ENSC 833-3: NETWORK PROTOCOLS AND PERFORMANCE

Session Initiation Protocol User Agent Prototype Spring 2001

Final Project

by

Meng-Chauug Peter Lee(mclee@sfu.ca)
Kwok-Cheong Thomas Pang(ktpang@sfu.ca)

April 13, 2001

ABSTRACT

Internet telephony is evolving from its use as an "inexpensive" way to make long distance calls to a serious business telephony capability. Supporting the widespread use of Internet telephony requires a host of standardized protocols to ensure quality of services (QoS), transport audio and video data, provide directory services, and enable signaling. Signaling protocols are of particular interest because they enable such advanced services as mobility, universal numbers, multiparty conferencing, voice mail, and automatic call distribution. Two signaling protocols have been emerged to fill this need: ITU H.323 and IETF Session Initiation Protocol (SIP). Analysts expect SIP to overpass H.323 in the next two years because of its strength - simplicity, scalability, extensibility, and modularity.[3]

SIP is developed by Internet Engineering Task Force (IETF) and is modeled after the simple mail transfer protocol (SMTP) and the hypertext transfer protocol (HTTP). SIP is an application-layer control protocol for creating, modifying application-layer control protocol for creating, modifying and terminating sessions with one or more participants. These sessions include Internet multimedia conferences, Internet telephone calls and multimedia distribution.

In this project, we will implement SIP messages that are required for a basic session initiation, and SIP user agent. Due to its complexity and tight schedule of the course, proxy/redirect server will not be implemented. All the SIP messages and user agent will be written in C. In order to demonstrate the functionality and capability of each SIP component, the following call scenario will be demonstrated: simple call establishment and release, busy call, call not answer, and call holding.

ACKNOWLEDGEMENT

We would like to thank Professor Ljiljana Trajkovic for her approval, support, and feedback of this project. Further thank goes to Department of Engineer Science, Simon Fraser University for providing computing facilities for this project. We would also like to express our appreciation to the knowledge acquired from ENSC 833.

GLOSSARY

AAL2	ATM Adaptation Layer 2
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Service Digital Network
ISUP	SS7 ISDN User Part
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RTP	Real-Time Protocol
SCTP	Stream Control Transmission Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SS7	Signaling System 7
TCP	Transmission Control Protocol
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
VoATM	Voice over ATM
VoIP	Voice over IP

TABLE OF CONTENT

1. INTRODUCTION

The primary goals of this project are to understand the Session Initiation Protocol (SIP) and to implement the SIP messages that are required for a basic session initiation. SIP network servers (i.e., proxy and redirect) will not be implemented due to its complexity and tight schedule of the course. The SIP User Agent is developed using C programming language on SPARC Solaris platform. In order to demonstrate the functionality and capabilities of each SIP component, the following call scenario will be demonstrated: simple call establishment and release, busy call, call not answer and call holding.

1.1 Overview

SIP is a control protocol for creating, modifying and terminating sessions with one or more participants. Sessions can be internet multimedia conference, internet telephone calls and multimedia distribution. SIP is a lightweight protocol in which it requires only six primary methods for managing a basic session: INVITE, BYE, OPTIONS, ACK, REGISTER, CANCEL and INFO. INVITE request is used to invite a user to join a call; BYE terminates the call between two of the users on a call; OPTIONS requests information on the capabilities of a server (will be discussed later); ACK confirms that a client has received a final response to an INVITE; REGISTER provides the map for address resolution, i.e., letting a server know the location of other users; CANCEL ends a pending request, but does not end the call; and INFO is used to carry mid-call information. In this project, the following methods will be supported: INVITE, BYE, OPTIONS, and ACK.

SIP is independent of the packet layer. It is typically used over User Datagram Protocol (UDP), Transmission Control Protocol (TCP), or even Stream Control Transmission Protocol (SCTP). In this project, UDP is used for transporting SIP message because of its simplicity.

1.2 SIP Architecture

A SIP system has two components: user agents and network servers. A user agent is an end system that acts on behalf of someone who wants to participate in calls. A user agent contains both a protocol client called a user agent client (UAC), and a protocol server called a user agent server (UAS). The UAC is used to initiate a call, and the UAS is used to answer a call. The presence of both in a user agent enables peer-to-peer operation to take place using a client-server protocol [3].

SIP provides for two different types of network servers: proxy and redirect. A SIP proxy server receives requests, determines where to send these, and then forwards the request to the next server, i.e., it forks the request. A redirect server receives requests, but rather than passing these onto the next server, it sends a response to the caller indicating the

address of the called user, and the caller contacts the called party at the next server directly.

The main function of a SIP network server is to provide for name resolution and user location (i.e., very much like domain name server and router). Due to their complexity and the tight schedule of the course, SIP network servers will NOT be implemented in this project. To better understand SIP architecture, lets go through a simple call setup procedure with the system shown in figure 1.

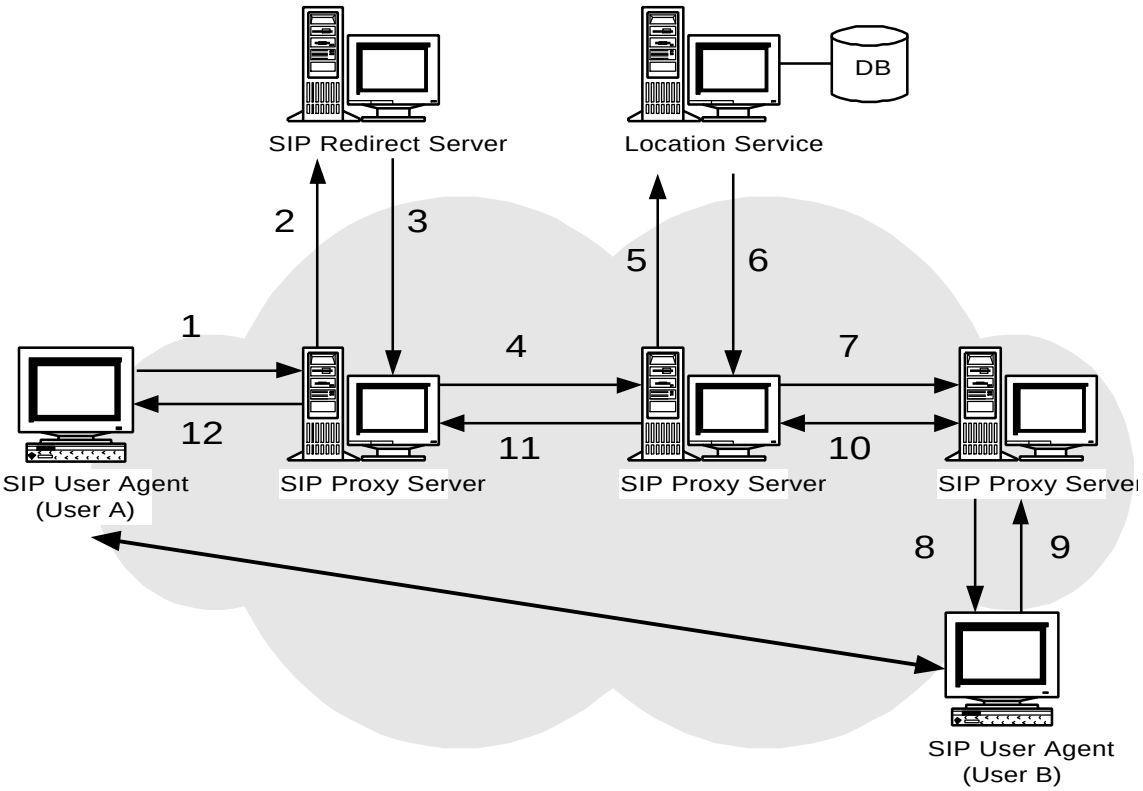


Figure 1. SIP Architecture

1. To initiate a session, the caller (User A) sends a request with a SIP URL of the called party (User B). If the client knows the location of the invited party, it can send the request directly to their IP address; if not, the client can send it to a locally configured SIP network server. In our example, the request is sent to SIP Proxy Server.
2. The SIP Proxy Server cannot resolve the called party's location and "forks" the request on behalf of User A to the redirect server.

3. Redirect server returns the address of the next hop. Please note that redirect server does NOT fork request.
4. Based on the returned address from the redirect server, the proxy server forks the request to the next hop.
5. The proxy server may consult a location server using other protocol.
6. The location server returns the address of the next hop.
7. Based on the returned address from the location server, the proxy server forks the request to the next hop.
8. This time the proxy knows the location of the called party and forks the invitation request to User B.

9-12. The acknowledgement message from User B will be returned to User A using the same path. As we will see later, the call setup involves multiple messages and the above procedures will therefore be repeated several times. Once the call setup procedure is complete, the voice path will directly be established between User A and B. For IP network, the voice stream will be carried over IP using Real Time Protocol (RTP). For ATM network, the voice stream will be carried using AAL2.

Another entity that comprises SIP is SIP Registrar. The user agent sends a registration message to the SIP Registrar and the Registrar stores the registration information in a location service via a non-SIP protocol. Once the information is stored, the Registrar sends the appropriate response back to the user agent. Please see Figure 2.

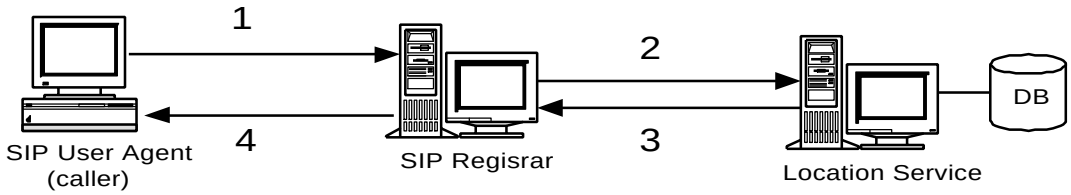


Figure 2. SIP Registrar

1.3 SIP Messages

SIP messages can be classified into request and response. A SIP request message consists of three elements: Request Line, Header, and Message Body. A SIP response message contains status line, header and message body. The request line and header fields define the nature of the call in terms of services, addresses, and protocol features. The message body is independent of the SIP protocol and can contain anything. The status line contains a 3-digit integer result code indicating the outcome of the attempt to understand and satisfy the request, and optionally the reason phrase.

SIP use the Session Description Protocol (SDP) for describing audio, video and multimedia sessions. The SDP will be included in the body part of a SIP message.

Figure 3 is an example of a sample basic session initiation and release using SIP. Machine A is initiating a session, and B is terminating the session.

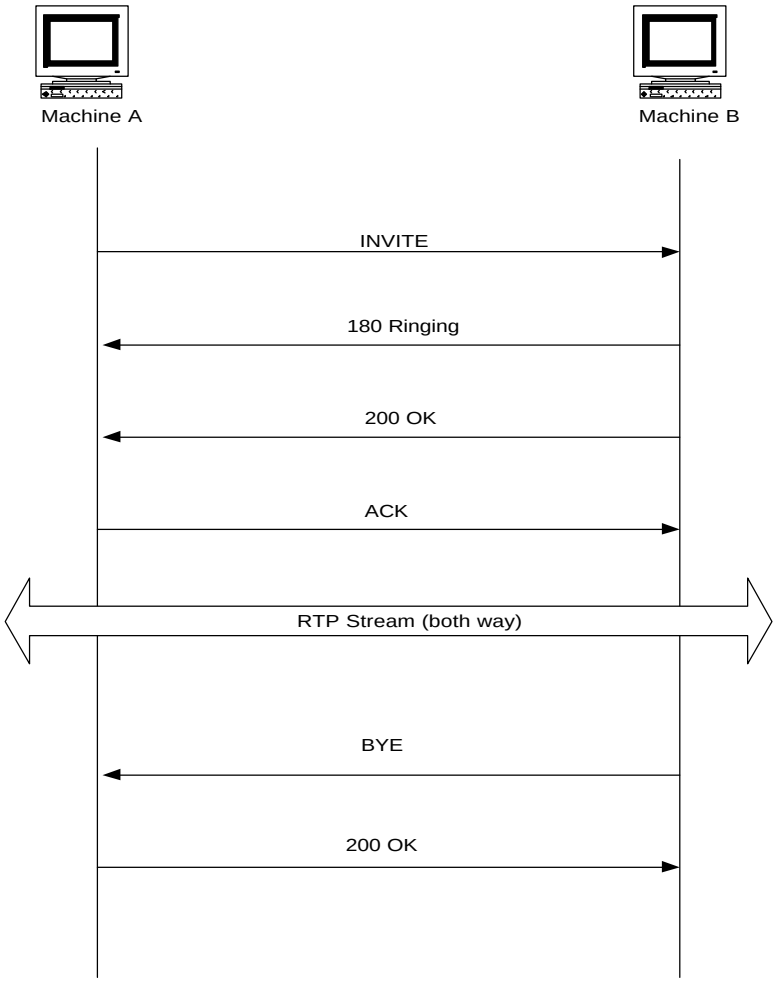


Figure 3. Basic SIP Call Message Flow

2 PROJECT REQUIREMENTS

Below summarizes the requirements and assumptions of this project:

- To implement User Agent Client and User Agent Server
- To handle the following call scenarios:
    - Normal call setup and release
    - Busy call
    - Call not answer
    - Call Holding
- In order to fulfill (1) and (2), the following SIP messages will be implemented: INVITE, ACK, BYE, 180 RINGING, 200 OK, 400 Bad Request.
- Assume the media type in the SIP request is always application/sdp.
- Session Description Protocol (SDP) will not be implemented and it will be faked in the body part of the SIP message.
- Proxy server will NOT be implemented. Messages such as TRYING and fields such as Record-Route, Route, and Server will NOT be supported.
- Only HTTP-like URL will be supported.
- Support IP call only, i.e., call generated from PSTN will not be supported. Therefore, the mapping of ISUP message to SIP message or how ISUP messages are carried by SIP message will NOT be considered. Please refer to [5] if reader is interested in supporting PSTN call.
- Simultaneous sessions will not be supported. Therefore, transaction management is not required in this project.

3 HIGH LEVEL DESIGN

3.1 System Overview

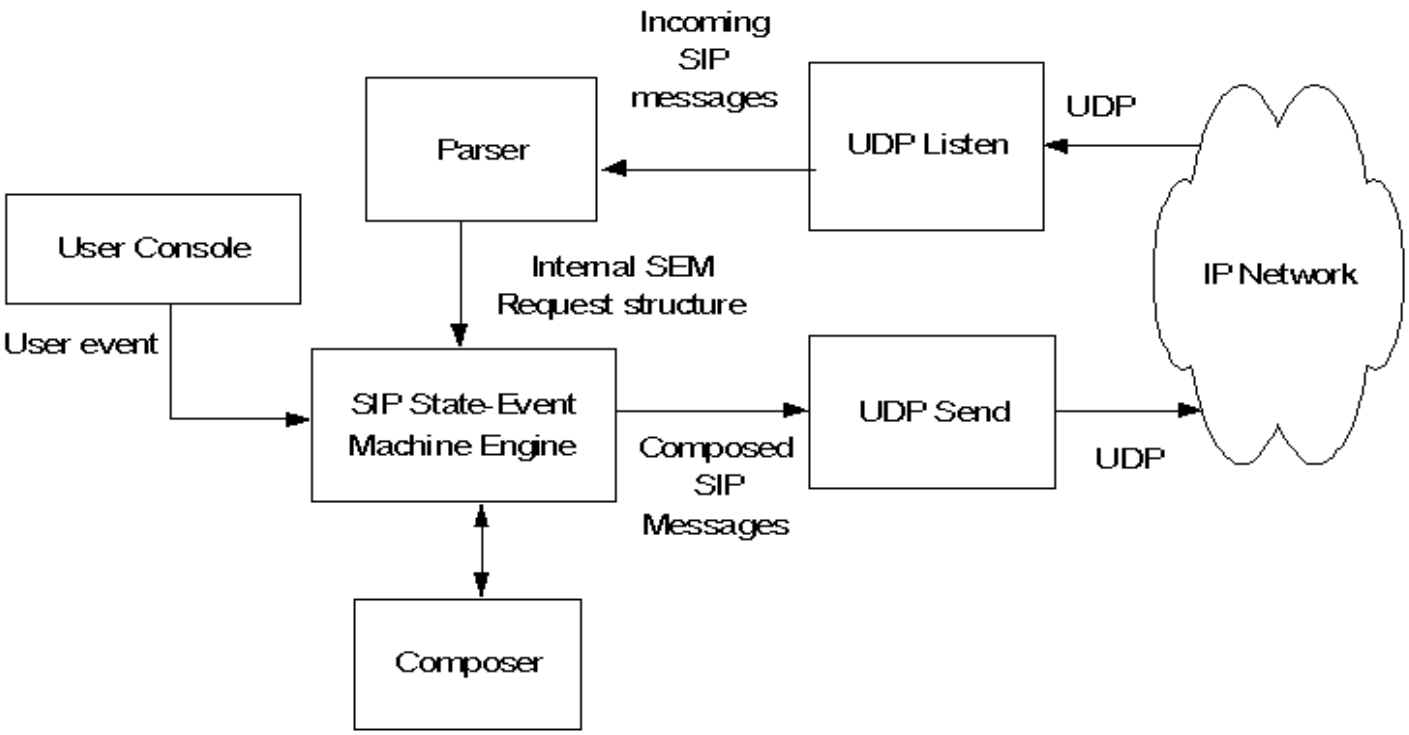


Figure 4. SIP User Agent System Diagram

Our SIP User Agent consists of the following components:

- User Console for initiating call, replying/rejecting call, and configuring the user agent.
- Composer - for building SIP request and response messages.
- Parser - for parsing the incoming SIP messages and converting them into internal SIP message structure sip_msg_t. Please see Data Structure section for detail.
- State machine - maintains the state of a SIP session and defines the action and the transition to be taken in response to the external events. Please see State Machine section for detail. It is running as a thread, called it sip_sem.
- UDP Transport (send and listen) - for receiving and sending SIP messages via the configured UDP port. To allow concurrent receipt of events from User Console and SIP messages from remote User Agent, "UDP Listen" is run as a thread, called it sip_udp_listen. Sip_sem and sip_udp_listen are communicating using shared memory.

3.2 Data Structure

Below are primary data structure definitions for our SIP User Agent program. Most of them are self-explanatory from their structure names.

```
/* SIP Event */
typedef enum sip_event
{
    sip_ui_invite_evt = 0,
    sip_ui_bye_evt,
    sip_ui_hold_evt.
    sip_ui_resume_evt,
    sip_ui_answer_evt,
    sip_invite_evt,
    sip_bye_evt,
    sip_hold_evt,
    sip_resume_evt,
    sip_rsp_ok_evt,
    sip_rsp_nok_evt,
    sip_rsp_ack_evt,
    sip_rsp_busy_evt,
    sip_rsp_ring_evt,
    sip_timeout_evt,
    sip_last_evt
}
sip_event;


/* SIP Message Type */
typedef enum sip_msg_type
{
    SIP_INVITE = 0,
    SIP_OK,
    SIP_NOK,
    SIP_ACK,
    SIP_BYE,
    SIP_BUSY,
    SIP_RING,
    SIP_CANCEL,
    SIP_OPTIONS,
    SIP_REGISTER,
    SIP_LAST_MSG_TYP
}
sip_msg_type;


/* SIP Via field */
typedef struct sip_via_t {
    char    address[32];
    int     udp_port;
}
sip_via_t;
```

```
/* SIP URL */
typedef struct sip_url_t {
    char    username[32];
    char    hostname[32];
    char    alias[16];
    int         tag;
}
sip_url_t;


/* Media type of the SIP message body */
typedef enum sip_media_typ
{
    SIP_APPL_SDP = 0,
    SIP_TEXT_HTM,
    SIP_LAST_MEDIA
}
sip_media_typ;


/* SIP Message */
typedef struct sip_msg_t {
    char          version[5];
    sip_msg_type  msg_type;
    sip_event     event_type;
    sip_url_t     from;
    sip_url_t     to;
    sip_via_t     via;
    sip_url_t     contact;
    char          callid[32];
    int           cseq;
    sip_media_typ media;
    int           content_len;
    sip_msg_type  rspto;           /* msg type of the orig. req. */
    int           ret;             /* return code */
}
sip_msg_t;


/* SIP Session State-Event Machine structure */
typedef struct sip_sem_tbl_t {

    sip_state state;   /* next state */

    /* Event handler
     * evt – sip event
     * cseq – call sequence number
     * status – status of the handler execution
     */
    void (*fn) (sip_event evt, int cseq, int *status);

} sip_sem_tbl_t;
```

3.3 Function Prototypes

This section describes the API interfaces that are exported to the developers of the SIP User Agent program.

Function	SIP_COMPOSE_MSG
Purpose	To compose SIP message (applicable to both request and response)
Syntax	char * sip_compose_msg(sip_msg_t *msg, char *body);
Input Parameters	msg - pointer to internal SIP message structure sip_msg_t body - pointer to the body text to be included in the SIP message
Output Parameters	None
Return	Composed message

Function	SIP_COMPOSE_SDP
Purpose	To mimic SDP message for the body part of a SIP message. For event of sip_ui_hold_evt, the connection address of the connection information line will be set to 0.0.0.0; otherwise, it will be set to the IP address of the local host.
Syntax	char * sip_compose_sdp(sip_url_t *url, sip_event evt);
Input Parameters	url – SIP URL evt – Event to SIP user agent program
Output Parameters	None
Return	Composed SDP

Function	SIP_NEW_TAG
Purpose	To generate a new tag for SIP message
Syntax	int sip_new_tag(void);
Input Parameters	None
Output Parameters	None
Return	New tag to be used in SIP message

Function	SIP_NEW_CSEQ
Purpose	To generate a call sequence number for SIP message
Syntax	int sip_new_cseq(void);
Input Parameters	None
Output Parameters	None
Return	New call sequence number to be used in SIP message

Function	SIP_GET_TAG
Purpose	To obtain the current tag for SIP message

Syntax	int sip_get_tag(void);
Input Parameters	None
Output Parameters	None
Return	Current tag to be used in SIP message

Function	SIP_GET_CSEQ
Purpose	To obtain the current call sequence number for SIP message
Syntax	int sip_get_cseq(void);
Input Parameters	None
Output Parameters	None
Return	Current call sequence number to be used in SIP message

Function	SIP_PARSE_MSG
Purpose	To parse the incoming SIP message
Syntax	int sip_parse_msg(char *msg, sip_msg_t *rsp);
Input Parameters	msg – incoming SIP message
Output Parameters	rsp – parsed message in internal SIP structure
Return	1 – the incoming message is SIP response 0 – the incoming message is SIP request

Function	SIP_UDP_LISTEN
Purpose	To listen UDP port for SIP message. The received message will be placed into the shared memory and set the flag indicating the receipt of the message. This function will be running as a thread.
Syntax	void* sip_udp_listen(void *arg);
Input Parameters	Pointer to arguments (not used)
Output Parameters	None
Return	None

Function	SIP_UDP_SEND_MSG
Purpose	To send UDP message to the designated UPD port of the specified host.
Syntax	int sip_udp_send_msg(char *msg_out, int rmtport, char *hostname);
Input Parameters	msg_out - point to the message to be sent out rmtport - udp port of the target host hostname - hostname of the target host
Output Parameters	None
Return	0 – success -1 – failure

Function	SIP_SEM
Purpose	SIP state-event main control loop. This function will be running as a thread.
Syntax	void* sip_sem(void *arg);
Input Parameters	None
Output Parameters	None
Return	None

Function	SIP_DB_OPEN
Purpose	To open shared memory for use in the SIP user program
Syntax	sip_db_t* sip_db_open(int* exist_p);
Input Parameters	None
Output Parameters	exist_p – flag to indicate if the shared memory has been allocated or not.
Return	Pointer to the shared memory

Function	SIP_HANDLER_X
Purpose	Event handler of SIP state-event machine
Syntax	void sip_handler_X(sip_event evt, int cseq, int *status);
Input Parameters	evt – sip event cseq – call sequence number
Output Parameters	status – status of the handler execution
Return	None

3.4 State Machine

SIP session state machine is the brain of the SIP User Agent (UA). It defines the behavior of the UA that expects the following events:

UI_INVITE - Call initiation from the input interface
UI_BYE - Call termination from the input interface
UI_HOLD - Call holding from the input interface
UI_RESUME - Call resume from the input interface
UI_ANSWER - Answer call from the input interface
INVITE - Call initiation from the remote end
BYE - Call termination from the remote end
HOLD - Call holding from the remote end
RESUME - Call resume from the remote end
RSP_OK - 200 OK from remote end
RSP_NOK - 400 Bad Request from remote end

RSP_ACK	- ACKnowledgement from remote end
RSP_BUSY	- 486 Busy from remote end
RSP_RINGING	- 180 Ringing from remote end
TIMEOUT	- Timeout before the expected response is arrived

The following states are expected:

IDLE	- Session is free for use
WAITING	- Waiting response for the INVITE request
CONNECTED	- Session connected
HOLDING	- Session is being put on hold
PAUSE	- Session is put on hold
RESUMING	- Session is being resumed
DISCON	- Session is being disconnected

Since we are not going to implement proxy server, some messages such as TRYING are not expected. Methods such as CANCEL and INFO are not supported neither.

The table below describes the session state machine of our SIP User Agent (applicable to both client and server). Each entry inside the matrix specifies the next state of the session and the number specifies the action to be taken.

Table 3.1 State Event Matrixes

Event \ State	IDLE	WAITING	CONNECTED	HOLDING	PAUSE	RESUMING	DISCON
UI_INVITE	WAITING, 1	WAITING, 5	CONNECTED, 5	HOLDING, 5	PAUSE, 5	RESUMING, 5	DISCON, 5
UI_BYE	IDLE, 3	IDLE, 3	DISCON, 19	DISCON, 19	DISCON, 19	DISCON, 19	DISCON
UI_HOLD	IDLE, 3	WAITING, 3	HOLDING, 15	HOLDING, 3	PAUSE, 3	RESUMING, 3	DISCON, 3
UI_RESUME	IDLE, 3	WAITING, 3	CONNECTED, 3	HOLDING, 3	RESUMING, 18	RESUMING, 3	DISCON, 3
UI_ANSWER	IDLE, 3	WAITING, 7	CONNECTED, 3	HOLDING, 3	PAUSE, 3	RESUMING, 3	DISCON, 3
UI_CANCEL	IDLE, 3	IDLE, 23	IDLE, 23	IDLE, 23	IDLE, 23	IDLE, 23	IDLE, 23
INVITE	WAITING, 2	WAITING, 6	CONNECTED, 6	HOLDING, 6	PAUSE, 6	RESUMING, 6	DISCON, 6
BYE	IDLE, 4	IDLE, 24	IDLE, 14	IDLE, 14	IDLE, 14	IDLE, 14	IDLE, 14
HOLD	IDLE, 4	WAITING, 3	HOLDING, 7	HOLDING, 4	PAUSE, 4	RESUMING, 4	DISCON, 4
RESUME	IDLE, 4	WAITING, 3	CONNECTED, 4	HOLDING, 4	RESUMING, 7	RESUMING, 4	DISCON, 4
RSP_OK	IDLE, 4	CONNECTED, 9	CONNECTED, 4	PAUSE, 16	PAUSE, 4	CONNECTED, 9	IDLE, 17
RSP_ACK	IDLE, 4	CONNECTED, 8	CONNECTED	PAUSE, 22	PAUSE	CONNECTED, 8	DISCON, 4
RSP_BUSY	IDLE, 4	IDLE, 10	CONNECTED, 4	HOLDING, 4	PAUSE, 4	RESUMING, 4	DISCON, 4
RSP_RINGING	IDLE, 4	WAITING, 11	CONNECTED, 4	HOLDING, 4	PAUSE, 4	RESUMING, 4	DISCON, 4
TIMEOUT	IDLE, 3	IDLE, 12	IDLE, 13	IDLE, 13	IDLE, 13	IDLE, 13	IDLE, 13
RSP_NOK	IDLE, 21	WAITING, 21	CONNECTED, 21	HOLDING, 21	PAUSE, 21	RESUMING, 21	DISCON, 21

Notes of using the above state machine:

- The number right besides the next state in the SIP Session State Machine (SM) indicates the action to be taken. Do nothing but simply change state as per the SM if there is no number besides.
- When printing out an error message, please specify the current state and the event received.
- Only the party who initializes the call hold can issue RESUME request.
- Two timer objects should be created in each session: timer for request/response pair and the maximum session hold time. Therefore, the TIMEOUT event occurs when the User Agent does not receive the expected response / event within the configured timeout period, or when the maximum session time expires. In this project, there is no limit applied to total session period.
- Re-transmission will not be implemented in this prototype.

Actions used in the state machine:

1. Issue INVITE to remote end.
2. Issue RINGING (optional) or OK (depending on configuration) to the remote end.
3. Print out the error log indicating invalid request/event.
4. Issue "400 Bad request" to the remote end. The remote should print out error indicating invalid request.
5. Print out error indicating the session in use, i.e., Busy session.
6. Issue "486 Busy" to the remote end
7. Issue "200 OK" to remote end.
8. Activate RTP transmission
9. Issue ACK to remote end and activate RTP transmission
10. Print out error indicating the remote end is busy.
11. Play waiting tone locally.
12. Print out error indicating request timed out.
13. Issue BYE to remote end and terminate RTP session
14. Issue "200 OK" and terminate RTP session
15. Issue INVITE (c=0) to remote end
16. Issue ACK to remote end and de-activate RTP transmission
17. De-activate RTP transmission
18. Issue INVITE to put the remote end off hold
19. Issue BYE to remote end
20. Do Nothing
21. Print out error indicating bad request
22. Issue "200 OK"
23. Handle User cancel call routine

3.5 Program Flow

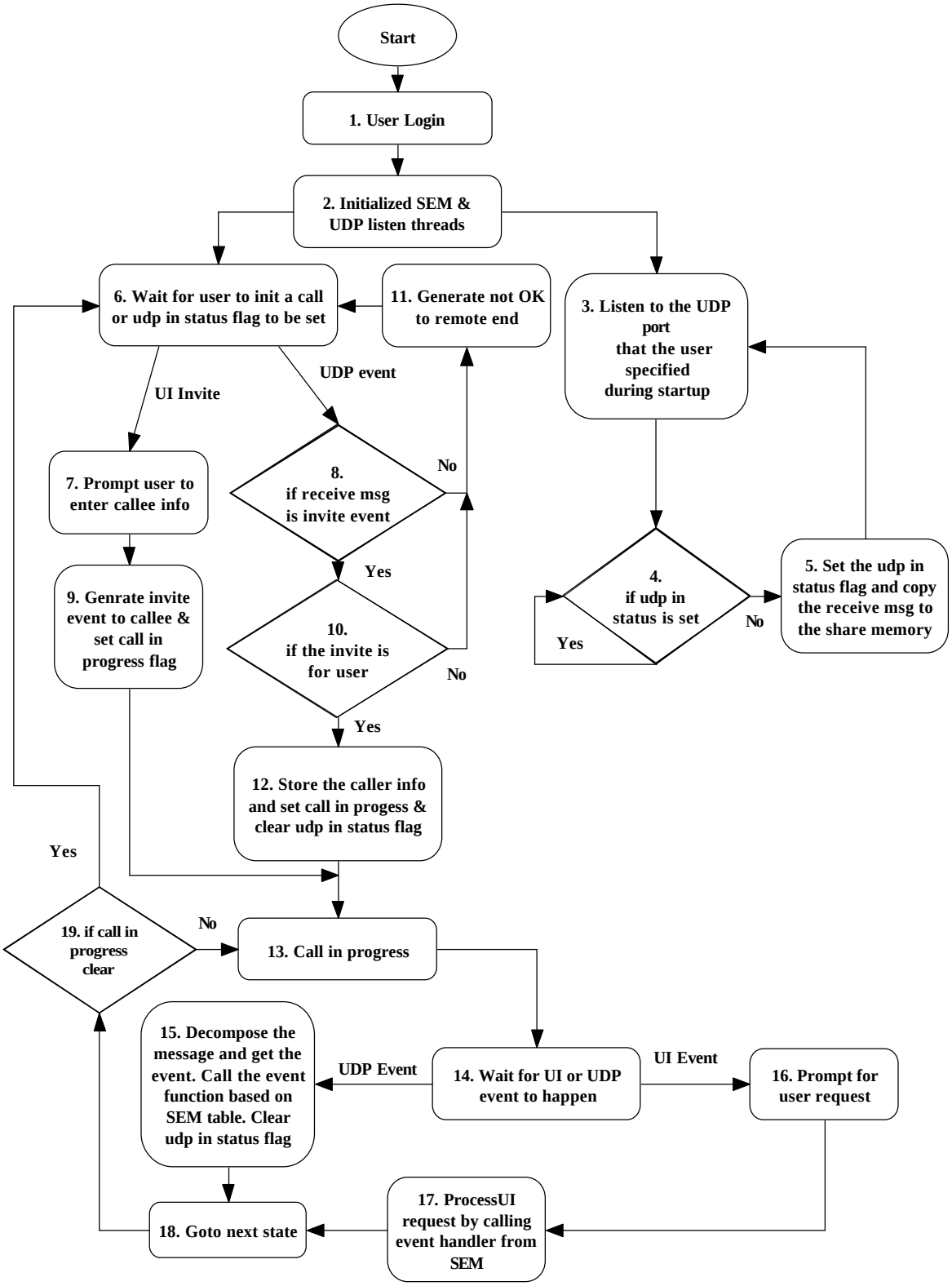


Figure 5. Program Flow Chat

Program Flow Description:

1. Program start. Prompt user to input User Name, Alias and UDP for listening.
2. Initialize the State Machine Thread and the UDP Listen Thread.
3. This is a loop that keep listening to the user specified UDP port for incoming message.
4. Receive new message, check the UDP_IN status flag, if this flag is set, this indicated that the previous receive message haven't been process by the SEM yet and wait here until the flag get clear.
5. Save the receive message to the shared memory and set the UDP_IN flag, this flag is also act as a UDP event trigger point. SEM is periodically checking for this flag.
6. Waiting for User to press enter to initialize a call or waiting UDP message coming from the network.
7. User presses enter to make a call. It will prompt the user to enter callee information: User Name, Alias, UDP port and Host Name and got to step 9.
8. Check the receive message is an Invite event. If yes, it will proceed with step 10, else it jump to step 11.
9. Callee information will be stored in a temporally memory and generate an Invite event to the callee at remote end.
10. Check whether the Invite is for the User that is login. If yes, it proceed with step 12, else it will jump to step 11.
11. Clear the UDP_IN flag and go back to step 6 to wait for User Input or UDP event.
12. Store the Caller information, set the Call In Progress flag, clear the UDP_IN flag and move on to step 13.
13. This is the Call In Progress loop enter point and will stay as long as the call is in progress.
14. Waiting for User Input and UDP receive event to be happened. If it detects an UDP event, it will proceed to step 15. If it detects a User Input event, it will proceed to step 16.
15. Call the parser routine to decompose the receive message and find out the remote request event. It will then pass the event to the SEM and clear the UDP_IN flag. The SEM will call the routine according to the state event matrix that shown in Table 3.1. It will then move on to step 18.
16. It will ask to the to select one of the following selections: End Call, Hold Call, Release Call, Answer Call and Cancel Call. It will pass the according event to the SEM and proceed with step 17.
17. The SEM will call the routine according to the state event matrix that shown in Table 3.1. It will then move on to step 18.
18. Move to next state.
19. Check the Call In Progress flag is clear. If yes, it will go back to step 6 to wait for user to initialize call again or waiting for remote party to call. Else it continues in the call in progress and move on to step 13. Note: The call in progress will be cleared in the routine that handle end call or cancel call.

4 CODING GUIDELINE

Different developers have different program style. This section specifies the minimal coding guideline that developers in this project should follow for three primary reasons:

- Improving quality of the software
- Facilitating software maintenance and future enhancement
- Improving clarity of the code, i.e., better understanding for readers

Below lists the coding guideline for this project:

- 3-letter module name: SIP
- All files in SIP module should start with "sip_", for example, sip_ua.c
- All data structures should be prefixed with sip_ and ended with _t. For example, struct sip_header_t {...}
- Try to use typedef for all structure definitions. Subsequent data definition based on those structures does not require the quantifier "struct".
- Avoid any unnecessary casting.
- No warning or error messages are allowed in the compilation of the final code.
- Check pointer prior to de-referencing it or segmentation fault
- Always initialize storage before using it.
- All functions should have the following heading and they should be ended with the function name. For example:

```
/*
 *   Purpose:
 *
 *   In:
 *       parm1 -
 *       parm2 -
 *
 *   Out:
 *       parm3 -
 *
 *   Return:
 *   Note:
 */
 void sip_ensc833_func(void)
 {

 } /* sip_ensc833_func */
```

- Try to implement all the messages specified in the protocol spec., even though we are not using all of them. Put some comments for those messages not supported. Provide reference for each message. For example:

```
/* SIP URL (RFC 2543 - section 2) */
typedef struct sip_url_t
{
    sip_userinfo_t user_info;
    sip_hostport_t hostport;
    sip_transport_t transport;
    int ttl;
    char maddr[16];
    sip_user_t userparm;
    sip_method_t method;
    char uuid[16];

    /* other parm not supported */
    /* telephone subscriber not supported */
}
sip_url_t;
```

- Replace nested if-else statements by case statement if possible.
- Always take error/exception into consideration.

5 TEST RESULTS

The following call scenarios will be tested: basic call setup and release, call holding, call not answer, busy call, invalid called party, and unknown host of the target user agent.

5.1 Test Environment

The test will be performed on SPARC Solar machine in room ASB 9884, Engineering Science building of Simon Fraser University. The following machines will be used:

- lorentz (IP address: 199.60.6.109)
- hall (IP address: 199.60.6.25)
- faraday (IP address: 199.60.6.19)

Below describes the start up script on each machine:

lorentz

```
lorentz 27: /ensc/grad1/ktpang/March21> bin/sip

**************************************
SIP User Agent Program
Authors: Peter Lee, Thomas Pang
Copyright @ 2001 Simon Fraser University

**************************************
Please Enter user name : tpang
Please Enter user alias : thomas
Please Enter local udp port : 4000
Please Enter SIP Version : 2.0
Automatic Call Answer(Y/N): N
Maximum call waiting period (in sec): 10

sip_user_session_info.user_name = tpang
sip_user_session_info.alias = thomas
sip_user_session_info.local_udp_port = 4000
sip_user_session_info.sip_version = 2.0
sip_user_session_info.hostname = lorentz

Press Return to make a call.
```

hall

```
hall 26: /ensc/grad1/ktpang/March21/bin> sip

**************************************
SIP User Agent Program
Authors: Peter Lee, Thomas Pang
Copyright @ 2001 Simon Fraser University

**************************************
Please Enter user name : mclee
```

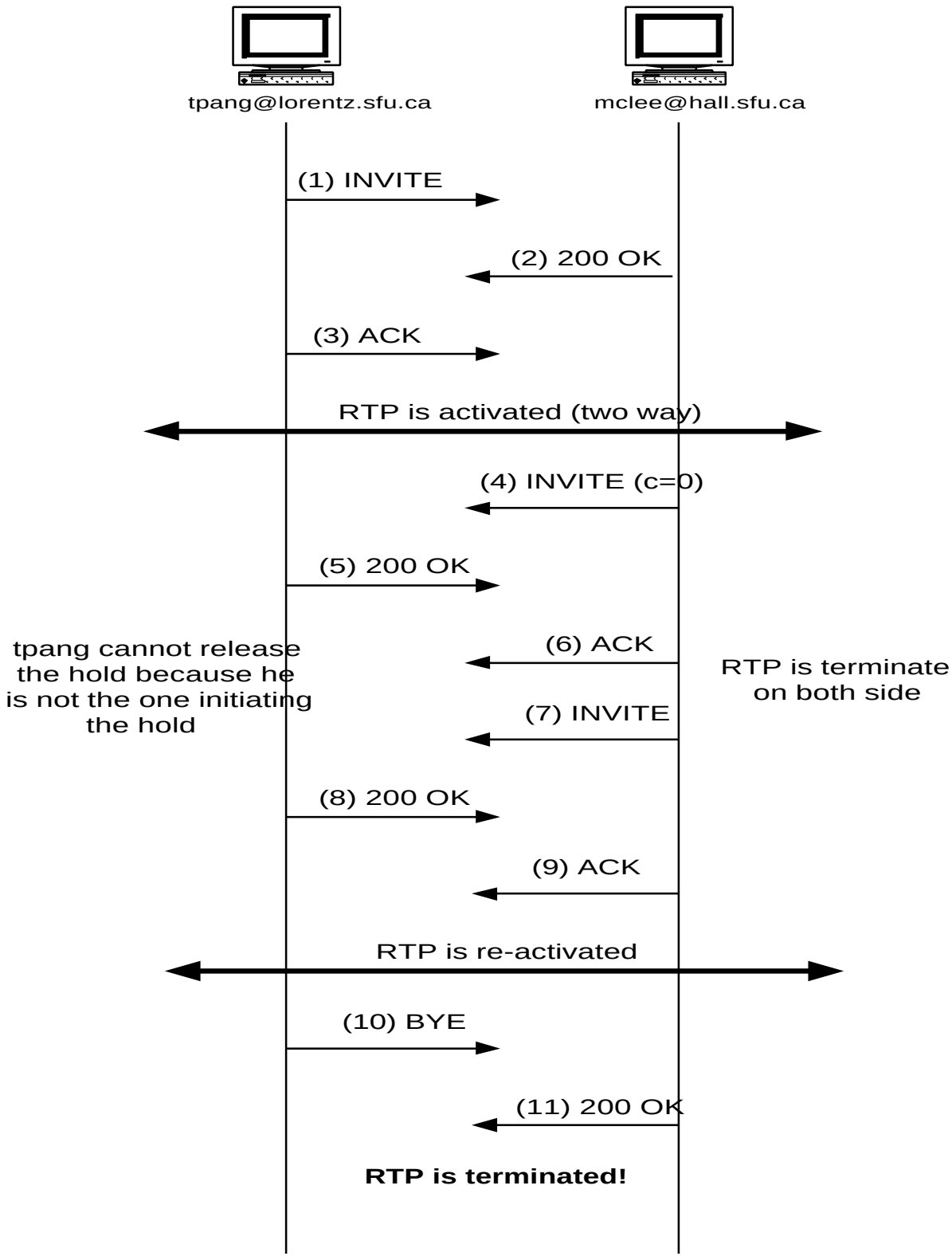
```
Please Enter user alias : peter
Please Enter local udp port : 4000
Please Enter SIP Version : 2.0
Automatic Call Answer(Y/N): Y
Maximum call waiting period (in sec): 10

sip_user_session_info.user_name = mclee
sip_user_session_info.alias = peter
sip_user_session_info.local_udp_port = 4000
sip_user_session_info.sip_version = 2.0
sip_user_session_info.hostname = hall

Press Return to make a call.
```

faraday

```
faraday 23: /ensc/grad1/ktpang/March21/bin> sip

**************************************
SIP User Agent Program
Authors: Peter Lee, Thomas Pang
Copyright @ 2001 Simon Fraser University

**************************************
Please Enter user name : userC
Please Enter user alias : userC
Please Enter local udp port : 4000
Please Enter SIP Version : 2.0
Automatic Call Answer(Y/N): Y
Maximum call waiting period (in sec): 10

sip_user_session_info.user_name = userC
sip_user_session_info.alias = userC
sip_user_session_info.local_udp_port = 4000
sip_user_session_info.sip_version = 2.0
sip_user_session_info.hostname = faraday

Press Return to make a call.
```

5.2 Basic Call Setup and Release, Call Holding

This test case will verify the basic call setup and release of our SIP user agent. It will also demonstrate the support of call holding.


tpang@lorentz.sfu.ca
mclee@hall.sfu.ca
(1) INVITE
(2) 200 OK
(3) ACK
RTP is activated (two way)
(4) INVITE (c=0)
(5) 200 OK
tpang cannot release the hold because he is not the one initiating the hold
(6) ACK
RTP is terminate on both side
(7) INVITE
(8) 200 OK
(9) ACK
RTP is re-activated
(10) BYE
(11) 200 OK
RTP is terminated!

```
1. Press Return to make a call.

   --- Please enter the Callee inforamtion ---
   Username: mclee
   Alias: peter
   Hostname: hall
   Callee UDP port: 4000


   ---- Message Sent ----
   INVITE sip:mclee@hall SIP/2.0
   Via: SIP/2.0/UDP lorentz:4000
   From: thomas <sip:tpang@lorentz>
   To: peter <sip:mclee@hall>
   Call-ID: 1@lorentz
   CSeq: 1 INVITE
   Contact: thomas <sip:tpang@lorentz>
   Content-Type: application/sdp
   Content-Length: 146

   v=0
   o=thomas 2890844527 2890844527 IN IP4 lorentz
   s=Session SDP
   c=IN IP4 199.60.6.109
   t=3034423619 0
   m=audio 3456 RTP/AVP 0
   a=rtpmap:0 PCMU/8000


2. ---- Message received ----
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP lorentz:4000
   From: thomas <sip:tpang@lorentz>
   To: peter <sip:mclee@hall>
   Call-ID: 1@hall
   CSeq: 1 INVITE
   Contact: thomas <sip:tpang@lorentz>
   Content-Type: application/sdp
   Content-Length: 141

   v=0
   o=peter 2890844527 2890844527 IN IP4 hall
   s=Session SDP
   c=IN IP4 199.60.6.25
   t=3034423619 0
   m=audio 3456 RTP/AVP 0
   a=rtpmap:0 PCMU/8000


3. ---- Message Sent ----
   ACK sip:mclee@hall SIP/2.0
   Via: SIP/2.0/UDP lorentz:4000
   From: thomas <sip:tpang@lorentz>
   To: peter <sip:mclee@hall>
   Call-ID: 1@lorentz
```

```
   CSeq: 1 ACK
   Content Length:0
```

RTP is activated at this point, i.e., tpang can now talk to mclee.

4. mclee put tpang on hold.

```
   ---- Message received ----
   INVITE sip:tpang@lorentz SIP/2.0
   Via: SIP/2.0/UDP hall:4000
   From: peter <sip:mclee@hall>
   To: thomas <sip:tpang@lorentz>
   Call-ID: 1@lorentz
   CSeq: 1 INVITE
   Contact: peter <sip:mclee@hall>
   Content-Type: application/sdp
   Content-Length: 141

   v=0
   o=thomas 2890844527 2890844527 IN IP4 lorentz
   s=Session SDP
   c=IN IP4 0.0.0.0
   t=3034423619 0
   m=audio 3456 RTP/AVP 0
   a=rtpmap:0 PCMU/8000
```

5.
```
   ---- Message Sent ----
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP hall:4000
   From: peter <sip:mclee@hall>
   To: thomas <sip:tpang@lorentz>
   Call-ID: 1@lorentz
   CSeq: 1 INVITE
   Contact: peter <sip:mclee@hall>
   Content-Type: application/sdp
   Content-Length: 0
```

6.
```
   ---- Message received ----
   ACK sip:tpang@lorentz SIP/2.0
   Via: SIP/2.0/UDP hall:4000
   From: peter <sip:mclee@hall>
   To: thomas <sip:tpang@lorentz>
   Call-ID: 1@lorentz
   CSeq: 1 ACK
   Content Length:0
```

RTP is terminated, session is in hold stage!
Please note that tpang cannot release the hold because it is mclee initiating the call hold.

```
   Press Return for more options.

   1. End Call.
```

```
   2. Hold Call
   3. Hold Release
   4. Answer Call
   5. Cancel Call

   Please make your selection: 3
   Invalid hold release.
```

7. mclee releases the hold.

```
   ---- Message received ----
   INVITE sip:tpang@lorentz SIP/2.0
   Via: SIP/2.0/UDP hall:4000
   From: peter <sip:mclee@hall>
   To: thomas <sip:tpang@lorentz>
   Call-ID: 1@lorentz
   CSeq: 2 INVITE
   Contact: peter <sip:mclee@hall>
   Content-Type: application/sdp
   Content-Length: 142

   v=0
   o=thomas 2890844527 2890844527 IN IP4 hall
   s=Session SDP
   c=IN IP4 199.60.6.25
   t=3034423619 0
   m=audio 3456 RTP/AVP 0
   a=rtpmap:0 PCMU/8000
```

8.
```
   ---- Message Sent ----
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP hall:4000
   From: peter <sip:mclee@hall>
   To: thomas <sip:tpang@lorentz>
   Call-ID: 1@lorentz
   CSeq: 2 INVITE
   Contact: peter <sip:mclee@hall>
   Content-Type: application/sdp
   Content-Length: 0
```

9.
```
   ---- Message received ----
   ACK sip:tpang@lorentz SIP/2.0
   Via: SIP/2.0/UDP hall:4000
   From: peter <sip:mclee@hall>
   To: thomas <sip:tpang@lorentz>
   Call-ID: 1@lorentz
   CSeq: 2 ACK
   Content Length:0

   The RTP is re-activated!
```

10. tpang terminates the call.

```
   Press Return for more options.

   1. End Call.
   2. Hold Call
   3. Hold Release
   4. Answer Call
   5. Cancel Call

   Please make your selection: 1
   Issue BYE to remote end.


   ---- Message Sent ----
   BYE sip:mclee@hall SIP/2.0
   Via: SIP/2.0/UDP lorentz:4000
   From: thomas <sip:tpang@lorentz>
   To: peter <sip:mclee@hall>
   Call-ID: 1@lorentz
   CSeq: 2 BYE
   Content Length:0


11.   ---- Message received ----
   SIP/2.0 200 OK
   Via: SIP/2.0/UDP lorentz:4000
   From: thomas <sip:tpang@lorentz>
   To: peter <sip:mclee@hall>
   Call-ID: 1@lorentz
   CSeq: 2 BYE
   Contact: thomas <sip:tpang@lorentz>
   Content-Type: application/sdp
   Content-Length: 0

   RTP is terminated!
```

5.3 Call Not Answer

mclee@hall calls tpang@lorentz but tpang does not answer. Eventually, the call session will be timed out and the session will be terminated. Please note that tpang have "automatic call answer" disabled and the maximum call waiting time for both users are set to 10 seconds (please see section 5.1).

Below are the traces for both tpang and mclee.

tpang@lorentz

```
---- Message received ----
INVITE sip:tpang@lorentz SIP/2.0
Via: SIP/2.0/UDP hall:4000
From: peter <sip:mclee@hall>;tag=97616
To: thomas <sip:tpang@lorentz>
Call-ID: 1@hall
CSeq: 3 INVITE
```

```
Contact: peter <sip:mclee@hall>;tag=97616
Content-Type: application/sdp
Content-Length: 141

v=0
o=peter 2890844527 2890844527 IN IP4 hall
s=Session SDP
c=IN IP4 199.60.6.25
t=3034423619 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000


---- Message Sent ----
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP hall:4000
From: peter <sip:mclee@hall>
To: thomas <sip:tpang@lorentz>
Call-ID: 1@lorentz
CSeq: 3 INVITE
Content Length:0

Paying Ring Tone.
Do Doooo Doooo.
Do Doooo Doooo....
```

No answer!

mclee@hall

```
--- Please enter the Callee inforamtion ---
Username: tpang
Alias: thomas
Hostname: lorentz
Callee UDP port: 4000


---- Message Sent ----
INVITE sip:tpang@lorentz SIP/2.0
Via: SIP/2.0/UDP hall:4000
From: peter <sip:mclee@hall>;tag=97616
To: thomas <sip:tpang@lorentz>
Call-ID: 1@hall
CSeq: 3 INVITE
Contact: peter <sip:mclee@hall>;tag=97616
Content-Type: application/sdp
Content-Length: 141

v=0
o=peter 2890844527 2890844527 IN IP4 hall
s=Session SDP
c=IN IP4 199.60.6.25
t=3034423619 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

```
---- Message received ----
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP hall:4000
From: peter <sip:mclee@hall>
To: thomas <sip:tpang@lorentz>
Call-ID: 1@lorentz
CSeq: 3 INVITE
Content Length:0

Playing Ring Tone.
dooooo do do
dooooo do do ...
```

No answer!

5.4 Busy call

While tpang is talking to mclee, userC make a call to mclee. The call attempt will be rejected because mclee's session is in use. Please see the result below:

```
--- Please enter the Callee inforamtion ---
Username: mclee
Alias: peter
Hostname: hall
Callee UDP port: 4000


---- Message Sent ----
INVITE sip:mclee@hall SIP/2.0
Via: SIP/2.0/UDP faraday:4000
From: userC <sip:userC@faraday>
To: peter <sip:mclee@hall>
Call-ID: 1@faraday
CSeq: 1 INVITE
Contact: userC <sip:userC@faraday>
Content-Type: application/sdp
Content-Length: 144

v=0
o=userC 2890844527 2890844527 IN IP4 faraday
s=Session SDP
c=IN IP4 199.60.6.19
t=3034423619 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000



Press Return for more options.
```

```
---- Message received ----
SIP/2.0 486 Busy
Via: SIP/2.0/UDP faraday:4000
From: userC <sip:userC@faraday>
To: peter <sip:mclee@hall>
Call-ID: 1@lorentz
CSeq: 5 INVITE
Content-Type: application/sdp
Content-Length: 0

Invite is not success: Remote session is busy!!
```

5.5 Unknown hostname

Attempt to make a call with unknown hostname and it will be reject as below:

```
--- Please enter the Callee inforamtion ---
Username: tpang
Alias: thomas
Hostname: hello
Callee UDP port: 4000

Target host not found!
```

5.6 User not found

UserC@faraday tries to make a call to Peter at the machine "hall" but put down a wrong username plee instead of mclee. When the call arrives hall, it will be discarded. The call invitation at faraday will eventually be timed out.

```
--- Please enter the Callee inforamtion ---
Username: plee
Alias: peter
Hostname: hall
Callee UDP port: 4000


---- Message Sent ----
INVITE sip:plee@hall SIP/2.0
Via: SIP/2.0/UDP faraday:4000
From: userC <sip:userC@faraday>
To: peter <sip:plee@hall>
Call-ID: 1@faraday
CSeq: 1 INVITE
Contact: userC <sip:userC@faraday>
Content-Type: application/sdp
Content-Length: 144

v=0
o=userC 2890844527 2890844527 IN IP4 faraday
s=Session SDP
c=IN IP4 199.60.6.19
```

```
t=3034423619 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000

<after 10 seconds>
```

No answer!

6 CONCLUSIONS

As we have seen in this project, SIP is a very simple, lightweight signaling protocol for creating, modifying and terminating sessions with one or more participants. It is client-server protocol and is independent of the packet layer. The protocol is an open standard and is scalable. SIP is more or less equivalent to the Q.931 and H.225 components of H.323. These protocols are responsible for call setup and call signaling. SIP system primarily consists of user agent, network server, and registrar. In this project, we have successfully developed SIP User Agent, both client and server for peer-to-peer operations. The program is written in C and is running on SPARC Solaris platform. We have successfully demonstrated the following call handling: basic call setup and release, call holding, busy call, call not answer, invalid called party, and unknown target host.

Since SIP has been gaining popularity in VoIP arena, analysts predict it will surpass H.323 in the next several years. We therefore suggests School of Engineering Science, Simon Fraser University, the following tasks for the future enhancement of our SIP User Agent program:

- support the rest of the basic SIP methods: REGISTER, CANCEL, OPTIONS
- implement transaction manager in the existing user agent program to allow multiple simultaneous SIP sessions
- implement SIP network servers (proxy and redirect) and registrar
- improve exception handling of the existing SIP user agent program
- include SDP
- support different call forwarding including 700-, 800-, 900- type calls, no answer, busy, unconditional, and other address-translation services
- verify the capability to reach a called party under a single, location-independent address even when the user changes terminals
- invitations to multicast conferences
- investigate how SIP call procedure affects the traffic pattern in the packet network

7 REFERENCES

[1] RFC 2543bis-02, SIP: Session Initiation Protocol, IETF, November 24, 2000

[2] SIP Telephony Service Examples, IETF, November 2000

[3] Henning G.Schulzinne and Jonathan D.Rosenberg, "The Session Initiation Protocol: Providing Advanced Telephony Services Across the Internet", Bell Labs Technical Journal, October- December 1998.

[4] "Overview of the SIP protocol", the SIP Center
http://www.sipcenter.com/overview.htm

[5] SIP for Telephones (SIP-T): Context and Architectures, IETF, November 21, 2000.

APPENDIX I - Interim Status (2001/03/08)

Below is our project plan proposed to Prof. Ljiljana Trajkovic on January 21, 2001 with the interim status:

Task	Target Date	Status
1. Preliminary study of SIP	2001/02/02	Done
2. Requirement Specifications	2001/02/11	Done
3. High Level Design	2001/02/18	Done
4. Coding	2001/03/18	60%
5. Testing	2001/03/24	20%
6. Demo Planning	2001/03/24	20%
7. Class Presentation	2001/03/29	-
8. Actual Demo	2001/04/03	-
9. Final write up	2001/04/06	-

As of March 8, 2001, our project is on schedule. We intend to complete the entire project by April 6, 2001 instead of April 13, 2001 proposed by the professor.

APPENDIX II – Code Listing

Below is the list of source files for our SIP User Agent program:

sip.h - contains all data structures being used by the SIP user agent program
sip_global.h – contains all the global variables being used by the SIP user agent program
sip_proto.h – function prototypes
sip.c – SIP User Agent main program which creates two thread: sip_udp_listen and sip_sem
sip_db_open.c – contains functions for allocating and opening shared memory for use in the SIP user agent program
sip_process_msg.c - contains functions for composing and parsing SIP messages
sip_sem.c - SIP session state machine and handlers
sip_udp_send_msg.c – to send the SIP message to other SIP user agent
sip_udp_listen.c – listen to the UDP port
sip_test.c - test driver for sip_process_msg.c
Makefile – for building the software