SIPC – a SIP user agent

Xiaotao Wu and Henning Schulzrinne
{xiaotaow,hgs}@cs.columbia.edu
Department of Computer Science
Columbia University, New York

April 27, 2001

Abstract

SIPC is a SIP user agent used for communicating with the other SIP endpoints or SIP servers. SIPC uses the SIP (Session Initiation Protocol) for signaling. It can send and handle basic SIP requests including INVITE, CANCEL, BYE, ACK, REGISTER and OPTIONS. In addition, it supports some SIP extensions including SIP for presence (SUBSCRIBE and NOTIFY request), SIP for instant messaging (MESSAGE request) and SIP for device control (DO request). SIPC does not do media transmission. Instead, it uses external media applications to handle the media transmission.

SIPC version 1.5 does not contain a complete service module. But it can provide some service-related functionalities such as uploading scripts to a SIP server. Ongoing sipc-related projects are focusing on service creation and execution.

1 Introduction

SIPC uses the SIP to do call signalling, supports both UDP and TCP transmission, runs on Unix, Linux and Windows. SIPC supports audio, video, text, whiteboard and desktop sharing as media types. There is an address book in SIPC for maintaining address related information such as SIP address, phone number, picture of the person and call history. SIPC can handle a call session to another SIP user agent, register the contact information to a SIP registrar server, work as a presence agent for the user, maintain a instant message session between another SIP user agent and control X10 devices through SIP-X10 gateway. In addition, SIPC provides some simple client side services such as call forwarding and call screening based on the predefined policies. The ongoing work for SIPC is to systematically build a service creation and execution environment.

1.1 Related work

A number of SIP user agent implementations have appeared since the first SIP bake-off. There are hardware phones such as Cisco SIP phone, 3Com SIP phone, Pingtel Phone and the software phone and stacks such as Vovida SIP implementations, DynamicSoft SIP implementations etc. The information about these implementations can be found at `http://www.cs.columbia.edu/~hgs/sip/implementations.html`.

1.2 Outline of the rest of the paper

The rest of the paper will describe SIPC version 1.5 in detail. Section 2 discusses the architecture of SIPC. Section 3 describes the functionality of SIPC. Section 4 provides the detailed information of SIPC's implementations. Section 5 discusses further works. The final section briefly summarizes SIPC's development history.

2 The architecture of SIPC

2.1 Design goals

The following things were considered when designing SIPC.

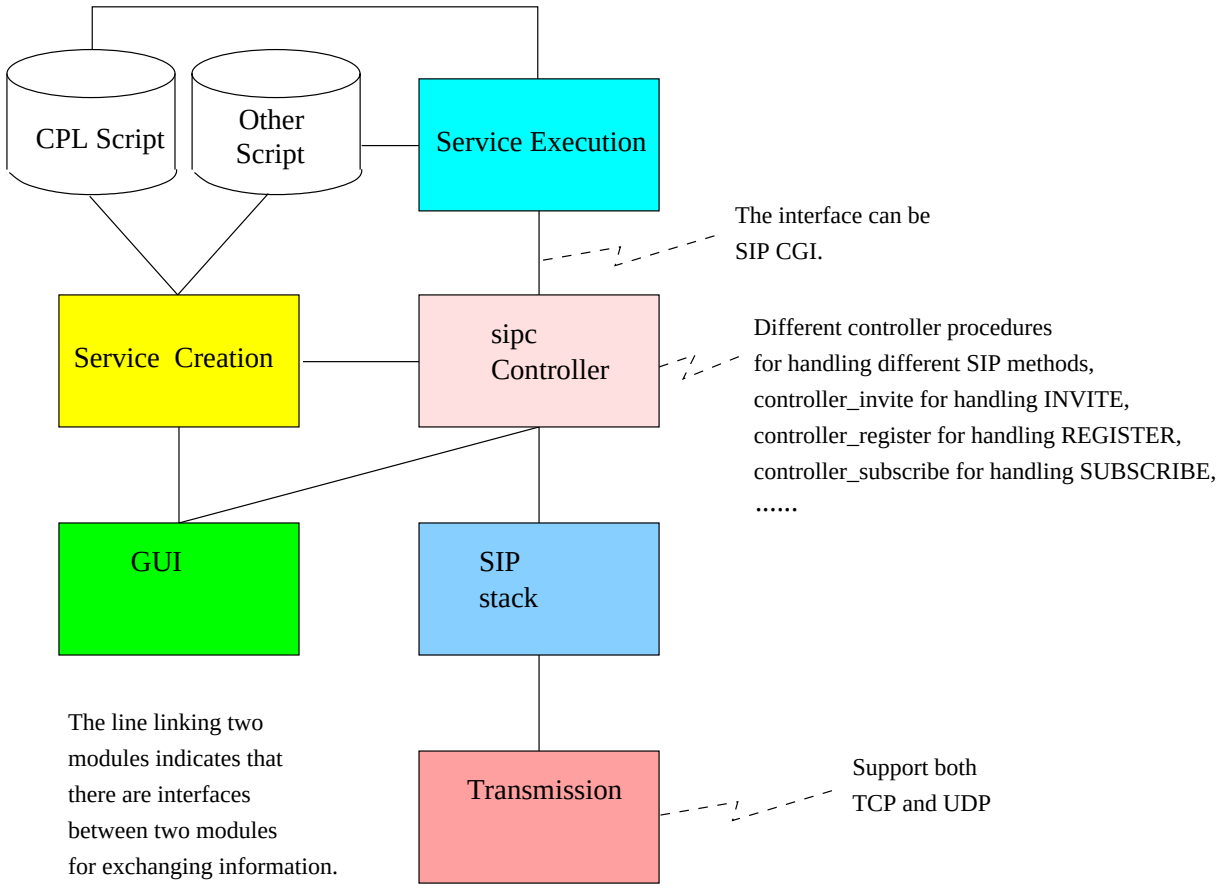

Figure 1: Architecture of sipc

Because we may need to use part of the code in SIPC to do some other projects such as building a HTTP-SIP gateway for web device control, building a SIP stack that can be separated from the whole program and can be re-used in the other applications is one of our design goals.

Since SIP has some extensions, to support new SIP method easily will certainly be one of the design goals. One of our design decisions for supporting this goal is to put all the operations that are common for all the different SIP methods such as handling Record-Route or parsing error in the SIP stack. By this way, when a developer wants to support a new SIP method, the developer only needs to deal with the headers specific for this SIP method.

SIPC needs to support both TCP and UDP. However, the SIP stack in SIPC should not care about what kind of protocol (TCP or UDP) is used for transmission. The SIP stack part only cares about the content of the packet. Hence, there should be only one common interface for transmission. To achieve this, we designed to have a transmission module to handle all the packet sending and receiving work.

In a telecommunication system, roughly, there are three layers, namely service layer, signaling layer and media layer. The separation between the service layer and the signaling layer can bring many benefits. With considering this, we designed to have a separate service module for service execution. (Service creation will be an independent project and won't be discussed in this document, though service creation part can be integrated as one of the subsystems in SIPC)

2.2 Modules

Figure 1 shows the architecture of SIPC. In this architecture, the transmission module handles the network events and calls the functions in the sipstack module for parsing the incoming packets. The sipstack module parses the SIP packets, keeps the transaction status information and deals with the retransmissions. The controller module collects

information from the service module and the sipstack module, interacts with the user through the GUI module, integrates all the information together to generate appropriate SIP signaling packets and invokes and controls the external media applications. The service creation and service execution modules provide services such as call screening to SIPC. These two modules are loosely connected to the controller module.

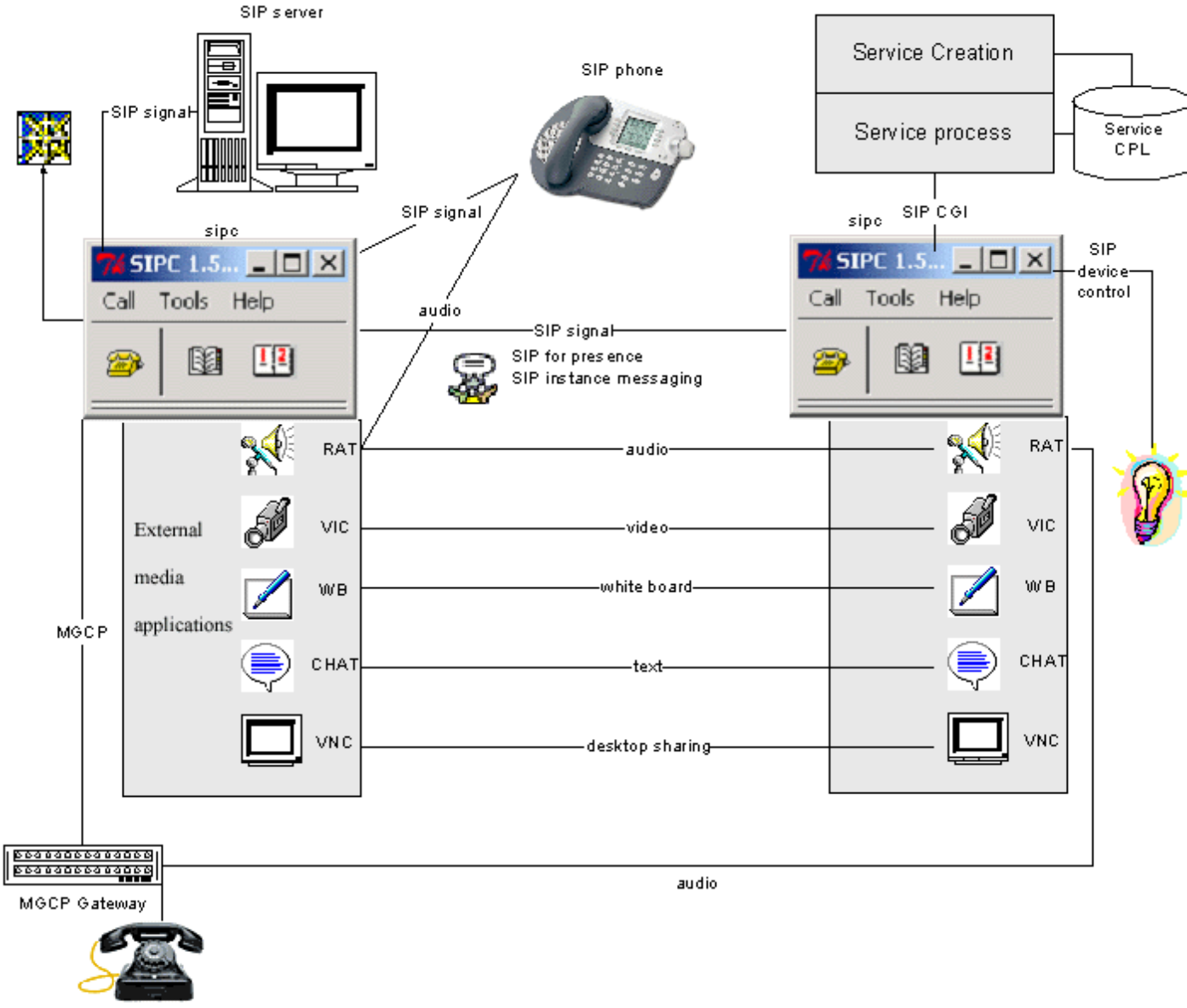

Figure 2: Functionality of sipc

When the transmission module receives an incoming packet, it passes the packet to the sipstack module. The sipstack module parses the packet and calls the handlers registered by the controller module to handle the request or response. The controller module gets the packet, collects information from the GUI and the service module for handling the packet, and then generate appropriate outgoing SIP headers and message content. The controller module passes the outgoing SIP header list and the message content to the sipstack. The sipstack then composes the headers and the content into SIP packet and pass the packet to the transmission module for sending out.

3 Functionality

SIPC can perform the functions shown in Figure 2:

- SIPC can handle basic phone calls with different media. The media transmission is handled by external media applications. Currently, SIPC can handle audio, video, white board, text and desktop sharing.
- SIPC can handle device control [1]. SIPC can issue the DO command for device control.
- SIPC can handle instant message [2]. SIPC can send the MESSAGE command for instant message.
- SIPC can handle SIP for presence [3]. SIPC can send and handle the SUBSCRIBE and NOTIFY command for presence information.
- SIPC can register to SIP registrar servers and upload service scripts such as CPL [4] or SIP CGI [5] scripts to the SIP servers.
- SIPC can use MGCP [6] to control MGCP gateway. (Not fully done)

3.1 Basic SIP Messages

SIPC can handle the following SIP messages specified in SIP specification [13]: INVITE, ACK, CANCEL, BYE, REGISTER and OPTIONS. The detail of the message handling is described below:

3.1.1 INVITE

SIPC can send INVITE request through the SIP outbound server such as sipd or directly to the other SIP user agent.

SIPC's INVITE request contains the following headers: Call-ID, CSeq, Via, From, To, Date, Subject, Call-Info, Priority, Expires, Contact, Content-Type, Content-Length and Route.

SIPC only puts the SDP message as the content of the outgoing INVITE request. The SDP message contains the information about the media. SIPC's INVITE request can support audio, video, white board, text and desktop sharing as media types.

For an incoming INVITE request, the following responses can be generated by SIPC

180 Ringing: SIPC sends this response immediately to the caller as soon as it receives the INVITE request. This response will stop the retransmission in the caller side.

200 Ok: When the user accept the call, SIPC will send this response to the caller. This response contains media information as its message content described in the SDP.

400 Bad Request: If SIPC's sipstack can not understand the request, SIPC will send this response back to the caller.

486 Busy Here: When the callee is in busy status (the callee chooses the 'Busy' button in the incoming call GUI), SIPC will send this response back.

603 Decline: When the callee is not willing to answer the call, (the callee chooses the 'Decline' button in the incoming call GUI), SIPC will send this response back.

Most of the headers in the response are copied from the request. SIPC will append a `tag` to the To header in the response.

The content of the response contains media information of the callee side.

To acquire the port number used for media transmission, SIPC will first check the port with the same port number as the caller. If that port number is not available, SIPC will try to find another available port.

SIPC can handle the following responses for an outgoing INVITE:

1xx: When SIPC receives 1xx response, it will stop retransmission. If the response is 180, SIPC begins to play the ring-back audio.

200: When SIPC receives 200 response, it will start the media application based on the SDP part of the response.

3xx: When SIPC receives 3xx response, it will redirect the call based on the URL in the Contact header of the response. If the URL is http:// or mailto: URL, SIPC will start Netscape to browse the web page or send email. If there are multiple URLs in the Contact header, SIPC will either automatically try all the contact URLs or ask the user to pick up one of the URLs to try. The configuration option 'Automatically try all the address in redirect response' in SIPC's Preference menu is used to decide which way to use.

4xx, 5xx, 6xx: When SIPC receives responses with the response code higher than 400, it will display the status message and stop the call session.

When the user want to change the media types during the call, SIPC can issue an INVITE with the same Call-ID but higher CSeq number. This is called re-INVITE. Re-INVITE is also used in a mobile environment when the IP address of the SIP user agent is changed. When SIPC receives an INVITE request with the same Call-ID as the existing call, it will check the CSeq number of the request. If it's higher than the existing CSeq number corresponding the Call-ID, this request is considered as a re-INVITE. If the re-INVITE request is about the IP address changing, SIPC will automatically change the IP addresses for the media applications (currently, restart the media application with the new IP address). If the re-INVITE is about to add or remove a media, SIPC will confirm with the callee for agreeing on doing so.

SIPC will send ACK for the final responses (A final response is a 2xx, 4xx, 5xx or 6xx response).

3.1.2 ACK

The ACK will be sent by SIPC for all the final responses for INVITE. If there is a Record-Route header in the response, SIPC will base on the URLs in the Record-Route header to set the destination which the ACK request will be sent to. (In fact, this is handled in the sipstack module.)

3.1.3 CANCEL

SIPC can send and handle CANCEL for an existing INVITE request. If a 180 response for the INVITE request has been received and the caller begins the ringing back, the caller side will stop the ringing back when the CANCEL request is issued. If a final response has been received, the caller cannot issue a CANCEL request.

When SIPC receives a CANCEL request, it first checks whether there is an existing INVITE request matches this CANCEL request (The INVITE request's Call-ID, CSeq, From and To headers will be matched against the CANCEL request's.) If there is, SIPC will then check the status of the session. If the final response for this INVITE has already been sent, SIPC will ignore the CANCEL request. If the final response for this INVITE has not been sent, SIPC first stops ringing, then sends 487 response for the INVITE request and sends 200 response for the CANCEL request. The session will be terminated after the response being sent out.

3.1.4 BYE

SIPC can send and handle BYE request. When SIPC sends a BYE, it stops all the media applications related to this call immediately without waiting for the 200 response from the other side.

When SIPC receives a BYE request, it will first check whether there is an existing call for this BYE (The INVITE request's Call-ID, CSeq, From and To headers will be matched against the BYE request's). If the caller (the UA which sends the INVITE) sends BYE, the callee (the UA which receives the INVITE) will match the From header of the BYE to the From header of the INVITE and the To header of the BYE to the To header of the INVITE. If the callee sends the BYE, the caller will match the From header of the BYE to the To header of the INVITE and the To header of the BYE to the From header of the INVITE. If there is an existing call matching the BYE, SIPC will send back a 200 OK response and stop all the media applications related to this call. Otherwise, SIPC will send back a 481 response.

3.1.5 REGISTER

SIPC can send REGISTER request and handle the response for the REGISTER request. SIPC can do third-party registration (The From and the To address in the REGISTER request are different). SIPC can unregister a registration. (Set with expiration time of the registration as 0). SIPC can automatically refresh registrations 30 seconds before their expiration time. SIPC can maintain multiple registrations and can delete registration locally. SIPC can also upload service scripts (CPL or SIP CGI script) in the REGISTER request. When SIPC get 200 response from

server, it first checks the validity of the response, then it tries to find the expiration information in the Expires header or in the response's Contact header (The Contact header in the response may contain multiple contact information, some of them may not be related to the contact URLs SIPC has registered for) and setups the refreshing time. If the expiration time is 0, SIPC will remove the registration from the local registration table. If the response contains message content, SIPC considers the message content as service script and save the message content into the file ~/.sipc/.upload_<username>_<servername>]

When SIPC gets a 401 response, if the registrar requires basic authentication, or if the registrar requires digest authentication and the value of the parameter 'stale' in the WWW-Authorization header is FALSE (which means the failure of the authentication is not due to the stale nonce value but something else), SIPC will pop up a dialog box to ask for the username and password. If the registrar requires digest authentication and the value of the parameter 'stale' in the WWW-Authorization header is TRUE (which means the user and the password are correct, but the nonce value is stale), SIPC will not ask the user for the username and password again, instead, it will use the same username and password as those in the original REGISTER request, and the new nonce value that indicated in the response to calculate the digest value and send REGISTER again with the new digest value.

3.1.6 OPTIONS

SIPC can handle incoming OPTIONS request but can not handle the response of an outgoing OPTIONS request. Thus, SIPC will not issue OPTIONS request to the other SIP entities. When SIPC gets an incoming OPTIONS request, it generates a response based on the information in the media.conf file. The response contains the information related to all the media types SIPC can support. The information is described in SDP.

3.2 SIP extensions

SIPC supports a number of SIP extensions such as SUBSCRIBE, NOTIFY, MESSAGE and DO. We describe the detail of the implementations of the extensions in below.

3.2.1 SIP extensions for presence

SUBSCRIBE and NOTIFY are the extended SIP methods used for exchanging presence information. SIPC can work as both subscriber and presentity. The presence information of the presentities are displayed in the address book.

SIPC follows the Internet draft, SIP extensions for presence [3], to do the implementation of SUBSCRIBE and NOTIFY methods. Since the subscriber and the presentity's information is changed so often, we did not save the presence information in the address book (The data stored in the address book are more static), instead, we save the information in the XML files, namely watch.xml, which saves the presentities' information, and subscribe.xml, which saves the subscribers' information.

3.2.2 SIP for instant messaging

SIPC can send and handle MESSAGE requests for instant messaging. The MESSAGE request uses the SIP message content to bring the instant message. The content types that can be handled by SIPC are plain text or HTML.

SIPC uses a separate window for instant messaging. The window is identified by the address of the person which SIPC is talking with instead of being identified by the Call-ID of the session. In this way, if one party accidentally stops the session and sends another MESSAGE again (another session with different Call-ID than before), this message will still be displayed in the same window on the other side.

When SIPC gets a MESSAGE request, it checks the From address to find out whether there is an existing window for the sender. If there is, the message will be displayed in that window. Otherwise, a new window will be created for the conversation.

For a valid MESSAGE request, 200 response will be sent back immediately.

3.2.3 DO

SIPC can send DO commands for device control. There are different kinds of devices, every kind of devices may have different control commands. The control commands for a specific device type are saved in ~/.sipc/devices/¡devicetype¿.conf file. For example, the commands for x10 devices are saved in ~/.sipc/devices/x10.conf file. When a user opens a device for controlling, SIPC will list all the valid control commands. The user can set the arguments for each control

command and send the control command to the SIP device gateway. The control commands are contained in the SIP message content in DMP format. When the response for the DO command comes back, SIPC will display the message content in the response. In the future, a better user interface will be created for showing the status and the information of the device.

4 Implementation

4.1 Programming language

For easier user interface implementation, we choose Tcl/Tk as the programming language. Compared with the other language, Tcl/Tk has the following advantages for developing a SIP user agent:

First, Tcl/Tk is a platform independent language so that a Tcl program can run on different platforms without modifications.

Second, there are many string related commands in Tcl. These commands are very helpful for building a SIP stack.

Third, Tk has provided many widgets such as radio button, text entry. These widgets can save a lot of time on user interface building.

Fourth, Tcl program can be integrated with C/C++ modules easily.

However, there are also some disadvantages in using Tcl/Tk as the developing language.

First, since Tcl/Tk is a script language, to accomplish the same task, the performance of a Tcl program is not as good as the performance of a C/C++ program.

Second, Tcl/Tk is not an object-oriented language. In implementation, we try to use object-oriented style to do the implementation.

Third, to run a Tcl program, the users need to install Tcl/Tk on their computer. It is not convenient. Currently, we are using 'freewrap' to wrap the Tcl/Tk files and our program files into one executable file. The wrapped file contains all the Tcl/Tk libraries, even though many of the Tcl/Tk commands are not used in our program. The size of the executable file is big. The executable file of SIPC is about 4M bytes on Unix and 3.3M bytes on Windows.

Fourth, Tcl/Tk does not have enough network functions for building a SIP user agent. For example, Tcl does not support UDP.

Because it's not worthy to spend a lot of time on building user interface for a research project and the string parsing capabilities in Tcl/Tk is quite helpful for building the SIP stack, we chose to use Tcl to build SIPC.

4.2 GUI

When we were doing user interface design for SIPC, we were trying to follow two rules: Rule one is to make the main interface as simple as possible so that for most users, they don't need to do any configurations. Rule two is to make the main interface as small as possible, because nowadays, there are so many programs running on the user's computer, the screen space becomes precious.

The screen dump of SIPC's main interface is shown in figure 3.

Figure 3: Main interface of SIPC version 1.5

This user interface occupies very small screen space in its initial state. When a user wants to make a call, the user can click on the 'Call' button, and the user interface is changed to what is shown in Figure 4.

Via this interface, the user can input the callee's address and the subject of the call. The user can click on the 'Call' button in the new interface to send out the INVITE. When the user click on the 'Cancel' button, if the INVITE has

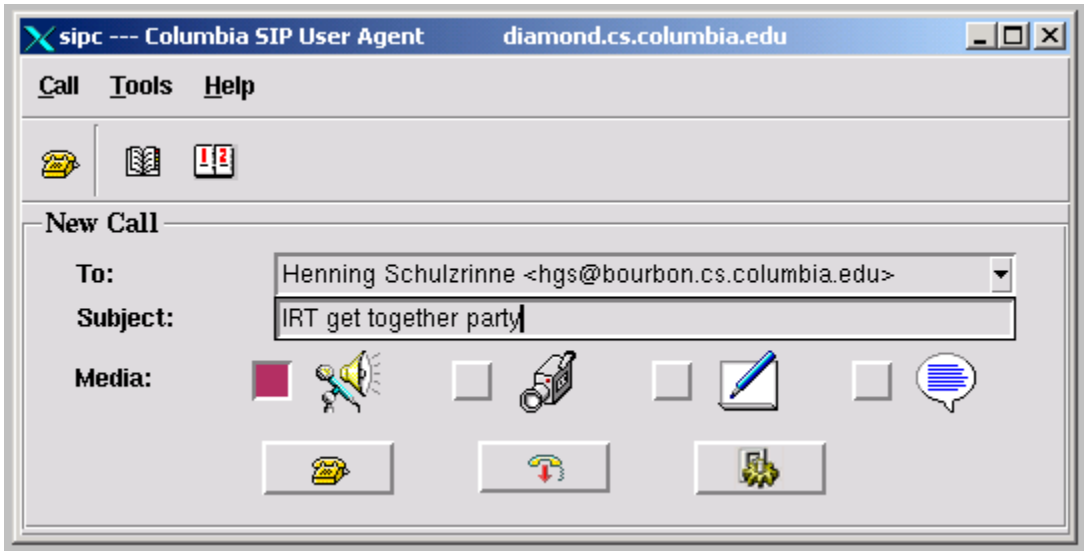

Figure 4: Making a call

been sent out, the CANCEL request will be sent out and the new call frame will be closed. If the INVITE has not been sent out, SIPC will only close the new call frame. without sending any packet out. The user can choose the media for a call. A call can contain multiple media. For further configuration, user can click on the 'Config' button to get the dialog box shown in figure 5.

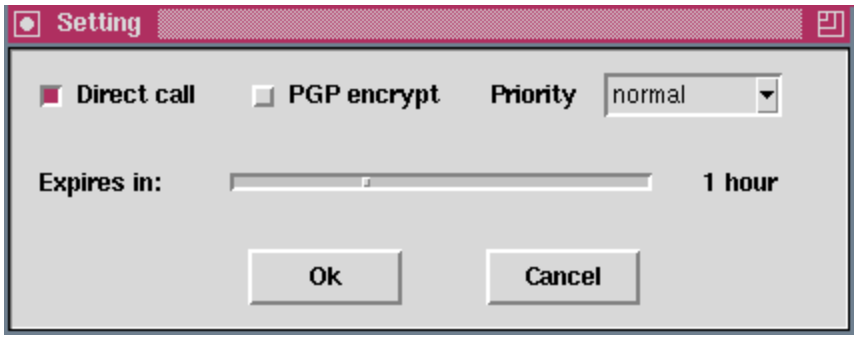

Figure 5: Call configuration

In the dialog box, if the user checks 'Direct call', the request will be sent directly to the URL in the To: input box. Otherwise, the request will be sent to the outbound server configured in the Preference. The user can also check PGP encrypt to enable PGP encryption for the request. The user can set the priority of the call. The priority can be "normal", "emergency", "urgent" and "non-urgent". "Expires in" is used to set the expiration time of the INVITE request.

When the call arrives at the callee side, the callee's SIPC will pop up a incoming call handling dialog as figure 6. The incoming call handling dialog will retrieve the information of the caller from the address book and display the call history of the caller.

In the menu of the main interface, a user can choose 'Tools – monitor' to start the monitor to watch the SIP packets (Figure 7). Besides displaying the content of the packet, the monitor window will display the time that the packet is sent out or received, and the source or destination of the packet.

SIPC has some other functionalities in addition to the basic call setup. We will introduce the GUI of the other parts in their own sections. The interface for registration will be introduced in section 4.7, the interface of instant message

Figure 6: Incoming call

will be introduced in section 4.9, the interface of the address book will be introduced in section 4.10.

4.3 Controller

The controller is the engine of SIPC. It contains the logic for processing the requests and the responses. The controller bases on the information from the service module and the GUI module to set the value of the headers in a SIP message and passes the header list to the SIP stack. For each SIP method, we have a corresponding Tcl file `controller_<method_name>.tcl(method_name can be invite, register, etc.)` to deal with the request and the response of this SIP method. This Tcl file generally contains four functions:

`Controller::<method_name>`
: This function gets the value of the headers from the GUI and the service part, then pass the header list to SIP stack for generating a SIP request.

`Controller::<method_name>Handle`
: This is a call back function for handling an incoming request. During the initialization of the controller, this function will be registered to the SIP stack. For example, `SIPStack::registerCallback "INVITE" Controller::inviteHandle` will register Controller::inviteHandle as the handler for the incoming INVITE request.

`Controller::<method_name>Response`
: This function is used to generate a response for an incoming request. It gets the information from the transaction history, the GUI and the service part, sets the values of the headers and passes the header list to SIP stack for generating the response packet.

`Controller::<method_name>ResponseHandle`
: This function is called by the `Controller::responseHandle` to handle the response of an outgoing request. The `Controller::responseHandle` is registered to SIP stack as the callback function for handling all responses.

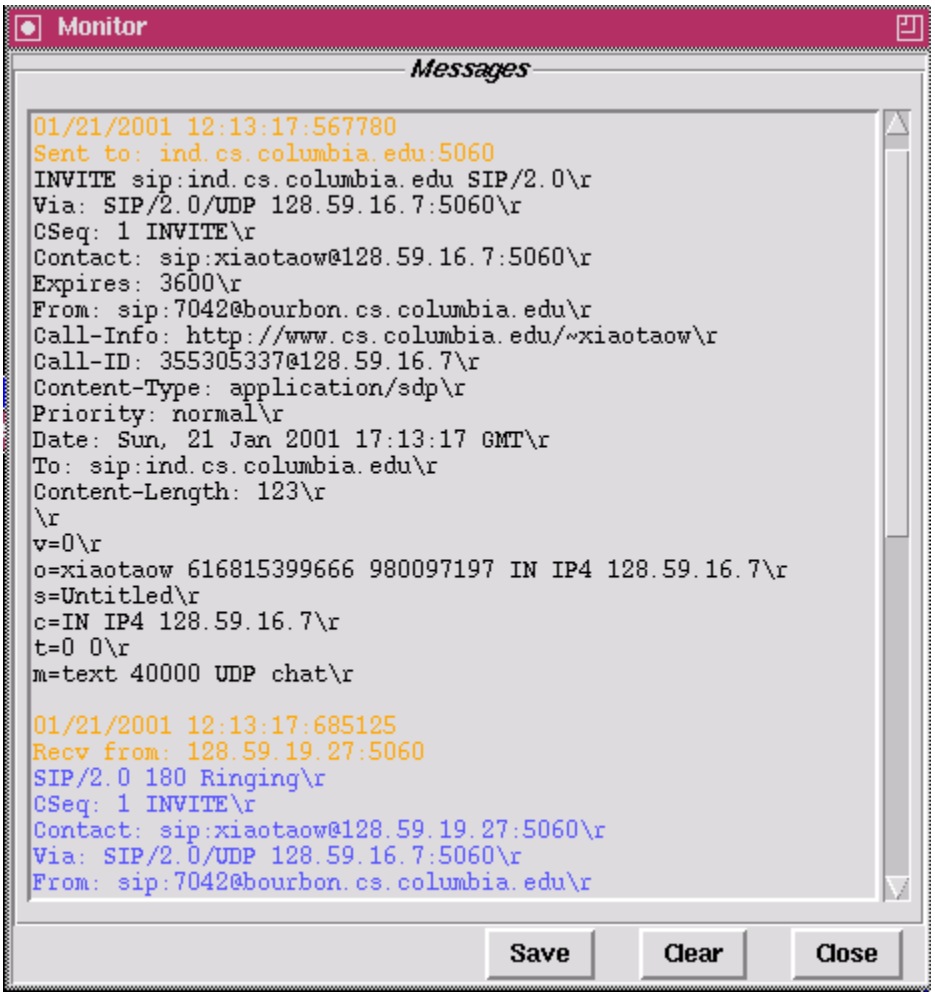

Figure 7: Monitor window

In SIPC, most of the headers' values are set in the controller. However, some headers' values are set in the SIP stack. These headers are common to all the requests and we push the program piece for handling them down to the SIP stack. These headers are:

Call-ID	If not specified in the controller, will be set by the SIP stack
CSeq	If not specified in the controller, will be set by the SIP stack
Via	Set by the SIP stack
Content-Length	Set by the SIP stack
Date	Set by the SIP stack
Route	Handled and set by the SIP stack
Record-Route	Handled and set by the SIP stack

4.4 SIP stack

The SIP stack handles the following tasks:

The SIP stack is responsible for parsing the incoming SIP packet into a structured representation. In SIPC, The structured representation is a list containing the name and the value of the headers and the content of the message. SIP stack then passes the parsed information to the controller.

The SIP stack is also responsible for tracking transaction status and handling the retransmissions. If the incoming

packet is a retransmission, the SIP stack will not pass it to the controller, instead, the SIP stack will handle it based on the transaction history.

There are some SIP headers are common to all the SIP requests and responses. Instead of repeatedly handling these headers in each controller function, in SIPC's design, we put the program piece for handling these headers in the SIP stack. These headers are described in section 4.3.

What the controller passes to the SIP stack is a list of headers. To compose these headers into a valid SIP packet is the task of the SIP stack. After the packet is generated, SIP stack passes the packet to the transmission module for sending out.

4.5 Transmission

Tcl itself does not support UDP. The transmission module provides procedures to support both TCP and UDP transmission in the same interface and provides some utility functions. The UDP extension library is introduced in section 4.6.

The transmission module contains the following functions:

`Transmission::connect {rHost rPort protocol callback}`:
This function will create a new client socket. Since we are not using connection oriented UDP socket, the arguments rHost and rPort are useless for a UDP socket but are required for a TCP socket. This function will return the socket name. The information of the socket is kept in the `Transmission::a_socket` array. If there is a packet arrives at this socket, the *callback* function will be called. The *callback* function only accepts one argument, which is the socket name. All the other information about the socket can be obtained from the Transmission::a_socket array through `Transmission::getProtocol, Transmission::getLocalPort, Transmission::getRemoteHost and Transmission::getRemotePort` function.

`Transmission::server {lPort callback}`:
This function will create a server socket on port *lPort*, return the socket name and keep the information in the `Transmission::a_socket` array. If there is a packet arrives at this socket, the *callback* function will be called.

`Transmission::send {sock rHost rPort packet}`:
This function will send the *packet* through the *sock* to the destination defined by *rHost:rPort*. If the packet monitor is on, the monitor windows will display the content of the packet as well as the time when the packet is sent out and the destination the packet is sent to.

`Transmission::recv {sock numchar blocking\_time}`:
This function is used to receive bytes from the *sock*. It will block until the *numchar* characters has been received or the `blocking_time` is reached. This function will return the bytes received. If *numchar* is equal to 0, this function will be in a non-blocking mode. In non-blocking mode, the function will receive whatever existing in the *sock* and return.

`Transmission::gets {sock blocking\_time}`: This function is used to get bytes from the *sock* until the end of a line `(character '\n')` is met or the `blocking_time` is reached.

`Transmission::acquireUdpPort {rport}`: This function is used to find an available UDP port. *rport* indicates the preferred UDP port the function should check first. If the preferred port is not available, the function tries to find another available port.

`Transmission::acquireTcpPort {rport}`: This function is used to find an available TCP port. *rport* indicates the preferred TCP port the function should check first. If the preferred port is not available, the function tries to find another available port.

4.6 Tcl extensions

The following Tcl extensions are used in sipc:

4.6.1 UDP extension

Tcl does not support UDP, so we implemented a Tcl extension for UDP. There are some existing UDP extensions are available. Such as TclDP, tcludp. However, TclDP has UDP support bundled with other things so that the total package is too big for us. In addition, TclDP does not support Tcl 8.4. The tcludp package does not have event handling support under Windows and it only supports Tcl 7.6. The UDP extension we implemented can send and receive UDP packet under Unix, Linux and Windows. It provides event handling support for these operation systems and supports the latest Tcl version (Tcl 8.4)

4.6.2 GDBM extension

SIPC uses the GDBM extension to provide database support for the address book. GDBM extension consists of two parts, namely GDBM library and GDBM Tcl interface.

GDBM library provides the database functionalities. It can be downloaded from the following URLs:

For Unix/Linux: http://www.gnu.org/software/gdbm/gdbm.html

For Windows: http://jlsysinc.home.netcom.com/download.htm or http://www.cs.columbia.edu/~xiaotaow/sipc/src/wingdbm.zip

GDBM Tcl interface provides Tcl commands for database operations. It can be downloaded from http://www.neosoft.com/tcl/ftparchive/sorted/databases/Tclgdbm/0.6/

4.6.3 Other extensions

In SIPC, the following Tcl extensions are used to provide convinience on both Unix and Windows platform:

Tcl command `util_base64_encode`: This command accepts a string as the input argument and outputs a base64 encoded string.

Tcl command `util_digest_encode`: This command accepts multiple strings that contain the value of nonce, opaque etc. as input arguments and outputs a digest encoded string.

Tcl command `util_gettimeofday`: Since the precision of the Tcl command 'time' is one second, which is not enough for us to monitor the sending and receiving time of a packet, we created this command. The precision of this command is one microsecond.

The following Tcl extensions are only for Windows:

`util_qc_check`: The command is used to check whether the quicknet card is installed or not.

`util_qc_init`: The command is used to initialize the quicknet card.

`util_qc_ring`: The command is used to ring the phone connected to the quicknet card.

`util_qc_hook`: The command is used to get the on-hook/off-hook status of the phone connected to the quicknet card.

`util_qc_DTMF`: The command is used to collect the DTMF digits from the phone connected to the quicknet card.

`util_wininfo`: This command will return a list containing the domain name, the host name, the IP address and the user name of the machine.

`util_ring`: This command can play an audio file as ring tone.

`util_ringstop`: This command is used to stop the audio playing.

The original `Freewrap` cannot handle Tcl command 'load' properly. That means if we use the `Freewrap` to build the executable version of SIPC, the Tcl extensions cannot be loaded. To solve this problem, we compiled all the extensions into the `Freewrap` so that the executable SIPC does not need to do any dynamical loading.

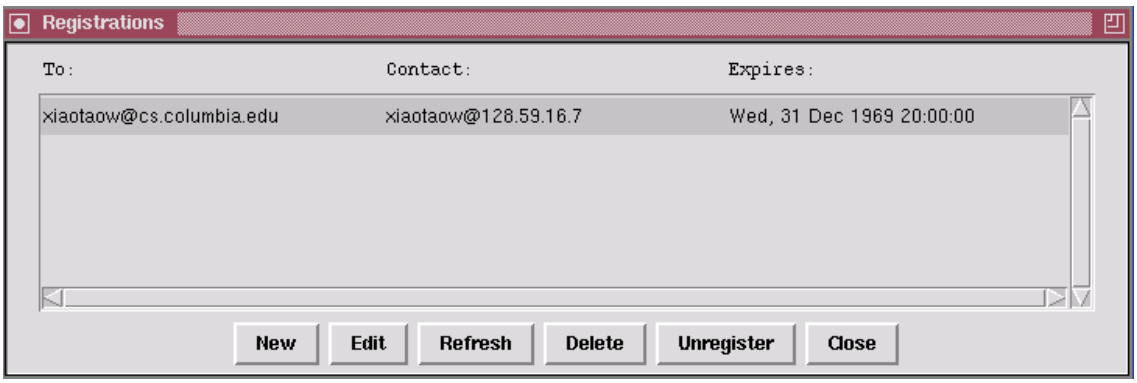

Figure 8: Registration interface

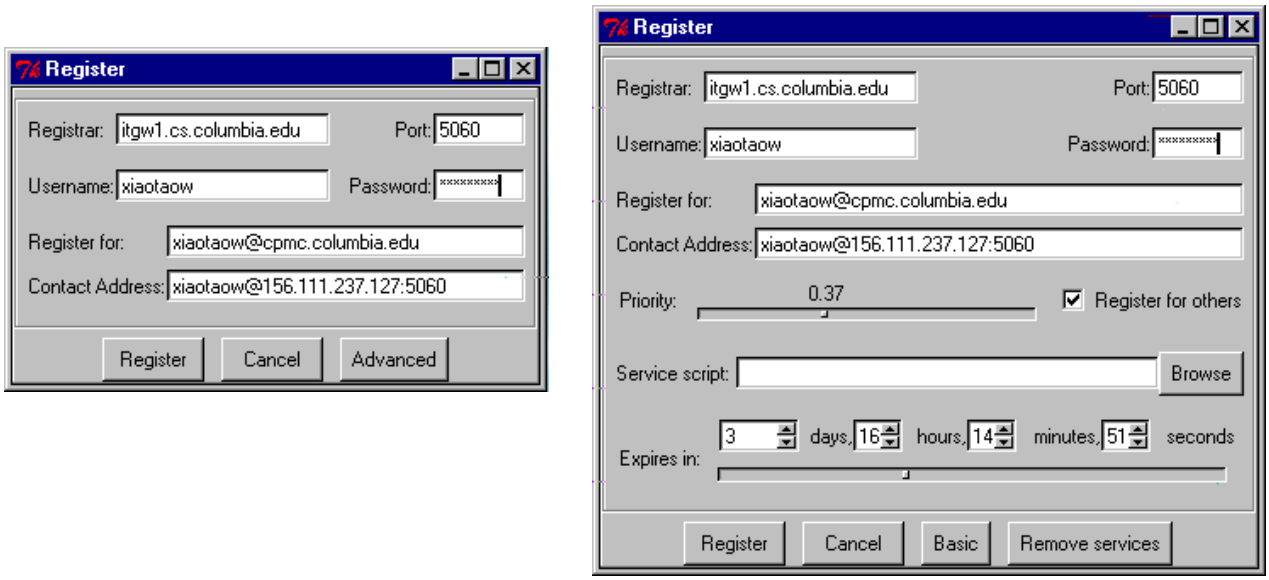

Figure 9: Basic registration interface and advanced registration interface

4.7 Registration user interface

As described in section 3.1, SIPC can send REGISTER requests and handle the response of the REGISTER requests. SIPC saves all the registration information in /.sipc/registry file and displays the information in the registration management window shown as figure 8:

From the registration management window, the user can create new a registration to a SIP server, unregister a contact address from a SIP server, delete a registration locally and refresh the existing registrations. figure 9 shows the interface to create a new registration.

In the new registration page, a user can set the priority of the registration and can upload or remove service scripts to or from the server. When SIPC tries to remove the services from a SIP server, it will set the 'action' parameter in the Content-Disposition header to be 'remove'. SIPC will set the script type parameter in the Content-Disposition header based on the extension of the filename and the content of the file. SIPC can support two script types, namely 'script' and 'sip-cgi'.

4.8 Media transmission

SIPC uses external media applications to do media transmission. In SIPC's Preference configuration, a user can con-

figure the external applications used to handle media data. Currently, the following media types are supported by SIPC:

Audio: By default, SIPC uses RAT (Robust Audio Tools) [7] to handle audio transmission.

Video: By default, SIPC uses VIC (Video Conferencing Tools) [8] to handle video transmission.

Whiteboard: By default, SIPC uses wb [9] on Unix/Linux and uses wbd [10] on Windows to handle white board drawing transmission.

Text: By default, SIPC uses chat (an internal Tk application) for text communication.

Desktop sharing: By default, SIPC uses vnc [11] to share the desktop screen between the caller and the callee. The caller will start `vncserver` and send the address and the port of the `vncserver` to the callee. The callee will start `vncclient` to connect to the caller's `vncserver`. The caller's desktop will be shared to the callee.

4.9 SIP for instant messaging

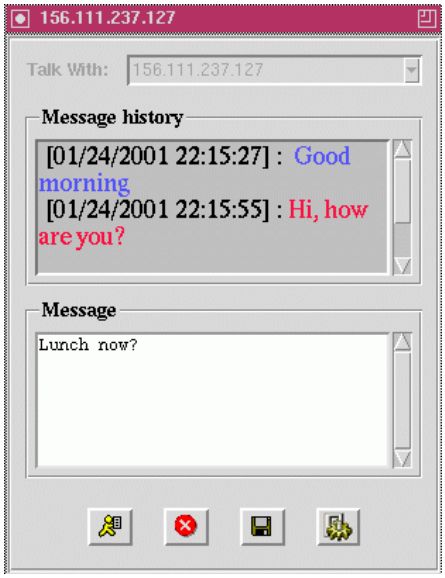

Figure 10: SIP instant messaging

Figure 10 shows the user interface for SIP-based instant messaging.

The bottom text field is used for the user to type the messages and the upper field is used to display the message history. The message history can be saved to a file. A user can input an SIP address in the 'Talk with' entry. Section 3.2 described how to send and handle MESSAGE request.

4.10 Address book

SIPC has a address book keeping the contact addresses as well as some call history information. The address book is also used in the SIP for presence project. Two columns in the address book interface contain the users' presence information. The main interface is shown in figure 11:

The main interface lists the name, the nickname, the organization, the SIP URL and the last call made by the user. In the new interface with presence information, there will be two more columns: subscriber, which indicates whether a user listed in the address book has subscribed to the owner of the address book; and watching, which indicates the status of the user that has been subscribed by the owner of the address book.

SIPC uses GDBM to store address information. The image of the person is saved as base64 encoded string. The address book can import LDIF file or export to LDIF file that is compatible with Netscape's address book.

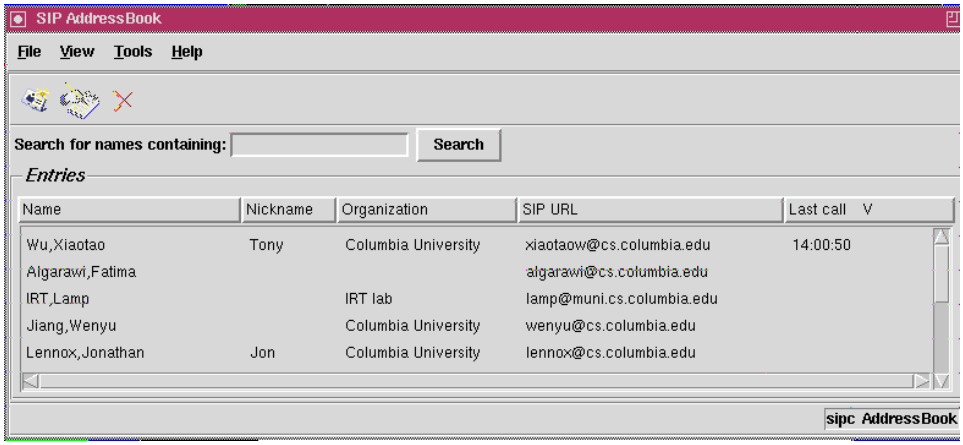

Figure 11: Address Book

The address book keeps the following information of a person: First name, Middle name, Last name, Email, Nickname, Work phone, Home phone, Fax, Pager, Cellular, Photo, Title, Organization, Department, Address, City, State, Zip, Country, Homepage URL, SIP URL and Notes.

5 Further work

The further work can be divided into three parts:

1 The first part is the signaling related work. It includes the following tasks.

For registration process, SIPC should be able to find the SIP proxy server automatically based on the DNS SRV records.

SIPC should be able to do call transfer by using REFER method.

SIPC should be able to support SIP for presence extensions.

SIPC should be able to handle 183 response with SDP as its content (Cisco SIP-PSTN gateway does so.)

SIPC should be able to handle PRACK method, which is a SIP extension used for ensuring the reliability of the provisional responses.

2 The second part is the media related work. In this part, we try to do two things. One is to use MBUS [12] interface to control RAT. The other is to start a web sharing session by using SIPC. In a web sharing session, the caller and the callee can browse the web together.

3 The third part is the service related work. This part is a long term work and need to be carefully designed. The ability to support SIP CGI and CPL will be the first step towards a fully functional service module.

6 Development history

SIPC's implementation began in the summer of 1998 when SIP [13] was an IETF Internet-draft. At the beginning, SIPC was developed as a trivial user agent for testing sipd's (Columbia University SIP proxy, redirect and registrar server) basic and digest authentication. SIPC was implemented in Tcl/Tk 8.0 at that time. As more and more SIP methods and functionalities were added in, in February 1999, SIPC version 1.0 was completed. SIPC version 1.0 could handle basic SIP calls, single user registration with basic and digest authentication, CANCEL and 1xx˜6xx responses. The main interface of SIPC 1.0 is as figure 12

In April 1999, SIPC participated in the 1st SIP bake-off. With many bug fixes to make the software more stable, some major modifications to make the architecture clearer, and some new added functions to provide more functionalities such as invoking Netscape to handle http:// and mailto: URL, SIPC version 1.1 was released by the end of May

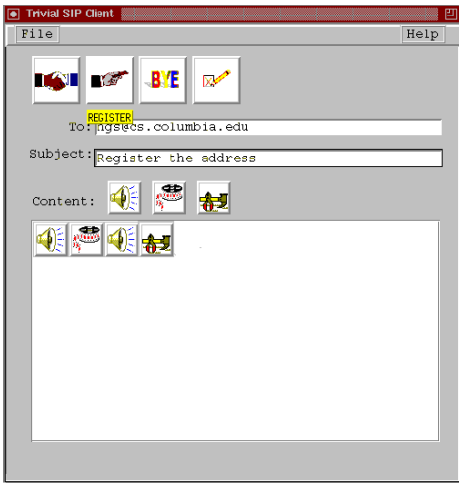

Figure 12: Main interface of SIPC version 1.0

1999. SIPC 1.1 contains all of the basic SIP user agent features, most of the intermediate level features and some of the advanced features. Figure 13 shows the main interface of SIPC 1.1.

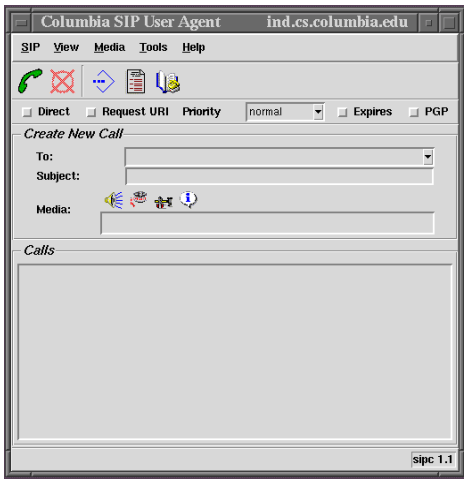

Figure 13: Main interface of SIPC version 1.1

SIPC1.1 has the following problems:

- Windows version was not stable due to the bugs in the UDP library;
  There is no SIPC binary version, user need to install Tcl/Tk before using SIPC;
- The user interface occupies screen space with mostly empty interface areas;
- The architecture of SIPC 1.1 was not extensible and there is no consideration on the service module;
- RAT (The tool used by SIPC for audio transmission) requires RTCP, at that time, many other implementations did not support RTCP so that SIPC was not able to communicate with the other implementations by audio, though the SIP signaling worked well with the other implementations.

SIPC 1.2 did not target to solve these problems, instead, SIPC 1.2 added more functionalities, such as address book, more complicated registrar. SIPC 1.2 was released in December 1999.

From version 1.3, SIPC had a binary version. The UDP library in SIPC was changed from Tcl-DP to the library implemented by ourselves. We use `Freewrap` to build the binary version. `Freewrap` is an open-source software used to wrap the Tcl/Tk files into single executable file. In SIPC version 1.3, we did some modifications on RAT so that RAT did not require RTCP for audio transmission. SIPC 1.3 was released in February 2000.

SIPC 1.4 made some major changes in the user interface. The new interface was more compact. The current user interface follows the user interface designed in SIPC 1.4. In the version 1.4, the UDP library for Windows was totally rewritten and from then on, the Windows version of SIPC became stable. SIPC 1.4 was released in May 2000.

SIPC 1.50 was released in November 2001 with major changes on the architecture and the building process. In this release, the architecture is clearer, the building process is easier and the address book can be used on Unix, Linux and Windows (Previously, it can only be used on Unix). This version contains a Tcl SIP stack that can be used separately in the other projects.

References

[1] P. G. Sumit Khurana and A. Dutta, "Device message protocol (DMP): An XML based format for wide area communication with networked appliances," Internet Draft, Internet Engineering Task Force, May 2001. Work in progress.

[2] J. Rosenberg *et al.*, "SIP extensions for instant messaging," Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

[3] J. Rosenberg *et al.*, "SIP extensions for presence," Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

[4] J. Lennox and H. Schulzrinne, "CPL: a language for user control of internet telephony services," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[5] J. Lennox, J. Rosenberg, and H. Schulzrinne, "Common gateway interface for SIP," Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

[6] M. Arango, A. Dugan, I. Elliott, C. Huitema, and S. Pickett, "Media gateway control protocol (MGCP) version 1.0," Request for Comments 2705, Internet Engineering Task Force, Oct. 1999.

[7] "Robust audio tool." http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/.

[8] "Videoconferencing tool." http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/.

[9] "Whiteboard." http://ee.lbl.gov/wb/.

[10] "Wbd: Whiteboard." http://www-mice.cs.ucl.ac.uk/multimedia/software/wbd/.

[11] "Virtual network computing." http://www.uk.research.att.com/vnc.

[12] "Mbus technology." http://www.mbus.org.

[13] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.