

SIPcast: a Toolkit for Fast Prototyping IMS Application Server

Wang Dawei, Hao Ruibing, Chen Jianfeng
(wdw, rbhao, chenjf@lucent.com)

Abstract:

IMS is the architecture of the converged 3G/mobile Internet and SIP is the signaling protocol adopted by IMS. Support for converged multimedia applications is the most advocated differentiator of IMS. In this paper, we introduce a toolkit for fast prototyping novel IMS converged multimedia application servers, SIPcast, and two demo applications developed with it. SIPcast provides a set of SIP agnostic application API, a fully scalable application running platform. The SIPcast API, combined with the application running platform, provides a fast prototyping capability to build IMS multimedia applications.

SIPcast is also a “magic stick” for converting existing “MuggleNet” applications into IMS applications. The API it provided is compatible with the popular Intel Dialogic API and thus can ease the migration of vast existing Intel Dialogic card based CTI applications to IMS.

Based on SIPcast, we have developed two demo applications: v-Chat, which is an IMS multimedia application for group communication and content sharing, and i-Karaoke, which is an IMS online karaoke application with streamed MP3-quality accompanying music.

Keywords:

IMS, SIP, Application server, Internet Karaoke

1. Background and motivations

Network convergence, or the combining of voice, data and video networks into a single system that uses Internet Protocol to process information, has become a hot trend within the IT industry during recent years. Convergence offers many advantages like adaptability, easier maintenance and increased reliability through redundancy has impelled the evolution of traditional applications. Traditional CTI applications are mostly based on specialized communication controllers such as Intel Dialogic Card. The capacities of such applications are limited by the capacity of those controllers. In the convergence network, however, VoIP signaling is handled in the packet switched networks so that the scalabilities can be easily improved through various computing technologies such as clusters. SIPcast is targeted for this movement. The API it provided is compatible with the popular Intel Dialogic API and thus can ease the migration of vast existing Intel Dialogic card based CTI applications to VoIP applications.

The Internet has experienced a dramatic growth in the last years. It has evolved from a small network linking a few research sites to a massive worldwide network. In the same time, cellular telephone networks provide services to over one billion users worldwide. The IP Multimedia Subsystem (IMS) is the technology that will merge the internet with the cellular world. It will make Internet technologies, such as the web, email, instant messaging, presence, videoconferencing, and multimedia applications available nearly everywhere. SIP is one of the

most active initiatives underway in the IETF today and adopted by 3GPP as the signaling protocol of IMS. This is another motivation for us to put forward SIPcast toolkit.

SIP is one of the most active protocols defined by IETF, which has many extensions. Development on the top of SIP needs good knowledge of SIP protocol. SIPcast provides a set of SIP agnostic application API. It is just like developing on a normal telephone. Therefore, SIPcast provides a fast prototyping capability to build IMS multimedia applications.

The rest of the paper is organized as follows. In Section 2, we will present SIPcast platform and then describe each component of SIPcast in detail. In Section 3, we will introduce the usage of SIPcast and the potential applications that can be built on it. In Section 4, we will introduce two demo applications developed on SIPcast, vChat and iKaraoke. Finally we conclude this paper in the section 5.

2. SIPcast introduction

SIPcast is a platform for developing SIP-based multimedia applications. It contains signaling processor for SIP signal processing, media processor as an embedded media server, resource manager for the management of various resources, media resources such as mixers and a set of API for applications (see figure 1). Scalability is one of our design targets. Therefore the SIPcast platform is highly distributed to allow backup and load balance. Each component mentioned above can be run in separated hosts. TCP connections are established as the internal signaling channels between each component. We use private protocols for the communications between components.

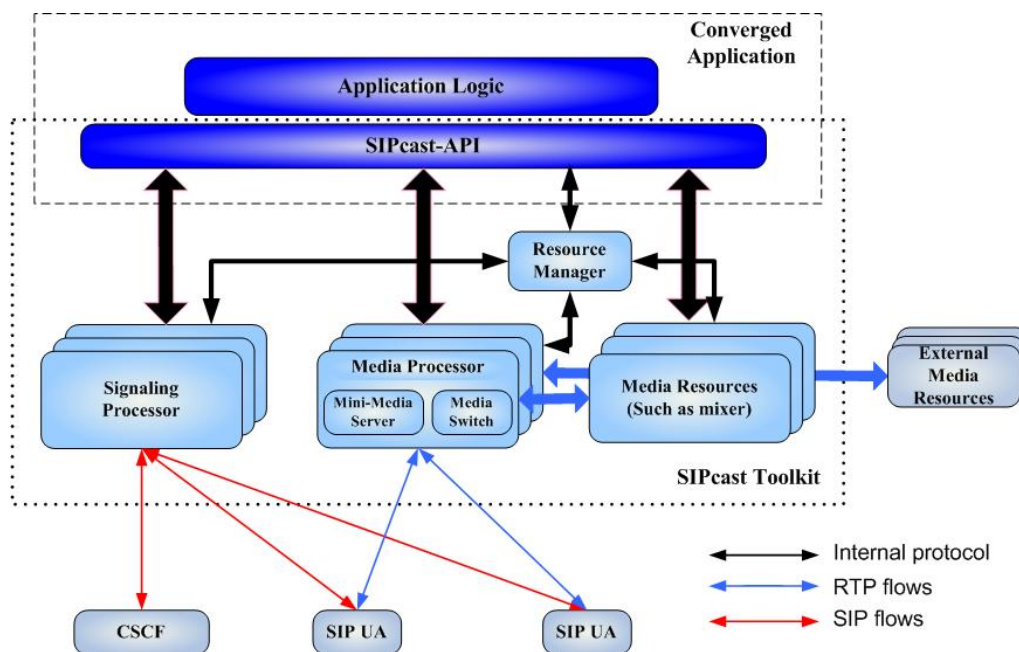


Figure 1 The architecture of SIPcast toolkit

2.1 Signaling processor

Signaling processor is the most important part of SIPcast platform. It contains SIP stack and all SIP signals are processed here. Originally we use open source SIP stack, KPhone stack, to implement this component. Now we have migrated it to Lucent proprietary SIP stack,

siptrans/isurf. Incoming SIP messages are translated into well-defined event to the SIPcast API module and a set of functions are provided for API module to customize the underlying stack and send out SIP messages. Combined with the encapsulation of SIPcast API module, SIP protocol is concealed from applications.

From figure 1, we can see the interfaces with other components include SIPcast API module and resource manager. When signaling processor starts up, it registers its control port with the resource manager using a TCP connection. The status of signaling processor can also be reported to the resource manager using this connection. The API module requests signaling processor resource from resource manager. The resource manager select one signaling processor from all the signaling processors registered with it according to the load of each signaling processor. The control port of the assigned signaling processor is passed to the API module, and then the API module sets up a TCP connection to the signaling processor's control port. All the upstream events and downstream commands are passed using this TCP connection.

From figure 1, we can also see that only the signaling processor communicates with peers through SIP protocol. SIP signals may be sent to the signaling processor directly or be routed by CSCF.

One SIPcast application only needs one signaling processor to serve for it while one signaling processor can serve for multiple SIPcast applications. To make the SIPcast platform even more scalable, multiple signaling processor instances can be started and register with the resource manager. It enables load balance among the multiple signaling processors in the SIPcast platform.

2.2 Media processor

The function of the media processor is the media processing. Media processor sends and receives RTP flows from SIP UA peers (figure 1). The media processing function can be divided into two parts. The first one is a mini-media server and the second one is a media switch. The embedded mini-media server can play an announcement, record user voice and detect user's DTMF input while the media switch can route the RTP flows to an external media processing resources or platform integrated media resources such as mixers. The media switch acts as a RTP proxy so that in the view of the SIP peer, it talks with a consistent peer while the RTP traffic is processed by different servers. Similar with signaling processor, several media events are defined to indicate SIPcast API module the media processor event such detection of user input or the end of announcement. In addition, a set of media processing functions are provided for API module.

From figure 1, we can see the interfaces with other components include SIPcast API module, resource manager or platform integrated media resources. Similar with the signaling processor, when the media processor starts up, it registers its control port with the resource manager using a TCP connection. The status of media processor can also be reported to the resource manager using this connection. When the SIPcast API module requests media processor resource from the resource manager, the resource manager selects one media processor from all the media processors registered with it according to the load of each one. The control port of the assigned media processor is passed to the API module, and then the API module sets up a TCP connection to the media processor's control port. All the upstream events and downstream media commands are passed using this TCP connection.

One media processor can serve for multiple SIPcast applications. Multiple media processor instances can be started and register with the resource manager. It enables load balance among the multiple media processors in the SIPcast platform.

2.3 Media resources

In the SIPcast platform, we also integrate other media resources. They are specific media processing component. They use the RTP flows routed by the media processor as the input media. After the specific media processing, output media are transferred back to the media processor, which routed the returned RTP flows to the SIP peers.

Currently we implement an audio mixer, which is the core component of the conference. The role of users in the mixer can be a listener or a speaker. Listeners can only listen to other and can not speak. Each user in the same mixer can receive an audio flow mixing all the users' audio except the audio from itself while listeners can receive an audio flow mixing all the speakers' audio.

When the mixer starts up, it registers with the resource manager with its control port. Each mixer can only serve for one conference room. When an application needs to create a conference room, it requests for a mixer from the resource manager and gets the control port of the mixer. And then the application can set up a TCP connection to the mixer. This connection is used by applications to control the behavior of the mixer.

The mixer we implemented also supports background music mixing. The application can customize the mixer with background music through the control connection. It can be customized at the user level. If the background music is selected, the user can hear the background music in the conference.

From the description above, we can know there are three interfaces between media resources and other components. The first one is the registration channel with the resource manager. The second is the control connection with application and the last one is the RTP exchange with the media processor.

2.4 Resource manager

The resource manager is used for the management of various resources in the SIPcast platform. The functions of the resource manager include resources registration, resources assignment and resources status monitoring. There are many kinds of resources such as signaling processors, media processors and mixers. More resources may be added to the current platform.

When each resource starts up, it registers to the resource manager with its control port. It also uses the registration connection to report the status of the resources, for example the current users or sessions in the resource. This information can be used by the resource manager as a resource assignment criterion.

The resource manager also provides an interface to query the status of the resources registered and set the policy for the resource assignment.

2.5 SIPcast API

The SIPcast API provides the APIs to applications. It is a static or dynamic library which must be compiled with applications. SIPcast provides a set of SIP agnostic application API. All the SIP protocol details are hidden through encapsulations. SIPcast API abstracts a set of well-defined functions and events to the application.

When the application initializes the SIPcast platform, the API module will set up a connection with the resource manager. Upon the demand of the application, SIPcast API module will request resources needed from the resources, and then, set up the connection to the resource assigned and

execute the command of the application. The details of the SIPcast API will be illustrated in the next section.

3. The usage of SIPcast and potential applications.

In the section, we summary the SIPcast API and introduce the usage of the SIPcast platform. SIPcast API abstracts a set of well-defined functions and events to the application. The event sent to the application is used as a notification of the underlying signaling, media and system events, while the functions are the application commands sent to the underlying components.

3.1 SIPcast events

SIPcast events can be divided into signaling events and media events. Signaling events are sent to the API module from signaling processors while media events are from media processors. The following table (table 1) lists all the events in the SIPcast API and the event parameters of each event.

evtType	callId	userId	userAddr	evtReason	Digit	mediaPort
ALERTING	√	√	√	√	×	×
RING_BACK	√	√	×	√	×	×
CONNECTED	√	×	*	√	×	√
DISCONNECTED	√	×	×	√	×	√
DIGITS	√	×	×	√	√	×
PLAY_END	√	×	×	√	*	×
REC_END	√	×	×	√	*	×
TIME_OUT	√	×	×	√	*	×
ERROR	√	×	×	√	×	×

√ — must be set, × — not available, * — depends on the event reason

Table 1 Events of SIPcast API

The event parameters include:

- callId — call ID which the event related to.
- userId — peer ID of the current call.
- userAddr — peer address of the current call.
- evtReason — the reason of the event.
- Digit — digits of the DTMF input
- mediaPort — peer's media address of the current call

Short description of events

- ALERTING — there is an incoming call.
- RING_BACK — an outgoing call is ringing.
- CONNECTED — the current call is connected.
- DISCONNECTED — the current call is disconnected.
- DIGITS — there are DTMF input from the call peer.
- PLAY_END — an announcement playing is ended.
- REC_END — voice recording is ended.
- TIME_OUT — timer set by the application expires.

ERROR — error occurs in the underlying components

3.2 SIPcast functions

The SIPcast functions can be divided into system functions, signal functions and media functions. All the functions are listed below.

3.2.1 System functions

System functions are used for SIPcast platform initialization, clean up and component management.

- `int SE_Startup(CFG_STRUCT *config);`

This function is used to initialize the SIPcast platform. The address of the resource manager is contained in the function parameter.

- `int SE_ShutDown(void);`

This function is used to shut down the SIPcast platform.

- `int SE_WaitEvent(int timeout);`

This function is used to wait for the underlying event. Similar with “select” socket function, the parameter of this function indicate whether the function returns immediately, blocks indefinitely or wait for a specified time.

- `int SE_GetEvent(EVT_STRUCT **event);`

This function is used to fetch the event when the SE_WaitEvent function returns an event.

- `int SE_FreeEvent(EVT_STRUCT *event);`

This function is used to free the resource allocate for the incoming event.

- `int SE_UsrReg(USR_STRUCT *usrinfo);`

This function is used to register to the SIP registrar. The address of the registrar and the user ID are contained in the parameter.

- `int SE_GetMixer(MIXER_HANDLE *mixer);`

This function is used to request a mixer from the resource manager.

- `int SE_FreeMixer(MIXER_HANDLE mixer);`

This function is used to release a mixer previously requested.

3.2.2 Signal functions

Signal functions are the commands sent to the signaling processor.

- `int SE_MakeCall(USER_ID userId);`

This function is used to make a new call. The callee’s phone number is contained in the parameter.

- `int SE_AcceptCall(CALL_ID callId);`

This function is used to accept an incoming call.

- `int SE_ReleaseCall(CALL_ID callId);`

This function is used to hang up the call.

- `int SE_GetCallInfo(CALL_ID callId, CALL_INFO *info);`

This function is used to get the call information.

- `int SE_SetTimer(CALL_ID callId, int timeout);`

This function is used to set a timer for the specified call.

- `int SE_StopTimer(CALL_ID callId);`

This function is used to stop the timer for the specified call.

3.2.3 Media functions

Media functions are the commands sent to the media processor.

- `int SE_Prompt(CALL_ID callId, char *voiceLink, TermParam termination)`

This function is used to play an announcement to the specified call. The URL of the announcement is contained in the function parameter. It can be a local file or a remote file which can be retrieved through HTTP and FTP. In the function, the termination condition is also set in the function parameter. The termination condition may be any DTMF input, specific DTMF input or maximum silence time. The termination parameter is also used in the following functions with the same meaning.

- `int SE_StopPlay(CALL_ID callId)`

This function is used to stop the announcement on the specified call.

- `int SE_Record(CALL_ID callId, char *recLink, TermParam termination)`

This function is used to record user voice on the specified call. The recorded voice is saved in a local file which is specified in the parameter. The termination condition is also set in the parameter.

- `int SE_StopRecord(CALL_ID callId)`

This function is used to stop recording on the specified call.

- `int SE_GetDigit(CALL_ID callId, char *recvDigit, TermParam termination) ;`

This function is used to get the DTMF input on the specified call. The DTMF input is returned in the digit buffer passed in the function parameter. The termination condition is set in the parameter.

- `int SE_ClearDigit(CALL_ID callId) ;`

This function is used to clear the digit buffer maintained in the API module for the specified call.

- `int SE_SwitchToMixer(CALL_ID callId, MIX_HANDLE mixer, int mode);`

This function is used to switch the user's RTP flow to a platform-integrated mixer. The mixer handle is specified in the parameter, which is requested previously using `SE_GetMixer` function. The switch mode can be unidirectional or bidirectional.

- `int SE_SwitchToRemote(CALL_ID callId, struct sockaddr_in remote, int mode) ;`

This function is used to switch the user's RTP flow to an external media server. The address of the media server is specified in the parameter. The switch mode can be unidirectional or bidirectional.

3.3 Usage of the SIPcast toolkit

The following C-style code illustrates the basic flow of SIPcast-based applications.

```

int main( int argc, char **argv ){
    // variable declaration and definition for the specific application
    EVT_STRUCT *event;
    ...
    // the initialization API should be called firstly
    if( SE_Startup( confStruct ) < 0 )
        ...
    while( 1 ){
        SE_WaitEvent( 0 );
        SE_GetEvent( &event );
        switch( event->evtType ) {
            case ALERTING :
                SE_AcceptCall( event->callId, 0 );
            case CONNECTED : // SIP call established event
            case DISCONNECTED: // SIP call disconnected event
            case DIGITS: // User key-pad pressing event
            case PLAY_END: // Playing out voice end event
            case REC_END: // Recording voice end event
            case TIME_OUT: // Time out event
            case ERROR :
                } // End of switch
        } // End of while(1)
        SE_ShutDown();
        return 0;
    } // End of main()
}

```

3.4 Potential applications of the SIPcast toolkit

Based on the SIPcast platform, SIP-based multimedia application server can be built without the knowledge of SIP protocol. It is very convenient to migrate traditional CTI applications to IMS-based applications. In the next section, we will introduce two demo applications developed using SIPcast platform.

4. An introduction of vChat and iKaraoke

4.1 vChat

vChat application is an IMS multimedia conferencing/chatting application for group communication and information sharing based on SIPcast. The key features include on-demand service creation and customization, multi-speaker support and flexible floor control, media mixing with customizable background music and participant list notification, “Look who’s talking” feature.

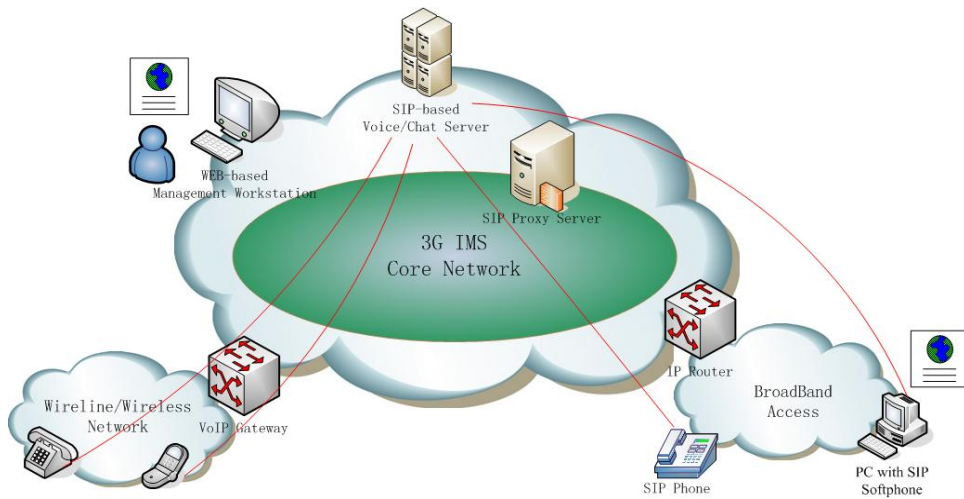


Figure 2 vChat demo application

4.2 iKaraoke

iKaraoke is an immersive SIP based application, which allows users to celebrate together with friends from different locations. Comparing with existing online KTV, this Internet Karaoke system allows singer to select accompanying music with MP3 format from server, all the singer and listeners can see the lyric synchronously; pictures or videos can also be shared among group members. The key features include streaming MP3 quality accompanying music, synchronized MP3 music/Lyric/Singing, “Look Who’s Singing/Look Who’s Waiting to Sing” and singing on demand.

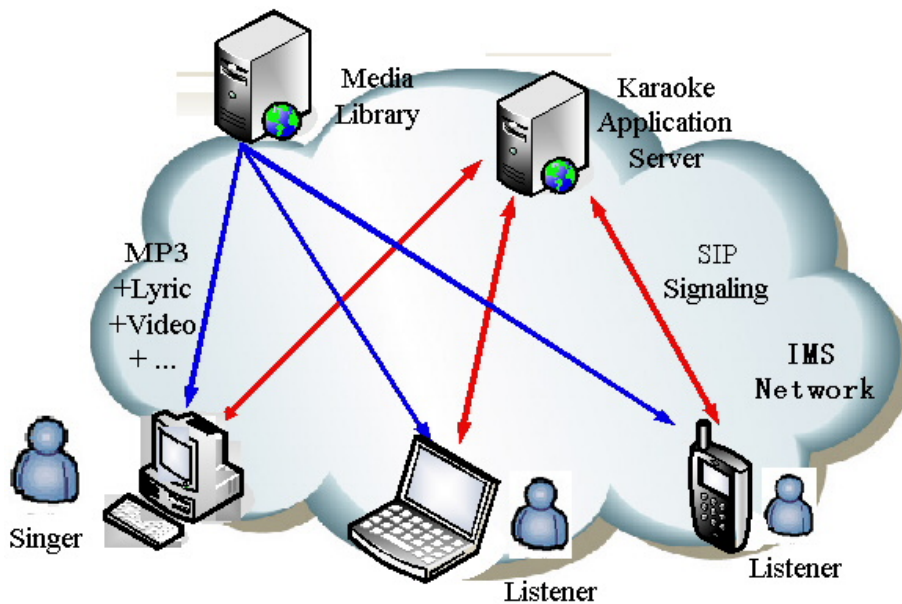


Figure 3, Internet karaoke application

5. Conclusion

The value of SIPcast toolkit lies in the following aspect. Firstly, SIPcast provides a set of SIP agnostic application API. It is very simple and compatible with the popular Intel Dialogic API and thus can ease the migration of vast existing Intel Dialogic card based CTI applications to IMS.

Therefore, it enables the fast prototyping novel IMS converged multimedia application servers. Secondly, scalability is a key factor when we design SIPcast. Each component can run in different hosts and can start multiple instances. It is easy to increase the system capacity through starting more instances of the scarce resources. It breaks through the capacity limitation of the traditional CTI application in this way. In the resource manager, an assignment policy can be also implemented to enable the backup and load balance.

Reference

- [1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [2] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC3550, July 2003.
- [3] Gonzalo Camarillo and Miguel A. Garcia-Martin, "The 3G SIP Multimedia Subsystem (IMS)", John Wiley & Sons, Ltd., August 2004.
- [4] Edwin Margulies, "For the record, this is Computer Telephony", Computer Telephony Expo Spring 2000
- [5] Levin, O. and R. Even., "High Level Requirements for Tightly Coupled SIP Conferencing", draft-ietf-sipping-conferencing-requirements-01 (work in progress), October 2004.
- [6] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol", draft-ietf-sipping-conferencing-framework-03 (work in progress), October 2004.