HELSINKI UNIVERSITY OF TECHNOLOGY

Faculty of Electronics, Communications and Automation

Department of Communications and Networking

Name of the author: Lu Xiaojun

Title of the thesis: Disruption Tolerance for SIP

Thesis submitted in partial fulfilment of the requirement for the degree of Master of Science in Technology in Espoo, Finland, 5th May 2008.

Supervisor: Professor Jörg Ott (Networking Laboratory)

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT of the Master's thesis

Author:	Lu xiaojun		
Name of the thesis:	Disruption Tolerance for SIP		
Date:	5th May 2008	**Number of Pages:**	76
Faculty:	Faculty of Electronics, Communications and Automation		
Professorship:	Networking Technology		
Supervisor:	Jörg Ott		

The wireless networks have been built with versatile wireless network technology, including both wide area wireless networks and local area wireless networks. In such heterogeneous network environment, mobile users may experience either short or long interruption for different reasons while having a multimedia conversation. A lot of emphasis is concentrated on improving radio signal and enhancing seamless handover. However, recovery and backup multimedia conversation from a temporary network failure is also an interesting topic to be discussed.

In this thesis, a SIP-based communication with enabling of disruption tolerance mechanism is introduced. We present the idea of media stream and SIP signaling based detection and recovery mechanisms, and come with an implementation of the prototype. The disruption tolerance functions include the ways of detecting network failure, storing the conversation during the meanwhile of the network disconnection, recover the previous broken multimedia session and replay the unheard voice. A brief experimental SIP network is built to evaluate the disruption tolerance functions of the software prototype. The result of the experimentation shows that the multimedia session can be recovered from the broken session in a short time, and the important conversation will not be lost during the short network disconnection. The replayed voice brings a delay to the recovered conversation which is not good experience for the users. However, the delayed conversation is much better than losing the conversation.

Keywords: SIP, disruption tolerance

ACKNOWLEDGEMENTS

This Master’s thesis was written for the Networking Laboratory at TKK (Helsinki University of Technology), Finland, under the supervision of Professor Jörg Ott.

I wish to express my most sincere and grateful thanks to Professor Jörg Ott for giving me the opportunity to work under his supervision. His guidance and encouragement during this thesis was one invaluable treasure of my studies at TKK.

Additionally, I appreciate the support from our laboratory staff that provided me with a comfortable and pleasant working environment. I also give many thanks to my good friend Jocy and writing clinic teacher William Martin for their proofreading of this thesis document.

Finally, I would like to express my gratitude to my family. Their encouragement and support was the source of my determination to complete my studies.

Helsinki University of Technology

Lu Xiaojun

Table of Contents

Abbreviations and Acronyms

SIP	Session Initiation Protocol
UMTS	Universal Mobile Telecommunications System
LAN	Local Area Network
WLAN	Wireless LAN Network
IP	Internet Protocol
VoIP	Voice over IP
DHCP	Dynamic Host Configuration Protocol
GPRS	General Packet Radio Service
CDMA	Code Division Multiple Access
PPP	Point to Point Protocol
SDP	Session Description Protocol
HTTP	Hypertext Transfer Protocol
SMTP	Simple Mail Transfer Protocol
IETF	Internet Engineering Task Force
3GPP	3rd Generation Partnership Project
IMS	IP Multimedia Subsystem
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
B2BUA	Back-to-Back User Agent
URI	Uniform Resource Identifier
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
BNF	Backus-Naur Form grammar
UTF-8	8-bit Unicode Transformation Format
MIME	Multipurpose Internet Mail Extensions
QoS	Quality of Service

AAA	Authentication, Authorization and Accounting
RTP	Real-time Transport Protocol
RTCP	Real-time transport control protocol
PBX	Private Brach exchange
BIND	Berkeley Internet Name Domain
JMF	Java Multimedia Framework
UML	Unified Modeling Language
GUI	Graphic User Interface
API	Application Programming Interface
DTMF	Dual-Tone Multi-Frequency

1. Introduction

The demand for wireless access is increasing with a lot of mobile network technology being invented. The stability and connectivity are the most notable challenges for wireless networks. When a mobile user accesses a wireless network, there are several reasons which may lead to loss of connection. For example, the user passes through an area with low signal coverage, the signal is attenuated, or unpredictable interference or overload occurs. In addition, seamless roaming through different wireless networks is difficult to realize. In the above cases, the mobile user may experience network disconnection for a short while. It is not easy for wireless network service providers to enhance wireless network coverage for both economic as well as technical reasons: both the cost of building and the complexity of maintaining a ubiquitous network are nowadays too high. An alternative solution is to develop the technology of disconnection tolerance, which is the purpose of this master thesis. This solution cannot substitute for broader range network coverage, but serves as a complementary solution to improve usability.

1.1 Mobility services

Mobility services are defined as a bundle of mobility aspects for heterogeneous networks. In general, the mobility services can be placed into four categories: personal mobility, terminal mobility, session mobility and service mobility. Knowledge of universal mobility services is necessary to understand this thesis work. This section gives a brief introduction to the concept of mobility services.

1.1.1 Personal mobility

Personal mobility refers to the ability of a mobile user to access mobile service anywhere. The mobility service is based on the user’s personal identifier, no matter what kind of mobile device is being used. The network capability relies on this user’s service profile.

1.1.2 Terminal mobility

Terminal mobility refers to the mobility of mobile devices, and may require several network interfaces for a mobile device to access different mobile networks. A mobile user can use the mobile device to move across heterogeneous networks while having the same subscribed mobility service. The mobile device takes care of the accessing network which is based on an access policy, e.g. signal strength or network bandwidth. The terminal mobility relates to the concept of handover or the transfer between networks. The ideal handover is called seamless handover [1], which minimizes packets loss and delay. There are two types of handover mechanisms: hard handover and soft handover. Hard handover breaks the current network connectivity first, and then builds a new network connection. Soft handover makes a new network access before terminating the current network connectivity. The mechanism of hard handover is much simpler than soft handover, but causes packets loss. The mobile device may receive the same packets from different network connective sources during soft handover, so it needs to merge those duplicated packets. This makes soft handover more complicated in cases of merging those asynchronous duplicated packets, but the benefits is no packets are lost.

1.1.3 Service mobility

Service mobility means a mobile user can consistently access personalized mobility services from the network services provider. The quality of service depends on the user’s service profile. The service mobility requires a set of central servers to store the user’s profile, provide registration service and keep trace of the user’s current location.

1.1.4 Session mobility

Session mobility is defined as maintaining an active session while switching between different network contexts. The aspect of switching the network context can be classified to horizontal handover and vertical handover. *The horizontal handover involves mobile nodes moving between access points of the same type (in terms of coverage, data rate and mobility), such as UMTS to UMTS, or WLAN to WLAN. The vertical handover involves mobile nodes moving between access points of different type, such as UMTS to WLAN.* [1] In most cases, the horizontal handover is much easier to be handled than the vertical handover for the session mobility. For example, a session switches between two wireless LANs may only change the IP addresses of the terminals, but the session switches from a WLAN to a UMTS network may change the media stream type and the transport protocol stack.

1.2 Motivation

A brief introduction of the mobile services has been given in the previous section. This thesis emphasizes the aspect of the disruption tolerance mechanism for mobile networks, which is related to maintain a call session context if the call is broken unexpectedly. Session mobility is the main concept used in this thesis as the disruption tolerance mechanism. The disruption tolerance mechanism also implicitly refers to personal mobility, terminal mobility as well as service mobility. This section continues with the motivation for this thesis.

VoIP (Voice over IP) technology is usually used as a low cost International call solution, as a pure VoIP call[1] is built on Internet resources without occupying the resources of the telephone networks or the cellular networks, making a long distance

[1] A pure VoIP call means both caller and callee are using an Internet resource for a call. Of course, if either participate uses a normal telephone system, it requires extra resources from the telecommunication networks.

call extremely cheap. For most mobile users, the cost of using the satellite or the cellular networks is still expensive, and therefore the low-cost autonomous wireless networks are more attractive to ordinary mobile users. Using local wireless networks for VoIP calls is a new value added solution for mobile users. Modern mobile devices may contain several different wireless network interfaces for achieving better connectivity. Therefore, any available wireless network resources can be used for the VoIP calls.

SIP (Session Initiation Protocol) [2] is a signaling protocol used to establish and tear down multimedia stream sessions between two or more participants. SIP has mobility features, and a brief introduction of SIP mobility is described in Chapter 2. As a signaling protocol, SIP has already been successfully used for building VoIP systems.

When a mobile user is using a SIP-based soft phone in heterogeneous wireless network environments, communication may be occasionally interrupted without explicit warning. For instance, any call participants may lose connectivity and drop the call in cases of weak connection signal or when crossing an area with poor coverage. In IP networks, it always takes time for the mobile device to acquire an IP address from the DHCP server[1]. While switching between different accesses networks, hard handover may lose connection temporarily for acquiring a new IP address. It also happens to soft handover when the mobile device is out of the signal coverage range of the old access network before the new access network is acquired.

A solution to the temporary disconnection challenges described above should be explored. The existing conversation shall be switched back and forth between the synchronous (real-time communication) and the asynchronous model (voice message based communication). If an ongoing call is robust enough to accept intermittent connectivity, the mobility aspect of the SIP-based communication is enhanced for

[1] In GPRS or CDMA networks, the acquisition method is PPP. It also takes time to get the access address.

those unstable wireless network environments. The design idea of the disruption tolerance mechanism is to temporarily keep the session context and the media content of the current ongoing call. Whenever a network disconnection occurs to break the current ongoing call, the broken call should be held on the mobile device and be recovered later automatically.

This thesis work builds upon the existing Open Source code to prove the disruption tolerance mechanism conceptually as well as to implement an intermittent connectivity prototype for SIP-based communication. The implementation of the prototype is based on an Open Source project - *SIP communicator*, which is a full java based SIP client that implements basic SIP functions following RFC 3261 [2]. These functions of the disruption tolerance are implemented as enhancement modules for the SIP communicator.

1.3 Outline of this thesis

The main task of this thesis is to implement a SIP-based communication software prototype incorporating with the disruption tolerance mechanism. Furthermore, an experimental SIP network has been built to perform testing against the reference implementation.

This thesis is structured as follows: The second chapter presents the basic knowledge of the Session Initiation Protocol. A brief introduction of the technical background is given in the third chapter. The fourth chapter describes the design architecture and analyzes each use case in detail. The implementation details are discussed in the fifth chapter. The sixth chapter explains an overview of testing and validation of the software prototype. The final conclusions of this thesis are presented in the last chapter. More detailed information refers to the thesis document which provides guidelines to build and configure the running environment are presented in the appendices.

2. Introduction to SIP-based multimedia

This chapter provides the basic knowledge of SIP (Session Initiation Protocol) [2]. The first section gives a brief introduction of the SIP history. The second section describes the terminologies and components of SIP networks. Basic knowledge of SIP is required to understand the implementation of this thesis work. The third section presents the SIP registration and normal call process, while in the fourth section SIP media session and SDP [3] are introduced. The fifth section explains SIP for mobility services and the final section summarizes the main point coverage this chapter.

2.1 History

A lot of applications of the Internet and the telecommunication industry require a signaling protocol to create and manage a session, as well as control the data exchanging between the end points of participants. The Session Initiation Protocol (SIP) [2] is designed for the purposes of soft switching technology, for providing versatile functions like building a single call session or multi-parts call conference, forwarding a call from a predefined user profile or forking a call to several places, providing instant message services and so on. SIP borrows a lot of ideas from other mature protocols like the Hypertext Transfer Protocol (HTTP) and the Simple Mail Transfer Protocol (SMTP). All of the above protocols are string based protocols meaning the content of protocol is readable by both machines and humans.

The first draft of SIP was called Session Invitation Protocol, which was published by IETF (Internet Engineering Task Force) on February 22, 1996. Later in the first publication draft on December 2, 1996, however, Session Initiation Protocol replaced the original name. Afterwards, there were twelve drafts of SIP published by IETF between 1996 and 1999. On March 17, 1999, RFC 2543 was eventually published by IETF. Three years later, the first version of SIP RFC was improved upon by the second version of RFC 3261, which was published on July 3, 2002. A lot related and enhanced specifications have also been published by IEFT since the RFC 2543 was published. In November 2000, the 3rd Generation Partnership Project (3GPP) accepted SIP as the signaling protocol and design architecture of IP Multimedia Subsystem (IMS).

2.2 Overview of SIP

This section describes the basic knowledge of SIP that is used in this thesis. The first and second parts discuss logical components in the SIP network, User Agent (UA) and Network server. The third part outlines the protocol structure and the details of SIP message construction are described in the fourth part.

2.2.1 User Agent of SIP

A SIP transaction comprises all messages which start from the first SIP request from the client, and end with the last SIP response from the server. A UA acts as the User Agent Client (UAC) when it is creating request, and the role of the UAC lasts until the end of the SIP transaction. When a UA is replying to a request from another UAC, it acts as the User Agent Server (UAS) in this transaction. The definitions of UAC and UAS are slightly different compared to that of the normal client and server. For example, in the web service, a web browser is called the client, and on another side the machine which generates the HTTP response is called the server.

The behavior between two user agents is represented by dialog. *The dialog facilitates sequencing of messages between the user agents and proper routing of requests between both of them.* [2] The SIP dialog may contain of one or more transactions to persist a conversation between the user agents. The identifier of a dialog is consisted of a CALL-ID, a local tag and a remote tag, which ensures a unique dialog identifier. The dialog identifier has the same value for both the UAC and the UAS. When a specified request method receives a non-failure response, a dialog is created for this request method. A dialog can also be generated in an early state if a provisional response has been received. After a dialog is created, it contains several pieces of different states. These states indicate the further behavior of both UAC and UAS, the details of which are defined in RFC 3261. [2]

2.2.2 SIP server

There are several different types of servers in SIP networks:

- **Proxy server**

  The proxy server is a logical SIP entity that acts as a message router. A Proxy has the capability of resolving SIP addresses and making the "routing decision" of next hop. A proxy only handles the requests which are under its ability to send responses. If the request message contains any error or needs security authentication, the proxy may respond with a corresponding error code. If a request routes through a sequence of proxies, the backward response will traverse through those proxies in reversed order. A proxy server can be running in either stateful or stateless mode. A stateful proxy server creates transactions for the incoming request, and maintains the transaction until a final response is received. A stateful proxy is mandatory to fork a request to multiple destinations. A stateless proxy works in a simple way by just forwarding requests to downstream destinations and sending responses to upstream destinations.

- **B2BUA**

  In SIP networks, a UA contains both UAC and UAS which is called B2BUA (back-to-back user agent). In a particular transaction, a B2BUA acts as a UAC or a UAS independently. A B2BUA usually acts as an intermediary entity that forwards SIP messages from a SIP client to a SIP server. Compare to a SIP proxy which only takes responsibility for routing function, a B2BUA acts more powerfully by functioning as an intermediate SIP server which contains the ability to modify the forwarded SIP messages.

- **Redirect server**

A redirect server is designed to provide routing information to a UA, which receives routing requests and generates 3xx class responses. The response contains the binding information from a SIP URI to an exact network address. If the redirect server and the location server are separated components, the redirect server queries the location server to obtain knowledge of the user's location.

- **Location server**

A location server is a network component which provides a callee's possible location(s) information to a SIP redirect server or proxy server. The location server may bind zero or more contact addresses to a user, and these bindings can be updated by the registration or by other (non-SIP) ways.

- **Registrar**

A registrar is a special UAS that receives REGISTER requests that creates a mapping of address-of-record to a set of contact addresses. An address-of-record is a SIP URI (it is quite similar to an e-mail address), which is combined by personal user information and a specified domain name. If there is no location server in the SIP network, the registrar is the only one which keeps the knowledge of a user's location and shares the same data with other SIP proxies and redirects servers in the same domain. The registrar co-operates with proxy servers and redirect servers to offer a full routing discovery capability for SIP network.

2.2.3 Protocol structure

SIP uses the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) as underlying protocols to convey protocol data units between servers and endpoints. In the designed protocol structure, SIP is logically a three layered protocol. The lowest layer defines the syntax and encoding of SIP. SIP specifies augmented Backus-Naur Form grammar (BNF) as its encoding method. The second layer defines the way of SIP message transportation, and it is called transport layer. SIP can use different underlying transport protocols to transfer messages over networks. The mechanism of lower layer transport protocols is transparent to SIP. The third layer is the transaction layer where the meaning of transaction is quite similar to that of HTTP, each transaction being a bundle of predefined requests and responses. A request from the UAC invokes a particular function of the UAS, and later the UAS responds to the request. One transaction refers to the interactive handshake which starts from the request sent from the UAC until the end of the response corresponding to the request.

2.2.4 SIP Messages

SIP uses the UTF-8 encoding to represent the text-based messages. One obvious benefit of a text-based protocol is to be readable by human beings, and it is easy to be tested manually. The protocol can be verified easily by reading the protocol messages instead of writing complicated testing programs. This feature improves many debugging processes and determining the potential errors of the protocol. Another benefit of the text-based protocol is that it is easy to be extended to include new features. This brings extra value to SIP when used in different environments, and makes it easy to define extension messages for SIP. The SIP messages can be separated into two types: request and response. A request is from the UAC to the UAS and a response is from UAS to UAC. All of the SIP messages use Internet Message Format.[17]

A SIP request starts with a Request-line:

Request-line = Method + Single space + Request-URI + SIP version + CRLF

For example, a request-line for an INVITE request is presented as the following:

INVITE sip:example@netlab.hut.fi SIP/2.0

The following table lists six methods which are defined in RFC 3261 [2]. Other methods are also defined in the extensive documentation related to SIP.

SIP method	Description
INVITE	Invites another user to start a call session.
BYE	Says good bye to another user to end a call session.
OPTIONS	Queries the capabilities of the server.
ACK	Final acknowledgement to an INVITE request.
REGISTER	Registers a user agent to a SIP registrar, the registrar keeps the knowledge of the user agent's location.
CANCEL	Cancels a previous INVITE request.

Table 2-1 SIP methods

SIP uses URIs to identify users. The SIP request URI has the following format:

Request URI = SIP/SIPS scheme:user:password@host:port;uri-parameter?headers

"sip:" is the SIP scheme, it refers to a SIP URI. Like the difference between "http:" and "https" schemes, "sips:" refers a security URI which is called SIPS scheme. In both SIP and SIPS URIs, only the scheme and host are mandatory. The SIP version defines the version number of the protocol, currently it must be "SIP/2.0" format.

A SIP response always starts with a status-line which is distinguished from a SIP request:

Status-line = SIP-version + Single space + Status-code + Single space + Reason-phrase + CRLF

The status-code is represented by a three digit number, and there are six classes of status-code defined in RFC 3261. [2]

- **1xx:** Provisional – This status-code means that the request was received by the proxy or server, and that the request processing is in progress. Usually, when a request needs to wait for more than 200ms, the status-code is sent to the UAC as a temporary response.

- **2xx:** Success – The success status-code is sent to the UAC after a request is received, parsed, processed and accepted by the UAS.

- **3xx:** Redirection – The redirection response gives the user's new location or information of alternative services which the UAC queried. Any redirection response must not contain the address which was already listed in the "Via" header field, otherwise a loop could be caused. On the other hand, the UAC should check the address from the redirection response, and make sure the redirection address was not routed before.

- **4xx:** Client Error – The client error response means the request cannot be processed by the server. The reason could be a malformed request or an unauthorized request. The UAC should receive the error reason from the error response, and then send a correct request to the server. The same request could be successful depending on the server's capability.

- **5xx:** Server Error – When the request is valid but it cannot be processed by the server, a server error response is returned. In this situation, the UAC should try the same request to another possible server.

- **6xx:** Global Failure – The global failure is the highest level failure response, which means the request will be discarded or not accepted anywhere. After receiving the global failure response, the UAC should stop sending the same request to any server.

Both SIP request and response contain at least one header field, the set of header fields ending with an empty line. SIP does not specify the sequence of those header fields, but they are mostly put in the same order used in the protocol specification. The format of header fields conforms to RFC 2234 [18] (new draft standard is RFC 4234 [19]). If a header field only appears in a request, it is called a request header field. Following the same rule, a response header field only makes sense of a response message. A UA must ignore these unsupported header fields when a message is processed. A minimum set of header fields of a SIP message must contain at least six header fields: “To”, “From”, “CSeq”, “Call-ID”, “Max-Forwards” and “Via”.

Header field	**Description**
To	Indicates the logical recipient or the address-of-record target in the request.
From	Indicates the logical initiator’s identity.
CSeq	The identifier of an ordered transaction.
Call-ID	For the purpose of grouping a series of SIP messages, Call-ID presents a unique identifier of the group messages.
Max-Forwards	Limits the maximum number of hops of a request.
Via	Records the path used for the transaction, the response is sent back following the routing information from the “Via” header.

Table 2-2 Mandatory SIP header fields

A message-body is an optional part to be added after the header fields. Usually a message-body encapsulates other protocols or conveys short messages. The content type must be declared in the Content-Type header field with the character set as an optional value when a specified encoding method is used in the message body. If the message body contains multiple parts in a message, MIME [20] is used to describe the message detail. The length of the message is counted as bytes that are introduced in the Content-Length header field. SIP may use an unreliable transport protocol to transfer message, so the Content-Length header field must be presented to indicate the end of the message.

2.3 SIP operations

The basics of SIP [2] have been introduced in the previous section. This section describes the basic SIP operations of registration, call setup and teardown. The SIP messages of the above operations are shown in message sequence charts. A message sequence chart is a graphical language for the description and specification of the signaling between system components.

To simplify the presumption of the use cases, only two clients are used to represent the call sessions in this thesis. An investigation of multipart call sessions is beyond the scope of this document. The SIP clients are named as client A and client B. The signaling between client A and client B are sorted in an order based on a vertical time line. The distance between the two signals is not in a ratio of real time.

2.3.1 Registration

This section describes the registration operation of SIP. SIP client A sends a REGISTER request to the SIP registrar, and then the registrar replies with a 200 OK response to SIP client A. The registration operation is an element of the node discovery mechanisms of SIP. The binding is created in registration for a particular domain that associates the address-of-record URI of user A with contact addresses.[1] The SIP registrar will keep SIP client A's contact addresses, and these contact addresses can be updated via new registrations. Therefore, if a SIP proxy receives a request whose Request-URI header field matches user A's address-of-record URI, the proxy will query the registrar to get SIP client A's contact address, and then forward the request to SIP client A. The entire registration process is depicted in Figure 2-1.

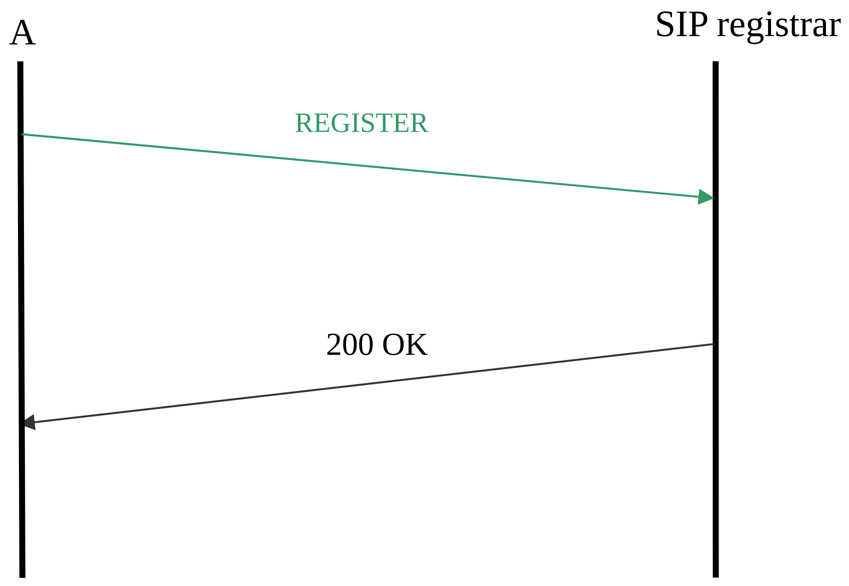


Figure 2-1 Registration

[1] The user's address-of-record URI can be associated with more than one contact address. The forking mechanism of SIP is not discussed in this thesis.

2.3.2 Normal call process

Figure 2-2 demonstrates a call setup and teardown process between SIP clients A and B. User A dials up user B’s SIP URI. SIP client A sends an INVITE request containing a SDP [3] message through the SIP server to SIP client B. The SIP server forwards the INVITE request from SIP client A to SIP client B, and then sends back a temporary 100 TRYING response to SIP client A. SIP client B receives the INVITE request from SIP client A and alerts user B to answer the phone call. Also, SIP client B replies to SIP client A with a 180 RINGING response, the response message is routed back along the reverse path of the INVITE request routing path. After user B picks up the SIP phone, SIP client B sends a 200 OK response which contains a SDP message to SIP client A. After SIP client A receives the 200 OK response from SIP client B, both SIP clients A and B have learned the endpoint IP address and media description from each other. An ACK message is sent from SIP client A to SIP client B to acknowledge the session is built. A media stream between SIP client A and SIP client B is built up, user A starts to have a conversation with user B. At the end of conversation between user A and user B, user B hangs up the SIP phone first. A BYE request is sent from SIP client B to SIP client A through the SIP server. A 200 OK response from SIP client A to SIP client B indicates the whole call is finished.

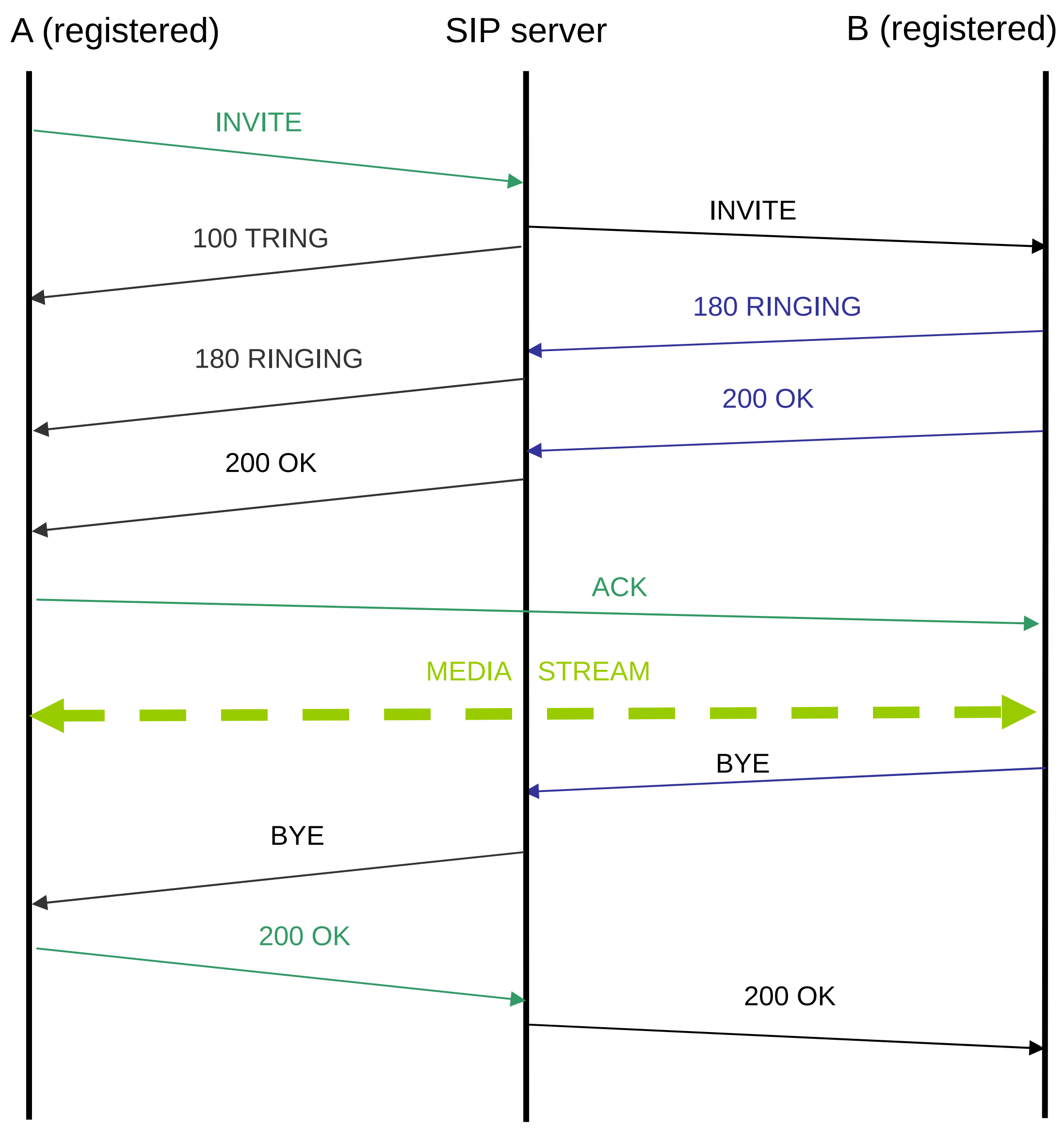


Figure 2-2 Call setup and turnoff

2.4 SIP media session

In the previous section we have discussed the basic operations of SIP. In the example of the setup and teardown of a call, the INVITE request conveys a SDP description for compatible media types. A brief introduction of SDP (Session Description Protocol) [3] is given in this section.

SDP is a standard description protocol of media content and how multimedia information is transported, which supports the negotiation of session content and media format. SDP must contain sufficient information to enable the session participants to understand each other. A session description includes session name and purpose, timing information, media comprising, the recipient's information [1], network bandwidth of the session as well as contact information. Like SIP, SDP is entirely text-based protocol which uses UTF-8 encoding. A SDP message consists of a series of lines of string in this form:

<type>=<value>

The <type> is one case sensitive character that represents the description type, and the <value> defines a formatted text that is the value of the description type. The details of the SDP types and values are described in [3].

Table 2-3 describes the basic SDP types which are used in this thesis.

Type name	**Description**
v=	Protocol version
o=	Originator and session identifier
s=	Session name
c=	Connection information
t=	Time the session is active
m=	Media name and transport address
a=	Zero or more media attribute lines

Table 2-3 SDP description

[1] The recipient's information includes IP address, ports, format, etc.

An example SDP description is presented as the following:

```
v=0
o=test 0 0 IN IP4 192.168.5.101
s=-
c= IN IP4 192.168.5 .101
t=0 0
m=audio 22224 RTP/AVP 18
m=video 22222 RTP/AVP 31
a=recvonly
```

SIP uses SDP to describe session media types, and SDP is also useful in SIP mobility services. A new INVITE request can be sent during a call session to inform a remote participant of the changed local transport addresses, media or codec, usually referred to as a re-INVITE request. SIP for mobility services are described in the next section.

2.5 SIP for mobility support

SIP is already designed for mobility services. SIP is an application layer signaling protocol, so the mobility support using SIP also refers to an application layer mobility management scheme. Chapter 1 has given a brief introduction of different types of mobility services, and this section continues to discuss SIP for mobility services.

SIP supports personal mobility very well. The URI scheme of SIP can be used as a personal identifier of the mobile user. The registration mechanism of SIP ensures multiple terminals can be registered at the same time. An incoming call will be forked to multiple terminals, and the current active terminal determined dynamically by the user’s location.

Terminal mobility always refers to handover between different networks. The solution for terminal mobility has two aspects that are called pre-call mobility and mid-call mobility. The pre-call mobility is the simplest case which can be solved by the registration process. The SIP location server will keep the mobile terminal's location up-to-date, so that incoming calls always reach the right addresses. The mid-call mobility is more complicated than the pre-call mobility. A simple registration process is not enough to update the current ongoing session because the packets from the remote participant are still sent to the old contact address. The mobile terminal has to notify the remote participant of the new contact address by sending a new INVITE request.

There are two basic requirements for service mobility: maintaining the adequate QoS (Quality of Service) for the current session and ensuring that users have access to all of their subscribed services regardless of the attached networks. SIP provides a combination of registration and AAA (Authentication, Authorization and Accounting) functions in order to support the service mobility.

2.6 RTP and RTCP

So far, the basics of SIP have been introduced in this chapter. SIP based real-time media streams, however, are usually carried in RTP (Real-time Transport Protocol) [23] packets. Knowledge of RTP is also important to understand SIP-based real-time communication.

RTP defines a standard packets format for multimedia transport on top of UDP (User Datagram Protocol) [9]. UDP is a stateless protocol with no error correction mechanism. Low delay latency lacks guarantee of packet loss. The UDP packets could be discarded in cases of network failure or long delay. A low rate of UDP packet loss may not affect the communication. RTP does not define a standard port for communication. Therefore, the transport port is negotiated by the SDP for each multimedia session[1]. The RTP data packets are sequenced by a 16-bit sequence number filed that is defined in the RTP packet header. The receiver may use the sequence number to detect packets loss or out-of-sequence packets. The 32-bit time-stamp field from the RTP packet header indicates the packet creation time. The value of the time-stamp filed can be used to determine the RTP packets transmission delay and jitter level. The RTP packet header also defines fields to identify source and participants for a multimedia session. The synchronization source (SSRC) identifier indicates the source where the RTP packets are generated. Participants of the multimedia session are identified by the contributing source (CSRC) identifiers. The number of contributing sources is defined in the 4-bit CSRC count (CC) field, which means the contributing sources are up to 15.

RTP and RTCP (Real-time Transport Control Protocol) [23] are commonly used together. The default transport port for RTCP can be derived algorithmically from the next higher odd port based on the RTP transport port.[2] The RTCP packets are generated periodically for adding additional system-level functionality to its related RTP stream. The RTCP is an integrated media synchronization function by exchanging RTCP messages to synchronize the system time clock for the multimedia session. The RTCP participation reports indicate the status of the participants. The participation details such as name, email and so on of each participant can also be exchanged by the RTCP messages. RTCP QoS (Quality of service) reports periodical provide statistics of packet loss, level of jitter and transmission delay, but they do not help to correct the transmission errors. Neither RTP nor RTCP provides a mechanism to ensure the delivery time, which means the QoS (Quality of service) must be controlled by another mechanism.

---

[1] The media descriptions ("m=") in SDP defines the media type, transport port, transport protocol and media format description.

[2] The RTCP transport port can be specified in SDP with a separate attribute which is "a=rtcp".

2.7 Summary

In this chapter, an overview of SIP was given, and SIP based solutions for mobility services were also discussed. This primary SIP knowledge is helpful for going deeper into this thesis work. The next chapter gives an introduction to the technical background from an implementation perspective.

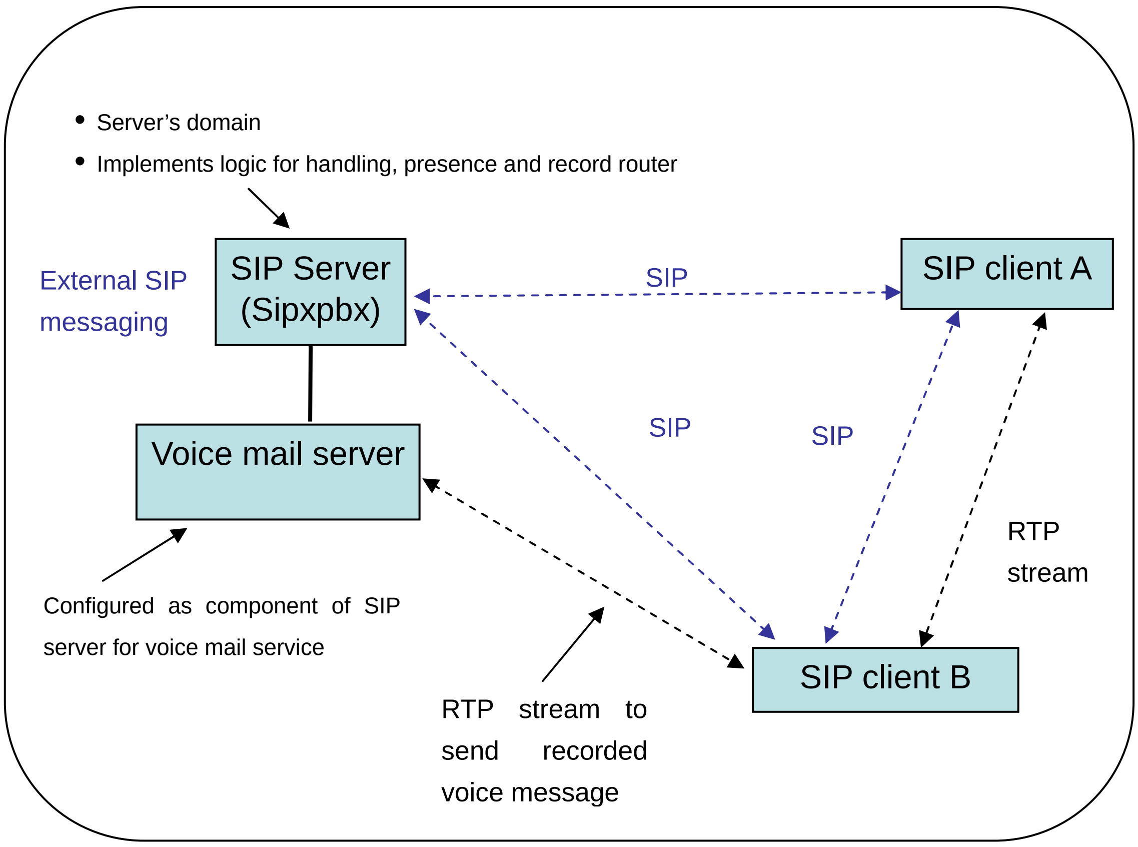
3. Technical background

This chapter introduces the technology that is used in the practical part of this thesis work. The first section presents an overview of the experimental SIP network. The following section continues with an introduction to the SIP server of the experimental SIP network. The third section describes the SIP client, and the final section introduces the Java Media Framework.

3.1 Overview of experimental SIP network

To build the experimental SIP network discussed in this thesis, both a SIP server and two SIP clients are needed. An Open Source SIP server is used for testing purposes, only requiring minor configuration work to be functional. The SIP server provides registration and record routing services. Additionally, a voice mail server is also connected to the SIP server for storing the voice mail from the SIP clients.

There are two SIP clients used in testing which are described subsequently. Both SIP clients have to register on the SIP server before they can call each other. In the use cases of this document, one SIP client acts as the caller and another one acts as the callee. Figure 3-1 illustrates the logical components of this experimental SIP network.



• Server's domain
• Implements logic for handling, presence and record router
External SIP messaging
SIP Server (Sipxpbx)
SIP client A
SIP
Voice mail server
SIP
SIP
RTP stream
Configured as component of SIP server for voice mail service
SIP client B
RTP stream to send recorded voice message

Figure 3-1 Logical components of the simulated SIP network

3.2 SIP server

The SIP server is the core of the experimental SIP network, which for the purpose of this thesis has not been modified. SipX [21] is chosen as the SIP server, and it can be replaced by any other SIP servers which implement RFC 3261 [2].

The SIP server runs on a Linux machine. The domain name server is optional and may be needed for those DNS names that can be used in SIP messages, so it is highly recommended. In the following text, we give a short description of the operating system, the SIP server application and the domain name server application.

- **Operating system:** In the experimental SIP network, Fedora core 5 Linux distribution packages with kernel 2.6 are installed as the operating system for the SIP server.

- **SIP server:** SipX [21] is one of the Open Source VoIP projects with high scalability to deploy on a Linux machine. The sipX solution provides most functions of SIP PBX (Private Branch eXchange). In this thesis work, it is running as a registrar and record router for signaling routing purpose. In addition, a voice mail server is also integrated into the SIP server.

- **Domain name server:** Although SIP accepts an IP address as a SIP address, the domain name server is still recommended to be installed for running the experimental SIP network. With the domain name server's support, the SIP clients can register with the SIP server using normal phone numbers[1], and later the registered phone numbers are used to call each other. BIND (Berkeley Internet Name Domain) [22] is used in this thesis work.

3.3 SIP communicator

The SIP communicator is a Java based SIP client supporting both voice and video calls. The reason for using the SIP communicator in this thesis work is due to less operating system dependence. Java is a programming language originally developed by Sun Microsystems. The Java code can be compiled to byte code for the virtual machine, and interpreted by the virtual machine at running time. This way, Java code can be written once and run on the most modern computers. The SIP communicator is built on top of JMF (Java Multimedia Framework). The architecture of JMF is introduced in the next section.

[1] If the phone number is 1234, then the SIP URI is 1234@netlab.hut.fi.

The disconnection tolerance functions are built on the original SIP communicator, which uses RTP streaming based detection and recovery mechanisms during a call session. The statistics of RTP packets are used for checking network disconnection. In addition, the RTCP reports are also used as references to prove the network status. The extended functions for the disconnection tolerance mechanism are described in the fourth chapter, where the details of the detection module are explained. The code architecture of the SIP communicator is given in the fifth chapter.

The SIP communicator does not recognize all SIP messages. Only those SIP messages described in the second chapter are accepted by the SIP communicator. If the SIP communicator receives an unrecognized SIP message, it will throw an “unknown message” exception. In some special cases, some SIP response messages from the SIP server are not supported by the SIP communicator. These exceptions however will not affect the testing and validation.

3.4 JMF

JMF is a java based multimedia framework which provides functions of multimedia data capturing, encoding/decoding, file system operations and network transport. This section gives an introduction to JMF.

JMF brings a lot of abstract components which need to be understood to implement the SIP communicator. This introduction to JMF starts with a short description of JMF components. Figure 3-2 demonstrates the mechanism of sending media data under the JMF architecture. Each component of JMF is introduced in the following.

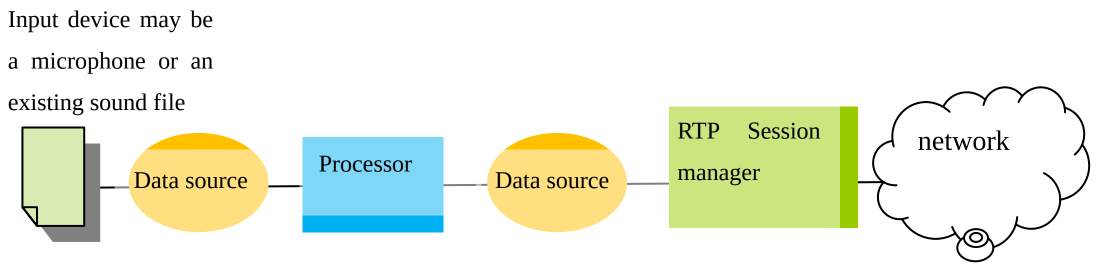


Figure 3-2 JMF components for sending data

A data source is an abstract component which is referenced as a media protocol handler. There are various sub-classes of the data source component which are concrete implementations of different protocols. When a data source is assigned to specified media content, JMF will choose a correct implementation class to handle it. Usually, the data source component can be used as the media data source for another JMF component which is transparent to high level data models regardless of the real implementation.

In JMF, the media controller is called the processor. The processor is also an abstract module which defines the way of processing and controlling time based multimedia data. The processor performs three major tasks: demultiplexing, data transcoding and multiplexing. A multimedia stream can be combined with several individual video/audio tracks. The demultiplexing task separates the multimedia stream into different tracks. Each track is a predefined media format and can be processed individually. The data transcoding task does the job of converting a media format to another media format. The individual multimedia tracks can be combined to a mixed multimedia stream by the multiplexing task.

The data sink is an output component for the multimedia data. If the data source is considered as the reader of the multimedia data, then the data sink is the writer of the multimedia data. The data sink component writes multimedia content to a specified media location.

JMF contains a ready-made RTP engine which builds RTP sessions transparently to the SIP communicator. The manager component of the RTP engine is called the RTP manager. The RTP manager takes care of the RTP sessions for the multimedia data transport, and also collects the global statistics of RTP/RTCP packets by default. The SIP communicator uses the RTP manager to create, maintain and shutdown the multimedia stream. The network detection and recovery mechanisms are built on top of the RTP/RTCP reports. The mechanism of network detection is described in the next chapter.

3.5 Summary

This chapter provided the technical background overview of this thesis work. The next chapter continues to discuss the system architecture of the disruption tolerance mechanism.

4. System architecture

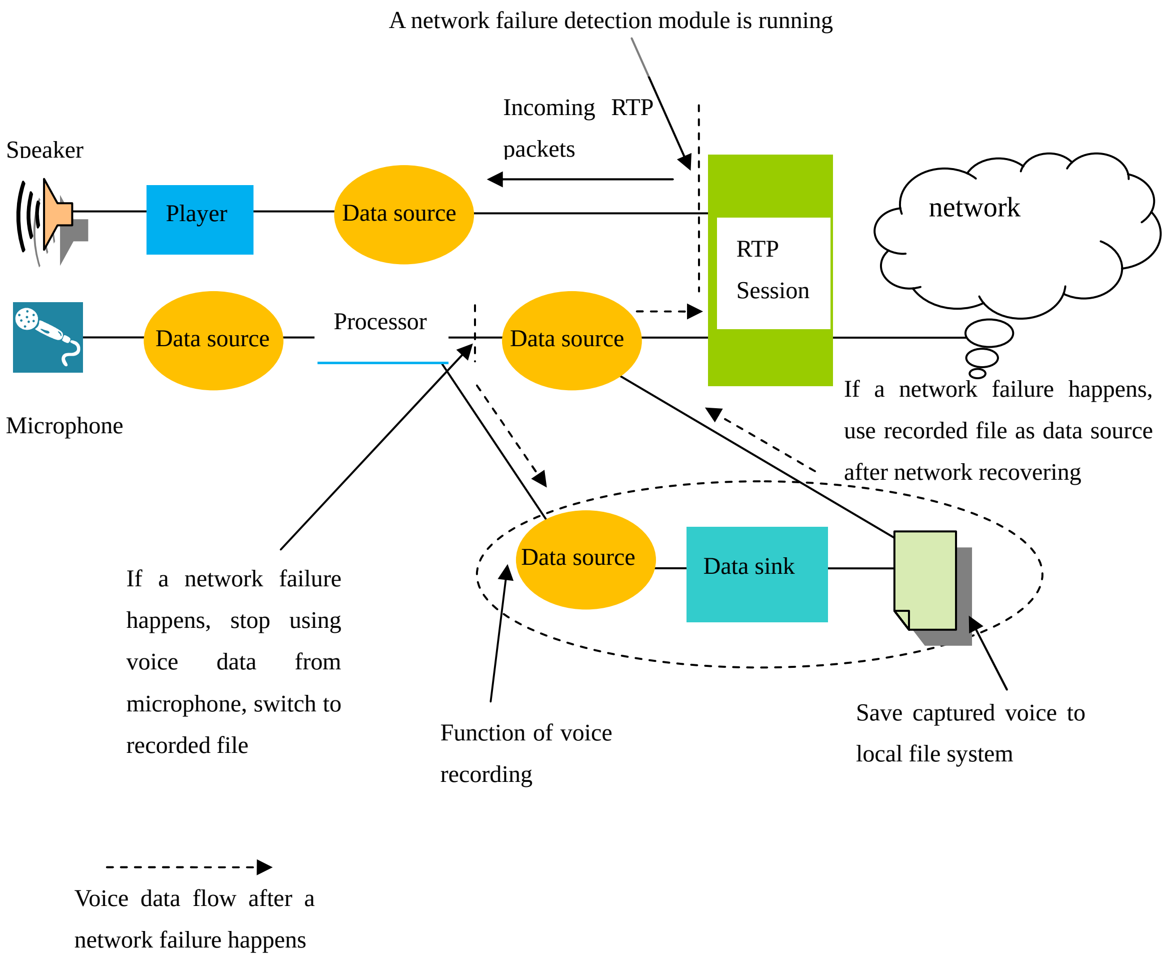
The basic technology for this thesis is discussed in the previous chapter. This chapter introduces the design architecture for the disruption tolerance mechanism, begins with a brief description of the enhanced SIP communicator, and afterwards offers an explanation of the enhancement modules. The use cases of disruption tolerance aspects are presented by the SIP message sequence charts.

4.1 Overview of the enhanced SIP communicator

This section gives an overview of the enhanced functions for the SIP communicator. The basic idea of the disruption tolerance mechanism is explained below.

The SIP communicator uses the network failure detection module to check the network condition. The network failure detection module contains a time based thread which is running at the beginning of the RTP session. The thread calculates periodically the statistics of RTP/RTCP packets for the RTP session. The detection module uses the statistic to detect network disconnection. If the network disconnection is detected, the SIP communicator switches temporarily to handle network failure mode. The users may continue to talk, and the voice is saved in the local file system. If the network is recovered after a short time period (less than 15 seconds), the SIP communicator recovers the previous broken call and replays the saved voice to the remote participant. The users may notice the obvious voice delay, but the delay time could be reduced by silence suppression. In case of an unpredictable disconnection time, the ongoing call could be either expired automatically by the session timeout or hung up by the users. Since the ongoing call is broken for a medium time period (15-180 seconds), the SIP communicator will try to rebuild the unfinished call automatically and continue the previous conversation with the saved voice. The users may continue to talk until they hang up the call. Finally, if the network connection cannot be recovered within a reasonable period, the recorded voice is sent to the voice mail server. Later, the users can fetch the voice message from the voice mail server.

In order to replay the unheard voice from the previous broken call, the SIP communicator uses different data sources for the out-going multimedia stream. Figure 4-1 shows the voice data flow of the SIP communicator.



A network failure detection module is running
Incoming RTP packets
Speaker
Player
Data source
network
RTP Session
Data source
Processor
Data source
Microphone
If a network failure happens, use recorded file as data source after network recovering
If a network failure happens, stop using voice data from microphone, switch to recorded file
Data source
Data sink
Function of voice recording
Save captured voice to local file system
Voice data flow after a network failure happens

Figure 4-1 Voice data flow

In the initialization phase, the SIP communicator detects voice capture devices by using JMF functions. The SIP communicator chooses the microphone as the default voice data source. The abstract data source concept was explained earlier in the previous chapter. Here, any encoded multimedia data can be considered as data sources, and the data sources can be mapped to different destinations. The SIP communicator uses both the voice capture device and the file data source for the out-going multimedia stream. When a call is started, the voice data from the microphone is forked to two destinations. One data stream is sent to the remote participant through the network, and another one is simultaneously saved in the local file system. The default folder for the recorded files is in the root folder of the file system. If network failure happens, the network detection component fires an event to notify the SIP communicator of the network disconnection. The event triggers the switching of the data source from the voice capture device to the file data source. When the call is recovered, the file data source is used for the out-going multimedia stream. The details of the modules are introduced in the following section.

4.2 Module description

This section concentrates on the enhancement modules for the SIP communicator. Two modules are implemented in this thesis in order to build the disruption tolerance functions. The sound record module provides the voice recording function for the entire call session. The network failure detection module provides the mechanism for detecting network disconnection.

4.2.1 Sound record module

During a call, the voice data is saved in cycle files. The reason for using different files to save voice data is for thread safety. One file cannot be written and read at the same time. When JMF is writing data into a file, this file is locked and cannot be read correctly by another thread. The default number of cycle files is three. The quota for each file is limited to 180 seconds of recording time in order to prevent the file from becoming too large. If the recording file reaches the time limit, the old recording content is discarded silently unless the data is usable for recovering the previous broken call. When a new call is created, the voice data from the microphone is written into the first file until network disconnection is detected. When the network failure occurs, the sound record module stops writing the first file, and releases the file to prepare for replaying of the voice right after the network connection is recovered. The task of recording the voice is switched to the second file. After network connectivity is recovered, the sound record module uses the first file as the data source for replaying the recorded voice. It also stops writing data to the second file. The second file is released and prepared for the reading task, which is called for at the end of the first file. The writing file task is switched to the third file. When the first file reaches the end of the file, the second file is ready to become the data source for replaying voice, and the sound record module stops writing data to the third file, and then switches back to overwrite to the first file. In this way, three different files are cycled for writing and reading till the end of the call. Figure 4-2 depicts the switching phases between cycle files.

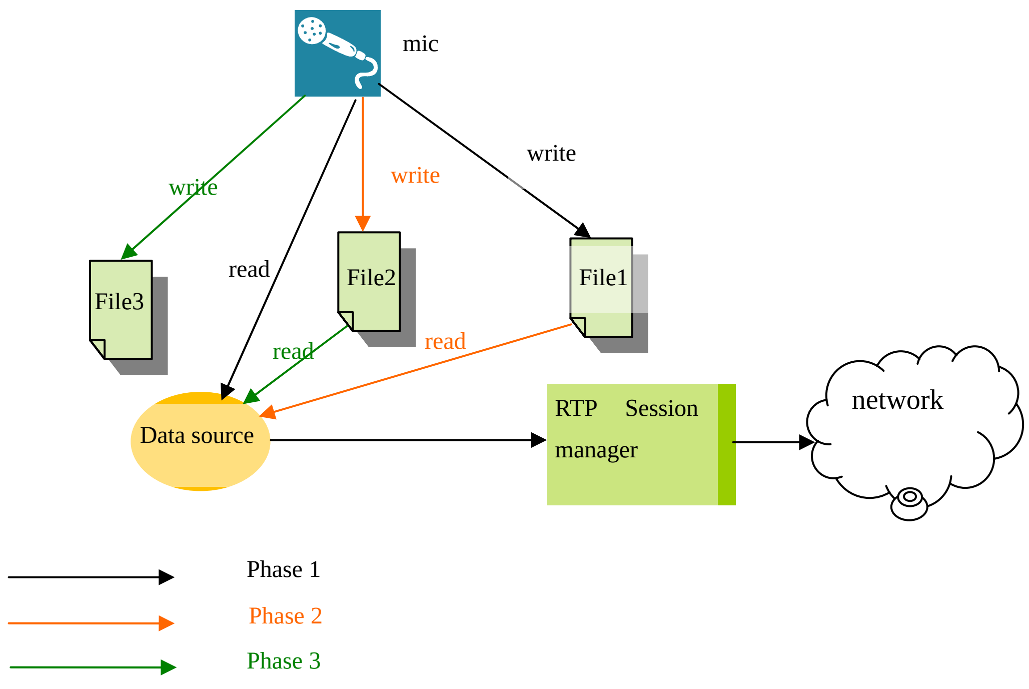


Figure 4-2 Sound record module

4.2.2 Network failure detection module

Unlike TCP which offers a reliable connection, RTP packets can be discarded without any notifications to the sender. If any call participant loses connection, it is difficult for the SIP communicator to detect the network condition of the remote participant. Also, even if the SIP communicator knows the network connection problem itself, there is no way to notify the remote SIP server or SIP client. The network failure detection module is implemented to monitor the network condition of the remote participant. The mechanism is described in the following text.

The network failure detection module contains a time based thread which is called the timeout checker. The timeout checker starts running in the background as soon as the first RTP packet is received by the RTP receiver. It wakes up every 3 seconds and checks the current network status. The timeout checker collects the global reception statistics of the RTP stream and counts the number of received RTP/RTCP packets since the last check point. If no packets were received in this period, the timeout checker fires a timeout event to inform the SIP communicator and sets the network status to failure. If the network status was already in failure, the total disconnection time is accumulated. If the network disconnection time exceeds the session expiration time, the timeout checker fires a local hang up event to notify the SIP communicator to hang up automatically. Otherwise, the statistical report shows which RTP packets were received since the last check point. The timeout checker examines the network status. If the network status was set to failure, the timeout checker fires a network recovery event to inform the SIP communicator. The timeout checker keeps on running for the whole RTP session, only stopping when the user hangs up the call or a local hang up event is fired. Figure 4-3 presents the process of detecting the network failure.

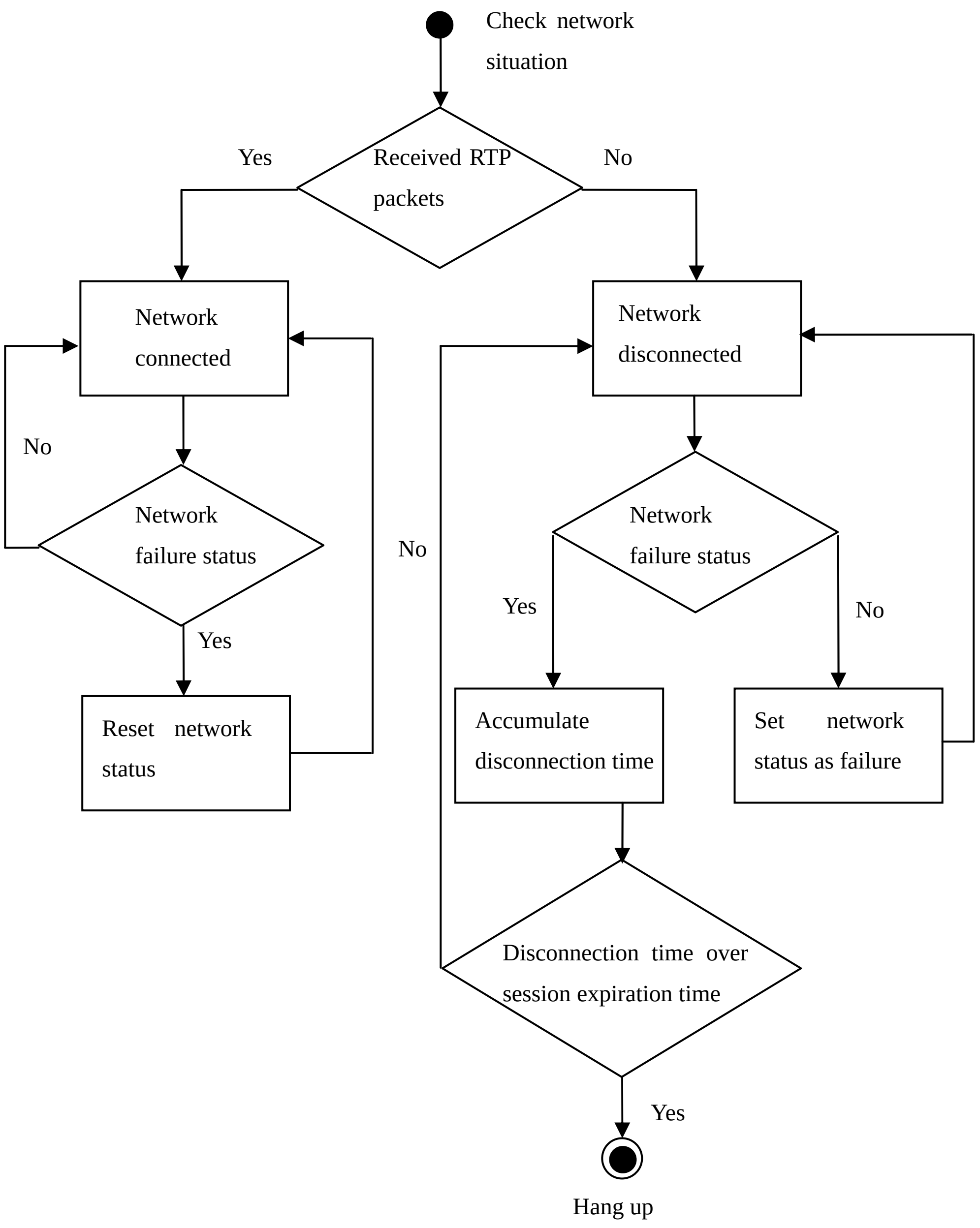


Figure 4-3 Network failure detection model

Silence suppression is the term used to describe the process of not transmitting information over the network when one of the calling participants is not speaking. The function of silence suppression has not yet been implemented in this thesis work. If silence suppression is to be used in future work, the RTP packets are only generated while the user is speaking. Depending on the implementation of silence suppression[1], the network failure detection module may not work correctly if the users have been silent for too long. However, RTCP packets are sent periodically regardless of the RTP packets. Therefore, the consecutive RTCP packets could be used as a reference for checking the network condition. If there are no RTP packets received since the last checking interval, the lost RTCP packets indicate when network failure occurs.

The RTP/RTCP packets statistics based network failure detection mechanism cannot detect the network failure time very precisely. As mentioned above, the check period is 3 seconds, so the SIP communicator would only know the network was disconnected in the last 3 seconds time period. Of course, there are more precise ways to check the exact disconnection time, for example, by monitoring network interfaces directly. The benefit of RTP/RTCP packets statistics based detection mechanism is to eliminate the complexity from lower layer network interfaces. If the mobile device has more than one network interface activated at the same time, monitoring the network traffic of each interface is dependent on the local policy of the mobile device.[2] Therefore, the soft handover between lower layer network connections will not affect the detection mechanism.[3]

The SIP communicator can only detect the disconnection time approximately. As described above, the timeout checker periodically counts the received RTP packets. The network failure detection module uses the number of RTP packets received since the last check point to detect the network disconnection. Therefore, the detection module would not know the exact disconnection time, but only the network lost connection since the last check point. The disconnection time is in the range of 0 to 3 seconds. In the replaying mode, the SIP communicator replays the recorded voice 3

[1] If background noise is totally filtered in silence suppression codec, there is no packet sent while the user remains silent.

[2] The local policy of choosing network interface can be signaling strength, network bandwidth, user specified and so on.

3 The precise monitoring solution is more devices specific.

seconds backwards from the time the network failure was detected. Figure 4-4 illustrates the voice replay time line.

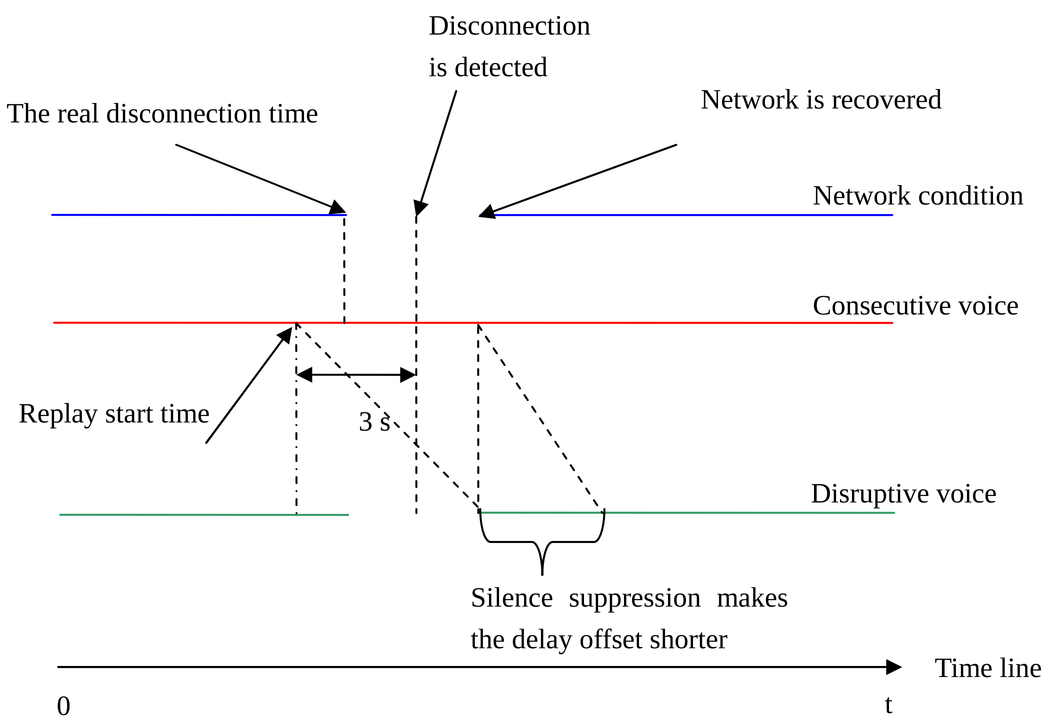


Figure 4-4 Voice replay time line

After the media path is recovered, an uncompensated disconnection interval makes a delay offset. The delayed voice creates a bad experience for the end user in the continuation of the disrupted conversation. Silence suppression could be a good candidate for reducing the delay offset.

4.3 SIP Message for Different Use Cases

Chapter two introduced the basics of SIP. This section describes the SIP signaling of different scenarios in which network disconnection occurs. The SIP messages are represented in message sequence charts. Three use cases are described in the following section.

To simplify the use cases, we assume all use cases are between two SIP clients. The SIP clients are named as SIP client A and SIP client B. The signaling between client A and client B is sorted in an order based on a vertical time line.

4.3.1 Short Duration of Network Failure

This section describes the scenario of a short duration network failure after the media stream is established. The second chapter introduced the sequence of building a successful registration. Both user A and user B are registered to the SIP server. User A calls user B. After a handshake between SIP client A and SIP client B, a media stream is created between those two clients. There is no SIP signaling exchange between the SIP clients during the media streaming phase, so that the SIP server will not notice if any SIP clients are losing connection until the registration time is expired. If there is a short time network connection failure, the SIP server has no way of detecting the failure. The SIP client also has the same limitation for detecting if the remote client is losing the connection. Therefore, any SIP based detection mechanism is not a good candidate for efficient disconnection detection.[1]

The mechanism of RTP/RTCP packets statistics based disconnection detection is explained in section 4.2.2. If a disconnection notification event is fired by the detection mechanism, the SIP communicator switches to network failure mode. The SIP communicator is not concerned about the real location of the network failure at either local or remote parts. Any type of network failure will break the communication and neither SIP clients can receive the remote media stream anymore. In network failure mode, the SIP communicator is still sending RTP packets to the remote SIP client, but

[1] If the SIP client uses session timers, fine granularity scalability is possible. But if the interval is too big, this is not suitable for short time disconnection detection. SIP session timer [10] can generate SIP messages in regular intervals, but the minutes based timeout value is not efficient for detecting the network condition.

it is also saving media data to the local file system. The last unheard media data of the users' conversation is kept in the local file system. If the network failure is recovered in a short time, both call parts will receive RTP packets from the remote SIP client. The network detector notifies the SIP communicator to switch to replay audio mode. The SIP communicator fetches the recorded media data and replays it to the remote participant. Both users will hear the saved voice data for the duration of the network failure from the remote call part. If the users keep on talking to each other, the conversation is delayed based upon the previous network failure time. Figure 4-5 depicts the message sequence. In this example, user B hangs up the call at the end.

As we mentioned in section 4.2.2, the delayed voice can be reduced by using silence suppression. The basic idea of silence suppression is to catch up with the silence intervals in the users' conversation, and then remove the silence intervals to make the conversation shorter. The function of silence suppression is not implemented in this thesis. The silence suppression function can be implemented as following: A filter performs pre-processing of the media data which is going to be saved to the local file system. The filter checks the media data and removes the silence intervals from the media data. This way, no redundant silence voice is saved, and the unheard voice becomes shorter. Further implementation of the silence suppression function is needed to enhance the user experience of the SIP communicator.

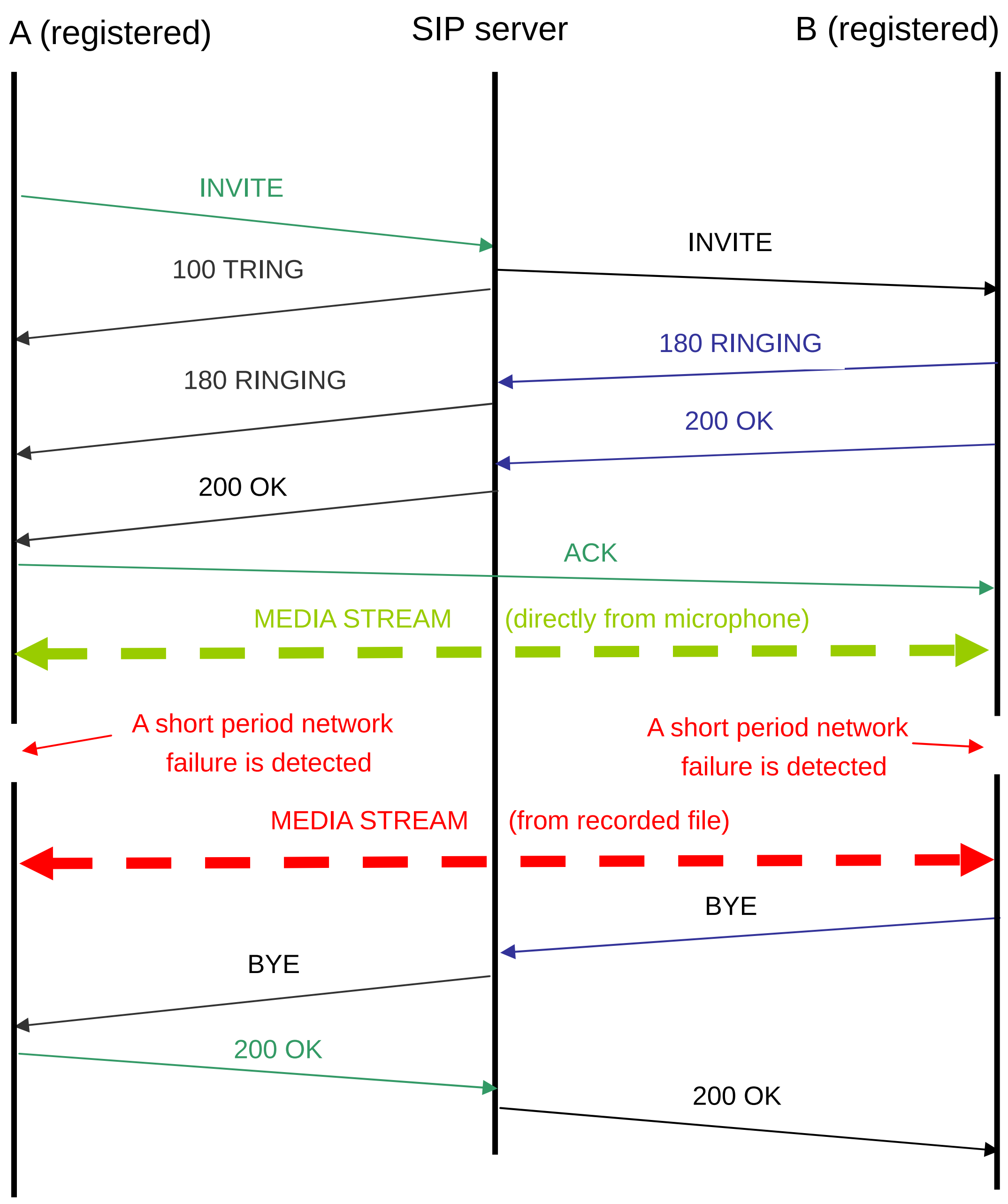


Figure 4-5 A short duration of network failure during a call

4.3.2 Rebuilding a call after a long duration of network failure

In mobile communication, unpredictable network failure may occur at anytime. Long failure duration causes media streams to time out, and users may lose patience during a long waiting time without hearing any response from each other when there is a sudden break in the call. The scenario of a short time network failure was described above in section 4.3.1. This section describes the solution for rebuilding a call after a long duration of network failure. The steps for creating a new call are the same as in previous sections. Network failure happens after the media stream is created. As section 4.2.2 described, the network failure detection model cannot tell the location of the network problem. Both participants in the call affected by network failure will observe the same result due to the network failure events. Although each SIP client knows the network condition itself, it cannot exchange information with another remote SIP client. After the network failure has happened, both SIP clients can detect the transmission problem. If the network disconnection time is long enough that may due to time out the active RTP session. After a predefined expiration time, the SIP communicator shuts down the media stream automatically. In this case, the user may continue to talk, so the unheard voice is also recorded to the local file system. After the media stream is closed, the RTP/RTCP packets based recovery mechanism will no longer work. SIP signaling is used here to rebuild the unfinished call. Both SIP clients periodically send new INVITE requests through the SIP server to the remote call participant. There are two cases explained below, the first case is when the remote SIP client encounters a network failure, and the second one is when the local SIP client encounters a network failure. If the network failure happens to both remote and local SIP clients, it counts as the second case. The assumption is based on the condition that the SIP server is always stable, which can be assumed in a well-functioning machine. Handling the case of an unstable SIP server is beyond the scope of this thesis, and it is assumed that both SIP clients can only handle a single call at a time.

- If the remote SIP client is not reachable, the INVITE request will reach the SIP server but it will not reach the remote SIP client. Later the SIP server will reply with a 408 REQUEST TIMEOUT[1]. After the local SIP client receives the 408 REQUEST TIMEOUT responses, it sends a new INVITE request to the remote

[1] Or a 404 NOT FOUND if the remote part's registration has expired in the meantime.

SIP client. The INVITE request will be sent periodically until a response or an INVITE request from the remote SIP client is received.

- The SIP client creates a local transaction for the INVITE request. If the local SIP client does not receive any responses or requests from the remote SIP server or the remote SIP client during this transaction, it indicates that the network failure has not been recovered. The SIP client sends a new INVITE request after the previous INVITE transaction has expired. New INVITE requests will be sent periodically until a response or an INVITE request from the remote SIP client is received.

The period between the sequenced INVITE requests should be randomly distributed around a specified time to avoid the INVITE requests being sent from each call participant at the same time. The SIP client must not send a new INVITE request before either 408 REQUEST TIMEOUT or 404 NOT FOUND responses have been received, or the local transaction has expired. When the SIP client receives an INVITE request from the remote SIP client, it firstly compares the received SIP message header fields to the previously broken call dialog with the following steps:

- The INVITE request matches the previous broken call: if a new INVITE request has not yet been sent, the SIP client must stop sending a new INVITE request and reply to the INVITE request as soon as possible. Otherwise the SIP client sends a 486 BUSY HERE response then generates a new INVITE request that is randomly distributed for a specified duration. If the SIP client receives a 486 BUSY HERE response for the previously INVITE request, it should send a new INVITE request immediately.

- The INVITE request is not from the previously broken call but from a third party: the SIP client handles the INVITE request as in normal case and sends a CANCEL request to the remote SIP client. Later, all of the INVITE requests from any remote SIP clients will be replied to with a 486 BUSY HERE response. If the SIP client receives a CANCEL request from the previous broken call part, it should no longer send a new INVITE request. The recorded voice is going to be sent to the voice mail server after the current call is hung up.

If neither SIP clients receive any new INVITE requests or responses from each other during a predefined period, they should stop attempting to rebuild the broken call and try another way to exchange information. A voice mail server is a permanent place to keep the recorded voice data, and later the SIP client can fetch the lost voice call any time. The way of connecting to the voice mail server is discussed in the next section.

Figure 4-6 shows a successful re-invitation after a network failure. In this example, SIP client B discarded the first two INVITE requests because of a network failure. SIP client A receives a 408 REQUEST TIMEOUT then generates a new INVITE request. This INVITE request is delivered to SIP client B and the broken call is rebuilt. Both SIP clients reply to the recorded voice, and later SIP client A hangs up the call. In the prototype implementation, there is no control button on the SIP communicator to turn off the automatic rebuild call function and the SIP client will always try to reconnect to the remote call part after a call is broken. This enhancement should be added in the future.

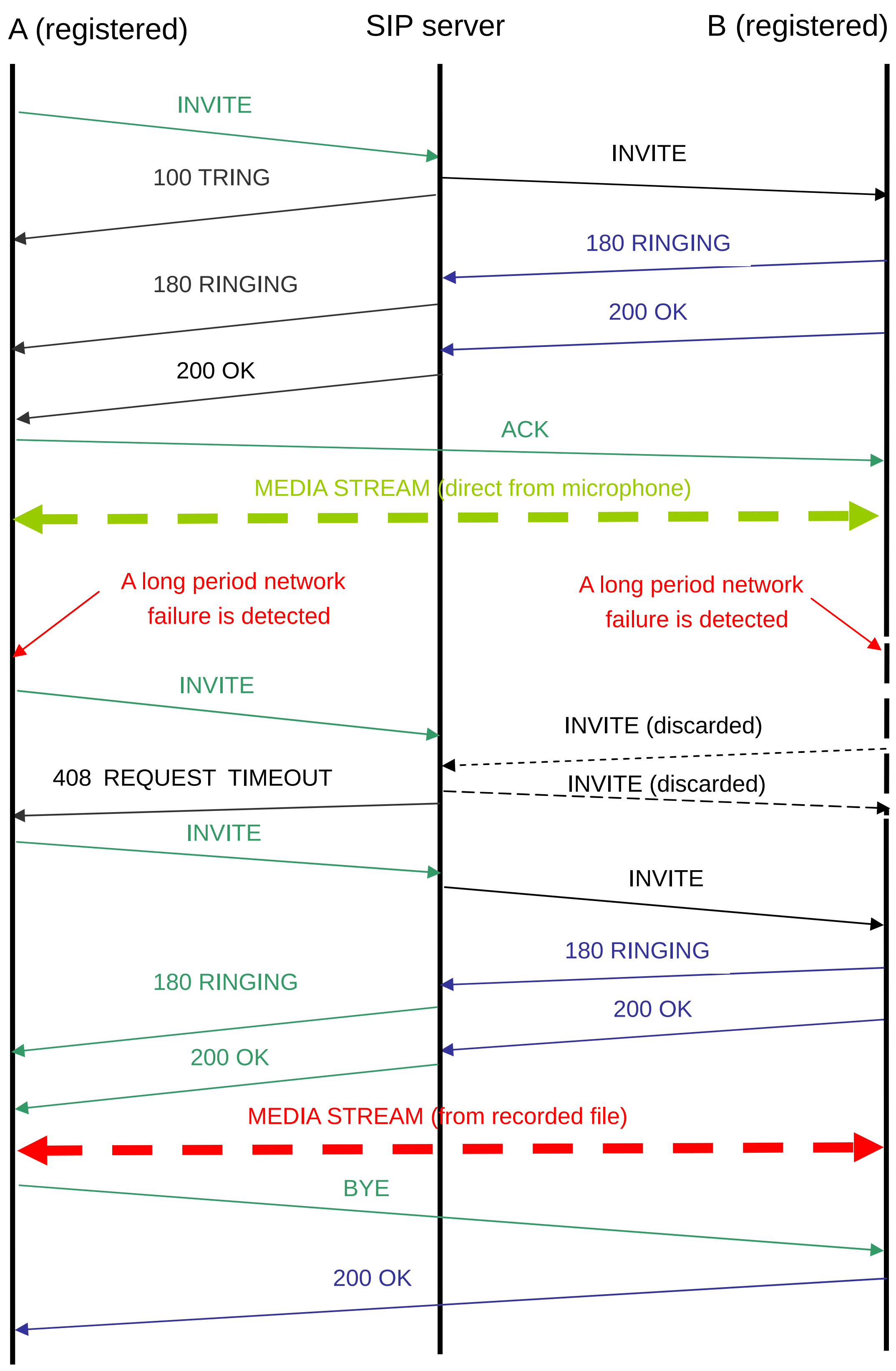


Figure 4-6 Rebuild a call after a long duration of network failure

4.3.3 Connect to voice mail server after a long period of network failure

This section presents and discusses the process of connecting to the voice mail server. If neither SIP clients reach each other actively, an alternative solution is to use a voice mail server as a permanent repository to save the recorded voice data. As in the previous sections, user A starts a new call with user B. There is a network failure which occurs during the call. Here, we assume that user B has lost network connection for a long time. SIP client A tries several times to invite SIP client B, but the broken call cannot be automatically rebuilt in the way described in the previous section. In this thesis implementation, there is a constant ‘N’ which defines the number of total invitation times. If the call cannot be rebuilt after ‘N’ time’s invitation, both SIP clients should not send any new INVITE requests. Instead, the SIP client sends the recorded voice to the voice mail server. Later, each SIP client will fetch the voice mail from the voice mail server independently. Figure 4-7 depicts the SIP message sequence of connecting to the voice mail server. The function of connecting to the voice mail server has not been implemented in this thesis. There are several missing functions from the SIP communicator in this case, for example, the DTMF (Dual-tone multi-frequency) function is not supported. When the SIP communicator connects to the voice mail server, users cannot interact with the voice mail server by using the SIP communicator. It is necessary to implement the DMTF function in the future for the SIP communicator to be able to interact with the voice mail server. The SIP communicator does not support persist functions to remember the call status permanently, which means that the call status is only kept in memory. If the user shuts down the SIP communicator before the network failure is recovered, the current call status will be lost immediately. It would be better to implement the persist function for the SIP communicator to write the unfinished call information to the local file system, and this call information can be fetched by the SIP communicator later at any time. This feature will improve the usability of the SIP communicator.

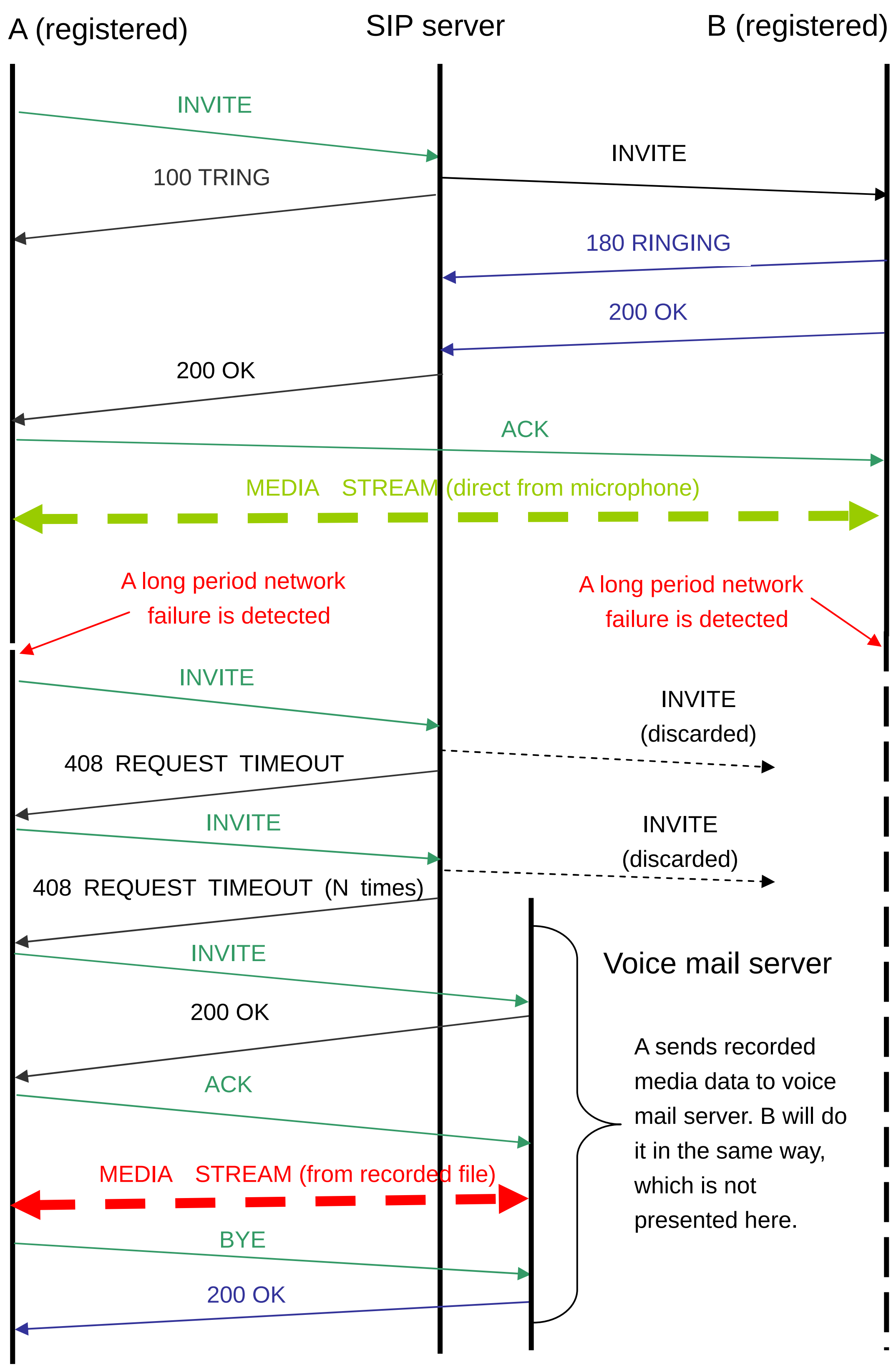

A (registered)
SIP server
B (registered)
INVITE
INVITE
100 TRING
180 RINGING
200 OK
200 OK
ACK
MEDIA STREAM (direct from microphone)
A long period network failure is detected
A long period network failure is detected
INVITE
INVITE (discarded)
408 REQUEST TIMEOUT
INVITE
INVITE (discarded)
408 REQUEST TIMEOUT (N times)
Voice mail server
INVITE
200 OK
A sends recorded media data to voice mail server. B will do it in the same way, which is not presented here.
ACK
MEDIA STREAM (from recorded file)
BYE
200 OK

Figure 4-7 Connect to voice mail server after a long period of network failure

4.4 Summary

In this chapter, the design architecture was explained in detail and the theory of implementing the disruption tolerance functions introduced. The sound record module and the network failure detection module for the disruption tolerance mechanism were described conceptually. Three use cases were assumed and the solution for each use case was recommended. For a short duration of disconnection, the SIP communicator temporarily stops the ongoing media stream and saves the users' conversation. The recorded voice is replayed automatically after the network recovery. For a longer duration of network disconnection, the SIP communicator tries to rebuild the previously broken call by sending INVITE requests. Finally, the voice mail server is a permanent repository for the recorded voice data. Each use case was explained with a SIP message sequence chart. The next chapter describes the implementation details of the SIP communicator with the enhancement modules.

5. Implementation

This chapter presents the details of the implementation. To simplify the description, UML (Unified Modeling Language) is used here to describe the interaction and relationship between those classes. This chapter does not aim to cover all of the implementation classes, but focuses on only those classes that are important for understanding the code architecture.

5.1 High level view

The SIP communicator can be subdivided into three major modules. The SIP package classes are implemented for processing SIP signaling. Classes from the media package take care of multimedia data and RTP transport. The GUI (Graphic User Interface) package contains classes which render the user interface and acting upon a user's action.

The class SipCommunicator is the main class of the SIP communicator, it is called when starting the whole program. This class reads the system configuration and initializes the program. The SipCommunicator does not know the implementation details of the user interface, SIP signaling and the network connection. All of these functions rely on underlying classes. The Class SipCommunicator contains three management classes which are called SipManager, MediaManager and GuiManager. These management classes refer to the packages which contain the classes of concrete implementation of SIP signaling, media transportation, and graphical user interfaces respectively. The relationship between these classes is presented in figure 5-1. The management classes are introduced in the following sections.

- **SipManager:** the manager class of the SIP package which is responsible for creating the SIP requests and generates the SIP responses.

- **MediaManager:** the manager class for controlling the multimedia stream and managing RTP packets at the transport layer. This class also deals with voice recording and rebuilding multimedia streams after the network disconnection is recovered.
- **GuiManager:** the manager class of the graphical user interface package. It is called by the SipCommunicator in the initialization phase to render the user interface.

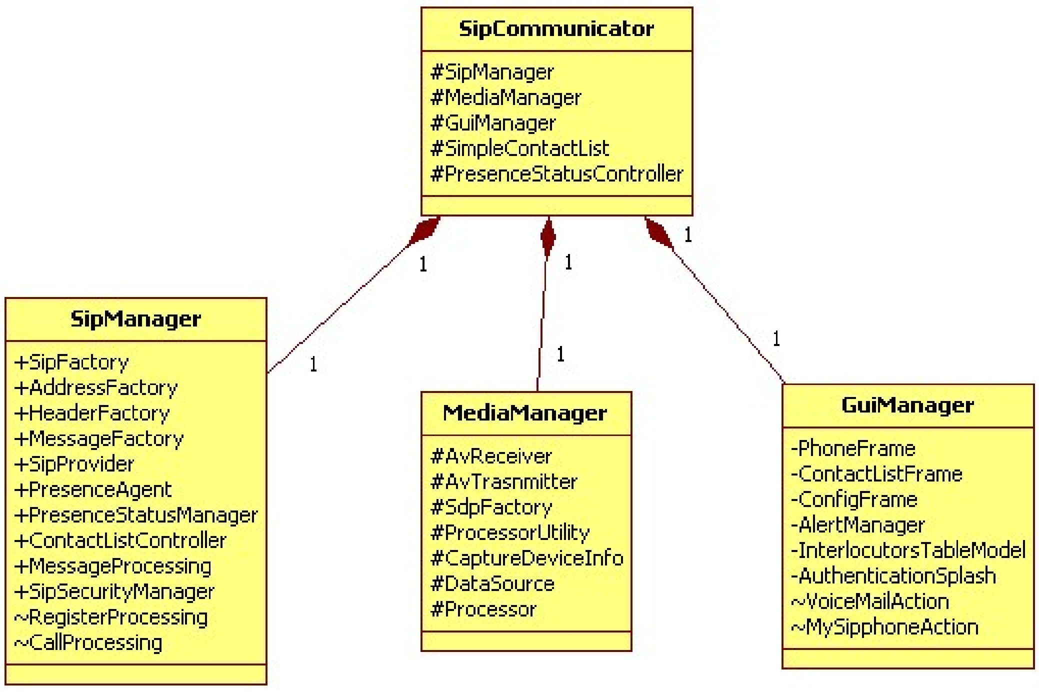


Figure 5-1 Top view of SIP communicator

5.2 SIP package

The SIP package provides all the functions of SIP signaling which are used in the SIP communicator. The SIP basics were introduced in Chapter 2, and Chapter 4 described the SIP messages for different user cases. This section gives a short introduction of the implementation of the SIP package. Figure 5-2 depicts the relationship between the classes in the SIP package. The functionality of each class is described below:

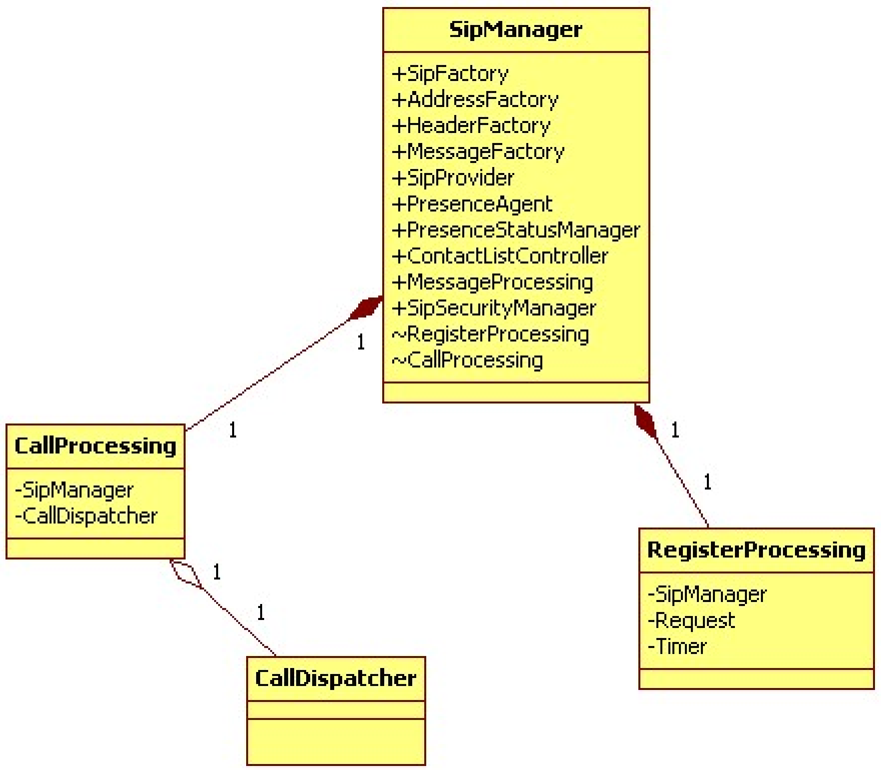


Figure 5-2 SIP package classes

The class SipManager is the factory class of SIP signaling. It manages the way of generating SIP messages for different use cases. It only knows which SIP message is going to be created, but it is not aware of the message detail. When a SIP request or response is generated by a concrete underlying class, the concrete class also takes the responsibility for maintaining the status of the respective SIP message.

As the name suggests, the class RegisterProcessing takes care of the entire register processing of the SIP communicator. The class SipManager calls the registration functions from this class to register the SIP communicator to the SIP server. When a

REGISTER request is sent, this class RegisterProcessing also takes care of the SIP stack of the REGISTER request.

The class CallProcessing is an implementation class for creating SIP requests, parsing SIP requests/responses and generating SIP responses. It provides the INVITE request messages which are used by the class SipManager for creating new calls. Also, this class parses the SIP INVITE requests which are forwarded from the class and builds the corresponding SIP responses. The class CallDispatcher is a container for existing calls. The call state information, transactions and dialogs of unfinished calls are kept in separate instances and managed by this class. The information about the calls is used to maintain the ongoing calls and generating stateful responses. This feature is very important for multi-user call sessions.

5.3 Media package

Chapter 4 has introduced the basic JMF architecture, which is the basic concept of JMF. The classes of the media package are built on top of the JMF APIs. The main task of the media package is to provide multimedia data encoding, decoding, recording and transport functions. Figure 5-3 shows the classes' relationship of the media package.

The manager class of the media package is the MediaManager, which controls the multimedia streams. The class MediaManager deals with the SDP (Session Description Protocol) messages to find the negotiated media formats, and initialize the corresponding data sources. The concrete implementation of the data transmission and reception are implemented in the classes AVTransmitter and AVReceiver.

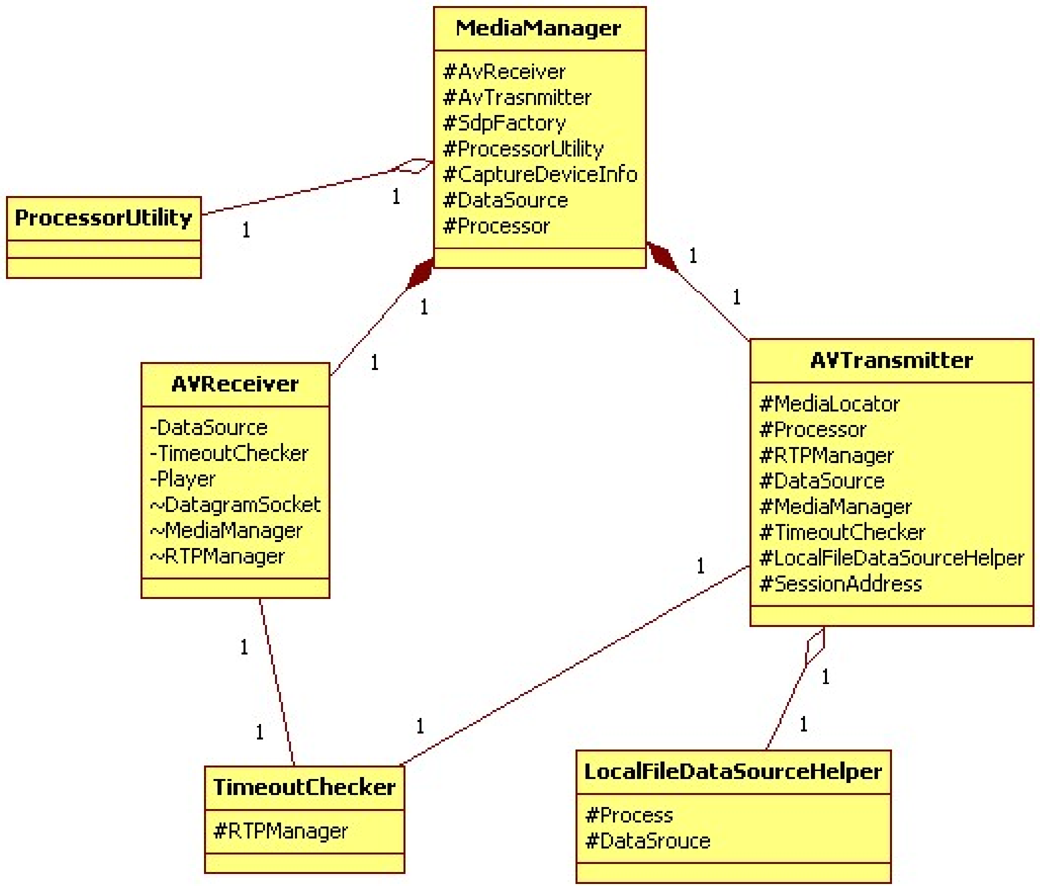


Figure 5-3 Media package classes

The class AVTransmitter is a concrete implementation class of the multimedia data transport which is based on JMF. This class builds upon RTP data streams to the remote clients and controls the media streams. Following the design idea explained in the fourth chapter, the voice data sources may vary in different use cases. The class AVTransmitter uses the microphone as the default data source when the call is started, and a copy of the voice data is saved to the local file system at the same time. Later the default data source could be replaced by the recorded voice data from the local file system after the network disconnection is recovered. The switching between the different data sources is invoked by the network status events which are generated by the class TimeoutChecker.

The class AVReceiver is a receiver of RTP data packets. This class listens to the RTP and RTCP ports and collects all incoming packets. This class uses the JMF APIs to decode the received multimedia data and replays the voice to the users.

The class TimeoutChecker is a network failure detector of the SIP communicator. This class is initialized by the class AVTransmitter after the first RTP packet from the remote participant is received. A new thread is created during the phase of class initialization. This thread counts RTP/RTCP packets at every 500ms, and then judges the network condition. If there are no RTP packets received since the last check point, a network failure event is fired. Later, the new coming RTP packets will trigger a network recovery event. The interface ConnectionTimeoutListener holds three methods which indicate three different types of network statuses: failure, recovery and remaining. The implementation classes of this interface will be notified immediately when the network condition events are fired.

The class LocalFileDataSourceHelper is a helper class for switching between the recorded files. In order to be threading safe, each file is not allowed to be written and read at the same time. This class tracks all of the recorded data files, and manages their I/O status.

5.4 GUI package

This section describes the graphic user interface implementation of the SIP communicator. The user interface components of the SIP communicator are mostly based upon the swing APIs. Swing is a lightweight GUI toolkit for Java.

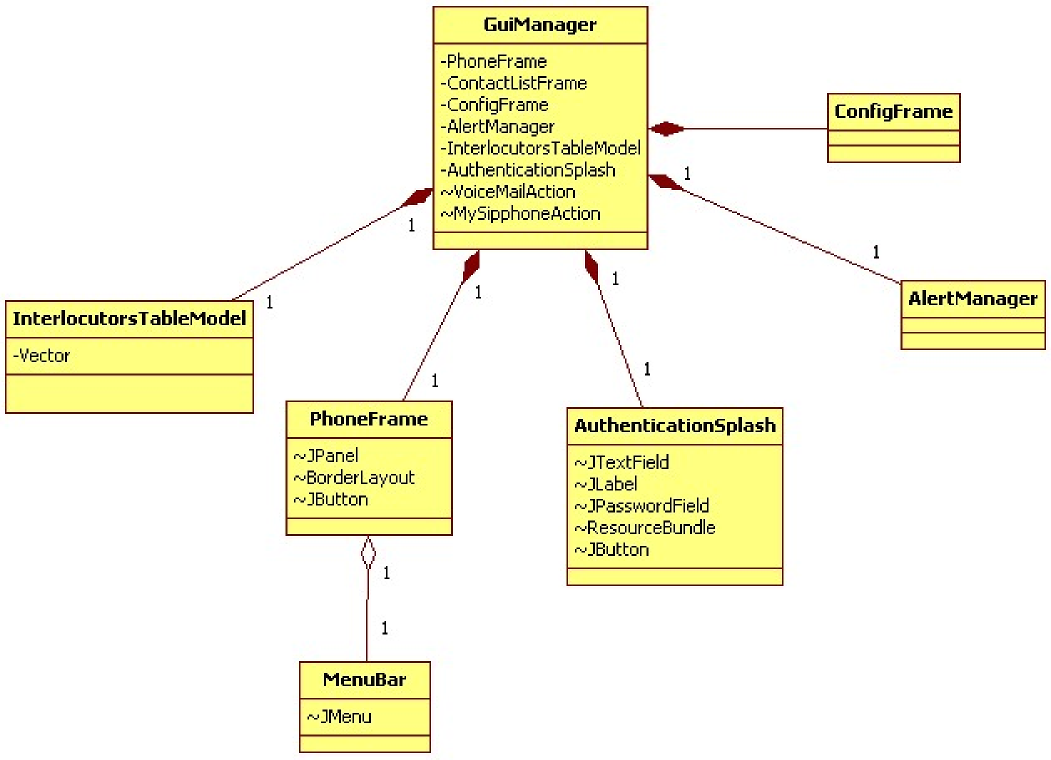


Figure 5-4 GUI package classes

Similar to the other packages, there is a manager class for this package which is named GuiManager. The manager class initializes each component of the user interface and adds them to the correct place. Also, this class listens to the events from the other modules and reacts to them.

The main function of the class InterlocutorsTableModel is to display a drop down list of the callees' information to the user. The callee's name, address and class status are shown in a table format.

The class PhoneFrame renders the main window of the SIP communicator. Laying out the frame of the SIP communicator is beyond the scope of this document, further details of rendering the layout are discussed in the Java swing technology. The class MenuBar is to create a menu of the SIP communicator, which is part of the widgets on the main window.

The class AuthenticationSplash provides a dialog window for authentication information. This class renders a popup dialog window for the user’s account name and password in the register phase.

The class AlterManager is an exception from the GUI package which does not render any graphic user interface. This class manages a set of audio clips for prompting and alerting, for example, the ring for incoming calls. Nevertheless, audio is also a way to interact with the user.

5.4 Summary

This chapter has given the implementation details of the SIP communicator. The first section gave the packages view of the SIP communicator where each package was explained in the subsequent sections. The introduction of the SIP package was given in the second section. The third section introduced the media package. The fourth section explained the GUI package for the user interfaces. The next chapter in this thesis continues by focusing on the testing and validation of the implemented prototype.

6. Testing and validation

This chapter focuses on the testing and validation of the SIP communicator. The first section describes the way of setting up the demonstration environment, while the second section presents the testing results and an analysis of these results.

6.1 Demonstration setup

The pre-condition for running a demonstration is to build an experimental SIP network. Based on the technical background described in the third chapter, we are building the experimental SIP network with one SIP server and two SIP clients. In the demonstration, one Linux machine is running the SIP server application, and two Windows machines with WLAN connections running the SIP communicator application. The windows machines are connected to the experimental SIP network by a WLAN router. The SIP server has an Ethernet connection to a normal router. The media path between the wireless router and the normal router is also an Ethernet connection. The setup is illustrated below in Figure 6-1.

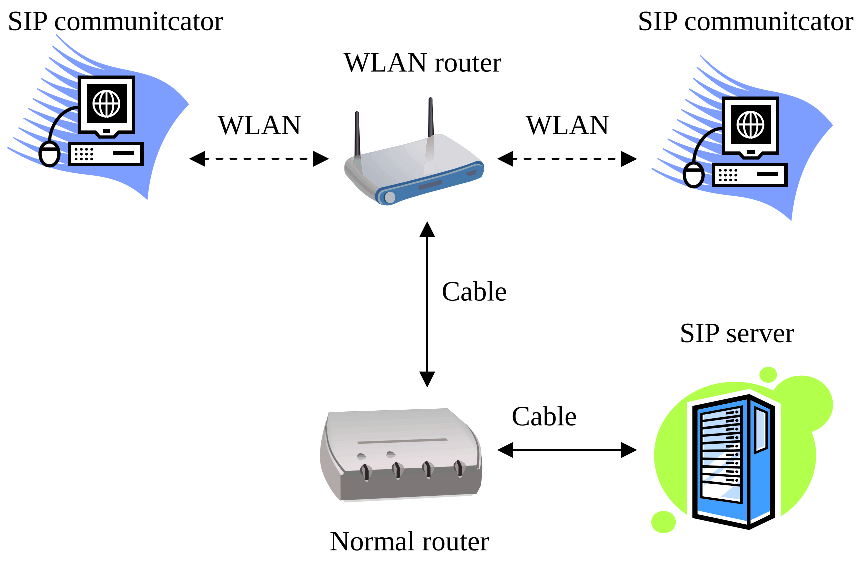
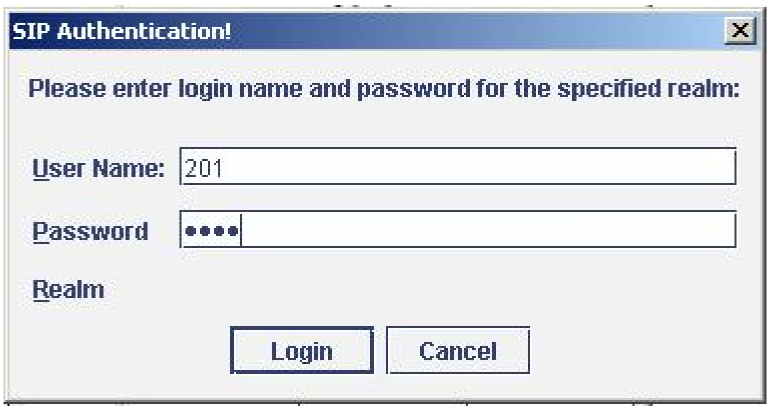


Figure 6-1 Demo setup

The SIP server is installed with all of the necessary server side applications which are described in the third chapter. The SIP server is controlled remotely by Linux commands through an SSH connection. The details of these control commands, however, are not presented in this section. Here we assume that the SIP server is booted and all of the required applications are running smoothly.

When the SIP communicator is started, a pop-up window appears on the screen and requests the authentication information. The pop-up window contains two input fields: the user name and password, as shows below in Figure 6-2.


SIP Authentication!

Please enter login name and password for the specified realm:

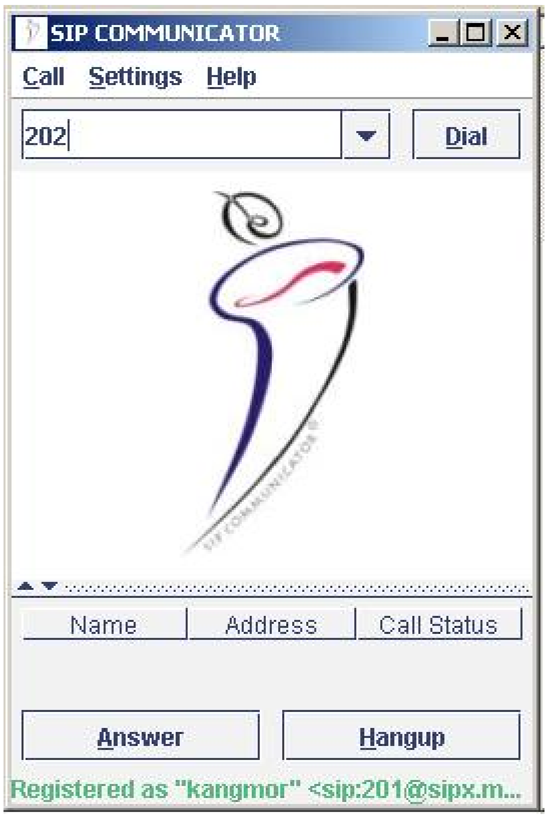
User Name: 201

Password ●●●●

Realm

Figure 6-2 Authentication window

We create two accounts in the SIP server for the purpose of testing. After the user's authentication information has been filled, the SIP communicator registers this information with the SIP server. The registered information shows on the button of the SIP communicator in a green color. After both SIP communicators are registered with the SIP server, they are ready to call each other. Figure 6-3 depicts the user interface of the SIP communicator after registration.



Registered as "kangmor" <sip:201@sipx.m...

Figure 6-3 UI of SIP communicator

6.2 Testing and validation

Integration testing is an important part of all software research and development life cycle. Presenting a full introduction of software testing is a huge topic of software engineering, and is far beyond the scope of this thesis document. The main purpose of testing in this section is to prove the disruption tolerance mechanism for the SIP communicator. In the following text, a brief introduction of the test cases is given.

6.2.1 Test cases

In integration testing, it is necessary to first validate the normal call function to make sure that both SIP communicators can call each other smoothly, then we address testing of the disruption tolerance functions. The fourth chapter gave a description of different use cases for these disruption tolerance functions. The testing verifies each use case in the same order as the design architecture. The testing of individual components is not within the scope of this thesis, that kind of testing is covered by unit testing. The test cases are presented in Table 6-4 below.

Test case	**Description**	**Result**
1. Normal call	User A calls user B without disconnection during the conversation, then user A hangs up the call.	Works correctly.
2. A short duration of network failure	User A calls user B. After the call is started, the network connection is deliberately broken for a short duration. The recorded conversation is replayed automatically after the network is recovered.	Works as assumed, the recorded voice comes with a large delay after the call is recovered.
3. Rebuild a call after a long duration of network failure	User A calls user B. After the conversation is started, the network connection is deliberately broken for a long duration. Later, both SIP communicators will try to re-invite each other automatically. The broken call will be rebuilt and both users will hear the recorded conversation.	Works as assumed, the broken call is rebuilt and both users can hear the voice replaying.

4. Connect to voice mail server when unpredictable network disconnection happens	User A calls user B. After the call is started, the network connection is deliberately broken for a longer duration. Later, both SIP communicators cannot rebuild the broken call. The SIP communicator will connect to the voice mail server.	This function is not fully implemented. The SIP communicator connects to the mail server after the network connection is recovered. Without the DTMF function, the SIP communicator cannot interact with the voice mail server.

Table 6-4 Test cases

The test cases table summarizes that most tests work well. The result for the second test case indicates that the recorded voice came through with a large delay. As was described in the fourth chapter, the delay can be reduced by using silence suppression. On the other hand, the fourth test case could not be finished because the SIP communicator cannot interact with the voice mail server, as the DTMF function is not yet supported by the SIP communicator. More details of each test result are discussed in the next section.

6.2.2 Result validation

The SIP communicator is real-time communication software. Therefore, performance is one of the most important criteria for validation. The order of testing still follows the test cases which are listed in the previous section. We use Ethereal to capture the RTP packets from the Windows machine's WLAN interface for each scenario. See the analysis presented below.

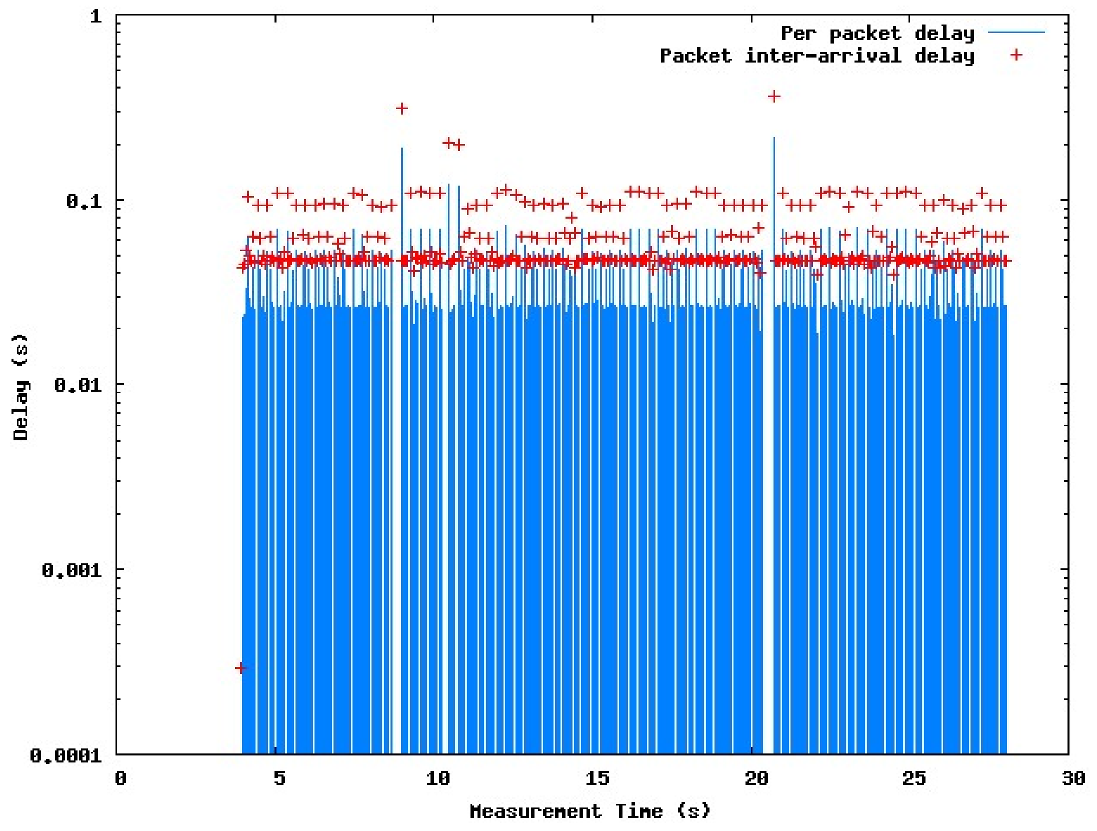


Figure 6-5 Normal call

Figure 6-5 depicts the RTP packets delay from test case 1. The average per packet delay is around 20ms. In the implementation, the G.729 codec is chosen as the default codec for audio. The media stream remains on a consistent rate of latency time, meanwhile, only a few packets take more than 100ms delay, which is acceptable for the RTP media session. Those occasional large packets delay could be caused by the concurrent applications. When the SIP communicator and the Ethereal are running on the same Windows machine, extra time is taken when the application processes are switching between each other. The user should not feel any obvious voice delay during the entire call session.

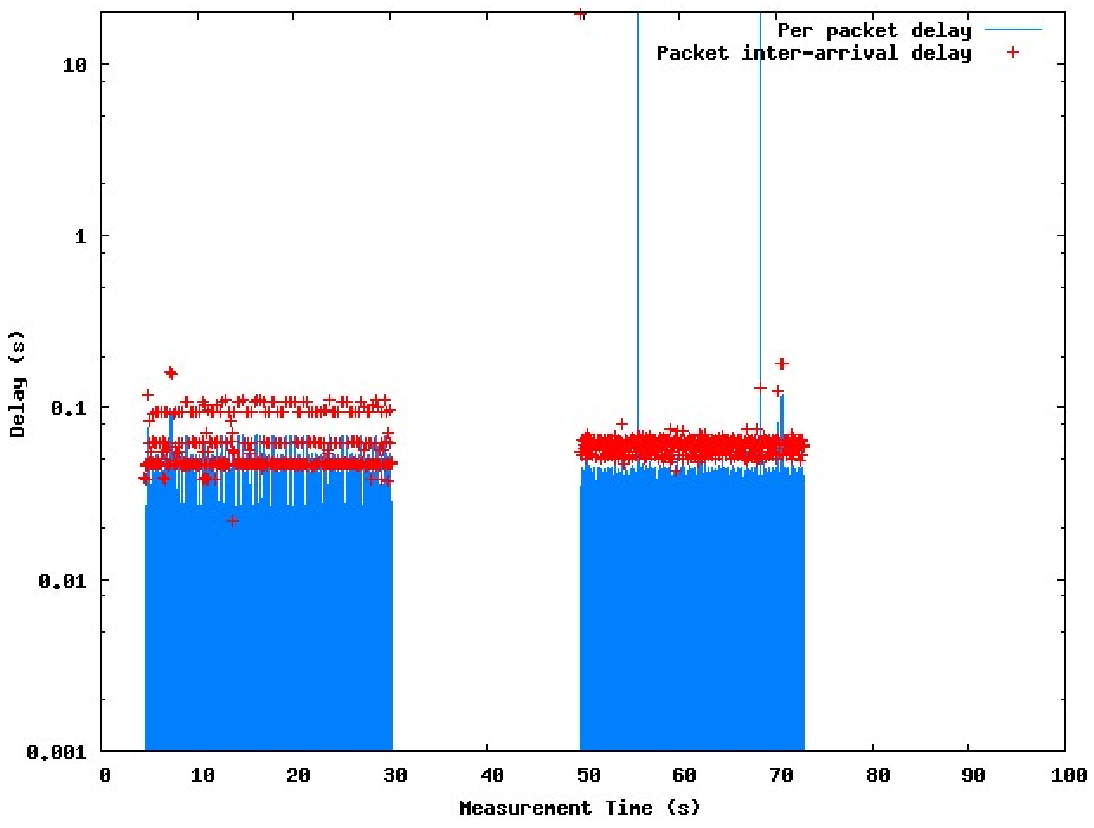


Figure 6-6 A short break in a call

Figure 6-6 shows the packets delay from test case 2 when a short network disconnection breaks the ongoing call. The network is deliberately broken for about 6 seconds, and then the network connection is recovered. However, the figure shows the disconnection time to be more than 6 seconds. There are two possible reasons which can result in longer disconnection time than expected. The first reason is the extra time needed for connecting to the wireless network. When the windows machine recaptures the wireless connection, it always takes several seconds for acquiring the network IP address from the DHCP server. The second reason was explained in the fourth chapter, which is caused by the imprecise RTP packets based detection mechanism. From the figure above we can see the time gap of the total disconnection time is around 18 seconds. The average packets delay is quite similar with the first test case, but occasionally a few packets come through the network with a large delay. Still, these packets should not affect the real-time conversation too much. The packets density after the network disconnection is a bit higher than before the network disconnection.

The reason lies in the replay mode where the packets are sent more regularly and frequently.

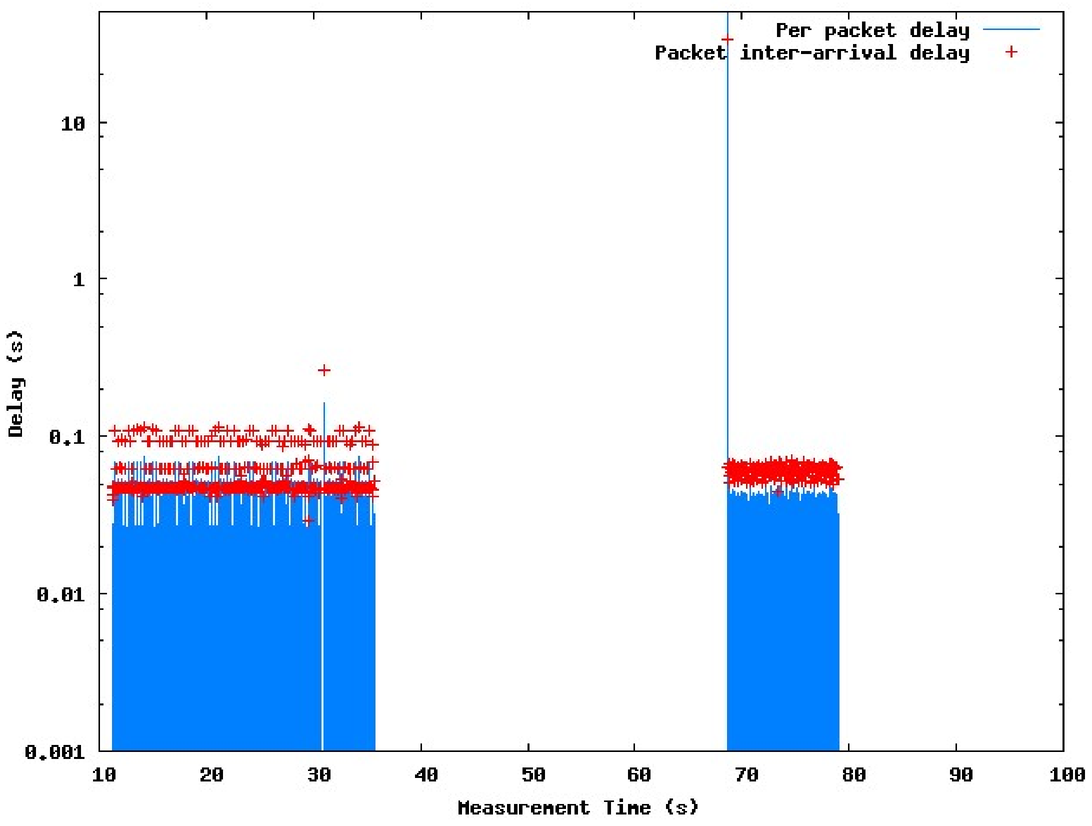


Figure 6-7 Rebuild a call after network disconnection

As Figure 6-7 depicts the third test case, the call is terminated by an equally long network disconnection. The broken call is rebuilt by re-inviting after the media path is recovered. Same as in the previous test case, the packets delay is kept to a consistent rate except few packets with large delay. Again, the occasional packets delay can be omitted in this case without affecting the entire media conversation. As in the previous test case, the packets density is a bit higher in the replay mode, but the average packets delay is kept at the same level.

The fourth test case is not presented in this section. Its behavior can be considered the same as in the third test case. The distinction between these two test cases is the different remote participant in the replay mode. The third test case is to rebuild a call to the remote SIP communicator, and it switches to the voice mail server in the fourth test case respectively.

6.3 Summary

A brief introduction of testing was given in this chapter. The demonstration setup explained the way of building the testing environment. There were four test cases defined against the use cases respectively. The results for those test cases were also analyzed in this chapter. The normal call worked quite smoothly. The behavior of the call with a short network disconnection worked as planned, but a large delay of the recorded voice was not as good as expected. The call with a long duration of network disconnection was rebuilt successfully after the network was recovered. In the fourth test case, the interaction with the SIP communicator and the voice mail server could not be executed because the DTMF function is not yet supported by the SIP communicator. The test results were acceptable for the prototype, but improvements are needed in the future. The final conclusion of this thesis is discussed in the next chapter.

7. Conclusion

This thesis has explored the disruption tolerance mechanism for SIP-based multimedia communication. The document started with an introduction of the background and motivation for this thesis, discussing different kinds of mobility services. The disruption tolerance mechanism was presented as a combined mobility service solution for unstable network communication. Then a brief introduction of the SIP basics is given which was necessary to understand this thesis. SIP for mobility services were discussed and some of these SIP mobility features were used for developing the prototype. This prototype was built on top of the SIP communicator. The implementation of the prototype started with the introduction of the technical background. The design architecture for the disruption tolerance mechanism was described first at the conceptual level. Three use cases were described in MSC charts that were differentiated by the duration of disconnection. Afterwards, the implementation details for the disruption tolerance mechanism were presented in UML figures. Finally, an experimental SIP network was built for the testing purpose. Each use case of the disruption tolerance mechanism was tested and the test result was analyzed.

The disruption tolerance mechanism for the SIP communicator is an enhancement of mobility service which helps a failed multimedia session to be recovered automatically. The disconnection detection mechanism has been proposed for end-to-end real time communication. The disconnection detection mechanism does not depend on the location of the media path failure and does not rely on the disconnection indication from any concrete network interfaces. This feature is especially useful to handle vertical handover when the connection breaks down. Three use cases for recovery the broken communication have been proposed which deal with the duration of network disconnection. A prototype has been implemented for the disruption tolerance mechanism which is built on top of the original SIP communicator. The implementation shows that the technical aspects of the disruption tolerance mechanism worked as well as expected. Basic testing was performed in an experimental SIP network to validate the disconnection handling functions, and the results were quite similar to expectations.

The prototype developed in this thesis has poor usability as mentioned previously. The silence suppression function is still missing in this prototype, which is the core function to reduce the time gap for the recovered conversation. The SIP communicator does not support the DTMF function, so the user cannot interact with the voice mail server. The user interface does not yet integrate the configuration functions for handling disconnection aspects, only the default aspects described in this thesis work are provided.

The experiment of the prototype has demonstrated the basic disruption tolerance functions are working well. Future work is needed to improve both usability and stability aspects mentioned above.

Reference

[1] Jukka Manner, Markku Kojo, "Mobility Related Terminology", RFC 3753, June 2004

[2] J. Rosenberg et al, Session initiation protocol", RFC 3261, June 2002

[3] M. Handley et al, "Session description protocol", RFC 4566, July 2006

[4] Henry Sinnreich, Alan B. Johnston, "Internet communications using SIP", WILEY, 2001

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

[6] J. Rosenberg, "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, September 2002

[7] R. Sparks, "Internet Media Type message/sipfrag", RFC 3420, November 2002

[8] SIP communicator official web site, https://sip-communicator.dev.java.net/

[9] J. Postel, user Datagram Protocol, RFC 768, 28 August 1980

[10] S. Donovan and J. Rosenberg, "Session Timers in the Session Initiation Protocol (SIP)", RFC 4028, April 2005

[11] Java media framework (JMF), http://java.sun.com/products/java-media/jmf/

[12] H. Schulzrinne, “RTP: A Transport Protocol for Real-Time Applications”, RFC 3550, July 2003

[13] H. Schulzrinne, “RTP Profile for Audio and Video Conferences with Minimal Control”, RFC 3551, July 2003

[14] Information Sciences Institute, “Transmission control protocol”, RFC 793, September 1982

[15] N.Banerjee et al. “Seamless SIP-Based Mobility for Multimedia Applications,” IEEE Network, Mar-Apr, 2006

[16] W.Wu et al. “SIP-Based Vertical Handoff Between WWANs And WLANs,” IEEE Network, Jun, 2005

[17] P. Resnick, “Internet Message Format”, RFC 2822, April 2001

[18] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997

[19] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005

[20] N.Freed, N.Borenstein, ”Multipurpose Internet Mail Extensions(MIME) Part Two: Media Types”, RFC 2046, November 1996

[21] SipX project, http://www.sipfoundry.org/

[22] BIND (Berkeley Internet Name Domain), http://www.isc.org/index.pl?/sw/bind/

[23] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “A Transport Protocol for Real-Time Applications”, RFC 3550, July 2003

[24] R. Sparks, “The Session Initiation Protocol (SIP) Refer Method”, RFC 3515, April, 2003

[25] .E. Perkins, “IP Mobility Support for IPv4,” RFC 3220, January 2002

[26] Q.Wang et al. “Mobility Management Architectures Based On Joint Mobile IP And SIP Protocols,” IEEE Wireless Commun, Dec, 2006