Common Alerting Protocol
(CAP)

over

Session Initiation Protocol
(SIP)

Prepared by:
Raj Gondara
MINT Capstone Project

Table of contents

1. Abstract.. 3

2. Introduction.. 3

3. SIP Overview.. 4

4. Common Alerting Protocol..12

5. Opnet..15

5.1 Preparations.. 16

5.2 Project Editor.. 16

5.3 Node Model Editor.. 17

5.4 The Process Model Editor.. 18

6. How to Create a Scenario in Opnet..19

7. SIP models in OPNET.. 21

7.1 Node Models.. 21

8. Network Model.. 26

9. Simulation Graphs.. 28

10. Modified Code.. 31

11. Future Work.. 37

12. References.. 37

1. Abstract

The Common Alerting Protocol (CAP) is an XML document format for exchanging emergency alerts and public warnings. I want to simulate CAP documents being distributed via the event notification mechanism available with the Session Initiation Protocol (SIP). I implemented this simulation in Opnet.

2. Introduction

The Common Alerting Protocol (CAP) provides an open, non-proprietary digital message format for all types of alerts and notifications. It does not address any particular application or telecommunications method. The CAP format is compatible with emerging techniques, such as Web services, as well as existing formats including the Specific Area Message Encoding (SAME) used for the United States' National Oceanic and Atmospheric Administration (NOAA) Weather Radio and the Emergency Alert System (EAS).

Session Initiation Protocol (SIP) is a signaling protocol, widely used for setting up and tearing down multimedia communication sessions such as voice and video calls over the Internet. Other feasible application examples include video conferencing, streaming multimedia distribution, instant messaging, presence information, and online games. The protocol can be used for creating, modifying, and terminating two-party (unicast) or multiparty (multicast) sessions consisting of one or several media streams.

I implemented the "common-alerting-protocol" event package simulation. It runs CAP on top of SIP to send emergency alerts. SIP uses SUBSCRIBE, PUBLISH and NOTIFY methods for sending the emergency alerts. The PUBLISH method is used to register an emergency event with notifier which sends an alert or early warning messages to clients that previously requested notification by subscribing through the SUBSCRIBE method. How clients subscribe and notifier sends alerts is described in the next paragraphs. The NOTIFY request contains one or more CAP document(s).

RFC 3265 defines a SIP extension for subscribing to remote nodes and receiving notifications of changes (events) in their states. From this SIP extension a node can subscribe to emergency notifications. RFC 3265 defines the terms Event Package, Event Template-Package, Notification, Notifier, State Agent, Subscriber, and Subscription as follows:

- An **Event Package** is an additional specification which defines a set of state information to be reported by a notifier to a subscriber. A notifier and a subscriber is defined in the next pharagraphs. Event packages also define further syntax and semantics based on the framework defined by this document required to convey such state information.

- An **Event Template-Package** is a special kind of event package which defines a set of states which may be applied to all possible event packages, including itself.

- **Notification** is the act of a notifier sending a NOTIFY message to a subscriber to inform the subscriber of the state of a resource.

- A **Notifier** is a user agent which generates NOTIFY requests for the purpose of notifying subscribers of the state of a resource. Notifiers typically also accept SUBSCRIBE requests to create subscriptions.

- A **state agent** is a notifier which publishes state information on behalf of a resource in order to do so, it may need to gather such state information from multiple sources. State agents always have complete state information for the resource for which they are creating notifications.

- A **Subscriber** is a user agent which receives NOTIFY requests from notifiers. These NOTIFY requests contain information about the state of a resource in which the subscriber is interested. Subscribers typically also generate SUBSCRIBE requests and send them to notifiers to create subscriptions.

- A **Subscription** is a set of application state associated with a dialog. This application state includes a pointer to the associated dialog, the event package name, and possibly an identification token. Event packages will define additional subscription state information. By definition, subscriptions exist in both a subscriber and a notifier.

Additionally, RFC 3903 defines an extension that allows SIP User Agents to publish event state. RFC 3903 defines the terms Event State, Event Publication Agent, Event State Compositor, Presence Compositor, and Publication as follows:

- An **Event State** is state information for a resource, associated with an event package and an address of a user.

- An **Event Publication Agent** (EPA) is the UAC that issues PUBLISH requests to publish event state. The UAC is defined in SIP overview section of this document.

- An **Event State Compositor** (ESC) is the UAS that processes PUBLISH requests, and is responsible for compositing event state into a complete, composite event state of a resource. The UAS is defined in SIP overview section of this document

- A **Presence Compositor** is a type of Event State Compositor that is responsible for compositing presence state for a presentity.

- A **Publication** is the act of an EPA sending a PUBLISH request to an ESC to publish event state.

Using these features of SIP. Event Publication Agents (EPA) will use PUBLISH requests to inform an Event State Compositor (ESC) of changes in the common-alerting-protocol event package. Acting as a notifier, the ESC notifies subscribers about emergency alerts and public warnings.

I completed the first part of the project as my capstone project, where I implemented a SIP simulation in Opnet. Opnet contains only basic SIP methods. I added the NOTIFY method to the Opnet SIP coding, and then I created an Opnet simulation of SIP notifications.

3. SIP Overview

The Session Initiation Protocol (SIP), defined by the Internet Engineering Task Force (IETF), is an application layer control (signaling) protocol for establishing, modifying, and terminating sessions with one or more participants. These sessions may be Internet telephone calls, multimedia distribution or multicast conferences. SIP has been standardized for invitation to multicast conferences and Internet telephone calls. In SIP, a user is usually identified by an email like address such as user@domain, where “user” is a user name or phone number, and “domain” is a domain name or numerical address of the user.

The main entities in SIP are the User Agent (UA), Proxy Server, Redirect Server and Registrar. They are defined as follows:

- A **User Agent** (UA) is the endpoint entity. User Agents initiate and terminate sessions by exchanging requests and responses. RFC 2543 defines the User Agent as an application, which contains both a UAC and a UAS. The UAC is client application that initiates SIP requests. The UAS is a server application that accepts the requests from a UAC and generates an accept, reject or redirect response on behalf of the user.

- A **Registrar** is a server that accepts REGISTER requests for the purpose of updating a location database with the contact information of the user specified in the request.

- A **Proxy Server** is an intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients.

- A **Redirect Server** is a server that accepts a SIP request, maps the SIP address of the called party into zero or more new addresses and returns them to the client. It returns zero addresses if there is no known address for the called party.

The message types of SIP are typically Request-Response messages, either a request from a client to a server, or a response from a server to a client. The request methods in SIP are defined in table 1. The Responses to request methods are described by a three-digit status code indicating success or failure. The response types in SIP are defined in table 2.

Table.1 SIP Request method

Message Name	Function
REGISTER	Register a user with a SIP server (with location service).
INVITE	Invite user(s) to a session. The body of the message contains the description with the address where the host wants to receive the media stream.
ReINVITE	ReINVITE is for changing the session (call) parameters.
ACK	Acknowledgment of an INVITE request.
CANCEL	Cancel a pending request.
BYE	Terminate a session (release a call).
OPTIONS	Query servers about their capabilities.
PRACK	Provisional acknowledgment.
SUBSCRIBE	Subscribes for an Event of Notification from the Notifier.
NOTIFY	Notify the subscriber of a new Event.
PUBLISH	Publishes an event to the Server.
INFO	Sends mid-session information that does not modify the session state.
REFER	Asks recipient to issue SIP request (call transfer).
MESSAGE	Transports instant messages using SIP.
UPDATE	Modifies the state of a session without changing the state of the dialog.

Table. 2 SIP Response code

Code Classes	Response Type	Function Description
1xx	Provisional	Request received, continuing to process the request.
2xx	Success	The action was successfully received, understood and accepted.
3xx	Redirection	Further action needs to be taken in order to complete the request.
4xx	Client Error	The request contains bad syntax or cannot be fulfill at this server.
5xx	Server Error	The server failed to fulfill an apparently valid request.
6xx	Global Failure	The request cannot be fulfill at any server.

The general structure of the SIP message is shown in figure 1. The SIP messages are composed of the following three parts: START LINE, HEADERS, and BODY (CONTENT). SIP messages appear both in request and in response messages. SIP makes a clear distinction between signaling information, conveyed in the SIP START LINE and HEADRS, and the session description information.

```
generic- message = start- line (= Request-Line | Status-Line )


message- header = (general-header
                  | request-header
                  | response-header
                  | entity-header)
CRLF (Carriage-Return Line-Feed: an empty line indicating the end of the header fields)


[message-body]
```

Figure 1: SIP Message Structure

Every SIP message begins with a START LINE. The START LINE conveys the message type (method type in requests, and response code in responses) and the protocol version. The START LINE may be either a Request line (requests) or a Status line (responses).The Request line includes a Request URI, which indicates the user or service to which this request is being addressed. The Status line holds the numeric Status code and its associated textual phrase.

SIP header fields are used to convey message attributes and to modify message meaning. Headers can span multiple lines. SIP headers include To, From,Via,Call-ID, CSeq, Contact, Max-Forwards, and Content-Type. RFC 3261 describe these terms as follows.

```
                     atlanta.com  . . . biloxi.com
                 .      proxy              proxy     .
               .                                       .
       Alice's  . . . . . . . . . . . . . . . . . . . .  Bob's
      softphone                                        SIP Phone
         |                |                |                |
         |    INVITE F1   |                |                |
         |--------------->|    INVITE F2   |                |
         |  100 Trying F3 |--------------->|    INVITE F4   |
         |<---------------|  100 Trying F5 |--------------->|
         |                |<-------------- | 180 Ringing F6 |
         |                | 180 Ringing F7 |<---------------|
         | 180 Ringing F8 |<---------------|     200 OK F9  |
         |<---------------|    200 OK F10  |<---------------|
         |    200 OK F11  |<---------------|                |
         |<---------------|                |                |
         |                       ACK F12                    |
         |------------------------------------------------->|
         |                   Media Session                  |
         |<================================================>|
         |                       BYE F13                    |
         |<-------------------------------------------------|
         |                     200 OK F14                   |
         |------------------------------------------------->|
         |                                                  |
```

Figure 2: Session setup example from RFC 3261

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

Figure 3: SIP header example from RFC 3261

- **Via** contains the address (pc33.atlanta.com) at which Alice is expecting to receive responses to this request. It also contains a branch parameter that identifies this transaction.

- **To** contains a display name (Bob) and a SIP or SIPS URI (sip:bob@biloxi.com) towards which the request was originally directed.

- **From** also contains a display name (Alice) and a SIP or SIPS URI (sip:alice@atlanta.com) that indicate the originator of the request. This header field also has a tag parameter containing a random string(1928301774) that was added to the URI by the softphone. It is used for identification purposes.

- **Call-ID** contains a globally unique identifier for this call, generated by the combination of a random string and the softphone's host name or IP address. The combination of the To tag, From tag, and Call-ID completely defines a peer-to-peer SIP relationship between Alice and Bob and is referred to as a dialog.

- **CSeq** or Command Sequence contains an integer and a method name. The CSeq number is incremented for each new request within a dialog and is a traditional sequence number.

- **Contact** contains a SIP or SIPS URI that represents a direct route to contact Alice, usually composed of a username at a fully qualified domain name (FQDN). While an FQDN is preferred, many end systems do not have registered domain names, so IP addresses are permitted. While the Via header field tells other elements where to send the response, the Contact header field tells other elements where to send future requests.

- **Max-Forwards** serves to limit the number of hops a request can make on the way to its destination. It consists of an integer that is decremented by one at each hop.

- **Content-Type** contains a description of the message body.

- **Content-Length** contains an octet (byte) count of the message body.

A message Body is used to describe the session to be initiated or alternatively it may be used to contain opaque textual or binary data of any type, which is related in some way to the session. Message bodies can be written as <name> : < value>

Figure 4 below shows the interaction between a User Agent Client (UAC) and a User Agent Server (UAS) during a session. For each SIP session, there is a unique call identifier (Call ID) that identifies the session. If the session needs to be modified, the same Call ID is used in the initial request, in order to indicate that this is a modification of an existing session.

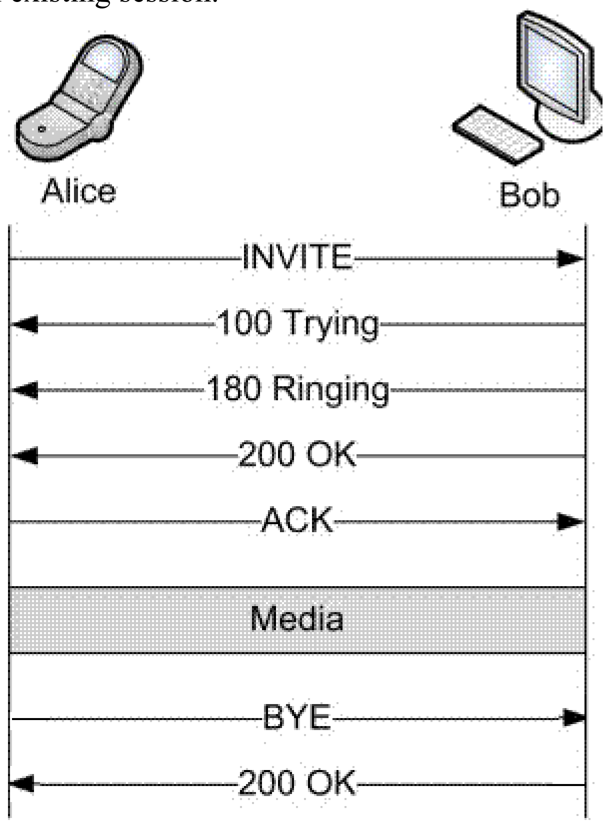

Figure 4: SIP Session Establishment and Call Termination

SESSION ESTABLISHMENT: CALL FLOW

1. The calling UAC(Alice) sends an INVITE message to Bob's SIP address: sip:bob@acme.com.
2. The UAS receives the request and immediately responds with a 100-Trying response message.
3. The UAS starts "ringing" to inform Bob of the new call. Simultaneously a 180 (Ringing) message is sent to the UAC.
4. Bob picks up the call and the UAS sends a 200 (OK) message to the calling UA.
5. The calling UAC sends an ACK request to confirm the 200 (OK) response was received.
6. Media bar represents the both way voice traffic time span.

SESSION TERMINATION: The session termination call flow proceeds as follows:

1. The caller decides to end the call and "hangs-up". This results in a BYE request being sent to Bob's UAS at SIP address sip:bob@lab.acme.com.
2. Bob's UAS responds with 200 (OK) message and notifies Bob that the conversation has ended.

SUBSCRIPTION: The Subscription and Notification call flow proceeds as follows:

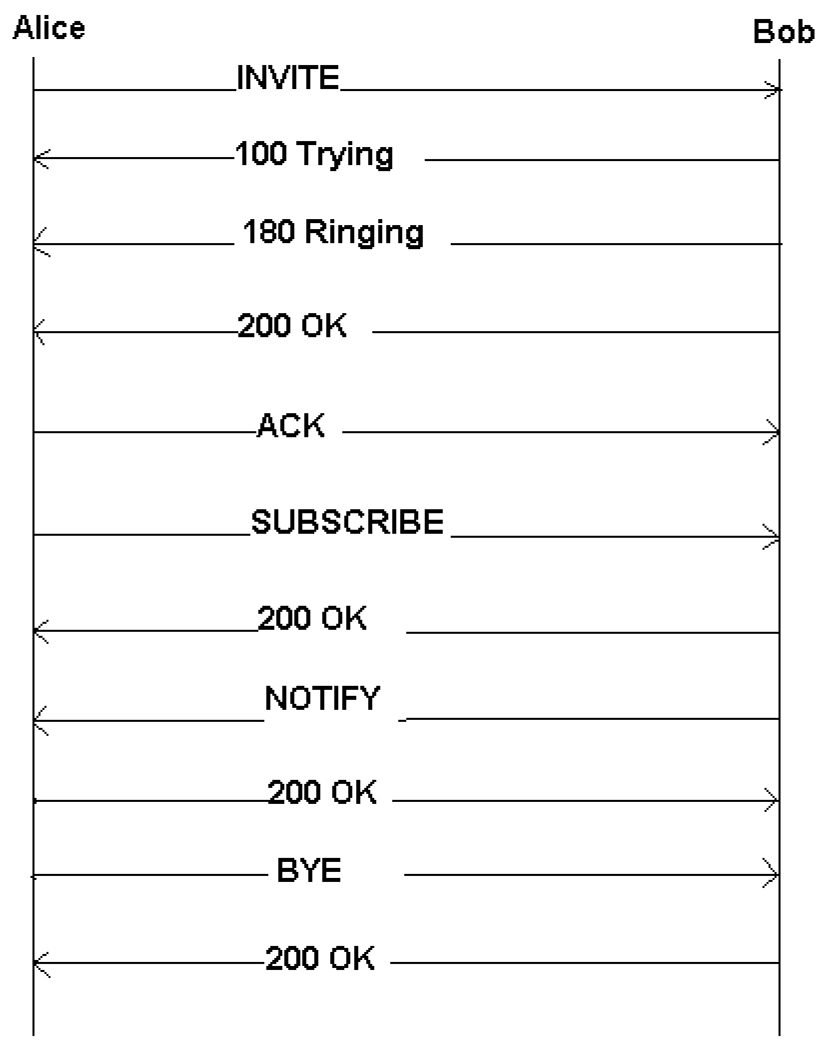

Figure 5: Subscription and Notification message flow.

1. The call establishment before subscription and call termination after notify are same as in figure 3 and figure 4.

2. Alice requests state subscription with method SUBSCRIBE and Bob acknowledge subscription with 200 OK and then sends the NOTIFY message which returns the current state information.

PUBLICATION: The Subscription-Publication-Notification call flow is as follows:

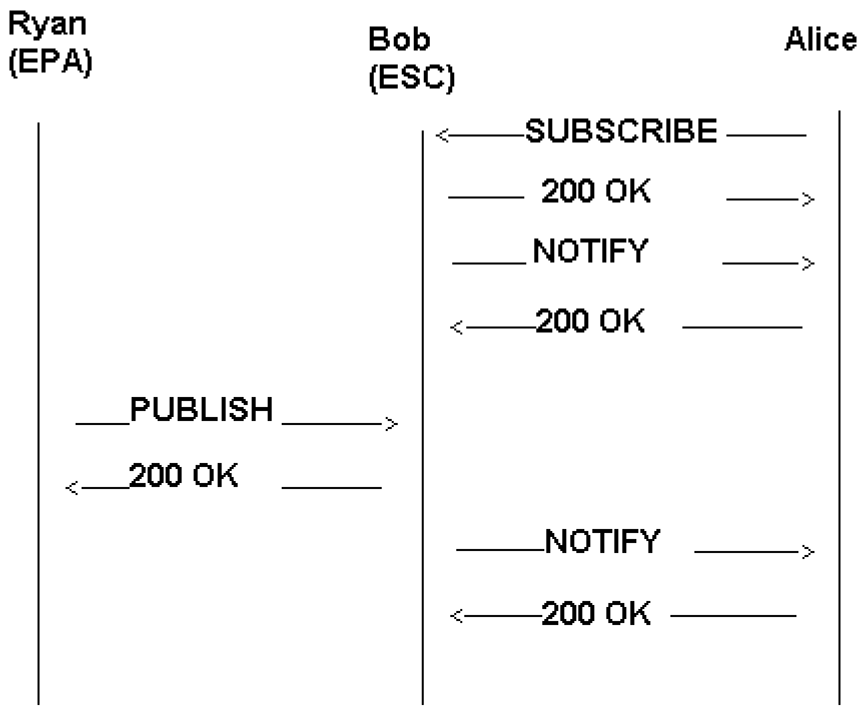

Figure 6: Subscription, Publication, and Notification call flow.

1. Alice subscribes to the Bob(ESC) in the same as described in figure in 5.

2. Bob notify the Alice for the state change with NOTIFY method after successful Publication of state change by the EPA Ryan. In an emergency alerting system the clients willing to receive the emergency alerts subscribe to the ESC and in case of an emergency published by an EPA all subscribed clients receive alerts from the ESC.

4. Common Alerting Protocol

The Common Alerting Protocol (CAP) provides an open, non-proprietary digital message format for all types of alerts and notifications. It is a Extensible Markup Language (XML) based alerting template. Rather than addressing one or few telecommunication applications, it can be used over all kinds of networks. The CAP format is compatible with emerging techniques, such as Web services, as well as existing formats including the Specific Area Message Encoding (SAME) used for the United States' National Oceanic and Atmospheric Administration (NOAA) Weather Radio and the Emergency Alert System (EAS). According to the OASIS Standard CAP-V1.1, The CAP offers following enhanced capabilities.

- Flexible geographic targeting using latitude/longitude shapes and other geospatial representations in three dimensions.
- Multilingual and multi-audience messaging.
- Phased and delayed effective times and expirations.
- Enhanced message updates and cancellation features.
- Template support for framing complete and effective warning messages.
- Compatible with digital encryption and signature capability, and facility for digital images and audio.

Being a general format for exchanging all emergency alerts and warnings CAP is very cost effective as it eliminate the need for multiple custom software interfaces to the many warning sources and dissemination systems involved in all-hazard warnings.

4.1 Structure of the CAP Alert Message

Being a XML based template each CAP Alert Message consists of an <alert> segment, which may contain one or more <info> segments, each of which may include a <resource>, and one or more <area> segments.

4.1.1 <alert>

The <alert> segment provides basic information about the current message, its purpose, its source and its status, as well as unique identifier for the current message and links to any other related messages.

4.1.2 <info>

The <info> segment describes an event in terms of its urgency, level of impact, and certainty.

4.1.3 <resource>

The <resource> segment provides the resource of the alert. It is an optional segment used inside the <info> segment.

4.1.4 <area>

The <area> segment provides the geographical area to which alert applies. It can be the postal code of a geographical shape.

I created an example template for the students of south campus University of Alberta, which alerts the students for the heavy snowfall. I include the <alert>, < info>, <resource>, and <area> segments. See figure 7.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<alert xmlns = "urn:oasis:names:tc:emergency:cap:1.1">
        <identifier>43b080713727</identifier>
        <sender>raj@ualberta.ca</sender>
        <sent>2003-04-02T14:39:01-05:00</sent>
        <status>Actual</status>
        <msgType>Alert</msgType>
        <scope>Public</scope>
        <info>
                <category>Geo</category>
                <event>Snowfall Update</event>
                <urgency>Immediate</urgency>
                <severity>Medium</severity>
                <certainty>Likely</certainty>
                <senderName>Raj Gondara</senderName>
                <description>There is a forcast of heavy snowfall in the next three days.</description>
                <instruction> All the students are adviced to drive slowly.</instruction>
                <resource>
                        <resourceDesc>Image file (GIF)</resourceDesc>
                        <uri>http://www.weatheroffice.gc.ca/city/pages/ab-50_metric_e.html</uri>
                </resource>
                <area>
                        <areaDesc>South campus Univeristy of Alberta.</areaDesc>
                </area>
        </info>
</alert>
```

Figure 7: Example template for CAP message.

It is an xml version 1.0 template with a unique namespace "urn:oasis:names:tc:emergency:cap:1.1" defined by xmlns attribute. The segments <identifier>, <sender>, <sent>, <status>. <msgType>, and scope are about description of the alert and tells unique identifier,sender, time and date, status, type of alert and scope of the alert respectively. The <info> segments actually tells the nature of alert and instructions to follow along with level of urgency and certainty. The <resource> segment describes that resource of the alert is an image referred by website "http://www.weatheroffice.gc.ca/city/pages/ab-50_metric_e.html". The <area> segments describes that this alert is for the South campus University students.

5. OPNET

OPNET Modeler is the industry's leading simulator specialized for network research and development. It allows to design and study communication networks, devices, protocols, and applications with great flexibility. It provides a graphical editor interface to build models for various network entities from physical layer modulator to application processes. All the components are modeled in an object-oriented approach which gives intuitive easy mapping to real systems. It gives a flexible platform to test new ideas and solutions with low cost. It comes with the following toolsets.

- node model that specifies interface of a network component.
- packet format defines protocols.
- process model that abstracts the behavior of a network component.
- project window that defines network topology and link connections simulation window that captures and displays simulation results.

5.1 Preparations

The Project Editor is used to create network models, collect statistics directly from each network object or from the network as a hole, execute a simulation and view results. As in figure 19, I created a network model with two workstations, a router, and a proxy server. I chose the SIP traffic related statistics for workstations and proxy server, and I described the SIP traffic statistics selection, running the simulation, and viewing and analyzing the results under section 6. See Workflow Figure 8.

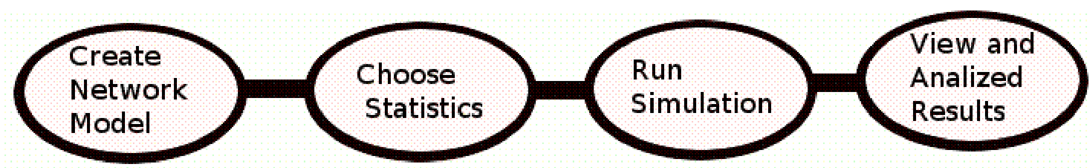

Figure 8: Workflow

5.2 Project Editor

The main staging area for creating a network simulation is the Project Editor. We start with project editor to create a network model from the File – New link. We use models from the standard library, collect statistics about the network, run the simulation and view the results. The figure 8 describes the workflow of an Opnet simulation . I describes the details of using the complete workflow under section 8 where I mentioned the details on how I built my network model.

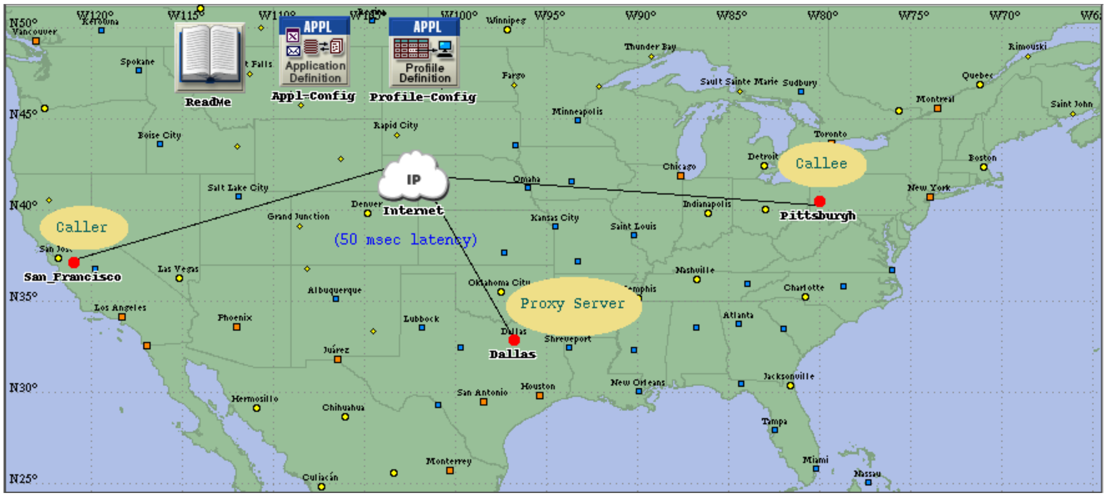

Figure 9: A network model built in the Project Editor

Depending on the type of network being modeled, a network model may consist of subnetworks and nodes connected by point-to-point, bus, or radio links. Figure 9 is my network model created by using SIP technologies.

5.3 The Node Editor

The Node Editor is the tool used to create models of nodes see figure 10. The OPNET node models have a modular structure. By looking at the figure 10 we can say it is a tree structure of nodes where each node is representing level of information flow in a network object. A node is defined by connecting various modules with packet streams and statistic wires. The connections between modules allow packets and status information to be exchanged between modules. The module types includes processors, generators, queues, processors, transmitters, and receivers.

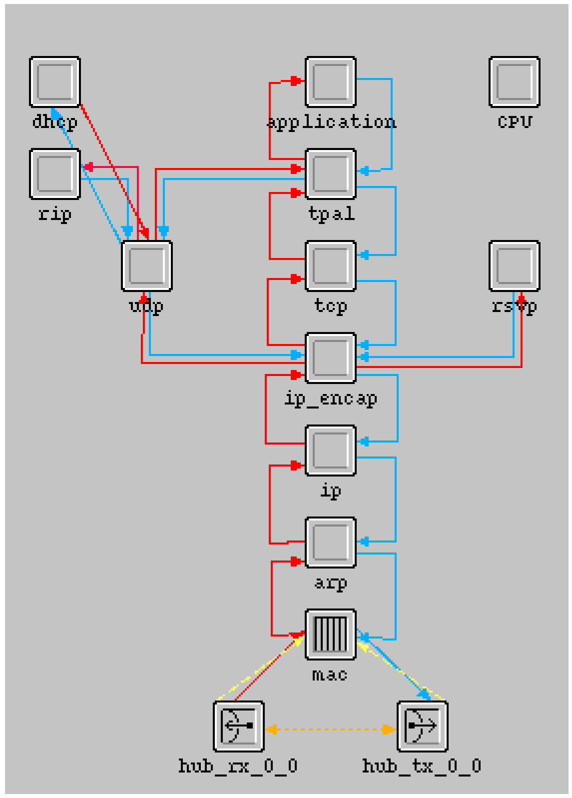

Figure 10: Node Editor

5.4 The Process Model Editor

The Process Model describes in the underlying function of a node model. The Process models are represented by finite state machines (FSMs) and are created with icons that represent states and lines that represent transitions between states. Operations performed in each state or for a transition are described in embedded C or C++ code blocks. As in figure 11, a Process model consist of States, Events, and Actions. The States are the blocking points for a process or It is a reaction to a particular event and they are represented by red and green circles, the difference in the green and red states is that red state represents the evolution of process model to the network, The flow never stops at a green state an event flow always stops at red state and transition always start from red state and comes back to red state. A transition from one state to another state happens with an action which is the label on the links and these action happens when a event occurs.

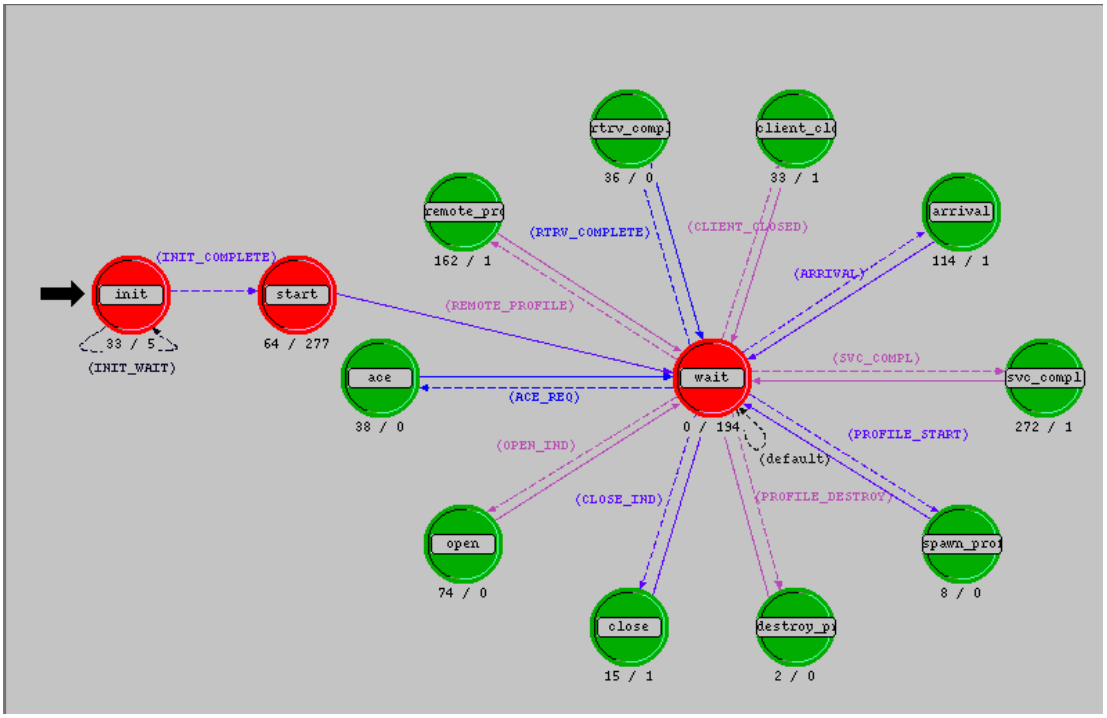

Figure 11: Process Model Editor

6. How to Create a Scenario in Opnet.

Figure 12 describe all the stages to create a scenario in Opnet modeler.

1. Start the Opnet modeler and from File menu double click on New to start a new simulation scenario.
2. Select project from the drop down menu of Create a new project screen and give a name to the project and to the scenario.
3. Select empty scenario from the Initial topology screen.
4. Choose network scale from World, Campus, Office, Enterprise, and Logical depending on your scenario.
5. Select the technology from the Technology wizard as in my project I enabled SIP.
6. Choose the objects from the Object Palatte as in my simulation I have two Workstations, a Router, a Proxy Server, and Links among each other.

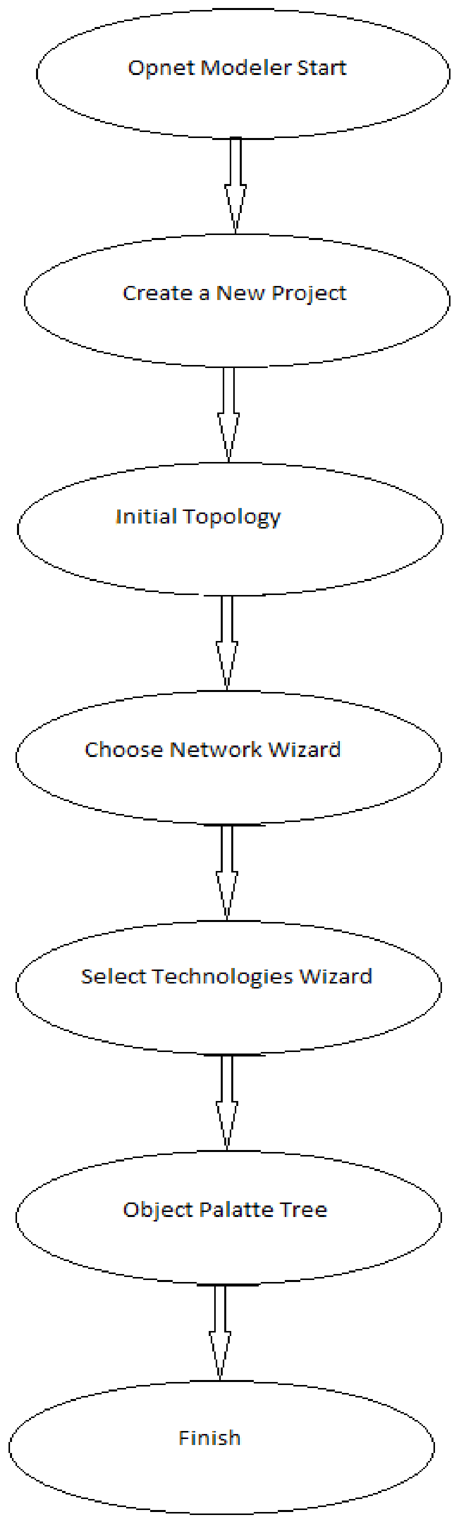

Figure 12: How to Create a Simulation Scenario in Opnet.

7. SIP models in OPNET

7.1 Node Models

In my project, I have a caller workstation and a callee workstation which are communicating through the sip proxy server the node models of a workstation and proxy server are figure 13 and figure 14.

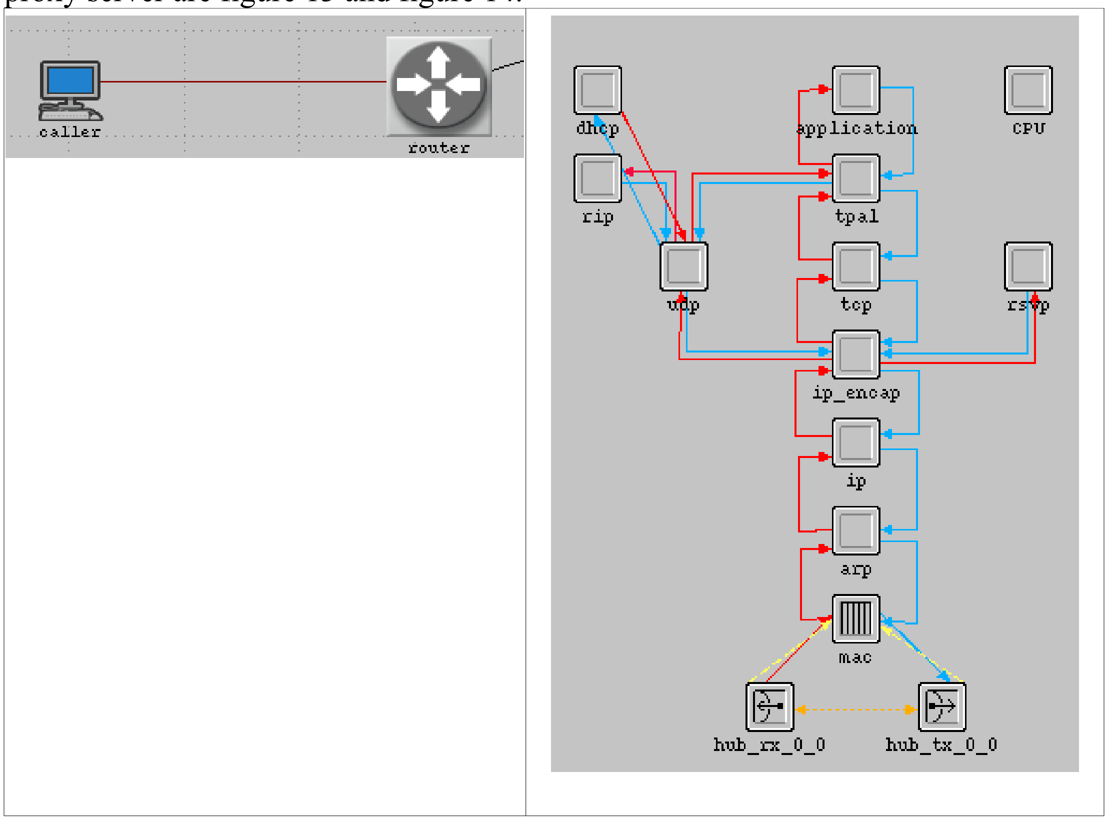

Figure 13: Workstation

Figure 13 has modules staring from the physical layer to application layer. The modules application, tcp, ip_encap, ip, arp, mac, rip, rsvp, udp,and dhcp provides the functionality of initiating voice call, end-to-end connectivity for communication, IP tunneling for the reason of connecting two disjoint IP networks, IP address to packets, converts the IP address to MAC and vice-versa, assigns the mac address, dynamic routing protocol, resource reservation, connectionless transportation of data, and automatic IP address configuration respectively. The TPAL module is not very familiar, Details of it are as follows:

- **TPAL(Transport adaptation layer)** : In the OSI model, the transport layer provides end-to-end, reliable transmission over a potentially unreliable network. Most networking protocol families include some sort of transport protocol. There are a number of different transport layers that might be modeled in Modeler. Each has its own interface description (sometimes more than one) and features not found in other transport protocols. TPAL presents a basic, uniform interface between applications and transport layer models. All interactions with a remote application through TPAL is organized into sessions. A session is a single conversation between two applications through a transport protocol. (For some transport protocols, a server application may actually receive data from more than one remote host during a single session.)

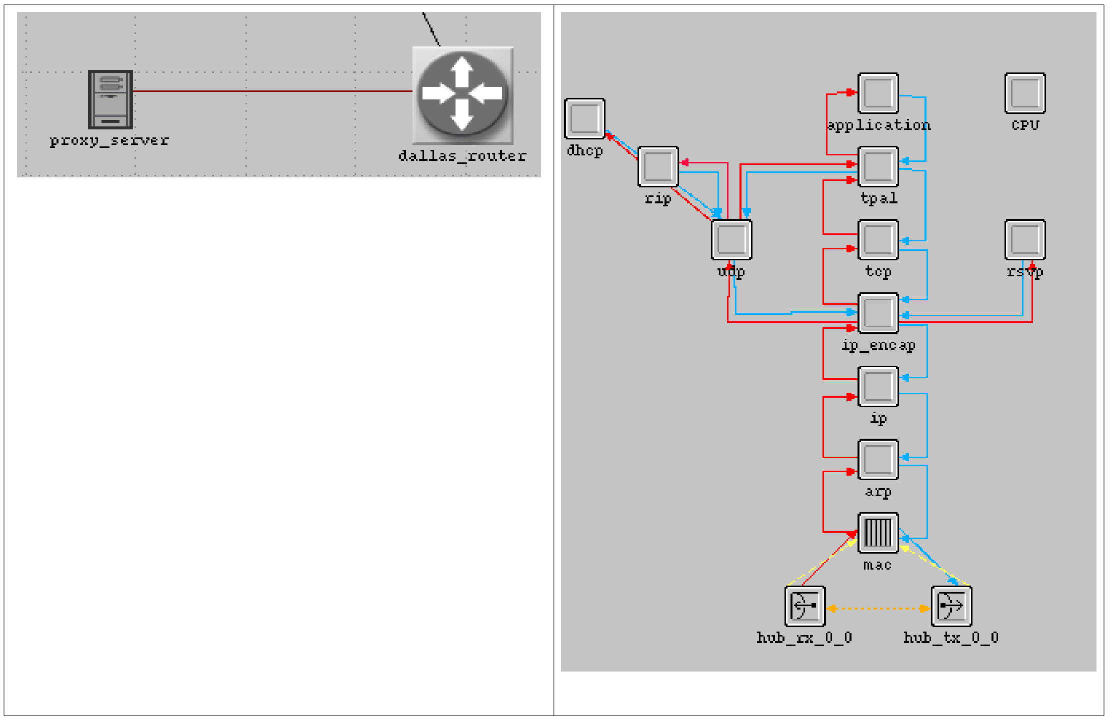

Figure 14: Proxy server

OPNET has contributed necessary node models, which have been described in the previous sections.

Besides all this, we used SIP related process models and some process models concerned with voice application that uses SIP as its signaling protocol. A list of these process models is given next and we will give a brief introduction to them.

gna_clsvr_mgr: Process model for the application module of the WLAN workstation. It is used to manage the generic network application defined in Application Configure, initiating corresponding network application on this workstation node. See section 6 for Application configure details and figure 15 for details of process model.

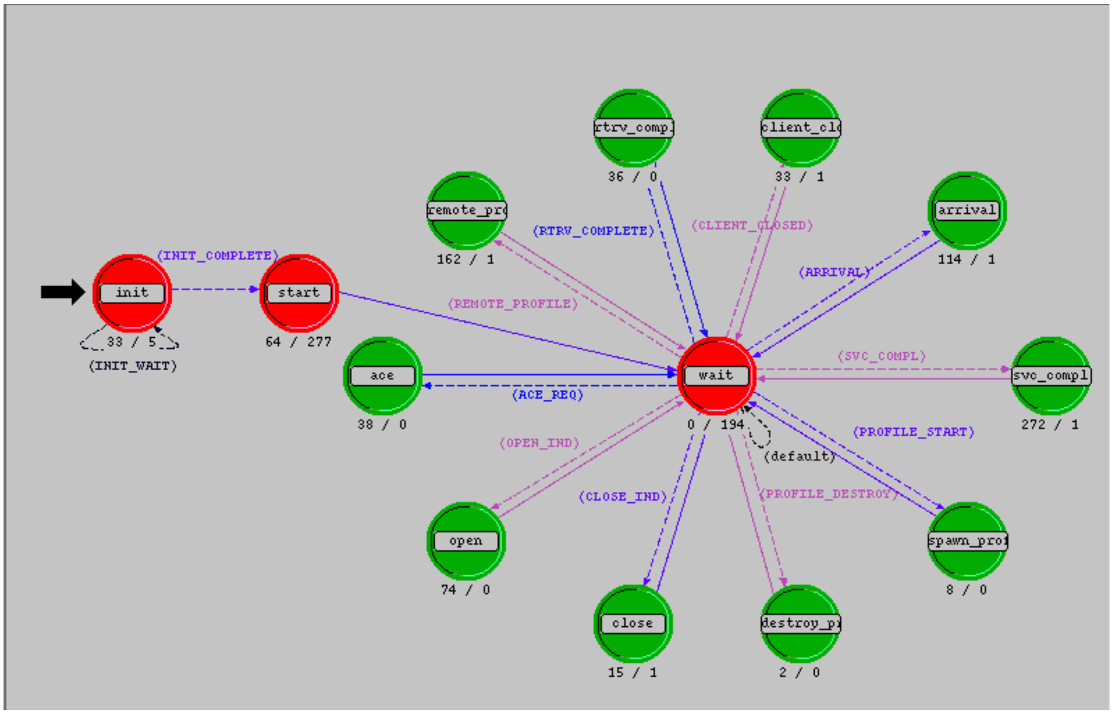

Figure 15: gna_clsvr_mgr

gna_profile_mgr: Process model that is created by gna_clsvr_mgr to manage a profile for each client. See figure 16.

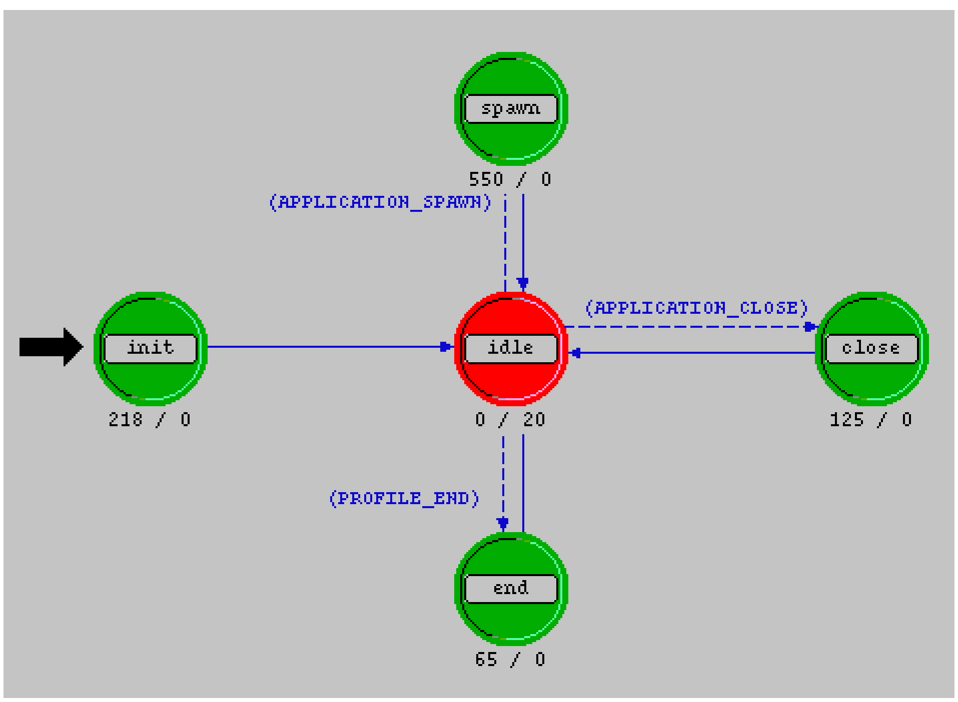

Figure 16:gna_profile_mgr

gna_voice_calling_mgr: Process model that is specifically related to voice application. It is created by gna_profile_mgr to deal with the voice application generated by gna_clsvr_mgr. See figure 17.

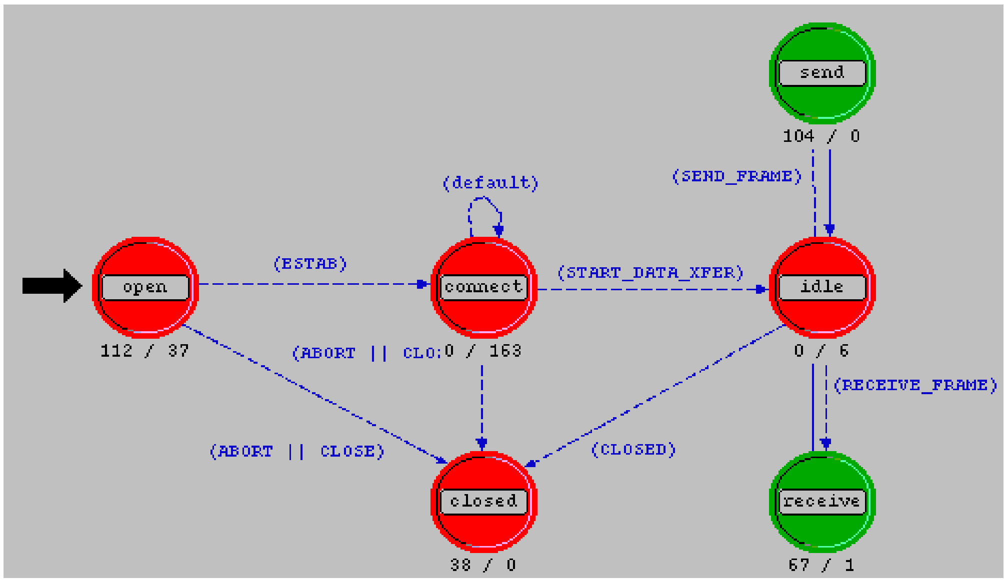

Figure 17: gna_voice_calling_mgr

sip_UAC_mgr: process model that is called by gna_voice_calling_mgr when it wants to setup a voice session. A sip_UAC_mgr usually holds control over several sip_UACs and call status that is mapped to a specific sip_UAC. See figure 18.

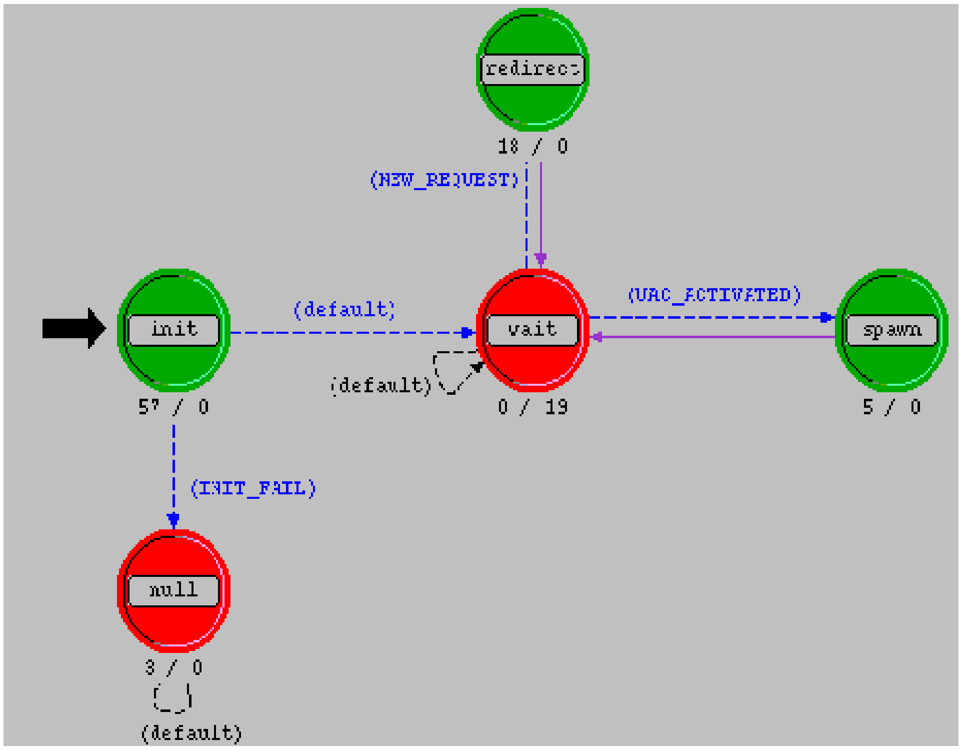

Figure 18:sip_UAC_mgr

sip_UAC: Process models that are created by sip_UAC_mgr to setup a session between a client and a server. See figure 19.

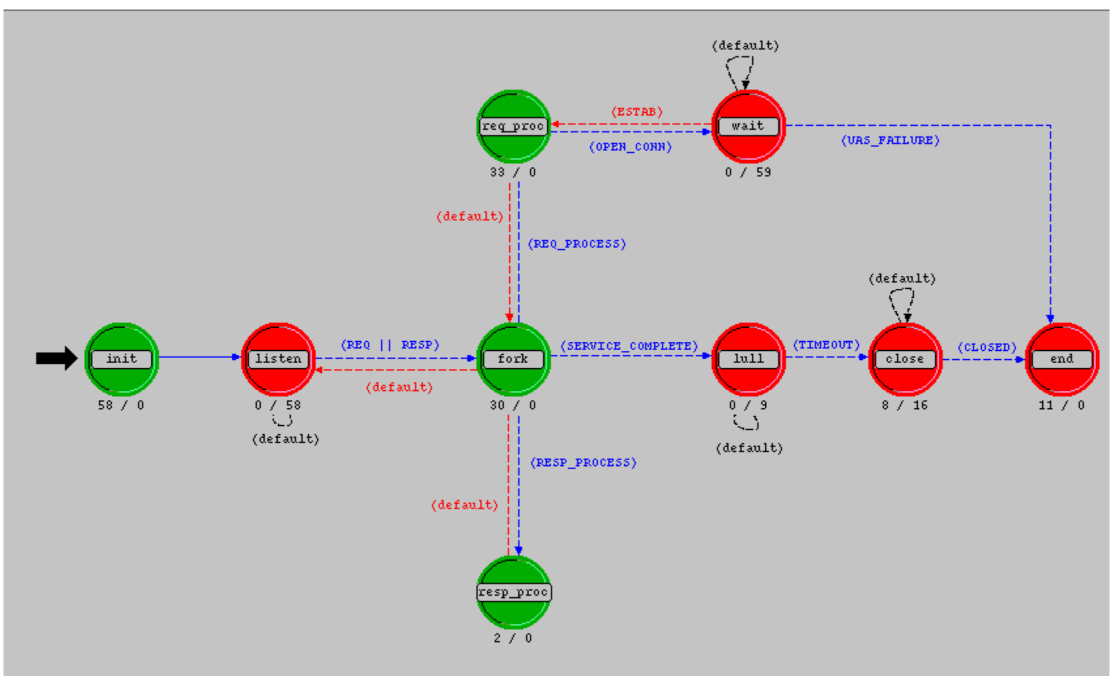

Figure 19: sip_UAC

We can clearly see the hierarchy relationship in the order of our introduction. Note that all these process models are located within application layer, from the top to the bottom, which is shown in Figure 19 below.

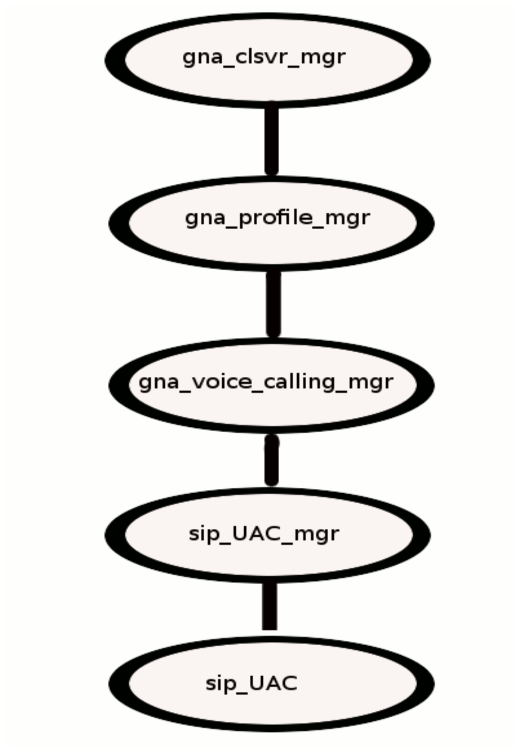

Figure 19 - Hierarchy relationship within application layer

8. Network Model

From Figure19 , we may note that *gna_voice_calling_mgr* is the process model that starts to directly interact with SIP process model. In fact, it keeps a state variable which points to a specific *sip_UAC_mgr*. When this voice-calling manager wants to setup a session with the remote host, it will call the function *sip_request_invite( )*, trying to open and maintain an active connection with the remote side. Using all these models, I built up a network topology to implement the basic SIP-based voice call setup and terminating procedure, which is shown in Figure 20.

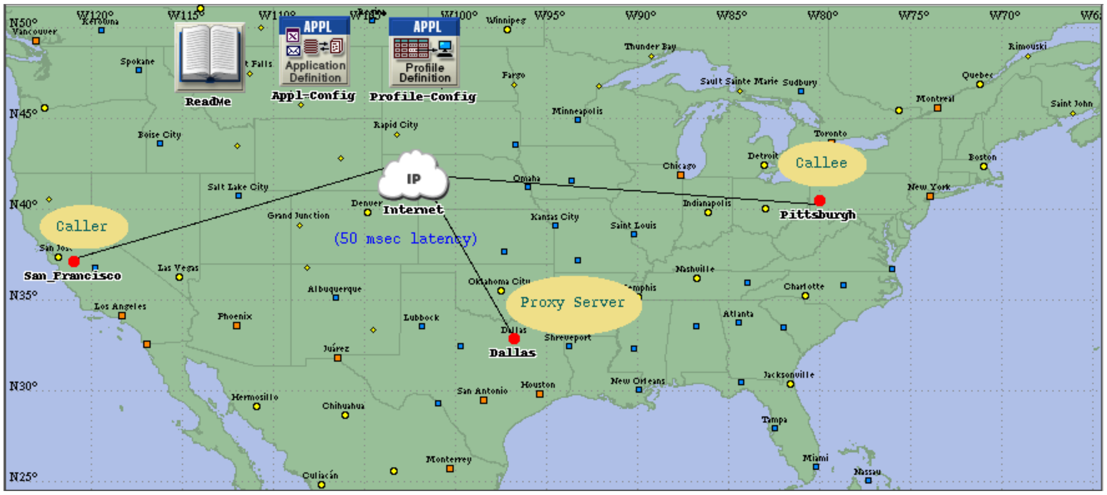

Figure 20: Network topology

Caller workstation is the caller at San Francisco. and it is calling to callee at Pitisburg and the session set ups are happening through the proxy server which is at Dallas. Application configuration tells Opnet which applications we are going to be use and Profile configuration is used to create the all needed type of traffic. The Details of the each object that I set through the edit attributes property are as follows.

Application Configuration: The Attributes I set are below:

Name	:	Appl-Config
Description	:	Default other avialable traffic types are of Database, E-mail, Http, Ftp, Video Conferencing, Print remote Host, and Voice.
Signaling	:	SIP.

Profile Configuration: The attribute details are below:

Name	:	Profile-Config.
Applications	:	Appl-Config

Proxy Server: The attribute details are below:

Name	:	Proxy Server.
Application Supported Services	:	Appl-Config.
Proxy Services	:	Enabled.

Caller: I attributes set are below:

Name	:	Caller.
Application Destination Reference	:	Appl-Config.
Application Supported Profiles	:	Profile-Config.

SIP Service	:	Proxy Server.

Callee: The attributes set are:

Name	:	Callee
Application Destination Reference	:	Appl-Config.
SIP Service	:	Proxy Server.

9. Simulation Graphs

Following are the screens for simulation results for caller, callee and proxy server.

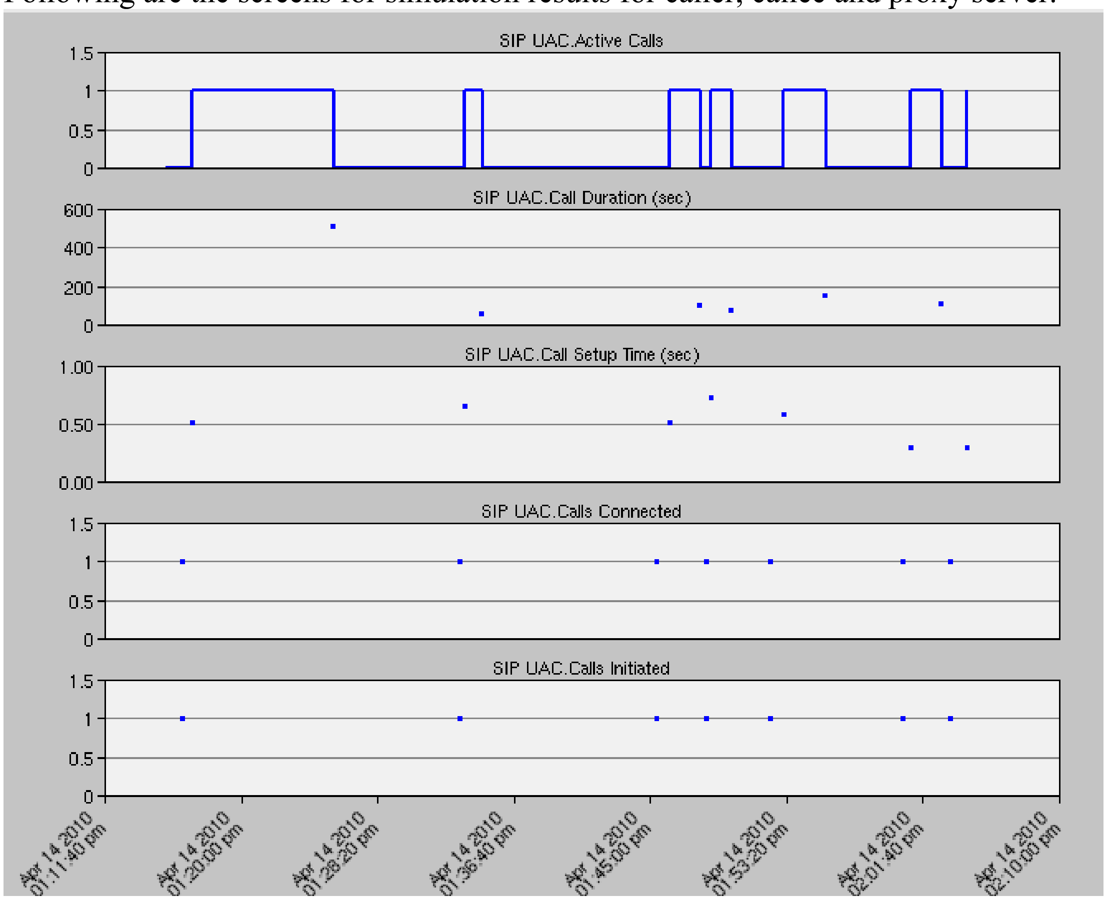

Figure 21: Statistics from Caller

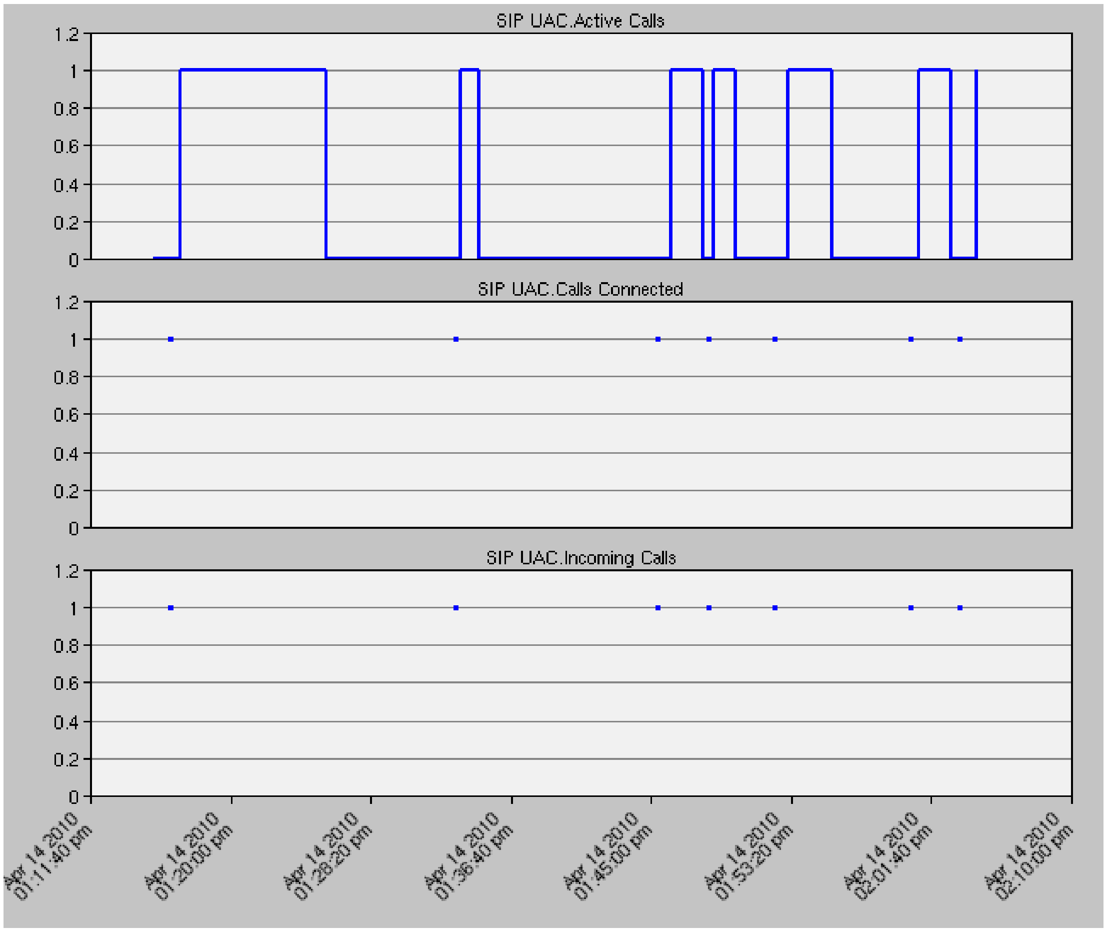

Figure 22. Statistics from Callee.

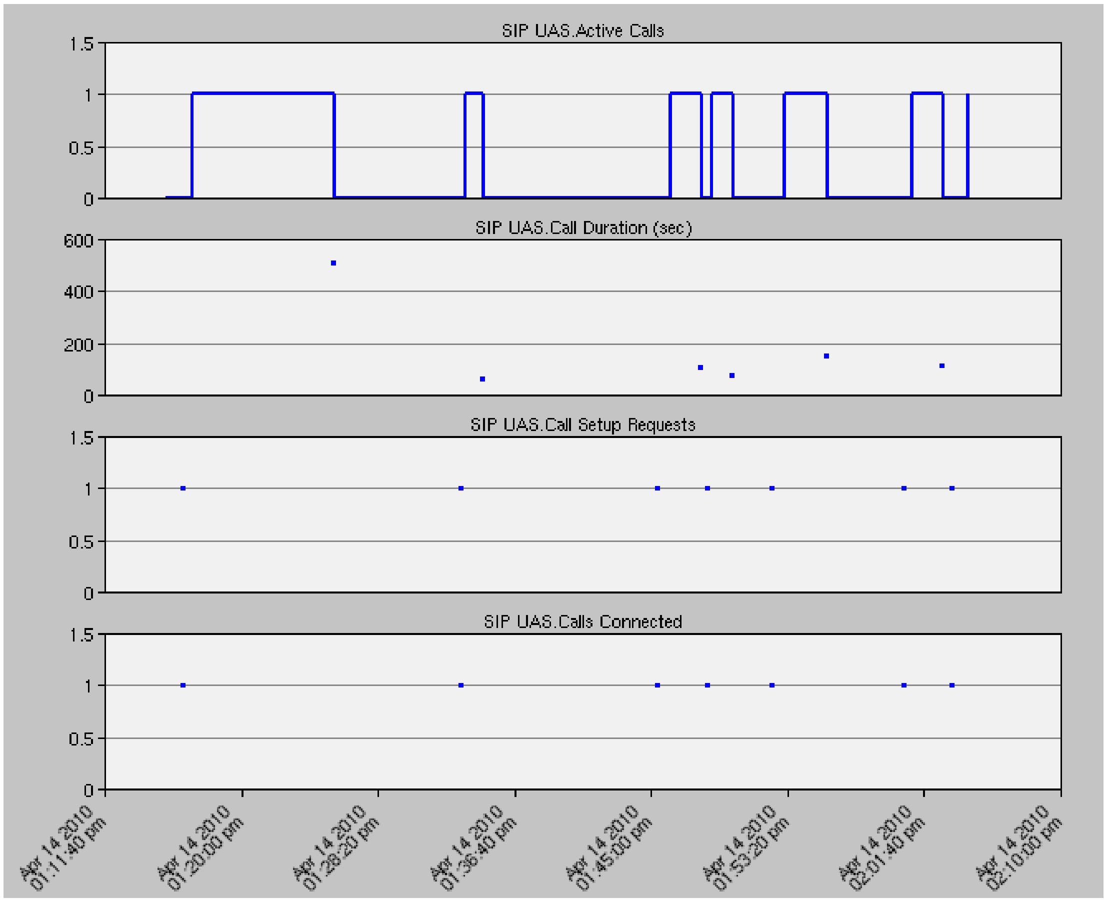

Figure 23: Statistics from Proxy Server.

From figures 21,22, and 23 we can see all the call initiated by the Caller, how Proxy Server is involved for the time of call set up and Callee is involved from the point of call connected. Though from the graphs I can not show when notify method is called but as here I made it a part of the successful call set up so whenever a call sets up the notify method is called. Figure 24 show how notify method is called on the successful call set up. Figure 24 is a screen shot of console output that I get from the Run Simulation screen. I injected the printf output lines in the code to track the flow of the code, It helped me to understand the flow of the code and this way I find the spots to add my Notify method and call this method. The line `Here in the notify method` as highlighted in the figure 24 show that flow is inside the Notify method.

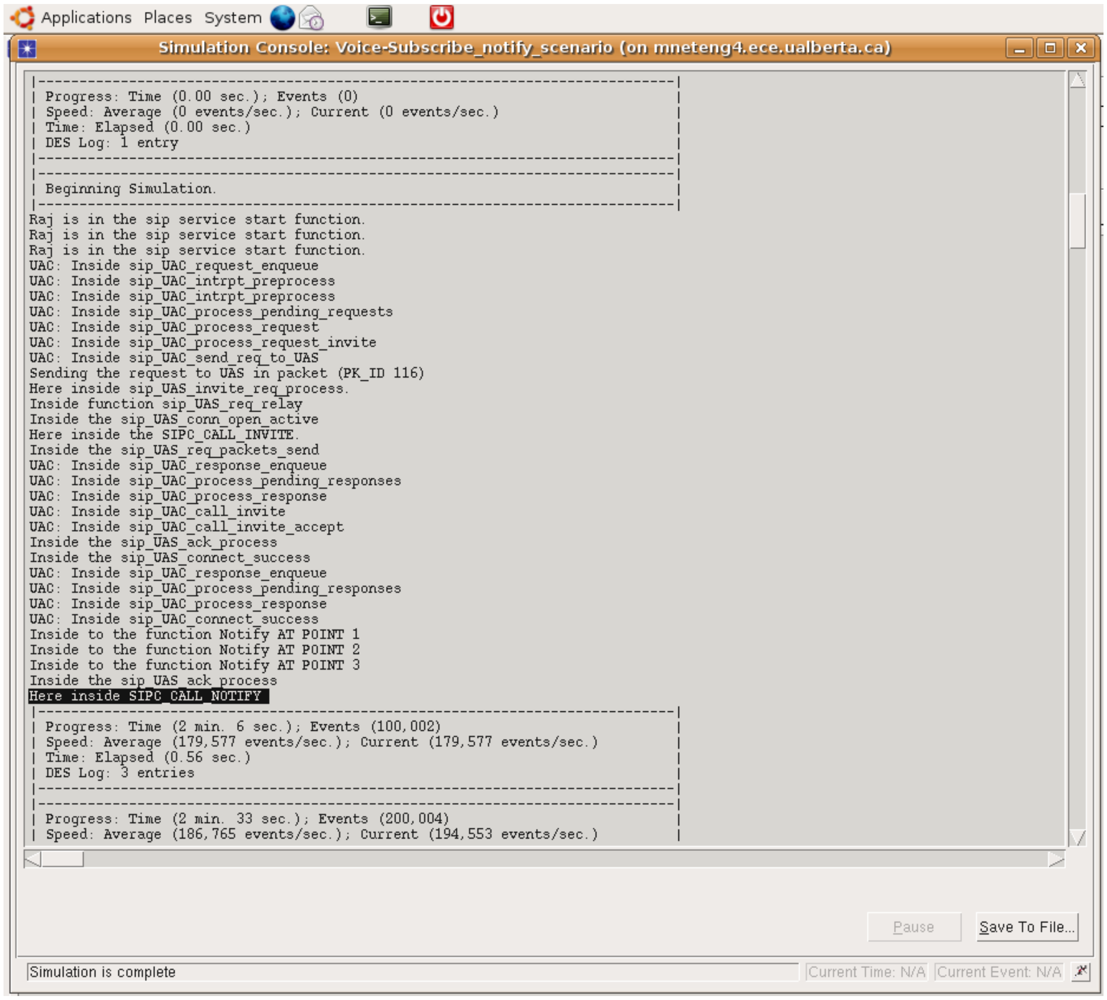

Figure 24: Console Output.

10. Modified Code

I modified the code as I have to introduce the NOTIFY method which can notify the users about various kinds of emergency events. In this scenario I added the Notify method and this method is called every time a new session is set up. I updated the sip_api.h through File-Open links, as highlighted in Figure 25. I selected the header files type from the File type drop down and open sip_api.h.

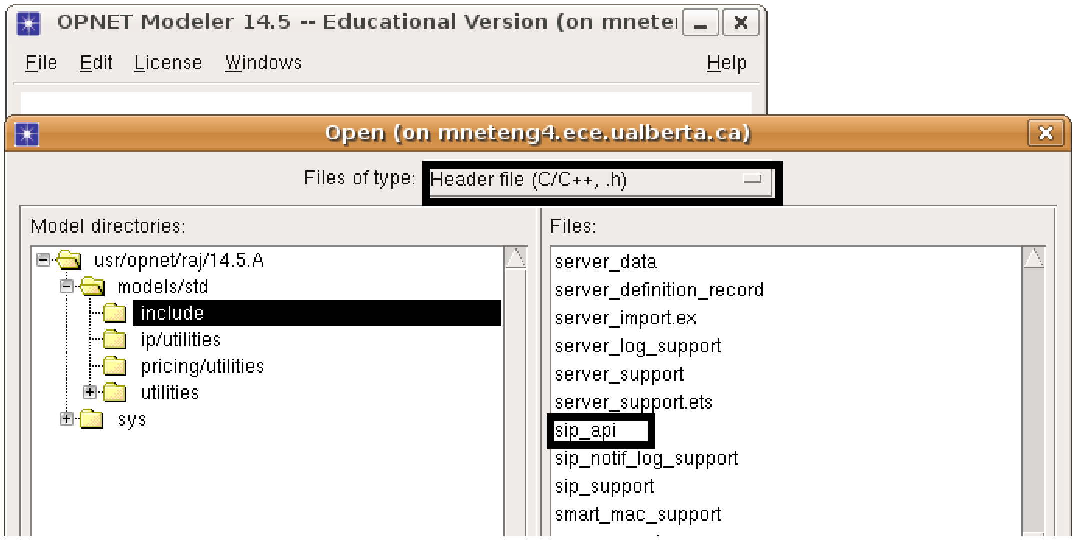

Figure 25: Editing files in opnet.

Modified sip_api.h is as follows:

```
/* sip_api.h: Definitions and declarations for the SIP model */

/****************************************/
/*                Copyright (c) 1986-2008                    */
/*          by OPNET Technologies, Inc.            */
/*                (A Delaware Corporation)         */
/*        3400 International Drive,  N.W.          */
/*                Washington, D.C., U.S.A.         */
/*                  All Rights Reserved.                     */
/****************************************/

#ifndef _sip_api_h_
#define _sip_api_h_

/***** Include files *****/
#include <opnet.h>
#include <oms_qm.h>
#include <sip_support.h>

#if defined (__cplusplus)
extern "C" {
#endif
```

10. /***** Symbolic Constants and Macros ****/

11. /** SIP Response Codes **/

12. /* SIP communicates with the Application Process that is using its services */
13. /* by using Process Interrupts (i.e. intrpt_type = OPC_INTRPT_PROCESS) and */
14. /* one of the following interrupt codes: */
15. #define SIPC_CALL_INVITE 100 /* Remote Application Process wants to Connect a Call */
16. #define SIPC_CALL_BYE 101 /* Remote Application Process wants to Disconnect the Call */
17. #define SIPC_CALL_CONNECT_SUCCESS 200 /* Call has been Connected */
18. #define SIPC_CALL_CONNECT_FAIL 400 /* Call could not be Connected */
19. #define SIPC_CALL_DISCONNECT_SUCCESS 201 /* Call has been Disconnected */
20. #define SIPC_CALL_DISCONNECT_FAIL 401 /* Call could not be Disconnected */
21. **#define SIPC_CALL_NOTIFY 202 /* Raj Added it**

I added a new macro SIPC_CALL_NOTIFY 202 to sip_api.h file on line 21. Reason for creating this macro with number 202 is to keep it under the SIPC_CALL_CONNECT_SUCCESS category and notification can occur only after establishing a successful sip session.

Next I modified the SIP_UAC.c file. As I did modifications for my scenario, so I went to function block of the SIP_UAC process model as highlighted in the figure 26. I went to the function block from the process model screen. The way I went to SIP_UAC process model is by selecting the SIP_UAC for the application module of the workstation node model. After doing my modifications to the code I compile code to reflect the changes in the simulation results.

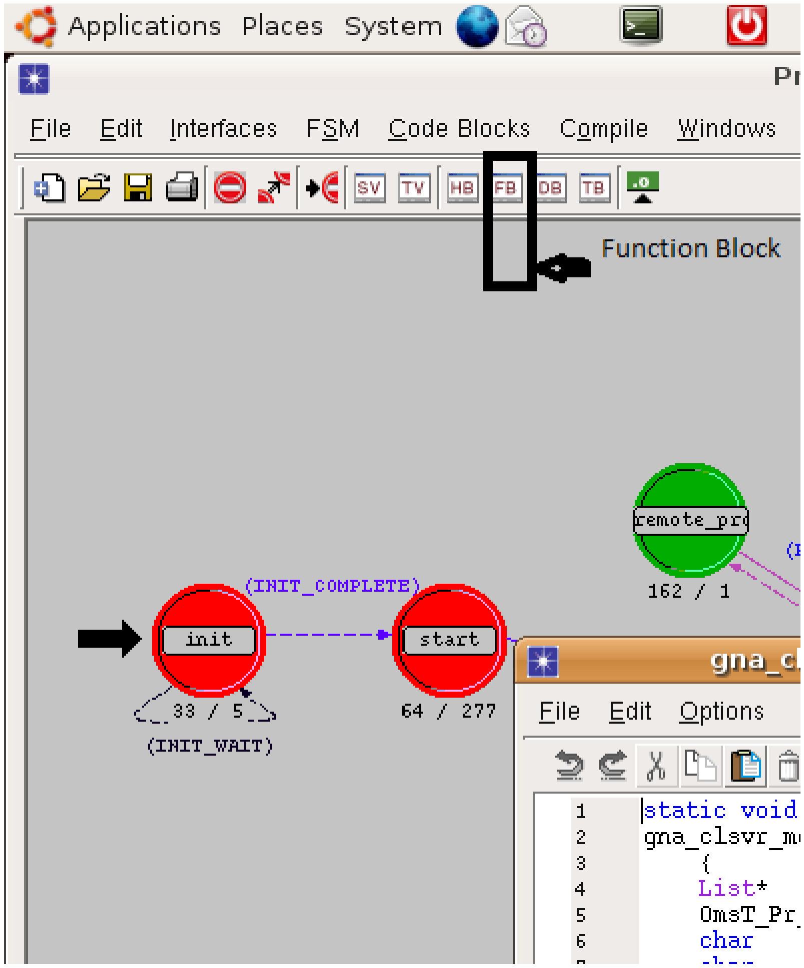

Figure 26: Function Block.

I added a new function notify and call this function on line 57 from inside function sip_UAC_connect_success. Function sip_UAC_connect_success is executed on a successful call set up, So a notification will occur when a caller set up a session with the callee. Also I added line 9 to print 'UAC: Inside sip_UAC_connect_success' in output console when function sip_UAC_connect_success is being executed.

```
1.  static void
2.  sip_UAC_connect_success (Packet* sip_resp_pk_ptr)
3.          {
4.          char                            msg1 [128];
5.          char                            msg2 [128];
6.          double                          call_init_time;
7.          SIPT_Call_Info_Shell*    sip_call_info_shell_ptr;
8.
9.          printf("%s \n", "UAC: Inside sip_UAC_connect_success"); /* Raj
```

```
added this line */

	/* This function processes the SIPC_CALL_CONNECT_SUCCESS message */
	/* received from the UAS response							*/
	FIN (sip_UAC_connect_success (sip_resp_pk_ptr));

	/* Get the information of the call associated with this packet */
	op_pk_nfd_access (sip_resp_pk_ptr, "call_info", &sip_call_info_shell_ptr);

	/* Reset the status of the call */
	sip_call_info_shell_ptr->call_status = SIPC_Call_Connected;
	UAC_call_status = SIPC_Call_Connected;

	/* Stamp the time at which this call got connected */
	sip_call_info_shell_ptr->sip_call_info_ptr->UAC_call_connect_time = op_sim_time ();

	if (ltrace_sip_UAC_active || ltrace_sip_UAC_resp_active ) /* || ltrace_sip_UAC_invite was included as third argument in this line*/
		{
		sprintf(msg1, "UAC (PID %d) processing SIP SIPC_CALL_CONNECT_SUCCESS Response", my_pro_id);
		sprintf(msg2, "Forwarding the response to Application Process (PID %d)", op_pro_id (*appl_prohndl_ptr));
		op_prg_odb_print_major(msg1, msg2, OPC_NIL);
		}

	/* Inform the caller that the call is connected */
	op_intrpt_schedule_process (*appl_prohndl_ptr, op_sim_time (), SIPC_CALL_CONNECT_SUCCESS);

	/* Get the time the request was initiatesip_UAC_request_enqueued */
	call_init_time = sip_call_info_shell_ptr->sip_call_info_ptr->call_init_time;

	/* Write out the statistic for Call Setup Time */
	op_stat_write (UAC_ptc_mem_ptr->local_call_setup_time_stathandle, op_sim_time() - call_init_time);
	op_stat_write (UAC_ptc_mem_ptr->global_call_setup_time_stathandle, op_sim_time() - call_init_time);

```

```
/* Decrement the counter for expected responses */
expected_resp_count--;

/* Increment the number of active calls */
(*(UAC_ptc_mem_ptr->active_calls_count_ptr))++;

/* Write out the calls connected statistic */
op_stat_write ( UAC_ptc_mem_ptr->active_calls_stathandle, *(UAC_ptc_mem_ptr->active_calls_count_ptr));

/* Write out the calls connected statistic */
op_stat_write ( UAC_ptc_mem_ptr->calls_connected_stathandle, 1);

/* Destroy the packet */
op_pk_destroy (sip_resp_pk_ptr);

notify(); /* Calling the notify*/

FOUT;
} /* End sip_UAC_connect_success () */
```

The notify method I added is below.

```
/* Code added by Raj */

static void
notify ()
{

Packet*     sip_pk_ptr;

/* Create a SIP packet to communicate the request to the UAS */
sip_pk_ptr = op_pk_create_fmt ("sip");

/* This function processes the SIPC_NOTIFY */
FIN (notify ());

/* Set the packet msg_index field to indicate Notify */
op_pk_nfd_set (sip_pk_ptr, "msg", SIPC_CALL_NOTIFY);

op_pk_nfd_set (sip_pk_ptr, "type", SIPC_Packet_Type_ACK);


/* Send the packet to the UAS */
```

```
22.     op_pk_send (sip_pk_ptr, strm_to_tpal);
23.
24.     FOUT;
25.
26.     }
```

On line 7 a new packet pointer is created to create a sip packet which is being created on line 10 by function op_pk_create_fmt. Line 13 FIN(notify()) is a Macro used to label a function it ia used by OPNET to populate function call stack, It is placed after local variables in C. On line 15, I set the message index to SIPC_CALL_NOTIFY, which I set to a value 202 in sip_api.h. On line 18 I set the type of the message as acknowledgment as It is going to the Caller. On line 21 send the packet. Line 24 is FOUT which is end of the function. In the future work there will be CAP message template similar to the CAP template in the figure 7 and line 22 will send that message.

11.Future Work

It includes to set up a subscription server in the network, which can keep track of the phones, workstation etc. that were subscribed to get the emergency alerts and then also adding the PUBLISH method to the sip model so that emergency events can be published as soon as an emergency is detected. The Notify method that I created will be used along with subscribe and publish methods. As I mentioned in last paragraph of section 10 that there will be an XML based CAP template for the emergency messages and alerts and notify method will send these templates to the subscribed clients.

12. References

1. RFC 3265: Session Initiation Protocol (SIP)-Specific Event Notification by A. B. Roach, June 2002.
2. RFC 3903: Session Initiation Protocol (SIP) Extension for Event State Publication by A. Niemi, October 2004.
3. OASIS Standard CAP-V1.1, October 2005.
4. Introduction to opnet simlator by Yi Pan.
5. Opnet Modeler by Tommy Svensson and A Popescu.
6. SIP-Based Emergency Notification System by K Arabshian.
7. SIP – IMS Model for OPNET Modeler by Enrique Vázquez, Alberto Hernández, José Ignacio Fernández.
8. Network Simulations With Opnet by Xinjie Chang.
9. An OPNET-based Simulation Approach for Deploying VoIP by K. Salah.
10. SIP: More Than You Ever Wanted To Know About by Jiri Kuthan.
11. Practical Considerations for Extending Network Layer Models with OPNET

Modeler by Vasil Hnatyshin, G Gramatges, and M Stiefel.
12. OPNET Implementation of the Megaco/H.248 Protocol:Multi-Call and Multi-Connection Scenarios by Edlic Yiu, E Yiu, and L Trajković.
13. On Analysing Cost for Optimizing the Watcher Subscription Time in the IMS Presence Service by Muhammad T. Alam.
14. SIP Presence Location Service by WilsonWu.
15. Aggregation Efficacy of Resource List Servers in IMS Presence Services by Michael Pitman and N Ventura.
16. https://www.opnet.com/training/network_rd/modeler.html. Training presentations and videos by Universoty of Opnet.
17. TPAL model user guide:TPAL_model_desc.pdf It is a user guide.