San Jose State University
SJSU ScholarWorks

Master's Projects | Master's Theses and Graduate Research

1-1-2010

Security in Peer-to-Peer SIP VoIP

Richa Marwaha
San Jose State University

Follow this and additional works at: http://scholarworks.sjsu.edu/etd_projects

Recommended Citation
Marwaha, Richa, "Security in Peer-to-Peer SIP VoIP" (2010). *Master's Projects.* Paper 160.

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Security in Peer-to-Peer SIP VoIP

A Writing Project
Presented to
the Faculty of the Department of Computer Science
San José State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
by

Richa Marwaha

September 2010

Abstract

VoIP (Voice over Internet Protocol) is one of the fastest growing technologies in the world. It is used by people all over the world for communication. But with the growing popularity of internet, security is one of the biggest concerns. It is important that the intruders are not able to sniff the packets that are transmitted over the internet through VoIP.

Session Initiation Protocol (SIP) is the most popular and commonly used protocol of VoIP. Now days, companies like Skype are using Peer-to-Peer SIP VoIP for faster and better performance. Through this project I am improving an already existing Peer-to-Peer SIP VoIP called SOSIMPLE P2P VoIP by adding confidentiality in the protocol with the help of public key cryptography.

Contents

1. Abstract …….2

2. Introduction ……..6

2.1 Existing Architectures in VoIP…….7

2.1.1H.323 Architecture …….....7

2..1.1.1H.323 Protocol …….....7

2.1.1.2H.225 Protocol…….....10

2.1.1.3H.245 Signaling …….....12

2.2 SIP Architecture …….....13

3. Peer-to-Peer VoIP…….15

3.1 Introduction……..15

3.2 Structure Overlay/Distributed Hash Tables ……...15

3.3 Chord Algorithm……..16

3.3.1 Chord ……...16

3.3.2 Finger Table……....16

3.3.3 Predecessor Peer …….....16

3.3.4 Successor Peer……...16

3.3.5 Hash Algorithm and Identifier……....16

3.3.6 DHT Linker Header…….....16

3.3.7 Link Type and Depth Value……...16

3.4 Chord Overlay Algorithm……..19

3.4.1 Finger Table, Predecessor Peer, and Successor Peer……20

3.5 Peer-to-Peer Overlay Structure……...20

3.6 Session Establishment…….20

4. Initial Proposal ……20

5. Security Threats……..22

5.1 Security Threats in VoIP……...22

5.2 Security Threats in SIP…….23

6. SOSIMPLE: A Serverless, Standard-based P2P SIP VoIP……24

6.1 Introduction…….24

6.2 Background Challenges……...25

6.2.1 SIP and SIMPLE……...25

6.2.2 Scenarios Requiring a New Approach……25

6.3 Requirements…….25

6.4 SOSIMPLE P2P SIP VoIP Architecture…….26

6.4.1 Structure and Message……...26

6.4.2 Node-Level Operation ……27

6.4.3 User Operation ……28

6.4.5 Security and Authentication ……..28

7. My Approach…….29

7.1 Improvement over Existing Protocol……...29

7.2. Security and User Authentication…….……

8.Implementation……..…...32

9. Performance Graphs…….....41

10. Kademlia Algorithm in SOSIMPLE ……..42

11. Comparison between Chord and Kademlia……45

12. Conclusion…..……..46

13. Appendix ……..46

14. Future Work ……47

15. References ……....47

2. Introduction

Voice over Internet protocol (VoIP) is used to transmit voice communications over the Internet using IP packets.

Voice —-- > ADC —-- > **INTERNET** —-- > DAC —- > Voice
(Source) (Analog to Digital Convertor) (Digital to Analog Convertor) (Destination)

Figure 1. VoIP functionality

The following are the steps to set up VoIP:

1. We convert the analog voices into digital signal (Bits) using Analog Digital Convertor (ADC).
2. After the conversion, the digital signals have to be compressed and in a good format; there are many protocols that can be used to transmit these bits.
3. Here the voice packets must be inserted into data packets using a real-time protocol.
4. A signaling protocol is needed to call users.
5. At the receiver end, the data packets are disassembled, extracted, converted back to the analog signal.
6. Everything is done at real-time as the users cannot wait too long for a vocal answer.

Voice)) ADC —-Compression Algorithm —-Assembling RTP in TCP/IP
|
|
——-->
<——-
|
Voice ((DAC —--Decompression - -- Disassemble
Algorithm RTP from TCP/IP

Figure 2: Base Architecture

2.1 Different Architectures of VoIP

2.1.1 H.323 Architecture

2.1.1.1 H.323 Protocol

H.323 protocol is one of the first multimedia conferencing protocols that includes voice, video, and data conferencing. H.323 protocol uses packet-switching network. This architecture was one of the first standards for VoIP because of its complex design; it is not as popular as SIP architecture. It has a client/server model in which there is a gateway and a gatekeeper. The sessions created using H.323 architecture can be dynamically adjusted and attributes modified such as encoding and decoding formats for the media, bandwidth required for the session, and data types (such as voice or video). The biggest disadvantage of this architecture is that it does not provide guaranteed quality of service (QoS). The H.323 stack is implemented at the application layer and it contains the following:

- H.225 and H.245 for signaling control
- Audio codec
- H.263 and H.261 for video codec
- T.120 series used for data transmission

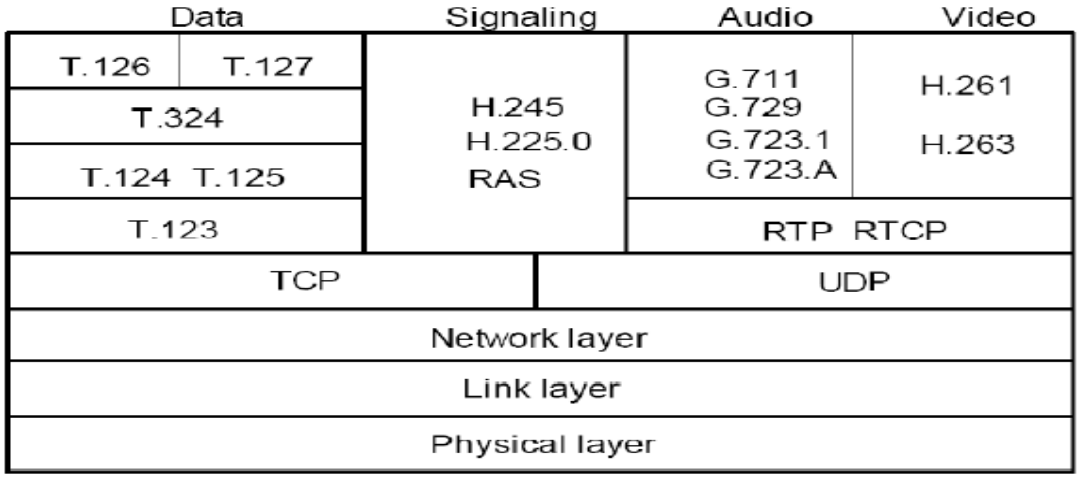


Figure 3. H.323 Stack

H.323 architecture has the following:

- Terminals
- Voice gateway
- Gatekeeper (optional)
- Multipoint control unit (MCU)
- Border elements

a) Terminals

Include telephones, VoIP phones, IVR phones, voicemail phones, and soft phones.

b) Voice gateways

Gateways contain 2 parts:

- Media gateway controller (MGU), helps to handle call signal and other non-media-related media
- Media gateway (MG), which helps in handling media

c) Gatekeeper (optional)

Even though Gatekeeper is optional but it acts like the brain of the architecture. It provides the following features:

- Call control services
- Call signaling routing
- It helps in monitoring the call made by the gatekeeper and helps in controlling the calls that made in network.
- It also helps to make the routing decision based on various factors.
- IP telephony

d) Multipoint control unit (MCU)

Helps in managing the multipoint conference. It does it with the help of the Multipoint Controller (MC) which looks at calls signaling and can also do multipoint processing with the help of multiple processors (MP).

e) Border and peer elements

Peer elements does exchange-addressing information, it also participate in doing call authorization not just within but also between administrative domains. Border elements are a special type of peer element. They are between two administrative domains.

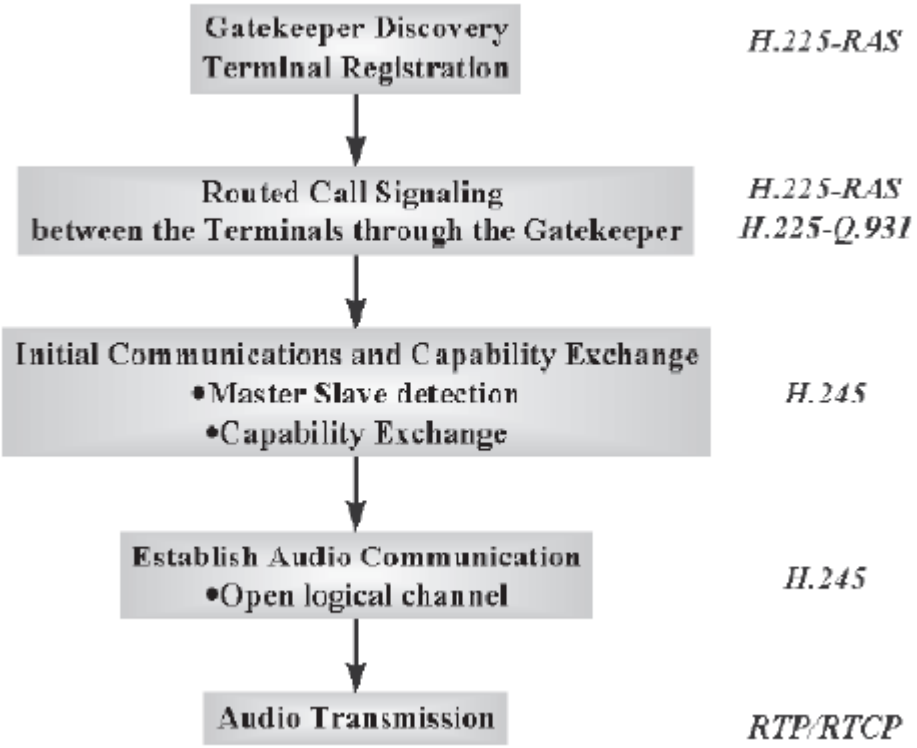


Figure 4. H.323 Connection and Session Setup

2.1.1.2 H.225 Protocol

The H.225 protocol is one of the protocols that have been specified by H.323 architecture. It contains registration, admission, and status (RAS). It is also known as the H.225 RAS protocol.

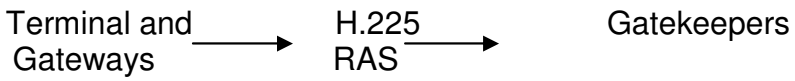


Figure 5. Basic architecture of H.225 Protocol

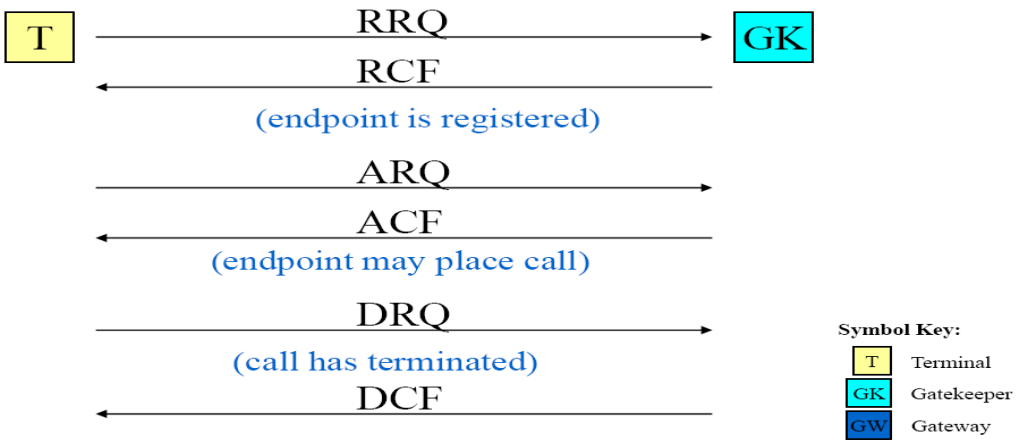


Figure 6. General Protocol of H.225 Registration, Admission, and Status

The purposes of this protocol are:

- Admission control
- Access control
- Assistance in gatekeeper discovery (GRQ)
- Assisting in locating and registering the endpoints

The biggest disadvantage of this protocol is that the channel through which the message is passed is highly unreliable and the message exchange can sometimes lead to timeouts.

2.1.1.2.1 H.225 Call Signaling:

H.225 call signaling is a protocol that uses established calls between two H.323 entities. It also contains information that helps in carrying additional information that can be related to specific messages. It allows a user to initiate and end a call with other users.

The users use entities to exchange messages. They use a setup entity to set up a VoIP call between the users; with the help of call processing, they create the voice call between the users. After exchanging these entities, user 2 to which user 1 wants to connect gets an alert that user 1 is trying to call him, and once user 2 agrees, the protocol uses a connect entity to create a connection between the users. With the help of information entities, they are able to send voice data to each other. The users can also use entities such as status, status inquiry, progress, and notify, with the help of which they know how the call between the users is going. They terminate the call using release complete. With the help of these entities in the protocol, the users are able to connect, terminate, and exchange information themselves.

2.1.1.3 H.245 Signaling

H.245 signaling is a protocol that helps in handling conference calls. It helps in negotiating the capabilities and controlling aspects of a conference among many users.

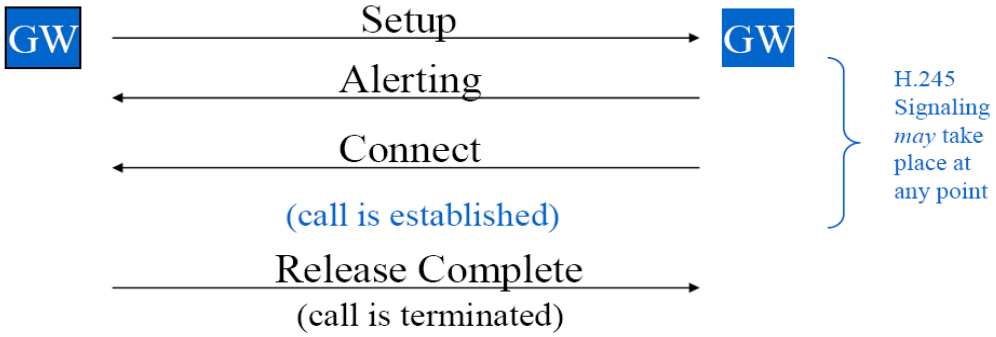


Figure 7. General Protocol of H.245 Signaling

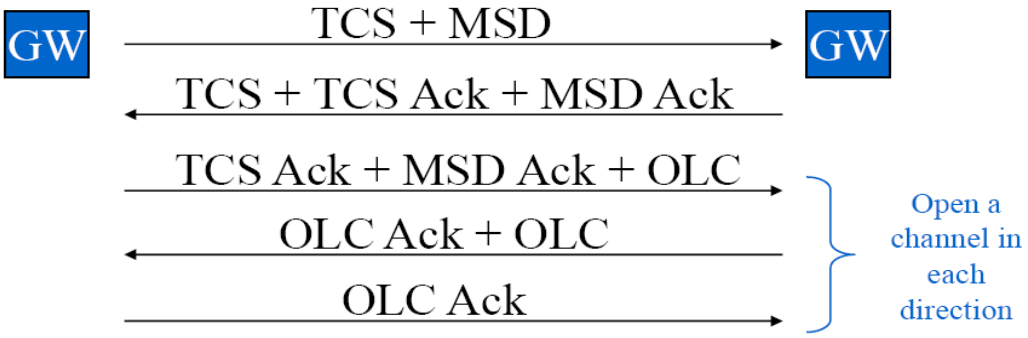


Figure 8. Message Exchanged Using H.245 Signaling

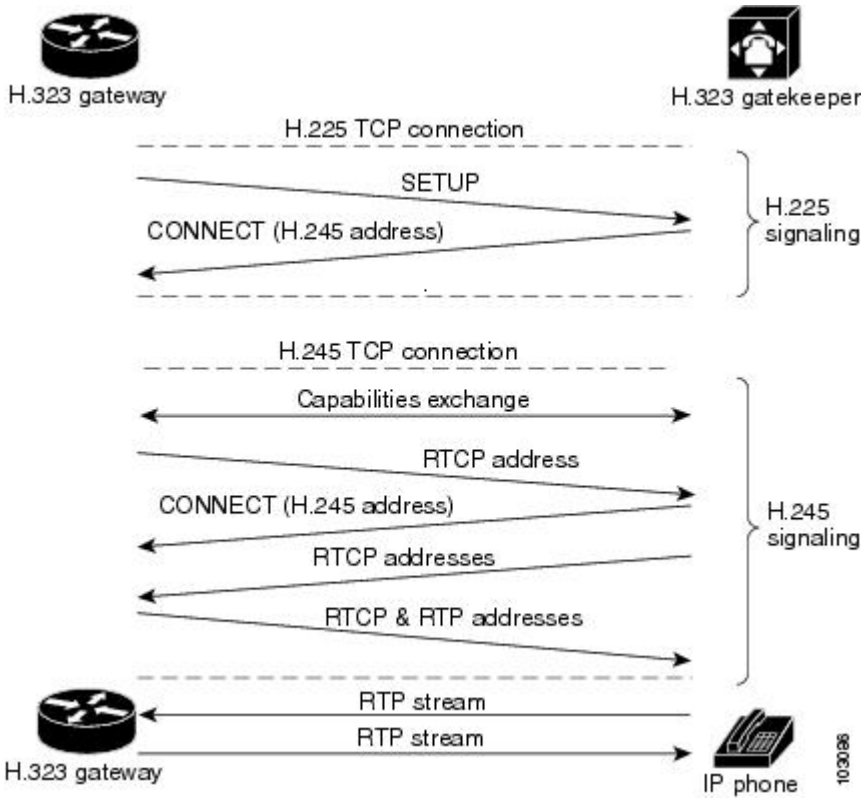


Figure 9. Complete Architecture of H.323 Protocol

2.2 Session Initiation Protocol (SIP) Architecture

The architecture of SIP is based on a simple HTTP protocol of request/response exchange. Its simplicity is the reason that SIP is more popular compared to H.323 architecture.

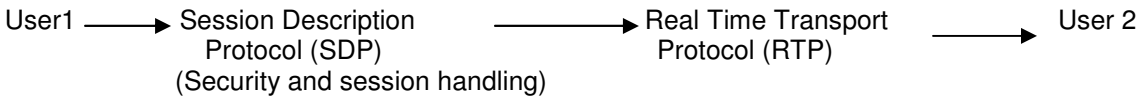


Figure 10. Basic Architecture of Session Initiation Protocol

SIP converts voice into VoIP, which can be done in the following ways:

- Codec in PSTN
- Voice gateway function, which includes packetization, silence suppression, echo cancellation, jitter buffering, DTMF, fax, and modem

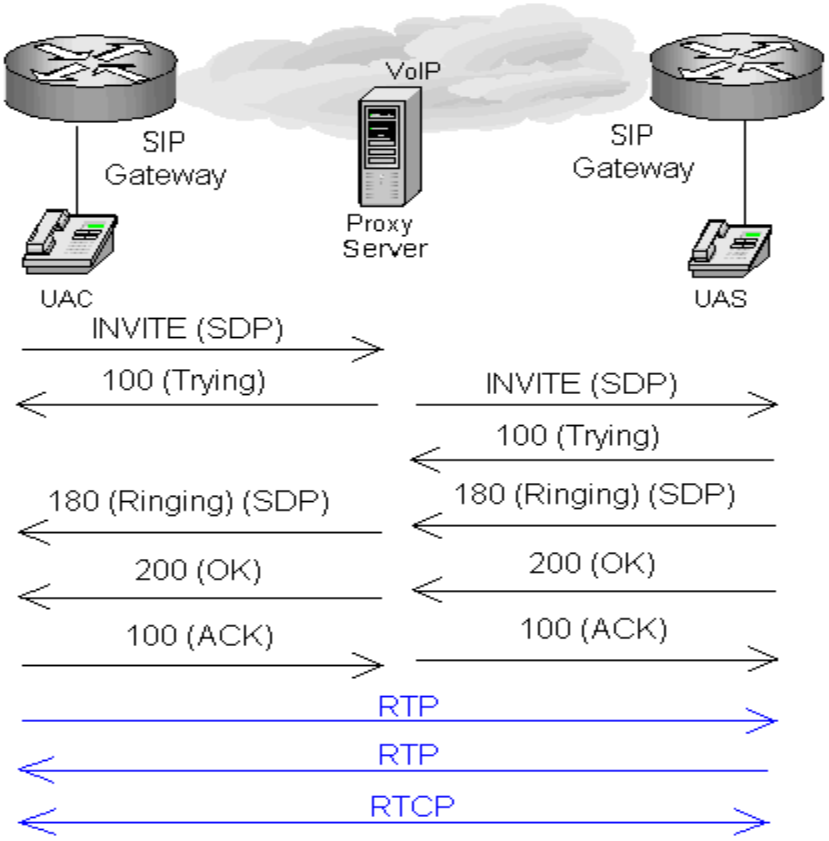


Figure 11. Basic SIP Architecture

SIP invokes the following methods:

1) INVITE – initiate call
2) ACK – confirm final response
3) BYE – terminate (or transfer) call
4) CANCEL – cancel searches and “ring”
5) OPTIONS – feature support by the other side
6) REGISTER – register with location services
7) INFO – mid-call information
8) COMET – precondition met
9) PRACK – provisional acknowledgement
10) SUBSCRIBE – subscribe to an event
11) NOTIFY – notify subscribers
12) REFER – ask recipient to issue SIP request

3. Peer-to-Peer SIP (P2P SIP) VoIP

3.1 Introduction

In P2P SIP network it does not uses servers to find the user location. Instead, it uses distributed hash table (DHT) to find the user and to register the user. DHT is more scalable, readily available, and robust than server-based SIP. The basic reason for using peer-to-peer is that it removes the need to use central servers and there is no dependence on a third party.
The reason that we are using SIP with P2P rather than another protocol because SIP is a widely used protocol in VoIP, and it can use the existing Internet telephony infrastructure. The advantage of using P2P SIP is that it requires no maintenance or configuration; it can self-organize and provide interoperability.

3.2 Structure Overlay/Distributed Hash Tables

Peer-to-Peer network uses distributed systems as it has no dependency on the central server and does not follow any known hierarchal organization. The Distributed Hash Table (DHT) helps to provide a lookup services. Since this network is at the application layer, it uses the underlying network to exchange the messages that is the reason they are called overlay networks too. This system provides content distribution to the application using the overlay it creates. It also provides an efficient lookup of the node that is responsible for a particular key. Distributed Hash Table (DHT) is distributed among the nodes in the network and all the nodes store small portion of the DHT. P2P networks are capable of handling large number of nodes that are entering and leaving the network at the same time.

3.3 Chord Algorithm

3.3.1 Chord

Chord algorithm is one of the most popular algorithms of DHT. It creates a ring-structure to place the peer in the network. In this structure, a peer that contains a hash value of zero is placed adjacent to the peer containing the highest possible hash value. In this algorithm, every resource is assigned a resourceID. If a resourceID is k then it will be stored in the first peer of the network with a peerID that would be greater to or equal to the value of k, it also ensures that every resourceID that is generated is associated to some peer in the network.

3.3.2 Finger Table

The Finger Table contains the list of the peers that the peer would use to send messages to. This table contains the information of the neighbors of the peer and the peers with similar ID's and very rarely, it would have the information of remote ID's.

3.3.3 Predecessor Peer

"Predecessor Peer" is a peer that is a predecessor to a particular peer in a given address space. A predecessor peer is not a peer that has peerID one less than the peerID of the particular peer in the address space, it means that there is no other peer between the predecessor peer and the peer.

3.3.4 Successor Peer

"Successor Peer" is a peer that is successor to the particular peer in a given address space. A successor peer is not a peer that has peerID one more than the peerID of the particular peer in the address space, it means that there is no other peer between the successor peer and the peer.

3.3.5 Hash Algorithms and Identifiers

All ID's generated in an overlay are calculated using the same algorithm. This implementation supports SHA-1 algorithm that generates a 160-bit hash value. The hash algorithm by the overlay is specified in the peerID header in DHT.

Peer-IDs and resource-IDs have to be in the same range of values.

Formally:

PeerID = token

After using SHA-1 algorithm:

PeerID = 40LHEX

All of the P2PIDs are generated using SHA-1; therefore, all P2PIDs are hash values; in the example below, the peer-ID is a04d371e24 (40LHEX hash value).

DHT Name Parameter:

In this protocol we need to set the dht-param token to "Chord 1.0". If any message does not have dht-param token equal to "Chord 1.0" then that message will be rejected by the peer and it will return a 488 message of not acceptable in response.

3.3.6 The DHT-Link Header

The peers transfer information about where the other peers are located in the network using DHT-Link Header. DHT-link header is also used for transferring the information of the successor, predecessor and finger table store in the peers to the other peers in the network. The depth and linktype values used by the peer are not dependent on the DHT algorithm used by the peer. We use depthtype-token and linktype-token to help the peer implement the Chord algorithm.

3.3.7 The Linktype and Depth Values

For the linktype, it has to be one the following character: P, S or F. P tells the receiving peer than the information they are receiving is for the predecessor of the sending peer. S tells the receiving peer that the information they are receiving is for the successor of the sender peer. F tells the receiving peer that the information they are receiving is the finger table of the sending peer.

The depthtype cannot be a non-negative integer and should be describing the predecessor, successor and finger table entry.

For example, “P0” indicated that it is the sender itself, whereas the “S9” indicates that it is the ninth successor.

3.4 Chord Overlay Algorithm

3.4.1 Finger Table, Successors, and Predecessors

All the peers have to keep a track of minimum one predecessor in the routing table. The predecessor peer can never point to itself, but it can be set to null if the peer is the only peer in the overlay. All peers have to keep a track of minimum one successor in the routing table. The successor can point to itself. Peers can have more than one predecessor and successor information in its routing table for reliability.

When we are using chord, it recommends us to keep the number of finger table entries equal to the size of the bits in the hash space. These entries should be pointing to the first peer which should be two^i away from the peer. The peers divide the circular overlay into segments. The peer then stores the entry in the

finger table for the first peer only if the peerID has to be equal or greater than the starting of this interval. This helps the peer to point to the nearby peer and less for the remote peers. It is flooded when the peer joins the overlay and is updated periodically.

If we are using SHA-1 hash algorithm then it is recommended to have 160 entries in the finger table, which could be 16 if the network is small and 32 if the network is bigger. This helps in improving the efficiency of the client.

3.5 Working of P2P Overlay Structure

1) Use of a DHT P2P structured based on the Chord algorithm and the SHA-1 hash algorithm
2) Each node
   a) Has a unique node-ID through hashing the IP address and port.
   b) Has the information about some other nodes in the network that helps it when it has to send messages across the overlay.
3) Every resource has a resource-ID
4) All messages are SIP messages of two kinds: maintain DHT and communication
5) Encoding P2P in SIP

3.6 Session Establishment

- Caller node creates INVITE/MESSAGE message and hashes caller name
- It sends the message to the node that is nearest to the peerID it wants to send message too.
  a) If that node is not responsible for the resource-ID, a 302 message is returned, then resend the INVITE/MESSAGE message to this node
  b) When the node storing that registration is located, it sends *either* a 302 to the actual address of the caller's node *or* a 404(called is not registered)
- Caller resends the SIP message to the caller node as usual

4. Initial Proposals of P2P SIP VoIP

a) Johnston and Sinnereich

Johnston and Sinnereich proposed that we could use P2P network in SIP server to store information and for location lookup information. They also proposed avoiding the use of protocol for DHT message exchanges. They separated the DHT and SIP functionality into two layers

b) Singh and Schulzrinne

Singh and Schulzrinne suggested Open DHT for SIP location services. They suggested to use an externally managed overlay for a client which used put/get messages to send messages to the nodes in that overlay with the help of remote procedures calls.

c) Singh and Schulzrinne

Singh and Schulzrinne proposed a hierarchical architecture in which multiple P2P networks represented a DNS domain. A global DHT was used for inter-domain routing of messages. To avoid the introduction of new SIP messages, the node ID and key ID are represented in SIP URI: for example, [hash (IP address)] @ IP address or [hash (IP address)] @ domain. This contains a lot of redundant information. User ID (the key helps us to look up for the node in the overlay) should be a valid e-mail address in the domain and e-mail based authentication is done. The SIP protocol is used under Chord. The SIP register method helps in storing key/data pairs. With the help of INVITE messages we can do user query.

d) Bryan, Lowekamp, and Jennings

Bryan, Lowekamp, and Jennings proposed a SOSIMPLE protocol as a P2P enhancement to the SIMPLE protocol. It is based on an Internet draft for P2P SIP registration and user location. The only differences are:

1) IP address and port are hashed and stored in nodeID
2) New SIP header (DHT-NodeID, DHT-Link): helps in transferring more than one routing table in a single SIP message.

3) SIP registration: helps in registrations and queries of nodeID/userID
4) Pure P2P without a super node
5) Iterative overlay routing
6) PKI recommended for user verification

5. Security Threats

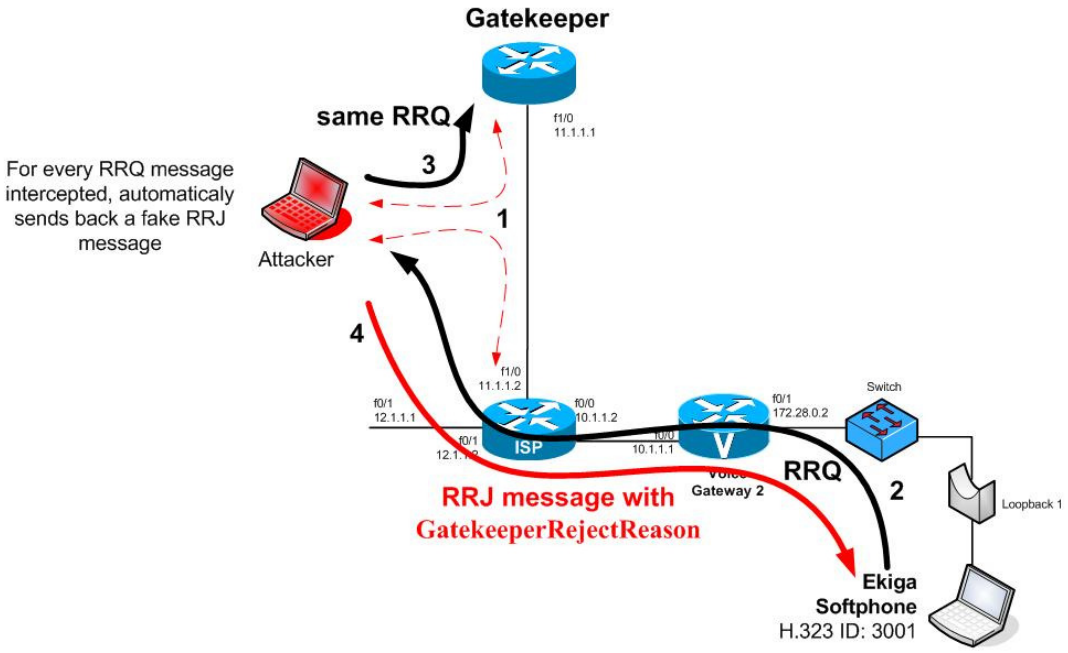
Security is one of the biggest concerns for VoIP in today's world. With the increasing popularity of Internet, a secure protocol is very important so that no one else can peek into your data that you are transmitting over the Internet. In this section we are going to discuss the security threats in VoIP architecture.

5.1 Security Threats in VoIP

1. Authentication spoofing and replay attack
   In such an attack, the attacker can sniff the packets been transfers and get the valuable information such as IP addresses. Then the endpoint can be attacked using a DoS attack or registration reject. The replay attack can be done by sniffing the RAS register request messages, through which MD5 can be retrieved and the message can be replayed.

2. DoS attack
   A DoS attack is one of the most common attacks on a network, and it is done by sniffing the messages and retrieving the endpoint IP address and then sending messages to that IP address again and again.

3. Registration attacks
   Registration attacks can be done with the help of a man-in-the-middle attack. The attacker stands between the ISP and the gatekeeper. When a soft phone sends a message via ISP to the gatekeeper to join the network, the attacker sends a fake register reject message to the soft phone. Then the attacker can use the same request message to join the network.



1. Man in the middle with ARP Poisoning – the H225regreject tool listens for incoming Registration requests
2. Terminal with H.323 ID 3001 sends a RRQ to Gatekeeper
3 H225reject extracts the IP addresses and ports form the RRQ, then forwards it to Gatekeeper
4. Creates a RRJ with the GatekeeperRejectReason of SecurityDenial and sends it to the 3001 terminal
RESULT: the H.323 terminal is unable to register to the Gatekeeper

Figure 12. Registration Attacks

5.2 Security Threats in SIP Architecture

1) Virus and software bugs

   Viruses and software bugs happen due to DoS attacks in which the attacker tries to continuously send messages to one IP address and then confuses the computer and gets unauthorized access to the computer. Viruses and bugs can be removed by adding an antivirus application or software patches.

2) Eavesdropping

   Eavesdropping means that someone sniffs the messages and decodes them. In other words, eavesdropping is an unauthorized interception and decoding of signaling messages.

3) Replay

   Replay is the retransmission of general messages for reprocessing. The attacker sniffs the messages, stores them, and then replays them. This can be done by a DoS or man-in-middle attack.

4) Message tampering/integrity

   Message tampering occurs when a message sent by a user is tampered with/changed by the attacker. This leads to loss of integrity. It can be avoided by encrypting messages sent using the mechanisms such as IPSec, TLS, or S/MIME.

5) Spoofing

   Spoofing is the impersonation of a legitimate user. It can be avoided by sending address authentication between the call participants.

6) Preventing access to network services

   Preventing access to network services can be done by flooding SIP proxy servers/registrars. This kind of attack is called a DoS (denial of service) attack. It can be stopped by configuring the systems.

6. SOSIMPLE: A Serverless, Standard-based P2P SIP Communication

6.1 Introduction

This protocol creates an ad-hoc network for connections. It creates a distributed hash table (DHT) overlay based on the Chord algorithm that is using SIP messages to look up the nodes to connect. This protocol is an improvement of the SIP/SIMPLE protocol. A P2P overlay is created using a DHT that is created through exchanging SIP messages

.

The primary contribution of this procedure is as follows:

- It creates a fully distributed network, open P2P system for VoIP and IM by extending existing standards
- Helps in identifying security requirement

6.2 Background and Challenges

6.2.1 SIP and SIMPLE

SIM and SIMPLE are text-based protocols that are derived from HTTP. SIP is a general protocol mostly used widely for VoIP, but it can also used to establish and control multimedia sessions. It also has session description protocol (SDP) embedded in it, which helps in specifying the media parameters.
SIMPLE is a set of SIP extensions for IM systems. SIMPLE and SIP are same but the IM systems is different as the messages that are to be passed are carried directly to the signaling path and no separate stream is used for when we are using IM systems.

6.2.2 Scenarios Requiring a New Approach

The scenarios that led to the motivation of the SOSIMPLE approach were as follows:

A) Security-conscious small organization
B) Limited or no Internet connectivity
C) Ad-hoc and ephemeral groups
D) Censorship or impeded access
E) Scalability

6.3 Requirements

A) No central server
   Sometimes we are not able to contact to the server because of either the security or scalability reason. With the help of P2P systems, we are able to cut down the option of having central servers.

B) No central naming authority

User have the freedom to select their own names and no central authority is required or involved in naming.

C) Simple system discovery

It is important to have a simple mechanism to discover a system, such as a broadcast mechanism.

D) Privacy

It is important that users can exchange messages without having other users interfere in the middle. It is important to have privacy for the users in the network.

E) Scalable number of users

It's important that the network grows and that no additional resources are required except those obtained by new users.

F) Compatibility and reuse

It is important that the system is compatible with the existing infrastructure. It's also good to use existing code as much as possible rather than build new code.

6.4 SOSIMPLE P2P SIP Architecture

6.4.1 Structure and Message

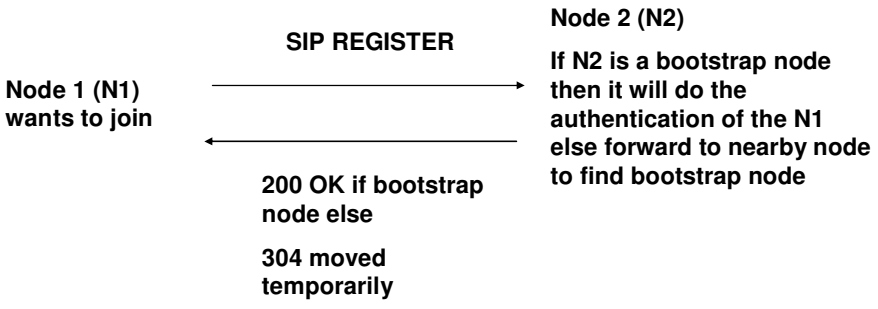
The nodes in the network are stored in the DHT based on the Chord algorithm and the NodeID is created using the SHA-1 hashing function. The protocol uses the Chord algorithm to maintain the overlay. The nodes store the information about the resources that it has and the information about the successor node. SOSIMPLE maintains a smaller number of finger tables entries in each node that contains information from the Chord. SIP can be improved by adding new headers that help to add new functionalities in the protocol.

With the help of SIP REGISTER we can transfer messages or information between the nodes in the overlay

1) To register the user's
2) Register the nodes in the network for DHT operations such as entering the overlay, leaving the overlay, or maintaining the overlay

6.4.2 Node-level Operation

A new node can join the network after exchanging many REGISTER messages. Once a node joins the network, the node is responsible to store information associated with the overlay that helps to map and calculate NodeID. The joining node is responsible for finding the node of that region; the node also has to join the network and transfer information about the region to the node currently storing the information.



. Figure 13. Messages between the Nodes to Join the Network

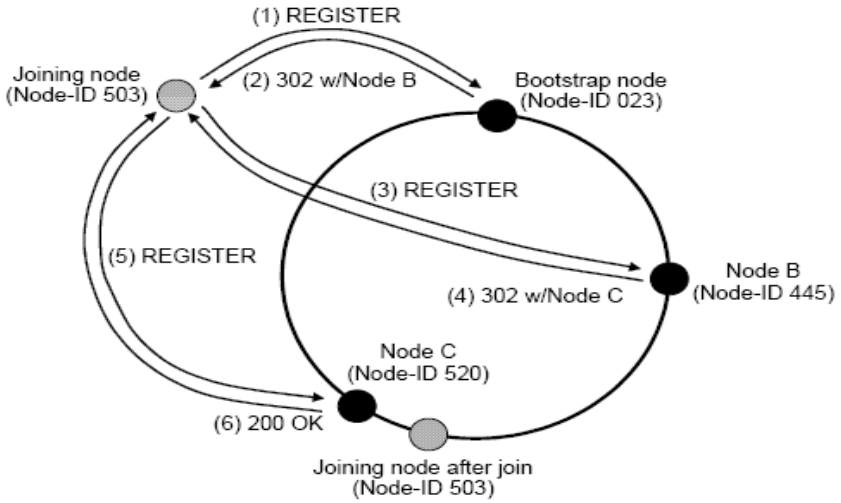


Figure 14. An example of Node-ID 503 Joining the Overlay

6.4.3 User Operation

Since the architecture has no central server and no central naming authority, so if we have to register or contact a user, it is important to find a user containing the information of the other user. When a node wishes to know the information of a particular user or wants to contact a particular user, it hashes its username to generate a resource ID. Since the node is in the overlay, the finger table has the information about the nodes and points to the correct node it wants to connect to. If a node wants to find/contact another node, it looks up into its finger table to find the peerID nearest to the resource ID. If that node is not the correct node, it will send back a 302 message which means Moved Temporarily, it also includes the node it thinks to be closet to the sender peer , it forwards the message to it. This goes on until the user does not get either a 200 message (the OK message) from the node or a 404 (node not found) message back. User operation also includes SIP SUBSCRIBE/NOTIFY and SIP PUBLISH (for user mobility use) messages, too.

7. My Approach

7.1 Improvements over the Existing Protocol

The SOSIMPLE algorithm has no security implementation. Privacy is one of the most important aspects nowadays in network security. It is important that the messages transmitted in the network are secure and that no outsider can read the message. P2P architecture needs a system that with trust between the users and the P2P system.

7.1.1 Security and User Authentication

With the help of RSA algorithm, I am generating public keys for each node, and they are stored in the overlay. With this procedure, there is no guarantee of authenticating a user joining the network, but when a user receives a message it checks the certificate and if it is a valid certificate then it accepts the message. The users can exchange messages among each other, but this certificate encrypts them. With the help of this mechanism, multiple users can have the same names but can be distinguished with the help of this certificate and previous conversations.

We are using RSA to generate public keys and digital signatures. This algorithm is more secure than others as it uses sufficiently longer keys. The following is the code segment to generate private and public keys:

```
KeyPairGenerator keyGen =
   KeyPairGenerator.getInstance("RSA);
KeyPair pair = keyGen.generateKeyPair();
PrivateKey priv = pair.getPrivate();
PublicKey pub = pair.getPublic()
```

The following is the code segment for encryption and decryption:

```
public String encrypt(byte[] text, PrivateKey priv)
   {
      byte[] cipherText = null;

      try
```

```
        {
            //
            // get an RSA cipher object and print the provider

            Cipher ecipher = Cipher.getInstance("RSA");


            ecipher.init(Cipher.ENCRYPT_MODE, priv);
            cipherText = ecipher.doFinal(text);
            return new sun.misc.BASE64Encoder().encode(cipherText);
        }
        catch (Exception e)
        {

            e.getMessage();
        }
        return null;
    }

public String decrypt(byte[] text, PublicKey pub) {

         byte[] dectyptedText = null;

        try
        {
            // decrypt the text using the public key
            Cipher dcipher = Cipher.getInstance("RSA");

            dcipher.init(Cipher.DECRYPT_MODE, pub);
            dectyptedText = dcipher.doFinal(text);
            return new sun.misc.BASE64Encoder().encode(dectyptedText);
        }
        catch (Exception e)
        {
            e.getMessage();
        }
        return null;
```

8. Implementation

1. In this screenshot, we create one node that starts a DHT and then, in the other prompt, we start another node that joins the overlay using the IP address of the first node.

```
C:\Users\richa\Desktop\project\project\bin>owdhtshell
DHT configuration:
  hostname:port:     richa-PC/192.168.1.101:3997
  transport type:    UDP
  routing algorithm: Chord
  routing style:     Iterative
  directory type:    VolatileMap
  working directory: .
A DHT started.
Ready.
```

Figure 15. Node 1 is Started

```
C:\Users\richa\Desktop\project\project\bin>owdhtshell 192.168.1.101
DHT configuration:
  hostname:port:     richa-PC/192.168.1.101:3998
  transport type:    UDP
  routing algorithm: Chord
  routing style:     Iterative
  directory type:    VolatileMap
  working directory: .
  initial contact:   192.168.1.101:3997
A DHT started.
Ready.
```

Figure 16.Node 2 is Started and Connected to Node 1 Using Node 1's IP Address

2. Typing Help at one prompt gives us all of the commands that we can run:

a) status [<verbose level>]

This command helps us to know our predecessors and successors to a user. With the help of this feature, a user can keep track of the neighbor nodes

b) init <host>[:<port>] [<port>]

With the help of this instruction, the joining node can join the overlay by contacting a specific node

c) get [-status] <key> [<key>…..]

This helps to get the values that the users have put using the put command using a specific key. The user can use multiple keys.

d) put [-status] <key> <value> [<value>…….]

This helps to place specific data with a specific key. The user can have multiple keys. The user can also put data in pairs separated by using “-” in the command.

e) remove: delete [-status] <secret> <key> [<value>…….]

This command helps to remove the data from a specific key. We do multiple deletes at a time separated by the “-” sign.

f) setttl <ttl>

It specifies the time to live <ttl>. The values stored by the put command can also be stored for ttl time.

g) localdata

This command helps to show all data stored in a node locally.

h) quit|exit

With the help of this command, we can stop the overlay.

i) halt|stop

This command the users.

j) clear routing table

This command clears the complete routing table.

k) clear dht

This command cleans the data in the nodes in the specified hash table.

l) suspend

This command stops a user temporarily.

m) resume

This command resumes a user who was suspended.

3. Once we run the status on any user node, it gives us the information about the user node neighbors. It is useful, as the user knows its neighbors.

```
C:\Users\richa\Desktop\project\project\bin>owdhtshell 192.168.1.101
DHT configuration:
  hostname:port:      richa-PC/192.168.1.101:3998
  transport type:     UDP
  routing algorithm:  Chord
  routing style:      Iterative
  directory type:     VolatileMap
  working directory:  .
  initial contact:    192.168.1.101:3997
A DHT started.
Ready.
status
ID and address: bec5af5cb5e343794f3ae40bcab06ebddae9f6db:richa-PC/192.168.1.101
3998
Routing table:
predecessor:
 e2b885b690b37829e7e496cc553ae0f94ac8ef8f:richa-PC/192.168.1.101:3997
successor list: [
 e2b885b690b37829e7e496cc553ae0f94ac8ef8f:richa-PC/192.168.1.101:3997
 bec5af5cb5e343794f3ae40bcab06ebddae9f6db:richa-PC/192.168.1.101:3998
]
finger table: [
 1: e2b885b690b37829e7e496cc553ae0f94ac8ef8f:richa-PC/192.168.1.101:3997
 159: bec5af5cb5e343794f3ae40bcab06ebddae9f6db:richa-PC/192.168.1.101:3998
]
Last keys & routes:
number of messages: 2 -> 2
key[0]: bec5af5cb5e343794f3ae40bcab06ebddae9f6db
route[0] (length: 1): [
 bec5af5cb5e343794f3ae40bcab06ebddae9f6db:richa-PC/192.168.1.101:3998 (0)
 e2b885b690b37829e7e496cc553ae0f94ac8ef8f:richa-PC/192.168.1.101:3997 (192)
]
root candidates[0]: [
 e2b885b690b37829e7e496cc553ae0f94ac8ef8f:richa-PC/192.168.1.101:3997
]
Ready.
```

Figure 17. Results after Typing Status in a User Node

1. Label 1 in the figure above shows that the IP address of the user is hashed and stored, as we are using IP address as our peer ID.
2. Label 2 has the routing table of the node in which the information of the predecessors and successors to the nodes is saved.
3. Label 3 contains information about the finger table, which has the list of the peers to which we send the messages.

4. Lable 4 has the number of messages sent and received between the nodes; for example, 2->2 means that 2 messages were sent and receieved by this node. Key has the key used to route the messages.
5. Label 5 shows the route that the message has taken and the nodes it has passed through to send a message to the correct node. In this example, the length is 1, as the destination node is the neighbor to this node.
6. Label 6 shows the root node of this network.

This information is usefult for testing, as it will help the developer to know whether the route followed is efficient.

4. We can put information in the overlay using the put command. We can get information from the overlay by writing the get command. Once we write the command with the word, we need the value thatit provides.

```
put foo bar
Ready.
get bar
key:   62cdb7020ff920e5aa642c3d4066950dd1f01f4d
value:
Ready.
get foo
key:   0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33
value: bar 10748 0x4c66c898545a69ad3f97654627836c31727b58f0
Ready.
```

Figure 18. Example of Put and Get Key and Value

In the example above, we are putting in the value “bar” in the distributed hash table with the key “foo.” In the next one, we are getting the value of bar and foo. Since bar is data itself, it has no value, but since foo is a key to store bar, it has its key information and its value “bar,” as in the figure above.

6. We use an emulator to control many DHT shells on one computer. The scenario used is as follows:

```
C:\Users\richa\Desktop\project\project\bin>owemu http://bit.ly/bl1U54
DHT configuration:
  hostname:port:      emu0:3997
  transport type:     UDP
  routing algorithm:  Chord
  routing style:      Iterative
  directory type:     VolatileMap
  working directory:  .
A DHT started.
A shell server is waiting on the port tcp/10000
DHT configuration:
  hostname:port:      emu1:3997
  transport type:     UDP
  routing algorithm:  Chord
  routing style:      Iterative
  directory type:     VolatileMap
  working directory:  .
A DHT started.
DHT configuration:
  hostname:port:      emu2:3997
  transport type:     UDP
  routing algorithm:  Chord
  routing style:      Iterative
  directory type:     VolatileMap
  working directory:  .
A DHT started.
DHT configuration:
  hostname:port:      emu3:3997
  transport type:     UDP
  routing algorithm:  Chord
  routing style:      Iterative
  directory type:     VolatileMap
  working directory:  .
A DHT started.
```

```
C:\Users\richa\Desktop\project\project\bin>owdhtshell
DHT configuration:
  hostname:port:      richa-PC/192.168.1.101:3997
  transport type:     UDP
  routing algorithm:  Chord
  routing style:      Iterative
  directory type:     VolatileMap
  working directory:  .
A DHT started.
Ready.
get a_value
key:   d3b1df29584275b892bd8a748c31d121cbfc2c56
value:
Ready.
get a_key
key:   5d8b23c071cb95840397933b5969c69413fe87cf
value:
Ready.
```

Figure 19. The Emulator Running and Controlling 4 Nodes on a Single Computer

In the figure above, we have the emulator working and invoking 4 user nodes. We can also start the nodes and retrieve the information a_value that we put in the user node 0 (emu 0) through the emulator scenario.

7. Similar to the scenario above, we have a scenario for invoking 20 nodes on a single computer. The screenshots below are the visual effect of the nodes communicating with each other.

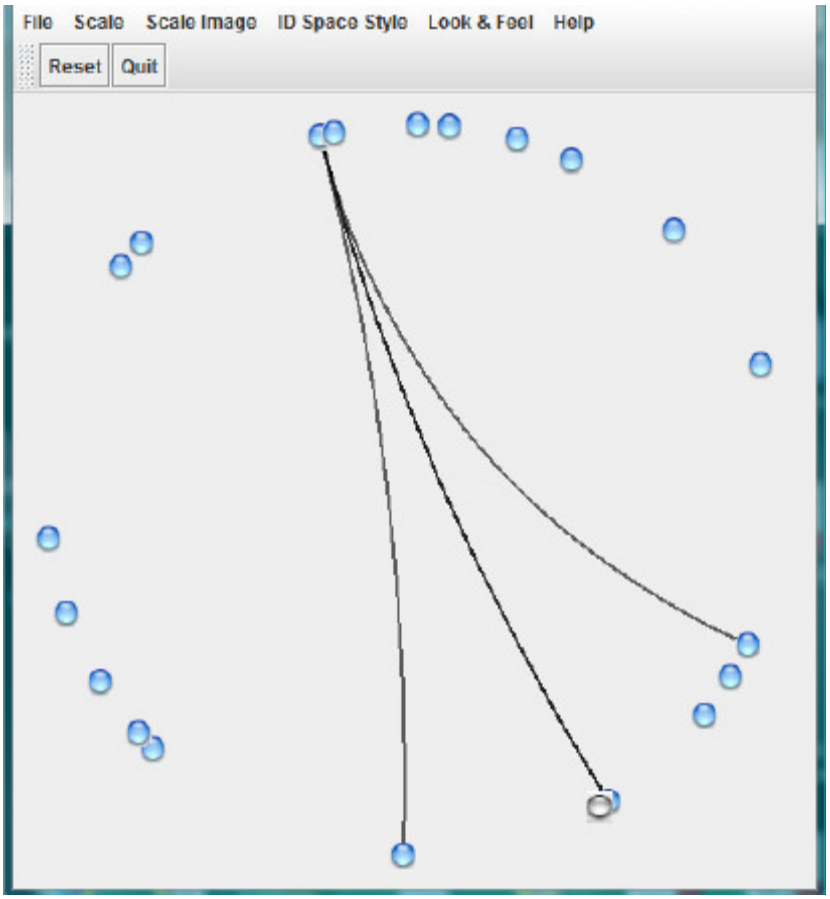


Figure 20. Visual effect of 20 nodes communicating in circle format

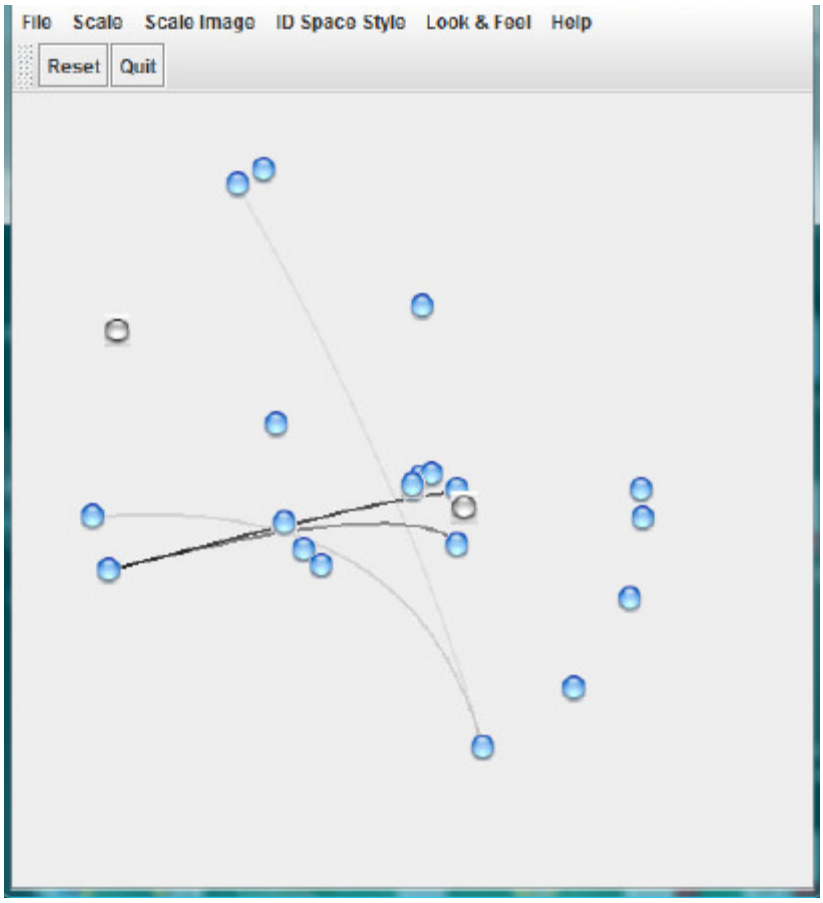


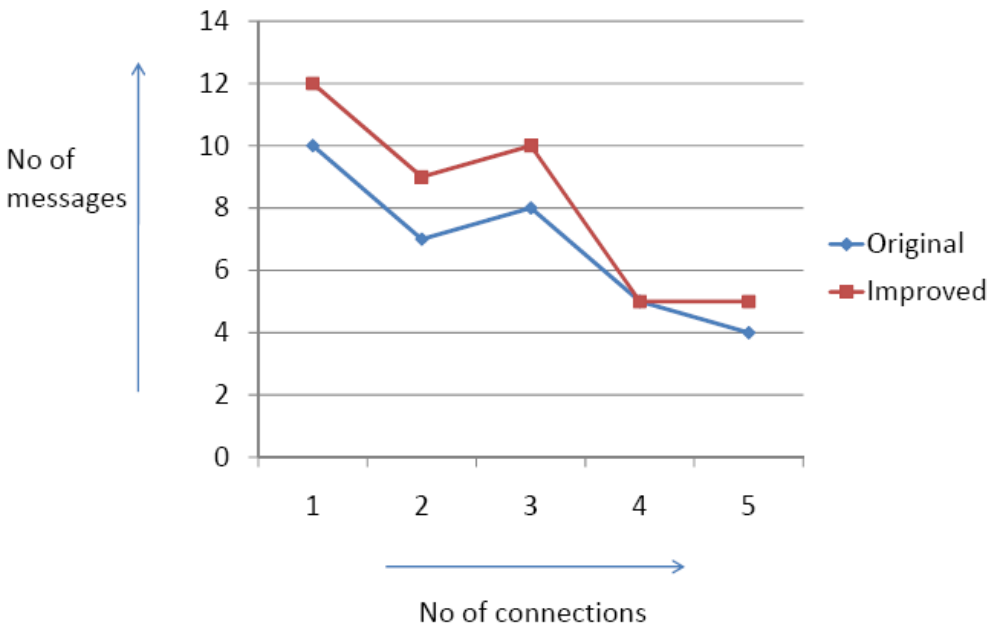
Figure 21. Visual effect of 20 nodes communicating in vortex format

In the screenshots above we have 20 nodes creating an overlay network on a single computer with the help of the Emulator. The dots in the screenshots are different nodes joining the network and the lines are showing the communication between the nodes via Message services. There are many formats in the visual effect like Circle, Vortex, Straight line, Waving line and Grid.

9. Performance Graphs

Scenario: I created a test case for the simulator in which five nodes are generated in the network and node1 tries to talk to the other four nodes in the network.

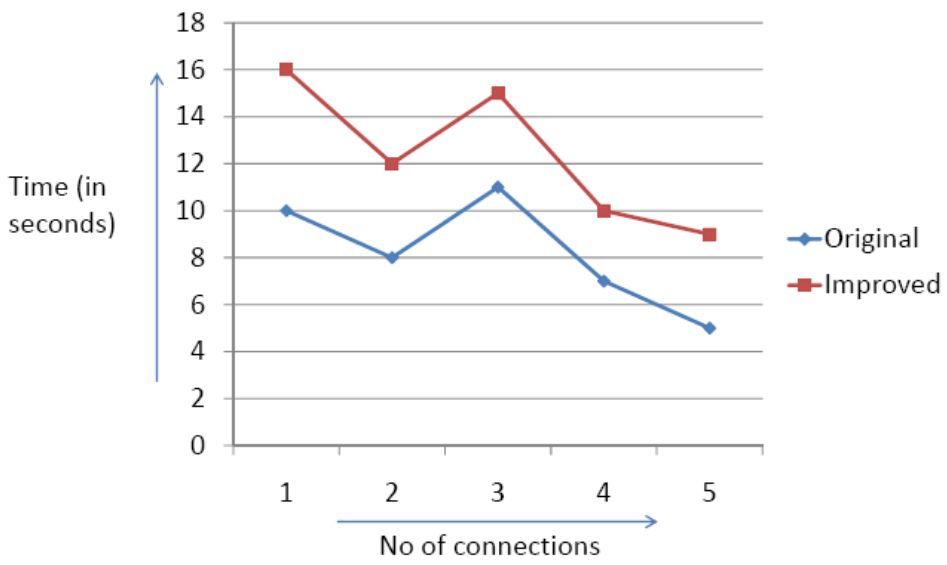
Graph 1: I have plotted a graph between the original SOSIMPLE protocol and Improved SOSIMPLE protocol (with user authentication). The X-axis represents the number of nodes that Node 1 connects to, while the Y-axis represents the number of messages passed between the nodes to build those connections.



Graph 1: Comparison with Original SOSIMPLE protocol and Improved SOSIMPLE protocol

Graph 2: I have plotted a graph with X-axis representing the number of

Connections made and the Y-axis represents the Time taken(in seconds) to make the connection.



Graph 2: Comparison between Original SOSIMPLE and Improved SOSIMPLE protocol.

Conclusion: With the security in the protocol, it is not significantly deteriorating the performance of the protocol.

10. Kademlia Algorithm in SOSIMPLE Protocol

10.1 Introduction

Kademlia is an existing P2P algorithm, like Chord, that I have implemented to make a comparison between Kademlia and Chord in order to determine which one is better.

Kademlia is a distributed hash table algorithm designed for using decentralized peer-to-peer network. It specifies the structure of the network and exchanges the information between the nodes with the help of the node lookups. The peer in the network is identified through the numbers or peer ID. In Kademlia, the peer ID is not just for identification of the nodes but also to locate the values. In other

words, with the help of the node ID, we can do a direct map for file hashes, and the nodes have stored information on where to look up files or resources. Kademlia uses a key to store the values; hence, for each value stored, there is an associated key. The nodes in each step will find the closest node to the key until they don't get a response or value back from the other nodes.
The advantages of using these decentralized structures are that we can increase resistance to many common attacks, especially denial of service attacks (DoS attacks). Even if we flood the network, it will have a limited effect on the network, as the network will be build around the holes for the users.

10.2 System Detail

This algorithm uses the "distance" calculation to determine how far two nodes are from each other. The Exclusive OR (XOR) of the node ID is the distance between them, since the key and the node ID are of the same format and length, it helps in calculating the distance in the same way, the node ID for each node is a random number chosen with the consideration that it has to be unique for each node.
Exclusive OR (XOR) is used as it has some common properties as that of geometric distance calculation.

10.2.1 Routing Tables

As with the Chord algorithm, we have a finger table to store the information about each bit; similarly, we have a list in the Kademlia algorithm that stores this information. The information stored in the list contains the data that can help us to locate other nodes. Generally, the entries in the list are the IP address, port, and node ID of the other nodes. As the network grows, every node that is encountered is recorded in the list. All of the nodes encountered are considered for inclusion in each node list.
In Kademlia, the lists of nodes are sometimes referred to as k-buckets, where k is a number that is system-wide, such as 20. Each k-bucket has lists with k-

entries, which means that each node would have lists of up to k nodes for a particular distance from itself. If the k-bucket is full and the node comes across a new node in the network, then it pings the node that has not been encountered for a long time to determine whether the node is still alive or dead. If it does not get a response, then the node assumes that the node is dead and deletes its entry from the list and adds the information about the new node encountered. If the node is alive, then the information about the new node is stored in the secondary list, which is the replacement cache.

10.2.2 Protocol Messages

- PING
- STORE
- FIND_NODE
- FING_VALUE

10.2.3 Locating Nodes

In Kademlia, we can do asynchronous lookups. When a node starts looking for a node by requesting a FIND_NODE, it starts by querying in its own node list (the k-bucket) and finds the closed one to the desired value. The requestor sends the message to the nearest node, which looks in its k-bucket and sends the nearest value to the requestor node. The requestor node will store the information sent in its results list. This iteration keeps going until the node does not send back a result that is much closer than the previous result.

10.2.4 Locating Resources

Information is stored using a key. Hash is generally used for mapping. The node that has to store the information will use the STORE message. The key used to store the value is found in the same way as locating nodes. The only difference is that it returns the exact key and the values

11. Comparison between the Chord and Kademlia Algorithms

Scenario 1:

I have created a test case in which a network is created that has 5 nodes and node one interacts with all the other nodes in the network. I made this test case work for both the algorithms. The reason I run this test case is to see how many messages does Chord and Kademlia take to establish a connection. This test case in the end shows the message count of how many messages are interchanged between the nodes to establish the connection.

Result:

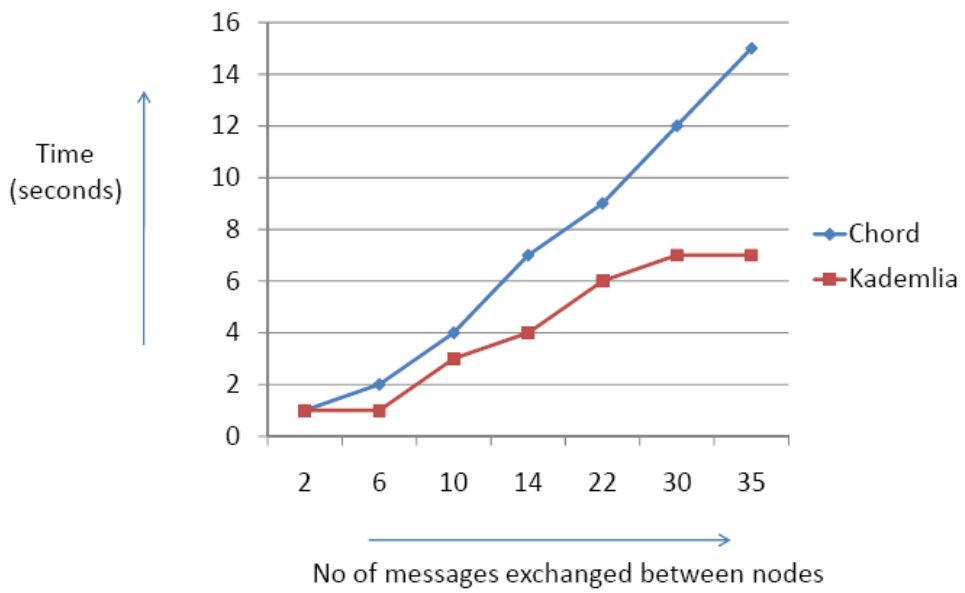
In this test case, I found that Chord algorithm takes 48 messages to establish 5 connections and Kademlia takes 19 messages to establish the 5 connections. The reason that Kademlia takes fewer messages and establishes a connection faster is that it supports asynchronous / parallel lookup for the nodes. Kademlia can do asynchronous lookup as it creates a XOR metric to calculate the distance between the two nodes. It also helps Kademlia routing table to extend more than a single bit. This helps Kademlia to do a faster lookup and establish a connection as compare to Chord.

Scenario 2:

I have a simulator in which networks of 20 nodes are created using Chord and Kademlia. Once the network is created the node 1 starts interacting with other nodes by sending PUT messages. I run my test case for Chord algorithm and then for Kademlia algorithm to build a comparison graph.

Graph 3:

In the graph below, I have plotted Time Taken (in seconds) to send particular number of messages between nodes with the help of Chord algorithm and Kademlia algorithm. In this test case, a set of messages are passed between the nodes to interact with each other in the network. The graph below shows the time taken by the different algorithms to pass those set of messages between the nodes in the network. The messages are sent between the nodes in this test case not just for establishing connection between the nodes but also to transfer more information between the nodes. The X-axis represents the number of messages sent and the Y-axis represents the time taken (in seconds) to send the message.



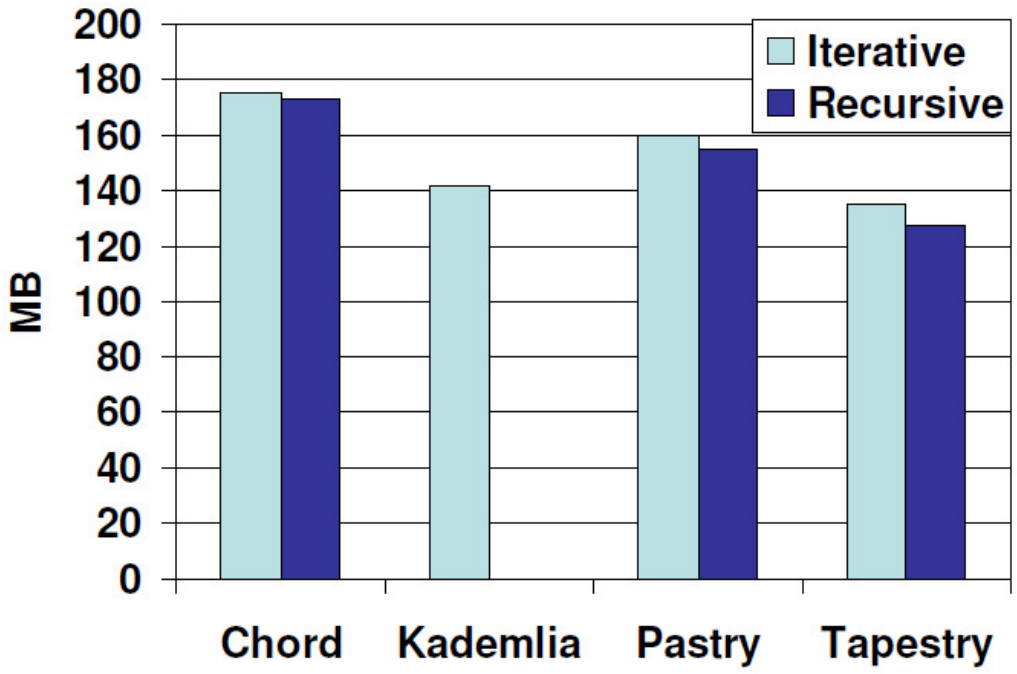
Graph 3. Comparison between Chord and Kademlia

Conclusion: In the above graph it shows that the time taken to send a particular set of messages between the nodes is less in Kademlia as compare to chord. The reason why Kademlia takes less time than Chord is because it can do asynchronous or parallel lookup which helps Kademlia to do faster message transfer.

12. Conclusion

With the help of the SOSIMPLE architecture, we were able to decentralize the system and remove the dependency on central proxy servers. It not only reuses the existing SIP client but can also establish the interface for the SIP systems. The security that we have added always authenticates without having a fully secure P2P system.

13. Appendix



Graph 4. Memory Consumed when 1000 Nodes Run on a Single Computer

Conclusion: A Chord algorithm with 1000 nodes running uses 177 MB of memory, which requires the maximum memory with respect to the other algorithms. This yields an estimate that with 1GB of free memory, we can run 5000 nodes on a single computer.

14. Future Work

To avoid over-flooding of messages, they have to resolve a crypto graphical puzzle: For a key k of data, determine an appropriate b so that the first c bits of H(k xor b) are equal to the first c bits for a particular node ID.

15. References

[1] RSA Algorithm, http://en.wikipedia.org/wiki/RSA

[2] SOSIMPLE P2P VoIP, A Serverless, Standard-based P2P SIP Communication, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.4225&rep=rep1&type=pdf

[3] Session Initiation Protocol, http://en.wikipedia.org/wiki/Session_Initiation_Protocol

[4] P2P SIP Information, www.p2psip.org

[5] Security Challenges for Peer-to-Peer SIP, http://www.jcbroadband.com/Library/jcbvoip5.pdf

[6] P2PNS: A Secure Distributed Name Service for P2PSIP, http://doc.tm.uka.de/P2PNS_2008.pdf

[7] SIP Security Mechanism: A State-of-the-Art-Review http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.7241&rep=rep1&type=pdf

[8] SIP Security: Status Quo and Future Issues, http://events.ccc.de/congress/2006/Fahrplan/attachments/1116-22c3_SIPsecurity_JanSeedorf.pdf

[9] SIP Security, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.5711&rep=rep1&type=pdf

[10] VoIP Howto, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.5711&rep=rep1&type=pdf