COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

COMS 6901 Unified Heterogeneous Networking

Control Middleware Team Project Report

Qing Lan	**ql2282**
Shen Zhu	**sz2609**
Kai He	**kh2789**

Contents

1. Introduction

1.1. Background

When several Wi-Fi networks are available, phones would connect to the network with the highest signal strength. However, this approach has some limitations: it cannot switch network intelligently, nor could it switch network seamlessly.

Let's consider two scenarios:

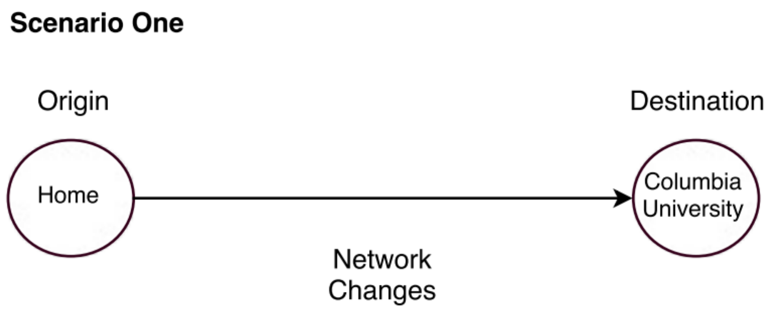


Fig. 1 Scenario One

For example, as shown in Fig. 1, say I am walking from home to Columbia University and listening to music. The networks around me change because my location changes, and my phone would choose the Wi-Fi network that has the highest signal strength. When my phone changes the network it connects to, the IP address of my phone changes, and I would lose connection to Internet and could not have smooth experience listening to music.

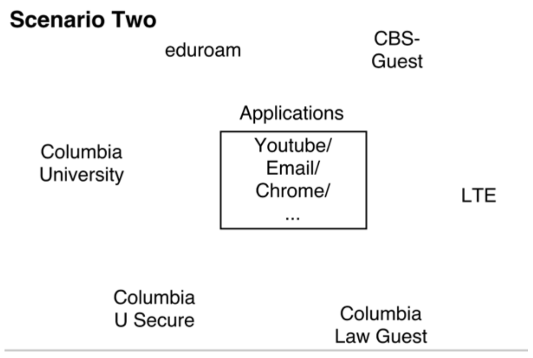


Fig. 2 Scenario Two

The second example: I am sitting in Butler Library and using several applications on my phone, such as YouTube, WhatsApp and Chrome. And many Wi-Fi networks are available around me, such as Columbia University, Columbia U Secure and CBS-Guest. Different applications have different preferences for networks: YouTube may need network with high bandwidth and WhatsApp may have preference for network with low latency. When I am switching among different applications, I want my phone to intelligently choose the most suitable Wi-Fi network based on my preferences.

1.2.Objectives

Inspired by [1], we want to build control middleware. Fig. 3 shows where the control middleware lies in the networking stack, which is on top of transport layer and under application layer.

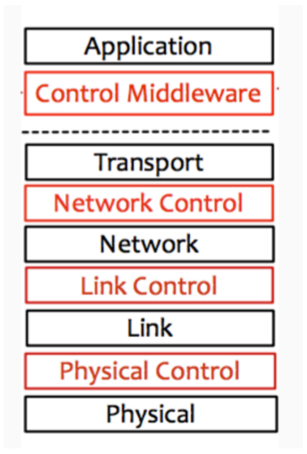


Fig. 3 Position of Control Middleware

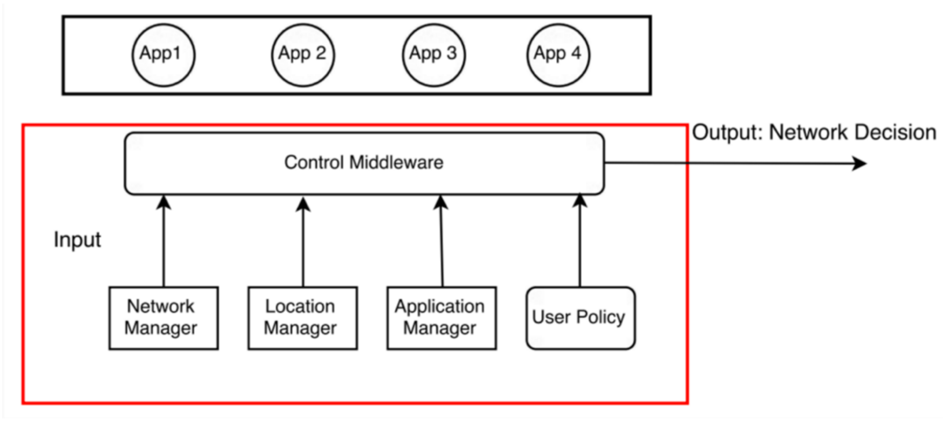


Fig. 4 Architecture of Control Middleware

Fig. 4 shows how this control middleware works. In an Android phone, this control middleware collects the network, location and application data, it takes input from network manager, location manager, application manager and user policy. These network data (SSID, latency and bandwidth), location data(longitude and latitude) and application data (application package name, upload statistics and download statistics) are sent to the policy engine inside this control middleware, and then this policy engine would make network switch decisions for this phone.

In this semester, our main goal is to finalize policy engine based on different use cases. In order to do that, we need to understand various use cases where network switches are needed, improve the UI for users to add different policies and finally generate the network decisions under different circumstances.

2. Methodology

2.1. Prior Work

2.1.1. Data Collection and Policy Engine

Last semester, a team of five people implemented the HetNet middleware which could collect system, network and location information. Specifically, signal strength, signal frequency, SSID and security protocol were captured successfully by network manager. All the information was captured through the event listener. After that, the data was merged in the policy engine class and stored on to the cloud database for future analysis. During this process, Wi-Fi manager and telephony manager APIs in Android system were used. However, the bandwidth and latency of the current network were not properly captured. The HetNet application also included a draft policy engine where user could add the network and location preferences.

2.1.2. EventBus Architecture

The EventBus Architecture [2] is an open source project widely used in complex Android system. It provides a convenient thread-management system where every thread in the system is either publisher or subscriber. This design was used in HetNet for tracking network, location and system information. Once there is a change in network, location or system status, the publishers would trigger an event to ask the related information fetcher to fetch the data. In HetNet design, the central message manager was policy engine class which contains several AsyncTask and would only be triggered when a new message is received. The messages are the event responses coming from each publisher.

2.2. Data for Modeling

As stated above, our objective is to make network switch based on user policy and current context (location, network and application information). Therefore, we need to create a decision maker that takes user policy, location, network environment and application data as inputs and outputs the network to switch. In order to build this decision maker, we need to collect network, location, and application traffic data and analyze them.

There two kinds of data we need to collect and analyze: the first one is network data, including the network SSID, location, bandwidth, latency and security protocol. Through analyzing this data, we can find which network has the highest bandwidth or lowest latency around one location. The second kind data is the application traffic data, this data contains information about the upload and download statistics of each application, thus we can find which application consumes most data. The analysis of application traffic data can be found in Chapter 3.2.1.

2.3. Data Study

2.3.1. Network Data

Only after we have collected and analyzed enough network data can we decide which network is the most suitable one based on user policy. Therefore, for each network, we want to collect the following properties:

SSID (service set identifier): A human readable identifier of Wi-Fi network.

Bandwidth: "The bitrate of available or consumed information capacity expressed typically in metric multiples of bits per second" [3].

Latency: "A time interval between sending network request and receiving response" [4].

Location: The geographic location of this network, including longitude and latitude.

Security Protocol: The security protocol used by this network, such as ESS and WPA2-PSK-CCMP.

2.3.2 Application Traffic Data

Network data helps us know what networks are available to us in the environment. Besides that, we also need to understand the users' behavior of using different applications so that the policy engine could make a proper network switch decision for them. For example, if by collecting users' application data we know that YouTube consumes the most data bandwidth, we can "remember" this behavior in the database. Later if we detect that the user is currently using YouTube again, it is very likely that switching to a higher bandwidth network would be a good decision for this user. In other words, we are trying to build profile for each application, which contains the following properties:

Application Package Name: Unique identifier for each application in an Android phone.

Upload Data: The upload data consumed by this application.

Download Data: The download data consumed by this application.

Device ID: Unique identifier for the device that running this application.

Location: The longitude and latitude of using this application. This information helps us filter data usage by a certain location, which is useful for making network switch decision based by location based policy.

2.4. Policy Engine

Policy engine is the decision-making part of our control middleware. The inputs of policy engine are events of application and location changes, and the outputs are decisions of whether to switch network or not.

In our control middleware, policy engine is an Android service running in the background which continuously listens to application and location changes. Once the user switch to use another application or moves to another location, policy engine check the analysis results for user data and the policies specified by the users. By this means policy engine can know what kind of network the user wants to user for this new application or location, and after that policy engine make the decision to switch to that certain network.

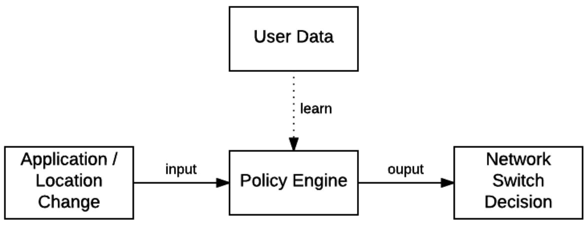


Fig. 5 How Policy engine makes decisions

3. Implementation

In our implementation, we only consider Wi-Fi networks. We did not experiment on cellular networks and make HetNet support cellular networks due to two reasons: First, the time to do this research project is limited. We were more focused on establishing the whole pipeline. Second, the demo phones we got only have access to Verizon network, but none of us have Verizon sim cards. We tried to request contract-free phones, but had not received any feedback.

3.1.Collect Data

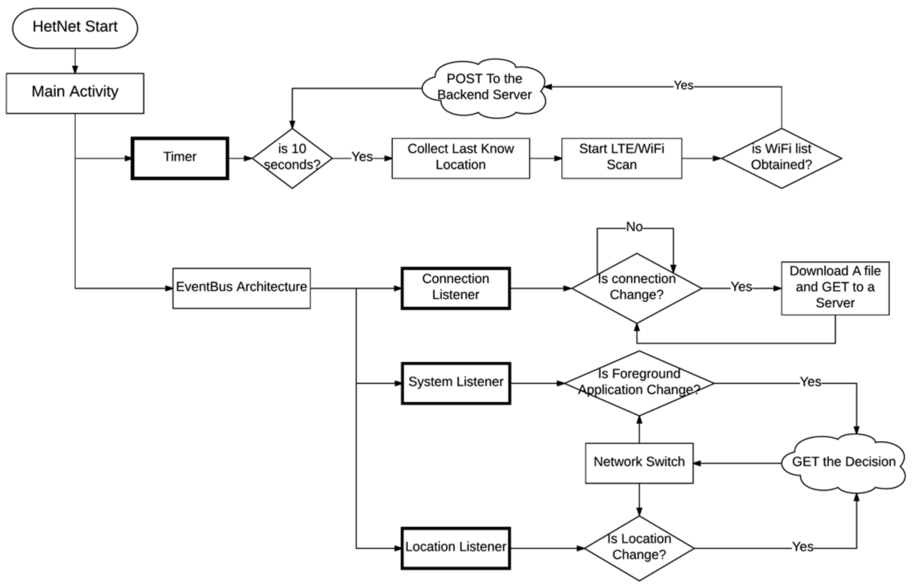


Fig. 6 Architecture Overview

3.1.1. Time-based Static Data Collection

Previously, the HetNet middleware would consume a great amount of the memory space due to the anomaly behavior of network manager. A design flaw found in “NetworkListFetcher” is that there is a while loop that may keep waiting for finishing collecting all network information, which caused the fetcher thread "unfinished" and remains in the system for a long time. Since two threads are executed at the same time, the while loop is waiting for the flag once the other one finished. However, the delay of one thread would cause the entire fetcher delayed for a long time. In that case, if the HetNet runs through a long period, the accumulation of the “NetworkListFetcher” thread would result in crash of application. In order to fix this problem, we create a thread checker to check whether the current thread is existed or not. If existed, new thread will be delayed until the previous one finished. In this case, the accumulation of “undead threads” problem is solved.

In this semester, we have simplified the previous managers to improve the efficiency of the middleware. The previous system listener/fetcher, location fetcher and network listener/fetcher were disabled in the EventBus system. A class called "AppDataService" is used to capture all network information and location information. At this stage, only data could be obtained directly from the APIs are captured. The APIs used to collect network and location information are the same as the previous HetNet. A timer set in the main thread triggers this service every 10 seconds. Comparing to the event-based design, time-based data collection reduces a lot of times of duplicated data capturing. It also solved the known issue with "unfinished" fetcher class in the previous network manager.

Besides network data, we also need to capture application data so that we could better understand users' behavior of using application, which can help the policy engine make an appropriate decision. There are mainly three methods to capture application data in Android phones:

1) Bionic Library

For the rooted Android phones, we can directly interact with the low-level operating system and capture the application traffic data through the Bionic Library [5]. However, we need to collect this application data from all the users, but apparently not all the users would use rooted Android phone. Therefore, this method is not applicable to us.

2) Android Library

For the unrooted phones, Android does provide APIs to get the upload and download data. However, this library can only provide simple upload and download data consumption, which can satisfy our current needs but is not sufficient for future usage. For example, when a user is using Chrome, we not only want to know the amount of data Chrome consumes, but also want to know what kind of website Chrome tries to communicate with. User's network need of watching video through Chrome and visiting Wikipedia are quite different, so we need to know the destination the user tries to connect to, so that the policy engine can make a better decision. Therefore, this method is not used in our implementation.

3) VPN Service

For the unrooted phones, we can use VPN Service [6] provided in Android system to capture more comprehensive application traffic data. Using VPN Service, the data packet will not go to the web server directly. Instead, it will be redirected through the TUN interface in Android and VPN Service can capture more information of the data packet in the meantime. Therefore, besides upload and download data consumption, we can also capture the source address, source port, destination address and destination port of the data packets, which can be useful for further data analysis of this project.

The application data capture is also time-based and will be triggered every 10 seconds. The mechanism is same as network data capture mentioned above.

3.1.2. Event-based Derived Data Collection

Apart from the data collected directly from the APIs, we still need to measure other data, especially for bandwidth and latency of the network. We are using the previous network listener registered on EventBus to track the connection change between Wi-Fi networks. Once an event is triggered, the system would start downloading a file from a web server. The downloading speed may vary based on network and location. In our design, we assume the downloading speed from server would be the same in everywhere. The start time and the end time would be captured to calculate the network speed. Latency data is captured through several HTTP requests to the server and monitored the delay on the response. In that case, the Event-based data collection would be finished through calculation, which is based on the following equations:

Bandwidth Calculation:

1) Record start time t_1
2) Download a size of s static file from http://download.thinkbroadband.com (This website is used for downloading file with known size to test network speed)
3) Record finish time t_2
4) Network bandwidth can be calculated by $\frac{s}{t_2 - t_1}$

Latency Calculation:

1) Record start time t_1
2) Send HTTP GET request to http://google.com n times
3) Record finish time t_2
4) Network latency can be calculated by $\frac{t_2 - t_1}{n}$

3.2. Analyze Data

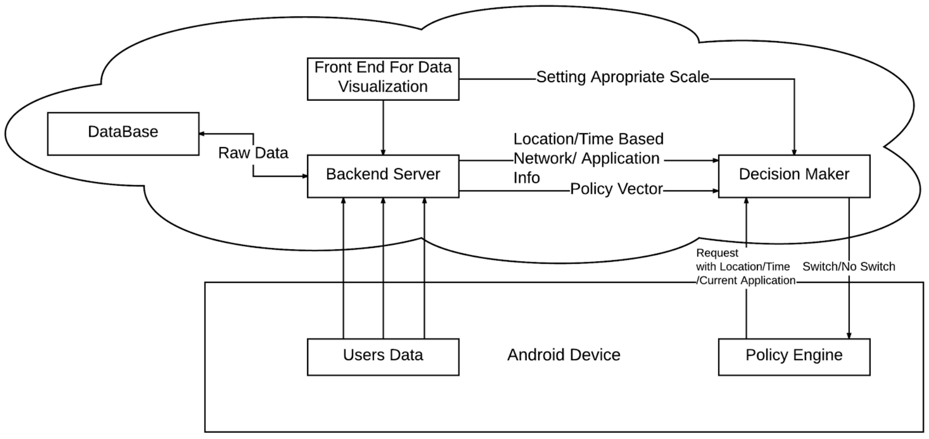


Fig. 7 Architecture of Data Analytic Engine

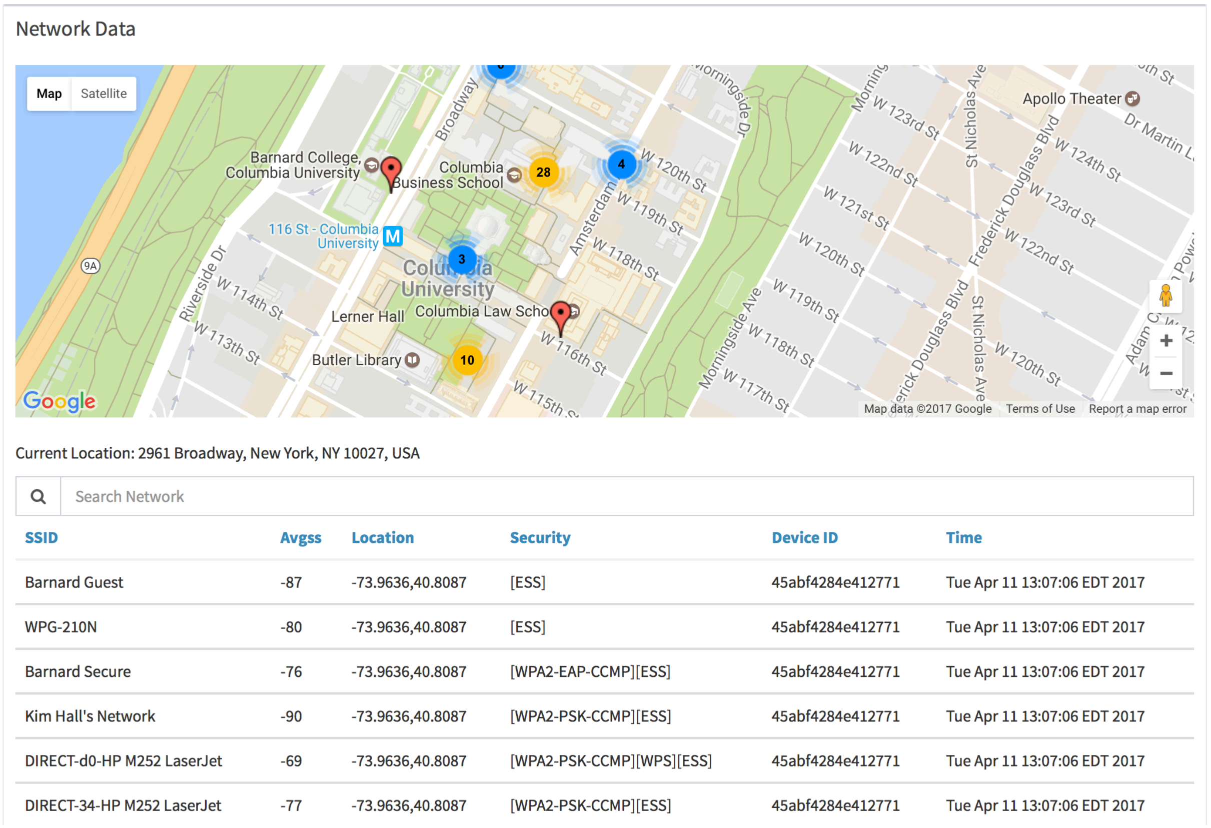
Fig 7 shows the architecture of our data analytic engine. In Android device, it first collects network and application traffic data and sends it to our backend server. Server receives this data and stores data into database. Besides, this server analyzes collected network, application data and responses to the request of frond end. Frond end receives and visualizes the results.

When location or application change, it will trigger an event in policy engine. Then policy engine will send the context information such as location and current application to decision maker. Decision maker receives this information and will query backend server about which network to switch. Backend server integrates the information of networks and sends it back to decision maker. Finally, decision maker will tell policy engine which network to switch to.

3.2.1. Data Analysis and Visualization

Network and application traffic data are analyzed in backend server and the results are shown in the frontend.

First, we built a map for different networks, so we can find all the networks around one location. As shown in the figure below we have collected network information around Columbia University and created this interactive map.



Network Data

Current Location: 2961 Broadway, New York, NY 10027, USA

Search Network

SSID	Avgss	Location	Security	Device ID	Time
Barnard Guest	-87	-73.9636,40.8087	[ESS]	45abf4284e412771	Tue Apr 11 13:07:06 EDT 2017
WPG-210N	-80	-73.9636,40.8087	[ESS]	45abf4284e412771	Tue Apr 11 13:07:06 EDT 2017
Barnard Secure	-76	-73.9636,40.8087	[WPA2-EAP-CCMP][ESS]	45abf4284e412771	Tue Apr 11 13:07:06 EDT 2017
Kim Hall's Network	-90	-73.9636,40.8087	[WPA2-PSK-CCMP][ESS]	45abf4284e412771	Tue Apr 11 13:07:06 EDT 2017
DIRECT-d0-HP M252 LaserJet	-69	-73.9636,40.8087	[WPA2-PSK-CCMP][WPS][ESS]	45abf4284e412771	Tue Apr 11 13:07:06 EDT 2017
DIRECT-34-HP M252 LaserJet	-77	-73.9636,40.8087	[WPA2-PSK-CCMP][ESS]	45abf4284e412771	Tue Apr 11 13:07:06 EDT 2017

Fig. 8 Network Map

Each marker in this map corresponds to a network, and if there are several networks around one location, we created a cluster in this map and marked the number of networks in that cluster. If you click one marker in this map, the information about the networks around this location will show up. For example, in Fig. 8, when I click the red marker near Barnard College, the frontend shows that this location is "2961 Broadway, New York, NY 10027, USA", and the information about the networks around this location is also presented. We also collect the device ID so that we can differ the data collected by different phones, therefore, the network decision made by our system differs on different phones.

For network data, we also analyze signal strength, bandwidth and create the following bar charts, as shown in Fig. 9 and Fig. 10.

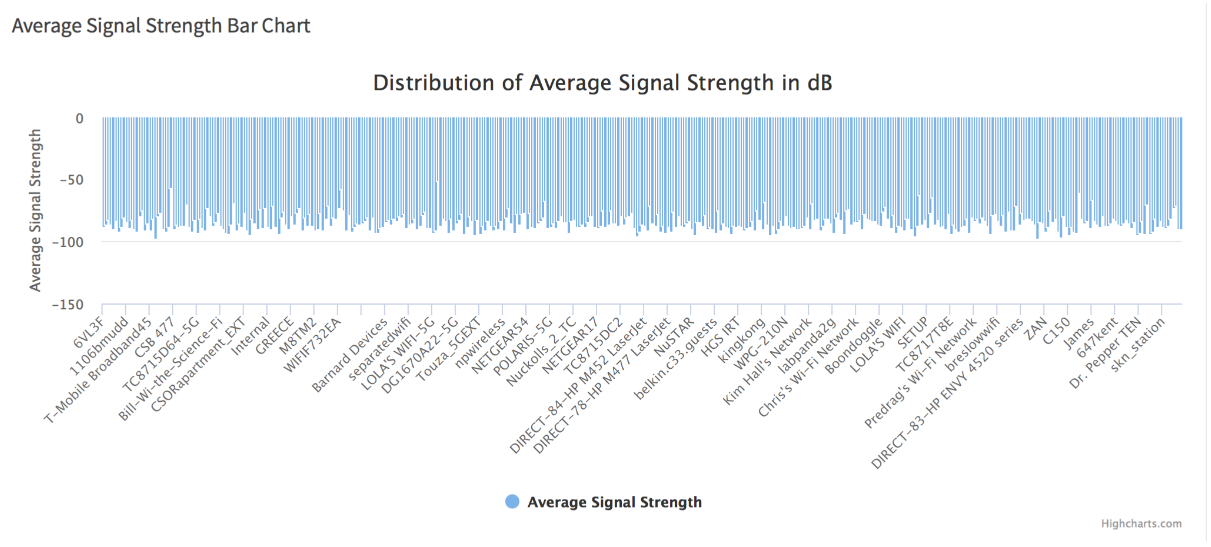


Fig. 9 Distribution of Averaage Signal Strength

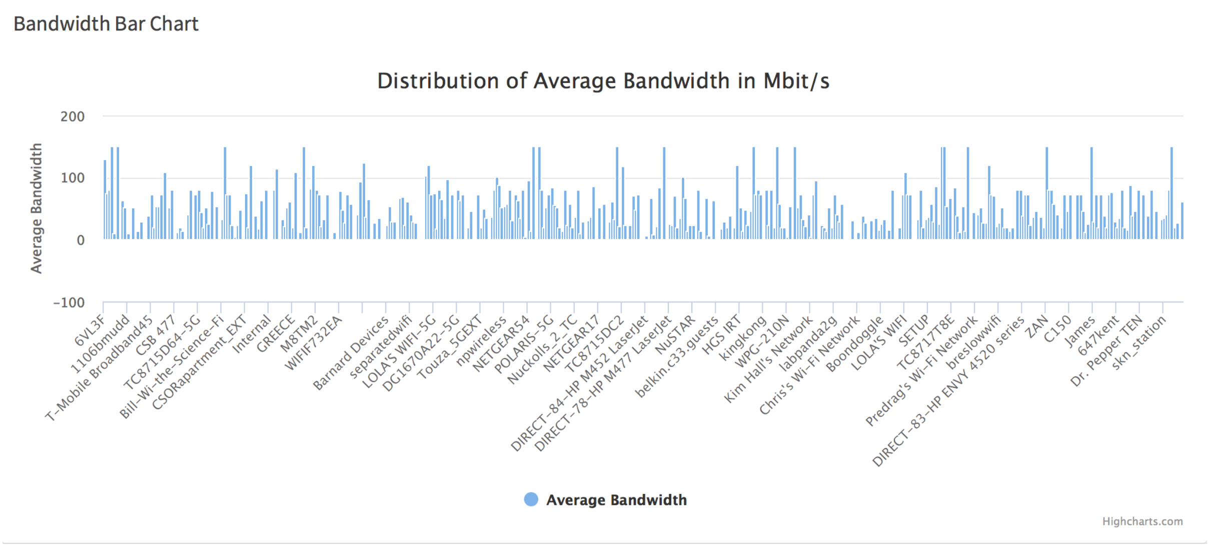


Fig. 10 Distribution of Average Bandwidth

These data allow us to know the range of bandwidth and signal strength of different networks, which is helpful if we want to do some normalization when analyzing data. These charts also give us an intuitive knowledge of which network has stronger signal strength and higher bandwidth.

For application traffic data, we first built the pie charts of upload and download statistics for different applications, as shown in Fig. 11 and Fig. 12. They show the percentage of upload and download data used by different applications.

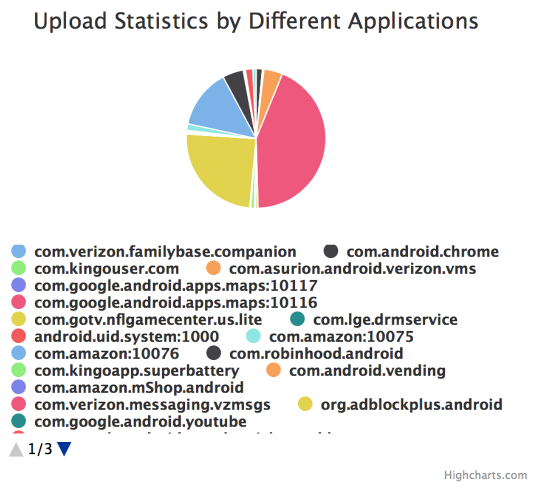


Fig. 11 Piechart for Upload Statistics

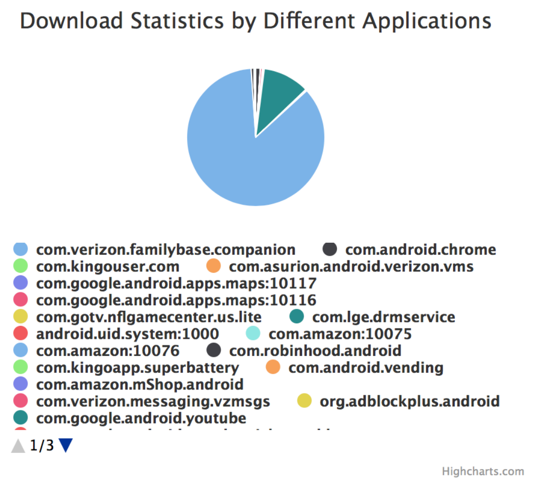


Fig. 12 Piechart for Download Statistics

Fig. 11 shows the upload statistics for different applications while Fig. 12 shows the download statistics. With these two pie charts, we can find out which application consumes the most data. For example, if we find that YouTube or Google Map uses most data, they may need networks with high bandwidth.

We also created the times series of download and upload statistics for different applications, as shown in Fig. 13 and Fig. 14. The library used to draw these figures is Highcharts [7]. It turned out that this library did some smoothing while drawing these figures. We stored the application traffic data in a table named "appdata" and the actual data we collected can be seen in Fig 15, which is the traffic data of using YouTube in one hour.

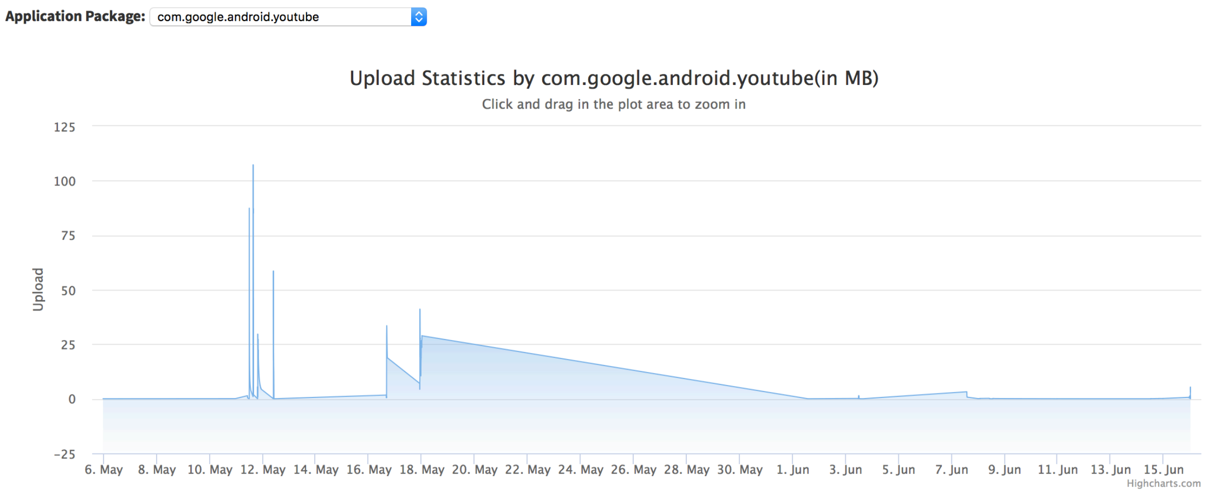


Fig. 13 Time Series for Upload Statistics of Youtube

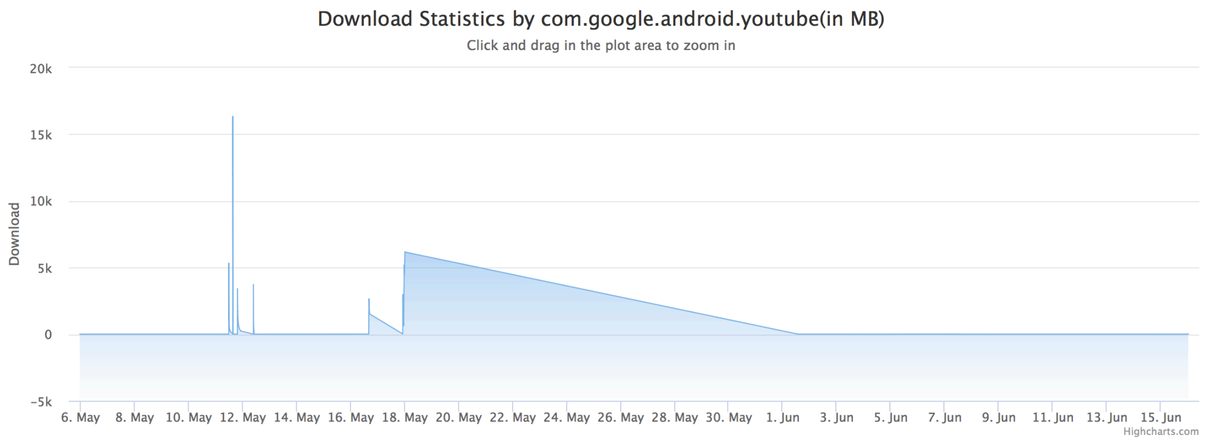
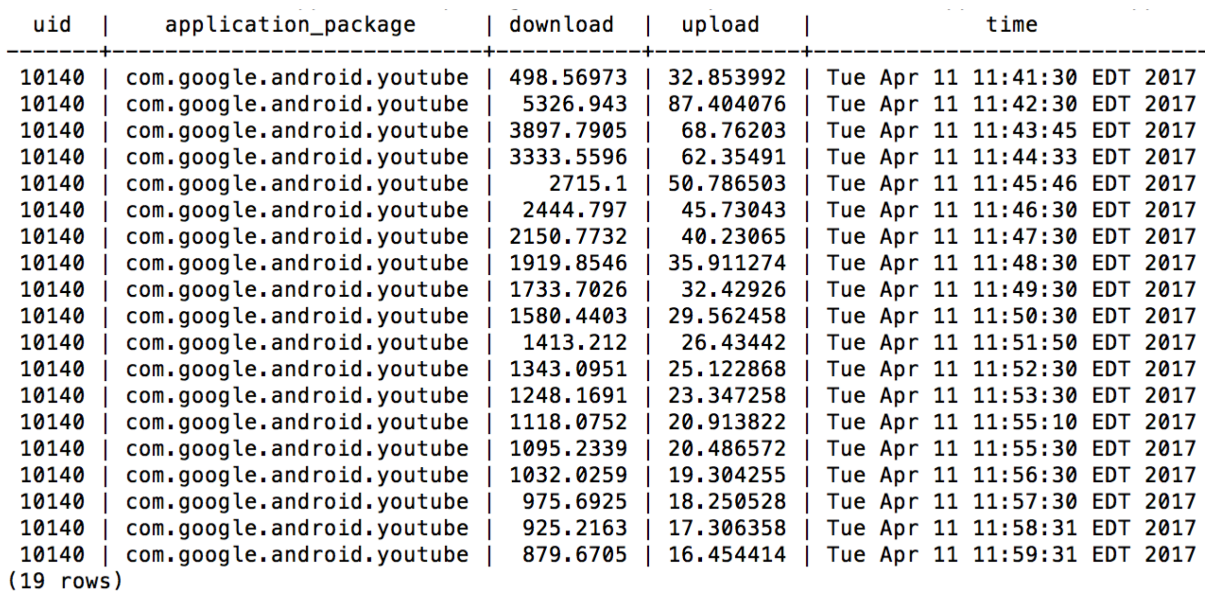


Fig. 14 Time Series for Download Statistics of Youtube



```
  uid  |    application_package     | download  |  upload   |             time
-------+----------------------------+-----------+-----------+------------------------------
 10140 | com.google.android.youtube | 498.56973 | 32.853992 | Tue Apr 11 11:41:30 EDT 2017
 10140 | com.google.android.youtube |  5326.943 | 87.404076 | Tue Apr 11 11:42:30 EDT 2017
 10140 | com.google.android.youtube | 3897.7905 |  68.76203 | Tue Apr 11 11:43:45 EDT 2017
 10140 | com.google.android.youtube | 3333.5596 |  62.35491 | Tue Apr 11 11:44:33 EDT 2017
 10140 | com.google.android.youtube |    2715.1 | 50.786503 | Tue Apr 11 11:45:46 EDT 2017
 10140 | com.google.android.youtube |  2444.797 |  45.73043 | Tue Apr 11 11:46:30 EDT 2017
 10140 | com.google.android.youtube | 2150.7732 |  40.23065 | Tue Apr 11 11:47:30 EDT 2017
 10140 | com.google.android.youtube | 1919.8546 | 35.911274 | Tue Apr 11 11:48:30 EDT 2017
 10140 | com.google.android.youtube | 1733.7026 |  32.42926 | Tue Apr 11 11:49:30 EDT 2017
 10140 | com.google.android.youtube | 1580.4403 | 29.562458 | Tue Apr 11 11:50:30 EDT 2017
 10140 | com.google.android.youtube |  1413.212 |  26.43442 | Tue Apr 11 11:51:50 EDT 2017
 10140 | com.google.android.youtube | 1343.0951 | 25.122868 | Tue Apr 11 11:52:30 EDT 2017
 10140 | com.google.android.youtube | 1248.1691 | 23.347258 | Tue Apr 11 11:53:30 EDT 2017
 10140 | com.google.android.youtube | 1118.0752 | 20.913822 | Tue Apr 11 11:55:10 EDT 2017
 10140 | com.google.android.youtube | 1095.2339 | 20.486572 | Tue Apr 11 11:55:30 EDT 2017
 10140 | com.google.android.youtube | 1032.0259 | 19.304255 | Tue Apr 11 11:56:30 EDT 2017
 10140 | com.google.android.youtube |  975.6925 | 18.250528 | Tue Apr 11 11:57:30 EDT 2017
 10140 | com.google.android.youtube |  925.2163 | 17.306358 | Tue Apr 11 11:58:31 EDT 2017
 10140 | com.google.android.youtube |  879.6705 | 16.454414 | Tue Apr 11 11:59:31 EDT 2017
(19 rows)
```

Fig. 15 Application Traffic Data of Using YouTube in one hour

As you can see, there is a dropdown button named "Application Package" in Fig. 13, you can choose different applications installed on Android phones, and our system will show the time series of upload and download statistics for this application. Now we chose YouTube, so the statistics is shown for YouTube. In these time series, we can find when the user uses this application and how much data this application consumes when being used. In other words, we can profile the application as well as the user: if we can find some pattern that user always watches YouTube at 11:00 pm, then in the future, when it is 11:00 pm, our policy can decide to switch to a network that is most suitable for YouTube. Currently, we didn't use this for network decision.

3.2.2. Backend APIs

To support different types of analysis and visualization, we built several APIs in the backend, here are some of them:

1) /network/getall

   Description: Get the data of all collected network

   Return Type: JSON

   Return: {

   "networks": [network, network,…]

   }

2) /network/bydeviceid

   Description: Get the network collected by the device with this device ID

   Parameter: Device ID

Return Type: JSON

Return: {

"networks": [network, network,…],

"device_id": device_id_param

}

3) /network/byssid

Description: Get the network that has the input SSID

Parameter: SSID

Return Type: JSON

Return: {

"networks": [network, network,…],

"ssid": ssid

}

4) /network/bylocation

Description: Get the network around input location

Parameter: location(longitude and latitude)

Return Type: JSON

Return: {

"networks": [network, network, …],

"location": location

}

5) /network/getalllocation

Description: Get all locations where network has been collected

Return Type: JSON

Return: {

"loaction": [location, location, …]

}

6) /network/getallssid

Description: Get the SSID of all collected network

Return Type: JSON

Return: {

"ssid": [ssid, ssid, …]

}

7) /network/getalldevice

   Description: Get all device IDs that have collected network data

   Return Type: JSON

   Return: {

   "device_id": [device_id, device_id, …]

   }

8) /appdata/getall

   Description: Get all application data

   Return Type: JSON

   Return: {

   "appdata": [appdata, appdata, …]

   }

9) /appdata/bydeviceid

   Description: Get the application data of one device specified by device id

   Parameter: Device ID

   Return Type: JSON

   Return: {

   "appdata": [appdata, appdata, …],

   "device_id": device_id

   }

10) /appdata/byuid

   Description: Get the network traffict data of one application specified by uid

   Parameter: UID

   Return Type: JSON

   Return: {

   "appdata": [appdata, appdata, …],

   "uid": uid

   }

11) /appdata/getbyapplicationpackage

   Description: Get the network traffic data of one application specified by application package

   Parameter: Application Package

   Return Type: JSON

```
Return: {
        "appdata": [appdata, appdata, …],
        "application_package": application_package
}
```

12) /appdata/downloadstats

Descriptions: Get percentage of download data by different applications

Return Type: JSON

```
Return: {
        app: percentage,
        app: percentage,
        ...
 }
```

13) /appdata/uploadstats

Descriptions: Get percentage of upload data by different applications

Return Type: JSON

```
Return: {
        app: percentage,
        app: percentage,
        ...
 }
```

14) /appdata/getallapplication

Description: Get all the applications that consumes network data

Return Type: JSON

```
Return: {
        "application_package": [application_package, application_package, …]
}
```

3.3.Build a Prototype for Policy Engine

In our implementation, policy engine mainly consists of three parts:

1) AddPolicyActivity Class for the users to define their own user policies
   public class AddPolicyActivity extends Activity {...}
2) PolicyEngine Class that collects inputs (application and location data) of policy engine
   public class PolicyEngine extends Service {...}

3) ApplicationDecision Class making the network switch decision based on the inputs
public class ApplicationDecision extends IntentService{...}

We set AddPolicyActivity Class to be an Android Activity since this class acts as an UI that interact with the users and accepts the input policy from the users, which must run in the foreground. Therefore, extending Android Activity can best satisfy our needs. We set PolicyEngine Class and ApplicationDecision Class to be Android Service due to the reason that these two classes should run in the background of Android System and continuously listen to the changes of the cellphone, and that is the feature of Android Service. For example, when user moves to a new location, PolicyEngine Service should detect the coordinate change immediately and triggers the decision making process. Once ApplicationDecision Service listens to this trigger of decision making process, it will make network switch decision based on the user policies defined through the UI previously.

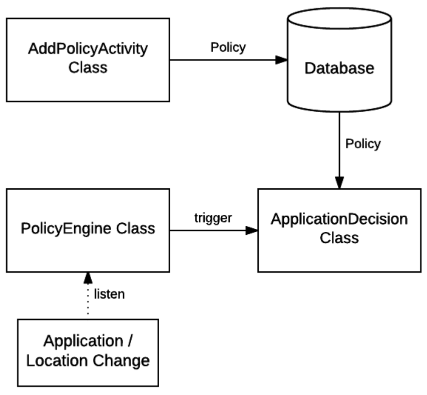


Fig. 16 Android Implementation of Policy Engine

4. Result and Analysis

4.1. Architecture for Automatic Data Collection and Analysis

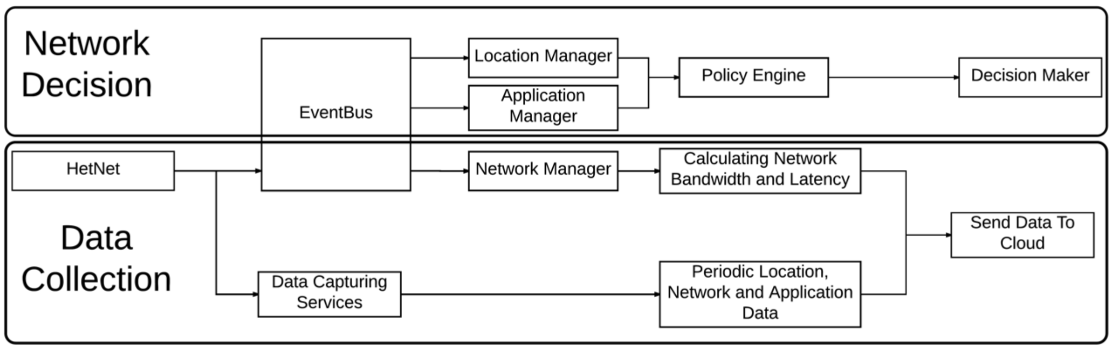


Fig. 17 Current HetNet

The current HetNet architecture is shown above. The complete middleware is divided into two sections. One is the data collection module that would collect the data in time-based (10 seconds) and event-based (connection change). Another module is the decision module which contains location change and foreground application change triggers. These triggers are also part of the EventBus architecture. The end point for two sections are all the cloud-backend with different APIs. A cloud database built on AWS RDS stores the policy vectors and all information collected for different users. The decision maker would be triggered once the mobile device send event vector to the cloud and then do analysis based on the existing policy vector and historical network/application information. HetNet also supports Wi-Fi hard-switch when it receives the switch command sent from the server.

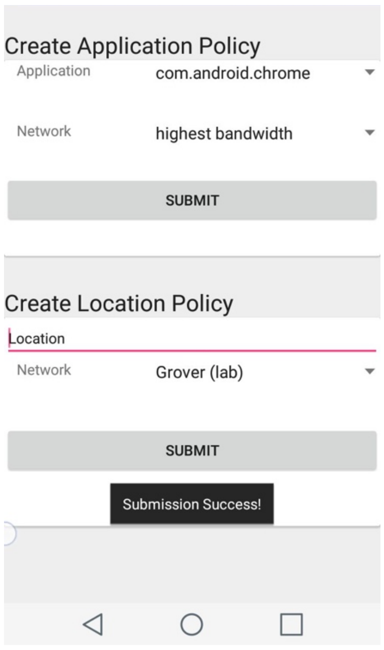
4.2.Prototype for Policy Engine

Current cellphones only support default network switch based on signal strength, which is not intelligent. The goal of our control middleware team is to provide intelligent network switch according to user policy defined by users. Currently, we are able to provide hard handover of Wi-Fi network when user changes location or application (Network switch is determined by the foreground application, and all the background application will use the same network as the foreground application does. This is due to the fact that currently we are not able to manage multiple routing tables.).

There are two factors that can lead to a decision of network switch: application change and location change. For better demonstration purpose, we show the results of application change on the Android phone by switching to different foreground application and see the hard network switch; and show the results of location change on our cloud side map by clicking different location on the map and see the network switch decision.

Demonstration of network switch when application change (Android side):

1) Assume a user is currently at Columbia University, and there are altogether three available networks: “Columbia University”, “Kai He’s iPhone”, “Shen’sIphone”. Since there are limited number of networks available to us around campus, we set two of our iPhones as hotspot for demo purpose (“Kai He’s iPhone” and “Shen’sIphone”).
2) In this example, the user will use Chrome, YouTube and Google Maps one after another, in the same location. For best user experience, when using Chrome, the user wants to have the highest bandwidth; when using YouTube, the user wants to have the lowest latency; when using Google Maps, the user wants to just use “Columbia University” network.


Create Application Policy

Application	com.android.chrome
Network	highest bandwidth

SUBMIT

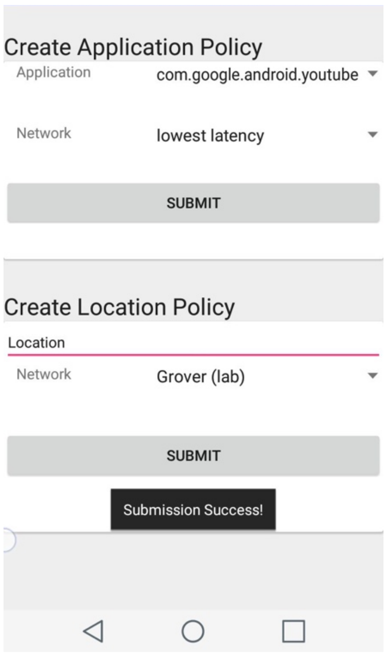
Create Location Policy

Location	
Network	Grover (lab)

SUBMIT

Submission Success!

Fig. 18 Set policy that Chrome should use network with highest bandwidth


Create Application Policy

Application	com.google.android.youtube
Network	lowest latency

SUBMIT

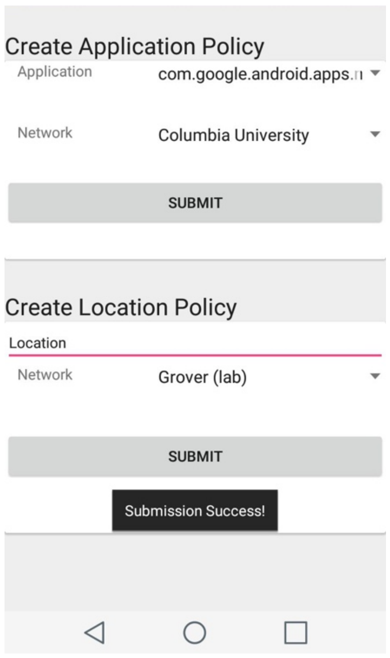
Create Location Policy

Location	
Network	Grover (lab)

SUBMIT

Submission Success!

Fig. 19 Set policy that YouTube will use network with lowest latency



Create Application Policy

Application com.google.android.apps.n

Network Columbia University

SUBMIT

Create Location Policy

Location

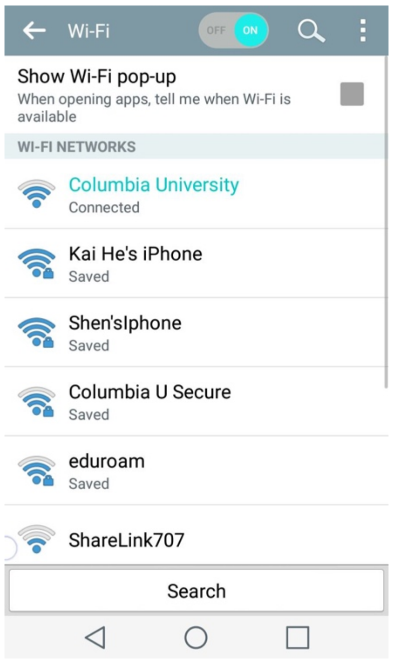
Network Grover (lab)

SUBMIT

Submission Success!

Fig. 20 Set policy that Google Maps should use Columbia University network

3) In the very beginning, the user is using “Columbia University” network. In order to check the effectiveness of our policy engine, then the user starts to use Chrome, YouTube and Google Maps one by one, and we check whether policy engine will switch network for the user according to his/her policy defined above.



Wi-Fi

Show Wi-Fi pop-up
When opening apps, tell me when Wi-Fi is available

WI-FI NETWORKS

Columbia University
Connected

Kai He's iPhone
Saved

Shen'sIphone
Saved

Columbia U Secure
Saved

eduroam
Saved

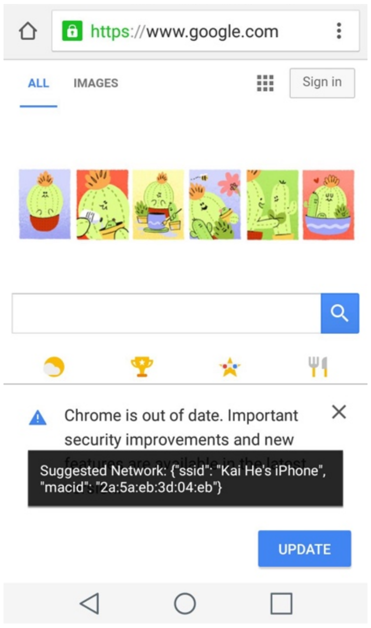
ShareLink707

Search

Fig. 210 Use Columbia University in the very beginning

4) When the user switch to Chrome on the foreground, policy engine makes the decision to switch to network with highest bandwidth. This information of which network has the highest bandwidth has been collected in the data stored in the data collection phase and stored in the

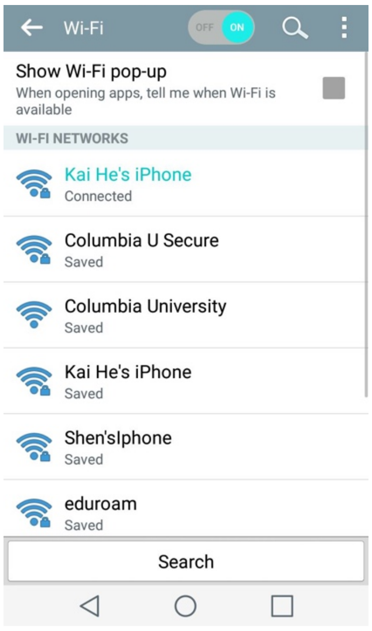
cloud database. Therefore, the policy engine will query the database, fetch the network information and connect to that network ("Kai He's iPhone" in this case).



Chrome is out of date. Important security improvements and new

Suggested Network: {"ssid": "Kai He's iPhone", "macid": "2a:5a:eb:3d:04:eb"}

Fig. 22 When starting to use Chrome, network is switched to network with highest bandwidth



Show Wi-Fi pop-up
When opening apps, tell me when Wi-Fi is available

WI-FI NETWORKS

Kai He's iPhone
Connected

Columbia U Secure
Saved

Columbia University
Saved

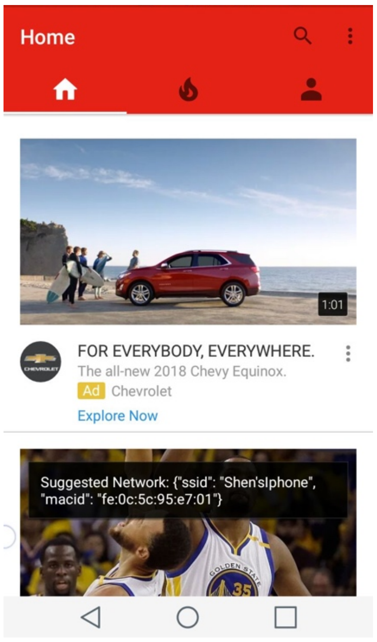
Kai He's iPhone
Saved

Shen'sIphone
Saved

eduroam
Saved

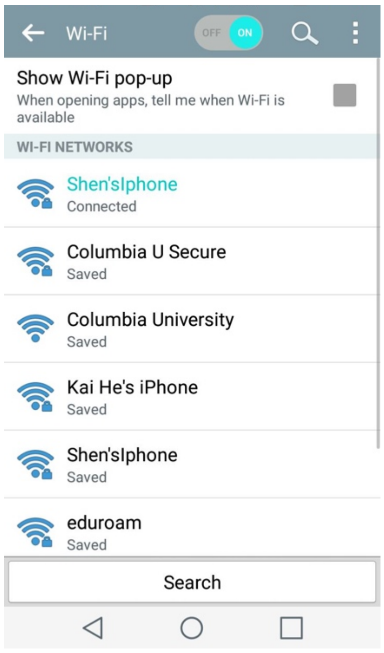
Fig. 23 Android system shows that we are indeed using "Kai He's iPhone" network

5) When the user switch to YouTube on the foreground, policy engine makes the decision to switch to network with lowest latency. This information of which network has the lowest latency has been collected in the data stored in the data collection phase and stored in the cloud database. Therefore, the policy engine will query the database, fetch the network information and connect to that network ("Shen'sIphone" in this case).



Suggested Network: {"ssid": "Shen'sIphone", "macid": "fe:0c:5c:95:e7:01"}

Fig. 24 When starting to use YouTube, network is switched to network with lowest latency



Show Wi-Fi pop-up
When opening apps, tell me when Wi-Fi is available

WI-FI NETWORKS

Shen'sIphone	Connected
Columbia U Secure	Saved
Columbia University	Saved
Kai He's iPhone	Saved
Shen'sIphone	Saved
eduroam	Saved

Fig. 25 Android system shows that we are indeed using "Shen'sIphone" network

6) When the user switch to Google Maps on the foreground, policy engine makes the decision to switch back to "Columbia University" network.

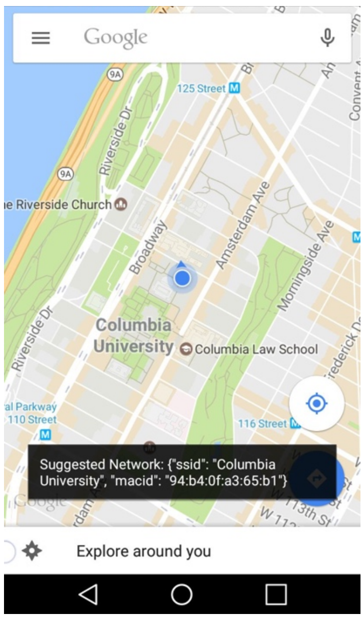

Suggested Network: {"ssid": "Columbia University", "macid": "94:b4:0f:a3:65:b1"}

Fig. 26 When starting to use Google Maps, network is switched to "Columbia University" network

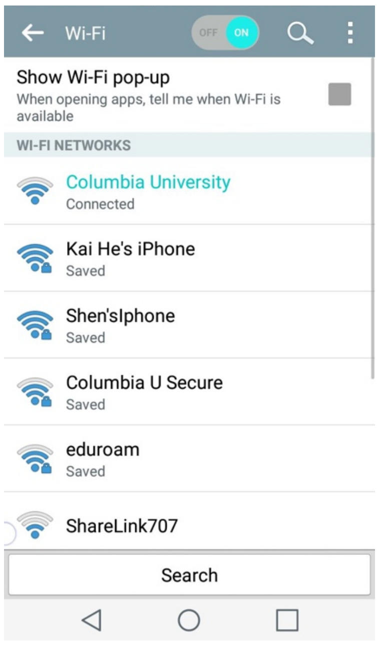

Wi-Fi
Show Wi-Fi pop-up
When opening apps, tell me when Wi-Fi is available
WI-FI NETWORKS
Columbia University
Connected
Kai He's iPhone
Saved
Shen'sIphone
Saved
Columbia U Secure
Saved
eduroam
Saved
ShareLink707
Search

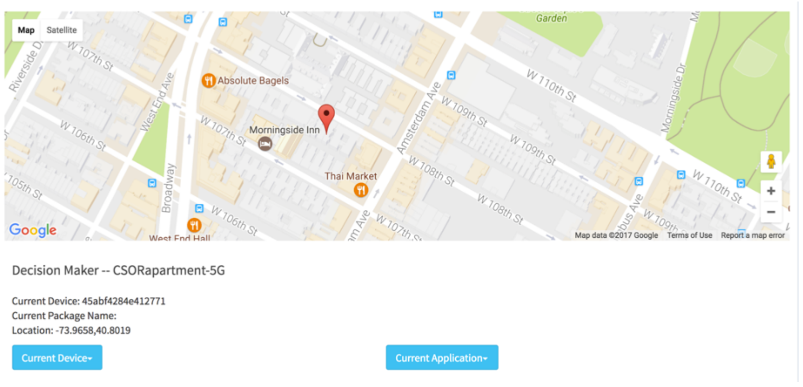
Fig. 27 Android system shows that we are indeed using "Columbia University" network

7) This example shows that the policy engine can switch networks for users according to the application-based policies defined by them. This switch process is triggered when the foreground application changes, and all the background applications will use the newly switched network too. For example, if the user starts to use Google Maps in the foreground, and policy engine switch to “Columbia University” network, all the background application such as Gmail and Chrome will also use this network. Application-specific network switch is not achieved since we still cannot manage multiple routing tables in Android phone. This could be finished in the future semester of this project.

We also recorded a video to demo this part, here is link for this video:
https://youtu.be/bMJkRZKcovk

The following is the demonstration of network switch when location change (cloud side):

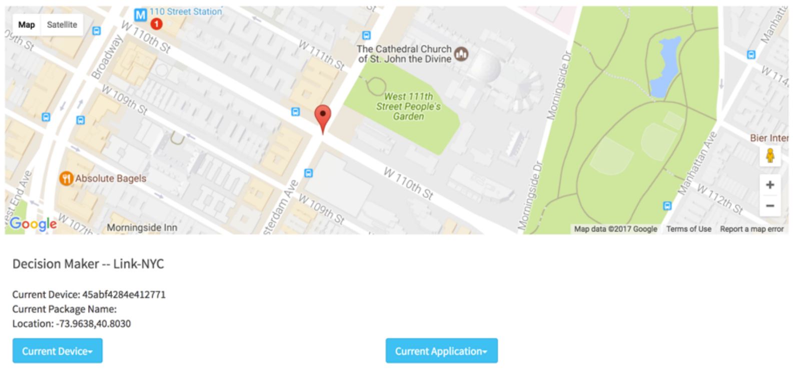
1) Assume a user walks from home to Columbia University every day and there are five available networks along the route: "CSORapartment-5G", "Link-NYC", "Costa-Conf", "linksys" and "Columbia University".
2) In this example, the user will start from 204 West 108th Street, New York and walk along Amsterdam Avenue to Columbia University. Since the user repeats this route every day, he/she knows what networks are available and has the best user experience to him/her. He/she can set user policy through our user interface and this information can be saved in our cloud database.
3) Assume the user wants to use "CSORapartment-5G" at home, "Link-NYC" around 110th Street, "Costa-Conf" around 113th Street, "linksys" around 116th Street and "Columbia University" inside campus. Then we simulate the policy engine decision when user travels from home to Columbia University.
4) When the user is at home, policy engine makes the decision that network should be switch to "CSORapartment-5G".


Decision Maker -- CSORapartment-5G

Current Device: 45abf4284e412771
Current Package Name:
Location: -73.9658,40.8019

Fig. 28 Policy engine makes decision to switch to "CSORapartment-5G" network at home according to user policy

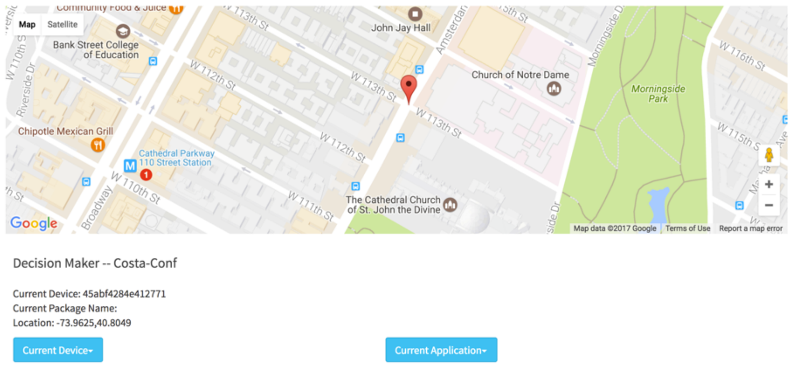
5) When the user is at 110th Street, Amsterdam Avenue, policy engine makes the decision that network should be switch to "Link-NYC".


Decision Maker -- Link-NYC

Current Device: 45abf4284e412771
Current Package Name:
Location: -73.9638,40.8030

Fig. 29 Policy engine makes decision to switch to "Link-NYC" network at 110th Street according to user policy

6) When the user is at 113th Street, Amsterdam Avenue, policy engine makes the decision that network should be switch to “Costa-Conf”.

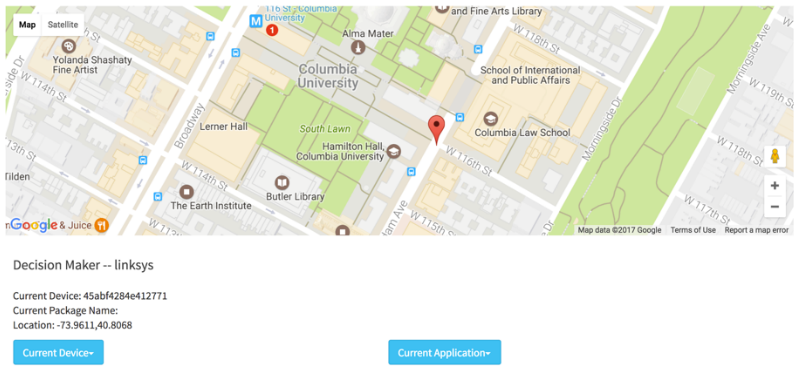


Decision Maker -- Costa-Conf

Current Device: 45abf4284e412771
Current Package Name:
Location: -73.9625,40.8049

Fig. 30 Policy engine makes decision to switch to "Costa-Conf" network at 113th Street according to user policy

7) When the user is at 116th Street, Amsterdam Avenue, policy engine makes the decision that network should be switch to “linksys”.

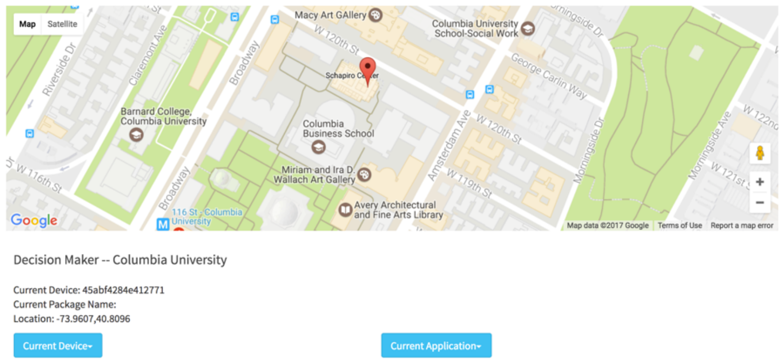


Decision Maker -- linksys

Current Device: 45abf4284e412771
Current Package Name:
Location: -73.9611,40.8068

Fig. 311 Policy engine makes decision to switch to "linksys" network at 116th Street according to user policy

8) When the user is inside Columbia University, policy engine makes the decision that network should be switch to “Columbia University”.



Decision Maker -- Columbia University

Current Device: 45abf4284e412771
Current Package Name:
Location: -73.9607,40.8096

Fig. 312 Policy engine makes decision to switch to "Columbia University" network inside campus according to user policy

9) This example shows that the policy engine can switch networks for users according to the location-based policies defined by them. This switch process is triggered when coordinate of the user changes, and the switched network will be applied to all the background application

too.

5. Discussion

5.1. Project Outcome

In this semester, we have built a complete Android-cloud data collection pipeline and a prototype of policy engine.

The data collection pipeline consists of Android side and cloud side. Android side runs as an Android service in the background that continuously collect users' network data as well as application data and send them to the cloud side. The cloud side receives the data and save data in the database. Also, the cloud side can analyze data and tell which network is most suitable for users. This whole pipeline is complete and stable. If other types of data need to be collected, they can easily be incorporated into this pipeline with the least modification.

Another outcome of the project is the prototype of policy engine, which can make decisions for the users based on simple policy such as using networks with highest bandwidth and lowest latency. First, we design a user interface for users to input their own policy according to their needs. Then it can fetch the network information in the database through our cloud and know which network should be switched to for the user. Finally, we implemented the hard network switch based on our decision made for the user.

In short, this semester we build a foundation for control middleware team. The above outcomes can be extended to more complicated usage without many changes in architecture. Future students in control middleware team do not need to bother with issues like how to collect user data, how to store data and how to let users define their policy. Instead, they can immediately start working on how to design the most suitable policy that can best satisfy users' network needs.

5.2. Security and Privacy Analysis

5.2.1. Security

Currently, we have built the sign up and log in system on the cloud server to validate the users' identities. We are using JSON format to send the users' account and password information, which could be sniffed and attacked by hackers. The ideal way to deal with this issue should be generating an authentication token on the client side in the login process and sending the token to the server. The token can be used as a session key in header or cookie field or as a part of the data. It should automatically expire in a certain time period. The connection between cloud backend and database are secure since it happened on the AWS internal networks through the authentication tool provided by Amazon. The database is protected by the username and password. There is no open port (can be accessed without username/password) from cloud backend to the database.

5.2.2. User Privacy

The database will store the location and network information in the areas where the user has visited. This information would not be shared with other users. However, the user has the right to use the information on multiple devices as long as the credential is matched. In this case, HetNet users should have the device management tools available on the cloud backend. Different devices should be registered with the unique identifier. Users also have the right to remove any information collected from the server and the information must be removed once the user no longer use the services. The cloud backend is designed to use all of the application traffic data for analysis, and during this analysis, users' personal information will not be used.

5.3. Analysis

5.3.1. Features We Discarded

In the final model, the signal strength result is not selected to make the decision. Since the Android system is using this value to do the network switch automatically, there is not much need for this data to make the decision again. Considering the fact that the bandwidth and latency for certain network in a certain location is obtained, signal strength is not collected.

5.3.2. Why Using VPN Service

As we discussed in previous part, VPN Service is a good tool to capture more comprehensive application data information which can help policy engine to make better decision. Specifically, VPN Service can not only capture upload and download data consumption, it can also tell us the source and destination of the data packets, which can help us better understand the behavior of the user. When a user is using a multi-functional application such as Chrome, knowing whether the user is watching videos or reading news makes a difference for the decision making process.

5.3.3. Variance on Bandwidth

Although the bandwidth calculation would be performed when connection changed. The variance of the data would not be measured. Hence, we evaluate the bandwidth based on three factors: location, file size and time.

Table 1 Location Test

Location	Time	File size	Bandwidth
Business School Entrance	9:10	5M	7.59 Mb/s
Business School Library	9:12	5M	6.82 Mb/s
Business School Cafe	9:15	5M	6.32 Mb/s

The first topic chosen to measure is the location. The time was chosen in the morning that fewer people would be in the library. Network used for testing is the "Columbia U Secure". Three location were at cafe area, main entrance and inside of the Library. The results shown above are the average of 5 measurements in each area. The bandwidth differences are not large.

Table 2 Test on the Time

Location	Time	File size	Bandwidth
Business Library 1	9:00	5M	12.60 Mb/s
Business Library 1	10:15	5M	10.97 Mb/s
Business Library 1	11:36	5M	7.01 Mb/s
Business Library 1	12:50	5M	5.62 Mb/s
Business Library 1	13:44	5M	5.53 Mb/s
Business Library 1	14:55	5M	6.94 Mb/s
Business Library 1	16:20	5M	8.32 Mb/s

Then we chose a Tuesday and did a couple of testing during the day. The result is as shown below: the bandwidth is strongly related to the time during a day. It may be caused by the number of people currently in the same networks. However, the historical data of bandwidth during different days may be a good model to measure the quality of the network.

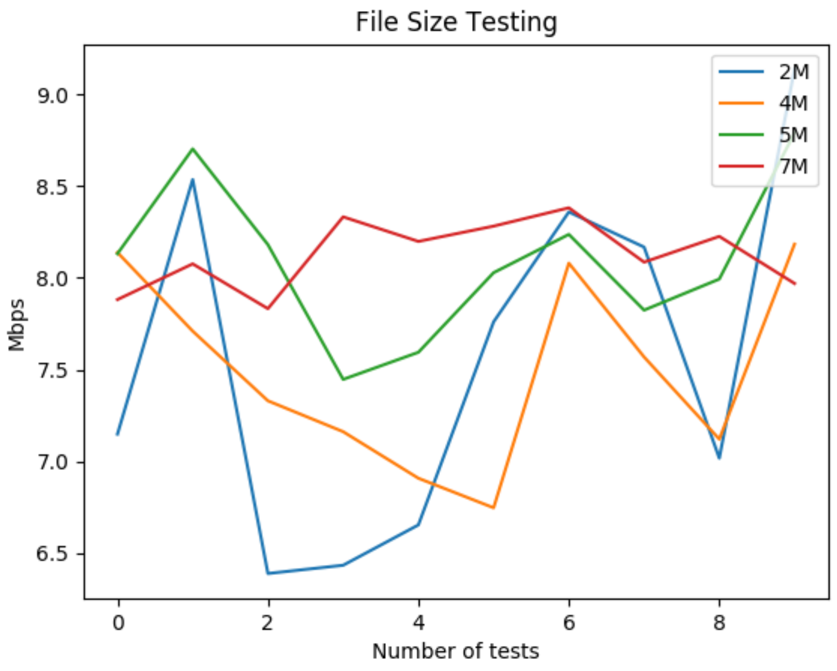


Fig. 33 Graph of Different File Size

Finally, several tests were applied on the file size: file sizes of 2MB, 4MB, 5MB and 7MB were chosen to do the measurement. All tests were run during 9:00 - 10:00 AM in the same location of library. The result suggests a larger file would likely to provide a stable bandwidth testing. As the difference between 5MB and 7MB was not significant, 5MB would be an ideal option for network bandwidth measurement.

In summary, the variance of the bandwidth is more related to the file size and different time of a day (how crowed the room would be). The location effect would not be considered for a certain network. Whenever a bandwidth will be chosen, it should also be accompanied with a timestamp.

5.3.4. Variance on Latency

Similarly, we tested latency based on location and time. The first factor we choose to measure is location. we calculated latency for network "Columbia U Secure" in the morning in three different locations, Business School Entrance, Business School Library and Business School Café. The results are shown in the following chart. We can find that there are not too many differences among them.

Table 3 Latency in Different Location

Location	Time	Latency

Business School Entrance	9:03	0.264s
Business School Library	9:11	0.218s
Business School Cafe	9:07	0.247s

The second factor we tested is time. In Business School Library, we calculated the latency for “Columbia U Secure” at different times, the results are also shown in the following chart:

Table 4 Latency in Different time

Location	Time	Latency
Business Library	9:00	0.218s
Business Library	10:15	0.200s
Business Library	11:36	0.233s
Business Library	12:50	0.228s
Business Library	13:44	0.294s
Business Library	14:55	0.374s
Business Library	16:20	0.351s

Generally speaking, the latency for network “Columbia U Secure” does not change too much, but it can be found that the latency is larger at noon and in the afternoon. It may be due to the reason that there are more people connecting to this network at noon and in the afternoon.

5.3.5. Relationship between Bandwidth and Latency

Currently, we are not making any assumptions on the relationship between bandwidth and latency. From the network data we measured at business school, it seems that there is correlation between bandwidth and latency, as shown in Table 2 and Table 4. However, this is not always the case. For example, in the demo video we recorded in Chapter 4.2, the network with highest bandwidth and lowest latency are not the same.

5.3.6. File Size

File size may influence the bandwidth testing process when fetching network information. The reason of doing experiments on different file size is to find the minimal file size that we can provide us with acceptable accuracy.

5.4. Current Issues

During the time-based data capturing process, it is not strictly triggered every 10 seconds. We set the interval to be 10 seconds so that we can gather location and application traffic data repeatedly. This “10 seconds” parameter has not been tuned rigorously by experiments.

Sometimes it could be 15 seconds and even longer due to the Wi-Fi scan process in network capturing process. The timer would only trigger a new task when the previous one is finished in order to avoid the accumulation problems happened in last semester.

The battery consumption is also an issue that user need to be aware of. Since the Wi-Fi scanning and GPS locating services are used more frequent than usual, it will influence battery usage greatly. However, once the user switch to the network suggestion mode, these services will not be used very often and the cost would be reduced. The data collection mode would consume much more battery than network suggestion mode.

6. Conclusion and Future Work

The control middleware team has finished most of the tasks mentioned in our project proposal, including maintaining the data collections as well as a draft policy engine. A universal network data capturing method is implemented to apply on both rooted and unrooted phone. The backend server is designed to connect closely with the database and provide data fetching APIs for visualizer and decision maker. A complete cloud analytic engine contains the data visualization and management is built by these components. Several analyses were applied to evaluate the variance of the parameters as well as the usage of them. Finally, the complete work would be easy to extend and scale. The visualizer tool could be reused to provide more visualization of the data. Comparing with the proposal, around 75% of the tasks were finished. The remaining part was the bridge to the seamless transaction and the complicated use cases handling based on the data now available.

The latter part requires the skill of the data analysis and algorithm building. The data should be transformed into a format that could be applied to the classification/clustering algorithm in order to identify the different condition such as, different period of a day, networks available, which foreground application is using. The future goals would be the automated network switch without any prior user defined policy. The policy engine would learn from the data to build the user specific policy on the user account. It should also have the ability to label the data-consuming application and most frequent used application during different time of a day for certain user. The application data usage that we collected for now may have a strong impact on the automated learning of the policy engine.

7. Reference

[1] Singh, A., Ormazabal, G., Schulzrinne, H., Zou, Y., Thermos, P., & Addepalli, S. (2013, October). Unified heterogeneous networking design. In Proceedings of Principles, Systems and Applications on IP Telecommunications (pp. 1-7). ACM.

[2] [Online]. Available: http://greenrobot.org/eventbus/.

[3] [Online]. Available: https://en.wikipedia.org/wiki/Bandwidth_(computing).

[4] [Online]. Available: https://en.wikipedia.org/wiki/Latency_(engineering).

[5] [Online]. Available: https://android.googlesource.com/platform/bionic/.

[6] [Online]. Available: https://developer.android.com/reference/android/net/VpnService.html.

[7] [Online]. Available: https://www.highcharts.com/.

8. Appendix

We put the codes we've written in this part, they can be divided into two parts: Android side and cloud side.

8.1. Android Side

GitHub Repository: https://github.com/lanking520/HetNet

Our contribution to the repo in this semester:

- HetNet/app/src/main/java/android_network/hetnet/application/ApplicationDecision.java
- HetNet/app/src/main/java/android_network/hetnet/cloud/AppDataService.java
- HetNet/app/src/main/java/android_network/hetnet/cloud/SendCloud.java
- HetNet/app/src/main/java/android_network/hetnet/cloud/HttpService.java
- HetNet/app/src/main/java/android_network/hetnet/cloud/PostTask.java
- HetNet/app/src/main/java/android_network/hetnet/data/NetoworkEvaluation.java
- HetNet/app/src/main/java/android_network/hetnet/data/PolicyEngineData.java
- HetNet/app/src/main/java/android_network/hetnet/location/LocationFetcher.java
- HetNet/app/src/main/java/android_network/hetnet/network/NetworkListFetcher.java
- HetNet/app/src/main/java/android_network/hetnet/networkcap/ConnectionEvalFetcher.java
- HetNet/app/src/main/java/android_network/hetnet/networkcap/ConnectionListener.java
- HetNet/app/src/main/java/android_network/hetnet/networkcap/ConnectionResponseEvent.java
- HetNet/app/src/main/java/android_network/hetnet/networkcap/ConnectionTriggerEvent.j

ava
- HetNet/app/src/main/java/android_network/hetnet/policy_engine/PolicyEngine.java
- HetNet/app/src/main/java/android_network/hetnet/vpn_service/ServiceSinkhole.java
- HetNet/app/src/main/java/android_network/hetnet/vpn_service/AcitivityTraffic.java
- HetNet/app/src/main/java/android_network/hetnet/vpn_service/DatabaseHelper.java
- HetNet/app/src/main/java/android_network/hetnet/AddPolicyActivity.java
- HetNet/app/src/main/java/android_network/hetnet/MainActivity.java

8.2.Cloud Side

GitHub Repository: https://github.com/lanking520/HetNet_Cloud
Our contribution to the repo in this semester:

- HetNet_Cloud/webserver/application.py
- HetNet_Cloud/webserver/table.py
- HetNet_Cloud/webserver/routes/appdata.py
- HetNet_Cloud/webserver/routes/event.py
- HetNet_Cloud/webserver/routes/index.py
- HetNet_Cloud/webserver/routes/login.py
- HetNet_Cloud/webserver/routes/network.py
- HetNet_Cloud/webserver/routes/sampledata.txt
- HetNet_Cloud/frontend/index.html
- HetNet_Cloud/frontend/css/signin.css
- HetNet_Cloud/frontend/home/applicationData.html
- HetNet_Cloud/frontend/home/applicationDataVisualization.html
- HetNet_Cloud/frontend/home/decisionMaker.html
- HetNet_Cloud/frontend/home/main.html
- HetNet_Cloud/frontend/home/networkData.html
- HetNet_Cloud/frontend/home/networkDataVisualization.html
- HetNet_Cloud/frontend/js/highcharts.js
- HetNet_Cloud/frontend/js/login.js
- HetNet_Cloud/frontend/js/main.js
- HetNet_Cloud/frontend/js/markerclusterer.js
- HetNet_Cloud/frontend/js/ng-map.min.js
- HetNet_Cloud/frontend/login/logintable.html
- HetNet_Cloud/frontend/login/registertable.html