

# Prototyping SIP-based VoIP Services in Java.

*Hua Zou (\*), Hongman Wang (\*), Wenxin Mao (\*), Bai Wang (\*),  
Stephane Focant (+), Koen Handekyn (+), Dominique Chantrain (+) and Nick Marly (+)*

(\* National Key Laboratory of Switching Technology & Telecommunication Networks,  
Beijing University of Posts & Telecommunications, Beijing, China.

(+) Alcatel Corporate Research Center, Antwerp, Belgium.

(\* wangbai@bupt.edu.cn, (+) nick.marly@alcatel.be

## Abstract

Next to the binary-based H.323 signaling protocol for IP Telephony developed by the Telecommunication World (ITU-T) [1], a more lightweight text-based protocol was deemed necessary to address the specific needs of the Internet World. The Session Initiation Protocol (SIP) [2] was introduced in the IETF Conference Control working group and has meanwhile gained maturity and support.

This paper presents a Java-based implementation of the SIP protocol stack, based on a layered SW architecture, reusable object oriented software components and Application Programming Interfaces (API). This protocol stack has been used as a basis to realize the main network components necessary for SIP based call setup: user agent, proxy server, redirect server, and registration server. The interaction between these components is illustrated through a number of service scenarios, focusing on a PC-to-PC communication within the context of a Virtual Private Network (VPN) service.

## 1. Introduction

The integration of fixed voice (PSTN), mobile, broadcasting (cable) and data (Internet) networks induces a broad spectrum of opportunities for value added applications. In the past networks were designed with one specific service in mind, nowadays one network has to support multiple services. It looks like IP-based networks will be the key infrastructure

for these full-service networks.

However, deploying carrier class telephony services (quality of service or QoS, availability, and reliability) on networks originally designed for data transport is not straightforward. Traditional IP-networks don't support QoS [3][4], nor real-time signaling [1][2] and media transport [5]. Both issues are currently being addressed in the IETF and once solved, Internet Telephony is likely to replace, or at least complement, the existing switched telephony networks. In any case, the look, feel, and hear should be the same.

SIP, together with the Session Description Protocol (SDP) for media description [6], is a valuable candidate for call signaling, in analogy with the traditional signaling for circuit switched telephony networks. SIP is a lightweight, text-based signaling protocol, client-server oriented, with two types of messages (Requests and Responses). The network architecture contains user agents, stateless & statefull proxies, redirect servers, and registration servers.

The implementation of the SIP protocol stack described in this paper is based on the Java 2 platform and follows a layered approach with clearly defined interfaces and functionality of each layer. This approach assures the reusability and flexible evolution of the developed software components. Specific functions include the communication handling, retransmission, message parsing, retransmission and expiration scheduling, status handling and the Finite State Machine (FSM).

## 2. Brief Overview of SIP

SIP is a stateless session and call initiation, invitation and control protocol able to establish, modify and terminate multimedia sessions or calls. The client and server exchange messages (Requests: Invite, Ack, Bye, Cancel, Options, Info and Register; and Responses: final and provisional). It is text-based with about 36 headers and supports TCP and UDP (with reliability support through a retransmission protocol) connections.

The message body can contain a media description based on SDP or any information for which the MIME-type is specified in one of the headers. The Info request is used to carry session related messages such as ISUP and ISDN signaling or DTMF digits.

A SIP based Internet Telephony network architecture contains user agents, proxies, redirect servers and registration servers. Interworking with PSTN is established by means of a SIP-URL for telephone subscribers, signaling gateways, media gateways (MG), and MG controllers. Location of proxies able of routing the call signaling is established with a protocol like the Service Location Protocol (SLP) [8]. For simplicity, the prefix 'SIP' will be omitted in the rest of the paper (e.g. proxy means SIP proxy).

## 3. The Prototype

### 3.1. The Network Architecture

The SIP protocol stack implementation described further on, has been used to build a prototype in which a Voice over IP service is introduced as value-added service on top of a basic platform for Remote Access Service (RAS) to VPNs. This context implies that SIP functionality is available in the domain of the access provider. The business & technical motivations for this approach are explained in [7]. A typical architecture is shown in Fig. 1.

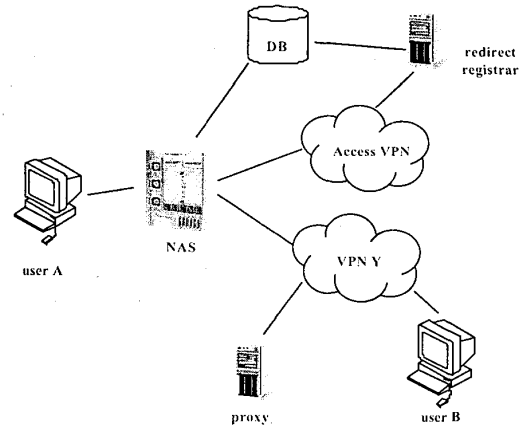


Fig. 1 - SIP enabled Access Network.

The Network Access Server (NAS) operates as a traffic concentrator and IP-router or bridge, allows service selection (VPN), and assigns an IP-address to the user.

Typically, when a user gets online, he will first register or check his SIP contact-URLs with the registration server. This data is stored in a database DB and will be used by a redirect server to find out the current user IP-address. From the moment the user starts a session, the NAS will update the DB with the user's current IP-address. Each time the user connects to a different VPN (and gets a new IP-address) or disconnects, the DB will be updated with the new presence information.

Typically each VPN will contain a proxy server. The database, registrar and redirect server on the other hand, is typically located in the domain of the Access Provider (or Access VPN). The proxy will forward each request for call setup (incoming call for a certain user) to the redirect server, which will retrieve the current user IP-address as contact information. From that moment on the proxy can forward the request to the user. If the user is not connected to the VPN from where the call originates, he will be requested to connect to this VPN, and will receive a new IP-address. After this IP-switch, the proxy can again forward the request and the call can be set up.

### 3.2 The Software Architecture

To provide a structured model for the protocol stack

with clearly defined interfaces, a layered architecture was developed (Fig. 2). Between each layer synchronized message queues are defined and each component in a layer runs in its own thread. Multiple components can be started (each with their own thread) allowing load balancing according to the amount of messages that have to be processed.

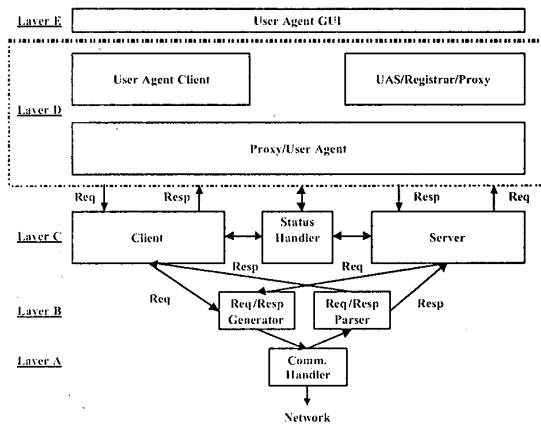


Fig. 2 - SIP Protocol Stack Software Architecture.

Layer A contains the communication component capable of UDP, TCP and Multicast transmissions. It was designed using the singleton principle, meaning that there is only one communication handler providing an interface to the higher layers. Multiple connections can be set up, and messages contain meta-information concerning originating IP-address and port, protocol, etc.

To offer reliability in case UDP is used, a retransmission scheme was defined in [2]. Layer B contains the necessary timing components to perform the retransmission of outgoing messages. It also provides parsing of the incoming messages. From this point on (layer B-C interface) there is a difference between requests and responses and the individual information of the fields in a SIP message is available.

Layer C takes care of scheduling the retransmission and the preferred retransmission scheme (constant interval, or exponential interval spacing) is based on the type of message and on the status of the call. The

two main components are the client (sending requests and receiving responses) and the server (receiving requests and sending response). This layer will also maintain status information and process requests and responses that don't require user intervention (e.g. sending a '100 Trying' response on receipt of an incoming request, or processing a retransmitted message).

The Finite State Machine (FSM) is implemented in layer D. Again the singleton principle is used to manage the client and server FSM. Filling in the necessary fields in the message, setting up a call and providing an interface towards the user is taken care of in this layer. Typically a proxy and user agent contain a client and server, where a redirect and registration server only contain a server.

Finally the graphical user interface (GUI) is provided in layer E (only for a user agent). Here the settings can be adjusted (proxy, contact URL, registration server, contact information) and registration, call setup and call answering can be performed.

The user client application has been plugged in to a 'Soft Terminal,' allowing remote management of applications on the user terminal. The Soft Terminal is based on the Java Management Extensions (JMX) and incorporates a Remote Method Invocation (RMI) registry. Also the interface between the different nodes and the DB is based on RMI, allowing easy software upgrades as long as the class interface isn't changed.

#### 4. Prototype Implementation

Based on the network and software architectures described in the previous section, a complete prototype has been built in Java. This prototype includes logical components such as user agent, proxy server, redirect server, registration server and even a simple location server. Together, these components can be used to construct an entire environment for SIP-based Voice over IP Services. In order to simplify the implementation while retaining the basic functionality, 25 SIP headers (out of 36) are supported in the

prototype, and only UDP is supported as the transport protocol.

The prototype is implemented in JDK 1.2.2 Standard Edition platform. The whole implementation consists of about 100 classes. The size of all Java files is about 550k, approximately 20 thousand lines of source code including comments. The size of the bytecode is about 310k.

This experience demonstrates that Java is well suited to implement complicated applications, such as a SIP server. With the rapid development of Java itself and computer technology, the performance of Java is not a big issue anymore. In fact, the Java platform is continuously being extended with new classes and APIs that are very useful for various application developments. Particularly its "implement once, run any where" policy makes it more attractive in the Internet environment. In our prototype for example, we can easily change the hardware environments among Win 95, Win NT, UNIX and even others without modifying any code.

However, as a relative new programming language, Java still has some defects that should be taken care of. For example, currently Java does not support ICMP messages, which are used to signal network problems such as congestion. Consequently, the SIP prototype cannot take into account these situations, although this function is required by SIP. Furthermore a Java Virtual Machine doesn't detect when the IP address of a PC changes, which typically occurs in a VPN context, when the user switches from VPN A to VPN B e.g. to accept an incoming call from VPN B.

The layered SW architecture described earlier has proven to be very adequate. In this prototype, the SIP stack is the kernel, the essential part of all logical components except for the location server. After grouping each function to the proper layer, we can easily reuse components, because the reusing element is based on layers rather than classes. Furthermore, by reducing the dependence between adjoining layers, the implementation mechanism of each layer is relative

independent so that if one layer is being modified, other layers do not need to change at all.

Much attention should be paid to threads. Each layer has several threads, and messages between layers are processed synchronously. Another possibility would have been to start on thread per message. It has been found that the number of threads in each layer, the synchronization mechanism among these threads, and the priority of each thread are all very important factors that affect the performance directly and heavily. Thus these factors should get much attention in the design phase. During the debugging and testing phases they must be tuned very carefully.

The current prototype still offers some room for improvement. For example, reliability could be further improved as follows. In layer D, each call leg (transaction) can be associated with one thread. For example, in the user agent, a thread is only responsible for one call leg, and in a proxy/redirect/registrar, a thread is mapped to one transaction. In this way, if one call leg (transaction) failed, it would not influence the others.

To improve performance, every call (call leg/transaction) should have its own call data block rather than using a shared data block, which needs a very strict synchronization mechanism to allow a number of threads to access it concurrently. The call data block can reduce the requirement of synchronization so as to improve performance.

Another interesting add-on considers the interface to the location service. Because the implementation for the location service is not specified by SIP, it can in fact be anything. For example an independent LDAP server may provide it. Thus, as an extension, it may be interesting to introduce an LDAP interface in the SIP stack.

## **5. Value Added Services and Applications**

To demonstrate the protocol stack and components in an application, a scenario was designed showing the establishment of a PC-to-PC call in a multiple VPN

context (Fig. 2). User A (callee) contacts a NAS, and chooses to connect to VPN X (e.g. corporate LAN of his company) [7]. Meanwhile, user B (caller), who is connected to a different VPN (Y), wants to establish a SIP call to user A. Therefore he sends an 'Invite' request, which will be routed via a proxy in VPN Y and the redirect server in the Access VPN. Before user A is able to accept an incoming phone call from a different VPN, he has to switch to the same VPN as the caller. Therefore the NAS, on command of the redirect server, will ask user A to connect to VPN Y.

Once user A and user B are connected to the same VPN, the redirect server will send the correct contact address (with the current IP-address) to the proxy and the call can be further established (forwarding the 'Invite' and accepting the call with a '200 OK').

## 6. Future Work

Work on incorporating media (based on the Java Media Framework) and media description (SDP) are ongoing. The introduction of open APIs (currently discussed in the upcoming standard Java API from JAIN) on the different elements, will offer a fully open interface for introducing value-added services.

Introduction of scripting mechanisms (such as the Call Processing Language or CPL [9]) will allow Service Providers to develop customized value-added services. New opportunities in the field of Instant Messaging, for which SIP seems ideally suited due to its real-time character, will be explored.

## 7. Conclusions

This paper has described an object oriented and extensible design for a SIP protocol stack, with clearly defined interfaces, that enables the development of advanced prototypes. The prototype implementation has illustrated the value of the chosen design approach, and has revealed some critical issues typically associated with matching on the one hand software requirements (e.g. extensibility) with on the other hand

communications requirements (e.g. real-time performance). In addition it offers a basis for future prototyping of a wide range of value-added services and applications, as well as experiments with new service technologies such as scripting.

## References

- [1] ITU-T Standard, "H.323: Packet-based Multimedia Communication Systems," February 1998.
- [2] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg: "SIP: Session Initiation Protocol", IETF-RFC 2543, March 1999.
- [3] M. Gibson and J. Crowcroft, "Use of SIP for the Reservation of QoS guaranteed Paths," IETF-Draft draft-gibson-sip-qos-resv-00.txt, October 1999.
- [4] H. Sinnreich, S. Donovan, D. Rawlins and S. Thomas, "Interdomain IP Communications with QoS, Authorization and Usage Reporting," IETF-Draft draft-sinnreich-interdomain-sip-qos-osp-00.txt, October 1999.
- [5] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RTP-RTCP: Real-time Transport (Control) Protocol," IETF-RFC 1889, January 1996.
- [6] M. Handley and V. Jacobson, "SDP: Session Description Protocol," IETF-RFC 2327, April 1998.
- [7] Nick Marly, Dominique Chantrain and Jurgen Hofkens, "Exploiting Similarities Between SIP and RAS: The Role Of The RAS Provider In Internet Telephony," accepted for publication at ISS 2000.
- [8] E. Guttman, C. Perkins, J. Veizades and M. Day, "SLP: Service Location Protocol v2," IETF-RFC 2608, June 1999.
- [9] J. Lennox and H. Schulzrinne, "CPL: Call Processing Language," IETF-Draft <draft-ietf-iptel-cpl-00.txt>, February 1999.