CSE 622 Project Report of End-to-end Connectivity on Phones

Abstract

Recent studies have shown that data usage in mobile phones has increased and browsing contributes to over half of the traffic. This finding is coupled with the fact that recent applications tend to use HTTP protocol for most of their network operations, making HTTP the greatest share in mobile data traffic. On the other side, although today's phones are equipped with both cellular and Wi-Fi interfaces, only one is maintained and available for applications at a time. Transition between two interfaces becomes a potential problem impacting application network activities and user experiences. Given these findings, we present an implementation that uses Wi-Fi and Cellular interfaces simultaneously to provide a better network operations for user applications based on HttpURLConnection library on Android platform. By utilizing both interfaces, we believe a better user experience can be provided in terms of seamless network handover, fast downloading speed and better optimization on energy consumption.

1. Introduction

In recent years, cellular networks deployment is increasing, along with the quality of service they provide and Wi-Fi networks are also growing with the same trend. The ever increasing smartphone users are more and more relying on 3G/4G and Wi-Fi networks for daily activities. Although most phones are equipped with both cellular and Wi-Fi interfaces, in the current design, only one interface can be available at a time. With Wi-Fi being the more preferable interface, mobile network cannot be used even if both are available. This design simplifies the implementation of network functionalities on phones but may impact user experiences in many scenarios. One case worth mentioning is when a user using Wi-Fi decides to step out of the AP's range; in the traditional implementation, the phone would be responsible to detect the unavailability of Wi-Fi, turn on the cellular interface, update all the network entries, e.g., DNS, routing table, etc. and then restart the communications via cellular interface. This procedure adds to the switching delay, between multiple interfaces, which is experienced by most mobile users.

With the increasing power of smartphones, it seems reasonable to utilize both interfaces simultaneously and reduce the load on any single network. An obvious gain is bandwidth, noticeable especially when downloading large objects. More importantly, simultaneous cellular and Wi-Fi connections will give users better experience on network connectivity. With that in mind, we present a novel idea of using both cellular and Wi-Fi interfaces on Android phones. In terms of protocols used by applications, study [1] shows that HTTP plus HTTPS take up more than 80% of the traffic on smartphones and study [2] shows that HTTP plus HTTPS contribute about 97% handheld traffic in campus Wi-Fi networks. They demonstrate that HTTP protocol dominate today's data traffic on smartphones. Therefore, we focus on HTTP protocol in this project and implement simultaneous cellular and Wi-Fi connections in HTTP library. On the one hand, such kind of implementation benefits the applications which contribute the most traffic in smartphones; on the other hand, it is transparent to app developers and possible to satisfy applications using HTTP without any modifications.

A question regarding such implementation is the usefulness. With the majority of traffic being dominated by browsing, one would question the purpose of using both interfaces for downloading web pages. The answer is, on the one hand, according to httparchive.org, the average size for the top 1000 sites on the internet has increased by 35% from last year; on the other hand, based on recent research studies, other applications are also making extensive use of HTTP for communication, even though their purpose is not browsing – an interesting trend is the increase in multimedia streaming with HTTP, and the increase in the average web page size. With the increased wealth of information in the web and the increase in subjects utilizing them, we believe more applications would benefit from our implementation.

A couple of challenges and questions become inherent in using both interfaces simultaneously. One is decisions and policies, regarding when to use both interfaces and switching, and how to choose download chunk sizes for both interfaces; the other one would be energy consumption. Studies [4] and [5] show that the energy inefficiency of these networks, with the most noticeable being mobile networks. However, the energy efficiency also depends on network dynamics; a possible tradeoff between cellular and Wi-Fi networks may exist in practice. Efficient power usage by mobile and Wi-Fi networks is an ongoing research. In the future, we plan to utilize energy measurement results to make interface selections on the fly. Energy efficient concern can be included in the policies, along with other considerations, for making decisions.

The rest of the report is organized as follows. We talk about our system design in Section 2. Section 3 describes our implementation details. Section 4 presents the screen shot of running our implementations. Decisions and policies regarding when to use both interfaces, and switching, are part of our future plan implementation and are briefly discussed in Section 5.

2. System Design

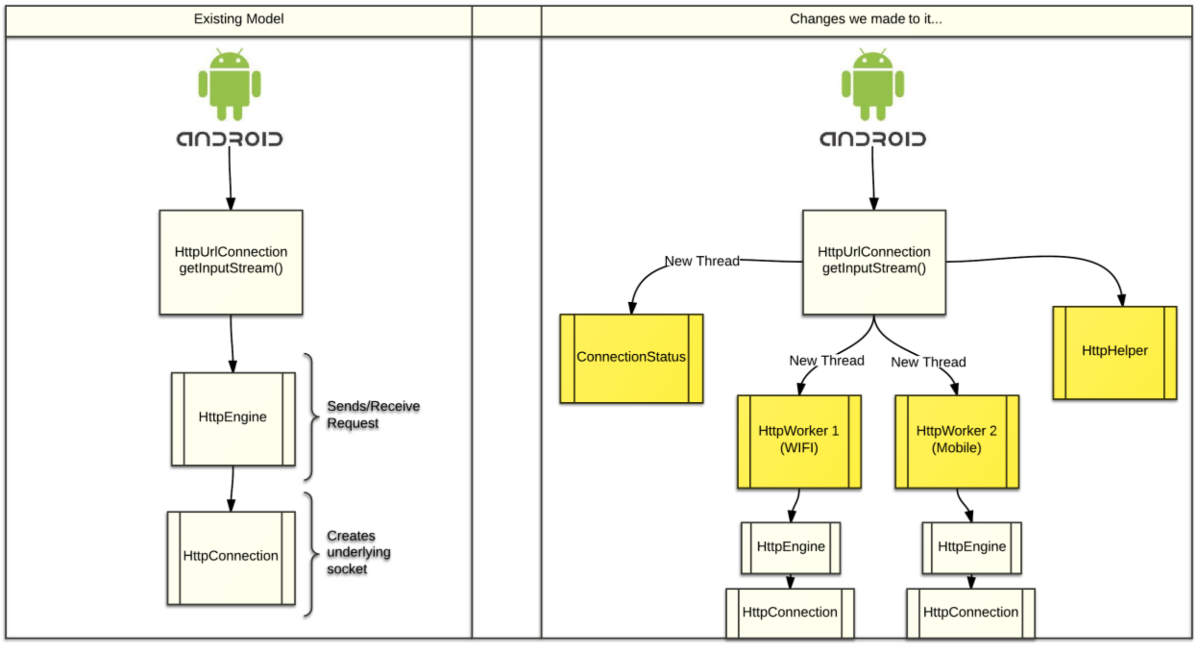


Figure 1. HttpURLConnection redesign compared with original design

Our implementation includes changes in the Core java library – libcore and Android framework library system_server. In Android framework, we modify ConnectivityService to enable both interfaces simultaneously and add two sub routing table to manage two interfaces independently. libcore is where app's HTTP request connection is handled. An HttpURLConnection object is used by the app to initiate the request and response reading, which takes place in the library. Figure 1 below shows the overall steps taken for Http communication on Android phones when using the above class. Originally, the implementation does not specify any local address, and would simply create socket not binded to any IP. In our design, we specify which interface to bind each worker with at the time of HttpURLConnection creation. The HttpHelper, among other purposes, holds the request byte range information. In our current implementation, each interface associated worker sends requests using a defined constant byte range, and checks with the helper for updating its request header with the next chunk to ask for. The workers maintain individual http connection, and implement simple work stealing to obtain the full file from servers. A dynamic size request, by utilizing information such as interface performance is something that could be implemented to enhance the performance.

3. Implementation

3.1 Implementation in Kernel

In order to create additional routing tables in kernel, we used iproute2 tool. First, we created iproute2 directory in /etc (which is /system/etc). To do this, we remounted /system as read-write as it is write protected. Second, we placed "rt_tables" file in /etc/iproute2 directory. This file contains two new entries corresponding to our routing tables and this file is read by kernel dynamically.

3.2 Implementation in Framework

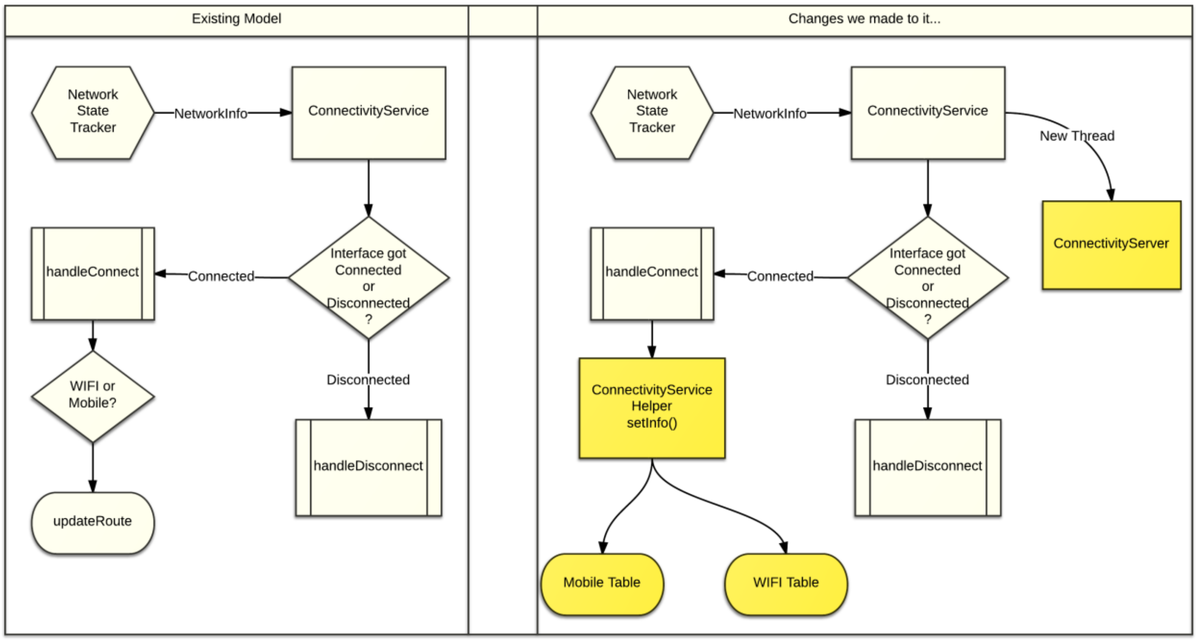


Figure 2. Network connection handler inside the Framework, and modifications.

Once we have created routing tables, the next important task was to populate these tables along with the "main" routing table whenever interfaces go up/down. Figure 2 shows the changes needed on the Framework's connection handler class, ConnectivityService. This is part of the framework where interface setup/teardown, default routing table setup, and other network configurations are taken care of. In our implementation, we modified the current implementation such that MOBILE interface would not be torn down even if WIFI interface is up and connected. Also, in order to update the individual routing tables, we used iproute2's "ip" utility. We added a native module that runs the "ip" command with "root" user privileges using "su" utility. This is needed, as any modification to routing tables requires "root" access and since all these commands were issued from ConnectivityService that runs as "AID_SYSTEM" user, we hacked in to "su" implementation so that "AID_SYSTEM" can use "su" utility apart from "AID_ROOT" and "AID_SHELL" user.

3.3 Implementation in libcore

In figure 1, ConnectionStatus is a thread that is initialized before sending requests and continuously checks, using a socket, with the framework's network connection handler (ConnectivityServer) to get the status for each interface. We used sockets instead of wide-range of available Android IPC mechanisms like intents, looper-handler, binder etc. because Android's framework is build on top of libcore library. So, one cannot import any of the Android packages from libcore but can do the opposite. The workers upon receiving an update about the status of their interface from ConnectionStatus proceed accordingly. When interface goes down, they go to sleep and wait for signal from ConnectionStatus. When the interface becomes available, they check with the HttpHelper and continue the download if it is still on going. Figure 3 below shows the "Work Stealing" logic used by workers to download a particular chunk of file. Each worker fetches the next chunk to be downloaded from HttpHelper and inserts it to a synchronized map. Thus, whichever interface (Worker) is up and fast fetches most of the chunks.

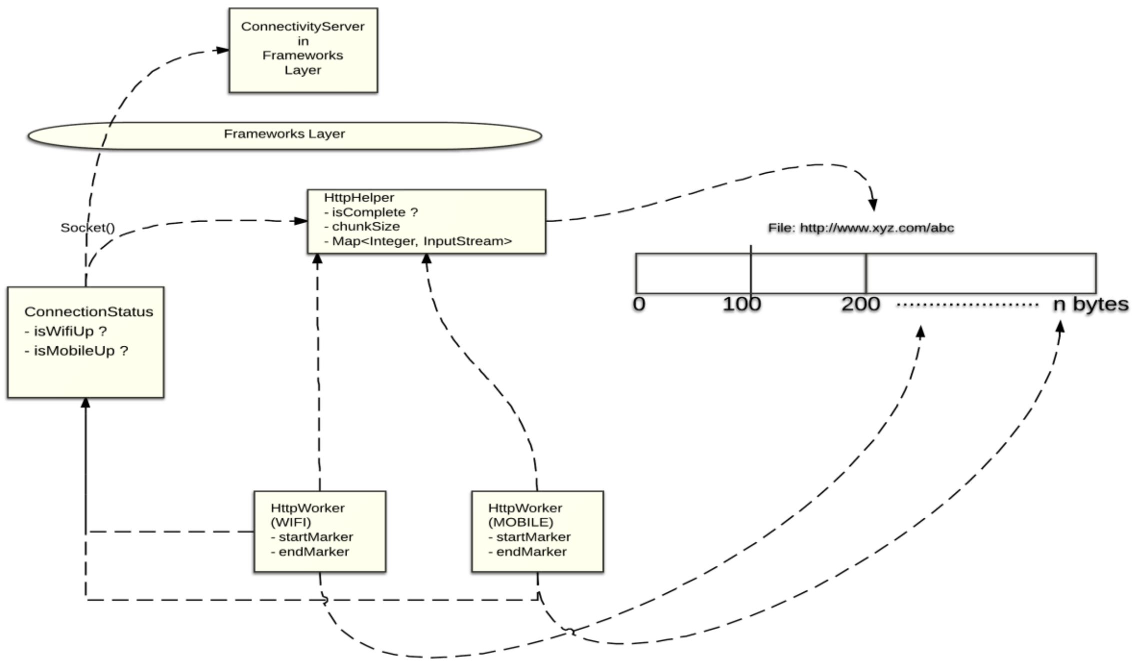
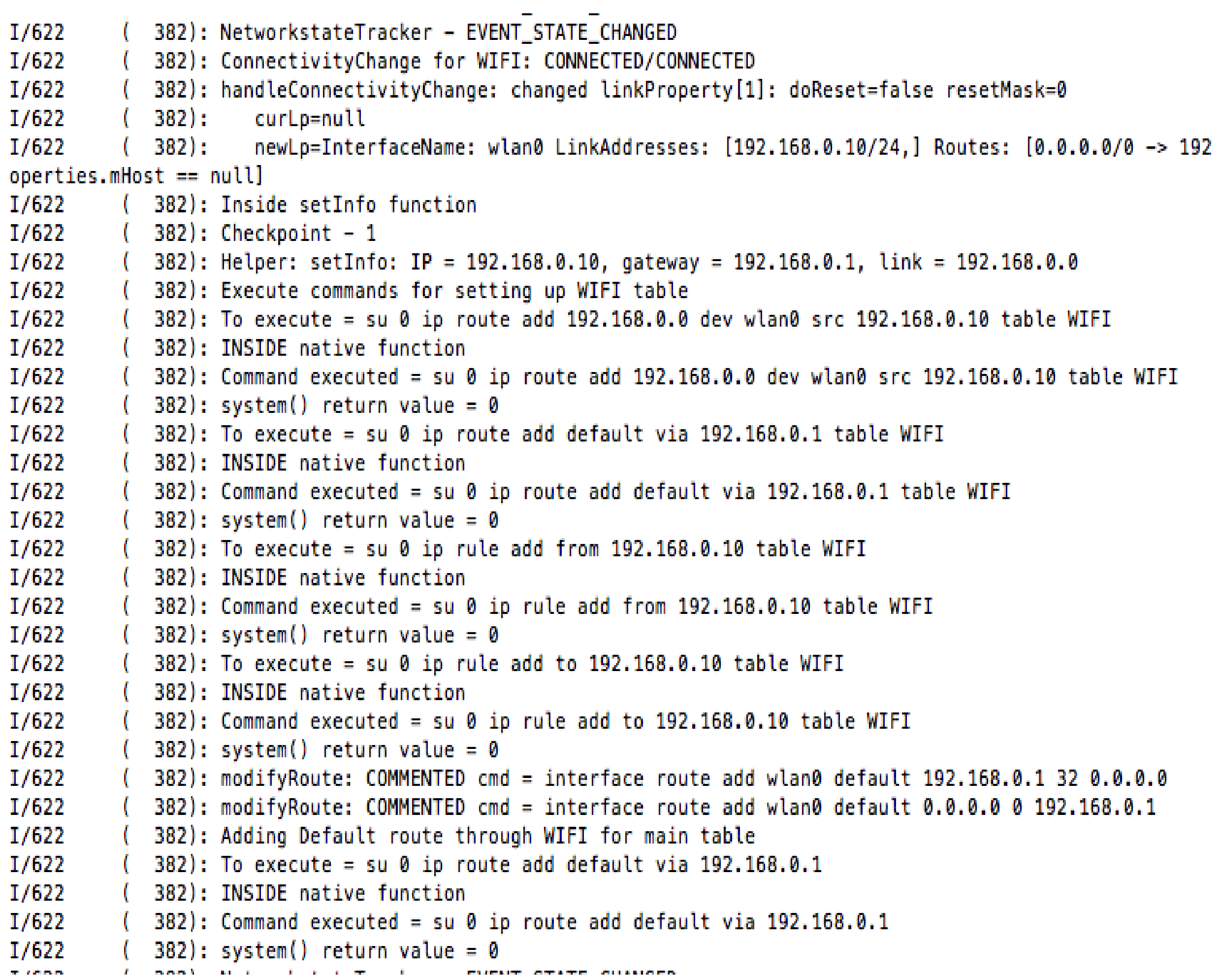


Figure 3. Work stealing implementation.

4. Screen Shots

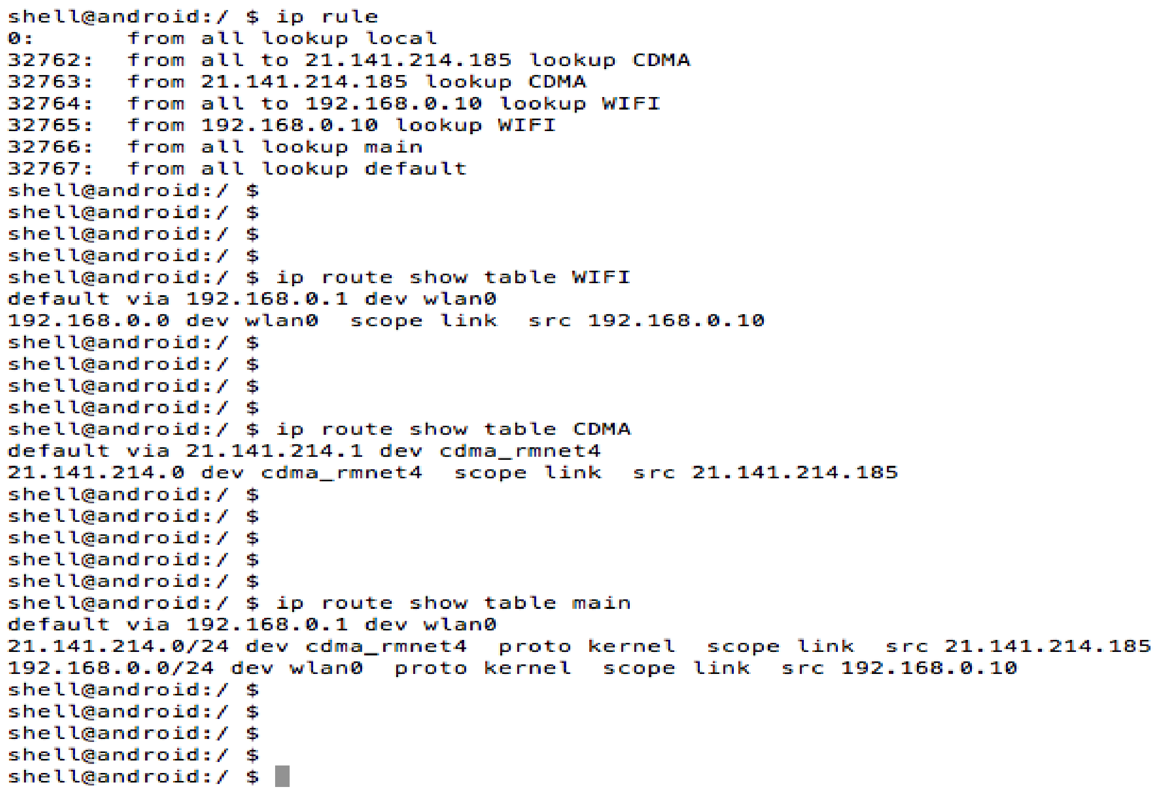
4.1 Enable both interfaces and manage routing tables (framework)

The below snapshot shows the use of native module we wrote to update routing tables using “ip” and “su” utilities. When WIFI is connected, we update the routing table corresponding to it by forming commands at the frameworks level and then issue them using ConnectivityServiceHelper which makes native calls that uses “system()” function to execute the commands formed. Basically, we issue commands to update WIFI’s routing table, “main” routing table and the “rule” list.



```
I/622     (  382): NetworkstateTracker - EVENT_STATE_CHANGED
I/622     (  382): ConnectivityChange for WIFI: CONNECTED/CONNECTED
I/622     (  382): handleConnectivityChange: changed linkProperty[1]: doReset=false resetMask=0
I/622     (  382):    curLp=null
I/622     (  382):    newLp=InterfaceName: wlan0 LinkAddresses: [192.168.0.10/24,] Routes: [0.0.0.0/0 -> 192
operties.mHost == null]
I/622     (  382): Inside setInfo function
I/622     (  382): Checkpoint - 1
I/622     (  382): Helper: setInfo: IP = 192.168.0.10, gateway = 192.168.0.1, link = 192.168.0.0
I/622     (  382): Execute commands for setting up WIFI table
I/622     (  382): To execute = su 0 ip route add 192.168.0.0 dev wlan0 src 192.168.0.10 table WIFI
I/622     (  382): INSIDE native function
I/622     (  382): Command executed = su 0 ip route add 192.168.0.0 dev wlan0 src 192.168.0.10 table WIFI
I/622     (  382): system() return value = 0
I/622     (  382): To execute = su 0 ip route add default via 192.168.0.1 table WIFI
I/622     (  382): INSIDE native function
I/622     (  382): Command executed = su 0 ip route add default via 192.168.0.1 table WIFI
I/622     (  382): system() return value = 0
I/622     (  382): To execute = su 0 ip rule add from 192.168.0.10 table WIFI
I/622     (  382): INSIDE native function
I/622     (  382): Command executed = su 0 ip rule add from 192.168.0.10 table WIFI
I/622     (  382): system() return value = 0
I/622     (  382): To execute = su 0 ip rule add to 192.168.0.10 table WIFI
I/622     (  382): INSIDE native function
I/622     (  382): Command executed = su 0 ip rule add to 192.168.0.10 table WIFI
I/622     (  382): system() return value = 0
I/622     (  382): modifyRoute: COMMENTED cmd = interface route add wlan0 default 192.168.0.1 32 0.0.0.0
I/622     (  382): modifyRoute: COMMENTED cmd = interface route add wlan0 default 0.0.0.0 0 192.168.0.1
I/622     (  382): Adding Default route through WIFI for main table
I/622     (  382): To execute = su 0 ip route add default via 192.168.0.1
I/622     (  382): INSIDE native function
I/622     (  382): Command executed = su 0 ip route add default via 192.168.0.1
I/622     (  382): system() return value = 0
```

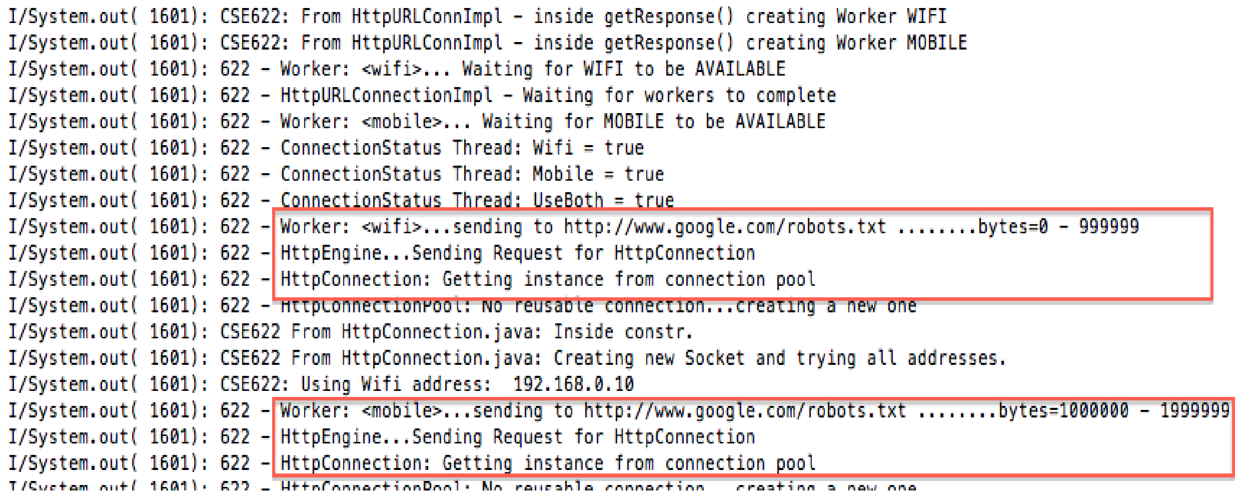
The following one shows the changes that were made to the “rule” list and routing tables when both the interfaces where brought up.



```
shell@android:/ $ ip rule
0:       from all lookup local
32762:   from all to 21.141.214.185 lookup CDMA
32763:   from 21.141.214.185 lookup CDMA
32764:   from all to 192.168.0.10 lookup WIFI
32765:   from 192.168.0.10 lookup WIFI
32766:   from all lookup main
32767:   from all lookup default
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $ ip route show table WIFI
default via 192.168.0.1 dev wlan0
192.168.0.0 dev wlan0  scope link  src 192.168.0.10
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $ ip route show table CDMA
default via 21.141.214.1 dev cdma_rmnet4
21.141.214.0 dev cdma_rmnet4  scope link  src 21.141.214.185
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $ ip route show table main
default via 192.168.0.1 dev wlan0
21.141.214.0/24 dev cdma_rmnet4  proto kernel  scope link  src 21.141.214.185
192.168.0.0/24 dev wlan0  proto kernel  scope link  src 192.168.0.10
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $
shell@android:/ $
```

4.2 Simultaneous downloading with HTTP library (libcore)

Here, we see the working of two HttpWorkers, requesting for their chunks using "Work Stealing" algorithm. Also, you may notice that initially both the workers wait for the ConnectionStatus thread to get the status of both the interfaces via socket to ConnectivityServer in frameworks layer.



```
I/System.out( 1601): CSE622: From HttpURLConnImpl - inside getResponse() creating Worker WIFI
I/System.out( 1601): CSE622: From HttpURLConnImpl - inside getResponse() creating Worker MOBILE
I/System.out( 1601): 622 - Worker: <wifi>... Waiting for WIFI to be AVAILABLE
I/System.out( 1601): 622 - HttpURLConnectionImpl - Waiting for workers to complete
I/System.out( 1601): 622 - Worker: <mobile>... Waiting for MOBILE to be AVAILABLE
I/System.out( 1601): 622 - ConnectionStatus Thread: Wifi = true
I/System.out( 1601): 622 - ConnectionStatus Thread: Mobile = true
I/System.out( 1601): 622 - ConnectionStatus Thread: UseBoth = true
I/System.out( 1601): 622 - Worker: <wifi>...sending to http://www.google.com/robots.txt ........bytes=0 - 999999
I/System.out( 1601): 622 - HttpEngine...Sending Request for HttpConnection
I/System.out( 1601): 622 - HttpConnection: Getting instance from connection pool
I/System.out( 1601): 622 - HttpConnectionPool: No reusable connection...creating a new one
I/System.out( 1601): CSE622 From HttpConnection.java: Inside constr.
I/System.out( 1601): CSE622 From HttpConnection.java: Creating new Socket and trying all addresses.
I/System.out( 1601): CSE622: Using Wifi address:  192.168.0.10
I/System.out( 1601): 622 - Worker: <mobile>...sending to http://www.google.com/robots.txt ........bytes=1000000 - 1999999
I/System.out( 1601): 622 - HttpEngine...Sending Request for HttpConnection
I/System.out( 1601): 622 - HttpConnection: Getting instance from connection pool
I/System.out( 1601): 622 - HttpConnectionPool: No reusable connection...creating a new one
```

5. Future Work

5.1 Policies

Information such as signal strength readings and current energy level could be used to dynamically make decisions about the usage of either interfaces, or when to switch off and transfer to either one if needed. Obtaining such information from the existing services within Android will be the future step to take. Once these are provided to the network handlers, both in the framework and library, more intelligent decisions can be made regarding the usage of multiple interfaces.

5.2 Chunk size selection

The request chunk size is currently implemented using a constant value (1000000). Both interfaces request the same chunk size by changing the starting and ending bytes to the server. This is continued until either of the worker receives a 416 (Requested Range Not Satisfiable) response code. Again, more efficient request division can be done between the interfaces. One thing we noticed when testing our implementation was that Wi-Fi interface requests and obtains most of the chunks while the mobile interface fetches only the few chunks. Such trends can be used to make dynamic request size updates. To enhance the implementation, more testing needs to be done to understand the effect of having different delays, TCP window sizes, and Http request pipes sizes, across these interfaces.

References

[1] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A First Look at Traffic on Smartphones. *IMC 2010*.

[2] Aaron Gember, Ashok Anand, Aditya Akella. A comparative study of handheld and non-handheld traffic in campus Wi-Fi networks. *PAM 2011.*

[3] http://www.lartc.org/lartc.pdf

[4] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. *IMC 2009*.

[5] Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, Oliver Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. *MobiSys 2012.*

[6] http://en.wikipedia.org/wiki/List_of_HTTP_status_codes