

SPLAT: A unified SIP services platform for VoIP applications^{||}

Aameek Singh^{1,*†}, Arup Acharya^{2,‡}, Priya Mahadevan^{3,§} and Zon-Yin Shae^{2,¶}

¹ College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA, U.S.A.

² Network Server System Software, IBM Watson Research Center, 19 Skyline Drive, Hawthorne, NY, U.S.A.

³ Computer Science, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA, U.S.A.

SUMMARY

The steady improvements in the networking infrastructure and ever increasing broadband penetration has fueled the resurgence of voice-over-IP (VoIP). One of the important contributors to this growth has also been the development and wide acceptance of the session initiation protocol (SIP). However, the current usage of SIP requires a per-application deployment (each application using its own SIP stack). In addition to shared port number problems, this leads to a narrow development of SIP based services, even though SIP, as a protocol, offers incredible opportunities for enabling various applications simultaneously.

In this paper, we propose SPLAT—a unified SIP platform, consisting of a client-side SIP service and supporting network infrastructure blocks, that provide unified mechanisms to execute generic SIP functions through an exported higher level API. Applications can leverage the API and ready-made building blocks for creating richer interfaces without significant and often repeated development effort, e.g. a conferencing server coupled with a gaming server can provide context-aware audio conferencing between occupants of a particular game room. Importantly, the SPLAT framework is available to *all* applications including the ones not inherently based on SIP and thus presents a great opportunity for enhancing such applications. The SIP service API is designed to be *extensible* and provides novel higher level functional primitives like *ad hoc* conferencing and seamless transition of sessions. In addition, it also exports a low level interface for specialized applications that need direct access to SIP call flows. Another feature of the service is that it allows a user to plug-in an end device (softphone, IP phone, PSTN phone) of his/her choice on a *per-session* basis. We demonstrate the richness of the API by describing prototypes for enhancing various applications as well as new converged applications. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: SIP; VoIP; SIP services; SIP API

*Correspondence to: A. Singh, College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332, U.S.A.

†E-mail: aameek@cc.gatech.edu

‡E-mail: arup@us.ibm.com

§E-mail: pmahadevan@cs.ucsd.edu

¶E-mail: zshae@us.ibm.com

^{||}An early version of the paper appeared in ICCCN 2004.

1. INTRODUCTION

Voice-over-IP (VoIP) refers to the technology that enables sending voice data over the IP network. The traditional public switched telephone network (PSTN) has a different infrastructure backbone than the IP-based internet and is only used to transmit audio data. On the other hand, VoIP enables transmission of audio on the regular internet network which till recently, has primarily been used to transmit data. This integration presents great opportunities for both service providers and consumers. While VoIP has various advantages like lower costs, greater consumer control and location flexibility, its typical problems have been quality of service (QoS) and standardization. However recently, VoIP has been able to win over both large corporations and consumer homes. Many VoIP products like Vonage [1], Skype [2] are becoming extremely popular and changing the telecommunication landscape as we know it. A number of factors are contributing to make VoIP a success. While corporations view it as a cost-cutting measure, leveraging their existing IP networks, the increasing broadband penetration has made it a valuable alternative to common households. We believe another contributor to this growth to be the development and wide acceptance of the session initiation protocol (SIP).

SIP [3] is a popular choice for establishing media sessions and instant messaging [4, 5]. It is also now the most popular and widely accepted protocol for VoIP. There are several IP softphones [6, 7] and hardphones [8] available in the market today that are SIP capable and are being used for VoIP. In addition to point-to-point calls, SIP is also used for multi-party conference calls [9, 10]. Some of the recent more innovative uses of SIP have been in multiplayer gaming [11, 12], and collaborative applications [13, 14].

Typically, each SIP application such as an IM client or a softphone rolls out *its own* implementation of SIP, based on an API like JAIN [15], which provides a low-level API for invoking SIP call flows. While this works when only one application is used (e.g. internet telephony), it leads to problems (due to shared port numbers^{**}) if running multiple applications, all using SIP. More importantly, there is no unified platform that can be used by non-SIP based applications to use the media management functionality of SIP. We believe this to be a very narrow view of SIP, a protocol that is much more generic and can be leveraged in many different ways. As we describe later, a platform like SPLAT can be used to enable various non-SIP applications like web browsers, multiplayer gaming, for richer interactive experiences using VoIP.

Overall, we view SIP as a new control pipe to the client desktop beyond just IM and VoIP, and offer SIP as a generic *system service* that is available to all applications. In this paper, we describe the design and implementation of a SIP platform, SPLAT, which provides a generic API in the form of a set of higher-level SIP primitives like point-to-point calls, multi-party conferencing, event notification, etc. SPLAT is designed in a manner that makes it easy to plug-in an end device of user's choice, thereby offering the opportunity to select on a *per-session* basis, one of multiple softphones, an IP hardphone or a regular PSTN phone. We demonstrate the richness of this API by enhancing existing applications such as native SIP click-to-call in web browsers and enabling *ad hoc* on-the-fly conferencing in a non-SIP aware messaging client, as well as describing new converged applications such as web browsing with out-of-band control information passed via the SIP service, as well as dynamic multi-conferencing support for multiplayer network games. This client service is supported on the network side by prototyping

^{**}Many SIP based VoIP hardware are unable to communicate on non-standard ports.

a number of building blocks such as a conferencing server with event notification, support and the ability to create conferences on-the-fly, web sites that use applets to control client SIP service and a gaming server that maps changing gaming contexts to one of multiple conferences. A commercially available packet-audio mixer was integrated into the network for media support.

The rest of the paper is organized as follows. We describe the basics of SIP based VoIP in Section 2. In Section 3, we describe the broad architecture of the SPLAT platform. We describe SPLAT's client-side service in Section 4. Section 5 details the network infrastructure blocks that are leveraged by the client side service. The exported SIP API is described in Section 6. We describe a number of prototype applications using SPLAT as a platform in Section 7. In Section 8, we discuss the advantages of the SPLAT architecture and related issues of security and privacy. The related work is discussed in Section 9. We finally conclude in Section 10 with a note of future course of work.

2. SIP AND VOIP

SIP is an HTTP like protocol, which distinguishes between the process of a session establishment and the actual session. The session between two user agents (UAs) is established using SIP signalling mechanisms which involve sending an INVITE, an OK response and an ACK to the response [3]. These messages contain user parameters (using session description protocol—SDP) for choosing appropriate IP address and port which will be used for actual data transmission as part of the session. This path for data is typically called *media path*, though any kind of data (even other than multimedia) can be transmitted. The IP/port combination can be for any networked device like an IP phone, game console or a PC. Typically, the media data is sent using RTP and signalling is accomplished using either TCP or UDP. The ability of a UA to accept certain encoding mechanisms is also negotiated through SDP as part of the signalling messages (this session negotiation is discussed in more detail in Section 2.1). Any of these parameters can be changed using the RE-INVITE message, which is identical to the INVITE message except that it can occur within an existing session. The session is terminated by using a BYE and an OK message. In addition, SIP allows UAs to refer a UA to another by using the REFER message. This instructs the UA to establish a session with the referred UA.

The UAs are identified by SIP URLs, which is a unique HTTP-like URL of the form *user@host*, for example, *sip:aameek@gatech.edu*. The mapping of the SIP URL to the appropriate physical UA device is done using intermediate SIP proxies, location and redirect servers, which form an overlay network. All UAs REGISTER with a SIP registrar server (can be at the SIP proxy itself), which maintains the address of the UA device. Then, all requests for the SIP URL are routed to the appropriate device for that particular UA. An extension of SIP also supports SUBSCRIBE/NOTIFY mechanisms, in which UAs subscribe to certain events at another UA and can be notified whenever that event occurs.

We refer the reader to Reference [3] for more details about SIP. The overall architecture of such an environment is shown in Figure 1. The overall process of a call setup and call teardown is shown in Figure 2.

2.1. Session negotiation

As mentioned earlier, SIP uses the SDP [16] to negotiate various session parameters between end-points. SDP is used to convey information about media streams in multimedia sessions to

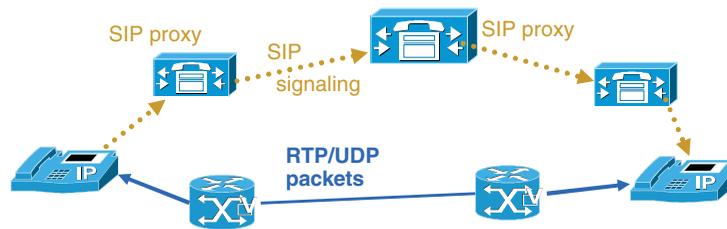


Figure 1. SIP based VoIP Architecture.

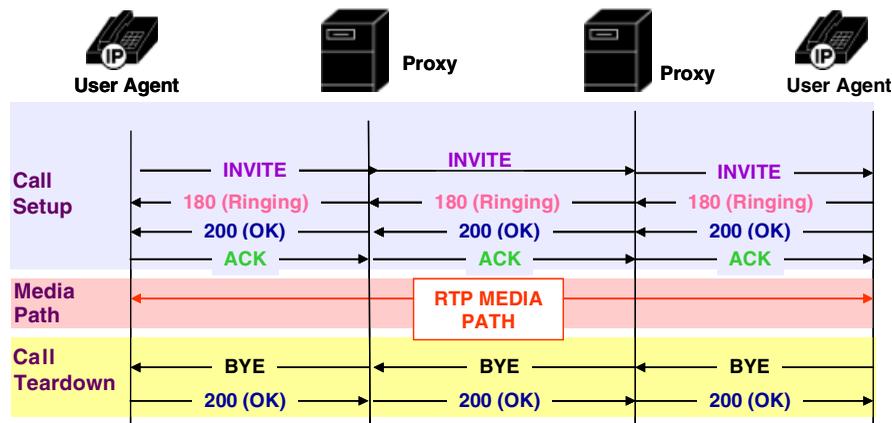


Figure 2. Call setup and teardown.

allow the recipients of a session description to participate in the session. It includes type of media (video, audio etc), the transport protocol (RTP/UDP/IP, H.320, etc), the format of the media (H.261 video, MPEG video, etc), remote address for media and the transport port. The SDP looks like^{††} [16]:

```

Session description
v = (protocol version)
o = (owner/creator and session identifier).
s = (session name)
c = †† (connection information)
t = (time the session is active)
m = (media name and transport address)
a = †† (zero or more media attribute lines)

```

^{††} Indicates optional parameters. Please refer to Reference [16] for all possible parameters.

2.2. SIP publish-subscribe

Another important characteristic of SIP is its light weight event notification mechanism using the SUBSCRIBE and NOTIFY messages [17]. Using these messages, a SIP UA can subscribe to certain events (defined within an *event package*) at another SIP UA and can be notified for the occurrence of these events. SIP allows for subscription expiration, subscription refreshing and unsubscribing for events as well. This inherent support for event notification in the protocol makes SIP an attractive mechanism for a variety of tasks. For example, ‘presence’ information used in IM, where a UA can subscribe to the status of another UA to be notified whenever it changes. Also, since pub-sub is a popular paradigm for multiplayer gaming, SIP is also well suited for such environments [11, 12].

3. SPLAT: ARCHITECTURE

The aim of a SIP platform like SPLAT is to provide a higher level API that can be used by applications for performing various SIP tasks like establishing a session, starting a conference, terminating a session, transferring a session, etc. With such a platform in place, various non-SIP applications can also be enhanced to provide richer VoIP interfaces. As a simple example, through SPLAT it is possible for a user’s home page to contain a SIP URL, clicking which any user browsing the page is connected to him/her via a VoIP call. (See Section 7 for a detailed prototype architecture).

Overall, SPLAT consists of two important components, as shown in Figure 3:

1. *Client-side SIP service*: This provides a SIP interface to all applications. Internally it consists of the extensible SIP API which exports all SIP functions to the applications, wrappers for audio/video end devices, a session arbitrator which is responsible for maintaining information about various sessions and a SIP thin client with the SIP stack which provides low-level SIP functions and used for providing seamless transitions between various sessions.
2. *Network infrastructure blocks*: This component is a group of essential SIP network elements required for a rich feature set. It includes a conference server, used to establish conference sessions, a ‘game’ server, used to maintain multiple concurrent states (audio conferences for clients sharing some context—see Section 5) and a SIP-aware web server.

Next, we describe the two components in detail.

4. CLIENT-SIDE SIP SERVICE

Our SIP service acts as a client side system service running on a particular port. It offers an API to applications at a higher functional level than the one offered by existing SIP APIs such as JAIN SIP [15]. However, in order to meet the demands of specialized applications that require a lower level control, we also provide a *tunnel* through our API providing direct access to SIP call flows. Importantly our API is targeted at the operating systems level to ensure its availability to all applications independent of the application execution environment such as a JVM. There are two modes (see Figure 4) in which applications can invoke this API: (a) by directly sending messages to the specified port of the service, and (b) using SIP as a protocol in the operating

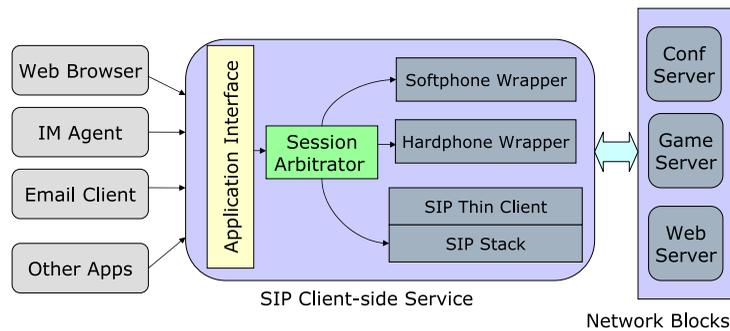


Figure 3. SPLAT architecture.

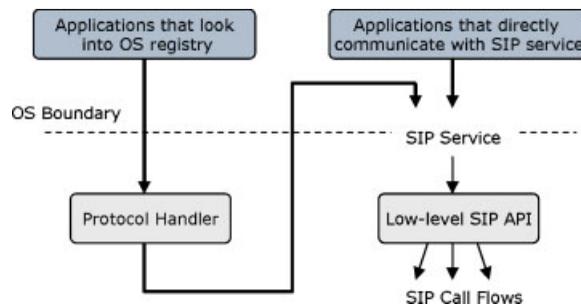


Figure 4. Client SIP service.

systems (by registering a protocol handler for Windows based systems, for example). The second mode can also be provided in Linux based operating systems either by encoding appropriate plugins or inserting modules within the OS itself. It allows any existing or new application, that looks into the OS to determine registered protocols, to automatically be able to handle SIP URLs by invoking the associated protocol handler (similar to *mailto* or *http*).

An important requirement of our SIP service is that the supporting API be extensible. Thus, the current set of functions which will be described later, is not by any means complete, but rather to illustrate an initial set that we found useful across multiple application scenarios.

A second motivation for a client-side SIP service is to offer all control functions that SIP has to offer, within a single service, instead of separate application bundling in specific subsets of full SIP functionality. For example, an IM client may incorporate publish—subscribe mechanisms of SIP, but may not allow an audio call to be setup, while a softphone may incorporate call control functionality but not necessarily support presence and IM functionality. In addition, when two such applications are executed concurrently, there is often a problem with sharing common port numbers (such as port 5060). More importantly though, this leads to narrowly focused SIP applications. Our motivation for an application-independent SIP service is to enable new applications that combine multiple control features of SIP in interesting ways. SPLAT uses a single SIP service that can take actions (based on API calls) on behalf of the

applications. The SIP ‘dialogs’ are routed to the appropriate application using dialog identifiers (as part of the SIP protocol).

Lastly, a requirement of our proposed SIP service is the ability to offer a choice of end-device (SIP UA) for user interaction. A user may choose a device with features that is best suited for the type of session, e.g. a cell-phone may have a built-in camera, or the desktop phone may offer good speakerphone support. This requirement has several advantages: firstly, it frees the SIP service from providing media I/O capability, which is best offered by specialized devices, while retaining the control of such devices from the SIP service. In terms of realizing this requirement, it implies that the SIP service be designed to allow integrating end-devices in different ways. In addition, when an external device is used, the user may still like to retain control of the session (rather than offloading both media and control to the device). This is especially true, when a user wants to utilize special SIP functions like SUBSCRIBE/NOTIFY, which a non-SIP device cannot provide.

The SIP service also allows users to switch devices in the midst of a session. This requirement places a burden on the design of SIP Service in that it should allow easy integration of other SIP devices and also PSTN devices. This is achieved through device-specific wrappers, which are responsible for translating such higher level functional demands to lower level device commands, either through SIP or non-SIP methods such as HTTP POST [6].

5. NETWORK INFRASTRUCTURE

Before we describe the detailed API, we first explain the building blocks used to facilitate SIP network services.

5.1. Conference server

The first of these is a conference control server, which when coupled with a commercially available SIP-controllable media mixer, provides a network service for setting up conferences and mixing audio streams. For every participant, the mixer combines the voice signals of every other participant into a single signal. There are various off-the-shelf SIP-enabled mixers available such as [18]. The conference server uses SIP signalling with the UA and the mixer to establish media paths between the two (Figure 5).

The unique properties of our conference control server are the ability to create *ad hoc* on-the-fly conferences^{**}, and also to support event notification services for events related to conferencing.

To establish an *ad hoc* conference, a user generates a unique ID (e.g. a username appended by a random number), creates a conference URL of the form ‘sip: <ID>@ <conf-server address>’ and sends an INVITE. To route such messages which have not been registered at the SIP proxy (as done for normal SIP routing), we set up the SIP proxy to forward any unregistered SIP URLs to the machine in the domain field of the URL (the conference server in our case). On receiving the INVITE, the conference server (CS) creates a new conference if no such conference id exists; else the user is added to an existing conference as a participant. Event notification is

^{**} Such conferences do not reserve resources in advance and are initiated on-the-fly, thus not registering conference SIP URL at SIP proxies. Routing messages for the conference is the main challenge.

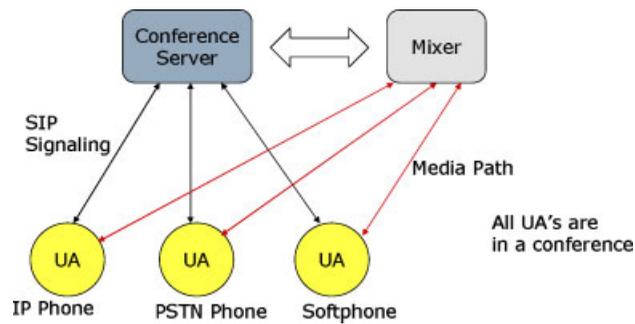


Figure 5. Conference server.

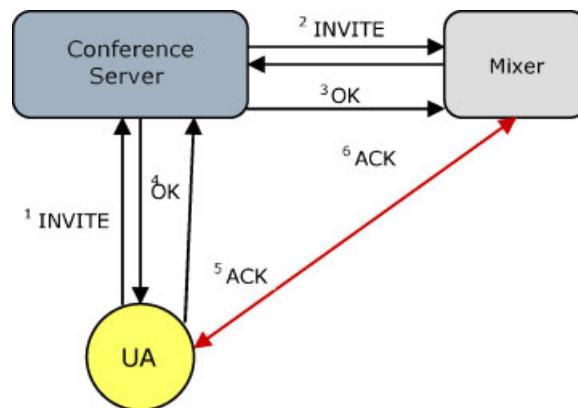


Figure 6. SIP workflow for a conference join.

achieved using the SUBSCRIBE/NOTIFY messaging feature of SIP by implementing a new event package for conferencing contexts.

The workflow for a conference join is shown in Figure 6. ¹INVITE contains UA SDP. The CS gets media path information from that SDP and sends it as the media info in ²INVITE. The mixer sends its SDP in ³OK. The CS extracts that media info and sends it as part of SDP in ⁴OK. After the ACKs media path is established between the UA and the mixer.

It is also possible for the CS to invite a UA into a conference. For example, the owner of the conference could instruct the CS to invite multiple users to that running instance of the conference. In this case, the CS would initiate an INVITE message to the UA's and conference them. Such an initiation is called a *dial-out*. The workflow for a dial-out is shown in Figure 7. ¹INVITE contains *no* SDP. The UA sends ²OK with its SDP. CS gets media path info from that SDP and sends it as media info in ³INVITE. The mixer sends its SDP in ⁴OK. The CS extracts that media info and sends it as part of SDP in ⁵ACK. Also ACKs the Mixer. After the ACKs media path is established between the UA and the mixer.

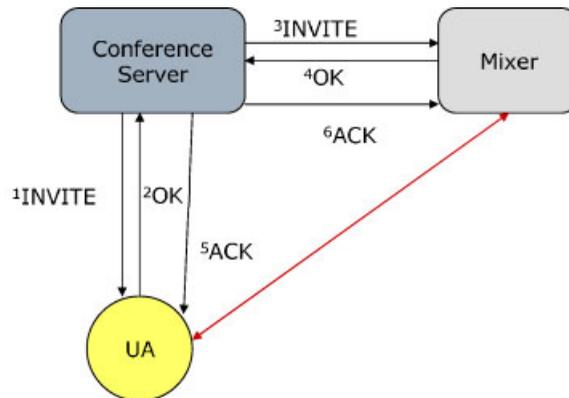


Figure 7. SIP workflow for a dial-out.

5.2. Gaming server

A second application building block that we introduced is modelled as a primitive gaming server which uses the conferencing server to enhance the multiplayer gaming experience. The purpose of this building block is to show that SIP allows multiple services in the network to be composed in interesting ways, rather than to demonstrate gaming *per se*. The gaming server should be viewed as representing an application server with multiple concurrent states, such that application clients are each *dynamically* associated with one of the server states. This basic abstraction is augmented by associating clients sharing a common state at the application server with a common conference, demonstrating that the conferencing service is useful not just as a standalone service but perhaps more so, when combined with another network service. This service composition can be achieved either by coupling the game and conference servers, or by coupling the game client with the SIP service API at the client-side. Both approaches are feasible.

In a corporate enterprise setting, the ‘game service’ is offered as an add-on service to conferencing that allows employees to participate in multiple simultaneous conferences, presented visually to the user as a set of boxes: dragging the mouse to a specified box seamlessly switches the employee’s current active conference without any perceptible break in audio (i.e. without requiring the employee to hang-up and dial in to the new conference). The feature of the gaming service that we wish to highlight is the ability to seamlessly and automatically switch the associated conference when a game client changes its gaming context (such as a dungeon). To perform such seamless and dynamic switches, we create appropriate API function primitives (discussed in Section 6). The overall architecture is shown in Figure 8. We demonstrate the application of this building block in Section 7.

5.3. SIP-aware web server

The final building block we introduce is a web-site that is cognizant of the SIP service API at the client. It can use embedded applets in its pages to instruct users to join particular conferences associated with those web pages. For example, a discussion forum web site, using this mechanism will instruct the viewers of a particular forum to be in a single audio conference and

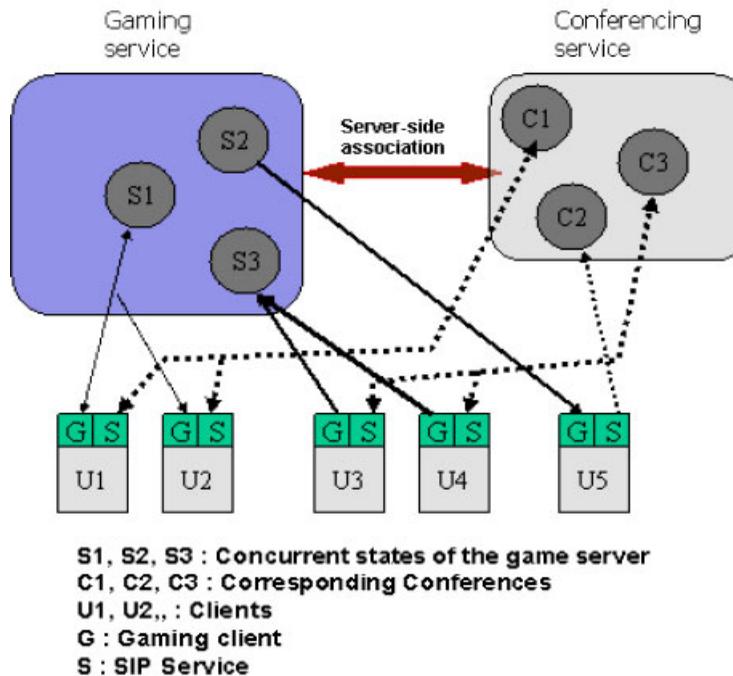


Figure 8. Gaming and conferencing composition.

hence exchange their views via voice. In addition, to keep users aware of other participants, SIP event notification is implemented using the SUBSCRIBE/NOTIFY features. For example, when a new user joins a particular forum, the existing participants are notified of the new participant and displayed accordingly by the SIP service.

The idea is to create greater collaborative environments by utilizing the SIP service at the clients. Embedding a signed applet in a web page allows a web server to co-ordinate client audio sessions and create meaningful groupings. The applet writes SIP API commands (as discussed in Section 6) to the SIP service ports and users can join conferences based on those web pages. Note that since we use the client-side SIP service, the user still gets to enjoy the call control features provided by the SIP service. Moving to a different web page will seamlessly transition the user into a new conference based on that web page. It is important to notice the distinction of control paths between the web server and the gaming application server examples. While in the gaming server the call control was done via the server itself (based on client *gaming* actions), in the web server the call control is through the client-side SIP service. This building block helps create interesting applications like community web browsing, demonstrated in Section 7.

6. API

Applications communicate with the SIP service using XML messages, which encode SIP service API calls. The use of XML allows standardized mechanisms of interaction with the SIP service

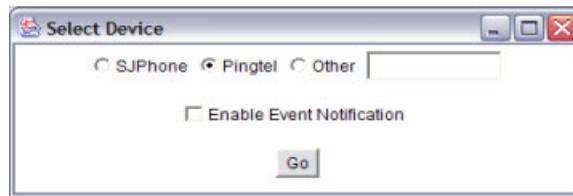


Figure 9. External join device selection.

and also provides extensibility to the API. New functions can be easily added by using appropriate XML messaging tags. Below, we define an initial set of API calls:

- **ExternalJoin:** This command is used to call a particular party when no end-user device is selected. The format for such a message is:

```
<ExternalJoin id=SIP URL/>
```

This indicates that a call is to be made to a SIP URL such as sip:arup@research.ibm.com. The action associated with this function is that the SIP service pops up a dialog box (Figure 9) asking the user to select from one of a set of end-user devices such as softphone, IP phones etc. Once a device has been selected, the SIP service uses the appropriate wrapper for that device to make the call. The wrappers were small Java implementations of primitive SIP functions for the device.

- **Join:** This command is used to call a particular party by using a particular device. The format is:

```
<Join id=SIP URL1>
```

```
<Use dev = dev-id/>
```

```
</Join>
```

This indicates that a call is to be made to destination URL1 using a end-user device pointed to by dev-id. The dev-id is either the local softphone identifier, which indicates the SIP service to launch the appropriate softphone, or the SIP URL for an external hardphone device (appropriately SIP-gatewayed if required). Softphones are especially distinguished in this manner since we can easily launch them using their specific wrappers. However, it is also possible to use a similar hardphone mechanism using the SIP URL for the softphone as the chosen device. In cases when a SIP URL is used, the SIP service needs to establish a control path with the appropriate devices and set up a connection. This can be achieved in two modes, referring to the SIP service involvement in the process.

1. *Transfer mode:* In this case, the SIP service establishes a session between the end-device (identified by Dev-ID) and the called party URL1. The call is completely transferred to the device using a SIP REFER. In this scenario, the SIP service has no further control on the call and the media transfer takes place between endpoints identified by the two URLs (Figure 10).
2. *Loop mode:* The SIP service acts as a Back2Back User Agent [19] between the two endpoints. The media path is still end-to-end between the two URLs; however, the SIP service stays in the control path between the two end-points (Figure 11). The SIP client initiates two dialogs (with URL1 and Dev-ID), however uses the media path information of one while sending the SDP for the other, for the media path to be set

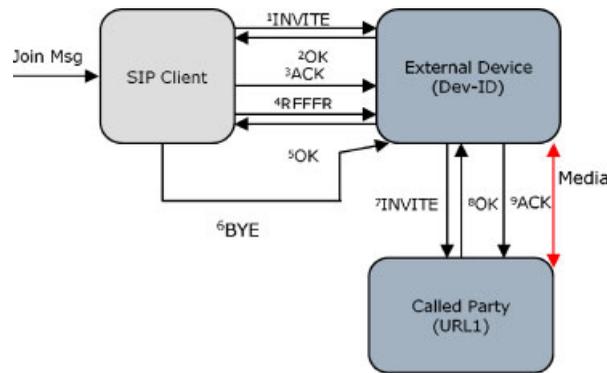


Figure 10. Transfer mode.

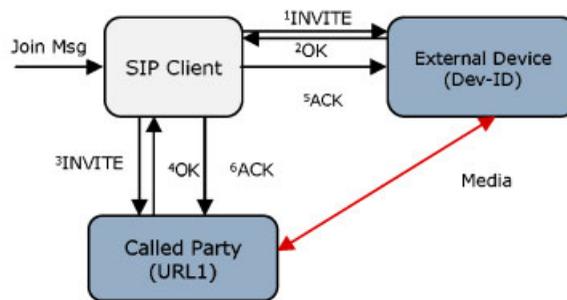


Figure 11. Loop mode.

between the two. Concretely, in Figure 11, ¹INVITE does not contain any SDP. The SIP Client sends the media path information of Dev-ID (received as part of SDP in ²OK) to URL1 in the ³INVITE message. Then it forwards URL1's media path (received with ⁴OK message) to Dev-ID with the ⁵ACK message. This mode is useful for the SIP service to receive event notifications. For example, in a conference call, a user may subscribe to join/leave events (using SIP SUBSCRIBE) and be notified of other participants joining/leaving the conference. Staying in the loop allows the SIP client to display any such notifications. This would not be possible using the transfer mode since the device represented by URL2 may either be incapable of handling the events or exposing them to the user in an appropriate way and then capturing user response to those events. The loop mode is also useful in the context of the next API call described below.

- **SameDeviceJoin:** SameDeviceJoin allows a user to seamlessly switch the called endpoint (e.g. conference) without changing the end-device currently in use. The format for a SameDeviceJoin message is:
<SameDeviceJoin id= SIP URL/>

This functionality cannot be realized by dropping the entire call and setting up a new call from the same end-device to the new target, since this would mean hanging up the external

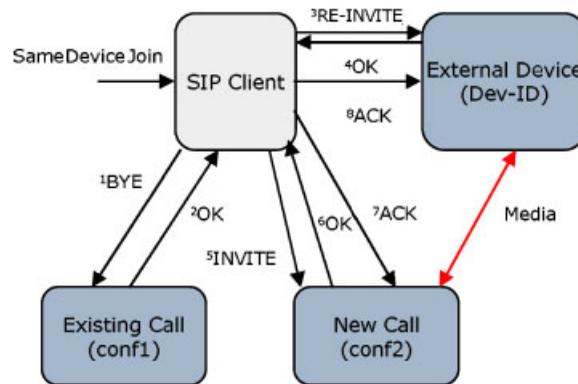


Figure 12. SIP workflow for SameDeviceJoin.

device and picking it up again, i.e. the switch would be perceptible. We need to provide a seamless switch, i.e. never terminate the session with the external device. We use the loop mode SIP service, i.e. the SIP service is on the control path between device Dev-ID and say, conf1 created by an earlier Join command in loop mode. The steps involved in realizing this function is to drop the leg to the current called party (conf1), setup a new call to the endpoint referred to by the URL in the SameDeviceJoin message (maybe a new conference, conf2), and then exchange the IP addresses and port numbers of the two media endpoints (Figure 12).

- **Multi-Invite:** This API call is specific to conferencing, and instructs the SIP service to invite specified additional participants to the current conference. In case the invoking user is not in a conference, a new *ad hoc* conference is created and desired participants are invited to that conference. The format of the message is as follows:

```
<Invite>
<Add id=SIP URL1 />
<Add id=SIP URL2 />
.....
</Invite>
```

The workflow for a multi-invite is shown in Figure 13. After the SIP client sets up the conference at the CS, it instructs the CS to invite multiple users into the same conference. This is achieved by the dial-out process described earlier.

- **Tunnel:** This API call provides a tunneled access to low level SIP workflow commands (specific SIP messages). The motivation for providing such an access is to allow specialized applications to take greater control of the SIP call flows. This mechanism is provided through an add-on plugin implementing the interface providing access to low level SIP APIs. Using the tunneling feature would require developers to write code (as opposed to exchanging XML messages), however, considering the target of specialized applications, this should not be a major hindrance.

As mentioned earlier, the generic nature of the API is of critical importance. The API has to be able to cater to the need of a wide variety of applications in order to be adopted successfully. We believe the two important characteristics of the SPLAT API: (a) extensibility (ability to enhance the API) and (b) the ‘tunnel’ API mechanism (providing greater control to the

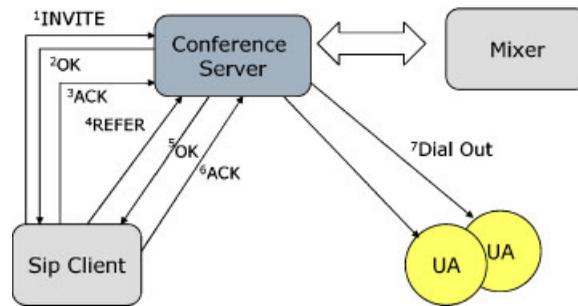


Figure 13. SIP workflow for multi-invite.

developers) make it a very attractive. Indeed, as we demonstrate in Section 7, the SPLAT client API can be used with a number of traditional desktop applications like web browsers and instant messaging clients. In addition, a number of innovative collaborative applications such as context-aware gaming [20] can also be supported.

7. PROTOTYPE APPLICATIONS

In this section, we describe some of our prototype applications that combine the SIP service API and building blocks in the network infrastructure.

7.1. SIP URLs in web browser

After registering SIP as a protocol and defining a protocol handler in the Windows registry, browser programs that refer to the registry invoke the SIP protocol handler when an user clicks on a SIP URL embedded in a web page (Figure 14). The protocol handler initiates an ExternalJoin SIP Service API call, which asks user input for device selection and then sets up a session to the URL. Note that the browser code does not need to be modified.

7.2. Enabling a non-SIP aware IM client

We selected an IM client and server system [21] that uses a proprietary non-SIP protocol, modified the client code to recognize SIP URLs within message bodies, highlighted the SIP URLs as *clickable* links (Figure 15), and on user click, invoked the Join SIP Service API to create an *ad hoc* conference via the conferencing service. We followed certain naming conventions to identify a URL as a conference URL (e.g. the host name in the URL is conf.ibm.com), and hence, the user would need to supply just the conference name (e.g. abc in the example). The key points to note: (a) the functionality of the IM system was enhanced without changing the application's native client-server protocol; (b) the messaging capability of the application is used to inform other participants of a conference (SIP URL); and (c) this highlights how the SIP client and network services are useful for a class of applications whose client code may be amenable to modification but not the server code.



Figure 14. SIP URLs in web browser.

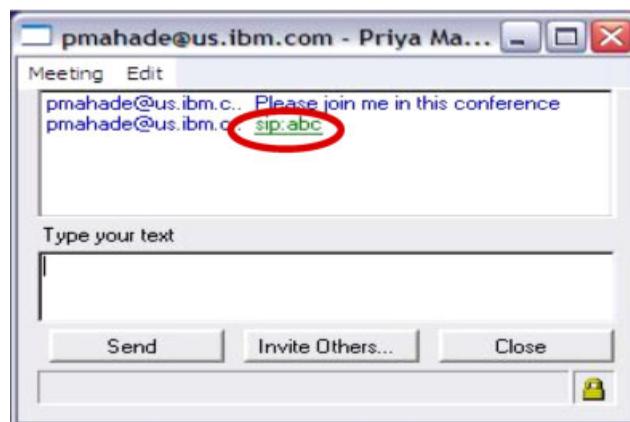


Figure 15. Enabling IM client.

7.3. Multi-conferencing/gaming with seamless conferencing

The ‘game’ consists of four quadrants and a user can move in any of the four directions. When a user crosses a quadrant boundary, he is seamlessly conferenced in with users in the new quadrant using the SameDeviceJoin API call. In the screenshot shown, arup and edie are able to hear each other, while aameek is not able to hear either. If arup were to move into the top-left quadrant, then arup would be added to the conference associated with that quadrant, and arup and aameek will hear each other (if aameek continues to remain in that quadrant) (Figure 16).

The workflows for the game join (initiation into the game) and the conference switch (moving into a different quadrant) are shown in Figures 17 and 18. The game join is a simple SIP session initiation with the game server acting as a back-to-back UA. The quadrant switch, as mentioned earlier, is primarily the SameDeviceJoin message initiated by the game server.



Figure 16. Multi-conferencing/gaming.

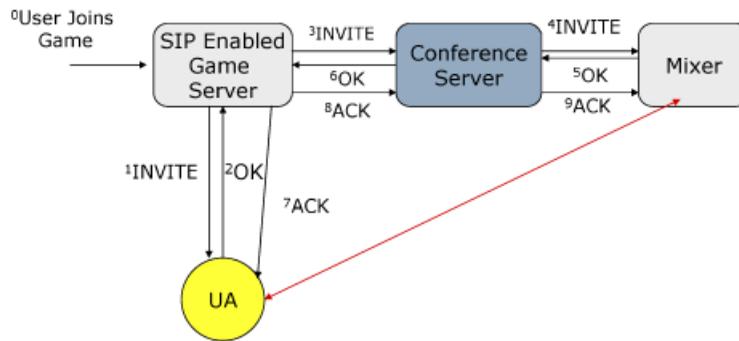


Figure 17. Game join.

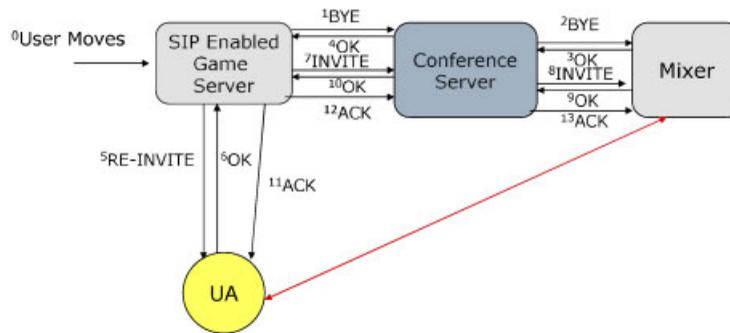


Figure 18. Quadrant switch.

7.4. Community web browsing

This application comprises of a group of web pages, such that viewers of a web page are informed of all concurrent viewers of that web page using SIP's event notification mechanism as well as being conferenced together. In addition, whenever a user moves to a different page, her conference automatically switches to the one associated with the new page. To facilitate this switching, we again use the client SIP API. All 'enabled' web pages contain a signed applet which simply writes a SameDeviceJoin message to the client SIP service socket. As a result whenever a client loads the page, the applet writes the SameDeviceJoin command and the user is brought into the conference of that particular web page. Note that in case there is no active device, the SameDeviceJoin acts as an ExternalJoin, requiring user input for device selection. In addition, to provide users with information about other viewers at that time, we use SIP based SUBSCRIBE/NOTIFY features notifying users of all Join and Leave events. This application is an attempt at community based communication. For example, this would allow a community of movie fans reviewing a particular movie to participate in a voice discussion amongst online fans in real-time, as opposed to text chat.

8. DISCUSSION

In this section, we describe the advantages of the SPLAT architecture and also discuss the issue of security and privacy with such SIP based services.

The primary advantages of using SPLAT, as designed, are as follows:

- *SIP as a system-service*: Using SPLAT, SIP becomes a first class protocol for operating systems similar to other OS communication protocols. This allows a wide range of application enablement (as demonstrated in Section 7) and development of new collaborative applications using the underlying SIP infrastructure.
- *Integration with essential infrastructure*: In order for SIP to become an attractive option for pervasive use, there is a requirement of complimentary tools such as multiparty audio conferencing, especially *ad hoc* and context based conference management. The SPLAT infrastructure blocks provide such essential tools.
- *Extensible API architecture*: SPLAT client side service uses XML messages to perform functions such as session initiation, session switch and so on. This extensible design allows for easy upgradation of the API. In addition, to support highly specialized applications, the 'tunnel' API call allows for developers to plug in customized code, thus serving a broader base of applications.
- *Isolation of audio devices from platform functionality*: In SPLAT, any SIP capable audio device can be used on a *per-session* basis without any reconfiguration. This is possible due to the higher level SIP service API which offers similar semantics for all devices using device specific wrappers.

A lot of recent research activity has focussed on the issue of security and privacy in SIP based VoIP. The use of 'sips'—*secure-sip* (analogous to https for http) provides encryption to the transmitted data. In relation to SPLAT, most of the current research is directly applicable. A recent work by Deruelle *et al.* [22] defines an extended SIP-specific Java security model to run customized services at SIP signalling servers. Such services can be used to block calls or take other autonomous actions on behalf of the UA in response to SIP messages.

9. RELATED WORK

There have been other efforts in designing APIs and languages for SIP services. Call processing language (CPL) [23] is an XML-based scripting language for describing and controlling call services. Users create CPL scripts and these run on signalling servers. Since CPL is essentially a server side utility, it focuses on network services and not end-user service. In contrast the SPLAT API is a client side utility and brings SIP to the client desktop. Our vision through SPLAT, is of SIP being an OS resident protocol with a packaged underlying SIP stack and the SPLAT client side service and the infrastructure block completing the SIP-enabled environment.

JAIN SIP [15] is a low level API and provides a standard portable interface to share information between SIP Clients and SIP Servers. Since it is a low level API, it does not provide high-level abstractions such as SameDeviceJoin. Coexistence of a range of services on a single end system all using the same SIP stack and API is also not defined by the JAIN API.

Language for end system services (LESS) [24] is a XML based language for end system services. It mainly differs from our utility in the fact that it is a language not an API. As demonstrated by us, a wide range of applications can effectively use our APIs to use SIP effectively and efficiently. In contrast, applications will have to be programmed using LESS, thus making it much harder for existing applications to exploit the richness of SIP.

10. CONCLUSIONS

In this paper, we designed a SIP based system service platform called SPLAT. SPLAT provides an extensible client-side API to provide a unified mechanism of executing generic SIP functions. Importantly, it makes SIP functionality accessible to all applications irrespective of their native protocols. The infrastructure is supported by a number of network building blocks like a conference server and a game server. The conference server provides the basic conferencing mechanism which allow multiple parties to communicate. The game server is used to establish context between multiple parties, within which they can communicate with each other. We demonstrated the utility of such a platform by enhancing existing collaborative applications like web browsers and instant messaging application as well designed new applications such as seamless conferencing for gaming and community-based web browsing. In future, we will further explore the use of SPLAT. The efficiency and richness of the SIP protocol makes it an attractive consolidation engine for various applications and we intend to use SPLAT as the enabling platform for such control centers.

REFERENCES

1. Vonage. <http://www.vonage.com>, 2005.
2. Skype. <http://www.skype.com>, 2005.
3. Rosenberg J, Schulzrinne H, Camarillo G, Johnston AR, Peterson J, Sparks R, Handley M, Schooler E. SIP: session initiation protocol. RFC 3261, Internet Engineering Task Force, June 2002.
4. Campbell B, Rosenberg J, Schulzrinne H, Huitema C, Gurle D. Session initiation protocol (SIP) extension for instant messaging. RFC 3428, Internet Engineering Task Force, December 2002.
5. Campbell B, Mahy R, Jennings C. The message session realy protocol. SIMPLE Working Group Internet Draft, February 2005.
6. Pingtel SIP Softphone. <http://www.pingtel.com/solutions/instantxpressa.jsp>, 2005.

7. SJPhone. <http://www.sjlabs.com>, 2005.
8. Cisco IP Phones. <http://www.cisco.com/en/US/products/hw/phones/ps379>, 2005.
9. Koskelainen P, Schulzrinne H, Wu X. A SIP-based conference control framework. *NOSSDAV*, Miami Beach, FL, U.S.A., 2002.
10. Milandinovic I, Stadler J. Multiparty conference signaling using SIP. *International Network Conference*, Plymouth, U.K., 2002.
11. Singh A, Acharya A. Using session initiation protocol to build context-aware VoIP support for multiplayer networked games. *NETGAMES*, Portland, OR, U.S.A., 2004.
12. Akkawi A, Schaller S, Wellnitz O, Wolf L. A mobile gaming platform for the IMS. *NETGAMES*, Portland, OR, U.S.A., 2004.
13. Singh A, Mahadevan P, Acharya A, Shae Z. Design and implementation of SIP network and client services. *ICCCN*, Chicago, IL, U.S.A., 2004.
14. Microsoft Live Communications Server. <http://office.microsoft.com/en-us/FX011353441033.aspx>, 2005.
15. JSR 32: JAIN SIP API Specification. <http://jcp.org/aboutJava/communityprocess/final/jsr032/index.html>.
16. Handley M, Jacobson V. SDP: session description protocol. RFC 2327, Internet Engineering Task Force, April 1998.
17. Roach A. Session initiation protocol (SIP)-specific event notification. RFC 3265, Internet Engineering Task Force, June 2002.
18. Conveidia. <http://www.conveidia.com>, 2003.
19. Rosenberg J, Peterson J, Schulzrinne H, Camarillo G. Best current practices for third party call control in SIP. RFC 3725, Internet Engineering Task Force, April 2004.
20. Singh A, Acharya A. Multiplayer networked gaming with the session initiation protocol. *Elsevier Computer Networks* 2005; **49**(1):38–51.
21. Lotus Sametime. <http://www.lotus.com/products/lotussametime.nsf/wdocs/about>.
22. Deruelle J, Ranganathan M, Deruelle J, Montgomery D. Programmable active services for SIP. National Institutes of Standards and Technology Report, 2004.
23. Lennox J, Wu X, Schulzrinne H. CPL: a language for user control of internet telephony services. Internet Draft, October 2004.
24. Wu X, Schulzrinne H. Programmable end system services using SIP. *IEEE International Conference on Communications*, Anchorage, AL, U.S.A., 2003.

AUTHORS' BIOGRAPHIES



Aameek Singh is a doctoral student in the College of Computing at Georgia Tech, where he is a member of the Distributed Data Intensive Systems Lab and the Center for Experimental Research in Computer Science (CERCS). His research interests are in the area of distributed storage systems, SIP/VoIP and WWW. He is a winner of the prestigious IBM PhD Fellowship for the year 2005–2006. He is a graduate of the Department of Computer Science and Engineering at Indian Institute of Technology, Bombay. More information is available at <http://www.cc.gatech.edu/~aameek>



Dr Arup Acharya is a Research Staff Member in the Internet Infrastructure and Computing Utilities department at IBM T.J. Watson Research Center and leads the advanced networking micropractice for On-Demand Innovation Services within IBM Research. His current research focusses on scalability and server enhancements for SIP and SIP-based applications such as VoIP, instant messaging & presence as well as cooperative protocols for wireless mesh networks. He has worked on consulting engagements worldwide applying his expertise in emerging technology areas such as SIP, IMS and IPv6 to assess and shape customer strategies in these areas. He actively collaborates with the academic community and holds a Visiting Professor position at WINLAB, Rutgers University. Dr Acharya has published in leading conferences and journals in the area of networking architecture and

protocols, holds seven patents and has chaired several conferences. He has been invited to present tutorials on SIP as well as mobile wireless networks at several international conferences. He received his BTech degree in Computer Science & Engineering from the Indian Institute of Technology, Kharagpur and a PhD in Computer Science from Rutgers University in 1995. He was with NEC C&C Research Labs prior to joining IBM Research. Further information is available at <http://www.research.ibm.com/people/a/arup/>



Priya Mahadevan is currently a PhD student at UC San Diego, working with Dr Amin Vahdat. She received her MS in Computer Science from Duke University in December 2003. Her research interests are broadly in the areas of network topologies, VoIP, large-scale distributed systems and mobile computing.



Zon-Yin Shae is with IBM T.J. Watson Research Center in Yorktown Heights, New York. He works in the areas of multimedia networking, network infrastructure resiliency and root cause analysis, SIP/VoIP converged networks, multimedia traffic and data analysis. He received his BA and MS in Electronic Engineering from the National Chiao-Tung University at Taiwan, and his PhD in Electrical Engineering from the University of Pennsylvania at Philadelphia. zshae@us.ibm.com