP working group (P2PSIP WG) [30] formed in IETF aims at developing concepts and terminology for use of the SIP in a peer-to-peer environment where the traditional proxy-registrar function is replaced by a distributed mechanism that might be implemented using a Distributed Hash Table (DHT) [39] or other distributed data mechanism with similar external properties. P2PSIP aims at exploiting the P2P concept in SIP in order to save on costs as already implemented by Skype. P2PSIP also aims at inducing freedom of talking to anyone in the system without intervention of higher authority, unlike the centralized systems, which handle, keep track and listen to every bit of data that you send to other peer(s), which is an added advantage of P2P systems. The use of DHT would reduce the resource look up time, which is high in unstructured P2P systems.

A P2PSIP overlay consists of P2PSIP nodes (peers) which are organized in P2P fashion for purpose of enabling real time communication using SIP. The peers in the overlay collectively run a distributed database algorithm and use a (DHT) to organize the data. The overlay stores the mapping between SIP URIs and Contact URIs (IP address or an address which resolves into an IP addresses though DNS or SIP proxy) for the distributed location service. Data is stored in the overlay in a distributed form using the algorithm. Data duplication can be done to achieve fault tolerance. The P2PSIP WG is in search of a solution on the basis of above requirements

1.4 Distributed Hash Table

Distributed hash tables are a class of decentralized distributed systems that provide lookup service to applications and try to optimize the resource lookup latency by using various distributed algorithms like CAN, CHORD, Pastry etc. The <name, value> pairs are stored in the DHT similar to a hash table. The participating nodes are responsible for maintaining the mapping from names to values. The pairs are distributed in such a way that a change in the set of participating nodes causes a minimal amount of disruption thus allowing DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures. Distributed hash tables use a more structured key based routing in order to attain both the decentralization and the efficient and guaranteed results. Many complex applications like BitTorrent, Overnet, YaCy run over the DHT infrastructure. We use an underlying DHT Passive Content Addressable Network (PCAN) [4] for lookup and structured storage of data. The existing peers control and initiate the join of new peers in order to prevent malicious peers damaging the overlay, so the name passive-CAN.

2 SIPDHT 2

SIPDHT 2 is an open source project and is a candidate solution for P2PSIP. It aims to develop a public API, which can be used by other applications to build Real Time Systems. SIPDHT 2 uses SIP as its signaling protocol. Tools like STUN and ICE are used to connect peers behind NATs.

2.1 Overview

SIPDHT 2 aims at developing library which provides tools for creating and using Sip based Distributed Hash Tables. SIPDHT 2 uses a DHT known as Passive Content Addressable Network (PCAN), which is a variant of Content Addressable Network (CAN). The DHT is passive in nature wherein the clients do not participate actively in the overlay. They do so only when invited by an existing peer in the overlay. The peers in the overlay act like Sip Proxies and are used to locate the node on which the user is located by forwarding the requests to the appropriate peers. The distributed overlay is maintained using the lightweight binary protocol using PCAN as its underlying DHT. The protocol provides functionalities for setting up the session, exchanging information within the peers, a new client joining the overlay and call tear down.

2.1.1 Passive Content Addressable Network (PCAN)

PCAN is a variant of the distributed hash table CAN [1]. In this DHT the clients do not actively participate in the overlay but do so only when invited by one of the existing peer (i.e. member of the overlay). CAN is preferred over other DHTs because

1. CAN is symmetric, which allows the connection to be accessed equally by both the peers.
2. In CAN the peers maintain a stable routing table with limited number of entries.

A possible drawback when using the CAN algorithm is its relatively low performance. While other popular algorithms claim to always be logarithmic in complexity, overlays based on CAN cannot scale indefinitely. CAN based overlays need to be configured during deployment with parameters (e.g. the number of dimensions for the hash space) depending on the size of the intended network. In our P2P overlay only a subset of interested nodes provide the service and by use of delay measurements between the peers the best path can be chosen which would then result in acceptable performance.

A. Why passive?

In the P2P overlay implementing PCAN, nodes are allowed to become member of the overlay only when invited by the existing peers of the overlay. The passive approach benefits in the following ways.

- The clients to be invited have already been registered on the existing peers, so the overlay has full information about the clients it is going to invite.
- Providing SIP connectivity to clients, storing their location, routing messages, in addition to maintaining tens or even hundreds of connections with neighboring peers could be very resource consuming. It is therefore important to confirm availability of such resources prior to inviting a client to join the overlay which would otherwise result in degradation of performance with the low functioning peers creating a bottleneck in the overlay.
- It is also important to confirm whether the client joining the overlay is not malicious. It is possible to limit the effects of malicious peers by only inviting the trusted peers.
- The passive approach allows the overlay to select among the available peers the best candidate peer based on performance and security characteristics of the peer.

B. Brief Description

Like all other DHTs, PCAN is based on mapping of keys onto peers. Such a functionality, unlike in algorithms based on the concept of consistent hashing like CHORD, PASTRY, KADEMLIA [22,23,24] is implemented in a virtual d-dimensional Cartesian coordinate space on a d-torus which is a d dimensional ring and is used for efficient node searching. Any peer is assigned a distinct zone of the overall space and is authoritative for all the keys falling in it. Zones are always defined by the coordinates of their lower left and top right corner, and contain the area locked between the borders including the bottom and left edges (thus excluding the right and top edges). At any point in time, the overlay is consistent if the whole space is completely covered. Figure 4 shows a bi-dimensional space partitioned into 5 zones among 5 peers. Zone A is directly connected to Zone B, so there is direct path between these two zones which would not exist in absence of the torus. A peer maintains XPP sessions with all its immediate neighbors, along with information about the zones they are authoritative for. When a peer receives an operation request for a key which falls out of its zone, it routes the request to the most appropriate neighbor.

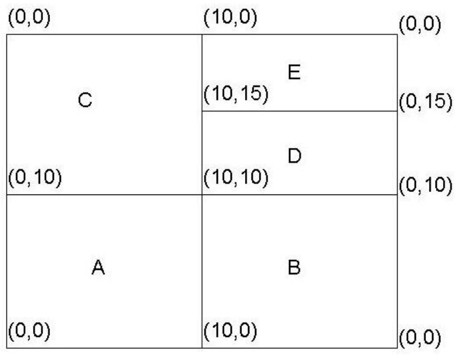


Figure 4 PCAN Overlay

Following are the types of operations used to maintain the overlay:

a) PUT

It is used to insert a <key, value> into the distributed database and is generally used to store SIP registrations. If a peer receives a PUT request it should first check that whether is it responsible for the KEY parameter in the first tuple of the PUT request. If yes then it should update or insert the tuple Otherwise it should route the request to the peer nearest to the target peer.

b) GET

It is used to retrieve data from the distributed database and contains one or more KEY parameters. These are generally used to query the location of the peers and clients for routing SIP messages. If a peer receives a GET request, it should first check whether it is responsible for the first KEY parameter in the request. If yes then it should send a GET response containing all records in its local table bound to keys matching one of those contained in the initial request. Otherwise it should route the request to the peer nearest to the target peer.

c) JOIN

It is sent by a peer in the overlay to the joining peer informing it the co-ordinates of the zone transferred and the list of its new neighbors. The JOIN request cannot be routed and is sent end to end.

d) QUERY

This operation is used to query the status of a peer which is authoritative for a zone in which the given point falls, which is specified in the request. If the given point in the request doesn't fall in the peer's zone it should forward it to the peer nearest to the target peer.

e) UPDATE

This operation is used by the neighboring peers to exchange the status information upon a change. This request must not be routed and is sent end to end.

f) TAKEOVER

This request is sent by a peer to its neighbors stating that a peer (specified in the request) is a most suitable candidate for taking over the zone. This request must not be routed and sent end to end.

2.1.2 Overlay Maintenance:

The distributed overlay formed using PCAN is maintained in the following way.

A. Clients joining the overlay

Any conventional SIP User agent can connect to the overlay. Once such a client has discovered peers which are already part of the overlay, it can set these peers as its SIP outbound proxy. The client should connect to a bootstrap peer to get the list of peers it can register at. It should then register using the peer as a registrar and following the registration procedures described in [9]. If needed, the SIP client may direct SIP outbound flows [8] to this peer in order to allow NAT traversal for SIP messages.

A client may be invited to join the overlay if there's necessity of resources.

B. Inviting a client to join the overlay

Due to resource limitations a peer may invite one of the existing clients for which it stores the registration. To do this the peer first selects the best option from the list of registrations, which it stores. The inviting peer must send a SIP INVITE request to the joining peer so that it can establish a connection with it. In network environments where it is expected that peers might be located behind NAT devices, the session negotiation should be completed using the ICE [31] mechanism. If the inviting peer is able to setup a connection with the client (now new peer) then the join procedure would continue as follows:

1. The inviting peer transfers all the information related to the new zones (the data bound to keys located in the new zone) to the joining peer by sending a PUT request.
2. The inviting peer would send an UPDATE request to all its neighbors informing them about the changes in the zones it owned and the zones owned by the new peer.
3. The inviting peer sends a JOIN request to the joining peer informing it the co-ordinates of the zone transferred and the list of its new neighbors.
4. The joining peer establishes new connections or sessions with all its neighbors.
5. The inviting peer now updates its zone list and terminates the sessions with those peers which are no longer its neighbors.

C. Failure Behavior

Its neighboring peer detects the failure of any peer. The neighboring peer after the detection of the failure starts a timer so that other neighboring peers can detect the failure too. When the timer is fired the peer starts the election process. The neighbors of the failed peer start exchanging messages and every peer stores an identity of the most appropriate peer it sees for overtaking the region of the failed peer. After the Stabilization interval the new peer is elected. The newly peer then exchanges messages with its new neighbors so that it can establish connections with them.

D. Routing Overlay Operations

Overlay operation requests are routed through the peers until it reaches its final destination. Whenever a peer receives a GET, PUT or QUERY operation request for a key which is not in one of its zones, it forwards the query to the next peer which is geometrically closest to the target zone. However a variant of this implementation may consider other factors such as link speed. In any case, in order to avoid loops, peers MUST NOT route messages to neighbors which are not geometrically closer than them to the destination zone. The responses should be routed back on the same path followed by the request and so the peers should maintain some amount of routing history. This document doesn't describe how long the routing history should be stored on the peers.

2.2 Extensible Peer Protocol (XPP)

XPP is a lightweight binary application layer transfer protocol which is designed to be NAT and Firewall friendly i.e. it can be used to connect and transfer data between peers behind NATs and Firewalls. XPP uses UDP as its transport protocol and uses simultaneous establishment of connection using SIP to connect two peers. XPP uses tools like ICE [31] and STUN [17] that further facilitate session establishment. XPP is developed by the SIPDHT2 group on the basis of P2PSIP concepts [2].

2.2.1 Simultaneous Session Establishment

In today's Internet scenario most of the firewalls allow only outgoing TCP and UDP traffic, blocking the incoming connections from an unknown user. But if a peer behind the firewall has initiated an outgoing connection with a peer on the other side of the firewall then the incoming connections from that peer are accepted and routed to the designated peer by the NAT. XPP uses this functionality of NATs and firewalls to establish a session between the peers behind them.

Consider two peers A and B behind different NATs and firewalls. If any one of the peers tries to establish a session with the other, on the other end the firewall will block the incoming request (TCP or UDP) thus preventing the establishment of the session. When both the peers simultaneously send connection requests to each other both of them open ports in their respective firewalls for incoming connections from the other peer. So finally when peer A and peer B simultaneously send connection requests and suppose that the firewall at peer A's end receives the request from peer B after it has sent peer B the connection request, it would accept the incoming connection and the NAT would route the request to peer A corresponding to the <IP, port> mapping stored in its routing table.

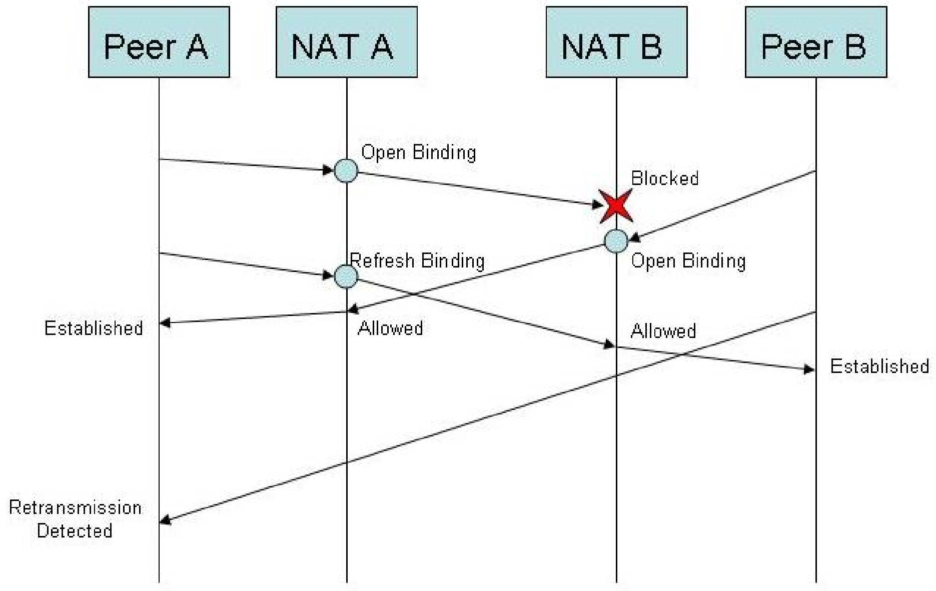


Figure 5 Simultaneous Session Establishment

2.2.2 Why UDP?

XPP uses UDP as its transport protocol because it is best suited for simultaneous session establishment used to traverse NATs.
The three way TCP handshake is a problem in the simultaneous session establishment when both the peers have public addresses and fails many times. To implement TCP in XPP you need to implement something like inherently complex ICE-TCP [32], inducing complexity in the protocol. TCP is useful in transport of larger pieces of data, but P2P protocols do not create much continuous traffic to consider TCP for XPP.

Use of UDP facilitates turning off the reliability, which is required when using DHT algorithms based on frequent optional routing table updates.

2.2.3 XPP Session

Before initiating a XPP session both the peers generate tags that are locally unique. A session is then identified on a peer by <local ID, remote ID> tuple where the tags are the local ID and remote ID on the local side respectively. So a session can be uniquely identified on a peer by this pair.

A XPP operation is a set of XPP request followed by zero or more XPP responses. A XPP response is always result of a XPP request i.e. a peer cannot receive a XPP response without sending a XPP request. Every operation is uniquely identified by use of sequence number and operation type which are contained in the XPP request and response. These tags match for request and response pairs.

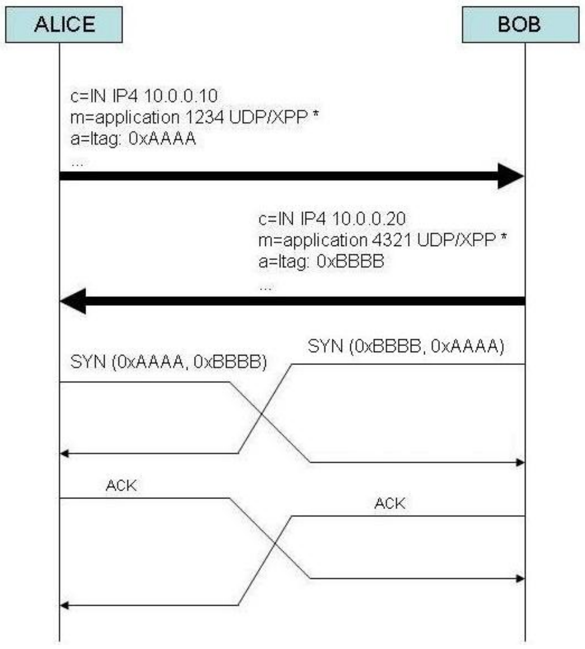


Figure 6 Session Establishment in XPP

2.2.4 XPP Session Establishment and Maintenance:

Before establishing a XPP session the parameters of the session like the remote ID's on both the ends are exchanged using mechanisms like SIP and SDP. Once the required information is exchanged and UDP holes are punched into the firewall the peers are ready to setup a XPP session. The peers exchange information using XPP after session establishment. A peer acknowledges every XPP operation by sending an ACK after receiving a request or a response.

When storing data in an overlay, or when simply exchanging information on neighboring zones, P2P applications are likely to have to exchange data chunks exceeding the path MTU. XPP therefore defines mechanisms for fragmentation that allow sending long XPP messages over multiple XPP fragments. For every XPP fragment received, the peer sends an ACK to other side and asks for the next fragment if any. In case the sending peer does not receive an ACK it retransmits the XPP fragment.

In case of a session teardown the peer who wants to end the session should send a SIP BYE request containing parameters of the corresponding INVITE request using which the session was started.

In case of session failure, the failure must be reported to the application by the XPP protocol stack. The failure may be as result of expiration of keep-alive timeout or loss of network connectivity.

A XPP operation starts by sending the request. An expiration delay ‘D’ is specified to the XPP stack for the request being sent. The protocol stack would consider the operation to be terminated after ‘D’ since the last response for that operation was received after which all responses received are discarded by the protocol stack or are directly notified to the application. The expiration delay is implementation specific.

XPP messages are transported using UDP as its transport protocol. Depending on the purpose of the message it can be transported in a reliable or unreliable way. When the size of the message exceeds the path MTU it must divided into fragments and sent in different UDP packets. XPP fragments are transmitted one at a time. Using UDP as a transport implies that intermediate devices may drop some fragments. Reliability is therefore accomplished through XPP fragment retransmissions.

XPP uses keep-alive messages in order to maintain the NAT bindings i.e. keep alive the XPP session between peers.

3 SIPDHT 2 Software

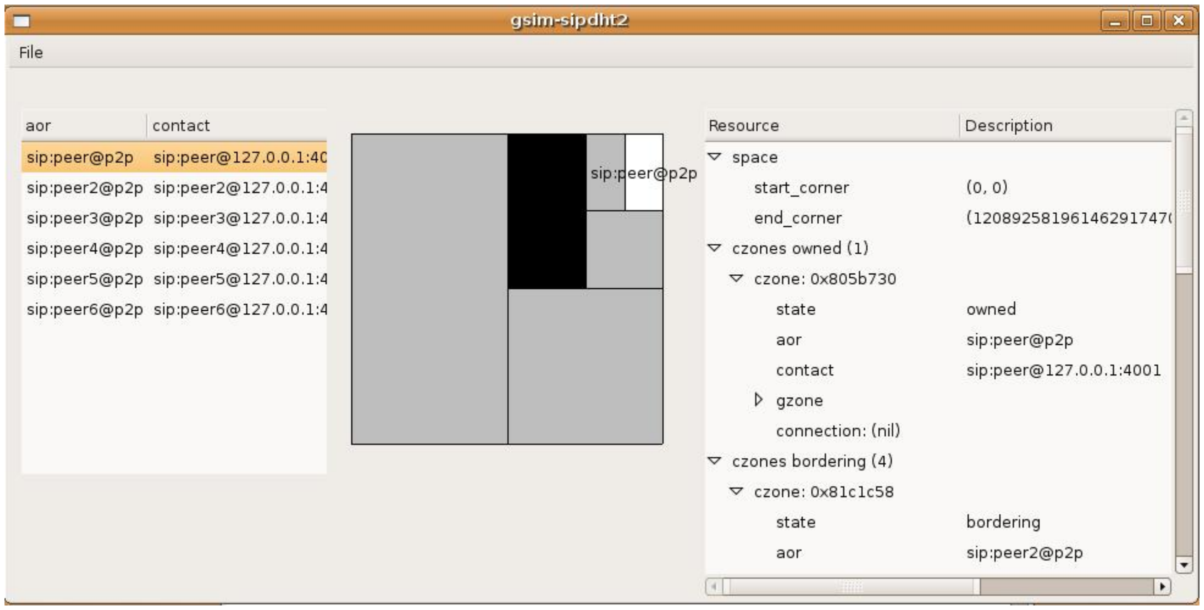
The software developed by the SIPDHT 2 group is based on the Sofia-Sip library, which is an open-source SIP User-Agent library developed at the Nokia Research Center. The software is developed using the ideas and information collected after days of discussion in the open source community. This library is used for implementing SIP in P2P fashion. The software is developed using the XPP public API and has following features.

1. A SIPDHT Client registering on an existing peer in the distributed overlay.
2. Form and maintain an overlay of the existing peers using XPP.
3. Inviting a client to join the overlay.
4. Exchange information (in form of messaging) between peers in the overlay and also the clients, which have registered onto the peers but are not part of the overlay.

The software is available in textual as well as graphical user interface. It consists of the following modules which are downloadable from SourgeForge(http://sourceforge.net/project/showfiles.php?group_id=166130).

1. libxpp – It is a lightweight XPP library developed using the sofia-sip library so the sofia-sip library is required before using it.
2. libsipdht2 – It is a lightweight PCAN implementation using libxpp.
3. csipdht2 - Command line application implementing a PCAN node and uses libsipdht2 library.
4. gsipdht2 – Graphical application implementing PCAN node and uses libsipdht2 and sofia-sip library.
5. gsim-sipdht2 – Graphical application simulating an overlay with arbitrary number of PCAN nodes and uses libsipdht2 and sofia-sip library.

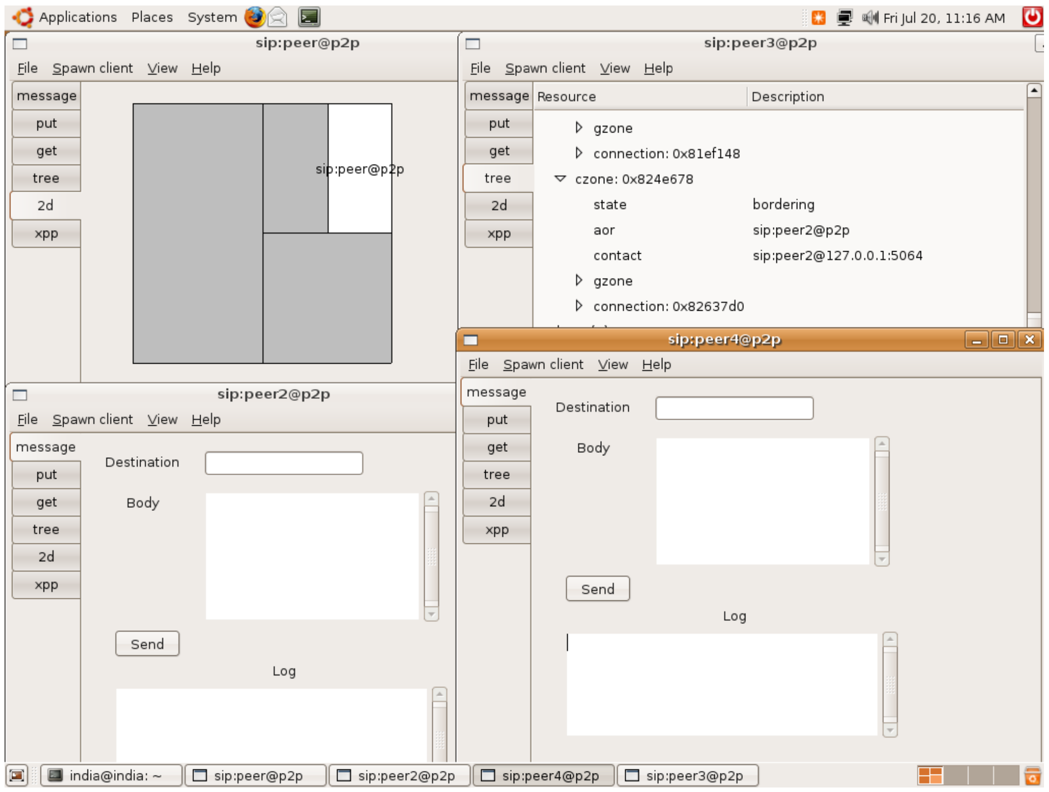
3.1 Screenshots



aor	contact
sip:peer@p2p	sip:peer@127.0.0.1:40
sip:peer2@p2p	sip:peer2@127.0.0.1:4
sip:peer3@p2p	sip:peer3@127.0.0.1:4
sip:peer4@p2p	sip:peer4@127.0.0.1:4
sip:peer5@p2p	sip:peer5@127.0.0.1:4
sip:peer6@p2p	sip:peer6@127.0.0.1:4

Resource	Description
▽ space	
start_corner	(0, 0)
end_corner	(120892581961462917470
▽ czones owned (1)	
▽ czone: 0x805b730	
state	owned
aor	sip:peer@p2p
contact	sip:peer@127.0.0.1:4001
▷ gzone	
connection: (nil)	
▽ czones bordering (4)	
▽ czone: 0x81c1c58	
state	bordering
aor	sip:peer2@p2p

Figure 7 Gsimdht2 – Screenshot



```
Resource                    Description
▷ gzone
▷ connection: 0x81ef148
▽ czone: 0x824e678
    state                   bordering
    aor                     sip:peer2@p2p
    contact                 sip:peer2@127.0.0.1:5064
▷ gzone
▷ connection: 0x82637d0
```

Figure 8 Gsipdht2 – Screenshot

3.2 Test Driven Approach

While developing software many decisions are to be made which is a very difficult task. Test-First Development or Test-Driven Development (TDD) [34] is a methodology that uses tests to help developers to develop software. TDD uses tests to create software in a simple incremental way. This helps in speeding up the development and also increases the quality of the code developed. TDD takes a simple incremental approach to the development of software. This is a less risky approach then trying to build the entire system all at once hoping it will work when all the pieces are put together.

One of the most important feature of TDD is Constant Regression Testing which can be used to minimize the time required to correct a software struck by the domino effect [35]. The domino effect is very well known in software development. Sometimes a simple change to one module may have unforeseen consequences throughout the rest of the project. This is why regression testing is important. Regression testing is like self defense against bugs. Regression testing is usually done only when a new release is sent to Quality Analysis. By then it is some times hard to trace which code change introduced a particular bug and makes it harder to fix. TDD runs the full set of unit tests every time a change is made to the code. In effect TDD runs a full regression test every time a minor change is made. This means that any change to the code that has an undesired side effect will be detected almost immediately and be corrected. This should prevent any regression surprises when the software is handed over to QA. The other benefit of constant regression testing is that you always have a fully working system at the end of the every iteration of development. This allows you to stop development at any time and quickly respond to any changes in requirements.

TDD also results in Improved Communication and Improved Understanding of Required Software Behavior, Centralization of Knowledge, Improved Software Design, Better Encapsulation and Modularity and Simpler Class Relationships.

3.2.1 Basic building blocks of TDD

TDD uses the following steps to develop a better and more understandable code.

- Unit testing allows you to test the software in a more detailed way than you would with functional tests. With unit tests you can test small units of the code at the API level. Unit tests are also much easier to run than functional tests since they don't require a full production environment to run and can be run quickly by the developer as he/she is developing.
- Refactoring is the process of changing code for the sole benefit of improving the internal structure of the code without changing its function. Refactoring is basically cleaning up bad code.

3.3 My contribution

My work was to apply the Theory of Test Driven Development (TDD) to the project. Accordingly the advanced Test Programs were developed. These tests programs are used to check the different functionalities of the public XPP API. The tests are run every time a change is made to the library whether it is the change in the implementation or a minor change in the code. Every time a change is made it is assured that the tests pass. The tests check various parameters used in the functions in the public XPP API. The tests also check the functionality of the library as a whole.

3.3.1 Development of test programs

Following advanced test programs were developed.

A. Test Message

This test is used to check the parameters in a XPP message. This test creates a session and establishes a connection with other node. It then creates a message and adds parameters to it. The message is then sent to other the other end and then it is verified that the same parameters are present in the received message. This test also makes a copy of the message developed and verifies that the same parameters present in the copy developed of the message. The cloned message is then sent over the session established and then parameters are again verified as done in the previous case. This test is important because it checks the validity of the message sent over the overlay and messages are used for maintenance and data transfer in the overlay. An error in the message sent would result in improper functioning of the overlay thus resulting in breakdown of the complete system.

B. Test Stack

This test is used to test the functionalities of the XPP stack (not to be confused with the conventional meaning of stack). A XPP stack is created which is then used to store and handle all the details of a session. The XPP stack is responsible for functionalities like keeping track of all the requests sent and the corresponding responses received. It also takes care of the time intervals after which all the responses received for the request sent must not be considered and in turn triggers an error. The following scenario is checked in this test.

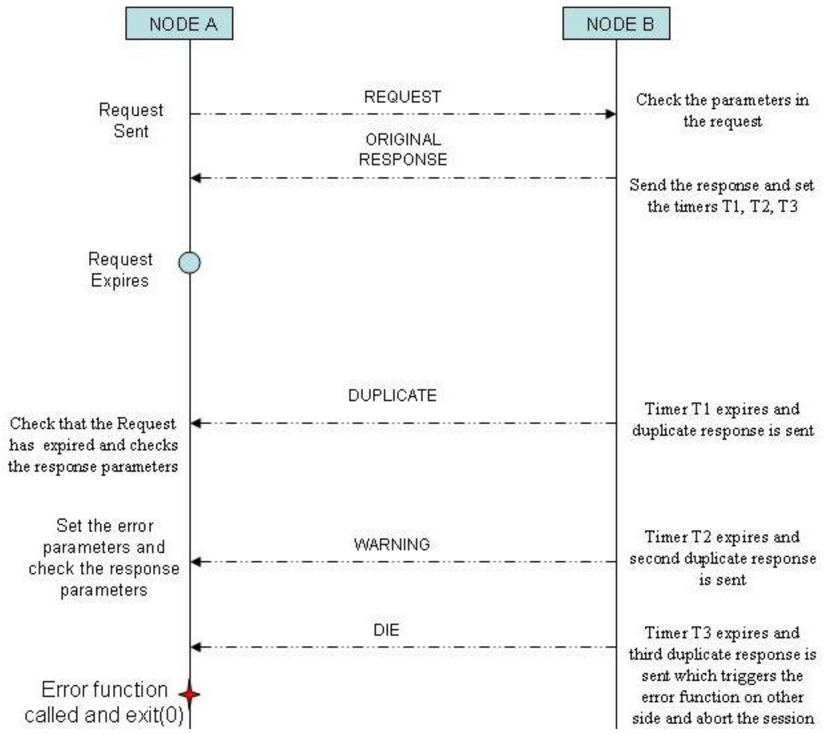

NODE A
NODE B
Request Sent
REQUEST
Check the parameters in the request
ORIGINAL RESPONSE
Send the response and set the timers T1, T2, T3
Request Expires
DUPLICATE
Timer T1 expires and duplicate response is sent
Check that the Request has expired and checks the response parameters
Set the error parameters and check the response parameters
WARNING
Timer T2 expires and second duplicate response is sent
DIE
Timer T3 expires and third duplicate response is sent which triggers the error function on other side and abort the session
Error function called and exit(0)

Figure 9 Test Scenario – Test_stack

3.3.2 Testing the API using Autotools

Autotools is a suite of tools provided by the GNU-project which assists in making the source code packages portable to many UNIX like systems. Autoconf is a tool for producing the configuration scripts generally known by the name "configure". Automake is a tool for automatically generating "Makefile.in". "Makefile.in" file is a configuration template from which configure generates the configuration files used for building the source package.

Test programs were integrated into the API by specifying their respective paths in the file “Makefile.am” which is used by the Autotools (Automake and Autoconf) to generate the configuration scripts. So every time the API is build using the Makefile generated by Autotools, the test programs are executed to check the integrity of the API.

3.3.3 Application Development

An application was developed using the XPP API as a working model demonstrating use of XPP to connect the peers and transfer data between them. Queries such as request for time on the other peer were implemented in the application representing the data transfer. The application also demonstrated the proxy support of the XPP API. The peers first registered on the proxy and then requests were sent to the proxy in order to initiate connections. The application is run on the terminal and is not available in GUI (Graphical User Interface).

Execution of the application provides the user a XPP prompt, which can then be used to give commands like dial (invite), accept, hang-up, decline or busy. Once the session has been established the user can query about the time at the other node by giving command at the XPP prompt. The XPP prompt gives the information about the status of the user whether it is connected to other node along with the session ID, which uniquely identifies the session. At any time after the session has been established the user can hang-up the call (or session). Once the call has been hanged-up at one end, the other end is notified accordingly. In order to get the details of the underlying SIP messages used to handle the sessions, the environment variable TPORT_LOG has to be set to 1.

4 Conclusion

Structured P2PSIP systems in general tend to solve the problems of scalability, high churn rate but external mechanisms such as STUN and ICE are required to establish connection across NATs and Firewalls which is the demand of today’s IP telephony. XPP fulfills the problem of scalability by using PCAN and the passive approach helps in adding security. Use of UDP makes it possible for XPP to easily go across NATs and make the protocol light. SIPDHT 2 group aims at finding solutions to many of the issues in the peer to peer services to provide IP telephony. in general is targeted towards many of the issues in IP-telephony.

5 Acknowledgments

I am thankful to my guide Mr Enricco Marocco for being available when needed with wholehearted support, Ms Antonieatta Manella for reviewing and precious feedback. Finally I thank Telecom Italia for giving me this opportunity.

6 References

[1] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Shenker: A Scalable Content Addressable Network (Jan 2001)

[2] D. Bryan, P. Matthews, E. Shim, and D. Willis, Concepts and Terminology for Peer to Peer SIP draft-ietf-p2psip-concepts-00 (June 2007)

[3] E. Marocco and E. Ivov, Extensible Peer Protocol (XPP) draft-marocco-p2psip-xpp-00 (June 2007)

[4] E. Marocco and E. Ivov, XPP Extensions for Implementing a Passive P2PSIP Overlay Network based on the CAN Distributed Hash Table draft-marocco-p2psip-xpp-pcan-00 (June2007)

[5] E. Marocco and D. Bryan, Interworking between P2PSIP Overlays and Conventional SIP Networks draft-marocco-p2psip-interwork-0 (March 2007)

[6] Kundan Singh and Henning Schulzrinne: Peer-to-Peer Internet Telephony using SIP (Jan 2005) Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)," draft-ietf-sip-gruu-12 (work in progress), March 2007

[7] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)," draft-ietf-sip-outbound-07 (work in progress), January 2007

[8] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R.,Handley, M., and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, June 2002

[9] "SIPDHT: A SIP-based Distributed Hash-table." www.sipdht.org

[10] Skype. http://www.skype.com

[11] Kazaa. http://www.kazaa.com

[12] SkypeOut. http://www.skype.com/products/skypeout/

[13] SkypeIn. http://www.skype.com/products/skypein/

[14] Baset, S. and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol".

[15] J. Rosenberg, J. Weinberger, C. Huitema AND R. Mahy. STUN: Simple traversal of user datagram protocol (UDP) through network address translators (NATs). RFC 3489, IETF, Mar. 2003.

[16] Global Index (GI): http://www.skype.com/skype_p2pexplained.html.

[17] Rosenberg, J., Mahy, R., AND Huitema, C. Internet draft: TURN Traversal Using Relay NAT, (Mar. 2006) Work in progress.

[18] S. Guha, N. Daswani, and R. Jain, "An experimental study of the skype peer-to-peer Voip system," in Proceedings of the IPTPS'06, Santa Barbara, CA, (Feb. 2006)

[19] Rowstron, A. and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large- scale peer-to-peer systems."

[20] Stoica, I., Morris, R., Karger, D., and Frans Kaashoek, M., "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications."

[21] Maymounkov, P. and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric." Proceedings of IPTPS, Cambridge, USA, March 2002

[22] Network Address Translators (NATs) http://en.wikipedia.org/wiki/Network_address_translation

[23] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, September 2001

[24] IETF http://www.ietf.org/

[25] VOIP http://en.wikipedia.org/wiki/Voip

[26] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz : Tapestry: A Resilient Global-Scale Overlay for Service Deployment (Jan 2004)

[27] P2PSIP : http://www.p2psip.org

[28] Rosenberg, J., “Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,” draft-ietf-mmusic-ice-13 (work in progress), January 2007
[29] Rosenberg, J., “TCP Candidates with Interactive Connectivity Establishment (ICE),” draft-ietf-mmusic-ice-tcp-03 (work in progress), March 2007
[30] Rescorla, E. and N. Modadugu, “Datagram Transport Layer Security,” RFC 4347, April 2006.
[31] Rick Mugridge Department of Computer Science, University of Auckland, New Zealand: Test Driven Development and the Scientific Method Agile Development Conference (ADC '03), 2003
[32] J. D. (Denny) Carreker: The domino effect: A white paper 2004 http://www.carreker.com/main/media/art_wtpapers/2004/Domino_Effect_White_Paper_2004.pdf
[33] Message Session Relay Protocol – http:// xml.coverpages.org/draft-ietf-simple-message-sessions-03.txt
[34] M. Handley and V. Jacobson: SDP: Session Description Protocol (April 1998)
[35] Real Time Transfer Protocol - http://en.wikipedia.org/wiki/Real-time_Transport_Protocol
[36] Distributed Hash Table – http://en.wikipedia.org/wiki/Distributed_hash_table