VOCAL

Vovida Open Communication Application Library

Session Initiation Protocol (SIP) Stack

Software Version 1.2.1

Copyright Copyright © 2001, Cisco Systems, Inc.

Revision History The following table itemizes the revision history of this manual:

Software Version	*Guide Version*	*Date*	*Comments*

Table 1.

Revision	*Date*	*Originator*	*Comments*

Table 2.

Version This manual is written to support Vovida.org's SIP Stack Version 1.2.1.

Support The primary location for support, information and assistance for the VOCAL system is www.vovida.org. This site contains other documentation, training materials, development tools, development resources and informational mailing lists.

Preface

Introduction

This manual provides information about the Session Initiation Protocol (SIP) stack and how it is used by other applications within the VOCAL system.

Objectives of this manual

This manual is intended as a starting point for any developer who is interested in working with the SIP Stack.

Intended audience

Software developers, architects, product managers and anyone else who is interested in the specific details regarding the function and coding of the VOCAL system.

Organization

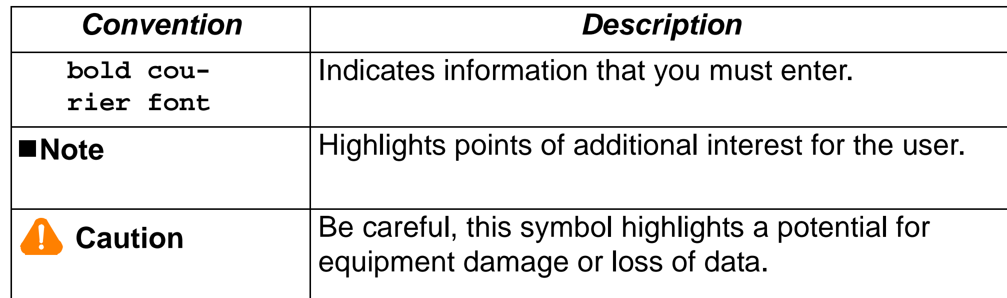
This guide is organized as follows:

Chapter	***Title***	***Description***
Chapter 1	Top Level Design	A high level overview of the SIP stack.

Documentation Conventions

The following is a list of conventions used in this guide:

Convention	***Description***
bold text	Names of elements found on the GUI screen, including buttons, and selectable entities such as, servers and server groups.
< >	Text that appears between angle brackets describes variables such as, <group name>.
`courier font`	System responses and prompts either from the CLI or GUI.



Convention	*Description*
`bold courier font`	Indicates information that you must enter.
■**Note**	Highlights points of additional interest for the user.
Caution	Be careful, this symbol highlights a potential for equipment damage or loss of data.

Additional resources

Publications

A System Administration Guide, which covers adding users, SNMP message flows, call flows and working with features is also available from http://www.vovida.org.

On-Line Resources

Vovida.org (http://www.vovida.org) is a community web site dedicated to providing a forum for open source software used in datacom and telecom environment. This site was created to provide an environment where open source communications information and software can be easily located, accessed, retrieved and shared.

Table of Contents

Preface. iii

Chapter 1.
Top Level Design
Architecture 1-2
Class Diagrams 1-3
Classes Involved 1-4
Components 1-6
Creating 1-7
Manipulating 1-8
Interaction with SDP Stack 1-9
SIP Transceiver 1-11
High Level Design 1-12
High Level Message Flow 1-13
Transaction Databases. 1-17
Data Structures. 1-18
Future Work 1-20
Bugs/Limitations 1-21

1

Top Level Design

Introduction

This provides the functionalities enlisted in RFC2453. The components that it interacts with are UDP stack, TCP stack, SDP stack.

The architecture is explained below, later leading to class diagrams, major modules, threads involved and sequence diagrams for important flows in the design

Topic	See Page
Architecture	1-2
Class Diagrams	1-3
Classes Involved	1-4
Components	1-6
Creating	1-7
Manipulating	1-8
Interaction with SDP Stack	1-9
An Example	1-9
SIP Transceiver	1-11
High Level Design	1-12
High Level Message Flow	1-13
Transaction Databases	1-17
Data Structures	1-18
Threads and Interaction	1-18
Future Work	1-20
Bugs/Limitations	1-21

Architecture

Major Components

The major components of the SIP stack are:

- creating (Constructing)
- manipulating(Parser)
- sending (Transceiver)
- receiving(Transceiver)
- maintaining transactions(TransactionDatabase)
- retransmitterTransceiver, TransactionDataBase) &
- filter(Transceiver, TransactionDataBase) of SIP messages.

Illustration

Figure 1-1 shows how the major components of the SIP stack work together.

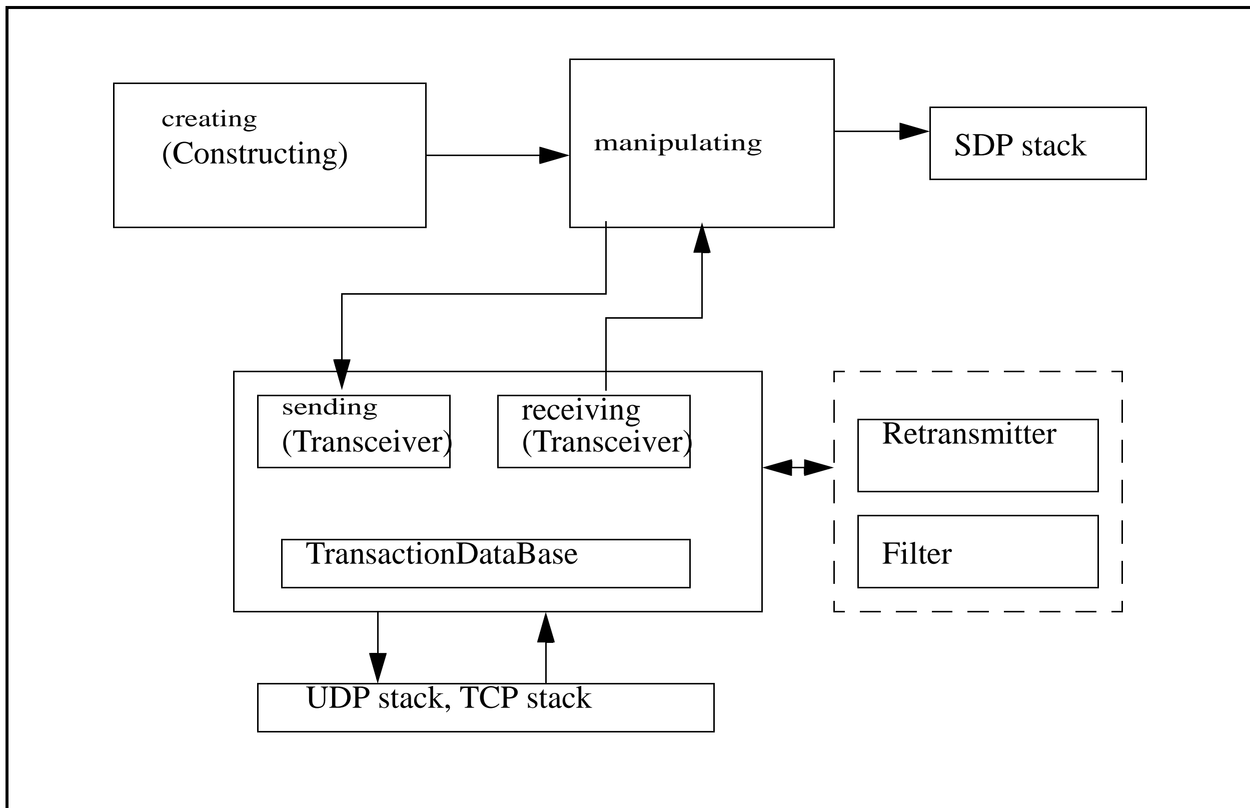


Figure 1-1. SIP Stack Components

Class Diagrams

Links

These diagrams are too large to fit into this document and are best viewed as links.

- Header.gif
- Message.gif
- MimeData.gif
- SipAgent.gif
- Transceiver.gif

Classes Involved

General information

SipMsg is the base class for all sip messages. Two things are derived fromit, the StatusMsg and the SipCommand. A StatusMessage carries one of the 3digit response codes.

SipCommand

A SipCommand is the base class for all the other SIP requests which include:

- AckMsg
- ByeMsg
- CancelMsg
- InviteMsg
- RegisterMsg
- SignalMsg
- OptionsMsg
- NotifyMsg
- SubscribeMsg
- ReferMsg.

Data Containers

The other class are just data containers. Basically they just keep track of a string the represents the information that heir name implies.

These include:

- SipAccept
- SipAcceptEncoding
- SipAcceptLanguage
- SipAllow
- SipAuthorization
- SipCSeq
- SipCallId
- SipCallLeg
- SipContactSipContentEncoding
- SipContentLength
- SipContentType
- SipDate
- SipEncryption
- SipExpiers
- SipFrom
- SipHide
- SipMaxForwatrds
- SipOrganization
- SipPriority

- SipProxyAuthenticate
- SipProxyAuthorization
- SipProxyRequire
- SipRecordRoute
- SipRepsonseKey
- SipRequestLine
- SipRequire
- SipRetryAfter
- SipRoute
- SipSdp
- SipServer
- SipRequestLine
- SipStatusLine
- SipSubject
- SipTimestamp
- SipTo
- SipUnsupported
- SipUrl
- SipUserAgent
- SipVia
- SipWarning
- SipWwwAuthenticate

Components

Introduction

This section covers information about the following topics:

- Creating
- Manipulating
- Interaction with the SDP Stack

Creating

Example 1

Creating a SIP INVITE message - look at sip/test/InviteMsgTest.cxx
InviteMsg(Sptr<BaseUrl> sendToUrl, int SIPlistenPort, int rtpPort)
Refer to Class Diagram for BaseUrl explanation.

Example 2

Creating a SIP 200 OK message - look at sip/test/StatusMsgTest.c
StatusMsg(SipCommand& command, int statusCode)
SipCommand is any of the request objects.
Refer to Class Diagram for SipCommand explanation.

Example 3

Creating a SIP ACK message - look at sip/test/AckMsgTest.cxx
AckMsg::AckMsg(StatusMsg& statusMsg)
StatusMsg the response object
Refer to Class Diagram for StatusMsg explanation.

Example 4

Creating a SIP BYE message - look at sip/test/ByeMsgTest.cxx

Manipulating

Example 1

To change the Request Uri of the already constructed INVITE message:

```
InviteMsg inviteMsg;
SipRequestLine reqline = inviteMsg.getRequestLine();
Sptr<BaseUrl> url = reqline.getUrl();
//change hostname in the url, and set it back.
url.setHost(Data("newHost"));
reqline.setUrl(url);
inviteMsg.setRequestLine(reqline)
```

Look at header files of individual headers to identify the get and set methods provided.

Example 2

To decode a message:

```
InviteMsg inviteMsg(Data  message)
```

This calls SipMsg::decode(Data),

-- talk about MIME data being decoded.

//find out what happens here

If we receive an INVITE message, and require the user in the Uri, of From field, the following sequence happens. This can be added in the sequence diagram also.

```
string message;
InviteMsg inviteMsg(Data(message));
This calls : SipMsg(Data(message)) , which calls Sip-
HeaderContainer()... (bryan can comment here)
```

Now, when we request From header from the message,

```
SipFrom from = inviteMsg.getFrom(); --- >
```

the From header line is decoded here.

Example 3

If you want to encode the message.

```
InviteMsg inviteMsg;
Data inviteStr =  inviteMsg.encode()
```

InviteMsg::encode()

calls SipMsg::encode(),

which calls the individual Headers::encode(),

and MimeEncode().

Interaction with SDP Stack

Introduction

When we need to encode, decode or alter SDP MIME information, we use SDP stack.

An Example

Encoding SDP MIME

To encode a SDP MIME information.

Parser

When a SIP message is received, as a char string, the Parser functions are invoked to form the corresponding SIP object, as explained above. Here, lazy parsing is implemented, which is that the Header objects don't actually get constructed, when the parser is invoked. In fact, it gets constructed when the objects are accessed.

Each header object takes care of parsing its line. The MIME data is parsed by another class, SipContentDataContainer, as explained above.

Sending/Receiving Messages:

The stack supports sending/receiving messages on both UDP and TCP. The Via field is looked at to figure, if the message needs to get sent on TCP or UDP. For more information about the threads, see "Threads and Interaction" on page 1-18.

The stack maintains state of all requests and final response for INVTE, to match with the ACK. These are maintained for a maximum of 32 seconds. This performs automatic retransmissions for UDP, and the maximum number of retransmissions is a configurable parameter. The T1 for retransmission is 600ms, and the next consecutive ones follow an interval of 2*T1.

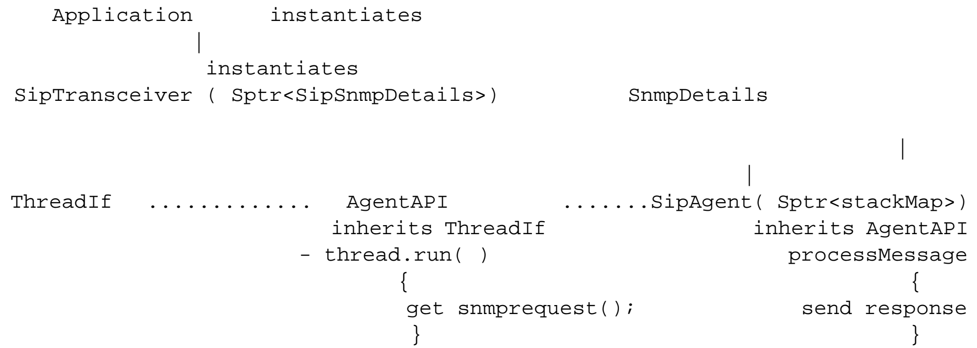
In case of commands, such as INVITE, the SipTranceiver sends a 408 Request TimedOut to the application if it gets no response after the max. number of retrans. However, it is up to the application to handle timeouts for responses.

NAT Traversal

If this is turned on, the stack, puts the address, where the packet was actually received from, in the received param, in the Via header, iff, the actual Via address is different from that on which the packet was received on. Also, while sending responses, the packet will be sent to the received param in the Via field, if present, else it will be sent to the Via address.

SNMP Support

The sipagent maintains a table of the mib entries of the particular instance of the sipstack. When the transceiver is instantiated a sipagent is also instantiated within with an arguement of the application name. The SipAgent class talks to the snmp manager, which queries the SipAgent regarding the statistics of the sipstack, snmp Manager can also setcertain attributes but this functionality is not implemented as this point. The class which handles all of the above is SipTransceiver.



SIPSnmpDetails

SipSnmpDetails maintains the uptodate information a map and a reference is passed in the constructor of the SipAgent throuh which the SipAgent gets the uptodate information about the stack.

SipSnmpDetails has all the related information about the stack since its instantiated by the application, which is stored in the map.

SipAgent

SipAgent has the table of the counters which are updated by the Transceiver by calling the updateCounter method of the SipSnmpDetails which in turn calls the updateCounter method of the SipAgent

ICMP messages

The stack handles ICMP errors, by returning the appropriate status codes to the application like 403, 404, 408 for Connection Refused, Host Down and Host Unreachable respectively. This is in SipUdp_impl class.

DnsSrv Support

The stack does a DNS SRV, or a DNS lookup, in case the requestline contains a address which could not be resolved. The DNS lookup returns the available list of servers and the portnumber information. The stack does a getNextRecord and this method returns the first highest priority entry which is immediate destination of the message. DnsSrv support assumes that the retransmissions should be on all the time in the application Finally if the request is send to the the servers available in the list and there is no response available then the appropriate status code of 403, 404 or 408 is sent to the application . In case of 403 the stack does not do a DNS lookup again, since this is a server side request rejection response. In all other cases, the above steps repeat. Again, this is configurable.

SipTranceiverFilter

SipTranceiverFilter is another class which encapsulates the SipTranceiver class.This can be used to filter out duplicate msgs *received* by the stack. So, the application is guaranteed to get a single msg. This is useful, to avoid timing issues between the request/ response.

DataStructures

As mentioned above, the messages sent/received are stored in the Data Structure. For more information, see “Data Structures” on page 1-18.

SIP Transceiver

Introduction

The requirement for the VOCAL Sip Transceiver is to manage the transactions, as defined in RFC 2543, and to implement a transport independent interface for sending and receiving of SIP messages by the application layer.

The message send operation is asynchronous, while the recieve operation is blocking with a user specified timeout parameter.

in the stateful mode, transceiver maintains a soft state for a transaction as long as there is atleast one undeleted message in the transaction database. the delay for message deletion can be specified as a compile time parameter. as part of its transaction management functionality, the transceiver takes care of filtering the duplicate messages and retransmitting un-responded messages.

High Level Design

Five Components

In order to implement its functionality, five major components are implemented at the transceiver level:

- Sent Request Database: this component manages transactions for the UAC (user agent client) side of the application. In the system level message flow this takes care of the down stream interface of the application.
- Sent Response Database: this component manages transactions for the UAS (user agent server) side of the application. In the system level message flow this takes care of the upstream interface.
- UDP Transport wrapper: this component provides a SIP specific interface with the UDP transport stack. hence, it takes care of retransmissions of UDP messages to be sent and decoding of recieved messages.
- TCP Transport wrapper: this component provides a SIP specific interface with the TCP stack.
- Transaction Garbage Collector: this component takes care of deleting the transactions as they expire.

High Level Message Flow

Introduction

The high level message flow is depicted in Figure 1-2. As it can be seen, the application as a whole has two parts, viz. UAS and UAC. the SentResponseDB corresponds to the UAS portion, and all the incoming requests (and corresponding responses) go thru this component. the SentRequestDB corresponds to the UAC portion of the application and all the outgoing requests and corresponding responses go thru it. it should be noted here that SIP UA's and other endpoints may only be using any one of the two mentioned components for a particular session, while the proxy's will be using both the components.

Illustration Figure 1-2 shows the high level message flow through the SIP stack.

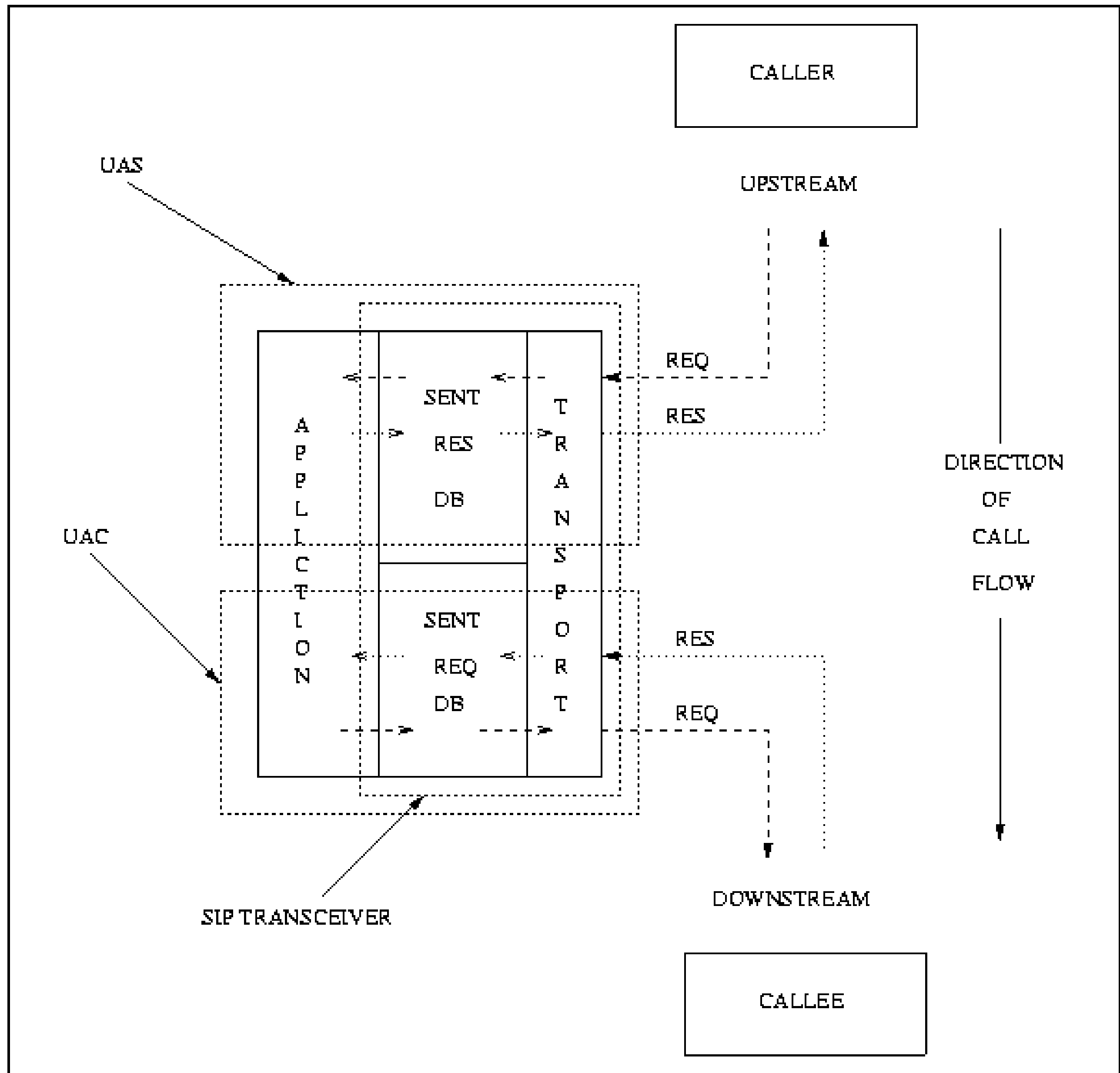


Figure 1-2. High Level Message Flow

Application Interfaces

At the transceiver level, the send and recieve of sip messages is depicted in Figure 1-3 and Figure 1-4. As can be seen in these figures, the transceiver provides three interface methods to the applications:

a) SendAsyc : this interface sends a SIP Request message asynchronously

b) SentReply: this interface sends a SIP Response asynchronously

c) Receive : this interface blocks for a user specified time (passed as method argument), and returns with the next available message from either of the two transport stacks.

Illustrations

Figure 1-3 shows the send logic flow through the SIP Stack.

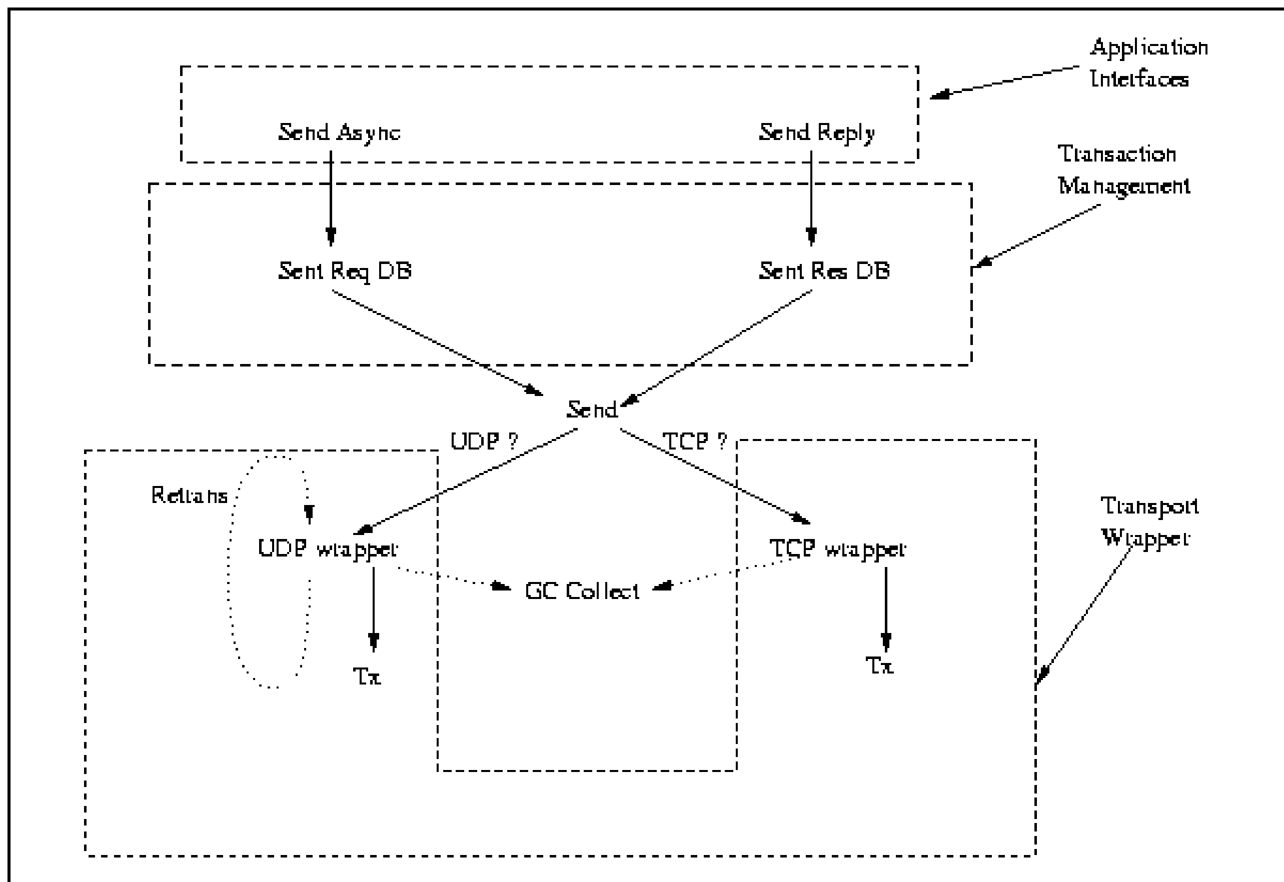


Figure 1-3. Send Logic

Figure 1-4 shows the receive logic through the SIP stack.

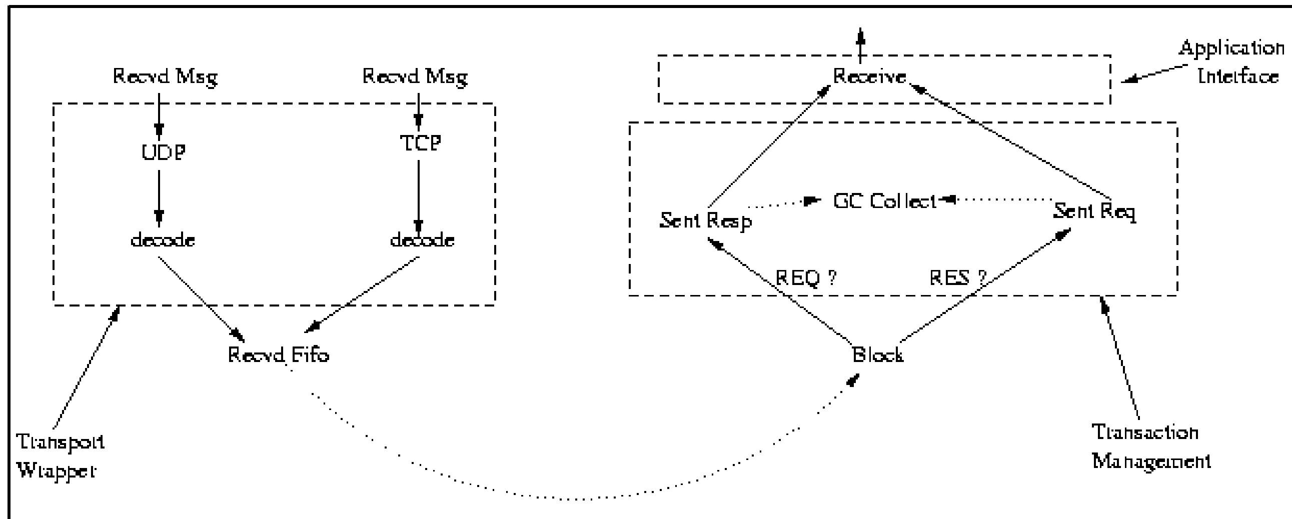


Figure 1-4. Receive Logic

Transaction Databases

Introduction

The SIP messages in the above mentioned two databases are organized based on a hierarchy, derived from the nature of the SIP transactions. these distinct, and somewhat detailed, levels of hierarchy are:

level-I

Consisting of [To, From, CallId], i.e., the call leg. this level is the root node of the whole transaction during a call.

level-II

Consisting of [CSeq#, top via branch tag] and gives a root node for every sequence/revision/forking during the call.

level-III

This level is based on [Cseq METHOD tag] and is required to distinguish between responses for INVITEs and CANCELs.

Apart from this hierarchy, the two databases also maintain some state specific information at the call leg level (i.e. with the LEVEL-I node), and they behave differently:

- the SentRespDB remebers the [To tag] of the outgoing response, to drop unmatched incoming requests (there is no need to take care of [From tag] here, since responses are copied over from requests)
- the SentReqDB remembers the [From tag] of the outgoing request to match the incoming responses, and discards the [To tag] of the incoming response.

The main functionality of these databases is to take care of filtering of duplicate messages (from UDP transport) and to implement a message ordering so that the applications can querry and access relevant information about a call’s transaction.

Data Structures

Key Structures

We have implemented some special purpose data structures, to achieve high concurrency and the above mentioned structural organization. the key data structures used are:

a) Transaction Id: this is implemented as a compound data type, having one field each for the above mentioned three levels of hierarchy.

b) Transaction Hash Table: this is a *special purpose* implementation of hash table to be used by the Transaction databases. reason for not using stl hash tables is that we don't want to lock the *whole* hash table, when doing a lookup. in this implementation, it only locks the hashed bucket which is quickly unlocked once the collision list has been processed.

c) Transaction Level Nodes: these compound data types are implemented for every level in the hierarchical organization. these structures store the information specific to the corresponding level, and also the direct references to parent node and container objects.

d) Transaction List: this is a template implementation of a doubly linked list. this implementation is required because the list items need to store direct reference to their container objects in the list.

Threads and Interaction

Five Threads

When application instantiates a Vocal Sip Transceiver, it spawns the following five threads:

- UDP reciever
- UDP sender
- TCP receiver
- TCP sender, and
- Garbage Collector

Illustration The interaction between these threads is shown in Figure 1-5.

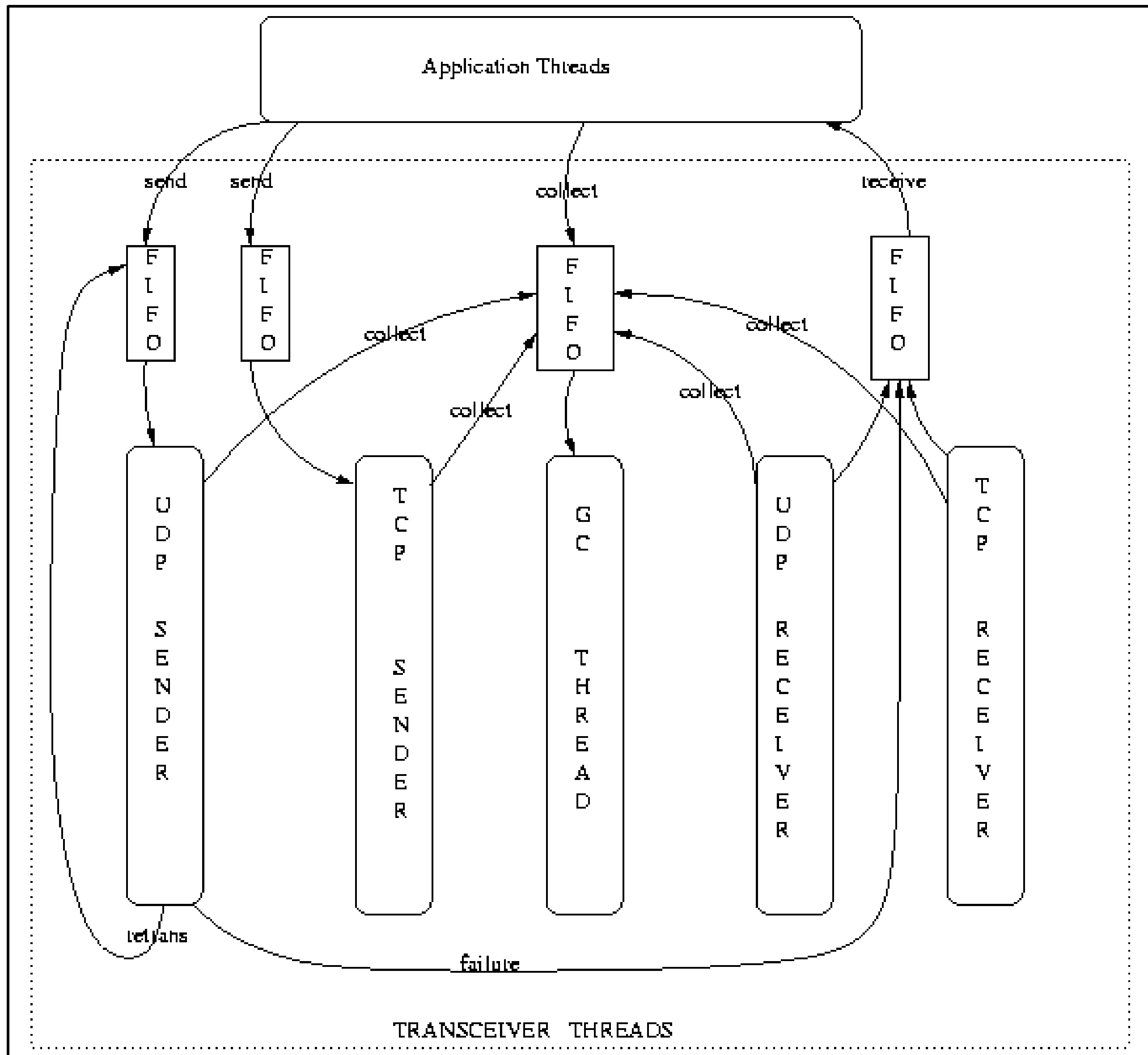


Figure 1-5. Transceiver Threads

Future Work

Introduction

The following possible improvements could be made to the current implementation:

- The transaction management can be turned on/off, making the transceiver stateful/stateless dynamically. (straight forward, minimal effort)
- The interfaces presented by the transceiver are thread-safe, hence making it possible to implement the thread-pools at the application level to handle multiple simultaneuos calls. (minimal changes in transceiver, some changes in the base code)

Bugs/Limitations

Unmatched ACK

There is a known issue of unmatched ACK if it travels a different path from the INVITE. the problem is that if it has a different top via branch, then the retrans of final responses are not cancelled, and they end up being transmitted 7 times + the penalty of resending the ACK's from the filter.

One solution is to break up the level-II key, and to skip branch matching in SentRespDB for ACKs while cancelling the retranses of the final response. but this solution will not work for a normal call flow, when there are two *forked* invites going thru the same egress marshal. will requires some more thinking!

Index