

Analyzing Performance of TCP during Handoff in Real Mobile Networks

Progress Report 2

by

Lovejeet Singh

Entry No. 2010CS50285

Shantanu Chaudhary

Entry No. 2010CS50295

Siddharth Batra

Entry No. 2010CS50297

Under the guidance of

Prof. Vinay Ribeiro



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
INDIAN INSTITUTE OF TECHNOLOGY DELHI

Contents

| | | |
|---|--|----|
| 1 | Introduction | 2 |
| 2 | Work done till now | 2 |
| 3 | Configuring Pluggable Linux modules | 3 |
| 4 | Problems with current testing approach | 3 |
| 5 | Solution to above problem | 5 |
| 6 | Data gathered to verify functioning of tools | 6 |
| 7 | Timeline | 9 |
| 8 | Conclusion | 10 |

1 Introduction

In this report, we describe the progress made so far in the phase 2 of the project. We begin by addressing the work done till now. Next we cover the specific application stack we used for data gathering. During this phase of the project when we started data gathering on the field, we encountered certain problems with our testing approach and a lot of time was spent to address this. We shall discuss these problems and how we plan to solve them.

2 Work done till now

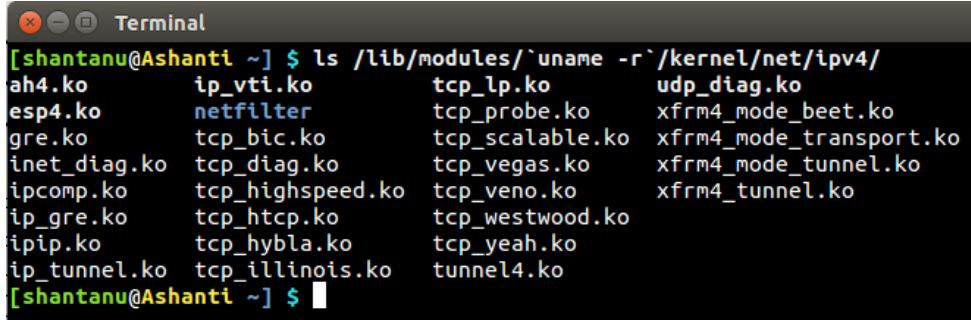
- Configured pluggable modules in Linux
- Identified problems with testing approach
- Worked on overcoming problems encountered in the data gathering phase
- Selected specific tools for data gathering and analysis

These points are further described in the sections below.

3 Configuring Pluggable Linux modules

In the previous report, we mentioned that we would need to add more modules to the linux build we are using for our analysis so as to compare atleast 3 implementations of TCP. After our research we have found that it is possible to add more TCP congestion control algorithms to the kernel as pluggable modules. For our analysis we will be using Cubic (default in linux), Westwood+ (modified for wireless links), Veno (modified for wireless links). we can determine the available modules for the kernel using the following command:

```
$ ls /lib/modules/$(uname -r)/kernel/net/ipv4/
```

A terminal window titled "Terminal" showing the command `ls /lib/modules/$(uname -r)/kernel/net/ipv4/` and its output. The output lists various kernel modules in a grid-like fashion:

| | | | |
|--------------|------------------|-----------------|-------------------------|
| ah4.ko | ip_vti.ko | tcp_lp.ko | udp_diag.ko |
| esp4.ko | netfilter | tcp_probe.ko | xfrm4_mode_beet.ko |
| gre.ko | tcp_bic.ko | tcp_scalable.ko | xfrm4_mode_transport.ko |
| inet_diag.ko | tcp_diag.ko | tcp_vegas.ko | xfrm4_mode_tunnel.ko |
| ipcomp.ko | tcp_highspeed.ko | tcp_veno.ko | xfrm4_tunnel.ko |
| ip_gre.ko | tcp_htcp.ko | tcp_westwood.ko | |
| ipip.ko | tcp_hybla.ko | tcp_yeah.ko | |
| ip_tunnel.ko | tcp_illinois.ko | tunnel4.ko | |

The prompt `[shantanu@Ashanti ~]$` is visible at the bottom.

Now the modules can be simply loaded using the following command:

```
$ echo X > /proc/sys/net/ipv4/tcp_congestion_control
```

where X is the congestion control algorithm listed for the linux kernel.

4 Problems with current testing approach

The approach for our analysis can be referred to in the previous project report. According to that scheme we were using iperf, a linux utility for analysing and tuning tcp parameters of the kernel. In order to use this tool, one needs a server and a client on the machine on which one wishes to capture tcp parameters. So we set up an Amazon EC2 Instance with the iperf server. When we started gathering data in the field using mobile data, we noticed that the wireless link of mobile network might not be the bottleneck link in

this case for our analysis to work. We investigated this matter and present the following screenshots as evidence:

```
[shantanu@Ashanti ~] $ ping 52.74.68.109
PING 52.74.68.109 (52.74.68.109) 56(84) bytes of data.
64 bytes from 52.74.68.109: icmp_seq=1 ttl=42 time=1849 ms
64 bytes from 52.74.68.109: icmp_seq=2 ttl=42 time=982 ms
64 bytes from 52.74.68.109: icmp_seq=3 ttl=42 time=122 ms
64 bytes from 52.74.68.109: icmp_seq=4 ttl=42 time=142 ms
64 bytes from 52.74.68.109: icmp_seq=5 ttl=42 time=1657 ms
64 bytes from 52.74.68.109: icmp_seq=6 ttl=42 time=1152 ms
64 bytes from 52.74.68.109: icmp_seq=7 ttl=42 time=222 ms
64 bytes from 52.74.68.109: icmp_seq=8 ttl=42 time=119 ms
64 bytes from 52.74.68.109: icmp_seq=9 ttl=42 time=137 ms
64 bytes from 52.74.68.109: icmp_seq=10 ttl=42 time=1648 ms
64 bytes from 52.74.68.109: icmp_seq=11 ttl=42 time=1021 ms
64 bytes from 52.74.68.109: icmp_seq=12 ttl=42 time=451 ms
64 bytes from 52.74.68.109: icmp_seq=13 ttl=42 time=341 ms
64 bytes from 52.74.68.109: icmp_seq=14 ttl=42 time=361 ms
64 bytes from 52.74.68.109: icmp_seq=15 ttl=42 time=421 ms
64 bytes from 52.74.68.109: icmp_seq=16 ttl=42 time=301 ms
64 bytes from 52.74.68.109: icmp_seq=17 ttl=42 time=420 ms
64 bytes from 52.74.68.109: icmp_seq=18 ttl=42 time=331 ms
64 bytes from 52.74.68.109: icmp_seq=19 ttl=42 time=361 ms
64 bytes from 52.74.68.109: icmp_seq=20 ttl=42 time=351 ms
64 bytes from 52.74.68.109: icmp_seq=21 ttl=42 time=331 ms
64 bytes from 52.74.68.109: icmp_seq=22 ttl=42 time=361 ms
64 bytes from 52.74.68.109: icmp_seq=23 ttl=42 time=431 ms
64 bytes from 52.74.68.109: icmp_seq=24 ttl=42 time=471 ms
64 bytes from 52.74.68.109: icmp_seq=25 ttl=42 time=331 ms
64 bytes from 52.74.68.109: icmp_seq=26 ttl=42 time=351 ms
64 bytes from 52.74.68.109: icmp_seq=27 ttl=42 time=381 ms
64 bytes from 52.74.68.109: icmp_seq=28 ttl=42 time=659 ms
64 bytes from 52.74.68.109: icmp_seq=29 ttl=42 time=116 ms
64 bytes from 52.74.68.109: icmp_seq=30 ttl=42 time=121 ms
64 bytes from 52.74.68.109: icmp_seq=31 ttl=42 time=122 ms
64 bytes from 52.74.68.109: icmp_seq=32 ttl=42 time=136 ms
64 bytes from 52.74.68.109: icmp_seq=33 ttl=42 time=140 ms
64 bytes from 52.74.68.109: icmp_seq=34 ttl=42 time=133 ms
64 bytes from 52.74.68.109: icmp_seq=35 ttl=42 time=120 ms
64 bytes from 52.74.68.109: icmp_seq=36 ttl=42 time=119 ms
64 bytes from 52.74.68.109: icmp_seq=37 ttl=42 time=119 ms
64 bytes from 52.74.68.109: icmp_seq=38 ttl=42 time=126 ms
64 bytes from 52.74.68.109: icmp_seq=39 ttl=42 time=140 ms
64 bytes from 52.74.68.109: icmp_seq=40 ttl=42 time=137 ms
64 bytes from 52.74.68.109: icmp_seq=41 ttl=42 time=1569 ms
64 bytes from 52.74.68.109: icmp_seq=42 ttl=42 time=702 ms
64 bytes from 52.74.68.109: icmp_seq=43 ttl=42 time=132 ms
64 bytes from 52.74.68.109: icmp_seq=44 ttl=42 time=122 ms
64 bytes from 52.74.68.109: icmp_seq=45 ttl=42 time=121 ms
64 bytes from 52.74.68.109: icmp_seq=46 ttl=42 time=121 ms
64 bytes from 52.74.68.109: icmp_seq=47 ttl=42 time=140 ms
64 bytes from 52.74.68.109: icmp_seq=48 ttl=42 time=1507 ms
64 bytes from 52.74.68.109: icmp_seq=49 ttl=42 time=1060 ms
64 bytes from 52.74.68.109: icmp_seq=50 ttl=42 time=120 ms
64 bytes from 52.74.68.109: icmp_seq=51 ttl=42 time=149 ms
64 bytes from 52.74.68.109: icmp_seq=52 ttl=42 time=151 ms
```

In the above screenshot, there is a variable consistency in the pings to our EC2 instance.

Even normal pings to google india servers were consistent.

```
[shantanu@Ashanti ~] $ ping www.google.com
4PING www.google.com (173.194.38.148) 56(84) bytes of data.
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=1 ttl=49 time=389 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=2 ttl=49 time=357 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=3 ttl=49 time=298 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=4 ttl=49 time=376 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=5 ttl=49 time=294 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=6 ttl=49 time=370 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=7 ttl=49 time=301 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=8 ttl=49 time=318 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=9 ttl=49 time=319 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=10 ttl=49 time=508 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=11 ttl=49 time=307 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=12 ttl=49 time=313 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=13 ttl=49 time=314 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=14 ttl=49 time=331 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=15 ttl=49 time=331 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=16 ttl=49 time=320 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=17 ttl=49 time=369 ms
64 bytes from sin04s01-in-f20.1e100.net (173.194.38.148): icmp_seq=18 ttl=49 time=337 ms
```

Hence, we cannot be sure that if we gather data using this approach, the wireless link will act as the bottlenecked link. Hence we need to rethink our approach.

5 Solution to above problem

A viable solution which we figured would be to arrange a machine with public IP which is local to us and we can then connect to it using our mobile data connection and carry forward with our analysis. For this approach, we thought of using the ownCloud servers of IIT. We uploaded some big files on the server. We thought that downloading the files from server using tools such as wget and curl would help us in this regard. We used tcpprobe tool for sniffing packets on a particular port. wget doesnt allow to specify client side port so we used curl. On inspecting the log from the probe, we saw that the window was fixed at 10 with each packet of 1400 bytes. This doesnt give much information to us regarding the performance of the congestion algorithm.

Finally, we have decided to use a linux based machine assigned to us by the instructor. Now, we can use iperf and wireshark tools to capture packet data and do our analysis.

6 Data gathered to verify functioning of tools

We present the following screenshots that verify the functioning of iperf for analysis.

```
-----
Server listening on TCP port 54545
TCP window size: 85.3 KByte (default)
-----
[ 4] local 172.31.20.139 port 54545 connected with 1.39.35.152 port 18000
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.0- 1.0 sec   143 KBytes  143 KBytes/sec
[ 4] 1.0- 2.0 sec   154 KBytes  154 KBytes/sec
[ 4] 2.0- 3.0 sec   154 KBytes  154 KBytes/sec
[ 4] 3.0- 4.0 sec   148 KBytes  148 KBytes/sec
[ 4] 4.0- 5.0 sec   142 KBytes  142 KBytes/sec
[ 4] 5.0- 6.0 sec   157 KBytes  157 KBytes/sec
[ 4] 6.0- 7.0 sec   159 KBytes  159 KBytes/sec
[ 4] 7.0- 8.0 sec   156 KBytes  156 KBytes/sec
[ 4] 8.0- 9.0 sec   153 KBytes  153 KBytes/sec
[ 4] 9.0-10.0 sec   129 KBytes  129 KBytes/sec
[ 4] 10.0-11.0 sec   139 KBytes  139 KBytes/sec
[ 4] 11.0-12.0 sec   156 KBytes  156 KBytes/sec
[ 4] 12.0-13.0 sec   159 KBytes  159 KBytes/sec
[ 4] 13.0-14.0 sec   159 KBytes  159 KBytes/sec
[ 4] 14.0-15.0 sec   158 KBytes  158 KBytes/sec
[ 4] 15.0-16.0 sec   148 KBytes  148 KBytes/sec
[ 4] 16.0-17.0 sec   150 KBytes  150 KBytes/sec
[ 4] 17.0-18.0 sec   96.9 KBytes 96.9 KBytes/sec
[ 4] 18.0-19.0 sec   90.6 KBytes 90.6 KBytes/sec
[ 4] 19.0-20.0 sec   144 KBytes  144 KBytes/sec
[ 4] 20.0-21.0 sec   159 KBytes  159 KBytes/sec
[ 4] 21.0-22.0 sec   140 KBytes  140 KBytes/sec
[ 4] 22.0-23.0 sec   150 KBytes  150 KBytes/sec
[ 4] 23.0-24.0 sec   150 KBytes  150 KBytes/sec
[ 4] 24.0-25.0 sec   159 KBytes  159 KBytes/sec
[ 4] 25.0-26.0 sec   158 KBytes  158 KBytes/sec
[ 4] 26.0-27.0 sec   159 KBytes  159 KBytes/sec
[ 4] 27.0-28.0 sec   152 KBytes  152 KBytes/sec
[ 4] 28.0-29.0 sec   134 KBytes  134 KBytes/sec
[ 4] 29.0-30.0 sec   139 KBytes  139 KBytes/sec
[ 4] 30.0-31.0 sec   98.2 KBytes 98.2 KBytes/sec
[ 4] 31.0-32.0 sec   138 KBytes  138 KBytes/sec
[ 4] 32.0-33.0 sec   134 KBytes  134 KBytes/sec
[ 4] 33.0-34.0 sec   145 KBytes  145 KBytes/sec
[ 4] 34.0-35.0 sec   143 KBytes  143 KBytes/sec
[ 4] 35.0-36.0 sec   150 KBytes  150 KBytes/sec
[ 4] 36.0-37.0 sec   157 KBytes  157 KBytes/sec
[ 4] 37.0-38.0 sec   130 KBytes  130 KBytes/sec
[ 4] 38.0-39.0 sec   159 KBytes  159 KBytes/sec
[ 4] 39.0-40.0 sec   158 KBytes  158 KBytes/sec
[ 4] 40.0-41.0 sec   158 KBytes  158 KBytes/sec
[ 4] 41.0-42.0 sec   152 KBytes  152 KBytes/sec
```


| [ID] | Interval | Transfer | Bandwidth |
|-------|---------------|-------------|-----------------|
| [4] | 0.0- 1.0 sec | 143 KBytes | 143 KBytes/sec |
| [4] | 1.0- 2.0 sec | 154 KBytes | 154 KBytes/sec |
| [4] | 2.0- 3.0 sec | 154 KBytes | 154 KBytes/sec |
| [4] | 3.0- 4.0 sec | 148 KBytes | 148 KBytes/sec |
| [4] | 4.0- 5.0 sec | 142 KBytes | 142 KBytes/sec |
| [4] | 5.0- 6.0 sec | 157 KBytes | 157 KBytes/sec |
| [4] | 6.0- 7.0 sec | 159 KBytes | 159 KBytes/sec |
| [4] | 7.0- 8.0 sec | 156 KBytes | 156 KBytes/sec |
| [4] | 8.0- 9.0 sec | 153 KBytes | 153 KBytes/sec |
| [4] | 9.0-10.0 sec | 129 KBytes | 129 KBytes/sec |
| [4] | 10.0-11.0 sec | 139 KBytes | 139 KBytes/sec |
| [4] | 11.0-12.0 sec | 156 KBytes | 156 KBytes/sec |
| [4] | 12.0-13.0 sec | 159 KBytes | 159 KBytes/sec |
| [4] | 13.0-14.0 sec | 159 KBytes | 159 KBytes/sec |
| [4] | 14.0-15.0 sec | 158 KBytes | 158 KBytes/sec |
| [4] | 15.0-16.0 sec | 148 KBytes | 148 KBytes/sec |
| [4] | 16.0-17.0 sec | 150 KBytes | 150 KBytes/sec |
| [4] | 17.0-18.0 sec | 96.9 KBytes | 96.9 KBytes/sec |
| [4] | 18.0-19.0 sec | 90.6 KBytes | 90.6 KBytes/sec |
| [4] | 19.0-20.0 sec | 144 KBytes | 144 KBytes/sec |
| [4] | 20.0-21.0 sec | 159 KBytes | 159 KBytes/sec |
| [4] | 21.0-22.0 sec | 140 KBytes | 140 KBytes/sec |
| [4] | 22.0-23.0 sec | 150 KBytes | 150 KBytes/sec |
| [4] | 23.0-24.0 sec | 150 KBytes | 150 KBytes/sec |
| [4] | 24.0-25.0 sec | 159 KBytes | 159 KBytes/sec |
| [4] | 25.0-26.0 sec | 158 KBytes | 158 KBytes/sec |
| [4] | 26.0-27.0 sec | 159 KBytes | 159 KBytes/sec |
| [4] | 27.0-28.0 sec | 152 KBytes | 152 KBytes/sec |
| [4] | 28.0-29.0 sec | 134 KBytes | 134 KBytes/sec |
| [4] | 29.0-30.0 sec | 139 KBytes | 139 KBytes/sec |
| [4] | 30.0-31.0 sec | 98.2 KBytes | 98.2 KBytes/sec |
| [4] | 31.0-32.0 sec | 138 KBytes | 138 KBytes/sec |
| [4] | 32.0-33.0 sec | 134 KBytes | 134 KBytes/sec |
| [4] | 33.0-34.0 sec | 145 KBytes | 145 KBytes/sec |
| [4] | 34.0-35.0 sec | 143 KBytes | 143 KBytes/sec |
| [4] | 35.0-36.0 sec | 150 KBytes | 150 KBytes/sec |
| [4] | 36.0-37.0 sec | 157 KBytes | 157 KBytes/sec |
| [4] | 37.0-38.0 sec | 130 KBytes | 130 KBytes/sec |
| [4] | 38.0-39.0 sec | 159 KBytes | 159 KBytes/sec |
| [4] | 39.0-40.0 sec | 158 KBytes | 158 KBytes/sec |
| [4] | 40.0-41.0 sec | 158 KBytes | 158 KBytes/sec |
| [4] | 41.0-42.0 sec | 152 KBytes | 152 KBytes/sec |
| [4] | 42.0-43.0 sec | 77.8 KBytes | 77.8 KBytes/sec |
| [4] | 43.0-44.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 44.0-45.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 45.0-46.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 46.0-47.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 47.0-48.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 48.0-49.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 49.0-50.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 50.0-51.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 51.0-52.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 52.0-53.0 sec | 0.00 KBytes | 0.00 KBytes/sec |
| [4] | 53.0-54.0 sec | 1613 KBytes | 1613 KBytes/sec |
| [4] | 0.0-54.8 sec | 7936 KBytes | 145 KBytes/sec |

In the above screenshot, one can see the response of the iperf server on the Amazon EC2 instance. The server reports bandwidth achieved for each

interval.

```
[shantanu@Ashanti ~] $ iperf -c 52.74.68.109 -f KB -p 54545 -o iperflog.log -t 50
-----
Client connecting to 52.74.68.109, TCP port 54545
TCP window size: 45.0 KByte (default)
-----
[ 3] local 192.168.42.164 port 55281 connected with 52.74.68.109 port 54545
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-53.9 sec  7936 KBytes  147 KBytes/sec
```

The above screenshot shows how successfully iperf client is connected to the iperf server and has achieved certain bandwidth. We also probed the port 54545 for packet data analysis. The output was piped to a log file which can be seen below:

```
401.680058533 192.168.42.164:55281 52.74.68.109:54545 32 0x31600a96 0x315f15b6 48 2147483647 123520 47 29312
401.690875200 192.168.42.164:55281 52.74.68.109:54545 32 0x316014ca 0x315f1ad0 49 2147483647 126208 48 29312
401.704847761 192.168.42.164:55281 52.74.68.109:54545 32 0x31601efe 0x315f1fea 50 2147483647 128768 49 29312
401.714851735 192.168.42.164:55281 52.74.68.109:54545 32 0x31602932 0x315f2504 51 2147483647 131328 50 29312
401.726850793 192.168.42.164:55281 52.74.68.109:54545 32 0x31603880 0x315f2f38 52 51 136576 51 29312
401.732831633 192.168.42.164:55281 52.74.68.109:54545 32 0x31603d9a 0x315f3452 52 51 139264 52 29312
401.744852877 192.168.42.164:55281 52.74.68.109:54545 32 0x316042b4 0x315f396c 52 51 141824 53 29312
401.745219975 192.168.42.164:55281 52.74.68.109:54545 32 0x316047ce 0x315f3e86 52 51 144512 55 29312
401.752701824 192.168.42.164:55281 52.74.68.109:54545 32 0x31604c08 0x315f4308 52 51 147072 55 29312
```

Each line of the log presents details of the packet such as time, senders ip address, receivers IP address, bytes in packet, etc. These details will then be parsed by us to plot how congestion window varies in each TCP algorithm along with other parameters.

7 Timeline

| | Feb 1-15 | Feb 15-28 | March 1-15 | Mar 16-31 | April 1-15 |
|-------------|----------|-----------|------------|-----------|------------|
| Milestone 1 | | | | | |
| Milestone 2 | | | | | |
| Milestone 3 | | | | | |
| Milestone 4 | | | | | |

Milestone 1: Understanding implementations of different variants of TCP on Ubuntu 14.04 and changing congestion control algorithm in Ubuntu 14.04

Milestone 2: ~~Development of Android Application.~~ Determine tool setup and data gathering tools and understand their working.

Milestone 3: Data gathering by using the android device on the field and exposing it to handoffs on the mobile carrier's network.

Milestone 4: Performance comparison and data analysis.

8 Conclusion

The above mentioned problems took a significant time of our project so we are behind schedule. But, in our opinion, it was necessary to address these problems as it could have generated incorrect data for our analysis and could have posed a serious problem for the project. We aim to cover up this delay by utilizing the holidays and the extended weekend. We will now use the tools defined above for various versions of TCP on the Linux kernels and gather various data about throughput, delays and other performance parameters.