Signaling for Multiparty Sessions in Peer-to-Peer Ad hoc Networks

Chunyan Fu

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

August 2004

Your file Votre référence
ISBN: 0-612-94698-3
Our file Notre référence
ISBN: 0-612-94698-3

ABSTRACT

Signaling for Multiparty Sessions in Peer-to-Peer Ad hoc Networks

Chunyan Fu

Multiparty sessions are the basis of several important applications. Audio/video conferencing, distance learning and gaming are some of the examples. So far, very little attention has been devoted to the applications for ad hoc networks. In contrast to conventional networks, ad hoc networks are formed by a set of devices that communicate without using a pre-existing network infrastructure. Multiparty session control faces new challenges in ad hoc networks: peer-to-peer communication, limited resources and transient network connectivity. This thesis addresses the most critical component of multiparty sessions in peer-to-peer ad hoc networks: signaling. Signaling requirements are derived and existing signaling protocols such as SIP, H.323 and ICEBERG signaling are analyzed. By discussing and comparing alternative signaling architectures, we propose a novel signaling scheme based on clustering architecture. The signaling scheme is implemented as an extension to SIP. A prototype is built to evaluate the feasibility and performance of the signaling scheme.

ACKNOWLEDGEMENTS

This thesis is the result of joint efforts of my supervisors: Dr. Rachida Dssouli and Dr. Roch Glitho. Without their patient guidance and thought-provoking ideas, I would not have come this far.

Thanks go to all the teammates working in the TSE research lab for their non-reserved help whenever I needed.

I would like to acknowledge the funding from the NSERC, Ericsson and SRT Concordia University.

I would also like to thank my parents and my brother. They encouraged me so much in my work.

Last but not least, to my husband Xiaoniu, he is always unconditionally supporting me.

Table of Content

List of Figures

List of Tables

CHAPTER 1 INTRODUCTION

The main goal of this thesis is to provide an application-layer signaling solution for multiparty sessions in peer-to-peer ad hoc networks. In this introduction, we first present the motivation behind our research. Then we describe the issues that will be tackled. Afterwards, the specific objectives assigned to the thesis are presented. The last section presents how the thesis is structured.

1.1 Motivation

Today's ad hoc networks aim to meet a number of requirements that conventional, infrastructure-based networks cannot satisfy. One requirement is investment reduction. Users may be so sparse or dense that the appropriate fixed infrastructure would not be economical. For example, large-scaled cooperative networks can be set up in communities without relying on any network service providers. Another requirement is to deal with some situations in which fixed infrastructure cannot be available, such as disaster recovery, battlefields. Yet another requirement is to complement the deficiencies of the infrastructure-based network. For example, ad hoc networks can be used to extend base station's coverage.

Ad hoc network has also been regarded as an important part of the 4G Wireless System. The goal of 4G is to provide a seamless and ubiquitous communication environment so that users can accomplish their tasks and access information at any time, anywhere, and from any device [1]. Ad hoc networks in 4G aim at meeting this goal.

The attractiveness of ad hoc networks makes it an active research area. Standards and technologies are emerging nowadays. However, today's research is mainly focused on the lower-layer, such as physical, data link and network layers [16,17,18,2,3,4]. Only few activities are related to middleware and application layers such as, group communication [5], file-sharing application [6] and SIP based framework [49]. No complete and effective solution has been proposed for multiparty session control applications.

Multiparty sessions are the basis of several important applications such as chatting, gaming, multimedia conferencing and distance learning. There are two key technologies for multiparty session management: signaling and media handling. Signaling is responsible for establishment, modification and termination of sessions. Media handling aims to provide real-time media transmission for those sessions. Standards of signaling and media handling for infrastructure-based networks have already been developed. Examples are SIP [7] from the IETF, H.323 [8] from the ITU-T and RTP/RTCP [9,10]. However, these protocols are unsuitable for ad hoc network environment (presented in chapter 3). Until now, there is no signaling or media handling protocols especially designed for ad hoc networks.

In fact, multiparty session control in ad hoc networks is challenging. The communication between participants will be based on peer-to-peer due to the lack of centralized and fixed controlling points. Besides this, participants may join or leave at any time. The signaling information needs to be propagated and the media needs to be handled in a distributed manner. Furthermore, resources need to be utilized optimally because the heterogeneous devices may take part in the ad hoc network. This implies that some devices may use consumptive power and low capacity processors while others may have more processing resources and stable powers.

1.2 Issues Tackled

In this thesis, we will tackle the critical issue: signaling for multiparty session. There are many research topics in the area of signaling for multiparty session in peer-to-peer ad hoc networks. Examples are dynamic session management, resource negotiation, scalability, node availability, security, billing, etc. In this thesis, we will focus only on some of these issues: the establishment, modification and termination of dynamic multiparty sessions, the optimal usage of resources and the scalability.

Although our signaling system will not provide conference control services such as floor control and voting, we did our research for multiparty session control in a specific conference mode. Conferences can be classified as pre-arranged and ad hoc. The former is a conference in which time, length, location and participants may all be scheduled in advance. The latter, on the contrary, is formed spontaneously by relatively random needs and by arbitrary participants. This kind of conference can be expanded from two-party to multiparty. To be more precise, two participants initiate a conference. Then, new parties can join in when participants who are already in the conference invite them. Participants are free to leave the conference at any time. Finally, the last two parties terminate the conference. We use ad hoc conference as multiparty session model because it fits quite well in peer-to-peer ad hoc network settings. It will be the basis of numerous applications such as gaming and public debates.

1.3 Goals

The goals of the thesis are as follows:

- ✧ Derive signaling requirements through the analysis of differences between conventional networks and peer-to-peer ad hoc networks.
- ✧ Find possible signaling mechanism that can satisfy the signaling requirements.
- ✧ Propose a clustering-based signaling scheme.
- ✧ Implement the scheme in the form of an extension to SIP.
- ✧ Build a prototype to evaluate the feasibility and performance of the scheme.

1.4 Thesis Organization

In chapter 2, we will present background knowledge on both ad hoc networks and peer-to-peer communication. In chapter 3, we will show the derived signaling requirements. We will also introduce today's signaling protocols SIP, H.323 and ICEBERG signaling. Furthermore, we will analyze their applicability to peer-to-peer ad hoc networks. In chapter 4, we will discuss several possible signaling architectures and make comparison among them. In chapter 5, we will propose a clustering-based signaling scheme that has the potential of meeting our signaling requirements. In chapter 6, we will describe how to implement the proposed scheme as an extension to SIP. In chapter 7, we will present a prototype that implements the user agent based on SIP extensions. JXTA [29,30] will be used as a peer-to-peer middleware and JAIN SIP Stack will be used to develop an extended SIP user agent. Preliminary results will then be analyzed. We will conclude our work in Chapter 8. Future work will also be presented at the end.

CHAPTER 2 BRIEF REVIEW OF PEER-TO-PEER AND AD HOC NETWORKS

In this chapter, we will first provide the background information on ad hoc networks and peer-to-peer communications. Then we will show how the two concepts are related and also how we use them in this thesis.

2.1 Ad Hoc Networks

An ad hoc network is a (possibly mobile) collection of communications devices (nodes) that wish to communicate, but have no fixed infrastructure available, and have no pre-determined organization of available links [11]. An important assumption for ad hoc network is the multi-hop. Each node in an ad hoc network may play roles of both routers and hosts. A node can forward packets on behalf of other nodes and run user applications. Devices in an ad hoc network can be heterogeneous, such as Personal Digital Assistants (PDAs), laptops, palm PCs and cell phones.

2.1.1 Evolution of Ad Hoc Networks

Early ad hoc networks called "packet radio networks" have primarily been used for military purpose to improve battlefield communications and survivability. The dynamic nature of military operations implies that it cannot rely on having access to a fixed pre-placed communication infrastructure in battlefield.

DARPA Packet Radio Network (PRNet in 1972) [12] can be considered as the first ad hoc network. It features a distributed architecture that consists of a network having broadcast radios with minimal central control. In addition, by using multi-hop store-and-forward routing techniques, the radio coverage limitation is removed. In the next twenty years, ad hoc networking technology had been subject to sporadic development. The typical scenario at that time was still setting up a communication network in a battlefield.

The term "ad hoc network" did not appear until the development of IEEE 802.11 in the 1990s, a standard for wireless local area networks (WLAN). The IEEE replaced the term "packet radio network" with "ad hoc network". With the adoption of the new name, the IEEE hoped to indicate an entirely new scenario other than a battlefield. References [12,13] give examples. They are disaster-relief networks, construction-site networks, inter-vehicle networks, home-environment networks, emergency networks, networks for visitors at airports, conference centers and shopping malls.

2.1.2 Taxonomy

We will introduce two primary classifications for ad hoc networks in this section. The first one focuses on the relationship between ad hoc networks and other networks. The second concerns the coverage area of ad hoc networks.

In relation to other networks, ad hoc networks can be classified as isolated ad hoc networks, integrated ad hoc networks and cellular ad hoc networks [14]. Isolated ad hoc network means that all nodes communicate with each other within the same network. It has no connection with any infrastructure-based communication network, such as the global Internet. Integrated ad hoc network means that there is a gateway to seamlessly integrate a small-scaled ad hoc network with some IP-based (mobile) access network. A cellular ad hoc network is self-organizing multi-hop network with multiple access points connected to a broadband network. The difference between cellular ad hoc networks and isolated ad hoc networks is that most of the traffic in cellular ad hoc networks is from or to access points. The difference between cellular ad hoc network and the integrated ad hoc network is that the former has multiple access points and it will be mainly used to extend the radio coverage of the base station.

According to the coverage area, ad hoc networks can be classified into four classes: Body, Personal, Local, and Wide Area Networks [15]. Figure 2-1 shows this classification. Wide area ad hoc networks are mobile multi-hop wireless networks. They are not likely to become available for some time due to a number of technical challenges (e.g., addressing, routing, location management, etc.). On smaller scales, ad hoc Body Area Networks (BAN), Personal Area Networks (PAN), and wireless LAN are beginning to appear on the market. Body Area Network is strongly correlated with wearable computers. The components of a wearable computer are distributed on the body (e.g., head-mounted displays, microphones, earphones, etc.), and BAN provides the connectivity among these devices. Personal area network is a network in the environment around the person. PAN connects mobile devices to other mobile and stationary devices. The typical communication range of a PAN is 10 meters. Wireless LAN has communication range of 100 – 500 meters so it will be the solution for home and office automation.

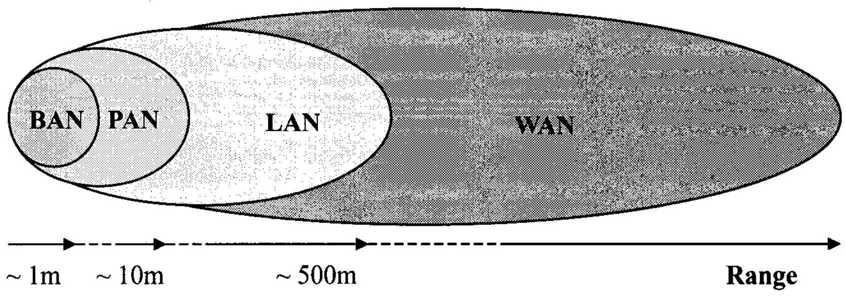


Figure 2-1 Ad Hoc Networks Taxonomy

2.1.3 Main Technologies and Standards

For physical and data link layers, there are two emerging personal or body area network standards. One is IEEE 802.15.3 [16], which provides high data-rate personal area ad hoc networks. The other is Bluetooth [17], which intends to serve as universal low-cost air interfaces that will replace the plethora of proprietary interconnecting cables between personal devices.

There are three main wireless LAN (WLAN) technologies. The most mature one is the IEEE 802.11 [18], which is commercially successful and widely used in enterprises and educational organizations. There are two possible settings in 802.11. One is infrastructure-based setting that defines Access Point (AP). Nodes in a cell of 802.11

networks communicate with a centralized and fixed AP. The AP is normally connected to the wired network, thus providing Internet access. Another is the ad hoc setting, which is a peer-to-peer network formed by a set of nodes. They dynamically configure to set up a temporary network. Digital Enhanced Cordless Telecommunications (DECT), another standard of WLAN, is a flexible digital radio access standard for cordless communications in residential, corporate and public environments. It was introduced by ETSI for cordless telecommunications, and mainly focuses on the telephony and the ISDN access [19]. HyperLAN2, supported in Europe and Japan, is a flexible radio LAN standard designed to provide high speed access to various networks including 3G mobile core networks, ATM networks and IP based networks, and also for private use as a WLAN system.

In the network layer, IETF MANET (Mobile Ad hoc Networking) working group has defined three IP routing protocols. They are Ad hoc On-Demand Distance Vector (AODV) routing [2], Optimized Link State Routing Protocol (OLSR) [3] and Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) [4]. OLSR and TBRPF are proactive link state routing protocols, while AODV is a reactive (table-based) on demand routing protocol.

2.1.4 Overall Architecture

Ad hoc networking is a multi-layered issue. Other than the physical layer, data link layer and network layer standards, there are also researches on middleware [5], application [6,49], quality of service [20], security [21] and energy conservation [22].

Figure 2-2 shows a layered architecture of ad hoc networks.

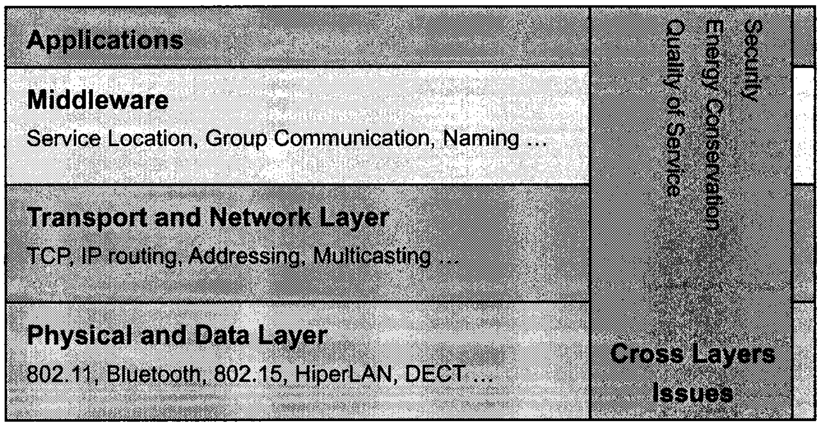

Applications
Middleware
Service Location, Group Communication, Naming ...
Transport and Network Layer
TCP, IP routing, Addressing, Multicasting ...
Physical and Data Layer
802.11, Bluetooth, 802.15, HiperLAN, DECT ...
Security
Energy Conservation
Quality of Service
Cross Layers Issues

Figure 2-2 An Overall Ad Hoc Network Architecture

2.2 Peer-to-peer Communication

Peer-to-Peer (P2P) is a way of structuring distributed applications such that the individual nodes have symmetric roles [23]. In this definition, "symmetric" is the most important idea in contrast to the asymmetric roles of client and server in Client/Server mode. Client/Server mode is a one-to-many relationship. One server deals with and responds to requests from many clients. However, P2P applications exhibits a symmetric one-to-one relationship, and a node may act as both a client and a server.

2.2.1 Evolution of P2P in Internet

Peer-to-peer is not a new idea. The precursor of the Internet -- Advanced Research Project Agency Network (ARPANET), started by 1966 [24], may be considered as the first P2P network [25] because ARPANET addressed the need to decentralize the network and to ensure that no single point of failure is present. Until the ARPANET was replaced by National Science Foundation Network (NSFNET) in 1990 [24], the Internet remained primarily a P2P network. The commercialization of Internet in 1994 brought with it the dominance of client/server networking at the Internet's application layer. Throughout the rest of the 1990s, successful deployment of client/server attracted most of the attention of the people. That is why they failed to notice what Napster [25,26] did in 2000. Since then, several P2P applications have been launched to date. The most famous ones are the Gnutella [25,27] and FreeNet [25,28]. JXTA [29,30] is also an active technology to provide a P2P framework. Today, P2P has become an increasingly hot topic for Internet computing technologies due to the attraction of improving the utilization of Internet resources and being able to enhance system reliability and fault tolerance.

2.2.2 Using P2P in Ad Hoc Networks

Since most of the Internet technologies today are still based on client/server, there exist two classes of P2P networks in the Internet domain: "pure" P2P networks and "hybrid" P2P networks [31]. When P2P is used in ad hoc network environment, it becomes naturally "pure" due to the similarities between P2P and ad hoc networking: both have decentralized architectures, transient connectivity and heterogeneity of resources.

Actually, applications in *ad hoc networks* rely on *P2P* communication. Some research works have put them together [6,32,33]; others have made comparisons between them [34,35]. In this thesis, we use the term *peer-to-peer ad hoc network* to represent the isolated ad hoc network in which peer-to-peer communication mode is used.

CHAPTER 3 SIGNALING REQUIREMENTS AND TODAY'S SIGNALING PROTOCOLS

The characteristics of peer-to-peer ad hoc networks are very different from that of conventional networks. Traditional networks are planned, designed and users are added or removed by the explicit actions of system administration. Signaling protocols in traditional networks like SIP [7], H.323 [8] and ICEBERG Signaling [36] assume that there exist fixed and infrastructure-based networks. They also assume that there may be centralized control points that can be pre-configured and pre-deployed. This is why all these signaling protocols have the definition of centralized entities although the entities may not be mandatory. In this chapter, we first derive the specific signaling requirements for peer-to-peer ad hoc networks. Then we introduce the three signaling protocols that are the state of the art. Finally, we analyze these protocols and see why they are not suitable for peer-to-peer ad hoc networks.

3.1 Signaling Requirements

A peer-to-peer ad hoc network is a spontaneous network without any pre-existing infrastructure or centralized administration point. Nodes in the network can communicate with each other directly or via relay nodes. Connections between nodes are transient. There are differences in the capabilities of processors, memory, and power resource of the nodes. A signaling protocol should adapt to these characteristics, and should also provide efficient session-control functionality. Again, our interest is in the ad hoc conference that we have presented in chapter 1. We derive five basic signaling protocol requirements.

1) There should be no centralized signaling entity, since there is no centralized entity in peer-to-peer ad hoc settings.
2) Signaling should be capable of maintaining dynamic sessions because participants may join or leave at anytime. This implies that information is properly propagated to all nodes (including nodes that join a conference at the same time).
3) The signaling system should be scalable because a multiparty session starting with a couple of participants may grow to thousands of participants depending on the application.
4) The signaling system should be light-weigh because nodes in ad hoc networks usually have limited processing capabilities.
5) Resources should be used optimally because some nodes may not have enough processing power and will have to rely on the processing power of other nodes.

Other requirements like the fault tolerance, billing and security are out of the scope of this thesis.

3.2 Today's Signaling Protocols

Signaling protocols such as H.323 [8] and SIP [7] are widely used in today's telecommunication world for controlling multimedia session over IP networks. ICEBERG signaling [36] provides valuable research input. They are our starting points in the research of signaling issues in peer-to-peer ad hoc networks.

3.2.1 H.323

The H.323 standard is a cornerstone technology specified by the ITU-T that defines the components, protocols and procedures that provide multimedia communication services over packet networks. It specifies four kinds of components: terminals, gateways, gatekeepers and multipoint control units. Instead of defining a signal protocol, the H.323 standard is an umbrella consisting of a set of protocols: audio CODECs, video CODECs, H.225 RAS (registration, admission, and status), H.225 call signaling and H.245 control signaling. Figure 3-1 gives an overview structure of H.323 networks. Next, we will introduce the components and signaling protocols in H.323. Then we look at the call procedures defined in H.323 and introduce how H.323 handles multiparty sessions.

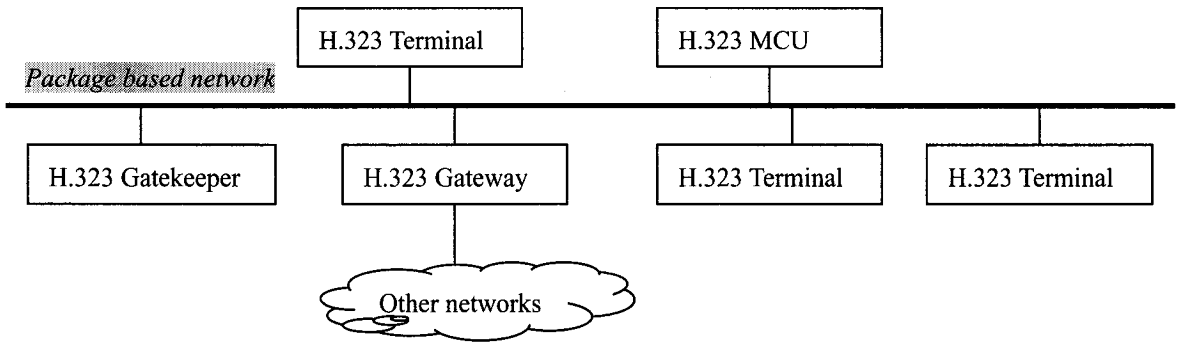


Figure 3-1 H.323 Network Structure

3.2.1.1 Components and Protocols

✧ *Terminals:* H.323 terminals are mandatory components in H.323 standards. An H.323 terminal can either be a personal computer (PC) or a stand-alone device running H.323 protocols and multimedia applications. It supports audio communications and can optionally support video or data communications.

- ✧ *Gatekeepers:* Although it is not a mandatory entity in the standard, a gatekeeper can be considered as the brain of the H.323 network. It is the focal point for all calls within the H.323 network. Gatekeepers provide important services such as addressing, authorization, authentication, bandwidth management, accounting, billing and charging. Gatekeepers may also provide call-routing services.
- ✧ *Gateways:* A gateway is not a required component in H.323 standard. An H.323 gateway provides connectivity between an H.323 network and a non-H.323 network. For example, a gateway can connect and provide communication between an H.323 terminal and public switched telephone network (PSTN). This connectivity of dissimilar networks is achieved by translating protocols for call setup and release, converting media formats between different networks and transferring information between the networks connected by the gateway.
- ✧ *Multipoint control units (MCUs):* MCUs provide support for conferences of more than two H.323 terminals. All terminals participating in the conference establish a connection with the MCU. The MCU manages conference resources, deals with media capability negotiation and handles the media stream.

- ✧ *H.225 RAS:* Registration, admission, and status (RAS) is the protocol between endpoints (terminals and gateways) and gatekeepers. The RAS is used to perform registration, admission control, bandwidth modification, and status collection procedures between endpoints and gatekeepers.

- ✧ *H.225 Call Signaling:* The H.225 call signaling is used to establish a connection between two H.323 endpoints. This is achieved by exchanging H.225 protocol messages on the call-signaling channel. The call-signaling channel is opened between two H.323 endpoints or between an endpoint and the gatekeeper.
- ✧ *H.245 Control Signaling:* H.245 control signaling is used to exchange end-to-end control messages. These control messages carry information related to the aspects of exchanging capabilities, opening and closing of logical channels, doing flow-control and exchanging general commands.

3.2.1.2 Call procedures

Call procedures are complex in H.323. They consist of six steps:

- ✧ Admission control: Before setting up a call, a terminal may exchange H.225 admission control messages with gatekeepers.
- ✧ Call setup: Call control messages defined in H.225.0 are used to set up a call. The setup procedure may go through a gatekeeper if terminals are registered.
- ✧ Initial communication capability exchange: Once both sides have exchanged call setup messages, the endpoints shall establish the H.245 control channel. Then they use the control channel to exchange media capability. After that, logical channels for media transmission should be opened.
- ✧ Establishment of audiovisual communication: Various information streams are transmitted using the logic channel.

- ✧ Call services: During the call procedure, a set of services can be provided such as bandwidth change, endpoint status check, multipoint conference and supplementary services (defined in ITU-T H.450.x).
- ✧ Call termination. Either endpoint or an intermediate call signaling entity may terminate a call. The procedures consist of terminating media transmission, closing logic channels and releasing signaling channels.

3.2.1.3 Multipoint conference

As mentioned previously, H.323 uses Multipoint Control Unit (MCU) to manage multiparty calls. There are three possible deployment approaches to implement multiparty conference using MCU. One is a centralized method. In this case, the MCU is responsible for mixing the media from all the participants and then sending the mixed media back. Another is a decentralized method in which the MCU multicasts the media to all the participants. The third way is a hybrid of the previous two. In this case, some terminals use centralized method and others use decentralized one. The MCU acts as a bridge.

3.2.1.4 Applicability to peer-to-peer ad hoc networks

H.323 is not suitable for peer-to-peer ad hoc networks because of the following reasons. Firstly, it is so complex that devices in ad hoc networks may not have enough processing capabilities to use it. Secondly, it needs the centralized MCU. From the implementation point of view, although a MCU can be combined with other components like terminals, a centralized session management is mandatory in H.323. Thirdly, scalability is limited by the processing capability of the MCU. Finally H.323 provides media capability negotiate mechanism, but it does not provide solutions for optimal usage of resources.

3.2.2 SIP

The Session Initiation Protocol is defined by IETF. It is an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. So far the IETF has defined a core SIP specification (RFC 3261 [7]) and a set of SIP extensions (examples are RFC 3515 [40], RFC 2976 [41] and RFC 3265 [42]). In this section, we will introduce the core SIP.

The core SIP defines a set of functionality. Other than session management, it also provides user location, user availability and user capability determination. There are two kinds of components defined in SIP: User Agents (UAs) and servers. Next we will introduce the components of SIP. Then we will look at the messages and call procedures. After that, we will discuss the possible conference models in SIP.

3.2.2.1 Components

- ✧ *User Agent (UA):* A user agent is a mandatory component in SIP. It represents an Internet end point. A UA contains a user agent client (UAC) and a user agent server (UAS). A UAC is capable of generating a request based on some external stimulus (the user clicking a button, or a signal on a PSTN line) and processing a response. A UAS is capable of receiving a request and generating a response based on user input, external stimulus, the results of a program execution or other mechanisms.

✧ *Proxy Servers*: Proxy servers are network hosts to which the user agent can send registrations, invitations and other requests. Proxy servers that facilitate the session establishment between end points may provide routing, locating, authenticating and other functionalities.

✧ Other servers, like *redirect server, registrar and location server*, may also appear in SIP settings to reduce the processing load on proxy servers and improving signaling path robustness. Servers are not mandatory entities in SIP.

3.2.2.2 Messages

Core SIP defines seven requests and six sets of responses. The messages are summarized in the table 3-1 and 3-2.

Table 3-1 SIP Requests

Method type	Usage
INVITE	Initiating a session
CANCEL	Canceling a pending request
BYE	Terminating a session
ACK	Acknowledging for the successful connection establishment
OPTION	Querying servers about their capability
REGISTER	Registering contact information of UA

Table 3-2 SIP Responses

Code Category	Code	Code Meaning	Code Category	Code	Code Meaning
Informational Or Provisional	100	Trying	Request Failure	413	Request Entity too large
	180	Ringing		414	Request-URI too long
	181	Call is being forwarded		415	Unsupported media type
	182	Queued	Request	420	Bad extension

Successful	200	OK
Redirection	300	Multiple choice
	301	Moved permanently
	302	Moved Temporarily
	305	Use Proxy
	380	Alternative Service
Request Failure	400	Bad request
	401	Unauthorized
	402	Payment required
	403	Forbidden
	404	Not found
	405	Method not allowed
	406	Not acceptable
	407	Proxy Authentication required
	408	Request timeout
	409	Conflict
	410	Gone
	411	Length required
Failure	480	Temporarily Unavailable
	481	Call leg does not exist
	482	Loop detected
	483	Too many hops
	484	Address incomplete
	485	Ambiguous
	486	Busy here
Server Failure	500	Server Internal error
	501	Not implemented
	502	Bad gateway
	503	Service Unavailable
	504	Gateway Timeout
	505	Version not supported
Global Failures	600	Busy everywhere
	603	Decline
	604	Does not exist anywhere
	606	Not acceptable

3.2.2.3 Call Procedures

SIP is lightweight. Its call procedures are simpler than that of H.323. SIP uses INVITE message to initiate a session. The session then can be modified by using a re-INVITE and terminated by using a BYE message. The initiation procedure is a three-way handshake while other procedures are HTTP-like requests and replies. Media capability can be exchanged through using SDP (Session Description Protocol).

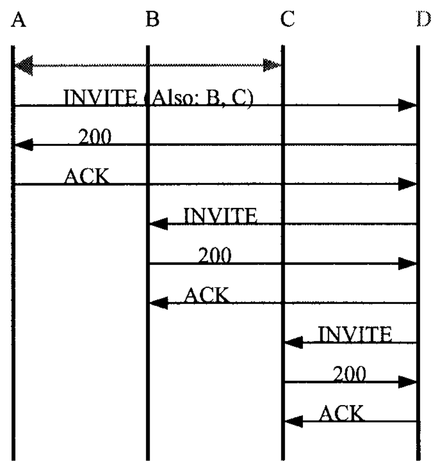
3.2.2.4 Multiparty Session Modes

SIP can be flexibly applied to different conference modes. Here we introduce some conference modes that are discussed in the Internet draft [43].

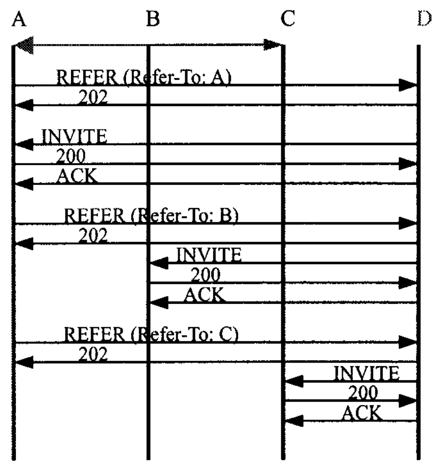
- ✧ Endpoint Mixing: This mode is applied to very small-scale conference because one endpoint (normally a caller) has the responsibility of mixing all conference media.
- ✧ Multicast conference: It relies on an IP multicast supported network.
- ✧ Dial-in: A centralized conference server is needed. Endpoints invite the conference server to join a conference. The conference server performs the media mixing.
- ✧ Dial-out: It also needs a conference server, but the server invites endpoints to join the conference.
- ✧ Ad hoc centralized: Two parties first set up a conference. Other parties join the conference through a centralized conference server.
- ✧ Centralized signaling, distributed media: This mode uses a centralized signaling server, while media are mixed at each endpoint.
- ✧ Full mesh: Sessions are managed in fully distributed manner. No servers or any other components control the conference.

3.2.2.5 Applicability to Peer-to-Peer Ad Hoc Networks

In the conference modes that we have introduced, full mesh is the closest to meeting the requirements of peer-to-peer ad hoc networks. There are two ways to organize a full mesh conference in SIP. One way is to use a new header field named “Also” [44]. The other way is to use the extended request “REFER” [40]. Sequence examples are shown in figure 3-2.



(a): The usage of "Also"



(b)The usage of "REFER"

Figure 3-2 SIP Full Mesh Conferences
A, B, C are in a conference. A then invites D to join the conference

However, the full-meshed mode has two main defects. One is that it is not scalable. Connections between endpoints will increase exponentially with the enlargement of the scale of a conference. The other defect is that the full-meshed mode cannot handle the situation where more than two users join a conference at the same time. In this case, those users will have no knowledge of each other.

As a matter of fact, SIP in full mesh mode has been considered for using in ad hoc networks. Research work [49] provides a SIP based framework allowing the users to exchange instant messages and establish conferences. It utilizes a full-meshed architecture for signaling connections and a centralized architecture for information propagation. This hybrid solution can make up the second defect of the full-meshed mode. However, the work is not a complete solution and does not solve issues such as scalability and optimal use of resources.

3.2.3 ICEBERG

ICEBERG project [36] was developed by U.C. Berkley. The purpose of the project is to implement a communication network and a service infrastructure that provides internet-based integration of telephony and data services spanning diverse access networks. The ICEBERG network provides functionality such as multiple accesses, call services, registry, personal mobility, service mobility, admission control and billing. The functionality is implemented through Iceberg Access Point (IAP) and Iceberg Point of Presence (IPOP).

A signaling system is accordingly developed by the ICEBERG project group to adapt to the dynamic multi-device session management. Next we briefly look over the signaling system since it provides some interesting solutions for dynamic session control.

3.2.3.1 Signaling Phases

The session management is based on a set of soft states in ICEBERG signaling. There are two signaling phases defined: *call setup* phase and *call control* phase.

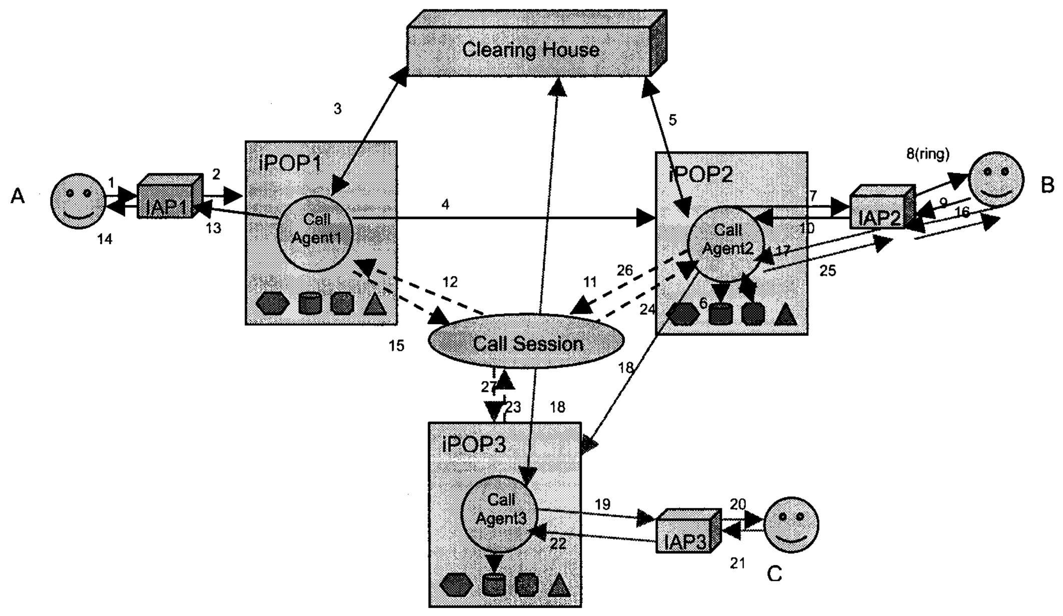
The *call setup phase* consists of *call session establishment* and *data path creation*. The *call session establishment* aims to set up a soft *call session* -- a control center that will maintain all sessions' soft states occurring in a multiparty call. The call session then creates *Call agents* in IPOPs. Each call agent serves one participant and manages its soft call states. The *data path creation* is responsible for creating the media transmission path.

Once a call has been set up, *call control* will be used to propagate state changes and modify the data path. The propagation procedures are performed in the following manner:

A serving call agent periodically receives the state report from the participant side, and forwards it to the call session. The call session, that maintains the states of all the participants in a table, then multicasts the table to all the call agents. It is important to know that this mechanism relies on the multicast in the network layer.

Figure 3-3 gives an example of call establishment procedures. The procedures follow the steps that we have described. In the figure, each participant accesses ICEBERG networks by an IAP. Admission control (performed by clearing house) may be needed before setting up calls. Other services such as preference-registry and name mapping may also be provided by iPOPs during the call establishment.

Figure 3-3 ICEBERG Call Procedures



3.2.3.2 Applicability to Peer-to-Peer Ad Hoc Networks

The ICEBERG cannot be applied to peer-to-peer ad hoc networks. The first reason is that ICEBERG relies on multicast that an ad hoc network might not provide. The second reason is that it has centralized session control unit: call session.

3.3 Summary

Summarizing three signaling protocols, we can see that none of the existing signaling systems meet all the requirements that we have derived. Table 3-3 presents key properties of three protocols. H.323 and ICEBERG has centralized features. SIP (full mesh) cannot achieve scalability.

Table 3-3 Today's Signaling Protocols

Requirements	H.323	SIP (full meshed version)	ICEBERG
Existence of centralized entity	Yes	No	Yes
Scalability level	Medium	Low	High
Level of dynamic session maintaining	None	Low (There are issues with the simultaneous join)	High
Light weight	No	Yes	Yes
Optimal usage of resource	No	No	No

CHAPTER 4 ALTERNATIVE SIGNALING ARCHITECTURES

When talking about signaling protocols, people normally focus their attentions on whether the protocols are efficient enough to provide the required functionality, whether they are easy to implement, and whether they are robust enough to handle unexpected situations. On the other hand, in the research on peer-to-peer ad hoc networks, it is primordial to draw our attention on how to create an effective signaling architecture. It is important because our signaling protocol will be implemented over a peer-to-peer communication fabric. The architecture will be used to aid in solving signaling problems such as information propagation and scalability.

The purpose of this chapter is to find a signaling architecture that is suitable for peer-to-peer ad hoc networks. We will examine several alternative architectures and analyze their pros and cons.

4.1 Centralization and Full Mesh

Centralization is a kind of architecture where there is a centralized control node. All other nodes are directly connected to the control node. In the architecture, the control node may be a conference server such as the server in dial-in and dial-out SIP conference, or a multipoint controller such as MCU in H.323. The advantage of this structure is its manageability. The control node that manages the conference may also provide admission control, billing and security.

Obviously, the centralized architecture is not applicable in peer-to-peer ad hoc networks since there are no fixed points available. It is also hard to find a node that can cope with the heavy load of the control point.

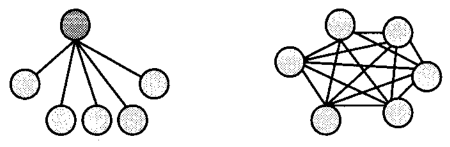


Figure 4-1: Centralization and Full mesh

In contrast, full mesh is a fully distributed architecture. There is no control point. Nodes have the same responsibilities and they are directly connected to each other. Full mesh directly reflects the peer-to-peer communication idea. It is applicable to some small-scaled cases where there is no control node available.

Although some existing documents (for example, the expired IETF Internet Draft [44] and the SIP based framework [49]) have discussed how to implement a conference by using full mesh architecture, we see very little of it in reality because a large number of signaling connections takes too much network and endpoint resources.

Figure 4-1 shows the architecture of centralization and full mesh.

4.2 Tree

The typical tree structure is shown in figure 4-2. It includes a root, some branches and some leaves. The root is the ancestor of all the leaves. The leaves refer to all the nodes except the root. Each leaf has a father and may have several sons. The branches are the connections between nodes.

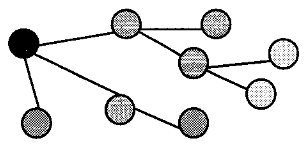


Figure 4-2: Tree

One of the advantage of this architecture is that information can be efficiently propagated through it. A tree algorithm used in ad hoc networks is presented in [6]. A new node can easily join in just by creating a connection with any nodes in the tree. Through a learning procedure, all the ancestors will have the knowledge of the new node. This is done by propagating information back to the root through the branches. "Node-leaving" is also not a big problem. When any father node leaves, it just "delivers" its sons to its father. Another advantage of this architecture is that it can be scalable. There is no limit in the growth of a tree. In addition, there is much less connections between nodes than those found in full mesh.

However, the tree is an asymmetric architecture. The root plays a critical role for maintaining the tree. It is subject to heavy burdens if the tree is big. It is hard to determine the most powerful node as a root because in ad hoc conferences, newly joined node may be more powerful than the pre-existing one. In addition, it is difficult to reorganize the whole tree when a new node joins and the old root cannot handle the tree anymore.

4.3 Clustering

Clustering is a kind of architecture that divides a set of nodes into different clusters and provides connections between them. Each cluster has a *cluster head*, which is responsible for maintaining a list of cluster members and also connecting to other clusters. Clustering is often used in communication networks [35,36,37] to handle the issues of distributed computing. Similar to the tree, clustering can be easily scaled. It organizes large number of nodes in small clusters and each cluster is self-governed. Clustering also provides possibility for efficient information propagation. Since the cluster heads are all connected, they can propagate information to each other. All cluster members can then get the information from their cluster heads.

Compared with the tree architecture, clustering is more symmetric. That is, instead of maintaining the table of all nodes, the burden of the root-node is shared by several cluster heads. If new nodes are added and the cluster head is no longer capable of maintaining them, it just splits the cluster into two smaller clusters. There is then the possibility to select the most powerful nodes as cluster heads.

In our opinion, two possible types of clustering: centralized clustering and mesh clustering can be used in our signaling system. Other clustering architectures are less interesting due to their unmanageability.

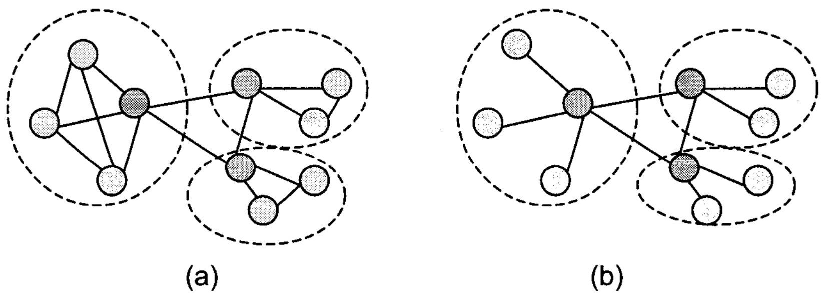


Figure 4-3: Clustering
(a) Mesh clustering (b) Centralized clustering

Clusters in centralized clustering are organized in a centralized manner. Heads are the center points of each cluster. In contrast, clusters in mesh clustering are organized in a full mesh manner. Nodes have full mesh connections with each other. Figure 4-3 shows these two architectures. If we compare the two, the meshed one has a better fault tolerance while the centralized one needs less signaling connections. Since the network resource is limited in peer-to-peer ad hoc networks, we conclude that the centralized clustering architecture is a better solution.

4.4 Summary

In this chapter we presented the pros and cons of the four architectures. By comparing these architectures, we chose the centralized clustering as the architecture of our new signaling system. We will refer to it as *clustering* in the rest of this thesis. Table 4-1 gives the evaluation of the alternative architectures.

	Centralization	Full mesh	Tree	Clustering	
				Mesh	Centralized
Without central control point	No	Yes	Yes	Yes	Yes
Potential for efficient information propagation	--	Low	High	High	High
Potential for signaling load	Low	High	Low	Middle	Low
Symmetric architecture	No	Yes	No	Partly	Partly
Level of scalability	Low	Low	High	High	High

Table 4-1 Alternative Architectures

CHAPTER 5 CLUSTERING-BASED SIGNALING SCHEME

As described in chapter 4, clustering enables scalability and does not require centralized control. The idea of clustering has been used to solve the routing [47] and the topology [48] problems in ad hoc networks. However, these researches differ from our work because of the following reasons: Firstly, our focus is on how to organize multiparty sessions while the researches are focused on how to organize the network. Secondly, we use the clustering architecture to help propagate information and to reduce signaling connections, while the previous researches use the idea to reduce the communication overhead for distributed routing and to guarantee strong connectivity for the network.

In this chapter, we will introduce the general principles and then present the preliminary thoughts on procedures for the concrete operations. In the last section, we will analyze the potential of the new architecture.

5.1 General Principles

The Signaling Agent (SA) is the only functional entity in our architecture. It resides in each peer and handles signaling protocols. At any given time, it either acts as *a member* or as *a super member.* The super member is the cluster head and is responsible for maintaining a list of members. There is only one super member in a cluster and all the members are connected to it. Super members are also interconnected. A cluster is identified by a unique cluster id. Both super member and members in the cluster hold this id. Figure 5-1 shows an example of the clustering-based architecture.

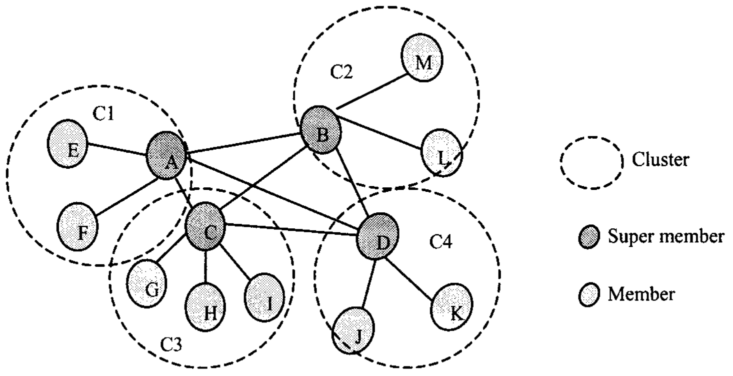


Figure 5-1: Clustering Based Signaling Architecture

Clusters are dynamically created and deleted. They can split and merge. The key parameters are the *Split value* (Sv), the *Merge Value* (Mv), and the *Level Value* (Lv). If the number of members in a cluster reaches Sv, the cluster splits into two. If it reaches Mv, the cluster merges with another cluster. Split value and merge value are critical for the clustering architecture. Different values will lead to different flavors of architectures. For example, if the split value is set to an excessively large number (larger than the number of participants in the conference), the architecture will become a centralized one. If it is set to 1, the architecture will become a full-meshed one. Moreover, the merge value should always be set to a number that is less than half of the split value. Otherwise, a merging procedure may lead to further splitting and merging procedures.

Procedures are needed to support the concrete operations, such as super member election, cluster creation/deletion, changes in cluster membership (i.e. including super member leaving the conference), split and merge. We present these procedures in the next section.

5.2 Detailed Description

5.2.1 Super Member Election

A super member is elected when a new cluster is created, when an old super member leaves, or when two clusters merge. The elected member is the one who has the most resource (e.g. processing power / memory).

In order to implement this, resource capability is to be exchanged between a super member and a member, as well as between different super members. A super member keeps two resource-lists: one for members, the other for neighboring super members. The list of members is used for super member selection when a super member leaves or when a cluster splits. The list of super members is used when two clusters merge.

5.2.2 Creation and Deletion

The first cluster is created when the first two participants are connected. One of the two participants will be elected as a super member by following the super member election procedures described above. A unique cluster ID will then be generated and parameters Mv, Sv and Lv will be set.

The last cluster is deleted when the last participant leaves the session. The cluster ID and parameters will be cleared.

5.2.3 Member Change – Joining

There are two ways for a participant to be added into the multiparty sessions: invited by a super member or referred by a member. In the latter case, the referred participant should establish a session with the super member because there are no direct connections between members in our architecture. In both cases, the super member should also update the list of its members and propagate the information to super members of neighboring clusters after the participant joins.

Figure 5-2 shows an example of joining in which super member H invites participant K to join the conference. If K is successfully added, H then propagates this change to its neighbor A.

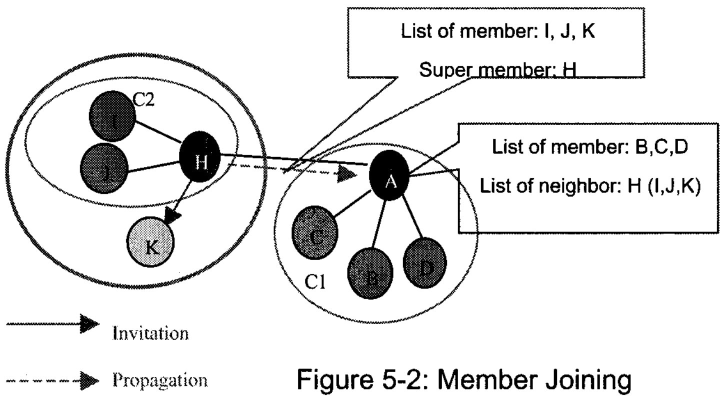


Figure 5-2: Member Joining

Many participants may join a multiparty session simultaneously. Our architecture can handle this situation gracefully. If participants join different clusters, super members of these clusters can have knowledge of the new participants. Through information propagations between clusters, all these participants can have knowledge of each other. When participants join the same cluster, the super member deals with them one by one and these new members can know each other through the super member.

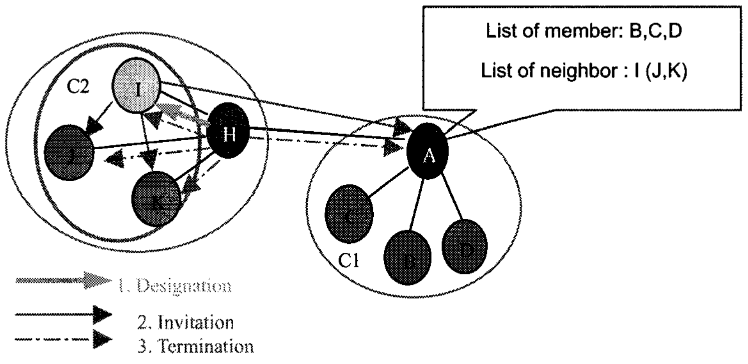
5.2.4 Member Change – Member Leaving

It is very simple for a member to leave the cluster. The member terminates the session with its super member. The super member updates the member list and propagates the change.

5.2.5 Member Change – Super Member Leaving

The leaving of a super member is more complex than that of the member because it holds a key position. Firstly, the leaving super member designates a new super member called *candidate*. Secondly, the candidate creates connections to members in the cluster and to super members of neighboring clusters. Only then will the candidate become the new super member. Thirdly, the old super member terminates all its sessions and leaves. We show a super member's leaving example in Figure 5-3 in which super member H wants to leave, so it designates member I as the new super member. Candidate I creates connections with member J, K and neighbor A. Finally, H terminates connections with new super member I, member J, member K, and neighbor A.

One problem that we may encounter during leaving procedure of a super member is that if the old super member keeps receiving and handling other inviting, terminating or propagating messages while the candidate is establishing connections, the candidate will have no knowledge of the latest changes. We can solve this problem by setting a "locked" status for the old super member until the candidate takes the position of the super member.



5.2.6 Split

A cluster may split into two when its size reaches Sv, or when the super member of this cluster reaches its processing limit. A split procedure only happens after a new participant joins. The super member of the cluster, called *splitter*, is the one who starts the split. The procedure is as follows:

The splitter chooses a new super member from its member list. It then divides some of its members to a new cluster. The candidate generates a new cluster ID and then creates connections to its members and to the super members of neighboring clusters. After this, the candidate becomes a real super member. Finally, the splitter terminates sessions with all the members of the new cluster.

Up to now, we did not describe the policy of the split. This means that the splitter decides how many members are to be part of the new cluster. There are two possible policies. The first one is that the splitter cuts off one member to keep the cluster's size within the Sv. For example, cluster *c* includes 10 members. They can be presented as *{csm, cm1, cm2, ..., cm9}*. The csm is the super member and *cm1~cm9* are members. If csm decides to use cm2 to form a new cluster called *c1*, the new clusters will be *c:*

{csm, cm1, cm3, cm4, ..., cm9} and *c1: {cm2}*. This algorithm is deficient because any further joins to *c* will lead to a new split procedure. The other policy is called *fair split*. In this method, a splitter divides its members into two halves. Two divided clusters are then formed. The detailed description is as follows: Assume that the number of users in the cluster is Cn. If Cn is even, the splitter splits Cn/2 members out to create a new cluster. Otherwise, it splits quotient(Cn/2) members out. If we use the previous example, one possible split result would be *c: {csm, cm1, cm3, cm4, cm5}* and *c1: {cm2, cm6, cm7, cm8, cm9}*. We prefer this policy because it eliminates the problem of further split procedures.

5.2.7 Merging

A cluster will merge with another cluster if its size is less than the merge value and if by merging, the total number of members is less than the split value. Similar to the split procedures, a super member is responsible for starting a merging. We call it *merger*. The super member, that the merger wants to merge with, is called *mergee*. The merger may designate itself or the mergee as the new super member according to the super member election result. The new super member creates connections to the members of the other cluster. And the old super member becomes a member and terminates all connections to its old members.

5.3 Summary

We have proposed a clustering-based signaling architecture in this chapter. The definitions and operation procedures were presented. From our work, we can see that the architecture can potentially meet all the signaling requirements. Firstly, there is no permanently centralized entity. SAs act as super member on a transient basis. Secondly, sessions are maintained in a distributed and dynamic manner. The clusters are dynamically created and deleted. Thirdly, the architecture can be of lightweight if appropriate technologies are selected for implementation. Furthermore, resources are used in an optimal way because super members are elected according to the resource capabilities of participants. Finally, the load induced in the network is significantly lower than the load induced when a full mesh approach is used.

However, the proposed architecture is not completely optimal because some procedures such as super member leaving, splitting and merging need the cluster to be reshaped. Extra network load will be induced in the reshaping procedures. Because of the time constraint, we will not optimize the procedures in this thesis but clarify it and put it to the work in the future.

CHAPTER 6 A SIP BASED IMPLEMENTATION

We implement our proposed architecture as an extension to SIP [7] because SIP is lightweight and extendible. It is important to stress that our signaling architecture does not rely on SIP. We use SIP just as a technology to implement it.

To simplify the implementation, we suppose that all participants in a conference have same processing and memory capabilities. Super members are elected randomly. In this chapter, we will present the extension, describe the behavior of SIP end systems and illustrate the signaling sequences.

6.1 The Extension

The extension includes new entity, new headers, new concepts and new semantics. We will introduce them one by one in this section.

6.1.2 An Identification

We define a key word "clustering" that will appear in "Supported" header field of SIP to identify this extension.

6.1.2 New Entity

The only functional entity we keep from SIP is the user agent (UA) since there is no centralized server in peer-to-peer ad hoc networks. In addition, we propose a new entity super user agent (SUA). UA maps to the member and SUA maps to the super member.

6.1.3 New Header Fields

Four new header fields are defined: Conf-ID, Participant-In-Cluster, Neighbor and Cluster-Parameters. *Conf-ID* header field acts as a unique identifier to group all the messages in a particular conference. The requests and responses within a conference should have the same conference id. Participant-In-Cluster header field is used to represent the participants in a same cluster. It consists of three parameters: super member, member list and cluster id. *Neighbor* header field represents a neighbor of the cluster represented by the Participant-In-Cluster header field. It also consists of super member, member list and cluster id. *Cluster-Parameters* header field is used to exchange cluster parameters. This header field consists of three parameters: split-value, merge-value and level-value. Appendix A shows the syntax of new header fields.

6.1.4 Extension to methods

Instead of defining new request methods, we extend the existing messages with new header fields. The messages are INVITE, CANCEL, ACK and BYE from the baseline SIP [7], and REFER [40], INFO [41] and SUBSCRIBE/NOTIFY [42] from existing SIP

Table 6-1: New Headers and SIP Requests

	INVITE	ACK	BYE	CANCEL	REFER	SUBSCRIBE	NOTIFY	INFO
Conf-ID	M	M	M	M	M	M	M	M
Participant-In-Cluster	M	M	M	M	M	M	M	M
Neighbor	C	C	C	C	C	C	C	C
Cluster-Parameters	M	C	C	C	C	-	-	-

"M": Must appear in a method "-": Cannot appear in a method "C": May appear in a method extensions. We also modify semantic of the REFER message. Table 6-1 describes the relationships between the new header fields and the request methods.

Next, we will talk more about REFER, INFO and SUBSCRIBE/NOTIFY requests because we define new ways of using these messages.

6.1.4.1 REFER

The REFER method will be used in four cases:

1. When a UA wants to invite a new participant, it sends a REFER request to the participant;
2. A SUA uses REFER to initiate a cluster split procedure;
3. A SUA uses REFER to initiate a cluster merging procedure;
4. When a SUA wants to leave the conference, it uses REFER to designate a new SUA.

6.1.4.2 INFO

The INFO method is used to propagate session-unrelated information within a dialog. We use it to propagate information when there is a change of membership in a cluster. To reduce the network load, the INFO request is only exchanged between SUAs.

6.1.4.3 SUBSCRIBE/NOTIFY

The SUBSCRIBE and NOTIFY methods are used to request notification from remote nodes indicating that certain events have occurred. Since we do not propagate any membership changes to UAs, a UA, that wants to get the changes, can send a SUBSCRIBE request to its SUA. The SUA then gives the last changes to the UA by sending a NOTIFY request. We define a new event type for this notification: *Update-participant*. It will be presented in the Event header field of the SUBSCRIBE request.

6.1.5 New Concepts

We keep some important concepts of SIP (like dialog and call) and define two more concepts: *cluster* and *conference*. The two are established and maintained by both a UA and a SUA.

A cluster indicates in which cluster a UA or a SUA is. At any given time, a UA and a SUA can only belong to one cluster. The cluster in a UA comprises three parameters: a *cluster id,* a *super member* and a *cluster-parameter*. The cluster in a SUA comprises five parameters: a *cluster id,* a *super member,* a *cluster-parameter, a member list* and *a neighbor list*.

A conference is used to store the context of a particular conference. It contains a *conference id,* a set of *dialogs* and a *cluster*.

6.2 Behavior of End Systems

There are two types of end systems in our signaling protocol: UA and SUA. A UA consists of a UA client (UAC) that is responsible for sending requests and processing responses, and a UA server (UAS) that is responsible for processing requests and sending responses. A SUA also consists of a SUA client (SUAC) and a SUA server

(SUAS). In this section, we will describe the behavior of UAC, UAS, SUAC and SUAS.

6.2.1 Initiation

a) UAC (SUAC) Behavior

✧ Generating and sending INVITE request:

Before generating an INVITE, a caller's UA designates itself as the SUA (since we assume that all the participants in a conference have same processing and memory capability, for simplicity, the caller's UA is set to the SUA when a conference is initiated).

Firstly, a SUAC creates header fields: From, To, Call-ID, CSeq, Max-Forwards, Via and Contact. In addition to that, the SUAC creates Supported header field using "clustering" and creates Allow header field using "INFO". That means that the clustering extension is supported by the SUA and INFO request is allowed to be sent later in the dialog. The SUAC also generates a unique conference id and uses it to create Conf-ID header field. Moreover, it creates a Participant-In-Cluster header field. In this header field, the super member parameter is set to the caller's SIP-URI, the members parameter is set to callee's SIP-URI, the cluster id parameter is set to a unique cluster id that the SUAC generated. Finally, the SUAC creates a Cluster-Parameters header field through setting level-value, split-value and merge-value. This extension does not define how to choose these values so they should be obtained from some pre-configuration or user's stimulus. The first INVITE request will not have a Neighbor header field.

The SUAC then generates a INVITE request using the header fields and sends the request.

✧ Processing responses:

If a UACs receives 501, it will not have any further behavior. The invitation is failed because the callee does not support this extension.

If a SUAC receives 200, it creates a dialog. It also creates a cluster using the cluster id, super member and member list from the Participant-In-Cluster header field, using cluster parameters from Cluster-Parameters header field, and using neighbor list from Neighbor header fields. If there is no Neighbor header field, the neighbor list of the cluster will be set to null. In addition, the SUAC creates a conference with conference id from the Conf-ID header field. It also adds the dialog and the cluster into the conference. Finally, it sends ACK and starts the media session.

If a SUAC receives other responses, the same operations will be performed as that in RFC 3261.

b) UAS Behavior

If a UAS receives an INVITE request and if it does not support clustering extension, it sends 501 as a response. If it supports the extension and the invitation is accepted, it generates and sends a 200 response. The UAS then creates a dialog, a cluster and a conference using the parameter values in the header fields: Participant-In-Cluster, Cluster-Parameters and Conf-ID of the INVITE.

✧ If a UAS receives ACK, it starts the media session.

6.2.2 User Joins

6.2.2.1 SUA Invites

a) SUAC Behavior

- ✧ Generating and sending INVITE request:

  A SUAC creates header fields Conf-ID, Cluster-Parameters, Participant-In-Custer and Neighbor using the parameters maintained in the conference. Other header fields are created through the same way as that in initiation. It then creates an INVITE request and sends the request.

- ✧ Processing responses:

  If a SUAC receives a 200, it creates a dialog and sends ACK. Then it updates the conference. It also sends INFO messages to its neighboring super members if the neighbor list is not null. The Participant-In-Cluster header fields of the INFO requests carry the latest member list of the cluster that the SUAC is in. The SUAC then checks the number of users in the cluster. If it is larger than the split value, the SUAC calls the split procedure.

  Other responses are processed same as that in initiation.

- ♦ Split procedure

  Before sending a split request, a SUA selects one UA as a new SUA following the *fair split* policy (presented in section 5.2.6).

  - ➢ Generating and sending REFER request:

    A SUAC creates Participant-In-Cluster header field by setting the super member parameter to the SIP-URI of the candidate. It then creates the

Neighbor header fields using the neighbor list in the cluster. It also creates Refer-To header fields with all the participants that the candidate should invite. After that, SUAC creates the REFER request and sends the request.

➢ Receiving responses:

If the SUAC receives 202, the REFER transaction is finished. The SUAS then waits for a NOTIFY. After a while, if it receives the NOTIFY, the SUAS sends 200 and inform its SUAC. The SUAC then sends BYE requests to the members belonging to new cluster. If the SUAC receives 200s of BYEs, it deletes dialogs and updates the cluster and the conference. It then generates and sends INFO requests to its neighboring SUAs.

If the SUAC receives other responses or it does not receive a NOTIFY after the time out that was presented in Expire header field of the REFER request, the split will be failed. Then the SUAC may choose not to split at this time or call split procedure again.

b) SUAS Behavior:

✧ If a SUAS receives an INFO request, it sends 200 and updates the neighbor list.

✧ If a SUAS receives an INVITE request and if the request is outside any dialogs that it maintains, it sends 200 and creates a dialog. Then it updates the conference.

c) UAS Behavior

✧ If a UAS receives an INVITE request that is not in a dialog, UAS creates a dialog and updates the conference.

✧ If a UAS receives a REFER request, if the super member parameter in Participant-In-Cluster is SIP-URI of itself, the UAS sends 202 and asks its UAC to send INVITE requests to all the users in Refer-To header fields. If it receives successful information from its UAC, the UAS changes itself to SUA and asks its UAC to send a NOTIFY request to the old SUA. Then it updates the conference.

d) UAC Behavior

If asked by its, a UAC generates and sends INVITE requests. It then receives responses and sends successful or not successful information to its UAS.

If asked by its UAS, a UAC generates and sends a NOTIFY request and receives 200.

6.2.2.2. UA Invites User

a) UAC Behavior

✧ A UAC generates a REFER request and sets Refer-To header field to the super member in the cluster. Then, it sends the REFER request and receives 202.

✧ A UAC generates an INVITE request if asked by its UAS. It then receives a response and informs the UAS.

✧ A UAC generates a NOTIFY request if asked by its UAS. Then it sends 200.

b) UAS Behavior:

✧ If a UAS receives a REFER request: it sends 202 and asks its UAC to send INVITE to the user presented in Refer-To header field. Then if it receives successful information from its UAC, the UAS asks the UAC to send a NOTIFY request.

✧ If a UAS receives a NOTIFY request, it sends 200 and updates the conference.

c) SUAS Behavior

✧ If a SUAS receives an INVITE request and if the request is from a user that is not in the conference, it sends 200 and creates a dialog. Then it asks its SUAC to send INFO to its neighboring SUAs. If the number of users in the cluster is larger than the split value, the SUAS asks its SUAC to do the split procedure.

✧ If a SUAS receives an INFO request, it updates the neighbor list and sends 200.

d) SUAC Behavior

✧ A SUAC sends INFO requests if asked by its SUAS.

6.2.3 User Leaves

6.2.3.1 UA Leaves

a) UAC Behavior

✧ A UAC generates a BYE message and sends to its SUA. Then it receives 200 and clears the conference.

b) SUAS Behavior:

✧ If a SUAS receives a BYE request, it sends 200, updates the conference and asks its SUAC to send INFO. If the size of the cluster (*Cn*) is less than the merge value (Mv) and if there is a neighbor (number of users in it is *Nn*) meeting the requirement of $Cn+Nn < Mv$, the SUAS asks its SUAC to do the merging procedure.

✧ If a SUAS receives an INFO request, it updates the neighbor list and sends 200.

- ✧ If a SUAS receives a REFER request, it sends 202. If the super member parameter in the Participant-In-Cluster header field does not equal to the SIP-URI of the SUA itself, the SUAS asks SUAC to send BYE requests to its members. Then it changes itself to a UA and updates the conference. Otherwise, the SUAS asks its SUAC to INVITE users presented in Refer-To header fields. Then if it receives success information from its SUAC, it updates the conference.
- ✧ If a SUAS receives a NOTIFY request, it sends 200.

c) SUAC Behavior

- ✧ If asked by its SUAS, a SUAC sends an INFO request.
- ✧ If asked by its SUAS, a SUAC sends BYE requests and receives 200s.
- ✧ If asked by its SUAS, a SUAC sends INVITE requests, receives 200s, and sends ACKs. Then it informs the SUAS.

- ♦ Merging Procedure

  To lessen the number of messages used in merging procedure, either the merger or the mergee who has more membership can be selected as the new SUA.

  - ➢ If a merger is elected as the new SUA, the SUAC of the merger will generate and send a REFER request to mergee. In the request, the super member parameter in Participant-In-Cluster header field is set to SIP-URI of the SUA and the members' parameter is set to the merged member list of two clusters. Then if it receives 202 response, the SUAC sends INVITE requests to all the members in mergee's cluster. And if the invitations are successful, the SUAC sends a NOTIFY request to the mergee's old super member and receives

200.

➢ If a mergee is selected as the new SUA, the SUAC of the merger will generate and send a REFER request to mergee. In the request, it sets the Refer-To header fields to members in the merger's cluster. Then the SUAC asks its SUAS to receive a NOTIFY request. After a while, if the SUAS receives the NOTIFY, the SUAC sends BYE requests to all its members and change itself to a UA.

6.2.3.2. SUA Leaves

a) SUAC Behavior

✧ A SUAC selects one of its members as the new SUA, called candidate. Then it generates and sends a REFER request to the candidate. In the request, it sets Refer-To header fields to its members and neighboring super members. After receiving 202, the SUAC asks its SUAS to receive a NOTIFY request. If the NOTIFY is received, the SUAC sends BYEs to all connected users. Finally, it clears the conference.

b) UAS Behavior

✧ If a UAS receives a REFER request from its SUA and if the super member in Participant-In-Cluster header field equals to the UA, it sends 202 and asks its UAC to send INVITE requests to the users presented in Refer-To header fields. If the UAS receives successful information from its UAC, it changes itself to a SUA and asks its SUAC to send NOTIFY to the old SUA. If the SUAS's cluster needs to merge, the SUAS asks its SUAC to do the merging procedure.

✧ If a UAS receives an INVITE request from the new SUA, it sends 200, receives ACK, and updates the conference.

c) UAC Behavior

✧ If asked by its UAS, a UAC sends INVITE requests, receives 200 responses and sends ACKs. It then informs its UAS.

d) SUAS Behavior

✧ If a SUAS receives an INFO request, it updates the neighbor list and sends 200.

✧ If a SUAS receives a NOTIFY request, it sends 200 and informs its SUAC.

6.2.4 UA Request Participant List

a) UAC Behavior

✧ A UAC generates a SUBSCRIBE request setting the Event header field to "Update-participant". Then it sends the SUBSCRIBE and receives 200.

b) UAS Behavior

✧ If a UAS receives a NOTIFY request, it sends 200 and updates the conference using the parameters in header fields: Participant-In-Cluster and Neighbor.

c) SUAS Behavior

✧ If a SUAS receives a SUBSCRIBE request, it sends 200 and asks its SUAC to send the NOTIFY request.

d) SUAC Behavior

✧ If asked by its SUAS, a SUAC generates a NOTIFY requests setting parameters in Participant-In-Cluster header field and Neighbor header fields to the values in the conference. It then sends the NOTIFY and receives 200.

6.2.5 Termination

6.2.5.1 UA Sends BYE

a) UAC Behavior

- ✧ A UAC generates and sends a BYE request to its SUA. Then it receives 200 and clears the conference.

b) SUAS Behavior

- ✧ If a SUAS receives a BYE request and if there is no neighbor in the neighbor list and only one member left in the member list, the SUAS sends 200 and clear the conference.

6.2.5.2 SUA sends BYE

a) SUAC Behavior

- ✧ The SUAC generates and sends a BYE request to a UA. Then it receives 200. If there is no neighbor in the neighbor list and only one member left in the member list, the SUAC clears the conference.

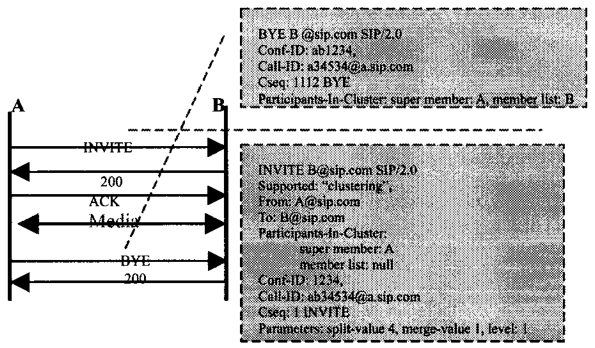
b) UAS Behavior

- ✧ If a UAS receives a BYE request from its SUA, it sends 200 and clears the conference.

6.3 Sequence Examples

6.3.1 Initiation and Termination

Figure 6-1 shows sequence examples of initiation and termination. In the example, we do not provide all the information in INVITE and BYE messages, but the new header fields.



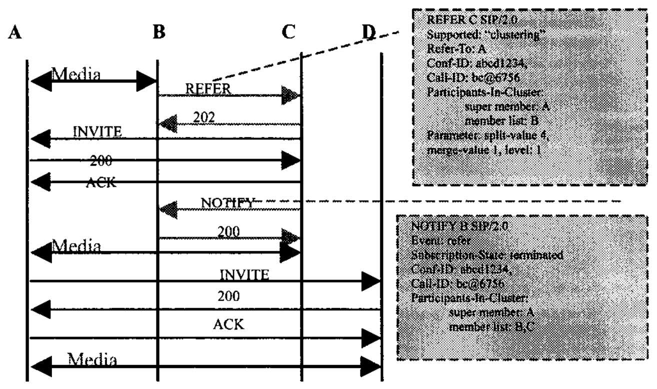
```
BYE B @sip.com SIP/2.0
Conf-ID: ab1234,
Call-ID: a34534@a.sip.com
Cseq: 1112 BYE
Participants-In-Cluster: super member: A, member list: B
```

```
INVITE B@sip.com SIP/2.0
Supported: "clustering",
From: A@sip.com
To: B@sip.com
Participants-In-Cluster:
        super member: A
        member list: null
Conf-ID: 1234,
Call-ID: ab34534@a.sip.com
Cseq: 1 INVITE
Parameters: split-value 4, merge-value 1, level: 1
```

Figure 6-1: Initiation and Termination

6.3.2 User Joins

Figure 6-2 shows two sequence examples of user join. In first example, UA B invites new participant C by sending a REFER request. C then invites the SUA A. In the second example, SUA A invites new participant D directly.



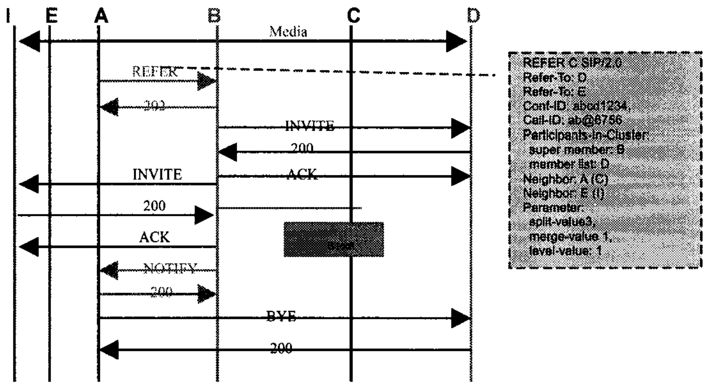
```
REFER C SIP/2.0
Supported: "clustering"
Refer-To: A
Conf-ID: abcd1234,
Call-ID: bc@6756
Participants-In-Cluster:
        super member: A
        member list: B
Parameter: split-value 4,
merge-value 1, level: 1
```

```
NOTIFY B SIP/2.0
Event: refer
Subscription-State: terminated
Conf-ID: abcd1234,
Call-ID: bc@6756
Participants-In-Cluster:
        super member: A
        member list: B,C
```

Figure 6-2: Inviting Users (B invites C, then A invites D. A is SUA)

6.3.3 Cluster Splits

Figure 6-3 shows a sequence example of split. In the example, we assume that the split value is 3. After the participant D joins, the SUA A splits the cluster.

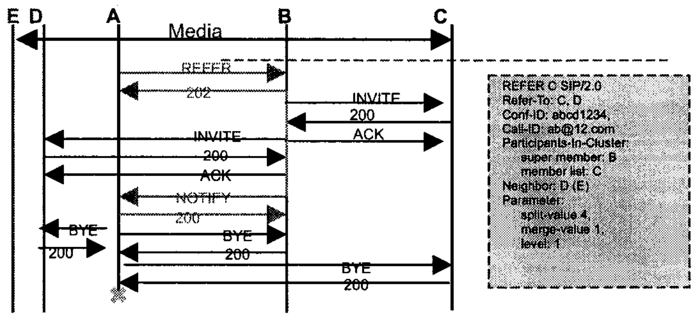


```
REFER C SIP/2.0
Refer-To: D
Refer-To: E
Conf-ID: abcd1234,
Call-ID: ab@6756
Participants-In-Cluster:
  super member: B
  member list: D
Neighbor: A (C)
Neighbor: E (I)
Parameter:
  split-value3,
  merge-value 1,
  level-value: 1
```

Figure 6-3 Split: A (BCD) is a cluster; E (I) is another cluster. A split the cluster into A (C) and B (D) (designate B as new SUA)

6.3.4 SUA Leaves

Figure 6-4 shows a sequence example of SUA leaving. In the example, SUA A, who wants to leave, designates UA B as new super user agent.



```
REFER C SIP/2.0
Refer-To: C, D
Conf-ID: abcd1234,
Call-ID: ab@12.com
Participants-In-Cluster:
    super member: B
    member list: C
Neighbor: D (E)
Parameter:
    split-value 4,
    merge-value 1,
    level: 1
```

Figure 6-4: SUA Leaves: A (BC) is a cluster; D (E) is another cluster. A wants to leave and designates B as new SUA

6.3.5 Cluster Merges

Figure 6-5 illustrates a sequence example of cluster merging. In the example, merger B has less membership than that of mergee D, so D is designated as new SUA.

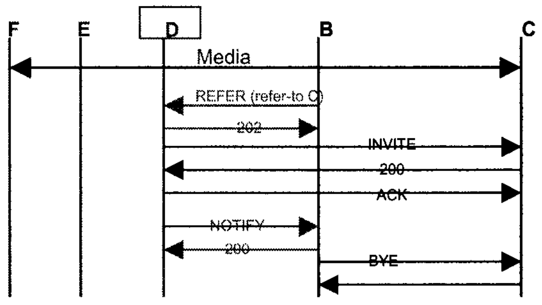


Figure 6-5 Merge: B(C) is merger's cluster; D (EF) is mergee's cluster. D will be the new SUA

CHAPTER 7 PROTOTYPE AND RESULT ANALYSIS

The purpose of the prototype is to implement and evaluate our clustering-based signaling scheme. It is built according to the SIP extension proposed in chapter 6. In this chapter, we will first give an overall architecture of the prototype. Then, we will introduce the software that we used. After that, system design and implementation will be presented. Finally, we will illustrate and analyze the preliminary results.

7.1 Overall Architecture

Figure 7-1 shows an overall architecture of the prototype. The application layer implements the actual multiparty session application including application logic and graphic user interfaces. The signaling layer implements the extended SIP user agent functionality. The middleware layer provides peer discovery and transport service to the signaling layer.

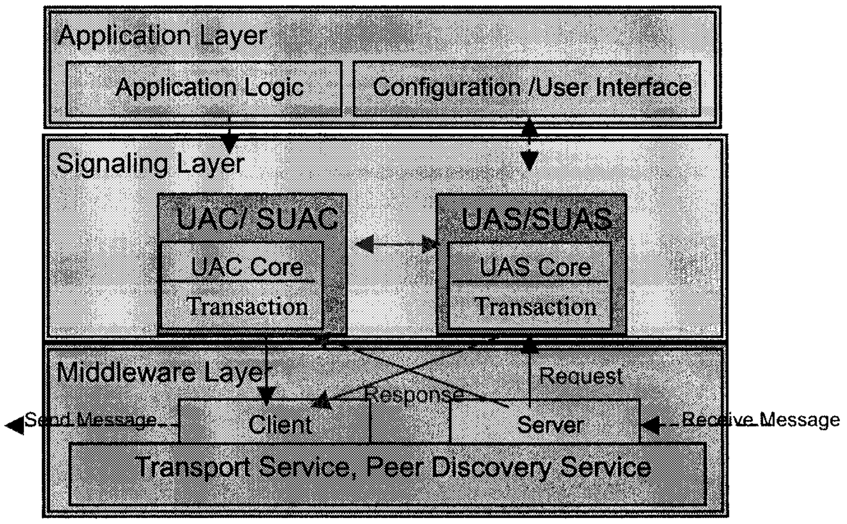


Figure 7-1 Architecture of the Prototype

7.2 Software Used in Prototype

Two kinds of Java-based freeware are used in our prototype: JXTA [29,30] and JAIN SIP Stack [45,46].

7.2.1 Peer-to-Peer Middleware -- JXTA

Peer-to-peer middleware is a kind of software that resides in each peer to provide P2P services, such as naming, grouping, peer discovering and secure transport, for P2P applications. JXTA is a successful open source implementation of the P2P middleware. It is a set of open, generalized peer-to-peer protocols, that have been specifically designed for ad hoc, pervasive and multi-hop P2P network computing, allowing any connected device (cell phone to PDA, PC to server) on the network to communicate, collaborate, and share resources with the other devices. There are six protocols defined in JXTA specification. They are shown in Figure 7-2. The protocols PDP, PIP and PBP provide services to applications. The other three protocols provide lower layer services. The JXTA organization [29] provides the latest documents and specifications.

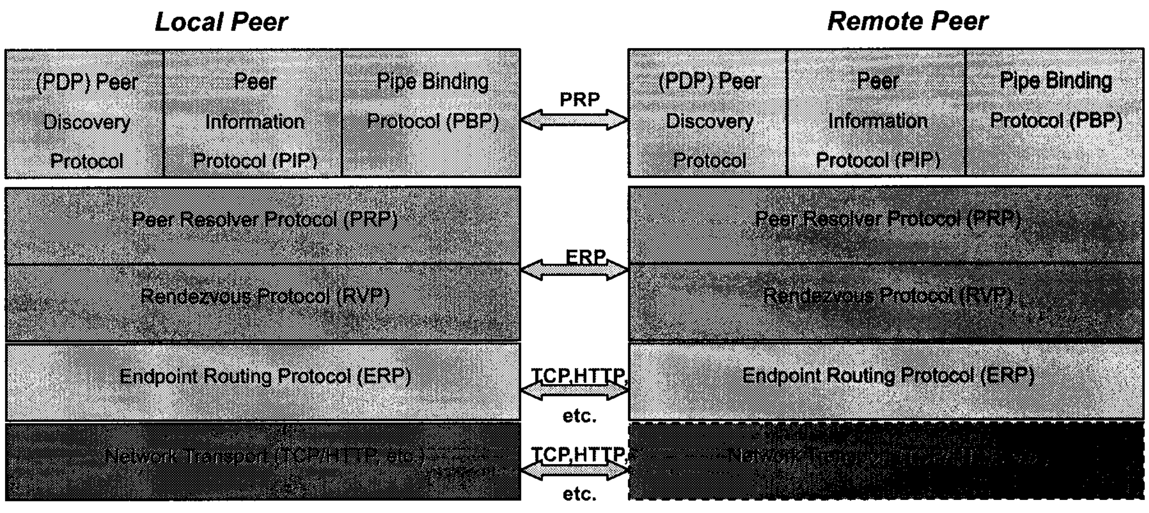

Local Peer
Remote Peer
(PDP) Peer Discovery Protocol
Peer Information Protocol (PIP)
Pipe Binding Protocol (PBP)
PRP
Peer Resolver Protocol (PRP)
ERP
Rendezvous Protocol (RVP)
Endpoint Routing Protocol (ERP)
TCP,HTTP, etc.
Network Transport (TCP/HTTP, etc.)

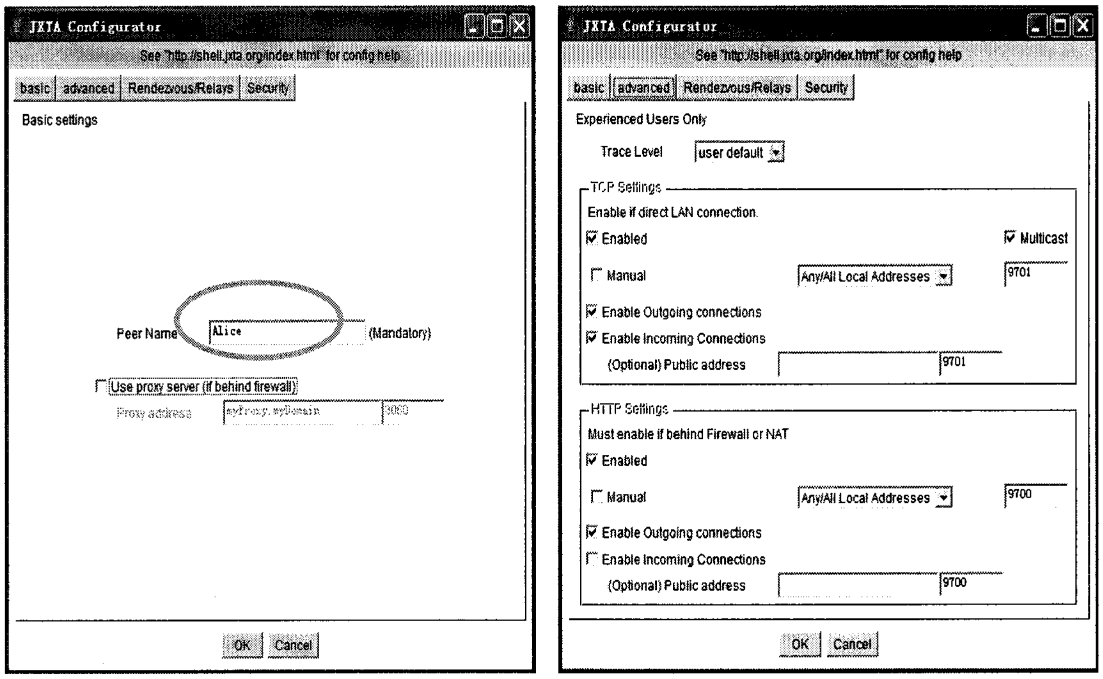
Figure 7-2 JXTA Protocol Stack

7.2.1.1 JXTA Platform

So far, the only implementation of JXTA is a Java based platform. It implements all the JXTA protocols and provides services such as peer discovery, group and pipe. The JXTA platform provides public APIs for programmer to implement applications on it. It is an on going project and new versions are always published. The earliest version was the V1.0 in 2001 while the latest version: V 2.2.1 was presented in March 2004. We use the version 2.0 in our prototype because it is a relatively stable version.

7.2.1.2 How Does a Peer Discover Other Peers

A JXTA peer can discover other peers in three steps. The first step is the *peer configuration.* JXTA provides the configuration interfaces shown in Figure 7-3. A user


JXTA Configurator
See "http://shell.jxta.org/index.html" for config help
basic | advanced | Rendezvous/Relays | Security
OK | Cancel

Basic settings

Peer Name: Alice (Mandatory)

Use proxy server (if behind firewall)

Proxy address: myProxy.myDomain 8080

Experienced Users Only

Trace Level: user default

TCP Settings

Enable if direct LAN connection.

[✓] Enabled [✓] Multicast

Manual Any/All Local Addresses 9701

[✓] Enable Outgoing connections

[✓] Enable Incoming Connections

(Optional) Public address 9701

HTTP Settings

Must enable if behind Firewall or NAT

[✓] Enabled

Manual Any/All Local Addresses 9700

[✓] Enable Outgoing connections

Enable Incoming Connections

(Optional) Public address 9700

Figure 7-3 JXTA Configurations

is required to enter a peer name that will represent the user in the network. A user can also make choices to set communication means, such as whether a peer uses a proxy, whether a peer needs relay/rendezvous peers, and what port and addresses a peer uses. Besides, the configuration provides secure settings. After the configuration, an XML based advertisement is generated, which stores all the information of a peer. In Figure 7-3, the peer name is set to “Alice” and the configuration advertisement of “Alice” is presented in Appendix B.

The second step is *peer publication.* Once a peer has been published, other peers online can find it. A peer can be published locally or remotely. If a peer is in a local area network and it does not connect to a relay/rendezvous peer, the advertisement will be published locally through multicast. Otherwise, the advertisement will be published remotely on the relay/rendezvous. The function for publishing is the *remotePublish()* in *Discovery Service*. The peer advertisement that needs to be published can be obtained from the configuration advertisement. Appendix C shows an example of Alice's peer advertisement.

The last step is *peer discovery*. A peer can discover other peers in two ways. One is to call the *getLocalAdvertisement()* method in *Discovery Service.* This method can obtain the peer advertisements in local cache. The other is to call the *getRemoteAdvertisement()* method. The method generates a discovery request and gets responses from the networks.

In the prototype, we use the JXTA peer discovery service following the above steps. We publish the peers locally and discover peers using *getRemoteAdvertisement()* method.

7.2.1.3 Pros and Cons

JXTA provides programmers an easy way to develop peer-to-peer applications. It provides a set of useful services such as the peer discovery, group service, and pipe service. This is the reason why we choose it as the middleware to support our signaling system. However, JXTA has deficiencies. Since it is not specifically designed for ad hoc networks, JXTA does not consider the network environment in which the network connectivity and network nodes are transient. In addition, the JXTA implementation is often updating and it is lack of stability. Furthermore, issues like pipe security, reliability and QoS are still under discussion.

7.2.2 Standard SIP Interface -- JAIN SIP API

JAIN SIP provides developers a standardized Java interface for SIP services. The latest version of it supports RFC 3261 functionalities, as well as INFO method (RFC 2976), reliability of provisional responses (RFC 3262), Event Notification Framework (RFC 3265), the UPDATE method (RFC 3311), the Reason Header (RFC 3326), the Message method defined for instant messaging (RFC 3428) and the REFER method (RFC 3515).

Since JAIN SIP has implemented all the messages defined in SIP and SIP extensions so far, we choose it as a basis of our SIP extension. We use headers, messages, and dialog that it provides. We also extend it using our new headers and new concepts.

7.3 System Design

In this section, we will briefly introduce the system design. Firstly, we will illustrate the classes designed for the prototype. Secondly, we will show the state diagram of the most important class: *PeerUserAgent* that implements the extended SIP user agent.

7.3.1 Class Design

Figure 7-4 shows the classes and their relationships in the prototype. Class *PeerUserAgent* is the most important class that implements the extended SIP user agent functionality. It creates instance of class *Conference* that implements the conference concept of the SIP extension. The *Conference* can create the *Cluster* that implements the cluster concept. The class *PeerUserAgent* also provides methods to generate, send and process requests and responses. The request and response messages contain the new headers. Each new header is implemented as a class. Class *MyHeaderFactory* provide methods to create instances of the headers' classes.

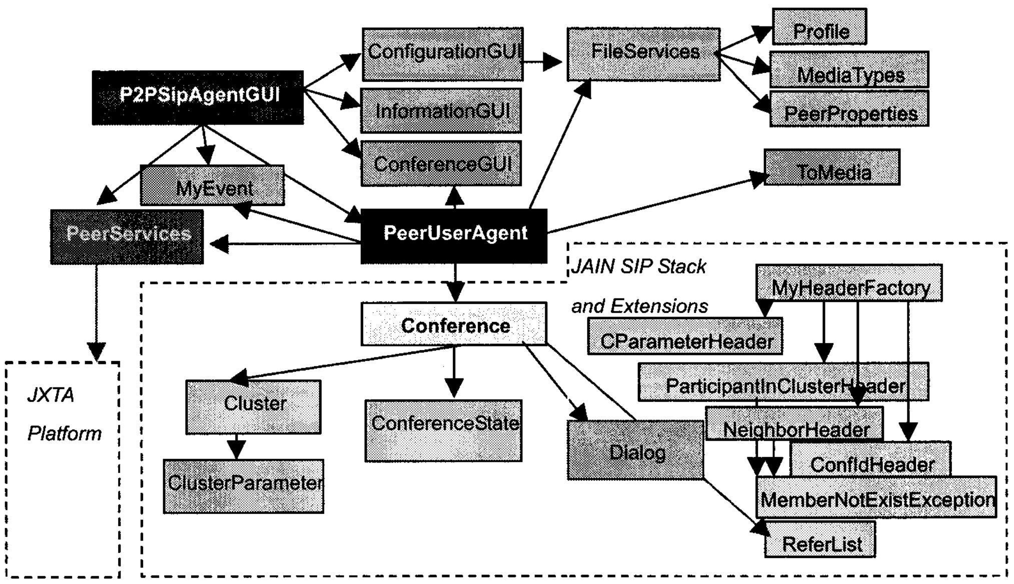
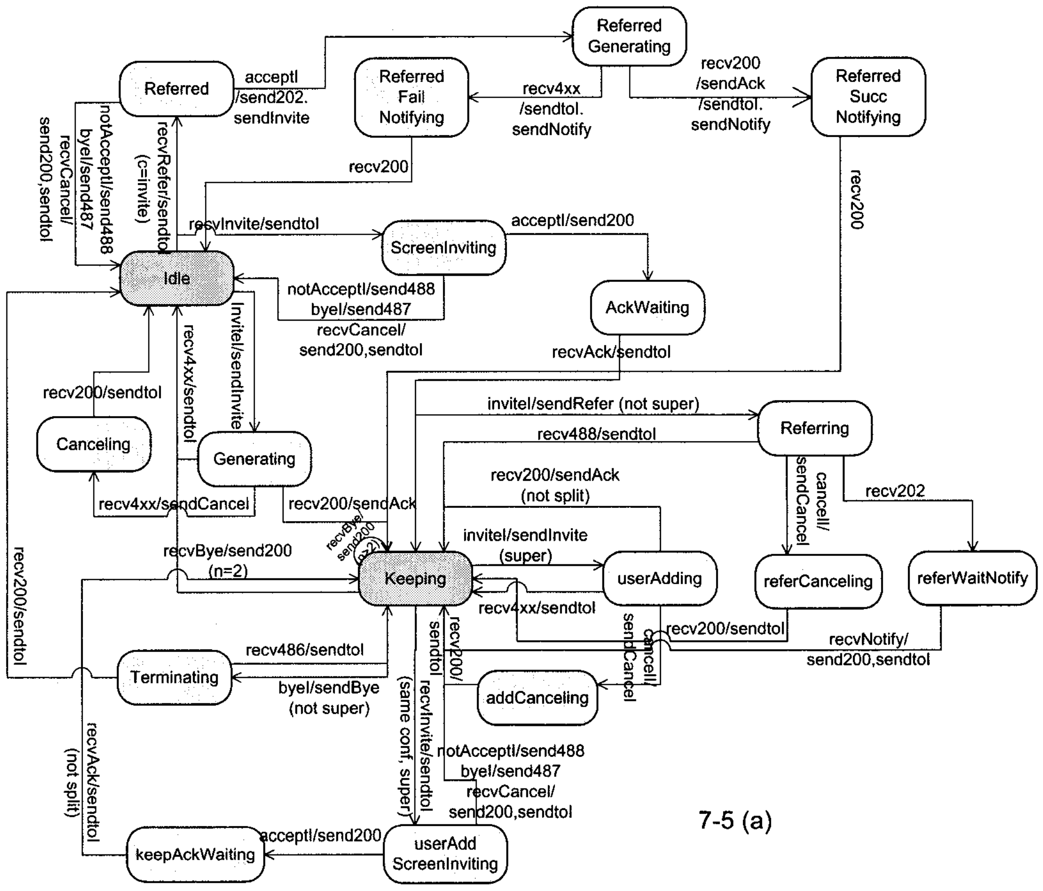


Figure 7-4 Class Structure

Class *PeerServices* uses the APIs of JXTA. It provides peer discovery service for peer user agent. Other classes such as the *P2PSipAgentGUI, ConfigurationGUI, InformationGUI* and *ConferenceGUI* implement the application logic and graphic user interfaces. Class *FileServices* provides read/write file services. Class *ToMedia* provides an interface to the media handling services. Common events are defined in the class *MyEvent*. They consist of the events from the user interface and the events of receiving SIP messages.

7.3.2 State Diagram

Class *PeerUserAgent* implements an extended SIP user agent through the



7-5 (a)

management a set of conference states. Events such as clicking a button and receiving a request or a response from other user agents will trigger state changes. Figure 7-5 (a) and (b) shows the state diagram of the *PeerUserAgent.*

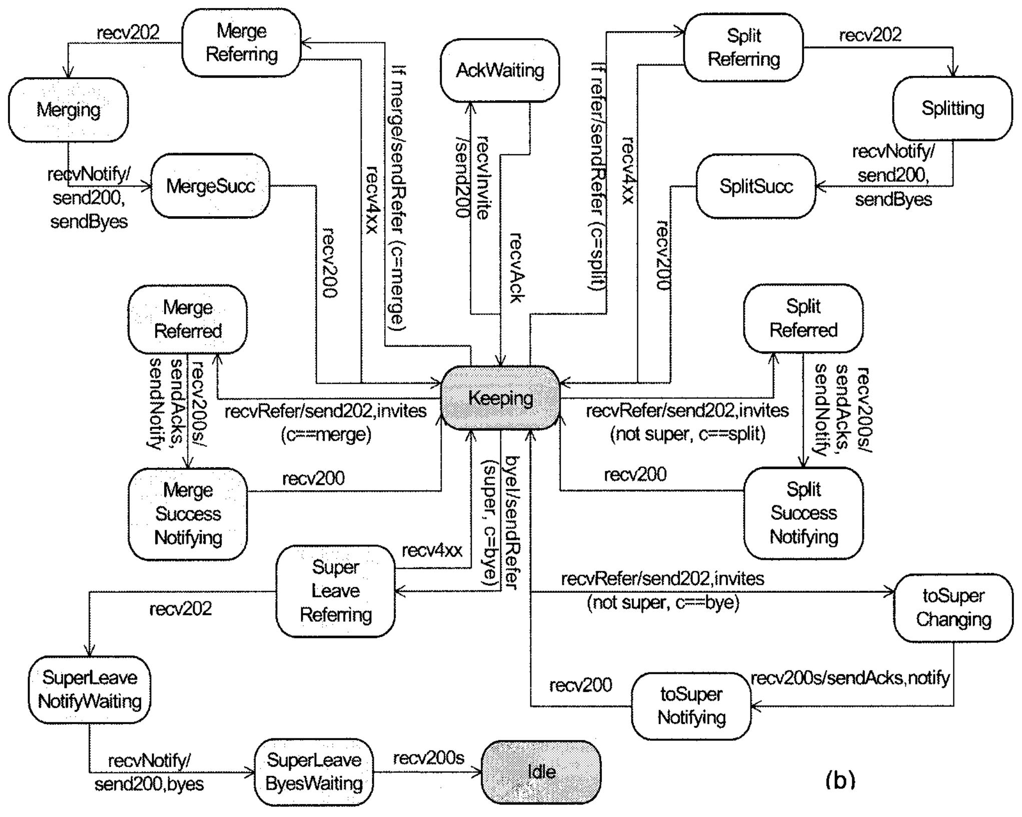


Figure 7-5: State Diagram for Class: PeerUserAgent

7.4 Implementation

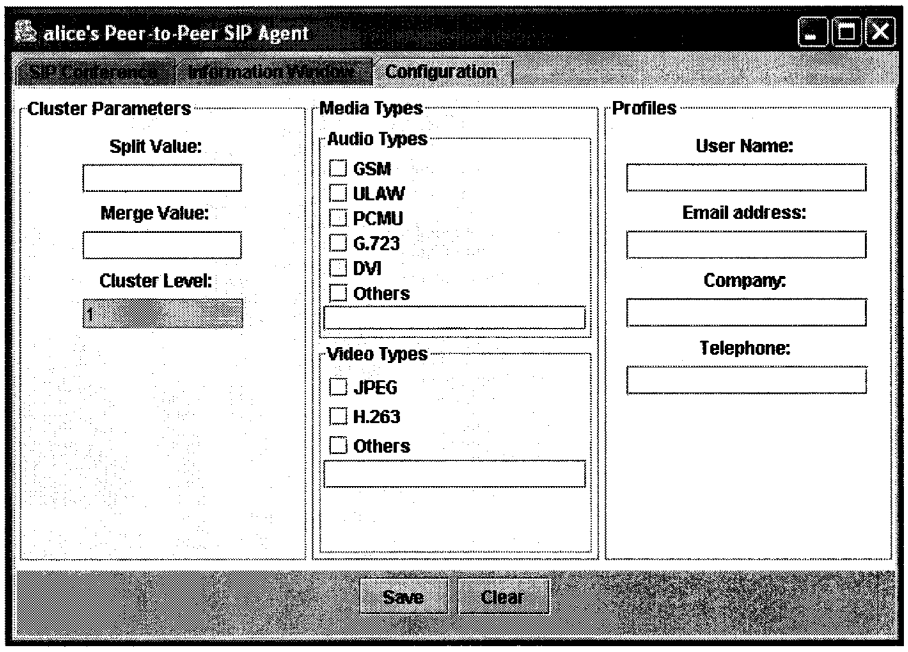
7.4.1 The Setup

We build our prototype on seven Pentium IV machines operating at 2.2 GHz with 512M RAM running Windows XP and on one machine of x86 Family 6 Model 8 stepping 3 AT/AT COMPATIBLE with 524M RAM running Windows 2000. A total of eight machines are used.

7.4.2 Functionality and Interfaces

The prototype implements the following functionality: configuration, peer discovery, conference control and signaling message measurement. Three user interfaces are developed. Figure 7-6 (a), (b) and (c) show them and we will introduce them one by one.

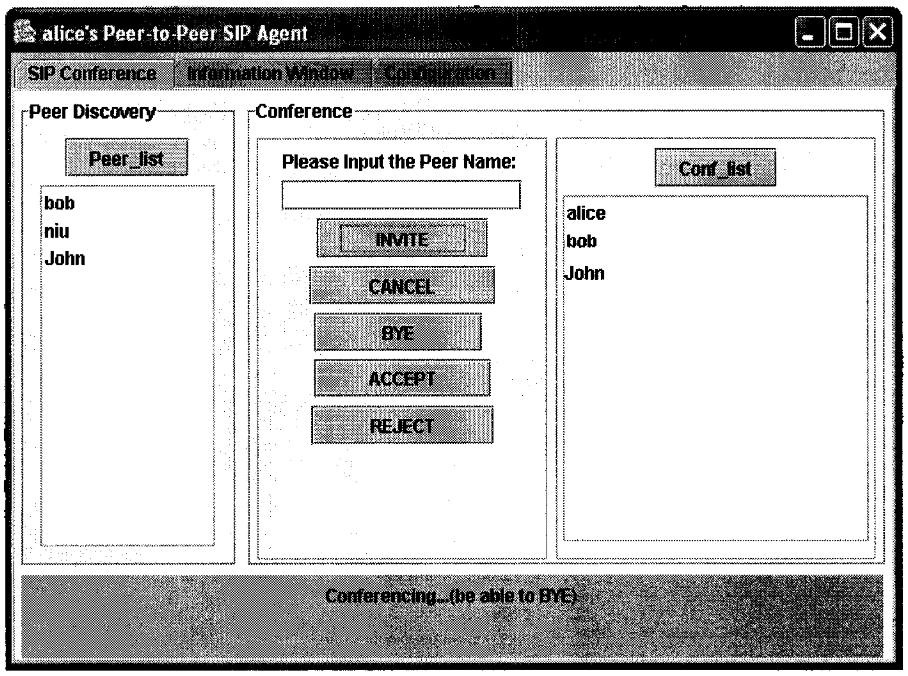
The Configuration Interface (7-6 (a)) provides three sorts of configurations. The first is to set and modify cluster parameters. The second is to set media types. Although we do not provide media handling functionality, this configuration helps to create SDP messages that may be needed in media handling. The third is to set user's profiles.



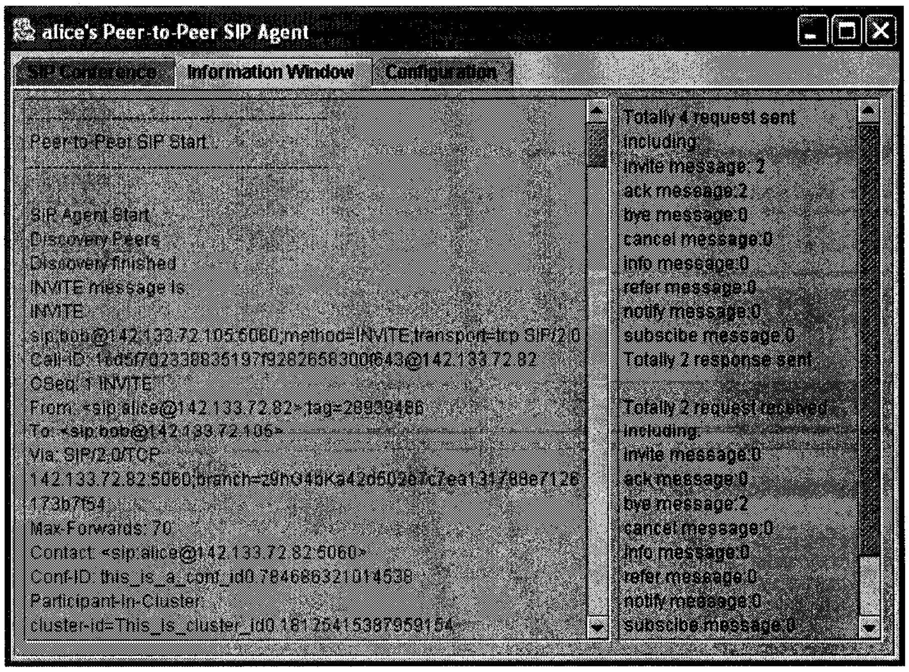
7-6 (a)

The SIP Conference Interface (7-6 (b)) provides peer discovery service and conference services. In the figure, a user named "alice" clicked "Peer_list" button and found that three other peers are online. "Alice" then invited Bob and John to join a conference. The current state of the SIP Agent is presented at the bottom of the interface. If there are incoming calls, "alice" can accept or reject the calls by clicking "ACCEPT" and "REJECT" buttons.

The Information Interface (7-6 (c)) shows two kinds of information. One is the SIP message sent and received during the conferencing. It aids to check the correctness of the signaling messages exchanged. The other is the statistic result of the number of messages sent and received. It is used for the performance measurement.



7-6 (b)



```
------------------------------------
Peer-to-Peer SIP Start...
------------------------------------

SIP Agent Start
Discovery Peers
Discovery finished
INVITE message is:
INVITE
sip:bob@142.133.72.105:5060;method=INVITE;transport=tcp SIP/2.0
Call-ID: 1dd5f7023388351971928265830016 43@142.133.72.82
CSeq: 1 INVITE
From: <sip:alice@142.133.72.82>;tag=28939486
To: <sip:bob@142.133.72.105>
Via: SIP/2.0/TCP
142.133.72.82:5060;branch=z9hG4bKa42d502e7c7ea131788e7126
173b7f54
Max-Forwards: 70
Contact: <sip:alice@142.133.72.82:5060>
Conf-ID: this_is_a_conf_id0.784686321014538
Participant-In-Cluster:
cluster-id=This_is_cluster_id0.181254153879591 54
```

```
Totally 4 request sent
including:
invite message: 2
ack message:2
bye message:0
cancel message:0
info message:0
refer message:0
notify message:0
subscibe message:0
Totally 2 response sent

Totally 2 request received
including:
invite message:0
ack message:0
bye message:2
cancel message:0
info message:0
refer message:0
notify message:0
subscibe message:0
```

(c)

Figure 7-6 Graphic User Interfaces

7.5 Preliminary Results

7.5.1 Metric and Comparison Data

The only metric that we will use is the number of signaling messages exchanged between peers. Since the sessions are distributed, we count messages received in each peer, and then sum them up. To simplify the measurement and get useful result, we only measure the successfully session-related sequences without counting any failed cases or re-Invite procedures. Since the notification sequence (SUBSCRIBE/NOTIFY) in our SIP extension is session-unrelated, we do not count it either.

We will make two kinds of comparisons. One is to compare our system with the full mesh approach. The purpose of this comparison is to evaluate the scalability of our architecture. The other kind of comparison is to compare different split values. It is used for studying the optimization of the clustering-based architecture.

In the first comparison, sessions of two to eight parties are evaluated and the session establishment and termination sequences are measured. In the measurement of our system, a fixed split value is chosen. The number of messages generated by the system varies depending on how new members are invited, and also on how super members and members leave sessions. In order to obtain the practical performance, we adopt the cases in which the minimum and the maximum number of messages are exchanged. In the measurement of the full mesh approach, we refer to the sequences discussed in [44]. In the document, if the number of participants in a conference is N (N>1), the messages used to establish sessions will be ((N*(N-1))/2)*3, and that to terminate sessions will be ((N*(N-1))/2)*2.

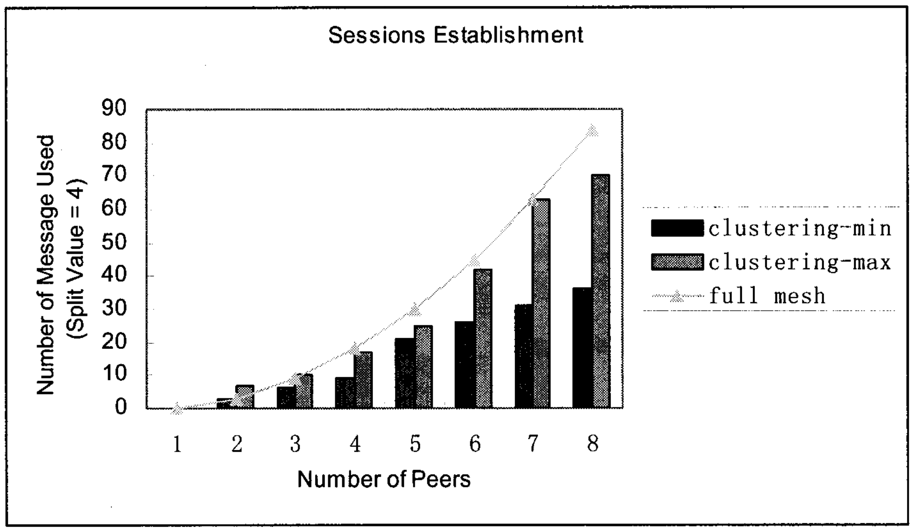
In the second comparison, we set the split value from one to eight and measure which value leads to a minimum number of messages exchanged for two to eight peers. Establishment sequences are measured.

7.5.2 Results and Analysis

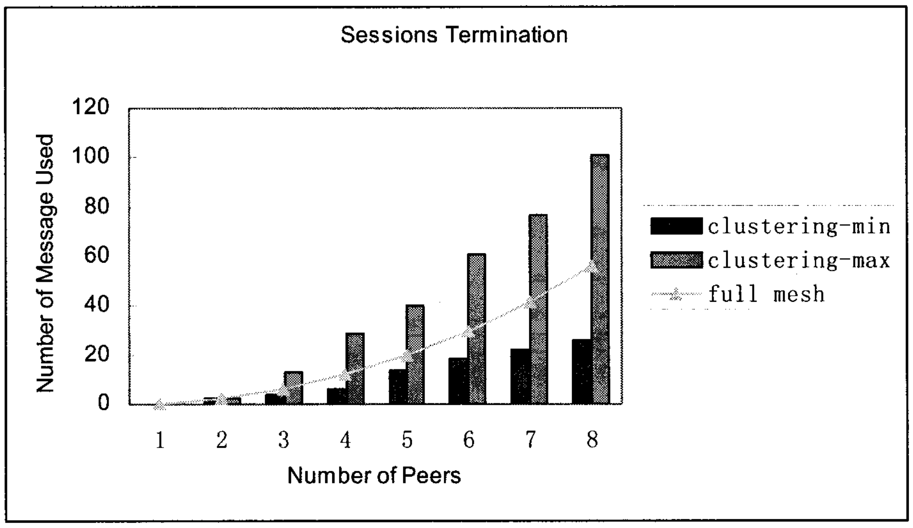
Figure 7-7 shows the results of the first comparison. Columns in the figure represent the number of messages used in clustering-based architecture while lines represent the ones used in the full-meshed one. In the columns, the "clustering-max" is the maximum number of messages exchanged while the "clustering-min" is the minimum.

The 7-7 (a) shows the result for session establishment. In the case, we set the split value as 4 so the first split happens when the fifth peer joins. We can see that more messages are generated in the full mesh than that in the clustering when the number of peers exceeds three. And the gap between the "full mesh" and the "clustering-min" becomes bigger and bigger with the enlargement of the scale of a conference. The result proves that the clustering-based approach is more scalable than the full mesh.

The 7-7 (b) shows the result for session termination. The minimum number of messages used in the clustering is still less than that used in the full mesh. However, the bad performance occurs when super members always leave clusters earlier than members do. This is due to the fact that the super member re-election and the cluster re-configuration generate a considerable number of messages. To avoid this, a super member, whenever it is elected, should leave the conference as late as possible. This result proves that our signaling scheme is still not an optimal one.



7-7 (a)



(b)

Figure 7-7 Comparisons Between Full Mesh and Clustering

Figure 7-8 shows comparison results of different split values. We can see that when the split value is 1, a large number of messages are exchanged between peers. This is because the signaling architecture is actually a full mesh in the case. When the split value is larger than the number of peers in a conference, a small number of messages are exchanged. But the architecture becomes a centralized one. The traffic of the super member will be heavy when the scale of a conference is large.

It is hard to find a proper split value that can meet requirements of both a minimum number of messages' generation and a balanced signaling traffic. However, an interesting result in Figure 7-8 shows that for six, seven and eight peers, when we set the split value to 4, the number of messages generated is less than that generated when other split values are set (not including the split value that results in a centralized architecture). Forasmuch, there may exist optimal split values for large-scaled, clustering-based architecture.

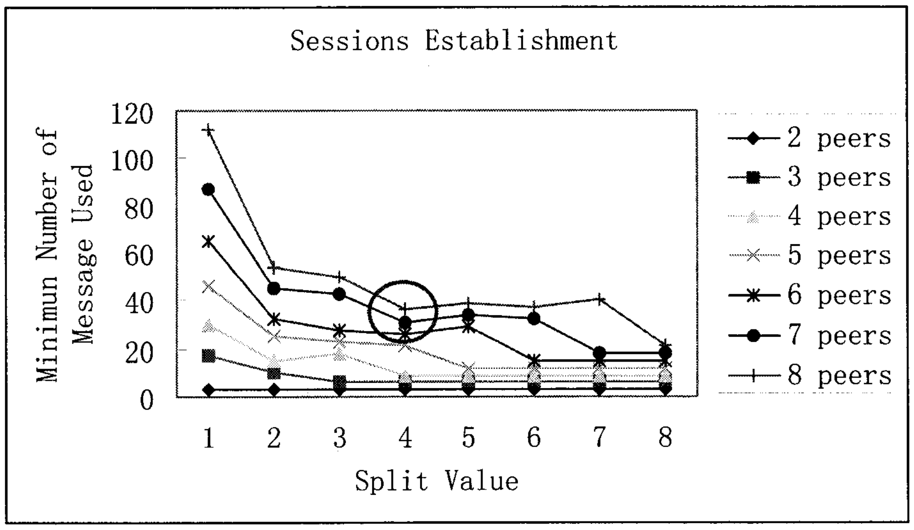


Figure 7-8 Comparisons of Different Split Values

7.6 Summary

We implemented a peer-to-peer SIP user agent that realizes the clustering-based signaling scheme. We used JXTA as a middleware and JAIN SIP as the basis of the SIP extension. We also built a simple application that uses the user agent to establish and terminate conferences. Performance measurements are made and preliminary results are obtained from the prototype. We compared our architecture with the full-meshed one and evaluated different split values.

Through the work, we can see that our signaling system is feasible to manage multiparty sessions in peer-to-peer ad hoc networks. Besides, in most cases, our system shows better performance than that found in the full mesh approach. Its worst performance is in the session termination when super members leave earlier than members do. Therefore, our scheme is not yet an optimal one. Moreover, the research on the optimization of the clustering architecture (e.g. optimally choose split value and merge value) is still at the start point of the road.

CHAPTER 8 CONCLUSIONS AND FUTURE WORK

In this chapter, we will first present the contributions of this thesis. Then we will discuss the lessons learned. Afterward, directions for future work will be presented.

8.1 Contributions and Lessons Learned

We have studied the evolutions and the related works in ad hoc networks and peer-to-peer communication. We also have related these two concepts and defined their usage in this thesis.

We have derived signaling requirements for the multiparty sessions in peer-to-peer ad hoc networks through comparing the networks with the traditionally infrastructure-based networks. Three existing signaling protocols: SIP, H.323 and ICEBERG have been studied as the state of the art.

In order to find an optimal signaling architecture for peer-to-peer ad hoc networks, we have analyzed four possible signaling architectures: centralization, full mesh, tree and clustering, and made comparison between them.

A clustering based signaling scheme has been proposed. We described the definitions, principles and detailed operations. We also analyzed its potential to meet the signaling requirements. We have implemented the signaling scheme as an extension to SIP. New entity, headers, concepts and semantics were proposed, behavior of end systems were depicted, and sequences of calls were illustrated.

A prototype has been built for implementing and evaluating the signaling system.

Preliminary results were then obtained from the performance measurement.

From all the works above, we have learned several lessons:

The first lesson is that the multiparty session control faces challenges in peer-to-peer ad hoc networks. The challenges include the peer-to-peer communication, the limited device and network resources, and the unreliable and transient network connectivity.

The second lesson is that none of the existing signaling protocols meet all the signaling requirements for peer-to-peer ad hoc networks. H.323 and ICEBERG signaling have the centralized control points. The SIP in full mesh cannot be scalable.

The third lesson is that our proposed signaling scheme: clustering-based architecture has potential to meet all the signaling requirements.

The fourth lesson is that SIP can be extended to implement the clustering-based architecture, and the extension can handle peer-to-peer multiparty sessions.

The fifth lesson is that the number of messages used in our architecture is less than that used in full mesh, but our scheme is not optimal because super-member leaving leads to bad performance.

The last lesson is that the research in this thesis is still a start point in the area of the signaling for peer-to-peer ad hoc networks. Significant amount of future work are ahead of us.

8.2 Future Work

The future work can be summarized as three directions.

The first direction is about the signaling scheme. Our signaling scheme needs to be optimized, especially for the sequence of super-member leaving. The research on cluster parameters' selection is also significant work. In addition, we have not yet touched the issues like security, fault recovery and billing in our signaling scheme.

The second direction is about the implementation and testing. We need to implement the resource negotiation mechanism so that the resources can be used optimally. We also need to open signaling APIs for application development. Furthermore, our implementation needs to be tested on various devices like PDAs, laptops and cell phones. It needs to be tested in a large scaled ad hoc network environment as well.

The last direction is about middleware. Although we use JXTA in the prototype, JXTA is not specifically designed for ad hoc networks. Middleware tailored for peer-to-peer ad hoc networks should be developed in the future.

REFERENCES

[1] M.Weiser, "The Computer for the Twenty-First Century", in Scientific American, 1991.

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

[3] T.Clausen, Ed. and P.Jacquet, Ed., "Optimized Link State Protocol (OLSR)", RFC 3626, Oct. 2003

[4] R.Ofier, F.Templin, M. Lewis, "Topology Dissemination Based on Reserve-Path Forwarding (TBRPF)", Feb. 2004.

[5] Stepnens. Yau and Xinyu Zhang, "A Middleware Service for Secure Group Communication in Mobile Ad hoc Networks", Proceedings of the 27th Annual International Computer Software and Application Conference (COMPSAC' 03), IEEE.

[6] Cristian Tuduce and Thomas Gross, "Organizing a Distributed Application in a Mobile Ad hoc Network", Proceeding of the Second IEEE International Symposium on Network Computing and Applications (NCA'03).

[7] J. Rosenberg et al., "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[8] H.323 series , ITU-T recommendations, Geneva 2003

[9] H.Schulzrinne et. al., "RTP: A Transport Protocol for Real-Time Applications", RFC1889, Jan. 1996

[10] H.Schulzrinne, GMD. Fokus, "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, Jan. 1996

[11] Ram Ramanathan and Jason Redi, "A Brief Overview of Ad hoc Networks: Challenges and Directions", IEEE Communications Magazine, 50th Anniversary Commemorative Issue, May 2002

[12] David Remondo, Ignas G. Niemegeers, "Ad hoc networking in future wireless communications", Elsevier Computer Communications 26, 2003, page 36-40

[13] Magnus Frodigh, Per Johansson and Peter Larsson, "Wireless ad hoc networking – the art of networking without a network", Ericsson Review No.4, 2000

[14] Bangnan Xu, Sven Hischks and Bernhard Walke, "The Role of Ad Hoc Networking in Future Wireless Communications", Proceedings of ICCT 2003

[15] Marco Conti, "Body, Personal, and Local Ad Hoc Wireless Networks", in: M. Ilyas (Ed.), Handbook of Ad Hoc Networks, CRC Press, New York, 2003 (Chapter I)

[16] IEEE 802.15.3 TM, "Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs)", IEEE Computer Society, Sep. 2003

[17] C.Bisdikian, "An Overview of Bluetooth Wireless Technology", IEEE Communications Magazine, Dec. 2001, page 86-94

[18] Pablo Brenner "A Technical Tutorial on the IEEE 802.11 Protocol", BreezeCOM Wireless Communication, 1997, online at: http://www.sss-mag.com/pdf/802_11tut.pdf

[19] ETSI, Digital Enhance Cordless Telecommunications (DECT); A High Level Guide

to the DECT Standardization, ETSI TR 101 178, Mar. 2000

[20] Kee Chaing and Seah Khoon Guan Winston, "Quality of Service Models for Ad Hoc Wireless Networks", in: M. Ilyas (Ed.), Handbook of Ad Hoc Networks, CRC Press, New York, 2003 (Chapter VII)

[21] Amitabh Mishra and Ketan M. Nadkarni, "Security in Wireless Ad Hoc Networks – A Survey", in: M. Ilyas (Ed.), Handbook of Ad Hoc Networks, CRC Press, New York, 2003 (Chapter VII)

[22] Ahmed Safwat, Hossam S. Hassanein and Hussein T. Mouftah, "Power-Aware Wireless Mobile Ad Hoc Networks" in: M. Ilyas (Ed.), Handbook of Ad Hoc Networks, CRC Press, New York, 2003 (Chapter VII)

[23] IETF Peer-to-Peer Research Group, online:http://www.irtf.org/charters/p2prg.html

[24] Barry M. Leiner et al., " A Brief History of the Internet", online at: http://www.isoc.org/internet/history/brief.shtml#LGR67

[25] Edited by Andy Oram, "Peer-to-peer: harnessing the benefits of a disruptive technology", Beijing ; Sebastopol, CA : O'Reilly, 2001

[26] The Napster Protocol, online at: http://opennap.sourceforge.net/napster.txt, Mar. 2000

[27] Matei Ripeanu, Adriana Iamnitchi, Ian Foster, "Mapping the Gnutella Network", in IEEE Internet Computing, Jan.--Feb. 2002, pages 50-57

[28] Ian Clarke, Scott G.Miller, Theodore W. Hong, Oskar Sandberg, Brandon Wiley, "Protecting Free Expression Online with Freenet", in IEEE Internet Computing, Jan.--Feb. 2002, page 40-49

[29] JXTA organization, online at: http://www.jxta.org/

[30] Li Gong, "JXTA: A Network Programming Environment", in IEEE Internet Computing, May—June 2001

[31] R. Schollmeier, "A Definition of Peer-to-Peer Networking for Classification of Peer-to-Peer Architectures and Applications", in First IEEE International Conference on Peer-to-Peer Computing, 2001, pages 101-102

[32] G. Kortuem, "When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad Hoc networks", n First IEEE International Conference on Peer-to-Peer Computing, 2001, pages 75-91

[33] Siddhartha K. Goel et al., "Efficient Peer-to-Peer Date Dissemination in Mobile Ad-Hoc Networks", IEEE Proceedings of the International Conference on Parallel Processing Workshops (ICPPW' 02)

[34] R. Schollmeier et al., "Routing in Mobile Ad Hoc and Peer-to-Peer Networks. A Comparison", online at http://www.elet.polimi.it/p2p/papers/1.pdf

[35] Barbosa e Oliveira et al., "Evaluation of ad-hoc routing protocols under a peer-to-peer application", in IEEE Wireless Communications and Networking (WCNC), March 2003, Volume: 2, pages 1143-1148

[36] Helen J. Wang, et al., "Iceberg: An Internet Core Network Architecture for Integrated Communications", in IEEE Personal Communications, August 2000

[37] Santivanez, C.A.; McDonald, B.; Stavrakakis, I.; Ramanathan,R., "On the scalability of ad hoc routing protocols", INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings,

IEEE, Volume: 3, 23-27 June 2002

[38] Feng Yong-Xin, et al., "A clustering algorithm applied to the network management on mobile ad hoc network", Info-tech and Info-net, 2001, proceedings, ICII 2001-Beijing, 2001 International Conference, Volume 2, 29 Oct. - 1 Nov. 2001.

[39] Ram Ramanathan, Martha Steenstrup, "Hierarchically -organized, multi-hop mobile wireless networks for quality-of-service support" Mobile Networks and Applications 3, 1998

[40] R. Sparks, "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003

[41] S. Donovan, "The SIP INFO Method", RFC 2976, Oct. 2000

[42] A. B. Roach, "Session Initiation Protocol (SIP) – Specific Event Notification", RFC 3265, June 2002

[43] J. Rosenberg, H. Schulzrinne, "Models for Multi Party Conferencing in SIP", IETF Internet Draft, Nov. 2000, expires: May 2001

[44] Mark/Kelley, "Distributed Multipoint Conferences using SIP", IETF Internet Draft, Mar. 2000, expires: Sep. 2000

[45] JAIN SIP Sources online at: https://jain-sip.dev.java.net/

[46] "This document specifies the JAIN^{TM} SIP API Specification v1.1", online at: http://www-x.antd.nist.gov/proj/iptel/src/nist-sip/jain-sip/docs/api/jain-sip-overview.html, June 2003

[47] Hwa-Chun Lin, Yung-Hua Chu,"A clustering technique for large multihop mobile wireless networks", Vehicular Technology Conference Proceedings, 2000 IEEE 51st, Volume: 2, 15-18 May 2000

[48] Chien-Chung Shen et al., "CLTC: a cluster-based topology control for ad hoc networks", Mobile Computing, IEEE Transactions on, Volume: 3, Issue: 1, Jan.-March 2004

[49] Khlifi, H., Agarwal, A., Gregoire, J.-C., "A framework to use SIP in ad-hoc networks", Electrical and Computer Engineering, IEEE CCECE 2003. Canadian Conference on , Volume: 2 , 4-7 May 2003

Appendix A: Syntax of Header Fields: Conf-ID, Participant-In-Cluster, Cluster-Parameters and Neighbor

Conf-ID	"Conf-ID" ":" Conference id
Conference id	token ["@" token]
Participant-In-Cluster	"Participant-In-Cluster" ":" Cluster-params
Cluster-params	cluster id ";" super member ";" members
cluster id	"cluster id=" token ["@" token]
super member	"super member=" SIP-URL
members	"members=" (SIP-URL *["," SIP-URL])
Neighbor	"Neighbor" ":" Cluster-params
Cluster-Parameters	"Cluster-Parameters" ":" Cparams
Cparams	split-value ";" merge-value ";" level-value
split-value	"Sv=" 1*5DIGIT
merge-value	"Mv=" 1*5DIGIT
level-value	"Lv=" 1*5DIGIT

Appendix B: Alice's Configuration Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:PA>
<jxta:PA xmlns:jxta="http://jxta.org" type="jxta:PlatformConfig">
    <PID>
    urn:jxta:uuid-59616261646162614A787461503250335DDD64F187764299BDA49795B0
F2552603
    </PID>
    <Name>
        Alice
    </Name>
    <Desc>
        Platform Config Advertisement created by : net.jxta.impl.peergroup.Configurator
    </Desc>
    <Svc>
        <MCID>
            urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000905
        </MCID>
        <Parm>
            <jxta:TransportAdvertisement xmlns:jxta=http://jxta.org
            type="jxta:TCPTransportAdvertisement">
                <Protocol>
                    TCP
                </Protocol>
                <Port>
                    9701
                </Port>
                <MulticastAddr>
                    224.0.1.85
                </MulticastAddr>
                <MulticastPort>
                    1234
                </MulticastPort>
                <MulticastSize>
                    16384
                </MulticastSize>
                <ConfigMode>
                    auto
                </ConfigMode>
            </jxta:TransportAdvertisement>
        </Parm>
    </Svc>
    <Svc>
        <MCID>
            urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000605
        </MCID>
        <Parm>
        </Parm>
    </Svc>
    <Svc>
        <MCID>
            urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000E05
        </MCID>
        <Parm>
            <isOff>
            </isOff>
        </Parm>
    </Svc>
    <Svc>
        <MCID>
            urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000805
        </MCID>
        <Parm>
            <MessengerQueueSize>
                20
            </MessengerQueueSize>
        </Parm>
    </Svc>
    <Svc>
        <MCID>
            urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000105
        </MCID>
        <Parm>
            <RootCert>
    MIICNjCCAZ+gAwIBAgIBATANBgkqhkiG9w0BAQUFADBjMRUwEwYDVQQKEwx3d
3cuanh0YS5vcmcxCzAJBgNVBAcTAlNGMQswCQYDVQQGEwJVUzERMA8GA1UEAxMIQ
WxpY2UtQ0ExHTAbBgNVBAsTFDY1QjhGMDIxMTFERTU3QTJGMzc2MB4XDTA0MDcw
NjE0MTU1MFoXDTE0MDcwNjE0MTU1MFowYzEVMBMGA1UEChMMd3d3Lmp4dGEub3J
nMQswCQYDVQQHEwJTRjELMAkGA1UEBhMCVVMxETAPBgNVBAMTCEFsaWNlLUN
BMR0wGwYDVQQLExQ2NUI4RjAyMTExREU1N0EyRjM3NjCBmzALBgkqhkiG9w0BAQE
DgYsAMIGHAoGBAIjWaetCNLyJ7CcVAcM2kcz3gPmnOVX26Lhw1PV4FeWH1UhsjSoR/v/Y
9BEkhWNKTiNpmCUfKC9z/fISgTjdq3sZ6KySFPumHJBX+1Xe3x+2MK4PKe4JmGLiGfYLf0
E+Hk1WezKwEPObZ7+1UIvM4MeRmP/rfLO2J/wwkGuvTfbtAgERMA0GCSqGSIb3DQEBBQ
UAA4GBAGEK8yEUT0SUu+XjwybhozBLMXszNK2z5IrE5WGL7C11Ua0t7V9E2GoIB0UA7p
```

```
Oli5xKqMhKYcFnQDjFy5/kj37dgOFnMd5fFgya6OquWsvuMsfzNLPlFDYie6WYWiiqj/jiR7btk
Wpf/jXBCRcI7ETceSjW2IjLSWCvqN2VEVJH
          </RootCert>
      </Parm>
   </Svc>
   <Svc>
      <MCID>
          urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000A05
      </MCID>
      <Parm>
          <jxta:TransportAdvertisement xmlns:jxta="http://jxta.org"
type="jxta:HTTPTransportAdvertisement">
             <Protocol>
                http
             </Protocol>
             <InterfaceAddress>
             </InterfaceAddress>
             <ConfigMode>
                auto
             </ConfigMode>
             <Port>
                9700
             </Port>
             <Proxy>
                myProxy.myDomain:8080
             </Proxy>
             <ProxyOff>
             </ProxyOff>
          </jxta:TransportAdvertisement>
      </Parm>
   </Svc>
   <Svc>
      <MCID>
          urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000F05
      </MCID>
      <Parm>
          <isServer>
             false
          </isServer>
          <ServerMaximumClients>
          </ServerMaximumClients>
          <ServerLeaseInSeconds>
          </ServerLeaseInSeconds>
          <isClient>
```

```
                    false
                </isClient>
                <ClientMaximumServers>
                </ClientMaximumServers>
                <ClientLeaseInSeconds>
                </ClientLeaseInSeconds>
                <ClientQueueSize>
                    20
                </ClientQueueSize>
                <isOff>
                </isOff>
            </Parm>
        </Svc>
        <Dbg>
            user default
        </Dbg>
    </jxta:PA>
```

Appendix C: Alice's Peer Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:PA>
<jxta:PA xmlns:jxta="http://jxta.org" type="jxta:PlatformConfig">
    <PID>
    urn:jxta:uuid-59616261646162614A787461503250335DDD64F187764299BDA49795B0F2
552603
    </PID>
    <Name>
        Alice
    </Name>
    <Desc>
        Platform Config Advertisement created by : net.jxta.impl.peergroup.Configurator
    </Desc>
    <Svc>
        <MCID>
            urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000905
        </MCID>
        <Parm>
            <Addr>
                tcp: //132.146.3.82:9071/
            </Addr>
            <Addr>
                jxtatls: //uuid-
        59616261646162614A787461503250335DDD64F187764299BDA49795B0F2552603/
        TlsTransport/ jxta-WorldGroup
            </Addr>
            <Addr>
                jxta: //uuid-
        59616261646162614A787461503250335DDD64F187764299BDA49795B0F2552603/
            </Addr>
            <Addr>
                http: // JxtaHttpClientuuid-
        59616261646162614A787461503250335DDD64F187764299BDA49795B0F2552603/
            </Addr>
        </Parm>
    </Svc>
</jxta:PA>
```