RnD-II Report : Network Load Generator

Guided by:
Prof Kameswari Chebrolu

Submitted by:
Ashish Sonone(110050022)
Sanchit Garg(110050035)

Objective: Design and implement a complete server client system that can be used to crowd-source the network traffic load generation intended to test performance of Wifi Access-Point. Here the clients are android mobiles running the Loadgenerator Android App.

Code details :

github link : https://github.com/ashishsonone/networks-rnd

clone https url: https://github.com/ashishsonone/networks-rnd.git
clone ssh url : git@github.com:ashishsonone/networks-rnd.git

App Link: https://play.google.com/store/apps/details?id=com.iitb.loadgenerator&hl=en

To download the code, fork the repository.

Android app (loadgenerator) code resides in “loadgenerator-plus” branch. Folder “loadgenerator”

Server code(server-plus) code resides in “server-plus” branch. Folder “ServerHandler/webapps/serverplus**”**

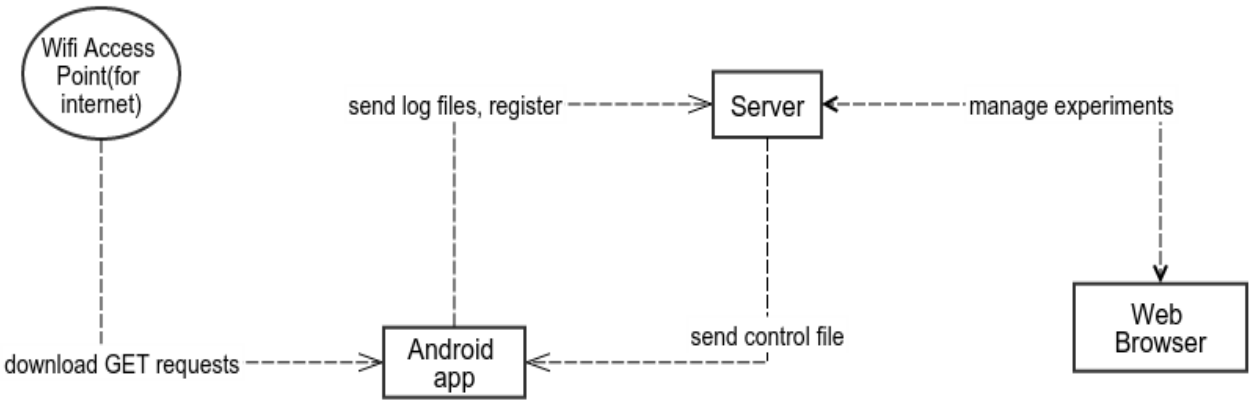
SQL scripts resides in “server-plus” branch. Folder “sql_scripts”

Some useful Terms

1. **Fresh State:** For user it means that no devices are currently registered for the session. For android-client it means that app has just opened for the first time after killing.
2. **Session:** A user create session and can conduct multiple experiments in that session. A session is persistent i.e. now sessions are not deleted after user is logged out.
3. **Session ID:** There is a unique session ID is associated with each session ever created.
4. **Experiment:** Select a pool of devices and they are suppose to generate load based on control file send to them.
5. **Experiment ID:** Experiment ID is unique across all experiments of all users and sessions.

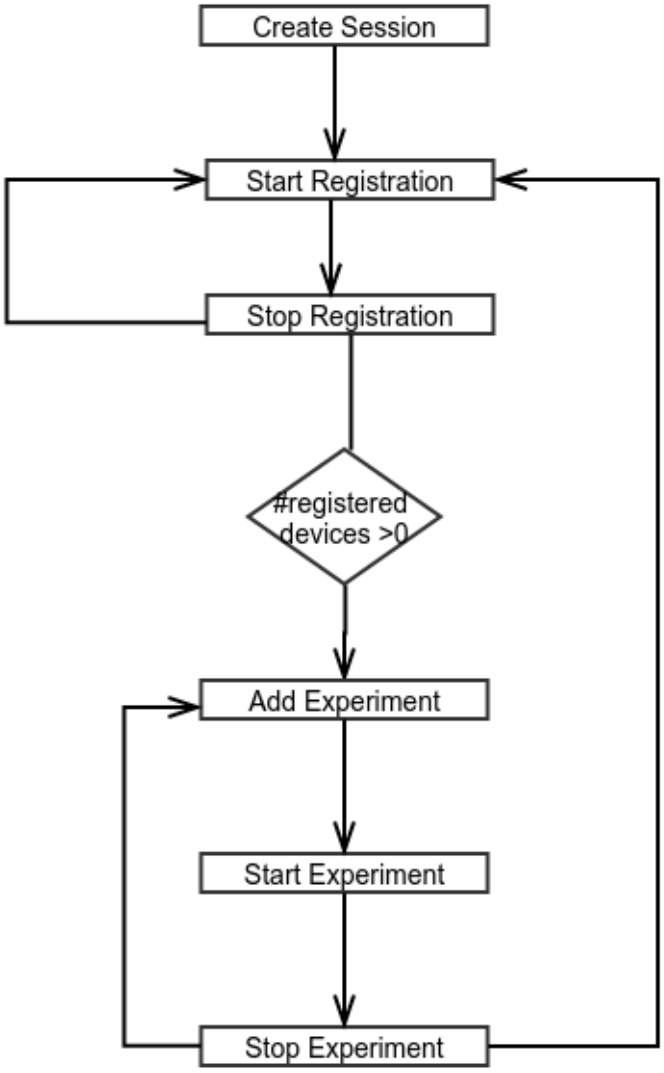
6. **Control File:** File containing get requests along with time offset (in sec) and download url with its mode. Before control file is sent to client When control file is received by client, an event is scheduled at time : file receive time + time offset.
7. **Trace File:** Link Layer level log collected at Access Point. One trace file for one experiment. Can be uploaded to server.
8. **Log File:** Application Layer log file containing response times and start time of each html-resource request.
9. **Event:** A get request generated by client is an event.
10. **Action:** action refers to one of the following
    a. Start Experiment
    b. Stop Experiment
    c. Refresh Registrations
    d. Clear Registrations

Overall Architectural View:



Server : Login and Architecture Details

1) Currently server webapp can be accessed at 10.129.5.155:8080/serverplus
2) login details: <username, password> = <admin, synerg>
3) After login you will be redirected to create session page. Here you can create a new session with duration in hours. You can enter into previous sessions also on clicking session name. Existing sessions can also be deleted by user. In a session a user can conduct multiple experiments.
4) Entering into session takes user to current state of the session. State diagram is as follows. Here arrow implies the clicking event.



5) User can see the current number of devices registered.
6) On clicking Start Registration, devices registrations are opened. Now devices can register, not otherwise. Also user will be shown the number of device registered on rhs.
7) On clicking Stop Registration, no registration request is allowed. If number of registrations are greater than 0, user can start experiment and not otherwise.
8) On clicking Refresh Registrations, user refreshes the list of registered clients i.e. ping all the registered devices and whoever replies back again get registered.

9) There is also an option for Clear Registration. On clicking, it will clear the list of device registrations and takes user to fresh state. It sends signal to all currently registered devices to reset their state as well.
10) On clicking Add Experiment, user is redirected to a page where she can enter the details of experiment, upload control file and select devices manually or randomly and then Starts Experiment.
    a) Manual Selection of devices: If user selects this option, the user is shown the list of existing devices from which the user can choose the devices manually. Devices information is also shown on hover.
    b) Random Selection of devices: If user selects this option, the user has is prompt to enter a number from 1 to #(devices registered). The server then chooses entered number of devices from the registered devices list randomly.
11) On clicking Start Experiment, devices are filtered from the list of registered devices. Then for each filtered device a new socket is opened and events from control file and current time at server(for time synchronization) are sent to device. If device responds with 200OK, server knows that device has received the control file successfully and puts the device into actual filtered devices list(as these are the devices with which the experiment is actually conducting) for that session.
12) When the experiment is started user can see the list of filtered devices(actual filtered devices) and the status of start experiment for the experiment. Status shows Total Req. Sent, Success, Pending and Failed request.
13) Running experiment can be stopped on clicking Stop Experiment.
14) On clicking Stop Experiment, server sends a stop experiment signal to all devices which are in actual filtered devices list and also Stop Experiment Status is displayed in which pending requests are shown.
15) A user can see the list of experiments she has conducted so far on clicking ‘Experiments’ in the menu bar. A user can
    a) See the details of experiment.
    b) See the list of devices participated in the experiment.
    c) Download the Control File for the experiment.
    d) Upload and download the trace file for the experiment.
    e) Delete the experiment.
16) On clicking on experiment name, user can
    a) See the list of the participating devices’ details
    b) Download the log files collected from the devices.

Note:

1. At Most 1 experiment can run at a time in a session.
2. An experiment cannot be started if number of device registered are zero.
3. An experiment cannot be deleted if it is running.
4. Whenever any action is performed, make sure that any other action has zero pending requests as shown in Summary section.

5. There are two log file types. One is summary log file that contains only the response time for each event. Other is detailed log file which contains:
    a. response time for each of the get request for embedded objects in the webpage
    b. the timestamps of file downloads for each percent progress.

DataBase Management:

- Database Management is done in MySQL.
- Database name is wifiloadgenerator
- There are 4 tables in database.
    - users→ username | password
    - sessions → id | name | description | datetime | user
    - experiments → id| name | location | description | user | filename | datetime | sid | tracefilereceived
    - experimentdetails → expid | macaddress | osVersion | wifiversion | numberofcores | storagespace | memory | processorspeed | wifisignalstrength | filereceived

Server Code Structure:

Classes:

1. **Constants.java**: Contains various constants used in the project.
2. **DBManager.java**: Contains methods which handles access to database
3. **DeviceInfo.java**: Storage class for device, contains device information such as, ip, port, macaddress and its other system information
4. **EventGen.java**: Reads control file and convert it into string containing events which then is sent to filtered devices on start experiment.
5. **Experiment.java**: Storage class for experiment details
6. **Handlers.java**: Contains methods for all the server-client interaction such as StartRegistration, StopRegistration, StartExperiment, StopExperiment and so on.
7. **Multicast.java**: Contains methods which spawns thread and sends control informations and actions parallelly.
8. **Initializer.java**: Called once during server start. Initializes list of free sessions id.
9. **Main.java**: Maintains separate map for the current active sessions and current running experiemnts. Also contains list of free session ids.
10. **Session.java**: Storage class for session details. Contains list of registered, filtered and actually filtered devices.
11. **Utils.java**: Contains some utilities functions.

Server Setup:

Requirements:

1. Java-OpenJDK
2. MySQL

3. tomcat7-server and tomcat7-client

Steps:

1. Create tomcat instance by running: `tomcat7-instance-create <folder name>`
2. Download the Server Code and put the content of webapp folder into webapp folder of the tomcat-server.
3. Also download the SQL Scripts
4. Create 2 separate directory named `experimentLogs` and `tempDir`
5. Open mysql console. Create new database e.g. server. Change into database server and run sql scripts in order:
    a. create.sql
    b. insert.sql

    This will create the required tables and a user is created with username ‘admin’ and password ‘synerg’.
6. Move into folder: `cd webapps/serverplus/WEB-INF/classes/serverplus/`
7. Open file DBManager.java and change parameters DB_URL, USER,PASS as your database url, username and password respectively. Save and close the file.
8. Open file Constants.java and change value of parameter mainExpLogsDir to absolute path of `experimentLogs` and value of tempFiles to absolute path of `tempDir`.
9. Compile java classes. Run following the command

    ```
    javac -cp ".:./../../libs/*" *.java
    ```
10. Now move into tomcat folder `<folder name>` and change into conf directory. Uncomment the line `<Manager pathname="" />` to avoid any user session persistence.
11. Start the tomcat server by running: `./bin/startup.sh`
12. Now access the webapp at `http://yourip:8080/serverplus`

Android App : User Manual (for students/volunteers) - a higher level overview

1. Launch the app
2. If device is not connected to wifi, app will prompt you to do so. Connect to the wifi network for which load testing is to be done.
3. Enter server ip, server port and session id provided to you in the respective input text boxes.(Or click on **use default** button to use the default server details). The ip and port information is saved in Android's sharedpreferences so that you don't have to fill them again next time)
4. Click on Start Button.(Note that registration window should be open on the server side for this session). It registers itself with the server and waits for control file. Shows message "Now listening…" if successful otherwise shows error message.
5. When success in above step, now this app will run in background and you can resume your work. App will show messages in its scrollable text view concerning important events happening during lifetime of the experiment such as receiving control file, download resource events, sending log file once experiment is over, and any error/interruption occurred.
6. To exit the app at any time, click on Exit button.

Android App: Download modes

Now the app support two kind of download events :

1) **SOCKET** : the corresponding file/resource is directly downloaded by opening a socket to that file.
2) **WEBVIEW**: the page if loaded and the embedded resources are also recursively downloaded so as to impart a browser like behavior to the events.

Android App : Detailed architecture and features

1. On launching the app, it first checks if device is connected to wifi or not. If not, it shows a dialog box to go directly to wifi settings page and connect to wifi access point for which load testing is to be done.
2. Next details for server ip and server port needs to be filled(if not auto filled already) on which server is listening. Also before starting the experiment, session id needs to be entered to identify which session this experiment belongs to. Session id is unique across all sessions ever created
3. On clicking start button, Start button is now disabled. Also the ip and port entered are stored in Sharedpreferences(small database kind of thing) so that next time these details are auto filled on launch. There is also default server and port which is 10.129.5.155:8080 which can be filled by clicking on **use default button.** It spawns a

service called BackgroundService to run in background. Following things happen in the service:

a. pings the server upto 3 times to make sure server is available/accessible
b. a background thread is created to send all pending log files (stored in <External Storage Directory>/loadgenerator/logs). If a log file is present in log folder this means that it has not been sent. Because after sending log file successfully, the log file is deleted locally. The structure of log file is presented in a later section. Once this thread completes it displays a concise message on screen stating number of success/failure.
c. A listening socket is opened which serves as point of contact if server wants to communicate with the device. Now a request is sent to server to register the device. This request contains following information:
    i. session id
    ii. device ip and listening socket port. App is listening on this socket for messages from server
    iii. os version, wifi signal strength, mac address, processor speed, number of cores, storage space, memory. This information is used on server side to select devices for a particular experiment.

   If the registration is open on server side, it receives a 200 OK response from server. Otherwise some problem occurred and registration is unsuccessful. A message is shown in the app screen for the same.
a. If registration failed due to some reason, the listening socket is closed. App state is reset and Start button is enabled again. Need to try again.
b. If registration is success, a new 'listen' thread is launched which will listen on listen socket for messages from server.
c. A timeout alarm is set after the maximum duration of experiment. Timeout is configurable in Constants class(It is **default 3 hours)**. This alarm trigger marks the end of experiment. So if app is still listening for server messages, its state is reset to as if the app was just started. Nothing is running in background now.

2. App is now waiting to receive control file from server in 'listen' thread which will contain experiment details. Control file contains multiple events where each event is specified in terms of timestamp and corresponding GET request - these can be of two types as mentioned in previous section - SOCKET & WEBVIEW. The timestamp is the moment or server time when the server wants the event to be triggered.
3. On receiving the control file, alarms are set for each event and once that alarms goes off, event is handled and next alarm is set.
4. **Time Sync :** To ensure that the events are triggered at the same time across all devices even if their local clocks differ, there is a need for time synchronization. The json that server sends when sending the control file also contains current Server Time(call it **ST**). When json is received , device knows its local current time (**LT**). Now the two clocks differ by **delta = ST - LT**. Now approximate server time at any moment is **current local time - delta.** This is used to schedule the events so that all the devices schedule events at approx the same moment.

5. Event handling involves spawning a new thread for downloading and logging activity. Log file format is explained later. Event handler can be of two types depending on the event type. All log messages are appended to one log file named <experiment id>. After each event, message is also displayed on screen stating its result.
6. Server can send a command to stop the experiment even if experiment is going on in device, in which case app will clear all pending alarms and send the partial log file.
7. If not interrupted in between and all download events complete normally, device sends the complete log file to server. Complete log file has EOF string at end denoting experiment completed successfully. If sending is successful, log file is deleted from local log folder.

Android App Code Structure:

MainActivity.java : starting activity
Constants.java : Contains constant values such as key names, timeout values, log folder etc
Utils.java : utility functions for sending exit signal, ping, getting ip, mac address, memory info, etc
Threads.java: functions called in threads such as listening server, download files, send log files in background.
RequestEventParser.java : class to parse the control file received from server and form a list of events
BackgroundService.java : Service that runs in background when pressed the Start button
DownloadService.java : Service that is called to handle each download request when alarm is triggered
AlarmReceiver : handler which is called when alarms are triggered
ResponseReceiver.java : class which receives broadcast messages to display on screen. It is the way threads communicate to UI thread when they want to show a message.
MyBrowser.java : This is the class to handle the WEBVIEW events received so that download behaviour is that like a web browser . It extends Android WebViewClient which is used to set the webviewclient of Android Webview(basic browser module) while serving that event.

Problems faced :

1) The control file that was sent contained the timestamp (according to server's clock) when a event should be triggered in all the devices simultaneously. But since the device clocks are usually out of sync, the events were triggered at different times, thereby reducing the effectiveness of the load testing experiment being performed, because all the devices try to download the resource/page at the same time, the contention creation won't be successful and hence load testing would be ineffective.
   Solution: Time synchronization between server's clock and device clock so that alarms for each event is triggered at approx the same moment across all devices. This has been explained how this is done.

2) We had a hard time finding a suitable library in android, so that a web page can be downloaded along with embedded resource objects and still we can measure the response times. There are many browser modules available but they don't provide the flexibility to measure individual response times for each resource request.
   Solution : Finally we were able to override some methods in Android WebViewClient class so that every time a request is generated(for e.g when downloading embedded objects), we can catch this request and then handle it ourselves by downloading it using a socket. This way, generation of recursing requests is done by this class, whereas downloading is done by us using sockets.
3) When we first performed the experiment in the lab to test scalability, we found that the server was taking a lot of time in sending the control files to the clients, when we realised that the server was sending these control files sequentially, i.e it sends the file to one android client, waits for 200 OK from client then sends the file to next client. This way it was taking a lot of time just to send the control files during which a lot of devices timed out which was just 15 minutes(when this experiment was performed)
   Solution: We came up with the idea that in the sequential approach, major time wasted was in waiting for reply from the client. Sending of control files can be done parallely in separate threads, and we can keep track of how many threads completed successfully completed their execution we can know the status of this task. Also the timeout duration was increased from 15 minutes to 3 hours.

Experiments & Results:

Venue : OSL, Mathematics Department
Number of devices : 20
Sample Control File Uploaded to the server:

GET 30 WEBVIEW http://www.iitb.ac.in/
GET 30 WEBVIEW http://www.cse.iitb.ac.in/
GET 30 WEBVIEW https://www.ee.iitb.ac.in/web

Sample Control file that device receives:

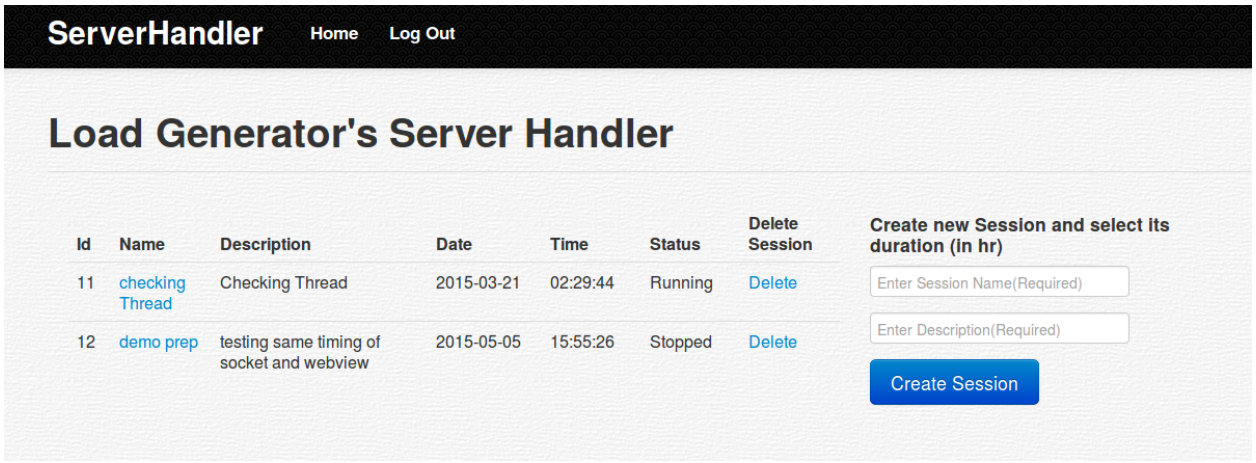
GET 2015 4 5 15 27 58 847 WEBVIEW http://www.iitb.ac.in
GET 2015 4 5 15 27 58 847 WEBVIEW http://www.cse.iitb.ac.in
GET 2015 4 5 15 27 58 847 WEBVIEW https://www.ee.iitb.ac.in/web

Total devices Registered : 20
Total devices Filtered : 17
Total Log Files Received : 15

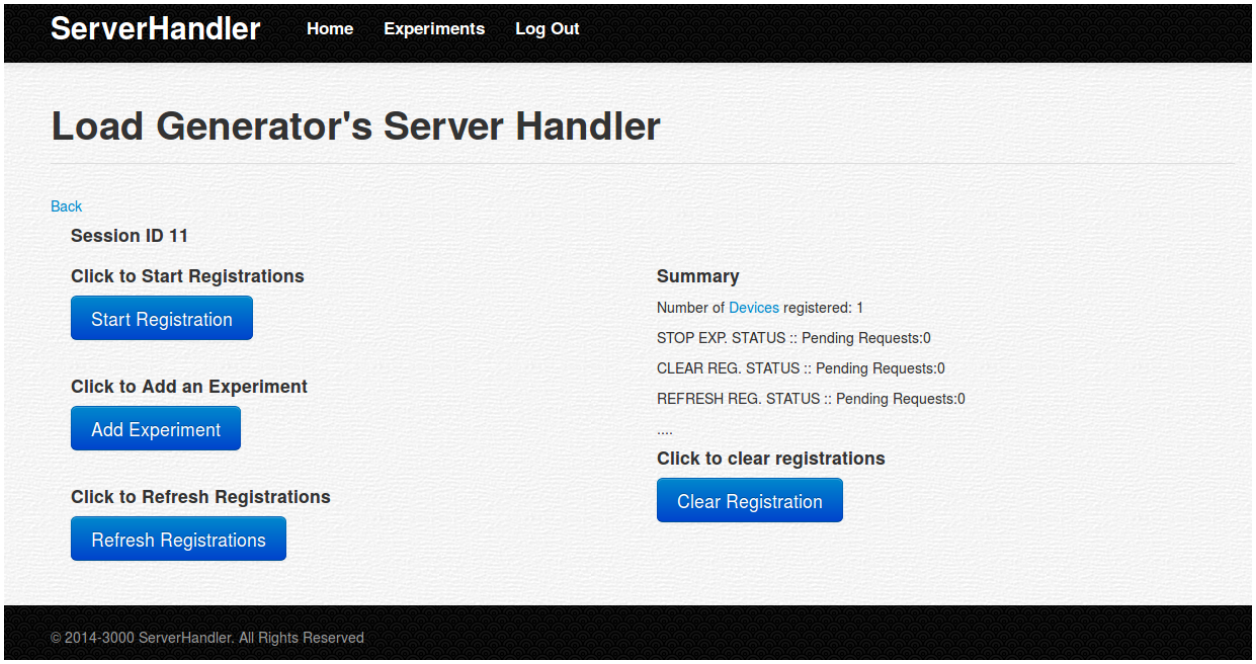
.

Snapshots:

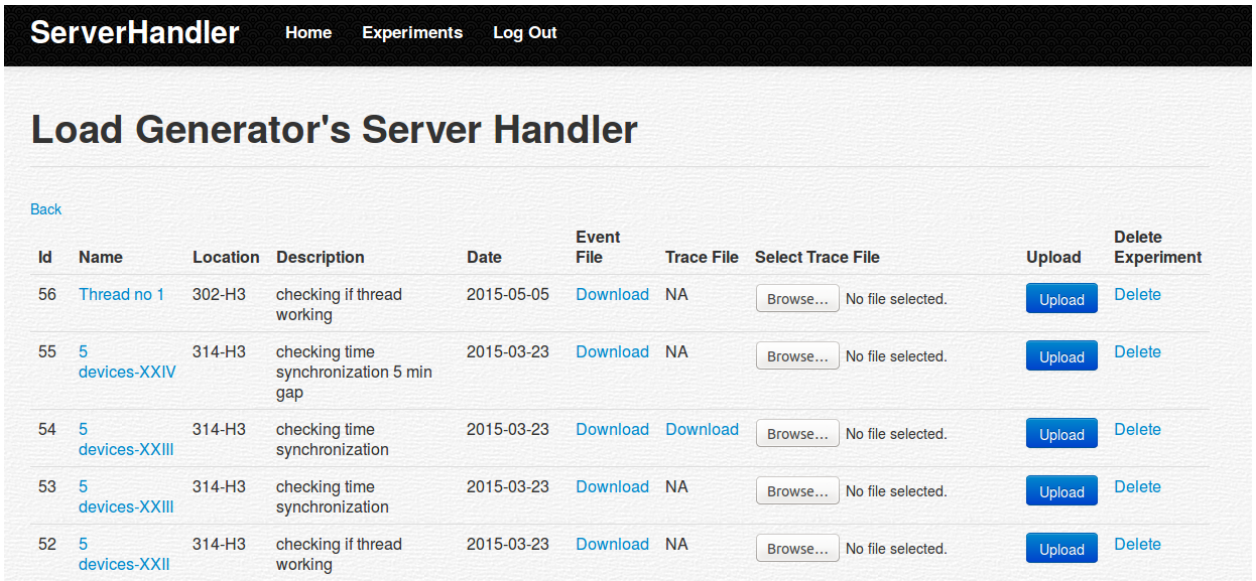
Home Page showing list of active sessions:



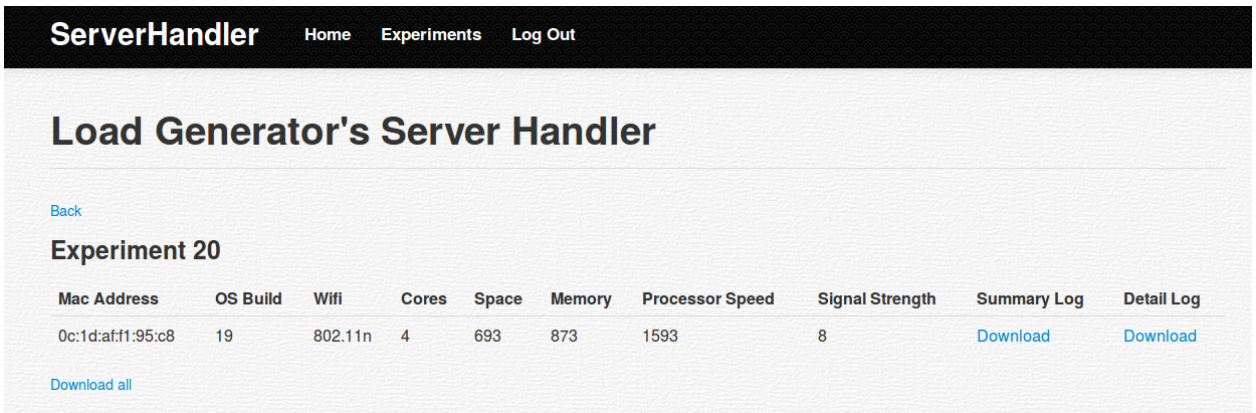
Session Page showing status of each action:



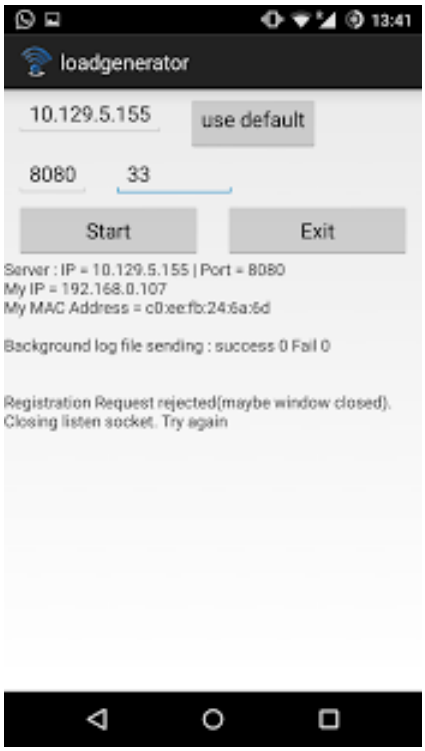
List of experiments in a session:

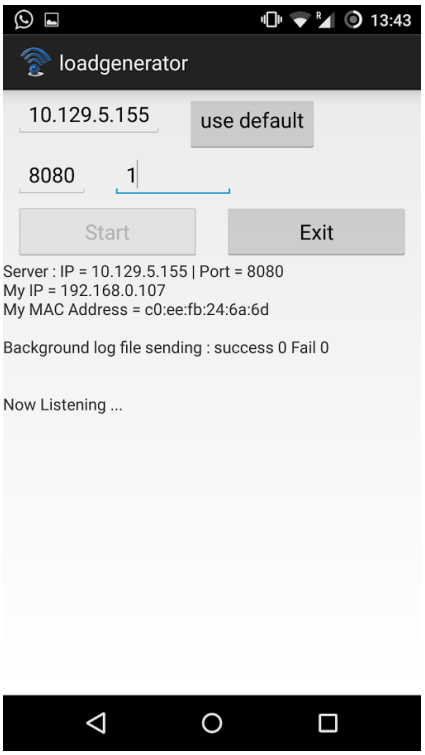


Details of experiment number 20.



Android App Screenshots:





Future Work:

- Even though we performed experiments on a larger scale this time i.e lab setting(where participating devices were around 20), but only a few times. We need to perform the experiment many times
- We could write scripts to analyse the log and trace files for each experiment so that it becomes the complete solution for the wifi load testing