Aalto University
School of Science and Technology
Faculty of Electronics, Communications and Automation
Master's Programme in Communications Engineering

Cheng Luo

Design and Implement Voice Application over Ad Hoc Networks Using UPnP

ESPOO, May 2010.

Thesis submitted in partial fulfilment of the requirements of the degree of Master of Science in Technology.

Supervisor: Prof. Jörg Ott

Instructor: Dr. Jose Costa-Requena

Department of Communications and Networking
Espoo 2010

Acknowledgements

This Master's thesis has been written during my study in Networking Laboratory at Helsinki University of Technology under the supervision of Professor Jörg Ott.

I would like to express my gratitude to Professor Jörg Ott for his guidance and support. Without him, I would not get this far. Dr. Jose Costa-Requena has reviewed and provided a lot of valuable comments on my thesis. I want to give my special thanks to him for many weekends that he spent on reading my thesis.

Finally, I would like to give my deepest gratitude to my wife Tong, my daughter Niuniu and specially to my parents for their endless support throughout my studies.

Otaniemi, May 2010

Cheng Luo

AALTO UNIVERSITY

ABSTRACT OF
MASTER'S THESIS

Faculty of Electronics, Communications and Automation
- Master's Programme in Communications Engineering-

Author: Cheng Luo **Title of thesis:** Design and Implement Voice Application over Ad Hoc Networks Using UPnP		
Date:	May 21st 2010	**Pages:** 11 + 65
Professorship:	Networking Laboratory	**Code:** S-38
Supervisor: **Instructor:**	Prof. Jörg Ott Dr. Jose M. Costa Requena	

The traditional voice service was based on circuit-switched network architecture. It has been deployed on the packet-switched based network since Session Initial Protocol (SIP) became the de facto standard for Voice over IP (VoIP) in 1999. Since then voice service has become simple and flexible.

Another important technology driving voice service more popularity is mobile Ad Hoc networks (MANET). Most of internet applications or services such as VoIP or instant messaging (IM) are designed with client/server architecture. This design requires the initiator of communication sessions to know the address of counterpart prior to building a connection. With Ad Hoc networks, there is no such requirement at all.

In this thesis, we will design an Ad Hoc architecture using Universal Plug and Play (UPnP) protocol, and implement a simple voice application over such network. By definition infrastructure is unavailable in Ad Hoc network, we will also present the solution of how to enable SIP-based session setup on Ad Hoc network.

This thesis consists of two parts. The first part is the theoretical part. In this part, we will review the technologies related to our design and implementation. The second part is the system implementation and validation part. We will test our implementation with the Nokia Internet Tablet N810/N800s for various scenarios.

Keywords: Ad Hoc, MANET, SIP, UPnP, peer-to-peer,
Language: English

List of Abbreviations

AMRoute	Ad hoc Multicast Routing protocol
AMRIS	Ad hoc Multicast Routing protocol utilizing Increasing id-numberS
API	Application Programming Interface
ARP	Address Resolution Protocol
CAMP	Core Assisted Mesh Protocol
DDM	Differential Destination Multicast
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSDV	Destination-Sequenced Distance Vector
FGMP-RA	Forwarding Group Multicast Protocol-Receiver Advertising)
FGMP-SA	Forwarding Group Multicast Protocol-Sender Advertising)
FXPP	Flexible XML Processing Profile
GENA	General Event Notification Architecture
HTML	HyperText Markup Language
HTTPMU	HTTP Multicast over UDP
HTTPU	HTTP Unicast over UDP
LAM	Lightweight Adaptive Multicast
LGT-based	Location Guided Tree
MAODV	Multicast Ad hoc On-demand Distance Vector protocol
MCEDAR	Multicast Core-Extraction Distributed Ad hoc Routing

ODMRP	On-Demand Multicast Routing Protocol
RTT	Round Trip Time
SOAP	Simple Object Access Protocol
SSDP	Simple Service Discovery Protocol
SIP	Session Initiation Protocol
UPC	Universal Product Code
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
USN	Unique Service Name
UUID	Universally Unique Identifier
Wi-Fi	Wireless Fidelity
XML	Extensible Markup Language

Contents

List of Abbreviations **iv**

1 Introduction **1**

1.1 Problems 3

1.2 Objective and Scope 4

1.3 Structure 5

2 SIP and Ad Hoc Network **6**

2.1 Overview of SIP 6

2.1.1 SIP Entities 7

2.1.2 SIP Message 9

2.1.3 SIP Transactions 10

2.2 Overview of MANET 11

2.2.1 History of Ad Hoc Networks 12

2.2.2 Ad Hoc Networks Routing 13

2.2.3 Multicasting in MANET 16

2.3 SIP over MANET 17

2.4 Summary 19

3 Peer Discovery and Service Discovery **20**

3.1 Peer Discovery 20

3.2 Service Discovery 21

3.2.1 Service Location Protocol 22

3.2.2 Jini 22

3.2.3 Bluetooth Service Discovery 23

3.3 Universal Plug and Play protocol 23

3.3.1 Overview of UPnP 23

3.3.2 Addressing 25

3.3.3 Discovery 26

3.4 Summary 29

4 Technology Background 30

4.1 Sofia SIP 30

4.1.1 Common Runtime Libraries 31

4.1.2 Signaling Module 31

4.2 Cybergarage UPnP Implementation 32

4.3 GTK+ 2.0 and Hildon-2.0 32

4.4 Development on Maemo 33

4.4.1 D-BUS 33

4.4.2 LibOSSO 34

5 System Architecture 35

5.1 System Overview 35

5.1.1 VoIP Application 36

5.1.2 SIP UA 36

5.1.3 UPnP Stack 37

5.2 Test Scenarios 39

5.2.1 Scenario 1: An Ad Hoc node to another Ad Hoc node . . 40

5.2.2 Scenario 2: An Ad Hoc node to a fixed node 42

5.2.3 Scenario 3: A fixed node to an Ad Hoc node 44

5.3 System Implementation 45

5.3.1 VoIP Components 46

5.3.2 Sofia SIP Components 46

5.3.3 UPnP Components 48

5.4 Summary 50

6 Testing and Analysis 51

6.1 Demonstration Setup 51

6.2 Results Analysis 53

6.2.1 Scenario 1 53

6.2.2 Scenario 2 54

6.2.3 Scenario 3 54

7 Conclusion and Future work 57

A Key APIs 63

List of Figures

1.1 World Internet Users 2

1.2 Growth of VoIP usage 3

2.1 SIP server in the role of INVITE and REGISTER 8

2.2 Example of SDP session description 11

2.3 Simple SIP transactions 12

2.4 Categorization of Ad Hoc routing protocols 13

2.5 AODV route discovery 14

2.6 OLSR multipoint relays and regular nodes 15

3.1 Hop-by-hop Peer discovery in IP layer and End-to-end Service discovery in Application layer 21

3.2 UPnP protocol stack 24

3.3 UPnP architecture 24

3.4 Example SSDP header fields 26

3.5 Example of Unicast M-SEARCH message 28

3.6 Example of Multicast M-SEARCH message 28

3.7 Example of 200 OK Response message 29

4.1 Example of D-BUS Service File 34

5.1 System Architecture 35

5.2 Voice Application Main Window 36

5.3 Scenario 1: Ad Hoc node to Ad Hoc node 40

5.4 System Flow Chart: Ad Hoc node to Ad Hoc node 41

5.5 Example of 200 OK message 42
5.6 Scenario 2: Ad Hoc nodes to fixed nodes 42
5.7 System Flow Chart: Ad Hoc node to fixed node 43
5.8 System Flow Chart: Fixed node to Ad Hoc node 45
5.9 Example of forwarding REGISTER request 46
5.10 Application Module Structure Overview 47
5.11 SIP Structure Overview 48
5.12 SIP parser example: BYE message 49
5.13 UPnP Module Structure Overview 50

6.1 Voice Application UPnP Search Window 52
6.2 UPnP Search Results Window 52
6.3 SSDP vs. SIP packets 53
6.4 STUN binding message 55
6.5 SIP REGISTER message 56

List of Tables

2.1 SIP status code and response class 10

2.2 SIP status code and response class 10

2.3 SDP parameters and meanings 11

2.4 Comparison of Ad Hoc multicast protocols 16

6.1 Test Environment 51

6.2 Scenario 1 53

6.3 Call Setup Delay in Scenario 1 54

6.4 Scenario 2 54

6.5 Call Setup Delay in Scenario 2 54

6.6 Scenario 3 55

6.7 Call Setup Delay in Scenario 3 55

Chapter 1

Introduction

The purpose of this chapter is to give a brief introduction of voice over Internet Protocol (VoIP) technology. We will focus on the VoIP service that is deployed over mobile Ad Hoc networks (MANET) and peer-to-peer (P2P) networks. In this chapter, we will firstly give the background of mobile VoIP technologies. Then we will present the problems of implementing voice service on mobile networks. At the end of this chapter, we outline the structure of this thesis.

VoIP uses the Internet Protocol (IP) for voice transmissions. The analog voice signal is converted into digital signal, compressed, and segmented into series of data packages. The idea of VoIP has been discussed since early 1970s. And it was commercialized in mid-1990s. It has several advantages compared with the traditional telephony system. Calls using VoIP are much cheaper than using traditional telephony system. In most cases it is free when the caller and the callee are both connected to the Internet. Moreover, VoIP allows use to have video conversation or IM services. Along with the increasing of the Internet accessibility in public and private, VoIP is overtaking the traditional telephony service and becoming the first option for the long distance communications. According to the report [1] made by TeleGeography, the increasing rate of the VoIP usage is faster than the usage of using the traditional circuit-switched technology such as time-division multiplexing (TDM) shown in Figure 1.2.

With the growth of smartphone and netbook usage, the mobile Internet usage is increasing rapidly. According to the latest report [2] made by Morgan Stanley Research on the Internet trends, the mobile internet users will be bigger than the desktop internet in 5 years shown in Figure 1.1. VoIP becomes even more popular on wireless networks than wired networks.

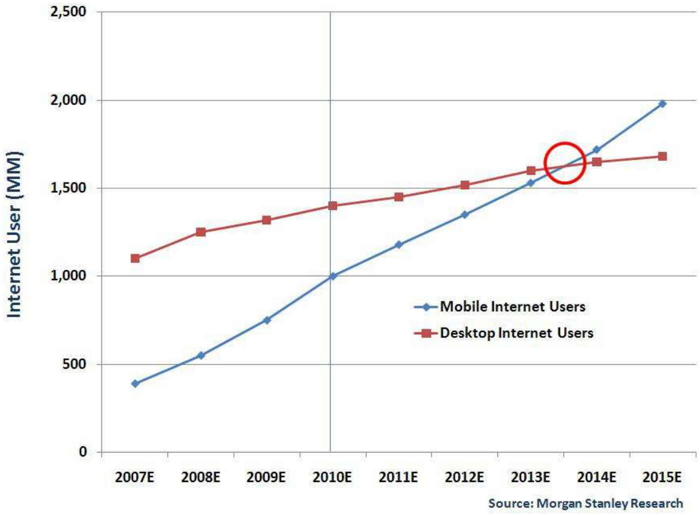


Figure 1.1: World Internet Users

Wireless networks differ significantly from wired neworks. The addressable unit also known as station (STA) is mobile in the wireless network. The IEEE standard 802.11 defines two modes: infrastructure mode and Ad Hoc mode. In the infrastructure mode, the wireless network consists of at least one access point (AP) connected to the wired network infrastructure and a set of STAs. This configuration is called a Basic Service Set (BSS). An Extended Service Set (ESS) is a set of two or more BSSs. The Ad Hoc mode (also called peer-to-peer mode or an Independent Basic Service Set, or IBSS) is simply a set of 802.11 wireless STAs that communicate directly with one another without using an access point or any connection to a wired network. This mode is useful for quickly and easily setting up a wireless network anywhere that a wireless infrastructure does not exist or is not required for services. MANET provides an alternative network for the VoIP. And it is free to use.

From the figures, we can conclude that the pattern of the voice communication from user perspective trends to the free and flexible usage. The biggest reason is that VoIP provides cheaper and affordable price compared with public switched telephone networks (PSTN) as we mentioned early. The P2P and MANET based VoIP telephony is threatening traditional telephony providers and mobile network operators (MNO) because it requires less or no infrastructure from internet service providers (ISPs) or MNOs. If VoIP service provider Skype were a carrier, it would be the largest carrier in the world.

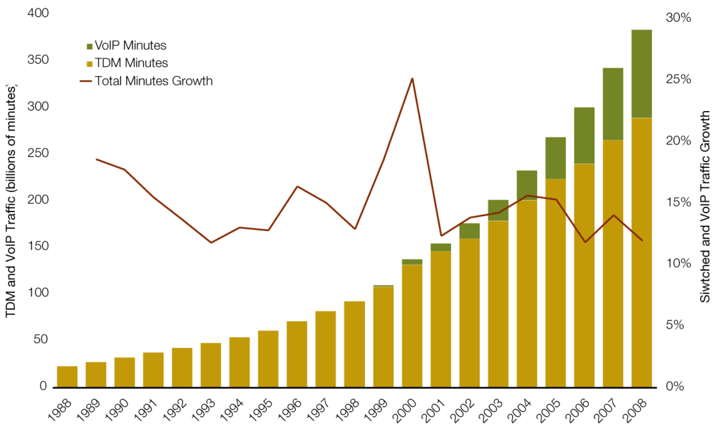


Figure 1.2: Growth of VoIP usage

1.1 Problems

However, VoIP is born with limitations because it is tightly coupled with the Internet access. The Internet is not designed for voice communications. Unlike circuit switched networks, IP networks have unpredictable packet lost and delay. When the VoIP service runs over wireless networks, the situation is even worse. There are several specific issues that implementation must deal with:

1) Efficient peer discovery protocols

The Ad Hoc network which we are presenting here works as a purely distributed network. So to implement an application on a peer-to-peer network, an efficient peer discovery mechanism is always a key issue. The peer discovery algorithm for peer-to-peer network is an active researching area nowadays. There are several widely used protocols, such as Chord [3], Distributed Hash Table (DHT) and Pastry [3]. There are no clear findings that determine which one has better stability, scalability and latency.

2) Using SIP in a de-centralizied network

VoIP normally use the Session Initiation Protocol (SIP) as its signalling protocol. But SIP is designed to rely on centralized components. It is difficult to deploy such components on a de-centralized Ad hoc network. To set up a normal SIP session on the Ad hoc network is a challenge.

3) Traverse between private networks and public networks

When using VoIP service on a private network, the network address translator (NAT) blocks the requests from the public network because it can not map the IP address at the application layer (e.g. SIP via header or contact header).

4) Security

There are also security problems which we have not dealt with in this implementation. But we should bear in mind that security problems are a highly discussed area of UPnP technology and also Ad Hoc network. As the nature of Ad Hoc network is to broadcast the notifying message, there are many drawbacks for such networks.

1.2 Objective and Scope

The objective of this thesis is to design and implement a VoWLAN service on an Ad Hoc network and study the connectivity between Ad Hoc and fixed network. The voice client (SIP client) registers to the server (SIP registrar) when the interconnection is detected. After the implementation, we will do the following analysis and evaluation:

Measure the traffic load of UPnP module and SIP module to the whole Ad Hoc network

- Scenario of communications between nodes in the Ad Hoc network
- Scenario of communications between node in the Ad Hoc network and nodes in the fixed network
- Scenario of frequently join and leave nodes in the Ad Hoc network

Evaluate the scalability of such service based on the Ad Hoc network.

Quality of service (QoS) of VoIP application

- Time of voice connection setting up
- Testing environment of office building (end-to-end delay)

After the analysis and measurement, we wish to find a trade-off point for our VoWLAN using UPnP. We aim to identity the proper announcement and notify UPnP packets frequency and the timeout of device notify packets.

1.3 Structure

This thesis is divided into 7 chapters. In Chapter 2, the technologies of Ad Hoc network and SIP are introduced. The Service discovery and UPnP protocol are unfolded in Chapter 3. Chapter 4 and 5 are devoted to the system design and implementation. Chapter 6 reports the measurement results and the performance analysis of the VoIP application. In the last chapter, we conclude the findings from our tests and give some suggestions for the future work.

Chapter 2

SIP and Ad Hoc Network

In Chapter 1 we have introduced the background of mobile VoIP communications and its drawbacks. In this chapter, we will unfold the SIP protocol and the mobile Ad Hoc network (MANET) routing protocols. As we will design an alternative way of using SIP in MANET, this chapter will go through the theoretical part of SIP protocol and MANET which are the two interesting areas in the multimedia P2P applications. Apparently we are not the first one to design a decentralized P2P SIP solutions.

2.1 Overview of SIP

SIP basically is a signalling protocol for creating, modifying and terminating multimedia sessions between two or more participants over Internet. It provides similar functions to ISDN User Part (ISUP) for Public Switch Telephony Network (PSTN). It makes telephone calls possible on Internet comparable to the legacy PSTN. According to RFC 3261[19], “SIP supports five facets of establishing and terminating multimedia communications:

- **User location**: determination of the end system to be used for communication;
- **User availability**: determination of the willingness of the called party to engage in communication;
- **User capability**: determination of the media and media parameters to be used;

- **Session setup**: ‘ringing’, establishment of session parameters at both called and calling party;
- **Session management**: including transfer and termination of sessions, modifying session parameters, and invoking services.” [19]

In this section, we will review some of the related SIP functionalities, such as user location, user availability and session setup, which are used for our voice application.

A user agent represents an end system. SIP defines two types of user agent in each single SIP transaction: User Agent Client (UAC) which generates the requests, and User Agent Server (UAS) which responds to them. Each SIP transaction is initalized by a UAC sending a request to a UAS, and is alway ended by the UAS response to the request. But if the UAS decides to redirect the request for the next transaction, it becomes the UAC. So the role of UAC and UAS, as well as proxy and redirect servers, are defined on a transaction-by-transaction basis.

2.1.1 SIP Entities

As mentioned above, user agent is the basic entry of SIP. Besides user agent, SIP also defines registrar, proxy server and redirect server. In SIP usage, some user agents are computers, some user agents could be laptops, PDAs or SIP phones. In some situations SIP transaction could be very complicated. In Figure 2.1, user agent (Alice) could carry multiple SIP devices, and use them at different places in different time. It is impossible for the caller (Bob) to know before hand which SIP device Alice is using now. So in this case, we need the proxy server (or redirect server) to get Alice’s current SIP address-of-record (AOR) from location service. A location server can always get the AOR from registrar when an endpoint registers its AOR to it.

Proxy Server

For locating prospective session participants, and for making the routing decsions, SIP enables an entry within SIP infrastructure called proxy servers shown in Figure 2.1. Proxy server decides to which user agents it can send registrations, invitations to sessions, and other requests in behalf of the requestor. In most of case, proxy server is associated with SIP transaction between two

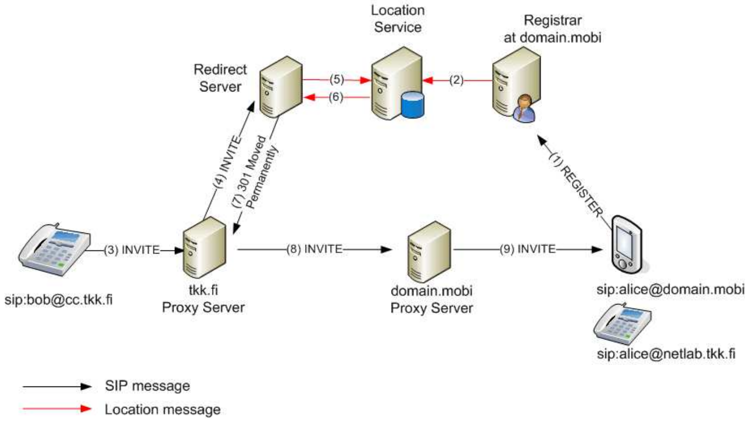


Figure 2.1: SIP server in the role of INVITE and REGISTER

different domains. The *tkk.fi* proxy server locates the proxy server at *domain.mobi*, possibly by performing a particular type of DNS(Domain Name Service) lookup to find the SIP server that serves the *domain.mobi* domain.

In addition to DNS and location service lookups shown in Figure 2.1, proxy servers can make flexible “routing decision” to decide where to send a request. For example, if Alice’s SIP phone returned a 486 (Busy Here) response, the *domain.mobi* proxy server could proxy the INVITE to Alice’s voicemail server. A proxy server can also send an INVITE to several SIP devices at the same time, for example the phone in living room and phone in kitchen. This type of parallel search is known as forking.

Redirect Server

Redirect server is a user agent that generates 3xx responses to the requester. Compared to proxy server, it does not generate any request by its own. Redirect server responds to the client with new location information. In Figure 2.1, redirect server is getting the new AOR of Alice from location service, and adds it to the Contact header of 302 Moved Temporarily. Another important functionality of redirect server is to reduce the processing load on proxy server, which improves signalling path robustness.

Registrar

Registrar is another important entity in SIP architecture. SIP offers a discovery capability. SIP is using endpoint contact address for discovery. A binding is established by the client sending REGISTER request to registrar, and registrar stores the address-of-record URI from the Contact header of REGISTER message as client current address. Once a client has established bindings at a registrar, it may send subsequent registrations containing new bindings or modifications to existing bindings as necessary.

Location service

As defined in [19], location service(or server) is the database that contains a list of bindings of address-of-record keys to zero or more contact addresses. It is used by proxy server or redirect server to obtain information of callee's possible locations. It works as DNS in SIP networks, but does not respond to or generate any requests.

2.1.2 SIP Message

Now we know all the entities in SIP networks. In this section, we will introduce the characters of SIP message itself. SIP is layered protocol. It defines different layers for the whole protocol. There are four defined layer in RFC3261 [19] for SIP. The first layer is syntax and encoding layer which is the lowest layer. It is responsible for message parsing. The second layer is transport layer which is responsible for sending and receiving messages. The third layer is transaction layer. It handles the request and retransmission. And the forth and also highest layer is transaction user layer. It creates and processes sessions, such as audio, video or text.

SIP is a text-based (UTF-8 encoding) protocol. It uses the same syntax defined by Backus-Naur Form grammar(BNF) [20] as HTTP 1.1. The advantage of text-based protocol is that it is simple and easy to understand. But it also increases the packet size. In [21], it recommends methods to compress SIP message headers. SIP messages can be generally divided into SIP requests and SIP responses. SIP responses are distinguished from requests by having a Status-Line as their start-line, shown as Table 2.1.

The start line in requests is referred to as the request line. It consists of a method name, the request URI, and the protocol version. Table 2.2 lists all SIP methods currently defined in [19]

As we state in the begin of this section, SIP is layered protocol, and it relies on many underlying protocols such as Session Description Protocol (SDP), User Data Protocol(UDP) and Stream Control Transmission Protocol(SCTP). SDP

Status code range	Meaning
100 - 199	Provisional(also called informational)
200 - 299	Success
300 - 399	Redirection
400 - 499	Client error
500 - 599	Server error
600 - 699	Global failure

Table 2.1: SIP status code and response class

Methods	Meaning
ACK	Provisional(also called informational)
BYE	Success
CANCEL	Redirection
INFO	Client error
INVITE	Server error
NOTIFY	Global failure
OPTIONS	Queries a server about its capabilities
PRACK	Acknowledges the reception of a provisional response
PUBLISH	Uploads information to a server
REGISTER	Maps a public URI with the current location of the user
SUBSCRIBE	Requests to be notified about a particular event
UPDATE	Modifies some characteristics of a session
MESSAGE	Carries an instant message
REFER	Instructs a server to send a request

Table 2.2: SIP status code and response class

is one of the most commonly used protocol by SIP. SIP uses SDP to describe its multimedia sessions, shown as Figure 2.2.

These parameters are explained in RFC 2327 [22]. Basically they describe the properties of multimedia sessions. More parameters can be found in Table 2.3:

2.1.3 SIP Transactions

A SIP transaction consists of a single request and any responses to that request, which include zero or more provisional responses and one or more final responses. A typical SIP transaction is described in Figure 2.3.

In the case of a transaction where the request was an INVITE (known as an INVITE transaction), the transaction includes the ACK only if the final response was not a 2xx response. If the response was a 2xx, the ACK is not considered part of the transaction. So in the example we shown in Figure 2.3,

```
v=0
o=bob 2890844526 2890844526 IN IP4 10.0.0.1
c=IN IP4 10.0.0.1
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Figure 2.2: Example of SDP session description

Session description	
v=	protocol version
o=	owner/creator and session identifier
s=	session name
a=	zero or more session attribute lines
Time description	
t=	time the session is active
r=	zero or more repeat times
Media description	
m=	media name and transport address

Table 2.3: SDP parameters and meanings

ACK is not included in INVITE transaction because final response 200 OK is received by requester. 100 Trying and 180 Ringing are the provisional responses here. As we present in last section, provisional responses are providing the information to the client. 100 Trying response indicates that the request has been received and processed by the UAS. It stops the UAC to retransmit INVITE to the UAS. 100 (Trying) response is different from other provisional responses, in that it is never forwarded upstream by a stateful proxy.[1]

A transaction is always driven from a client side to a server side. The client side is known as Client Transaction, and the server side is known as Server Transaction.

2.2 Overview of MANET

In recent decade, Ad Hoc network has evolved from its military background into a promised next generation network technology. The advantages of Ad Hoc network, such as cost efficiency, fast setup time, fault tolerance and possible better performance attracted intensive researches and studies.

[1] Stateful proxy and stateless proxy is not discussed in this thesis

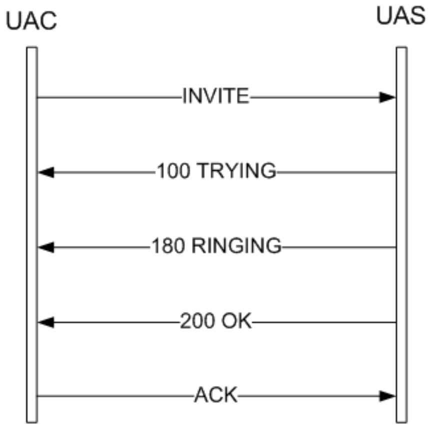


Figure 2.3: Simple SIP transactions

In 1997, the Internet Engineering Task Force (IETF) formed a working group in routing area -MANET. The study of Ad Hoc network becomes more standardised and specified.

2.2.1 History of Ad Hoc Networks

An Ad Hoc network is a (possible mobile) collection of communications devices that wish to communicate, but have no fixed infrastructure available, and have no pre-determined organization of available link [10]. The history of Ad Hoc network can be traced back of 1972 and the Department of Defence U.S. (DoD) sponsored Packet Radio Network (PRNET), which evolved into the Survivable Adaptive Radio Networks (SURAN) program in the early 1980's. The project aims to provide packet switching network to mobile battlefield elements. The PRNET used a combination of ALOHA and CSMA protocols for media access, and kind of distance-vector protocol for routing. The next version of PRNET, SURAN is improved in radio devices, scalability of algorithms, and resilience to electronic attacks. Its routing protocols are changed to hierarchical link-state protocols.

Again, DoD continued funding the program in projects such as the Global Mobile Information Systems (GloMo), and the Near-term Digital Radio (NTDR). With the improvements, GloMo networks can provide Ethernet-type multimedia connection anytime, anywhere with the handhelds in the office environment. NTDR used clustering and link-state routing, and self-organized into a two-tier Ad Hoc network. Both of GloMo and NTDR are the early stages of Ad Hoc networks.

As wireless devices are equipped with more powerful processor and larger battery capacity, the utilization of Ad Hoc network becomes easier for civilian use. Many open issues of Ad Hoc network are being researched and standardized.

2.2.2 Ad Hoc Networks Routing

Routing protocol is the most active researching area in Ad Hoc network. There are many Internet drafts and RFCs been proposed by MANET, such as Dynamic Source Routing (DSR) [11], Optimized Link State Routing Protocol (OLSR) [12], Ad Hoc On-demand Distance Vector (AODV) [13] routing and etc. According to the nature of routing protocols, Ad Hoc mobile routing protocols can be categorized into table-driven or proactive, and on-demand-driven or reactive protocols shown as Figure 2.4

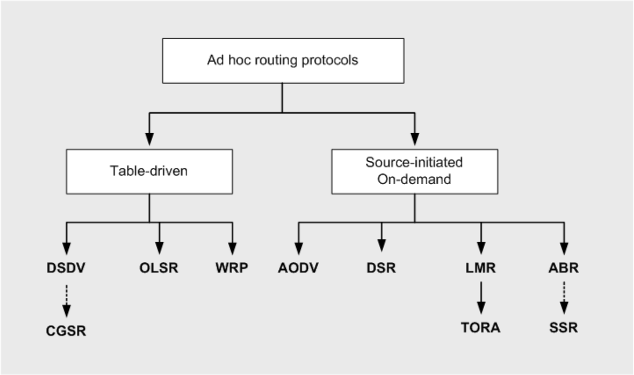


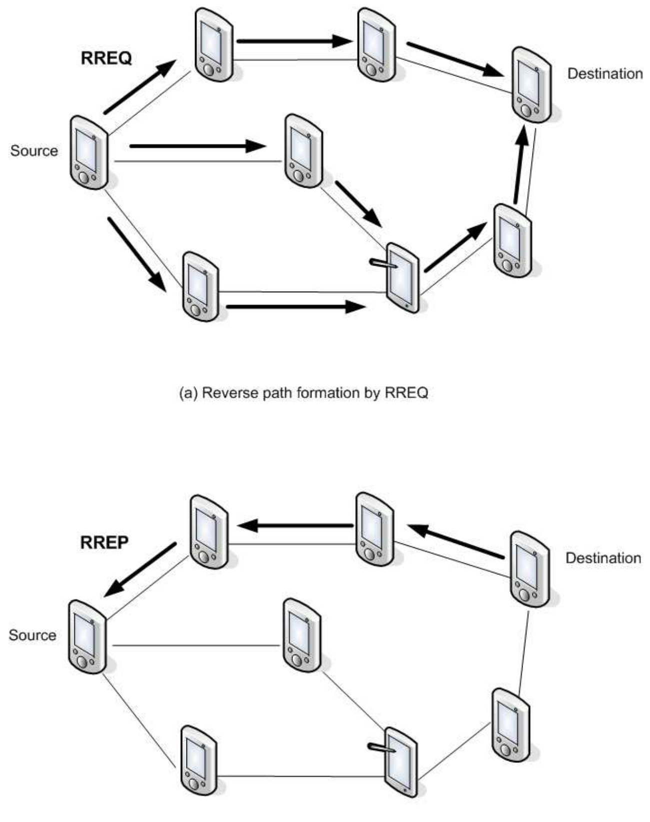
Figure 2.4: Categorization of Ad Hoc routing protocols

Within these variable routing protocols, AODV and OLSR are the most important MANET routing protocols and widely implemented on real routers.

AODV

AODV is developed from DSDV by minimizing the number of required broadcasts. The route is only created on demand instead of maintaining a complete list of routes as in the DSDV algorithm. AODV is a pure on-demand route acquisition system. The node which is not selected for the routing does not maintain routing information or participate in the routing table update. So this reduces the broadcast load for the Ad Hoc network.

When a source node wants to contact some destination node and does not have a valid route in its routing table, it will initiates a route discovery process by broadcasting a route request packet (RREQ). The source node broadcasts RREQ packet to its neighbours shown as Figure 2.5(a), and this process is repeated till an intermediate node is reached that has recent route information to the destination or till it reaches the destination. AODV uses destination sequence numbers to ensure that all routes are loop-free and contain the most recent route information. The intermediate nodes only reply (RREP) to route requests with latest information. Each node maintains its own sequence number, as well as a broadcast ID. The broadcast ID is incremented for each RREQ the node initiates, and together with the node's IP address, uniquely identifies an RREQ. Along with its own sequence number and broadcast ID, the source node includes in the RREQ the most recent sequence number it has for the destination [14]. Figure 2.5(b) represents the forward path setup as the RREP travels from the destination to the source node.



(a) Reverse path formation by RREQ

(b) Forward path formation by RREP

Figure 2.5: AODV route discovery

In AODV, routes are maintained as following scenarios: If a source node moves, it has to reinitiate the route discovery protocol to find a new route to the destination. If an intermediate node moves, its upstream neighbour notices the move and propagates a link failure notification message (an RREP with an infinity metric) to each of its active upstream neighbours to inform them of the erasure of that part of the route [15].

OLSR

The mechanism of OLSR is different from AODV. It operates as a table driven, proactive protocol, i.e., exchanges topology information with other nodes of the network regularly. Each node selects a set of its neighbour nodes as "multipoint relays"(MPR) (N1 and N6 in Figure 2.6). In OLSR, only nodes selected as MPRs, are responsible for forwarding control traffic, intended for diffusion into the entire network. MPRs provide an efficient mechanism for flooding control traffic by reducing the number of transmissions required.

Nodes, selected as MPRs, also have a special responsibility when declaring link state information in the network. Indeed, the only requirement for OLSR to provide shortest path routes to all destinations is that MPR nodes declare link-state information for their MPR selectors. Additional available link-state information may be utilized, e.g., for redundancy.

Each node in the network, for example node N2, selected a few neighbour nodes in the network. These nodes will send node N2-packets. These selected nodes, N1 and N6 are called MPR nodes of N2. N2 selects its MPR to cover all the nodes that are exactly two hops away from it. In our example: N7, N8, N9 and N4. A node which is not a Multipoint Relay can read the packet sent from N2 but cannot forward it.

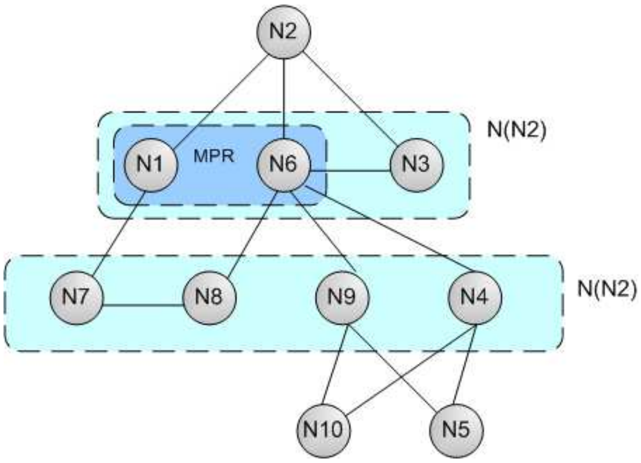


Figure 2.6: OLSR multipoint relays and regular nodes

2.2.3 Multicasting in MANET

The challenge of multicasting in Ad Hoc network is the frequent node movement and maintenance of group state information. The conventional tree based multicast protocols do not work well in Ad Hoc network because they cause excessive signalling overhead and frequent loss of packets. The tree organization activities in MANET are more frequent compared to wired network since the multicast protocols have to respond to the node movement in addition to the group.

There are many multicast protocols that have been proposed for Ad Hoc networks. Most of them are extensions version of static multicast network routing algorithms, such as flooding, Center Based Tree (CBT), Protocol Independent Multicast (PIM) and Reverse Path Forwarding (RPF). Table 2.4 is a comparison of different multicast protocols in Ad Hoc networks. Among them, Multicast Ad Hoc On-Demand Distance Vector (MAODV) [16] and On-demand Multicast Routing Protocol (ODMRP) [17] are the most widely implemented protocols.

Protocols	**Topology**	**Loop-free**	**Unicasting**	**Periodic Packet**	**Flooding**
Flooding	Mesh	Yes	No	No	Yes
AMRoute	Hybrid	No	Yes	Yes	Yes
AMRIS	Tree	Yes	No	Yes	Yes
MAODV	Tree	Yes	Yes	Yes	Yes
LAM	Tree	Yes	Yes	No	No
LGT-based	Tree	Yes	No	Yes	No
ODMRP	Mesh	Yes	No	Yes	Yes
CAMP	Mesh	Yes	Yes	Yes	Yes
DDM	Stateless Tree	Yes	No	Yes	No
FGMP-RA	Mesh	Yes	Yes	Yes	Yes
FGMP-SA	Mesh	Yes	No	Yes	Yes
MCEDAR	Hybrid	Yes	Yes	Yes	Yes

Table 2.4: Comparison of Ad Hoc multicast protocols

MAODV is associated with AODV. MAODV discovers multicast routes using a broadcast route discovery mechanism . A group leader (mobile node) is associated with each multicast group, and is responsible to initialize and maintain the multicast group sequence number. Periodically broadcast Group Hello (GRPH) messages are sent by group leader across the multicast group. The multicast route discovery begins either when a node wishes to join a multicast group or when it wants to send the data to a multicast group which it does have a route to.[18] When a node wants to join the multicast group, it sets the 'J' flag in the RREQ packet, and the destination IP address is always set to the multicast group address and that contains the multicast group's last known sequence number. Then the node broadcasts the RREQ to its neighbors as the normal AODV procedure. For the join request, a route is determined when the RREQ reaches a node that is already a member of the multicast tree, and the multicast group sequence number in this node's multicast route table is as great as the sequence number in the RREQ. Once the source node received the RREP packet within the RREP_WAIT_TIME, it selects the shortest route to the multicast tree and unicast the next hop a Multciast Activations (MACT) message along that route.

Another situation is the node want to send data to the multicast group. In this case, the flag of RREP packet will not be set. And if the node knows the group leader of multicast group and has a route to it, it may unicast RREP message with the group leader's IP address in the Multicast Group Leader extension.

ODMRP is mesh based routing protocol using a forward group concept. Forwarding group is a set of nodes responsible for forwarding multicast data, and to build a forwarding mesh for each multicast group. Thanks to the mesh topology and soft state approach, advantages of ODMRP can be found from its simplicity, low channel and storage overhead, usage of up-to-date shortest routes, reliable construction of routes and forwarding group, robustness to host mobility, maintenance and exploitation of muliple redundant paths, exploitation of the broadcast nature of wireless environments and unicast routing capability.

2.3 SIP over MANET

There is much related work [4, 5, 6, 7] that has been carried out on decentralized peer-to-peer SIP networks. The most important works are SOSIMPLE [4] and [7]. In [4], the authors use DHT based on Chord to organize the nodes. Each node is assigned a node ID which is the hash value of its real IP address. Ad Hoc nodes maintain a small number of finger table entries associated with SIP REGISTER message to pass the overlay information between nodes. When a new node join the overlay, it send a REGISTER message with its Node-ID to the bootstrap node to obtain the finger table information. If the bootstrap node is not the node currently responsible for this region, it will send a 302 redirect message with information about the nodes it knows nearest to where the joining node will be placed in the overly.

Chang et al. combined UPnP and SIP in a single hop Ad hoc network in [5]. UPnP protocol is used to collect all nodes information of this network, such as IP address and user corresponding name. SIP is used for signalling after the node discovery has been done. Each node in the network functions as both the control point and the device. It uses SUBSCRIBE message to request the state of remote nodes with an assigned period. But it does not cover the SIP session initialed from outside of the Ad hoc network. There is also no clear solution for SIP session between the public network and the private Ad hoc network.

Above two work are different with each other in two areas:

1) How to collect necessary information during the starts up time

2) How to initiate and manage the node joining and leaving

There are a lot of similarities between the peer-to-peer network and the ad hoc network: (1) both have a flat and frequently changing topology, caused by node join and leave in P2P overlays and MANETs and additionally terminal mobility of the nodes in MANETs; and (2) both use hop-by-hop connection establishment. Per-hop connections in P2P are typically via TCP links with physically unlimited range, whereas per-hop connections in MANETs are via wireless links, limited by the radio transmission range.[8]

In [6], Simone et al. defined an architecture for full decentralized Ad hoc SIP networks. The architecture consists of two major parts: Service Discovery frameworks and Session Management. The service discovery frameworks can be implemented by Service Location Protocol (SLP), Jini, UPnP [1], Bluetooth Service Discovery and Salutation.

Very similar to the work described in [6], the SIPHoc architecture in [9] is based on four components: MANET SLP layer, SIPHoc Proxy, Gateway Provider and Connection Provider. All components are running as independent processes within a node in MANET. The MANET SLP module clearly defined

[1] UPnP will be introduced in Chapter 3

the service registration and lookup process in MANET using SLP. It uses routing handler for receiving the raw routing packets and generating altered packets that include the piggybacked service information. This makes SIPHoc does not require any modification on the routing protocols.

2.4 Summary

In this chapter, we started with the overview of SIP and MANET technologies. Then we examined the work that is carried on SIP over P2P or MANET. We found the similarities between these work, such as they all defined two steps in their solutions: First step is to manage the peer discovery that is related with the routing protocols and underlaying networks. Second step is to manage the service discovery that is required to set up SIP sessions. In the next chapter, we will start to introduce the peer discovery and service discovery.

Chapter 3

Peer Discovery and Service Discovery

In previous chapter, we have described separately how Ad Hoc network routing and conventional SIP works. And we know due to the characters of SIP protocol, it is challenging to deploy such an infrastructure-based application layer protocol directly to Ad Hoc networks, specially on MANET. SIP is designed for a primarily "fixed" and relatively static network environment where communication links are stable and exhibit fairly uniform communication characteristics [23]. SIP endpoint discovery in an Ad Hoc network is semantically similar to the service or peer discovery process in P2P networks. To overcome *node mobility* challenges, we need to answer two important questions:

- Where are SIP endpoints and services allocated?
- How to find them?

In this chapter, we will try to answer the two questions, and introduce the peer [1] and service discovery frameworks in Ad Hoc networks. And finally focus on UPnP protocol which is used in our implementation.

3.1 Peer Discovery

Peer discovery refers to discovery of mobile node in MANET at network layer and below, shown in Figure 3.1. As we mentioned in Chapter 1, Chord and

[1] Here we use the term "peer" to refer both the mobile node in Ad Hoc networks (or endpoint in SIP sessions) and peer node in P2P networks

Pastry are the most common algorithms chosen as peer lookup algorithms in P2P networks. There are some similar features for both P2P networks and Ad Hoc networks. (a) There is no peer in either networks that acts as a server. (b) The major challenges in both networks are how to efficiently find the requested data or route. (c) The topology of both networks changes frequently.

But there are also big differences between P2P networks and Ad Hoc networks. (a) a P2P network is basically an application network, it refers to the application layer in the network stacks, while MANET focuses on the network and lower layers. (b) The peer in MANET is restricted to limited computing power, bandwidth and battery life, but peer of P2P network does not necessary have these limitations. (c) For the routing process, a P2P network only generates the key ID table based on peer's existing IP address, while MANET requires the real-time routing and IP address allocating.

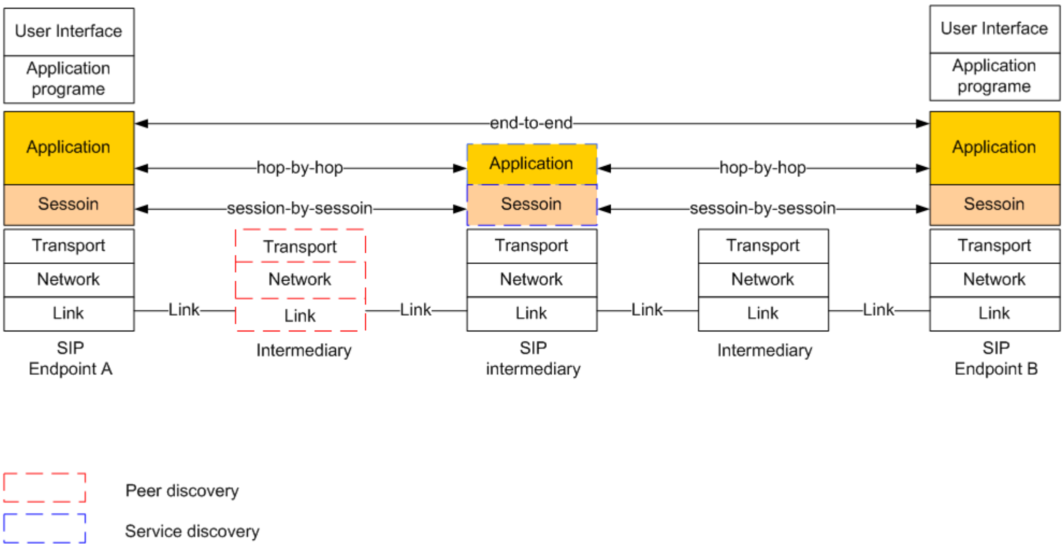


Figure 3.1: Hop-by-hop Peer discovery in IP layer and End-to-end Service discovery in Application layer

3.2 Service Discovery

We briefly described in previous chapter about related works on decentrialized SIP on Ad Hoc networks. In [7], SIP user agents supports more functionalities as they were only supported by SIP server in conventional SIP networks. As the result, the intermediate node in MANET is acting as proxy server itself during the session initial stage. And after the media session is set up, intermediate node receives and forwards the media data packets. The problem here is that not all MANET nodes are SIP capable or voice service enabled. So we need a mechanism to find the proper intermediate nodes which is enabled with SIP or voice service. The potential service discovery protocols that can be used are SLP [24], Jini [25], UPnP [26], Bluetooth Service Discovery [27] and Salutation. In this section, we introduce some of the important service discovery protocols. There are many other protocols used for service discovery, we only focus on standardized ones. Specifically the scope is set to use UPnP.

3.2.1 Service Location Protocol

"SLP provides a flexible and scalable framework for providing hosts with access to information about the existence, location, and configuration of networked services" [24]. It defines three entries within the framework: *User Client* (UA), *Server Agent* (SA)and *Directory Agent* (DA).The way of communications between each other is very flexible. (a) UA can directly multicast the *service request* (SrvRqst) message to SAs, and receive a unicasting *service reply* (SrvRply) message from the SA who has the required service. In this case, it is very like M-SEARCH message in UPnP networks. (b) If DA is present in the network, both UA can send unicast SrvRqst message to DA, and SA can send unicast *service register* (SrvReg) message to DA. UA or SA discovers DA in a bootstrapping mechanism. DA or SA sends multicasting SrvRqst message to the network and receives a unicast *DA Advertisement* (DAAdvert) message from DA. Another possible way for DA discovery is DA infrequently sends multicasting DAAdvert message to the entire network. So practically in both cases, DA is acting as a centerlized entry in SLP protocol.

3.2.2 Jini

Jini protocol is a service discovery framework that defines a programming model which exploits and extends Java technology to enable the construction of secure, distributed systems consisting of federations of well-behaved network services and clients [25]. It is developed by Sun Microsystems. The key concept is a *lookup service* is responsible for finding and resolving the *service.* Every service consists of a *service object* and *service attributes.* The communications between client service and lookup service is based on Java Remote Method Invocation (RMI) which is a Java native interface for Java's remote process call (RPC) mechanism. So it highly depends on Java enabled devices.

3.2.3 Bluetooth Service Discovery

Bluetooth special interest group (SIG) defines its own stack for data transport. It consists of physical layer and logical layer. Bluetooth service discovery protocol (SDP) is based on Logical Link Control and Adaptation Protocol (L2CAP) which is laid on top of Bluetooth logical layer. SDP involves communication between an SDP server and an SDP client. The services (service record) maintained by SDP server is uniquely identified by universally unique identifier (UUID). "There is a maximum of one SDP server per Bluetooth device. (If a Bluetooth device acts only as a client, it needs no SDP server.) A single Bluetooth device may function both as an SDP server and as an SDP client. If multiple applications on a device provide services, an SDP server may act on behalf of those service providers to handle requests for information about the services that they provide." [27] The information exchange between client and server is dynamically based on the RF proximity of the servers to the client. Whenever a server is available, the clients within the Bluetooth RF reachability range must be notified. The logic of Bluetooth SDP is very similar to SLP but in a reactive way.

3.3 Universal Plug and Play protocol

In above section, we briefly explained SLP, Jini and Bluetooth SDP protocols. And we know both Jini and Bluetooth SDP are highly dependent on their underlying technologies. This limitation makes them not easy to deploy on Ad Hoc networks. Although SLP is a more generic protocol, it is designed in a centralized manner. So after simple evaluation, UPnP is a simpler protocol which can be easily adapted to MANET without modification the protocol itself.

3.3.1 Overview of UPnP

"UPnP technology defines an architecture for pervasive P2P network connectivity of intelligent home appliances, wireless devices, and PCs of all form factors. It is designed to bring easy-to-use, flexible, standards-based connectivity to Ad Hoc or unmanaged networks whether in the home, in a small business, public spaces, or attached to the Internet." [26] The primary mission of UPnP Forum is to develop device control protocol that describe standard methods for device interaction. UPnP protocol stack consists of a set of standard protocols such as UDP, HTTP, HTTPU, HTTPMU, SSDP, GENA and SOAP illustrated in Figure 3.2 taken from [28]. It makes UPnP working seamless with other technologies such as TCP/IP and web.

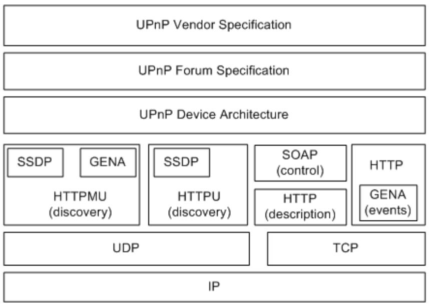


Figure 3.2: UPnP protocol stack

UPnP device specification defines three logical elements for UPnP networks: *service*, *devices* and *control points* showing as Figure 3.3.

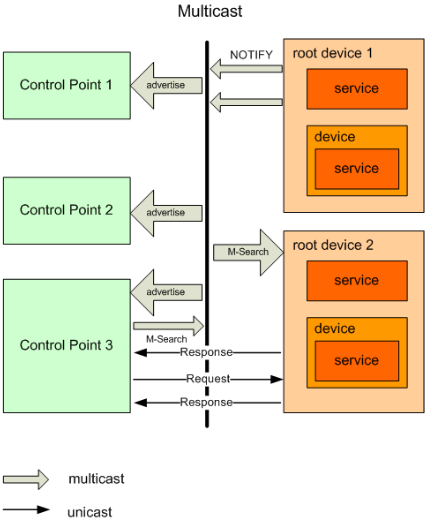


Figure 3.3: UPnP architecture

Every UPnP device is simply organized by *device*, *service* and *control point*. Extensible Markup Language (XML) is used as a core part in UPnP for describing device, service, control point. The specification also specifies six steps of interaction between above three elements:

- Step 0 *Addressing* indicates how control points and devices get an IP address

- Step 1 *Discovery* where control points become aware of the existence of devices
- Step 2 *Description* where control points learn details about devices and services
- Step 3 *Control* where control points send command to devices
- Step 4 *Eventing* where control points listen to state changing of devices
- Step 5 *Presentation* where devices display a user interface for devices

In following sections, we will briefly introduce the things that related to our implementation .

3.3.2 Addressing

The foundation of UPnP networking is IP addressing. There are two methods for UPnP control points and devices to get IP addresses, either by Dynamic Host Configuration Protocol (DHCP) or by automatic IP addressing (Auto-IP). Each UPnP control point or devices must have a DHCP client implemented if it is not implemented as DHCP server. Control points or devices first send a DHCPDISCOVER message, if a DHCPOFFER is received during the defined period, control points or devices must continue to obtain IP addresses from DHCP server accordingly. If no DHCPOFFER is received, they must use Auto-IP [29]instead. Auto-IP defines how control points or devices get proper IP address from link-local range (169.251.0.0/16) in UPnP networks when DHCP server is not available. This IP address assignment method enables more flexibility as control points and devices move between managed network and Ad Hoc network.

Upon successful selection of a link-local IP address, it has to be tested within the entire UPnP network. Control points or devices uses an Address Resolution Protocol (APR) [30] probe to determine whether the chosen IP is already in use.

An ARP probe request contains four critical fields SHA, SPA, THA, TPA[3] to determine IP collision. The broadcasting probe request sets SHA to UPnP

[3] Sender Hardware Address (SHA), Sender Protocol Address(SPA), Target Hardware Address(THA) and Target Protocol Address(TPA)

control point or device's hardware address, SPA to all zeros, THA to all zeros and TPA to the link-local IP address which it chooses. During the pre-defined period (PROBE_MAX minus PROBE_MIN seconds), (a) if the sender receives any ARP packet on the interface where the probe is being performed where the packet's SPA is the IP address being probed for, (b) or if the sender receives any ARP packet with TPA set as the same IP address being probed for, then the sender must treat this address as being in use by some other host [29] and it has to choose another IP address.

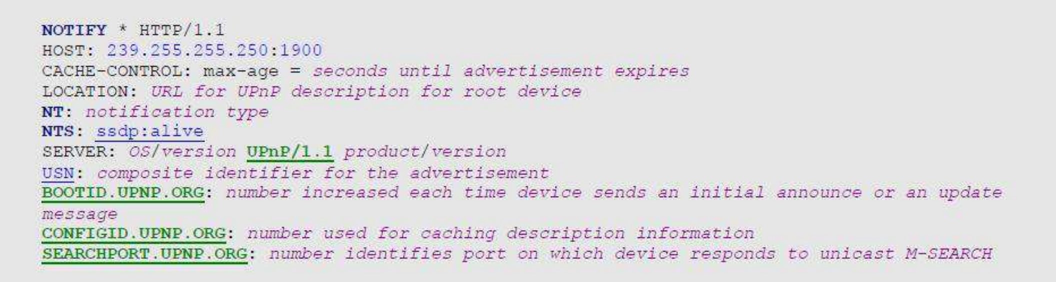
UPnP also allows universal resource location (URL) type of name for device's IP address. So it can be located and referenced on the network through that address. But the limitation of link-local IP address is that it does not support subnetworks. So UPnP network is a pure flat network.

3.3.3 Discovery

Discovery mechanism in UPnP network is very similar to SLP. Both of them use multicasting requests for discovery of interested service, and unicast reply is sent by matched device or client. The difference is there is no centralized component acting as DA in UPnP networks.

In UPnP networks, there are several situations involving discovery. (a)When a device joins or leaves the network, (b)When a control point joins or leaves the network, (c)When a control point wants to find device or service of interest.

In situation (a) and (b), device and control point needs to advertise the entire network with *NOTIFY* message, shown as Figure 3.4.



```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age = seconds until advertisement expires
LOCATION: URL for UPnP description for root device
NT: notification type
NTS: ssdp:alive
SERVER: OS/version UPnP/1.1 product/version
USN: composite identifier for the advertisement
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or an update
message
CONFIGID.UPNP.ORG: number used for caching description information
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```

Figure 3.4: Example SSDP header fields

Each message must have exactly one start-line and several header fields. In this case the start-line is NOTIFY * HTTP/1.1. The HOST request-header field specifies the Internet host and port number of the resource being requested. In above example, it specifies a well-known local scope multicast IP address 239.255.0.0/16 [31]. And the port 1900 is uniquely used by SSDP. The CACHE-CONTROL general-header field is used to specify directives that must be obeyed by all caching mechanisms along the request/response chain. UPnP uses the max-age directive to specify the live time (in seconds) of advertisement message. The value of max-age directive must be greater than or equal to 1800 seconds. LOCATION header gives the URL to UPnP description of the root device. SERVER header is defined for product tokens. Syntax of the three headers are defined in HTTP protocol.

UPnP advertisement also includes headers defined by SSDP. *Notification Type* (NT) contains a single URI indicating whether the notification is for root device, embedded device or service. *Notification Sub Type* (NTS) defines the purpose of notification. Such as notifying when:

- device or control point becomes alive (NTS:ssdp:alive)
- device or control point is unavailable(NTS:ssdp:byebye)
- device or control point is updated (NTS:ssdp:update)

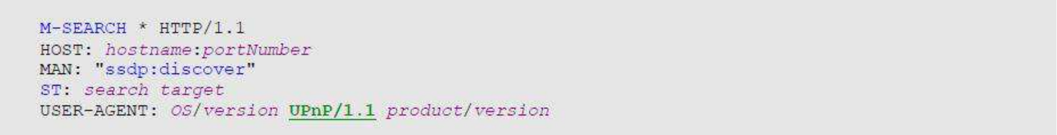
Unique Service Name (USN) also uses a single URI to identify a unique instance of a device or service. Some other headers from discovery advertisement are defined by UPnP Forum. Such as *BOOTID.UPNP.ORG*, *CONFIGID.UPNP.ORG*, *SEARCHID.UPNP.ORG* and *NEXTBOOTID.UPNP.ORG*.

In situation (c), the control point needs to send either a multicast or unicast *M-SEARCH* message. M-SEARCH message contains *Mandatory* (MAN) header, *Maximum wait time* (MX) header , *Search Target* (ST) header and *USER-AGENT* header. MAN, MX and USER-AGENT headers are defined in HTTP Extension frameworks.

MX header defines the time devices should respond to **multicast M-SEARCH** shown as Figure 3.6. Devices should wait a random time between 0 seconds and number of seconds specified in the MX field value of the search request before responding. The purpose for doing this is to avoid flooding the requesting control point with search responses from multiple devices especially in our Ad Hoc networks. For **unicast M-SEARCH** shown as Figure3.5, in case the control point has the cached IP address of target device, any device should respond within 1 second.

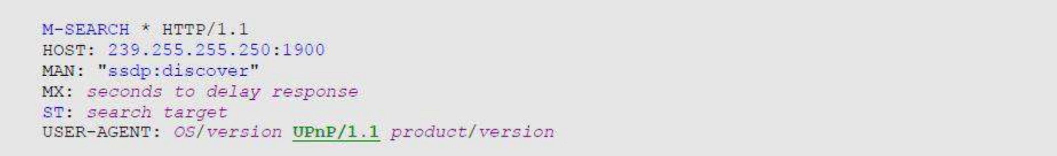
ST header specifies which type of services or devices that control point searches for. The value of ST header uses a single URI to indicate the search target. Search target can be one from following list:

- ssdp:all
- upnp:rootdevice
- uuid:*device-UID*
- urn:schema-upnp-org:device:*deviceType*:*version*
- urn:schema-upnp-org:service:*serviceType*:*version*
- urn:*domian-name*:device:*deviceType*:*version*
- urn:*domian-name*:service:*serviceType*:*version*



```
M-SEARCH * HTTP/1.1
HOST: hostname:portNumber
MAN: "ssdp:discover"
ST: search target
USER-AGENT: OS/version UPnP/1.1 product/version
```

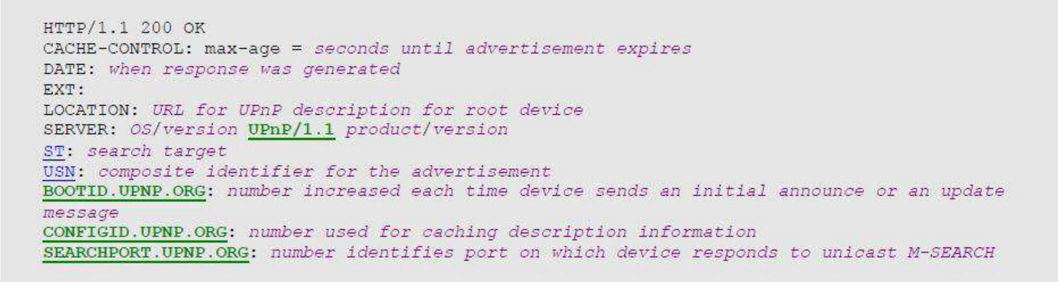
Figure 3.5: Example of Unicast M-SEARCH message



```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: search target
USER-AGENT: OS/version UPnP/1.1 product/version
```

Figure 3.6: Example of Multicast M-SEARCH message

Upon a successful search, *HTTP 1.1/200 OK* 3.7 replies will be send from target devices. In the responding message, CACHE-CONTROL header specifies how long the advertisement is valid in the network in seconds. And LOCATION header contains a URL to the UPnP description file of the root device. Normally it is presented in literal IP address rather than a domain name in unmanaged networks.



```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until advertisement expires
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for root device
SERVER: OS/version UPnP/1.1 product/version
ST: search target
USN: composite identifier for the advertisement
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or an update
message
CONFIGID.UPNP.ORG: number used for caching description information
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```

Figure 3.7: Example of 200 OK Response message

3.4 Summary

In this chapter, we first explained the difference between the peer discovery and the service discovery. The hop-by-hop peer discovery is referred as the node discovery in the network layer. While the end-to-end service discovery is referred as the overlay in the application layer. Secondly we explained some service discovery protocols in Section 3.2. Finally we focused on UPnP specifically because it would be used in our application design in the next chapter.

Chapter 4

Technology Background

In this chapter we will present some of the open source software (OSS) and technologies that have been used in our implementation. In Section 1, we will introduce open source library called Sofia SIP, a SIP user agent library compliant with RFC3261 [19] specifications. In Section 4.2, we will introduce another open source library called CyberLinkC UPnP library which is used by our service discovery part of implementation. And in Section 3, we will briefly explain how GTK+ and Hildon-2.0 libraries works on Nokia Maemo platform [32].

From Section 4, we start to explain the technologies and devices that have been used in our development and implementation. So in Section 4, we will introduce Nokia Internet Tablet device. And in Section 5, we will give a brief explanation about how to work on Maemo platform with Scratchbox [33].

4.1 Sofia SIP

Sofia SIP is based on RFC3261 and other RFCs used by SIP. It is developed by Nokia Research Center and has been widely used by many voice applications such as Telepathy, Gaim, and Far Sight. The reason for choosing Sofia SIP is because it supports major SIP standards and is under active development to track RFC changes. It provides better interoperability and extensibility to our application if other developer wants to extened the functionality of our SIP module.

Sofia SIP consists of different modules such as common runtime library, SIP signalling, HTTP subsystem and SDP processing in order to support full SIP functionalities. The modules used in our implementation are common runtime libarary and SIP signalling module called NUA (SIP User Agent Library).

4.1.1 Common Runtime Libraries

Common runtime libraries provided by Sofia SIP include sofia utility library (su), asynchronous DNS resolver (sresolv) and IP Telephony utility library (ipt). Utility library defined tags used in SIP message. The structure of Sofia SIP message is organized by different tags. Utility library also provides some APIs such as time, message digest 5 algorithm (MD5), thread handling, memory allocation, debugging and logging. Asynchronous DNS resolver library provides the interface for communicating with DNS server. IP Telephony library contains some useful routines for telephony applications, such as Base64 (charactors from A-Z, a-z, 0-9, + and \) encoding and decoding and HTTP header tokens encoding.

4.1.2 Signaling Module

One of Sofia SIP signalling modules we used is the user agent model (NUA). It is the primary API for SIP enabled applications. It gives the high-level application programmer transparent and full control to the SIP protocol engine below it (e.g. SIP transaction engine called nta). With NUA it is possible to create different kind of SIP user agents, like SIP terminals, gateways and multipoint conferencing unit (MCU).

Other libraries of signaling module include SIP event API, HTTP and SIP authentication API, SIP transaction engine, generic transport protocols API, SIP message parser, MIME header parser and URL parser.

The Sofia software suite is based on certain basic ideas and concepts that are used in all levels of Sofia software. We will describe some of the concepts that a user of NUA library has to understand to create a working application.

The NUA uses the reactor pattern for its event driven programming model. According to this model, the program can ask that the event loop invokes a callback function when a certain event occurs. An application using NUA services must create a root object and the callback routine to handle NUA events. The root object can be created by using `su_root_creat()` function and register the callback function by using `nua_creat()`.

4.2 Cybergarage UPnP Implementation

There are various UPnP reference implementations available from UPnP Forum website. The UPnP stack in our voice application is built on CyberLinkC library. The reason why we choose CyberLinkC is that it is a simple library based on C. It is small (binaries size less than 2Mb) and easy to be deployed on embedded Linux devices, such as PDA with limited memory. Also it is a native UPnP library for Nokia Internet Table device.

CyberLinkC library implements the functionalities of control points and devices defined in UPnP standard document [26]. CyberLinkC also provides simple SOAP message parser, HTTP server and SSDP server. As discussed in previous chapter, they are responsible for handling UPnP control and discovery events. Some more utility functions are also provided by CyberLinkC library including XML parsing functions, string functions and thread functions specifically used by UPnP.

An application implemented as UPnP device must define its description file as a string first. then simply use `cg_upnp_device_new()` to create the instance. But this API only initiates UPnP device without start the thread. So `cg_upnp_device_start()` has to be called to start the device, and `cg_upnp_device_stop()` to stop the device. The same mechanism for UPnP embedded devices, control points and services. It is also possible to configure control point and device to listen specific type of package, such as HTTP response listener and SSDP response listener by setting up callback functions for listeners. The callback functions are very critical for the application to respond the event requests, and programer has to set listeners before start the device or control point.

4.3 GTK+ 2.0 and Hildon-2.0

The GIMP Toolkit (GTK) was originally designed for a raster graphics editor called the GNU Image Manipulation Program (GIMP). It was originally used on the Linux operating system. GTK+2 is the second stable release cycle of GTK+. There are a few new features introduced since GTK+2, such as font-rendering engine called Pango and new enhanced theme engine. GTK+2 is not compatible with old GTK+1 branch since a lot of changes are introduced to GTK+2, and API compatibility is broken.

Hildon is an application framework introduced a new desktop environment for handheld devices. It was originally from Psion user interface code named Hildon after the bottled water. The current Hildon running on Nokia Linux-based tablet is an open source GTK based user interface toolkit. It comprises a lightweight desktop, a set of widgets optimized for handheld devices, a set of theming tools and other complementary libraries and applications [34]. It is adopted by Nokia Maemo graphic framework and handles Maemo specific features of applications, such as look and feel (colors, fonts and borders etc.), and menu in the title bar area instead of windows area, and hibernation feature.

A typical Hildon layouts consists of Task Navigation area, Titlebar area, and Application area and three areas of inactive skin graphic. All views could be manipulated by using GTK+ container widgets (e.g. GtkHbox, GtkVbox and GtkLabel).

4.4 Development on Maemo

Maemo is a customized Linux desktop distribution running on Nokia Internet Tablet devices. It is very similar with Debian distribution. Maemo mainly uses open desktop frameworks which includes GTK+ toolkit, Hildon, C programming language, and Scratchbox cross-compiler toolkit to enable easy software portability and familiarity. Maemo also uses Debian package management system (`dpkg`) for installing, removing and providing information about applications.

C and C++ language are the native programming language for Maemo. Anyone familiar with C and C++ will feel easy and comfortable with development on Maemo. Most of common used C libraries are supported by Maemo and can be installed by using `dpkg` tool. And Maemo uses message bus system (D-BUS) service and LibOSSO combination for managing remote procedure call (RPC) on Maemo platform.

4.4.1 D-BUS

D-BUS is a system for interprocess communication (IPC) widely used in many systems. D-BUS is designed for two specific cases:

- Communication between desktop applications in the same desktop session

```
[D-BUS Service]
Name=org.maemo.example_app
Exec=/usr/bin/example_app
```

Figure 4.1: Example of D-BUS Service File

- Communication between desktop session and the operating system

On Maemo platform, D-BUS is used for system notification message, separating UI application and service engine and launching applications from Task Navigator. Any application wants to be visible from Task Navigator has to have a service file. D-BUS service file is needed to be able to launch the Maemo application and connect it to D-BUS services. All applications have to register to D-BUS daemon and the service file is installed on `/usr/share/dbus-1/services/`. A sample service file `example_app.service` is shown as Figure 4.1

4.4.2 LibOSSO

LibOSSO is a wrapper library for user application for platform specific and frequently used D-BUS services. The purpose is to avoid D-BUS details from Maemo platform specific D-BUS services. Any process will be killed by D-BUS from the desktop environment if it is launched from Task Navigator but not registered to D-BUS services. LibOSSO can protect this process from terminating by D-BUS.

Chapter 5

System Architecture

In the previous chapter we introduced the technologies used in our implementation. In this chapter we will explain in detail how the system is designed. Firstly we will give an overview of Ad Hoc VoIP system. Secondly we will demonstrate different scenarios of communication between Ad Hoc nodes and fixed nodes. Finally, we will present our implementation and performance analysis.

5.1 System Overview

Our system consists of three modules *UPnP stack*, *SIP UA* and *VoIP application* shown in Figure 5.1.

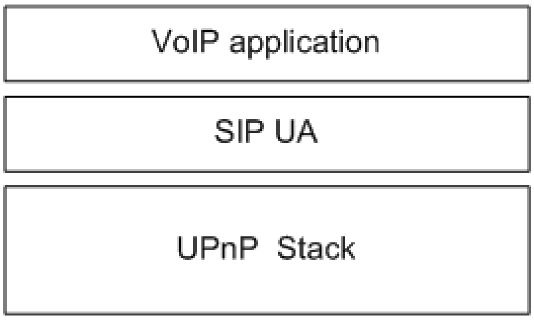


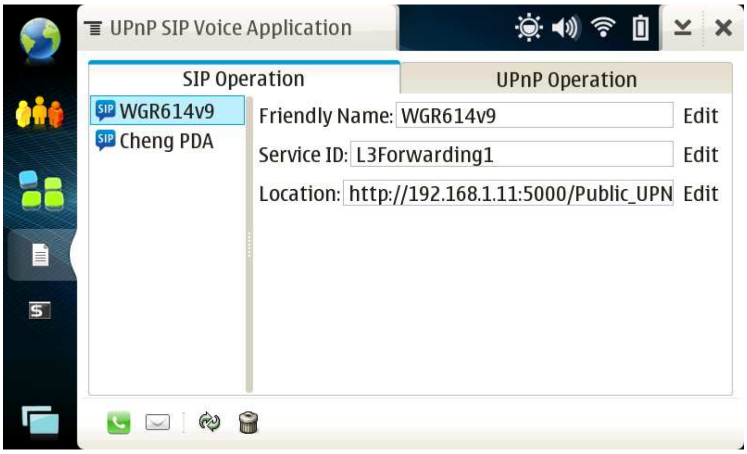
Figure 5.1: System Architecture

VoIP is the module that interacts with end users. SIP UA is the module responsible for handling SIP messages and initialization of SIP connections. UPnP stack works as the module for service discovery and collecting necessary information for SIP UA. We discuss each module in following sections in detail.

5.1.1 VoIP Application

The VoIP module is the user interface for our application. End users directly interact with this module to use the functions such as placing and terminating phone calls, sending and receiving text messages, searching UPnP devices and services. The VoIP module also provides the interface for end users to configure their SIP and UPnP profiles, such as SIP AOR, UPnP friendly name, device SSDP response repeat times etc.

The graphic user interface (GUI) of our application is based on the Maemo platform. It is implemented with GTK+ and Hildon. Figure 5.2 shows the main window of VoIP application. Under the SIP operation tab, SIP call, SIP messaging and SIP contacts refresh and delete icons are located in the toolbar area.



Friendly Name: WGR614v9

Service ID: L3Forwarding1

Location: http://192.168.1.11:5000/Public_UPN

Figure 5.2: Voice Application Main Window

5.1.2 SIP UA

Below the VoIP module, we define another middle layer called SIP UA module. It directly communicates with the underlying UPnP module. The SIP UA module is equivalent to presentation and session layers of OSI standard. The major function of this module is to set up the media connection between two nodes. Ad Hoc nodes can function as UA, proxy server or redirect server in our system according to different scenarios. We will explain in more detail in Communication Scenarios section.

As we mentioned in the previous chapter, SIP UA module is implemented with Sofia SIP libraries and the Sofia SIP client example called Sofsip-cli.

5.1.3 UPnP Stack

UPnP stack is the lowest module in our system. In order to enable communications between mobile nodes without a centralized server, all mobile nodes are implemented as both the control point (controller) and the root device. According to the UPnP device architecture document, each node has two XML files to present the properties of the UPnP root device and UPnP services.

Listing 5.1: Example Device Description

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0" configId="
   configuration␣number">
  <specVersion>
    <major>1</major>
    <minor>1</minor>
  </specVersion>
  <device>
    <deviceType>urn:schemas-upnp-org:device:PDA:1</
       deviceType>
    <friendlyName>N810</friendlyName>
    <manufacturer>Nokia</manufacturer>
    <manufacturerURL>www.nokia.com</manufacturerURL>
    <modelDescription>PDA</modelDescription>
    <modelName>Internet Tablet</modelName>
    <modelNumber>N810</modelNumber>
    <modelURL>http://www.forum.nokia.com/devices/N810/</
       modelURL>
    <serialNumber>00:19:4f:94:a5:90</serialNumber>
    <UDN>uuid:nokian810</UDN>
    <UPC>123456789012</UPC>
    <iconList>
      <icon>
        <mimetype>image/gif</mimetype>
        <width>48</width>
        <height>32</height>
        <depth>8</depth>
        <url>icon.gif</url>
      </icon>
    </iconList>
    <serviceList>
      <service>
          <serviceType>urn:schemas-upnp-org:service:sip:1
             </serviceType>
          <serviceId>urn:upnp-
```

```
            org:serviceId:sip:cluo@iptel.org</serviceId>
         <SCPDURL>/service/sip/audio_description.xml</
            SCPDURL>
         <controlURL>/service/sip/control</controlURL>
         <eventSubURL>/service/sip/eventSub</eventSubURL
            >
       </service>
    </serviceList>
    <deviceList>
    </deviceList>
    <presentationURL>/presentation</presentationURL>
  </device>
</root>
```

An example of the device description file is shown in List 5.1. A device description file provides information about the root device, embedded devices and available services of a UPnP device. This example device description file shows that the root device type is a *PDA* with friendly name *N810*, and service type is defined as *SIP*. The value of the service ID tag contains the SIP AOR information. We use the service ID to identity the SIP end-points.

Listing 5.2: Example Service Description

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
<specVersion>
  <major>1</major>
  <minor>0</minor>
</specVersion>
<actionList>
  <action>
    <name>getPresences</name>
    <argumentList>
       <argument>
         <name>newPresence</name>
         <relatedStateVariable>Presence</
            relatedStateVariable>
         <direction>in</direction>
       </argument>
       <argument>
         <name>Result</name>
         <relatedStateVariable>Result</
            relatedStateVariable>
         <direction>out</direction>
       </argument>
    </argumentList>
    </action>
    <action>
      <name>setPresence</name>
      <argumentList>
        <argument>
          <name>CurrentPresence</name>
          <relatedStateVariable>Presence</
             relatedStateVariable>
          <direction>out</direction>
        </argument>
      </argumentList>
    </action>
</actionList>
<serviceStateTable>
  <stateVariable sendEvents="yes" multicast="yes">
    <name>Presence</name>
    <dataType>string</dataType>
    <defaultValue>Offline</defaultValue>
  </stateVariable>
  <stateVariable sendEvents="no">
    <name>Result</name>
    <dataType>string</dataType>
  </stateVariable>
</serviceStateTable>
</scpd>
```

The other XML file is the service description file, shown in List 5.2, that describes the details of the UPnP service. In this XML file, we defined two types of action which allow the control point to get or set SIP presences of the device. In fact, each Ad Hoc node in the network is a control point, so the node can query SIP presences from other nodes. The presence service is normally provided by the SIP registra server in the centralized network.

5.2 Test Scenarios

In Section 5.1, we explained the basic functions of each module. In this section, we will further explain different test scenarios when using this application to make SIP calls. To better understand our application usage in real situations, we defined three different scenarios of communication between two nodes: 1) Communication initialized by an Ad Hoc node to another Ad Hoc node; 2) communication initialized by an Ad Hoc node to another fixed node; 3) communication initiated by a fixed node to another Ad Hoc node. We will go through each scenario individually later.

5.2.1 Scenario 1: An Ad Hoc node to another Ad Hoc node

The simplest scenario for our VoIP application is that the initiator and destination of SIP call are both located within the same Ad Hoc network. The network architecture is shown in Figure 5.3.

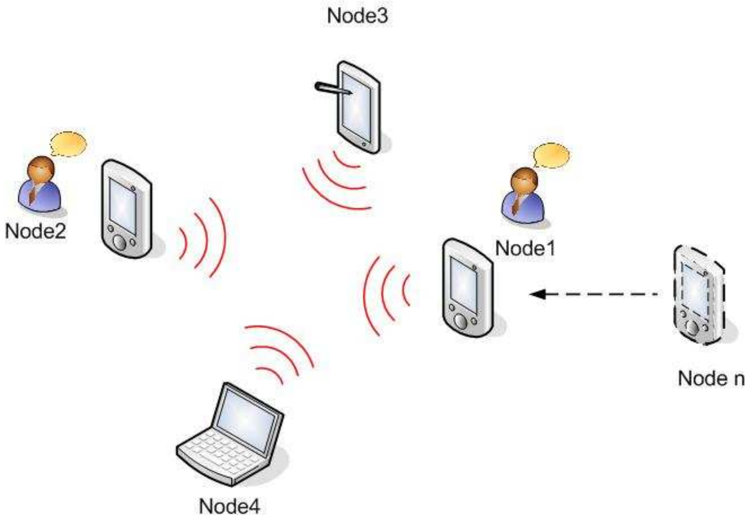


Figure 5.3: Scenario 1: Ad Hoc node to Ad Hoc node

In this scenario, there is no registrar server or redirect server needed for signalling. The service discovery is done by the UPnP module, and the VoIP connection is established by the SIP UA module. The flow chart of UPnP message and SIP signalling is shown in Figure 5.4.

We define the three different states for the call setup process, shown on the left side of Figure 5.4. The right side shows steps of UPnP. When a new node joins the UPnP network, it multicasts NOTIFY and M-SEARCH messages to the entire network. Ad Hoc nodes can search UPnP devices either by device type or service type as we described in Chapter 3. In our system, Ad Hoc nodes use service type for the search target because we are only interested in nodes capable with SIP service.

The Initialization state is the state that the new joining node collects the network information and caches the SIP routing table for the future use. The

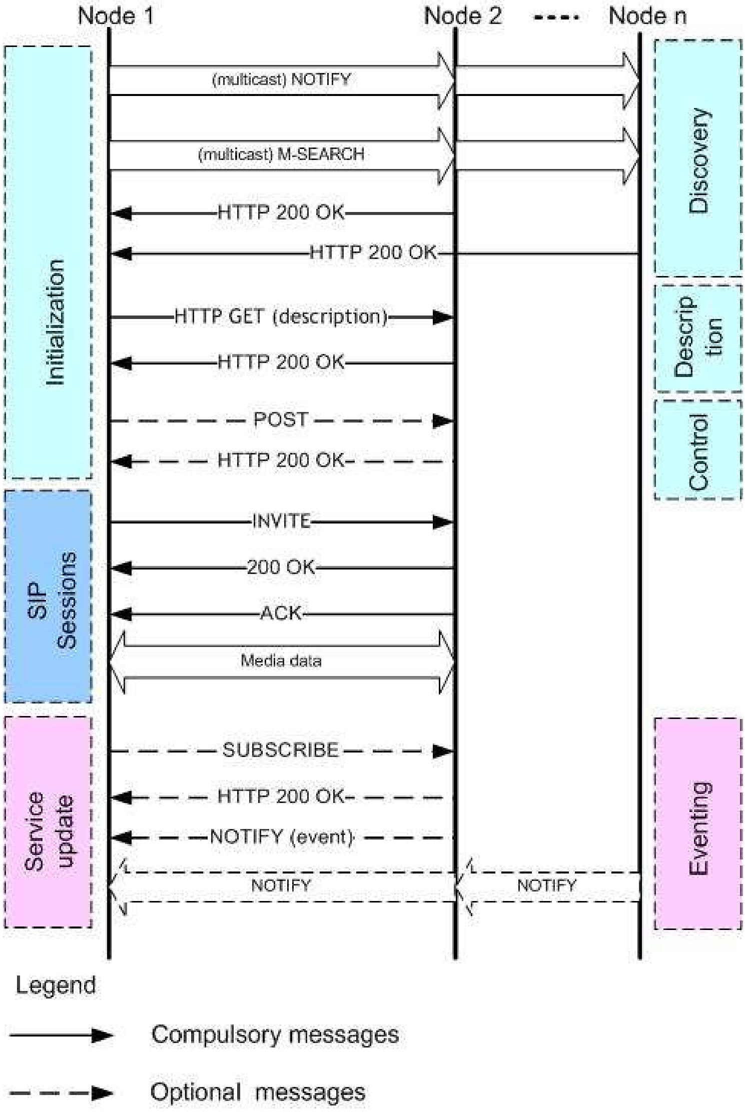
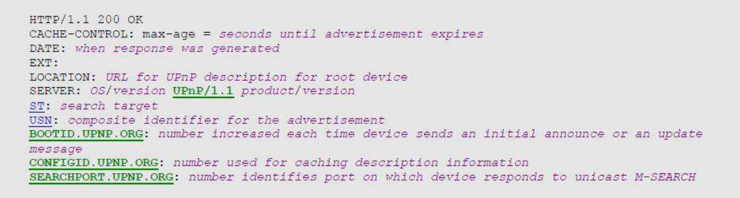


Figure 5.4: System Flow Chart: Ad Hoc node to Ad Hoc node

SIP sessions state is where SIP transactions are happened. The service update state is an option state for the nodes who wants to update and share the presence information.

Meanwhile every Ad Hoc node has SSDP response package listeners set up to be able to parse SSDP responses from other nodes. Figure 5.5 shows an example of 200 OK message responding to the M-SEARCH message. By sending a multicast M-SEARCH message, Node1 will randomly get 200OK responses from all devices. But using the randomly received SSDP responses is not possible to build SIP AOR and IP address bindings for the node. The LOCATION header of the 200 OK message only contains the primitive IP address of the SIP enabled node (e.g. Node2). So Node1 has to send a HTTP GET message to



```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until advertisement expires
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for root device
SERVER: OS/version UPnP/1.1 product/version
ST: search target
USN: composite identifier for the advertisement
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or an update
message
CONFIGID.UPNP.ORG: number used for caching description information
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```

Figure 5.5: Example of 200 OK message

the URL provided in Node2's SSDP response to get its description file. Within the device description file, Node1 will get Node2's SIP AOR information from service ID. Up to now, Node1 successfully builds up the binding of Node2's IP address and SIP AOR. Node1 can now call Node2 with its IP address.

5.2.2 Scenario 2: An Ad Hoc node to a fixed node

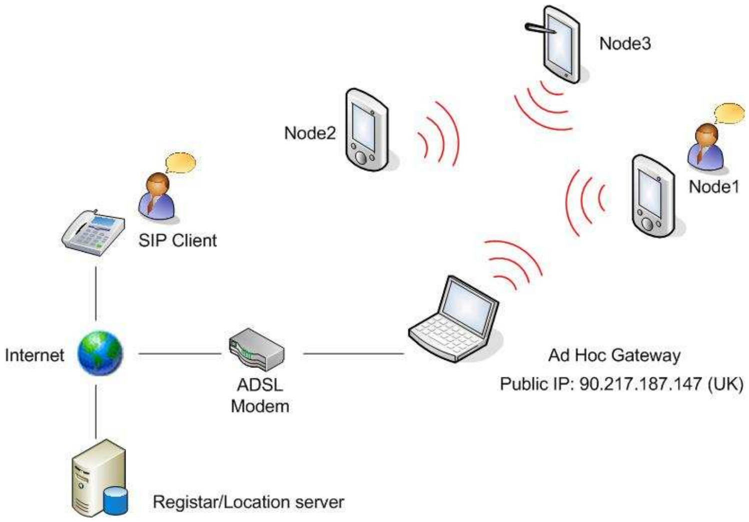


Figure 5.6: Scenario 2: Ad Hoc nodes to fixed nodes

Another common scenario is that an Ad Hoc node wants to contact a SIP client which is not available in the Ad Hoc network. Figure 5.6 shows the network architecture of this scenario. The Ad Hoc node is located in an Ad Hoc network, and the fixed node is located in a public IP network.

The flow chart of Scenario 2 is shown in Figure 5.7. Similar to Scenario 1, Node1 sends M-SEARCH messages and receives NOTIFY messages to collect as much as information it can. Once Node1 finishes the initialization stage, it is aware which devices have SIP services available. When Node1 wants to call SIP client with SIP AOR (e.g. sip:alice@example.com), it sends M-SEARCH message with service type defined as SIP. After a timeout, if Node1 has not received

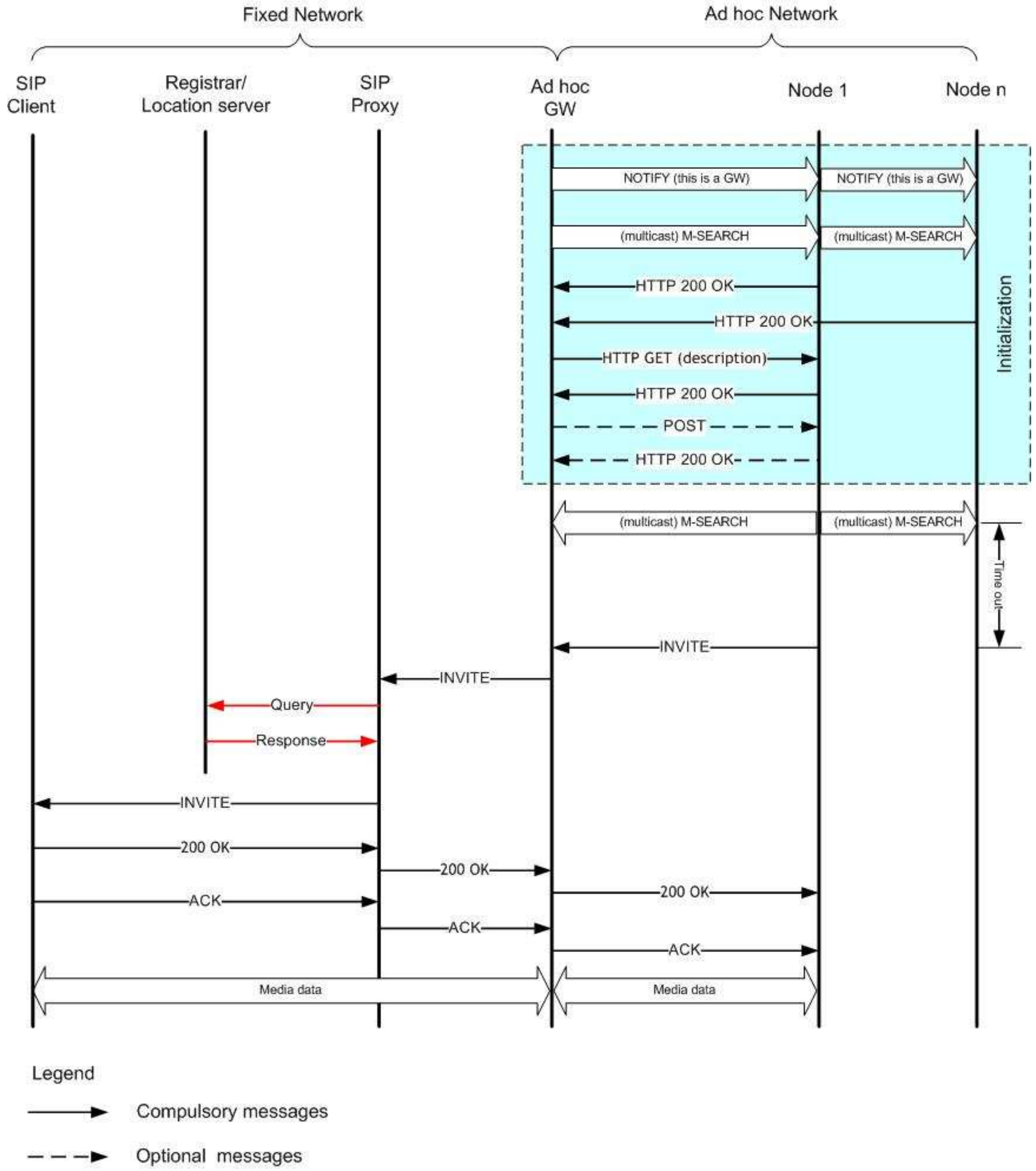


Figure 5.7: System Flow Chart: Ad Hoc node to fixed node

the SSDP response with destination SIP AOR (e.g. sip:alice@example.com), Node1 will send an INVITE message to an Ad Hoc node which has gateway service enabled. Once the Ad Hoc gateway node receives the INVITE message, it forwards the INVITE message to a SIP proxy server located in the public network. The SIP proxy should allocate the current location of the destination SIP client by sending a query message to a SIP registrar or location server. Upon a successful response from the SIP registrar or location server, the SIP proxy forwards the INVITE message to the destination SIP client.

In this scenario, we introduce an Ad Hoc node acts as a gateway. In our Ad hoc network, there is only one node acting as the gateway. It has the connections both to the current Ad Hoc network and the public Internet. The gateway node is also acting as a standard SIP proxy server.

5.2.3 Scenario 3: A fixed node to an Ad Hoc node

The network architecture of scenario 3 is similar to scenario 2 which is shown in Figure 5.6. But the caller and callee are located vice versa, and the behaviour of Ad Hoc gateway is quite different from scenario 2. The challenge is how the SIP client finds the IP address of a node inside an Ad Hoc network. To solve this issue, we design an Ad Hoc node acting as SIP proxy between the Ad Hoc domain and the public domain. Registration creates bindings in a location service for a particular domain that associates an address-of-record UIR with one or more contact addresses. So an Ad Hoc node which intends to be reachable from outside Ad Hoc network has to register itself first.

Figure 5.8 illustrates the message flow chart of scenario 3. When Node1 intends to be contacted from outside Ad Hoc network, it searches for the Ad Hoc gateway node first, and sends a REGISTER message to the Ad Hoc gateway node. The Ad Hoc gateway inserts a "`Via`" header with its public IP address, then forwards the REGISTER message to a SIP registrar which is designated as the registrar server. An example of a REGISTER message is shown in Figure 5.9. The binding of Ad Hoc gateway's public IP address and Node1's current SIP URI is added to the location service.

When a SIP client from the public network wants to call Node1, the SIP registrar sends back the location of Ad Hoc gateway. The SIP proxy server will forward the INVITE message to the Ad Hoc gateway additonally. Once the Ad Hoc gateway receives any INVITE messages from the public network, it checks whether the destination SIP URI of the INVITE message matches its own URI. If it does not match, the Ad Hoc gateway sends an M-SEARCH message to the Ad Hoc network and searches for all the SIP enabled nodes to see whether any of them match this SIP AOR. If Node1 is found, the Ad Hoc gateway forwards the INVITE message to Node1. From now on, Node1 send the 200OK message to the Ad Hoc gateway. The Ad Hoc gateway acts like a SIP proxy server when forwarding SIP messages between the local link IP addresses to the public IP address.

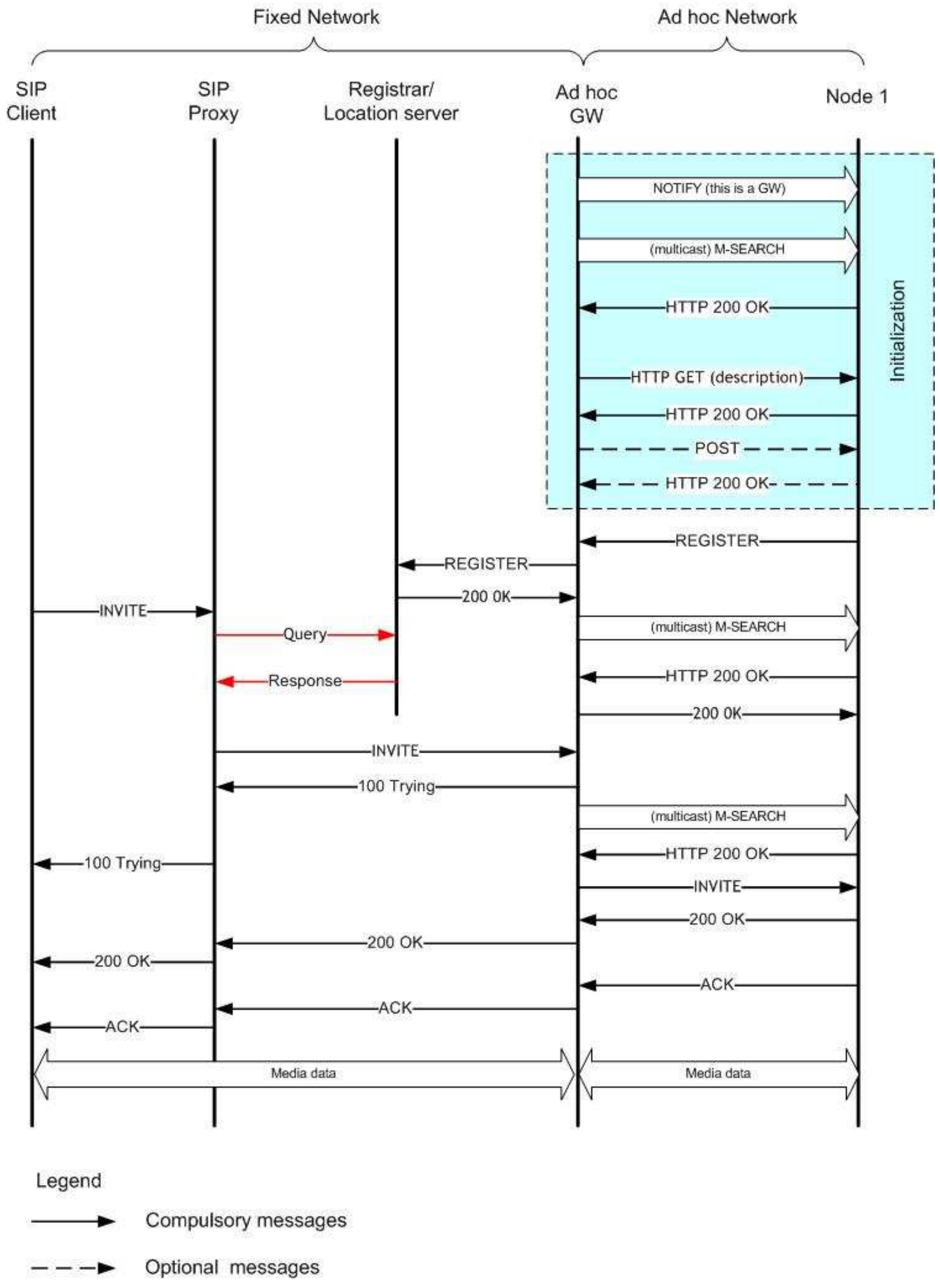


Figure 5.8: System Flow Chart: Fixed node to Ad Hoc node

5.3 System Implementation

In this section, we explain the high level design of the voice application implementation. As we described in the System Overview section, our application

```
REGISTER sip:iptel.org SIP/2.0
Via: SIP/2.0/UDP 90.217.187.147:5060;
branch=z9hG4bKjy8hGfv
Via: SIP/2.0/UDP 169.254.0.3:5060;
branch=z9hG4bKnashds7;received=3ffe:501:ffff:1::1
Max-Forwards: 69
From: Node1 <sip:chengluo@iptel.org>;tag=a73kszlfl
To: Node1 <sip:chengluo@iptel.org>
Call-ID: 1j9FpLxk3uxtm8tn@iptel.org
CSeq: 1 REGISTER
Contact: <sip:chengluo@pda.iptel.org>
Expires: 3600
Content-Length: 0
```

Figure 5.9: Example of forwarding REGISTER request

can be subdivided into three modules. The VoIP module contains GUI components implemented by GTK+ and Hildon; SIP UA module contains SIP parser and proxy components available from Sofia SIP library; UPnP module is based on CybergarageC UPnP reference implementation.

5.3.1 VoIP Components

VoIP components are top level components of entire application. A structure view of the VoIP module is shown in Figure 5.10. The top level data struct `AppUIData` contains both UI data and application data. `AppData` is the C global struct holding all the data needed in our application. Member variables `programe`, `osso`, and `window` are variables for initializing a GUI programme on Maemo platform as we introduced in previous chapter. `ctrl_point` and `root_device` are handlers for UPnP activities. `sip_cli` is variable for handling SIP messaging and phone call. `routing_table` is the linked list data structure for caching the SIP routing information.

5.3.2 Sofia SIP Components

We use the Sofia SIP library for our SIP user agent implementation. The structure view is show in Figure 5.11. Sofia SIP is written in C language. It does not strictly follow the object oriented model. The class diagram just gives a big picture of relationship between Sofia data structures. Sofia SIP provides

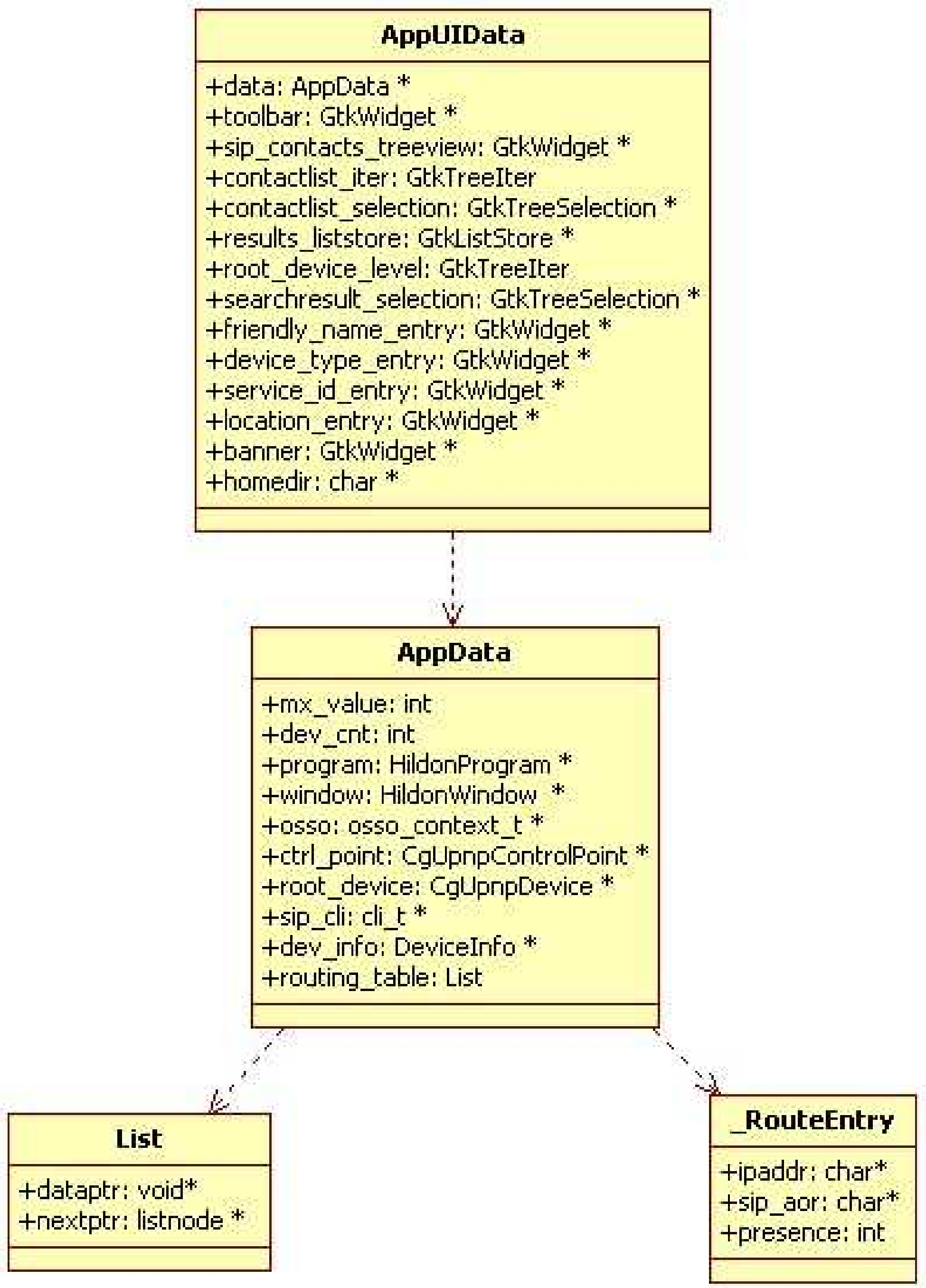

AppUIData
+data: AppData *
+toolbar: GtkWidget *
+sip_contacts_treeview: GtkWidget *
+contactlist_iter: GtkTreeIter
+contactlist_selection: GtkTreeSelection *
+results_liststore: GtkListStore *
+root_device_level: GtkTreeIter
+searchresult_selection: GtkTreeSelection *
+friendly_name_entry: GtkWidget *
+device_type_entry: GtkWidget *
+service_id_entry: GtkWidget *
+location_entry: GtkWidget *
+banner: GtkWidget *
+homedir: char *
AppData
+mx_value: int
+dev_cnt: int
+program: HildonProgram *
+window: HildonWindow *
+osso: osso_context_t *
+ctrl_point: CgUpnpControlPoint *
+root_device: CgUpnpDevice *
+sip_cli: cli_t *
+dev_info: DeviceInfo *
+routing_table: List
List
+dataptr: void*
+nextptr: listnode *
_RouteEntry
+ipaddr: char*
+sip_aor: char*
+presence: int

Figure 5.10: Application Module Structure Overview

a simple tag based SIP parser. Figure 5.12 shows how Sofia SIP generate and parses BYE message.

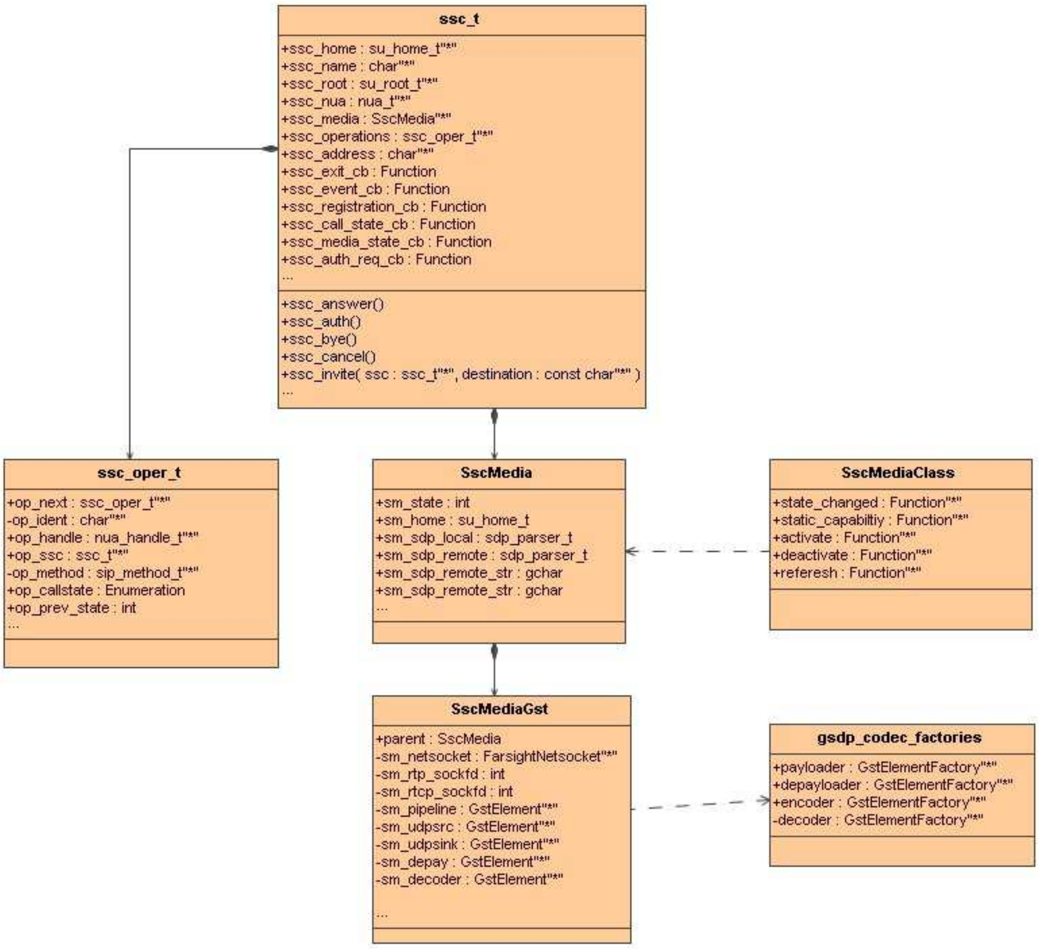

ssc_t
+ssc_home : su_home_t"*"
+ssc_name : char"*"
+ssc_root : su_root_t"*"
+ssc_nua : nua_t"*"
+ssc_media : SscMedia"*"
+ssc_operations : ssc_oper_t"*"
+ssc_address : char"*"
+ssc_exit_cb : Function
+ssc_event_cb : Function
+ssc_registration_cb : Function
+ssc_call_state_cb : Function
+ssc_media_state_cb : Function
+ssc_auth_req_cb : Function
...
+ssc_answer()
+ssc_auth()
+ssc_bye()
+ssc_cancel()
+ssc_invite(ssc : ssc_t"*", destination : const char"*")
...
ssc_oper_t
+op_next : ssc_oper_t"*"
-op_ident : char"*"
+op_handle : nua_handle_t"*"
+op_ssc : ssc_t"*"
-op_method : sip_method_t"*"
+op_callstate : Enumeration
+op_prev_state : int
...
SscMedia
+sm_state : int
+sm_home : su_home_t
+sm_sdp_local : sdp_parser_t
+sm_sdp_remote : sdp_parser_t
+sm_sdp_remote_str : gchar
+sm_sdp_remote_str : gchar
...
SscMediaClass
+state_changed : Function"*"
+static_capabiltiy : Function"*"
+activate : Function"*"
+deactivate : Function"*"
+referesh : Function"*"
SscMediaGst
+parent : SscMedia
-sm_netsocket : FarsightNetsocket"*"
-sm_rtp_sockfd : int
-sm_rtcp_sockfd : int
-sm_pipeline : GstElement"*"
-sm_udpsrc : GstElement"*"
-sm_udpsink : GstElement"*"
-sm_depay : GstElement"*"
-sm_decoder : GstElement"*"
...
gsdp_codec_factories
+payloader : GstElementFactory"*"
+depayloader : GstElementFactory"*"
+encoder : GstElementFactory"*"
-decoder : GstElementFactory"*"

Figure 5.11: SIP Structure Overview

5.3.3 UPnP Components

As we mentioned in the previous section, each Ad Hoc node acts as both a control point and a device. The purpose of this is to handle UPnP activities in an unmanaged network. We defined a initialization stage when mobile nodes join in the network. UPnP mobile nodes automatically send multiple NOTIFY and M-SEARCH messages to the UPnP network, and collects necessary information needed by SIP UA. The purpose of doing this is to provide shorter start up time.

CybergarageC provide a set of APIs: `cg_upnp_controlpoint_getdevices()` and `cg_upnp_device_getservices()` to get the device list and service list. From these APIs we can retrieve the available UPnP devices and services. Figure 5.13 demonstrates the class view of CybergarageC libraries. Similarly to Sofia SIP, CybergarageC is also written in C. The class diagram just illustrates

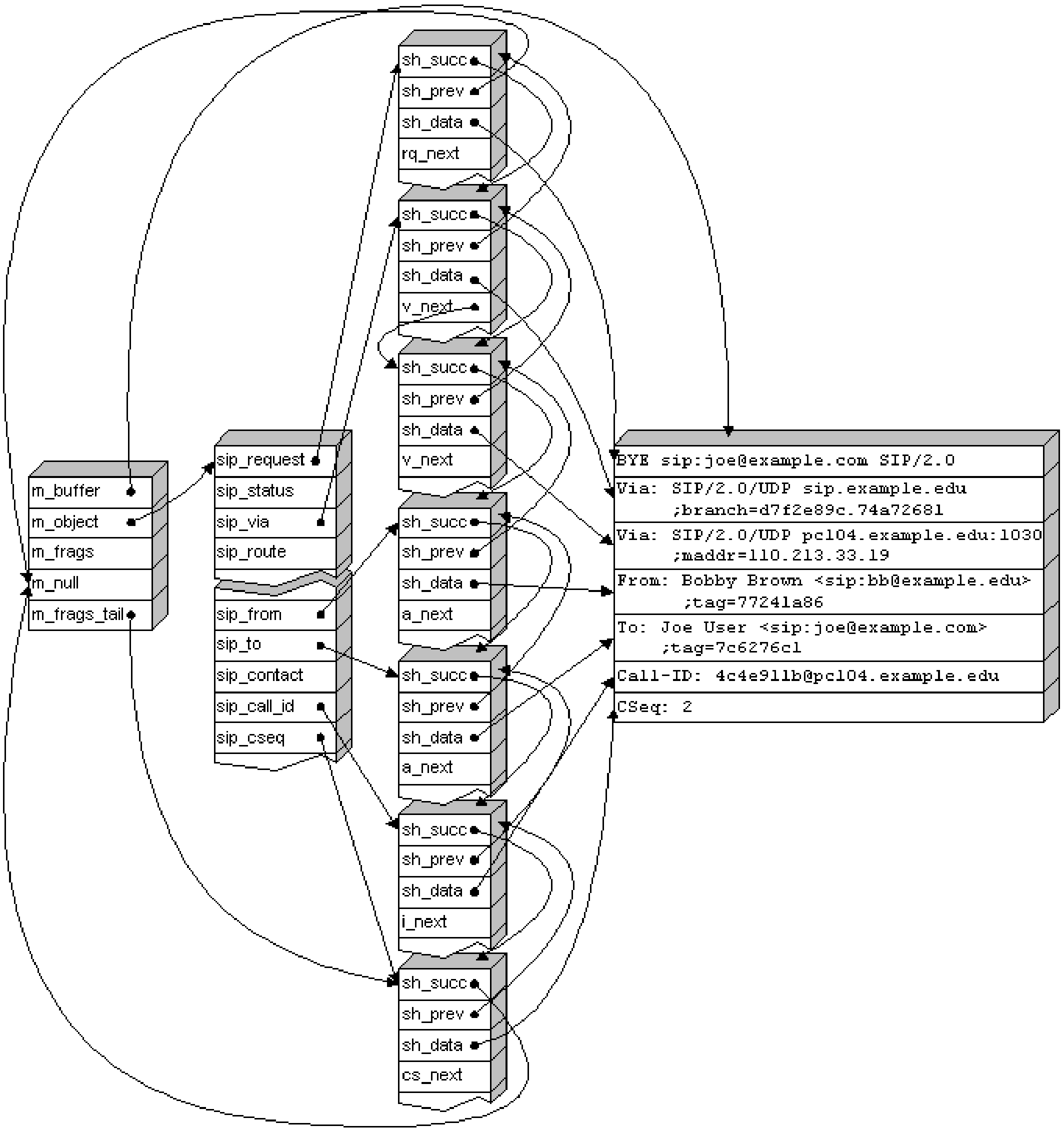


Figure 5.12: SIP parser example: BYE message

the relationship between different C data structures. `CgUpnpControlPoint` is the top level data structure and contains most of the other types of data. By setting up different type of listeners, we can parse information from different packets. As shown in the class diagram, we can get the LOCATION header from SSDP Responses by setting up SSDPResponseListener for the control point.

Another important piece of information for our application is the ServiceID of the device. To get the ServiceID, we use the internal methods of the UPnP control point to access the member variables of CgUpnpControlPoint.

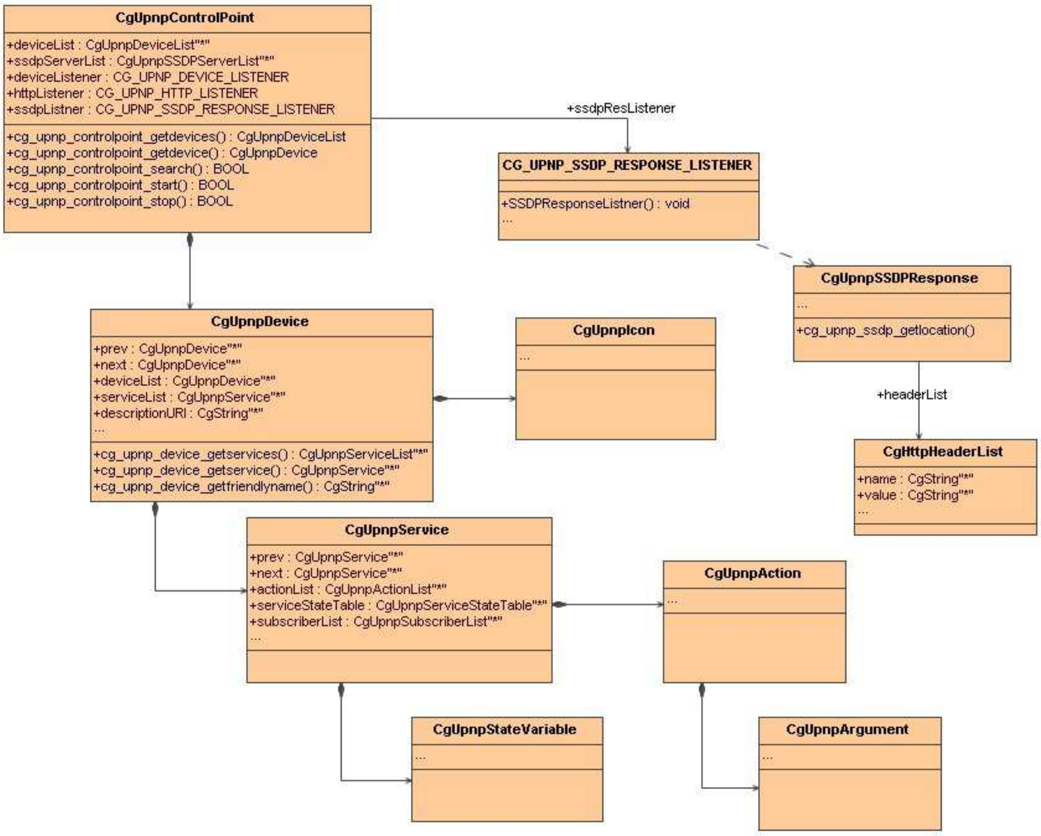

CgUpnpControlPoint
+deviceList : CgUpnpDeviceList"*"
+ssdpServerList : CgUpnpSSDPServerList"*"
+deviceListener : CG_UPNP_DEVICE_LISTENER
+httpListener : CG_UPNP_HTTP_LISTENER
+ssdpListner : CG_UPNP_SSDP_RESPONSE_LISTENER
+cg_upnp_controlpoint_getdevices() : CgUpnpDeviceList
+cg_upnp_controlpoint_getdevice() : CgUpnpDevice
+cg_upnp_controlpoint_search() : BOOL
+cg_upnp_controlpoint_start() : BOOL
+cg_upnp_controlpoint_stop() : BOOL
+ssdpResListener
CG_UPNP_SSDP_RESPONSE_LISTENER
+SSDPResponseListner() : void
...
CgUpnpSSDPResponse
...
+cg_upnp_ssdp_getlocation()
+headerList
CgHttpHeaderList
+name : CgString"*"
+value : CgString"*"
...
CgUpnpDevice
+prev : CgUpnpDevice"*"
+next : CgUpnpDevice"*"
+deviceList : CgUpnpDevice"*"
+serviceList : CgUpnpService"*"
+descriptionURI : CgString"*"
...
+cg_upnp_device_getservices() : CgUpnpServiceList"*"
+cg_upnp_device_getservice() : CgUpnpService"*"
+cg_upnp_device_getfriendlyname() : CgString"*"
CgUpnpIcon
...
CgUpnpService
+prev : CgUpnpService"*"
+next : CgUpnpService"*"
+actionList : CgUpnpActionList"*"
+serviceStateTable : CgUpnpServiceStateTable"*"
+subscriberList : CgUpnpSubscriberList"*"
...
CgUpnpAction
...
CgUpnpStateVariable
...
CgUpnpArgument
...

Figure 5.13: UPnP Module Structure Overview

5.4 Summary

In this chapter, firstly we have described the system architecture of our VoIP application. It consists of three modules: Application/UI module, SIP UA module and UPnP module. Secondly we have explained the functionality of each module in difference scenarios. At the end, we have presented the relationships between each modules. in the next chapter, we will start to test each scenarios based on our implementations.

Chapter 6

Testing and Analysis

This chapter focuses on the testing and operation of our application. The chapter is divided into two sections: The first section explains how to set up the test environment. The second section analyses results from the different test scenarios that were described in Chapter 5.

6.1 Demonstration Setup

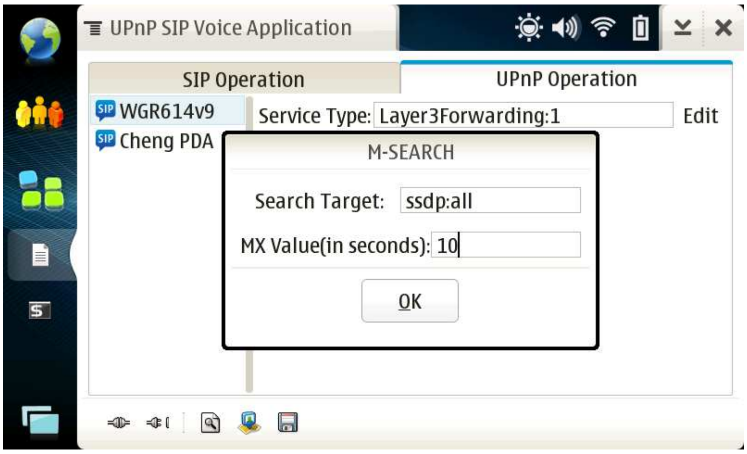
In our demonstration, we are not using any underlying Ad Hoc routing protocols for the multi-hop communication. For simplicity, we only test our application on the single hop Ad Hoc network shown in Figure 5.3. The network architecture of demonstration is shown in Figure 5.3. The real network consists of three Nokia Internet Tablets represented as Node1, Node2 and Node3, and one laptop. Table 6.1 shows the settings of different devices. Using the wireless utility tool (connection-switcher) provided by Maemo repository, we can set the static Ad Hoc IP address for each tablets, set the subnet mask to 255.255.255.0, and default gateway to 192.168.4.15. There is no Ad Hoc routing protocol implemented in our test bed, all tests are based on broadcasting nature.

Entry	**SIP AOR**	**IP**
N800-1	sip:n800-1@example.com	192.168.4.1
N800-2	sip:n800-2@example.com	192.168.4.2
N810	sip:n810@example.com	192.168.4.3
Laptop	sip:cluo@iptel.org	192.168.4.4

Table 6.1: Test Environment

Figure 6.1 shows the dialog of sending M-SEARCH messages. The MX value indicates the waiting time that control points will wait for responses from UPnP devices. By setting different values to the MX header, we can measure the overhead of UPnP messages in the network traffic.

We can also specify the Search Target. The example in Figure 6.1 shows that control point searches any UPnP devices without specifying the device type or service type. In our testing, we set the ST header to *urn:schemas-upnp-org:service:sip:1* because we are only interested in devices which provide SIP services.



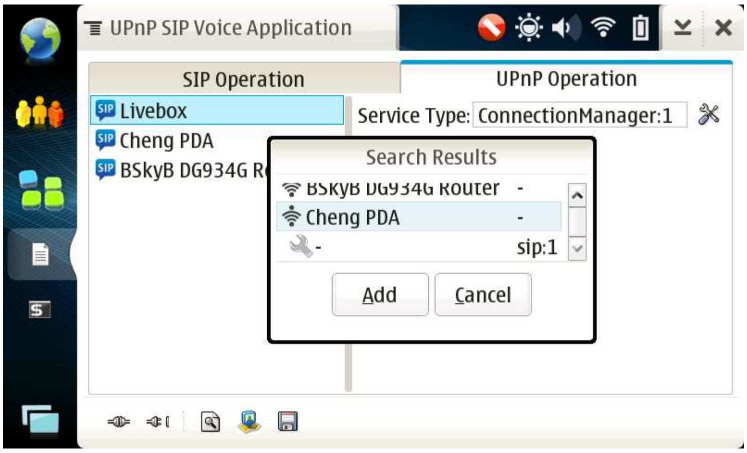
WGR614v9
Cheng PDA

Service Type: Layer3Forwarding:1

Search Target: ssdp:all

MX Value(in seconds): 10

Figure 6.1: Voice Application UPnP Search Window



Livebox
Cheng PDA
BSkyB DG934G R

Service Type: ConnectionManager:1

BSkyB DG934G Router	-
Cheng PDA	-
-	sip:1

Figure 6.2: UPnP Search Results Window

In Figure 6.2, the search results dialog presents found devices and services in a multi-column table. The top level is root devices, and lower level is services. The first column shows the friendly name of the found devices, and the second column shows the serviceID of the found services from particular device.

6.2 Results Analysis

6.2.1 Scenario 1

Figure 6.3 shows the comparison between UPnP traffic and SIP traffic. In this scenario, there are only 2 nodes set up in the Ad Hoc network. In Table 6.2, we can see that 98% of traffic occurs from SSDP packets. The reason is that UPnP is using multicast NOTIFY ssdp:alive message to announce its services. Our application is also designed to use multiple M-SEARCH message at a time when searching for other devices. It increases the chance to reach devices, but brings the burden of network traffic. The end-to-end delay in this scenario is mainly contributed by the UPnP discovery time.

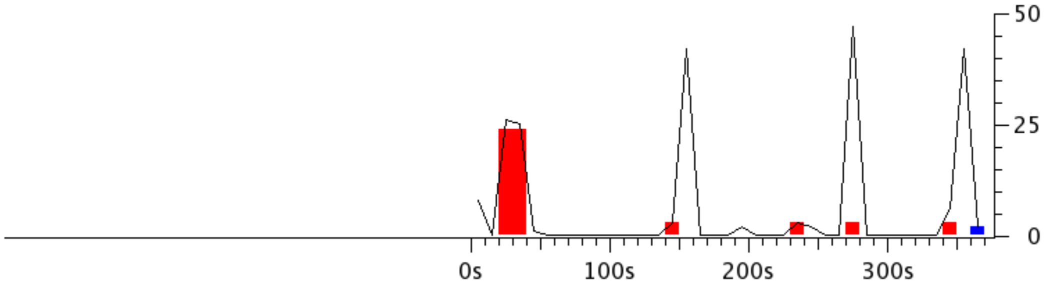


Figure 6.3: SSDP vs. SIP packets

We use formula 6.1 to calculate the total delay of establishing SIP call in Ad Hoc networks. D$_{UPnP}$ is the delay for discovery callee using UPnP, and D$_{SIP}$ is the delay for SIP signalling, and the result of total call setup delay (CSD) is shown in table 6.3.

$$CSD = \sum(D_{UPnP} + D_{SIP}) \tag{6.1}$$

Traffic	**SSDP/UPnP**	**SIP**	**Total**
Packets	60	3	63
Bytes	15112	1226	15338
Avg. pkt size(bytes)	235.199	613.000	247.387

Table 6.2: Scenario 1

	$\mathbf{D}_{UPnP}$	$\mathbf{D}_{SIP}$	$\mathbf{D}_{timeout}$	**Total**
Delay (sec)	10.387	0.089	0	10.476

Table 6.3: Call Setup Delay in Scenario 1

6.2.2 Scenario 2

In this scenario, we tested the SIP call initialed by an Ad Hoc node to a SIP node allocated in a public network. The Ad Hoc node of laptop is configured to use a STUN server (e.g. stunserver.org) to bind the port behind NAT. This can be done by setting the environment variable in Linux terminal . Figure 6.4 shows the STUN binding request and the STUN binding response message caught by Wireshark. It indicates the SIP invite messages traverse between the Ad Hoc network and the public network. In this scenario, the laptop is functioning as the gateway for the Ad Hoc network to public network.

To calculate the call setup delay for this scenario, we need to add the delay caused by the timeout value D$_{timeout}$. The formula is shown in 6.2. In this scenario, we have the 20.5 seconds delay in total shown in Table 6.5.

$$CSD = \sum(D_{UPnP} + D_{SIP} + D_{Timeout}) \tag{6.2}$$

Traffic	**SSDP/UPnP**	**SIP**	**Total**
Packets	60	3	63
Bytes	15112	1226	15338
Avg. pkt size(bytes)	235.199	613.000	247.387

Table 6.4: Scenario 2

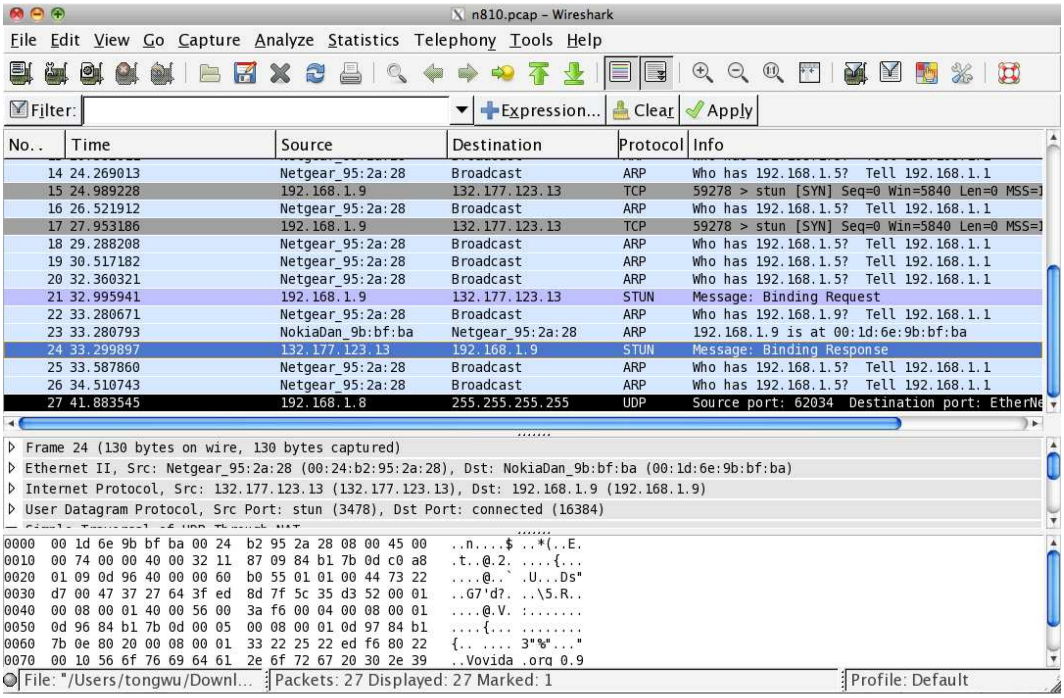
	$\mathbf{D}_{UPnP}$	$\mathbf{D}_{SIP}$	$\mathbf{D}_{timeout}$	**Total**
Delay(sec)	11.072	8.887	10	29.959

Table 6.5: Call Setup Delay in Scenario 2

From the results, we notice the end-to-end delay from SIP signals is increased because the STUN server is involved in the signalling path.

6.2.3 Scenario 3

The last scenario is making call from the public network to our private Ad Hoc network. The callee allocated in the Ad Hoc network is required to register


n810.pcap – Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression...
Clear
Apply

```
No. .  Time          Source              Destination        Protocol  Info
   14 24.269013      Netgear_95:2a:28    Broadcast          ARP       Who has 192.168.1.5?  Tell 192.168.1.1
   15 24.989228      192.168.1.9         132.177.123.13     TCP       59278 > stun [SYN] Seq=0 Win=5840 Len=0 MSS=1
   16 26.521912      Netgear_95:2a:28    Broadcast          ARP       Who has 192.168.1.5?  Tell 192.168.1.1
   17 27.953186      192.168.1.9         132.177.123.13     TCP       59278 > stun [SYN] Seq=0 Win=5840 Len=0 MSS=1
   18 29.288208      Netgear_95:2a:28    Broadcast          ARP       Who has 192.168.1.5?  Tell 192.168.1.1
   19 30.517182      Netgear_95:2a:28    Broadcast          ARP       Who has 192.168.1.5?  Tell 192.168.1.1
   20 32.360321      Netgear_95:2a:28    Broadcast          ARP       Who has 192.168.1.5?  Tell 192.168.1.1
   21 32.995941      192.168.1.9         132.177.123.13     STUN      Message: Binding Request
   22 33.280671      Netgear_95:2a:28    Broadcast          ARP       Who has 192.168.1.9?  Tell 192.168.1.1
   23 33.280793      NokiaDan_9b:bf:ba   Netgear_95:2a:28   ARP       192.168.1.9 is at 00:1d:6e:9b:bf:ba
   24 33.299897      132.177.123.13      192.168.1.9        STUN      Message: Binding Response
   25 33.587860      Netgear_95:2a:28    Broadcast          ARP       Who has 192.168.1.5?  Tell 192.168.1.1
   26 34.510743      Netgear_95:2a:28    Broadcast          ARP       Who has 192.168.1.5?  Tell 192.168.1.1
   27 41.883545      192.168.1.8         255.255.255.255    UDP       Source port: 62034  Destination port: EtherNe

▷ Frame 24 (130 bytes on wire, 130 bytes captured)
▷ Ethernet II, Src: Netgear_95:2a:28 (00:24:b2:95:2a:28), Dst: NokiaDan_9b:bf:ba (00:1d:6e:9b:bf:ba)
▷ Internet Protocol, Src: 132.177.123.13 (132.177.123.13), Dst: 192.168.1.9 (192.168.1.9)
▷ User Datagram Protocol, Src Port: stun (3478), Dst Port: connected (16384)

0000  00 1d 6e 9b bf ba 00 24  b2 95 2a 28 08 00 45 00   ..n....$ ..*(..E.
0010  00 74 00 00 40 00 32 11  87 09 84 b1 7b 0d c0 a8   .t..@.2. ....{...
0020  01 09 0d 96 40 00 00 60  b0 55 01 01 00 44 73 22   ....@..` .U...Ds"
0030  d7 00 47 37 27 64 3f ed  8d 7f 5c 35 d3 52 00 01   ..G7'd?. ..\5.R..
0040  00 08 00 01 40 00 56 00  3a f6 00 04 00 08 00 01   ....@.V. :.......
0050  0d 96 84 b1 7b 0d 00 05  00 08 00 01 0d 97 84 b1   ....{... ........
0060  7b 0e 80 20 00 08 00 01  33 22 25 22 ed f6 80 22   {.. .... 3"%"..."
0070  00 10 56 6f 76 69 64 61  2e 6f 72 67 20 30 2e 39   ..Vovida .org 0.9

File: "/Users/tongwu/Downl...   Packets: 27 Displayed: 27 Marked: 1   Profile: Default
```

Figure 6.4: STUN binding message

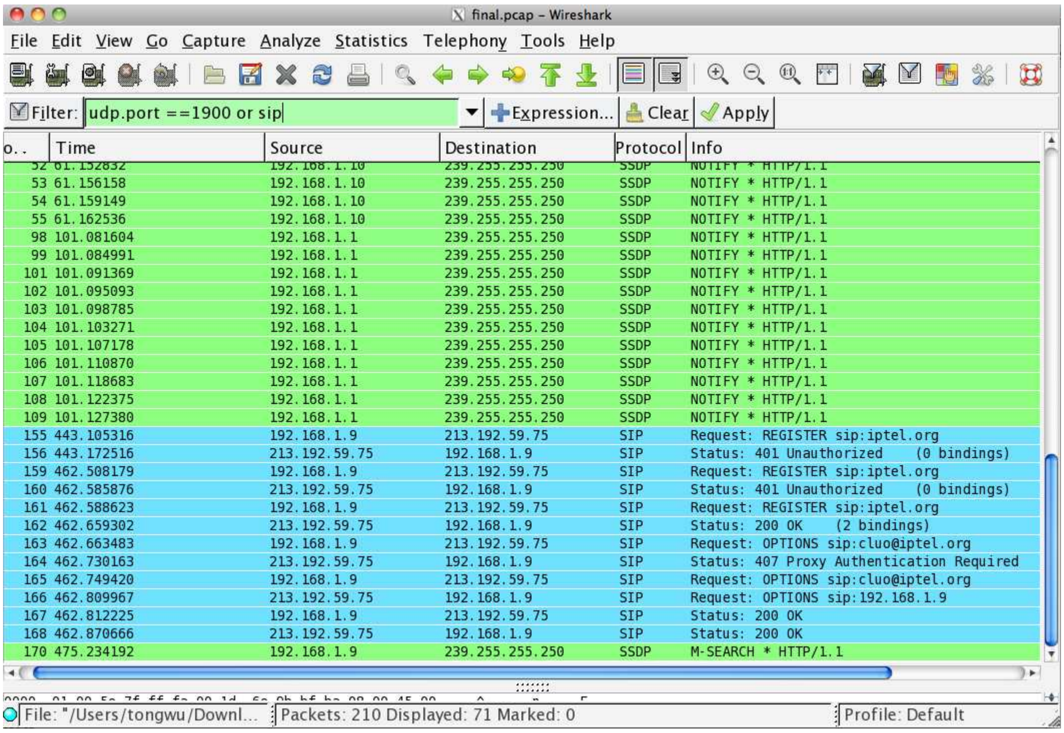
itself before the call. From the packets shown in Figure 6.5, we can see the REGISTER messages along with UPnP messages. The results in Table 6.6 indicates the end-to-end delay from SIP packets have increased. Also, the number of packets are increasing. In this case we use the same formula 6.1 to calculate the end-to-end delay.

Traffic	SSDP/UPnP	SIP	Total
Packets	59	12	71
Bytes	16274	6902	23176
Avg. pkt size(bytes)	275.830	575.167	326.423

Table 6.6: Scenario 3

	D_{UPnP}	D_{SIP}	$D_{timeout}$	Total
Delay(sec)	13.869	19.765	0	33.634

Table 6.7: Call Setup Delay in Scenario 3



Filter: udp.port ==1900 or sip

```
o..  Time          Source           Destination      Protocol Info
  52 61.152832     192.168.1.10     239.255.255.250  SSDP     NOTIFY * HTTP/1.1
  53 61.156158     192.168.1.10     239.255.255.250  SSDP     NOTIFY * HTTP/1.1
  54 61.159149     192.168.1.10     239.255.255.250  SSDP     NOTIFY * HTTP/1.1
  55 61.162536     192.168.1.10     239.255.255.250  SSDP     NOTIFY * HTTP/1.1
  98 101.081604    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
  99 101.084991    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 101 101.091369    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 102 101.095093    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 103 101.098785    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 104 101.103271    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 105 101.107178    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 106 101.110870    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 107 101.118683    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 108 101.122375    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 109 101.127380    192.168.1.1      239.255.255.250  SSDP     NOTIFY * HTTP/1.1
 155 443.105316    192.168.1.9      213.192.59.75    SIP      Request: REGISTER sip:iptel.org
 156 443.172516    213.192.59.75    192.168.1.9      SIP      Status: 401 Unauthorized    (0 bindings)
 159 462.508179    192.168.1.9      213.192.59.75    SIP      Request: REGISTER sip:iptel.org
 160 462.585876    213.192.59.75    192.168.1.9      SIP      Status: 401 Unauthorized    (0 bindings)
 161 462.588623    192.168.1.9      213.192.59.75    SIP      Request: REGISTER sip:iptel.org
 162 462.659302    213.192.59.75    192.168.1.9      SIP      Status: 200 OK    (2 bindings)
 163 462.663483    192.168.1.9      213.192.59.75    SIP      Request: OPTIONS sip:cluo@iptel.org
 164 462.730163    213.192.59.75    192.168.1.9      SIP      Status: 407 Proxy Authentication Required
 165 462.749420    192.168.1.9      213.192.59.75    SIP      Request: OPTIONS sip:cluo@iptel.org
 166 462.809967    213.192.59.75    192.168.1.9      SIP      Request: OPTIONS sip:192.168.1.9
 167 462.812225    192.168.1.9      213.192.59.75    SIP      Status: 200 OK
 168 462.870666    213.192.59.75    192.168.1.9      SIP      Status: 200 OK
 170 475.234192    192.168.1.9      239.255.255.250  SSDP     M-SEARCH * HTTP/1.1
```

File: "/Users/tongwu/Downl... Packets: 210 Displayed: 71 Marked: 0 Profile: Default

Figure 6.5: SIP REGISTER message

Chapter 7

Conclusion and Future work

In this thesis, we have designed and implemented a modularized system for the Ad hoc VoIP application. We use UPnP as both the peer discovery protocol and the service discovery protocol. To be able to achieve better interoperability, our design does not require any modifications on SIP messages nor dependencies on the underlaying routing protocols. We also defined different scenarios of making SIP calls using our application. And we explained the functionality of each modules in different scenarios. Finally, we tested our implementations on three Nokia Internet Tablets.

From description in Chapter 6, we found there were several facts that have an effect on the performance of UPnP service discovery performance, such as *MX value*, *number of repeat SSDP announcements*, and *number of devices* in the network.

The most important fact is the number of devices because the devices are constantly multicasting ssdp:alive message to the network, and it dramatically increases the overhead of network traffic. But in small scope networks, such as home or office networks, the traffic load caused by UPnP is not noticeable compared with the RTP traffic of media session. Regarding the end-to-end delay of establishing SIP call, UPnP service discovery and peer discovery is steady and does not increase dramatically. But the delay contributed by SIP signalling varies in different situations. From this observation, we can conclude that UPnP is suitable for peer-to-peer application in small scope home networks or office networks.

As we mentioned in Chapter 1, we have not considered security issues when we designed our application. It is highly recommended to encrypt conversations between mobile nodes. Due to the limitation of resources, we cannot test this application on a large number of real devices. Possible future work can be performed on these two areas.

Bibliography

[1] TeleGeography Research. Telegeography report executive summary, 2008. http://www.telegeography.com/products/tg/telegeography_samples.zip.

[2] Morgan Stanley Research. Internet trends, 2010. http://www.morganstanley.com/institutional/ techresearch/pdfs/Internet_Trends_041210.pdf.

[3] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[4] D.A. Bryan, B.B. Lowekamp, and C. Jennings. Sosimple: A serverless, standards-based, p2p sip communication system. *Advanced Architectures and Algorithms for Internet Delivery and Applications, 2005. AAA-IDEA 2005. First International Workshop on*, pages 42–49, June 2005.

[5] Chang Lin-huang, Chuang Ping-da, and Chen Yu-Jen. An ad-hoc voip system implementation using upnp. In *International Computer Symposium*, December 2004.

[6] S. Leggio, J. Manner, A. Hulkkonen, and K. Raatikainen. Session initiation protocol deployment in ad-hoc networks: a decentralized approach. In *2nd International Workshop on Wireless Ad-hoc Networks (IWWAN)*, May 2005.

[7] Kundan Singh and Henning Schulzrinne. Peer-to-peer internet telephony using sip. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 63–68, New York, NY, USA, 2005. ACM.

[8] Y. Charlie Hu, Saumitra M. Das, and Himabindu Pucha. Peer-to-peer overlay abstractions in manets. CRC Press, 2005.

[9] Patrick Stuedi, Marcel Bihr, Alain Remund, and Gustavo Alonso. Siphoc: Efficient sip middleware for ad hoc networks, 2007. available as. Technical report, In Proceedings of the 8th ACM/IFIP/USENIX International Middleware Conference, 2007.

[10] R. Ramanathan and J. Redi. A brief overview of ad hoc networks: challenges and directions. *Communications Magazine, IEEE*, 40(5):20–22, May 2002.

[11] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728 (Experimental), February 2007.

[12] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[13] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[14] C.K. K Toh. *Ad Hoc Wireless Networks: Protocols and Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[15] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[16] E. Royer and C. Perkins. Multicast Ad hoc on-Demand Distance Vector (MAODV) Routing. IETF Internet-Draft draft-ietf-manet-maodv-00.txt, July 2000.

[17] Sung-Ju Lee, William Su, and Mario Gerla. On-Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks. IETF Internet-Draft draft-ietf-manet-odmrp-02.txt, January 2000.

[18] Charles E. Perkins and Charles Perkins. *Ad Hoc Networking*. Addison-Wesley Professional, December 2000.

[19] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393.

[20] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 2234 (Proposed Standard), November 1997. Obsoleted by RFC 4234.

[21] M. Garcia-Martin, C. Bormann, J. Ott, R. Price, and A. B. Roach. The Session Initiation Protocol (SIP) and Session Description Protocol (SDP) Static Dictionary for Signaling Compression (SigComp). RFC 3485 (Proposed Standard), February 2003. Updated by RFC 4896.

[22] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327 (Proposed Standard), April 1998. Obsoleted by RFC 4566, updated by RFC 3266.

[23] Jörg Ott. Application protocol design considerations for a mobile internet. In *MobiArch '06: Proceedings of first ACM/IEEE international workshop on Mobility in the evolving internet architecture*, pages 75–80, New York, NY, USA, 2006. ACM.

[24] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608 (Proposed Standard), June 1999. Updated by RFC 3224.

[25] Sun Mircosystem. Jini architecture specification. http://www.jini.org/wiki/jini_architecture_specification#key_concepts.

[26] Universal Plug and Play Forum. Upnp $^{\mathrm{TM}}$device architecture, October 2008.

[27] Bluetooth SIG. Specification of the bluetooth system. http://www.bluetooth.com/bluetooth/technology/building/ specifications/default.htm.

[28] Universal Plug and Play Forum. Understanding universal plug and play. http://www.upnp.org/download/UPnP_UnderstandingUPNP.doc, 2000.

[29] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927 (Proposed Standard), May 2005.

[30] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826 (Standard), November 1982. Updated by RFC 5227.

[31] D. Meyer. Administratively Scoped IP Multicast. RFC 2365 (Best Current Practice), July 1998.

[32] Nokia maemo platform. http://www.maemo.org.

[33] Scratchbox. http://www.scratchbox.org.

[34] GNOME ORG. Hildon project homepage. http://maemo.org.

Appendix A

Key APIs

Listing A.1: API Sending HTTP GET

```
 // This method sends HTTP GET message to the given IP
 // address, port number and URI from the SSDP response
 // message received from UPnP network

char *SendHttpGet(char *ipaddr, int port, char * uri, PtrToXMLParserFunc callback_func)
{
CgHttpRequest *httpReq;
CgHttpResponse *httpRes;
char *content;
long contentlength;

httpReq = cg_http_request_new();
cg_http_request_setmethod(httpReq, CG_HTTP_GET);
cg_http_request_seturi(httpReq, uri);
cg_http_request_setcontentlength(httpReq, 0);
httpRes = cg_http_request_post(httpReq, ipaddr, port);

 // send HTTP GET failed
if(!cg_http_response_issuccessful(httpRes))
{
        cg_http_request_delete(httpReq);
        return NULL;
}
content = cg_http_response_getcontent(httpRes);
contentlength = cg_http_response_getcontentlength(httpRes);
return callback_func(content, contentlength);
}
```

Listing A.2: SSDP Response Listener

```
// SSDP Notify Listener
void SSDPNotifyListner(CgUpnpSSDPPacket *ssdpPkt)
{
  if (cg_upnp_ssdp_packet_isdiscover(ssdpPkt) == TRUE)
{
  cg_upnp_ssdp_packet_print(ssdpPkt);
}
else if (cg_upnp_ssdp_packet_isalive(ssdpPkt) == TRUE)
{
if(!strcmp(cg_upnp_ssdp_packet_getnt(ssdpPkt),
        "urn:schemas-upnp-org:service:sip:1"))
{
 char *ip = NULL;
 int port = 0;
 char *uri = NULL;
 char *sip_aor = NULL;
 char *service_id = NULL;

 PtrToXMLParserFunc callback_func =
    GetServiceIdFromXML_N;
 ip=get_ip_from_url_N( cg_upnp_ssdp_packet_getlocation(
    ssdpPkt));
 port=get_port_from_url(cg_upnp_ssdp_packet_getlocation(
    ssdpPkt));
 uri=get_uri_from_url_N(cg_upnp_ssdp_packet_getlocation(
    ssdpPkt));
 service_id=SendHttpGet(ip, port, uri, callback_func);

  sip_aor = find_token(service_id, ":", "serviceId");
  main_view->data->routing_table = routingtable_update(
     main_view->data->routing_table, ip, sip_aor, ONLINE
     );
 }
 else
 {
    printf("xxxxxxx␣non-sip␣service!\n");
 }
}
else if (cg_upnp_ssdp_packet_isbyebye(ssdpPkt) == TRUE)
{
  char *ip = NULL;
  int port = 0;
  if(strcmp(cg_upnp_ssdp_packet_getnt(ssdpPkt),
                "urn:schemas-upnp-org:service:sip:1") ==
                    0)
  {
    ip =  cg_upnp_ssdp_packet_getremoteaddress(ssdpPkt);
    port = cg_upnp_ssdp_packet_getremoteport(ssdpPkt);
    main_view->data->routing_table = routingtable_update
       (main_view->data->routing_table, ip, NULL,
       OFFLINE);
    }
   else
   {
     printf("xxxxxxx␣non-sip␣service!\n");
   }
  }
}
```