Eth2Phone Project Proposal

Sumit Bhansali, Susheel Daswani, Siva Gaggara, Ashish Shah, Satyam Vaghani
CS 344 Projects in Computer Networks
Stanford University
Fall 2000

Motivation:

To address the shortage of phone lines in the Gates building, we plan to build a telephony gateway between Gates' gigabit ethernet network and the Stanford PSTN. This gateway will allow Gatesians to use the Stanford Telephone Network to place local and long distance calls, and will also allow people outside Gates to call Gatesians at their computers. Another advantage of this system will be its ease of use and extension, with the hope that future generations of Gatesians will improve/extend the system for the posterity of all.

Detailed System Architecture:

Client:

The client will be written in Java to ensure portability between clients. In terms of the front-end look and feel, we'll present the user with a standard telephone interface. On the back-end side, the client will be speaking SIP with the gateway for signaling purposes: it will be used for call initiation/setup/teardown. We will use RTP/RTCP as the transport protocol for communicating voice between the client and the gateway. Optionally, we will also implement a UDP based transport protocol to exchange the digital voice data.
.

Protocols:

SIP (Session Initiation Protocol):
The client will use an existing user agent API written in Java. A Java API will ensure that the user agent will be portable. We will have one SIP user agent on the client and one on the gateway to talk to each other directly. SIP is only used for signaling (call-setup, call-termination, etc.) purposes.

SDP (Session Description Protocol):
SDP is closely tied with SIP for describing sessions. It identifies the underlying transport protocol and its relevant parameters (e.g. media, format, port numbers etc.). Using SIP and SDP, a particular call (i.e. SIP CallID) can be mapped to a particular session (using the end point IP addresses and port numbers). Thus the transport protocol packet need not contain any identification of the call.

RTP/RTCP:
The actual voice-data transfer will be done using RTP/RTCP, which is an end-to-end protocol for transmitting real-time data. The voice is transported using RTP and RTCP is used for control information. For more information on why we chose to use SIP and RTP, please see “Why SIP” and “Why RTP” below:

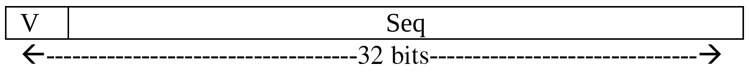
We will base our initial implementation on RTP/RTCP. If time permits, we will also implement a lightweight transport protocol over UDP. This will have less header-overhead than RTP. This protocol is described below:

UDP based lightweight transport protocol

This is a very simple protocol. It has only a version field for future extensions and a 30-bit sequence number. RTP packet includes a 32-bit timestamp, which is useful for deciding the playback time on the receiver. This takes care of the jitter in the replayed voice. Since our protocol is assumed to work in a high-speed, low-delay LAN environment, we can do without this timestamp. Thus, the receiver would play the voice packet as soon as it arrives. The sequence number is used for identifying out-of-order packets and discard them. (No attempt will be made to recover them.)

The header format is given in figure 1.

Header:



V = 2 bit version
Seq = 30 bit sequence number

Figure 1. Header for the UDP based transport protocol.

RTP header for a unicast session is 12 bytes. The extra 8 bytes of RTP header adds 3.2kbps bandwidth overhead assuming a voice packet is sent every 20 ms (a magic number that appears in the RTP RFC and other places many times). (a packet every 20 ms => 50 packets/second => 50*8 bytes/sec = 3.2 kbps).

To integrate our protocol with SIP and SDP, we will have to define a custom transport protocol/media format in SDP. Such an addition is allowed according to SDP RFC.

Modules:
The modules that comprise our client program are the following:

- **The client UI:** This is the interface that the user will be interacting with. It will be a graphical interface that will present the user with a standard telephone interface along with options to accept, reject, and hangup calls.

- **Sound Capture and PlayBack:** This module will capture the sound from a sound input device and send it to the transmitter/receiver (Tx/Rx) module described below. It will also receive sound data from the Tx/Rx module and play it back on the sound output device. We plan to use Java Streams to interface with the Tx/Rx module and the Java Sound API to interface with the sound input-output devices.

- **Transmitter/Receiver module:** This module will do the call-setup, call-termination and call-handling using SIP. It will encapsulate the digital voice data in RTP packets in the format and send it over the network to the gateway. It will also decapsulate the received RTP packets to get the voice data to send to the Sound Capture and PlayBack module.

Gateway:
The gateway will be written in C and C++ to reduce execution latency on the server side. We will be using an existing SIP C++ library, provided by Columbia University, to talk SIP with the client.

The Gateway consists of the following modules:

Server-side SIP User Agent
The server-side SIP user agent will interface with the client-side SIP user agent to initiate/setup/teardown calls. SDP will describe the session.

RTP/RTCP module
After a call has been setup using SIP, this component will handle the real-time transfer of the voice data. We will optionally have the UDP based transport protocol module.

The above two components exist on both the client and the gateway.

Switching/Glue Process
This piece of software will route incoming and outgoing calls and maintain relevant state/session information. Equally important is its function as the glue, which will wrap up various components of the gateway, including the SIP user agent and the dialogic card interface.

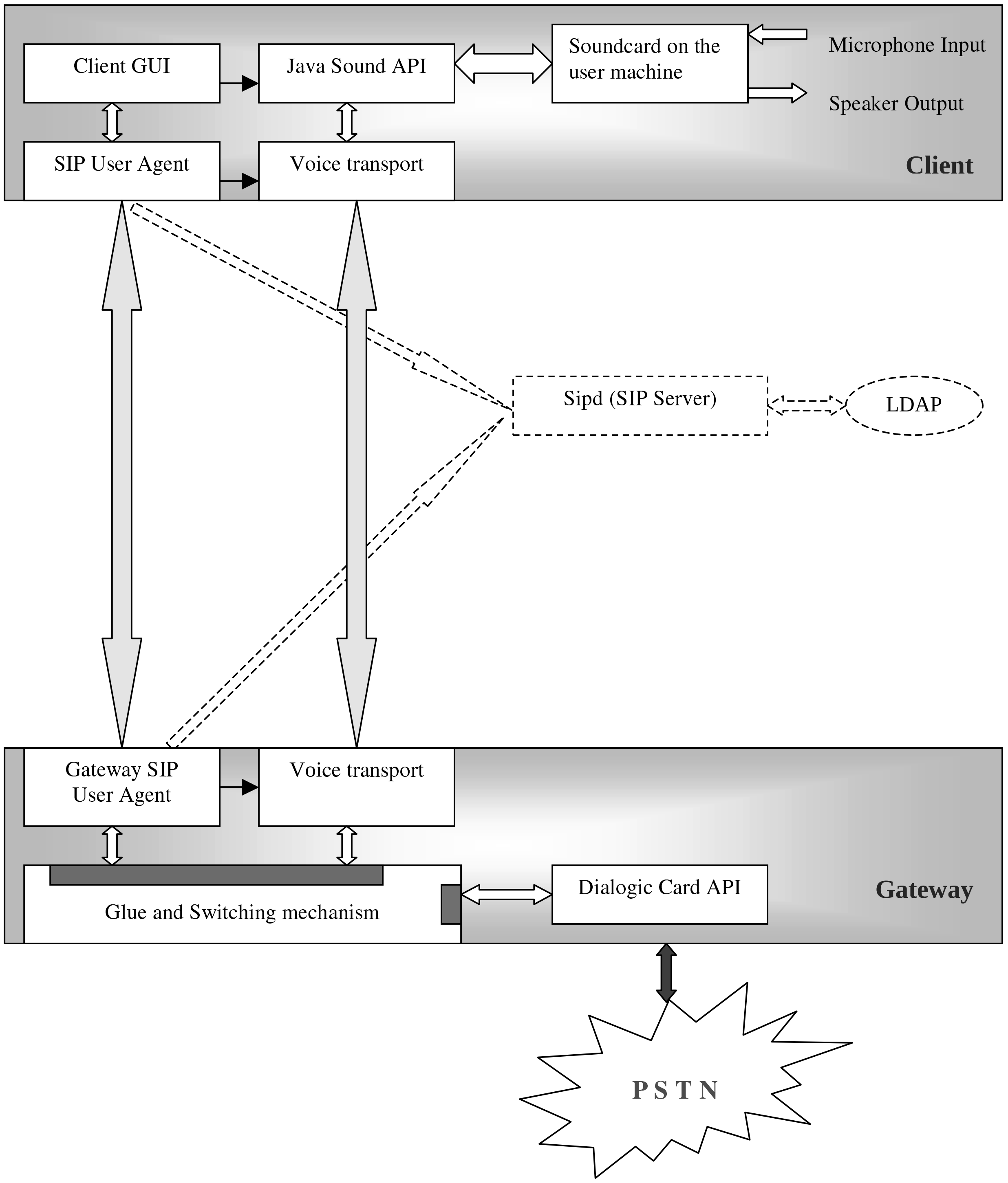


Figure 2: Eth2Phone System Architecture

This module is designed to limit the dependencies between the SIP user agent and the hardware interface (presently, the dialogic card). This additional level of indirection serves two purposes - it draws clean boundaries between different modules and it makes the system extensible with respect to support for different hardware interfaces to route calls to the PSTN, varied voice transport mechanisms (eg. Voice over UDP, RTP, etc).

Looking at a more granular level, the indirection provided by the glue component can be resolved to 3 layers (as shown in the figure 2). The two outermost layers provide an interface between the external component and the inner (middle) layer. In this scenario, replacing a component, say, the hardware interface to the PSTN, will require changing one of the outermost layers for the system to work. However, the product of the number of layers and the system extensibility achieved is a figure of merit.

Dialogic Card Interface:
This piece of software bridges the Gateway Server to the Stanford PSTN. It transfers our digital voice signals to analog signals and vice-versa. It also generates the DTMF tones corresponding to the dialed digits. Call-setup, call-termination services are provided by the card. After the call is setup, the card can dial additional digits to provide complete touch-tone functionality. There will be a timeout provision (in terms of number of rings) if the called party does not pick up the phone.

This module will provide a hardware-independent API to the switching process. However, the implementation will be dependent on the hardware-API.

Deliverables:

- A Java Client which will provide a Telephony UI to Gates Users and will interface to the Ethernet-to-PSTN Gateway.
- The Ethernet-to-PSTN Gateway that will glue together the Gates Ethernet network and the Stanford PSTN. Initially, only outgoing calls will be supported. However, time permitting, we plan to support incoming calls also (see Future Work below)

Future Work:

1. Supporting incoming calls will be the first extension to the project. To support incoming calls, we will have to map the phone number extensions to user’s current location (i.e. IP address). As the first step, we will go with a static directory service on the Gateway. Then, if time permits, we will integrate the SIP server from Columbia University (sipd) and LDAP into our system to provide the directory service.

2. At the moment, our architecture does not include a SIP Server (only SIP user agents, which are capable of basic SIP functionality). A SIP Server provides extra utilities such as serving as a proxy server, a redirect server, and directory services (in association with LDAP). These features are not necessary for our system's core functionality, so we don't see the immediate need to include it. As more features are added to the Eth2Phone gateway, it may be necessary to include this piece. We must build the system in such a way that integration of a SIP server is very straightforward.

3. Supporting the UDP based transport protocol as described in the document earlier.

Tasks and assignments:

Documentation: ALL
API Creation: ALL

Sumit Bhansali: Dialogic Card component of the Gateway Server
Susheel Daswani: Management and Front-End of Client
Siva Gaggara: SIP User Agent and voice transport issues in the Client
Ashish Shah: SIP User Agent and voice transport issues in the Gateway Server
Satyam Vaghani: Switching/Glue Process in the Gateway Server

Most Difficult Issues:

Largely, this project is an integration issue. Plugging pieces together so that they are inter-operable is the main issue involved. Moreover, documenting this process and building a clean and extensible system is another difficult and important issue.

Why SIP?

Our team chose SIP after looking at both SIP and H.323 to handle call management. H.323 comes from the Telephony world (ITU-T) whereas SIP comes from the IP world (IETF). While H.323 is more mature and morewidely deployed/supported, SIP is being promoted aggressively by the IETF. It is a lightweight protocol; It is more flexible and it does not try to build all the cases into it, unlike H.323. Some of the pointers on the debate of SIP vs. H.323 are:
http://www.fokus.gmd.de/research/cc/glone/projects/ipt/sip.html
http://www.cs.columbia.edu/~kns10/talks/von_new/sld001.htm
http://www.nuera.com/news/iptelephony_072000.cfm

Note that the tools available for both protocols are equally good. There are also other protocols also in the game, namely MeGaCo and MGCP. After talking to a few people (at Columbia University and at Cisco), as well as going through some of the above documentation, we decided to go with SIP.

Some resources on SIP:
http://www.sipcenter.com
http://http://www.cs.columbia.edu/sip/

To learn more about the Ethernet-2-Phone gateway, please visit:
http://www.stanford.edu/~smd/eth2phone.html.

Why RTP?

We decided to use RTP as our voice-transport protocol due to the following two reasons: First, many of the existing implementations of SIP user agents already have support for RTP; so using RTP will ensure higher interoperability. Second, RTP does not incur unreasonable overhead. As described in earlier in the document, this overhead is of the order of 3.2 kbps. This is negligible on a Gbps LAN

Some resources on RTP:
http://www.cs.columbia.edu/~hgs/rtp/