NG911 Report - Android Team

Under the guidance of:
Prof. Henning Schulzrinne

Team members:
Pranay Dalmia

JinHyung Park

Anuj Sampathkumaran

Introduction:

The Android client provides means of communicating with the 9-1-1 call taker using text messaging and image transmission. The app has two modes of text messaging, Instant Messaging and Real-Time Messaging. The app also supports transmission of location data to the 9-1-1 call taker to quickly dispatch help. Additionally, the user can also choose to take a photo of his/her surroundings and send it to call-taker. In a nutshell, the NG911 Android app is an effective way to communicate with a NG911 call-taker by text and/or images.

Application Features :

1. Instant Messaging

2. Real-Time Messaging

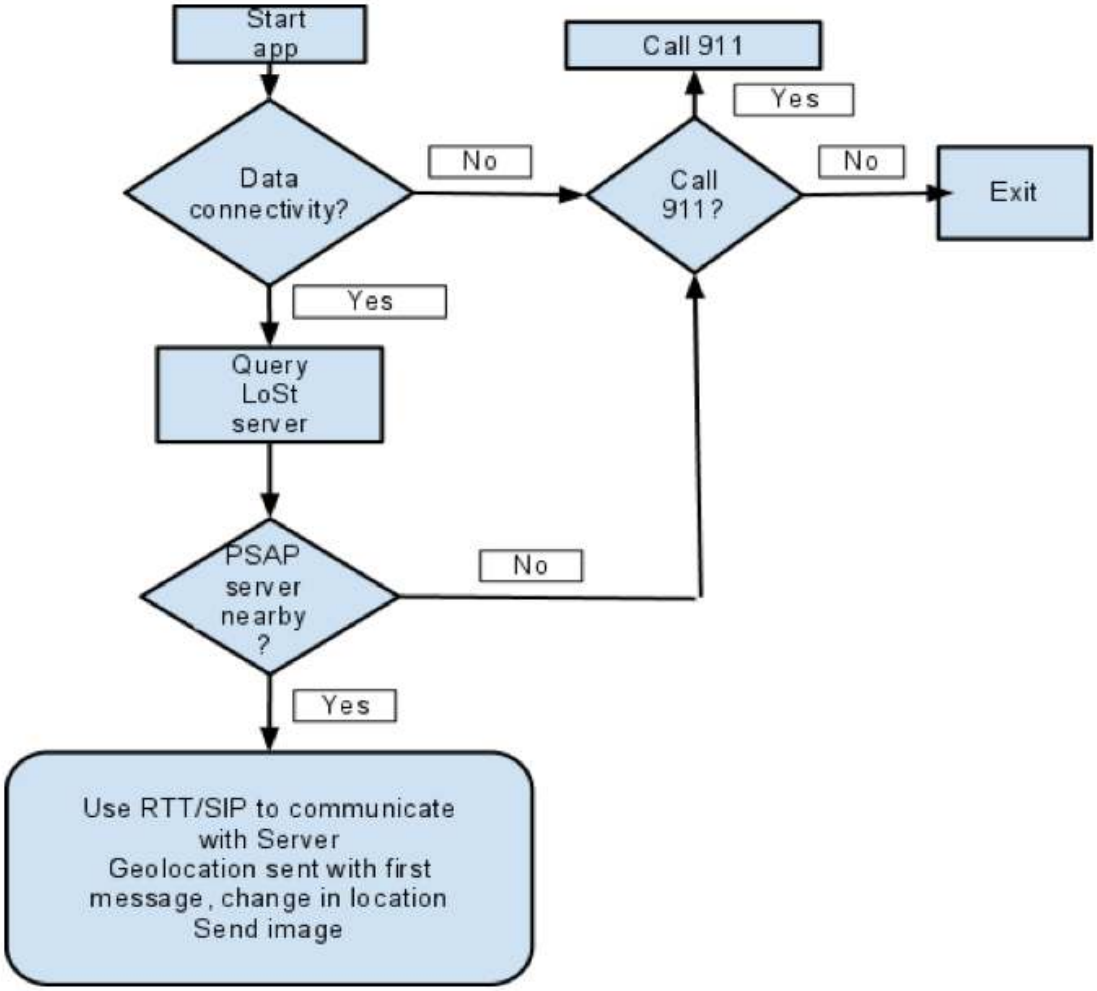
3. Image Transmission

4. Location Transmission.

Application flow:

When the application is launched for the first time by the user, it prompts the user for his name. This name is stored persistently within the application for future use. The application then checks if data connectivity is currently supported by the application. If there is no data connectivity then the user is informed that there is currently no data connectivity and is provided with the option of calling 9-1-1 directly or exiting the app. On the other hand, if there is data connectivity, then the application queries the LoST server to check if the local PSAP supports NG 911. If not, the user is once again provided the same interface which allows the user to place a direct call to 911 or exit the app.

Once network connectivity and PSAP support is confirmed, the chat user-interface is displayed to the user. The user has the option of switching between instant messaging and real-time messaging using a radio-button. This switching can be done on the fly without losing the conversation flow.



The instant-messaging provides a traditional bubble style chat UI to the user. Also, if a particular message was not received by the 9-1-1 call taker, the user is appropriately informed about the same.

The real-time messaging is initialized during launching the application. After receiving the result of the LoST querying, the main thread starts a new thread to send SIP INVITE message to get the SDP session information. The new thread finishes after making the RTP session between the application and the call taker automatically.

In either of the modes, the user is presented with an option of sending an image to the 9-1-1 call taker to better describe the situation. The user can invoke the camera by pressing the camera icon.

The application also updates the user's location to the 9-1-1 call taker. Along with the first message to the call taker, either using instant or real-time, the location data is send along with the message. Future location updates are sent to the call taker if the user communicates using instant messaging. That is, if the location of the user is updated, only then the location details are re-sent to the call taker.

Implementation details:

1. Instant Messaging

The instant messaging mode uses SIP protocol for communication. It uses the SIP 'MESSAGE' method for sending instant messages across. The application uses the MJSIP library for implementing this SIP protocol. [*ISSUE* : We had two Java libraries that supported the SIP protocol, namely, JSIP and MJSIP. Initially, we implemented the JSIP library. However, Android version 2.3.3 introduced built-in SIP library, but without the 'MESSAGE' extension needed for IM. This library had the same naming conventions as the JSIP, leading to naming conflict on version 2.3.3 and above. Hence the application would be only compatible on versions before 2.3.3. To get rid off these compatibility issues, we switched to the MJSIP library.]

The application sends SIP MESSAGES to the server in the stand-alone mode. These messages are sent over the default UDP protocol. Once sent, the application listens for the "200 OK" response from the server to ensure successful transmission. For each message we set a 4 second time-out. If the 200 OK response is not received from the server within this time, we push a notification to the user that message sending failed.

2. Real-Time Messaging

The Real-Time messaging uses T140 Protocol on the RTP session. To manage the SDP information for making the RTP session, we create UserAgent class, which manages a local SDP information and a SDP information from the server. To implement this part, we followed SipDroid's method of managing SDP information. To handle T140 Protocol, we referenced T140Handler class, which is used by SIPc. Additionally, we modified MJSIP's SIP INVITE messaging sending part to send the PIDF-LO object at the same time; however, the location information should be already sent if a user uses the instant SIP message first. In this case, MJSIP's INVITE messaging part works normally. The hard thing was that we had to debug the Real-Time Messaging on the real device, not the emulator because of the RTP session. Sometimes, the Real-Time messaging characters were missed, or sent several times. To check this problem, we used 'tcpdump' on the real device.

3. Image Transmission

The underlying protocol in image transmission is TCP by way of the SIP stack. Invoking the built-in camera API meant that the user would incur an additional click to approve the photo, once shot. Given that this is an emergency app, the primary goal is to minimize communication time between the user and the NG911 call taker. With this in mind, we decided to implement a custom camera interface which directly sends the image as soon as it is shot. We obtain a byte array of the image as soon as it is shot, and send the byte array as a tcp packet to the call taker.

Once we have get the image binary data, we invoke the SipStack. However, the SipStack had a limitation that it could only send data encoded in strings and not binary. So we had to modify the SipStack (MjSIP) to accept binary data. So once we have the image binary data, we switch the stream from UDP to TCP and send this binary data to the server.

4. Location Transmission

In order to send the location details to the call taker, we set up a location listener that constantly checks for changes in location. This listener provides us with two options: to set the frequency of location update checks(in milliseconds), and to set the minimum change of distance(in metres) to consider it as a location change. For testing, we set both these parameters to 0 but to deploy in the real world, these value will need to be updated to an optimum value. If set too low, it will cause the app to slow down and drain out the battery faster and on the flip side, if set too high, say 10000 metres and 100000 seconds, the call taker may not be updated frequently enough.

Based on these settings, the Android run-time notifies the application of any location update. Once updated, the application sends this location as a PIDF-LO object along with the first SIP Message or SIP INVITE (whichever is first), by creating a multi-part message. If the user decides to use the Instant-Messaging mode, future location changes are buffered in the application. If there is a new location in the buffer which hasn't yet been notified to the PSAP, it is sent along with the next SIP message, if any. This ensures that the application keeps updating the PSAP of location changes without sending too many messages.

5. LoST Querying

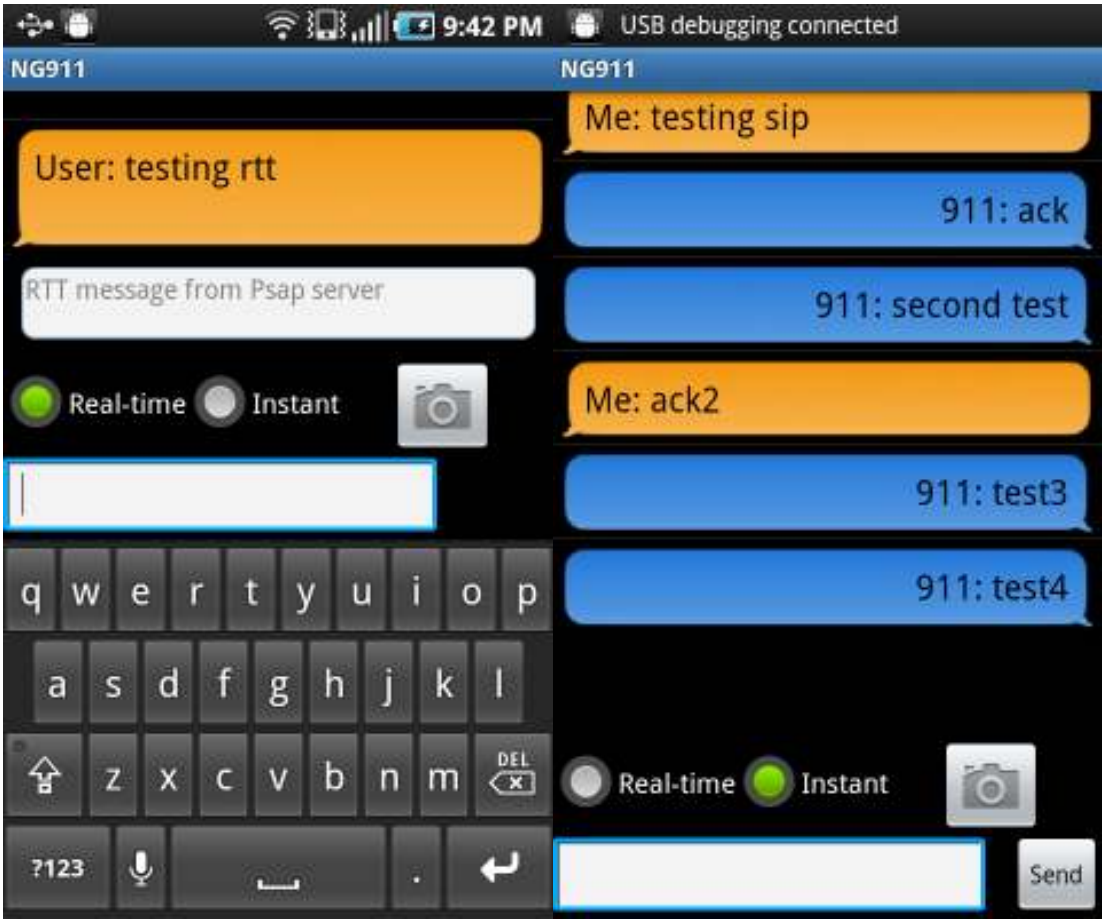
To get the nearest PSAP server, we have to send the query to the LoST server with the current location. This is done by LostConnector class. After acquiring the current GPS, sending the LoST querying is called automatically. Based on the query result, the application also initializes the Real-Time messaging session automatically.

6. User-Interface Implementation

The layout we have implemented follows closely on requirements and layout of TIPCon by Omnitor. The layout consists of a vertically oriented layout with the lowest quarter as controls. These controls include text input for the user to send the message, and a radio group to switch between Instant messaging and real time text. Additionally, we place a camera button in this quarter. The rest of the screen space is left for displaying the chat. We implement a bubble list with an orange bubble for outgoing messages, blue bubbles for incoming messages, and a red bubbles for error messages. When in rtt mode, a Text box (say rttTextBox) is present above the radio button control group, where the text that the call taker is typing is currently displayed.

We choose the vertical layout since it best fits the screen size of an Android phone. This suits both RTT and Instant messaging(by way of SIP) really well. This also matches Omnitor's requirements for RTT messaging in the following manner:

The bubble list shows only text that is finalized. What the user is currently typing, appears only on his/her input text field, and what the call taker is currently typing, appears only on the text box(rttTextBox) dedicated for it. Only when the call taker finalizes text by hitting the enter button, does the text get pushed up to the bubble list. Similarly, the user's text gets pushed up to the bubble list only on end of input entry as described in the Omnitor requirements sheet. This includes hitting the "enter" button, a brief period(currently set to 4 seconds) of inactivity after a period "." , and when the input text box reaches its horizontal limit and needs to wrap around to the next line. If the user enters text in RTT mode and does not perform and "end of input" action, then after a brief period of inactivity(10 s), the user is displayed a message prompting him/her to complete the text. Another point worth noting is that the "send" button is disabled when in RTT mode to clearly indicate to the user that the text is sent real time.



The Android API does not allow applications to intercept keyboard events for security reason. Therefore we implemented a TextWatcher which monitors and listens for change in text in a specified text field. Due to this, we need to make sure that the user does not use “predictive text” and “auto-complete” features. We can only enforce this if the user is working on the native Android keyboard because we cannot set flags for any other keyboard type.

7. Installation

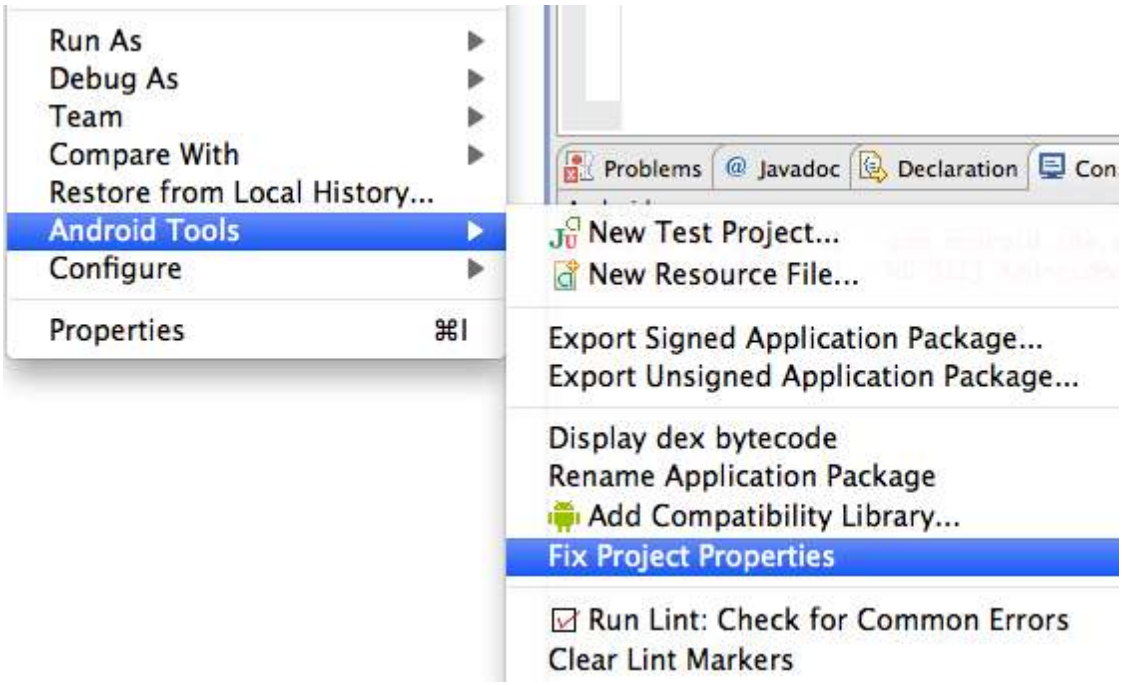
The source code repository of NG911 Android application is located at GitHub. The source code can be checked out at “https://github.com/aevs/NG-911” by the following command.

git clone git@github.com:aevs/NG-911.git

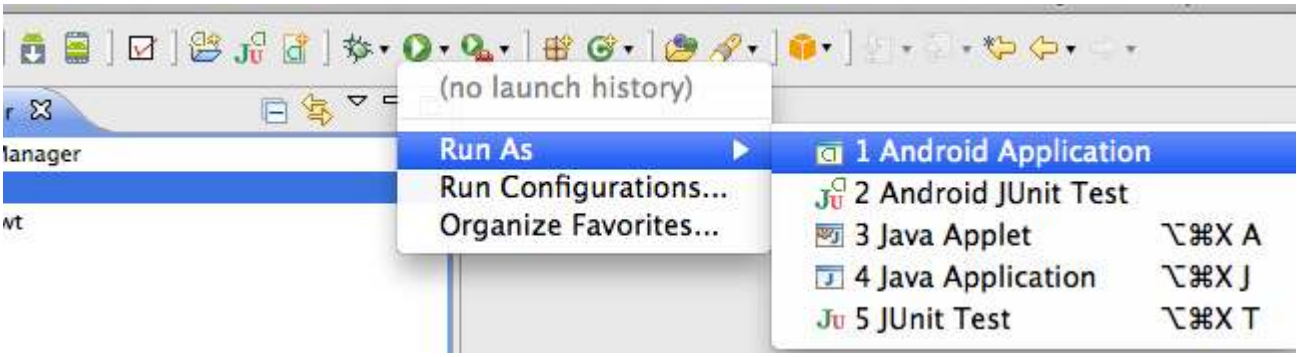
After cloning the project, import the project into Eclipse. At this point, The “project.properties” file should be added into the project. “project.properties” file contains only one line.

target=android-8

After adding “project.properties”, we need to fix the project properties. (This is always happen when we import the Android project into Eclipse.) Like below, right click on the project at Eclipse, click Android Tools->Fix Project Properties under the pop-up menu.



Now, connect the Android device to PC. To build and run the application, click the ‘1 Android Application’ menu under the play icon at the top toolbar like below



8. Demo Movie

On YouTube, we uploaded our demo video. The demo movie is at http://youtu.be/OBvyyeykfb8.