Monitoring Android Apps using the **logcat** and **iperf** tools

Michalis Katsarakis katsarakis@csd.uoc.gr

Tutorial: HY-439 http://www.csd.uoc.gr/~hy439/

22 May 2015

Outline

- **Introduction**
- Monitoring the Android App usage
  - Open source applications
  - Closed source applications
- Monitoring the network QoS
  - Passive measurements
  - Active measurements
    - Ping
    - Iperf
    - NDT

Application QoE vs. Network QoS

The **Quality of Experience (QoE)** of an Android application can be influenced by network **Quality of Service (QoS)**.

How can we assess this in influence?

We need to monitor:

- Events of application usage
  (e.g. clicks on specific buttons, progress in the usage scenarios)
- User experience
- Network performance

Outline

- Introduction
- **Monitoring the Android App usage**
  - **Open source applications**
  - **Closed source applications**
- Monitoring the network QoS
  - Passive measurements
  - Active measurements
    - Ping
    - Iperf
    - NDT

How do we monitor the App usage?

Case 1: We have access to the App source code
(e.g., App is open-source or developed by us)

Case 2: We have no access to the source code
(e.g., closed-source commercial application)

Case 1: We have access to the App source code

We can add code snippets on specific locations of the App source code.

- Write events in a text file
- Write events in a database
- Broadcast Intents or use a ContentProvider to send events on a monitor App.

Case 2: We have no access to the source code

Use **logcat** to read the logs of the App.

The Android logging system provides a mechanism for collecting and viewing system debug output. Logs from various applications and portions of the system are collected in a series of circular buffers, which then can be viewed and filtered by the logcat command.

[adb] logcat [<option>] ... [<filter-spec>] ...

Android Apps can execute a logcat process and parse its stdout stream.

Outline

- Introduction
- Monitoring the Android App usage
  - Open source applications
  - Closed source applications
- **Monitoring the network QoS**
  - **Passive measurements**
  - **Active measurements**
    - **Ping**
    - **Iperf**
    - **NDT**

How do we monitor network QoS?

Case 1: We have access to the App source code

We have also access on the App's sockets & packet streams. **Passively** record measurements on these packet streams (e.g., measure packet loss, jitter, bitrate).

Case 2: We have no access to the source code

- **Solution 1:** Use rooted Androids and tcpdump/wireshark (This would provide access to the App's packet streams).
- **Solution 2:** Send/receive additional packets and record measurements on these packet streams (**Active Probing**).

Case 1: We have access to the App source code

Passive measurements:

Add some code in specific locations of the App's source (e.g., where packets are received/sent) to record network measurements.

Example:

```
long t;
while (true) {
    // Wait to receive a datagram
    socket.receive(packet);

    // Record interarrival time
    long now = System.currentTimeMillis();
    interarrival = now –t;
    t= now;
}
```

Case 2: We have no access to the source code

Active probing:

The monitor application creates additional packet streams that at which it has access and performs measurements on these additional packet streams.

The active probing approach degrades network performance in order to measure it!
(if possible, choose a small transmission rate)

Popular tools for active probing:

ping, iperf, NDT

Ping

Ping uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway.

Syntax:

ping [-QRadfnqrv] [-c count] [-i wait] [-l preload] [-p pattern] [-P policy] [-s packetsize] [-S src_addr] [-t timeout] [host | [-L] [-I interface] [-T ttl] mcast-group]

Ping

Example:

ping localhost

Output:

PING localhost (127.0.0.1) 56(84) bytes of data.

64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.051 ms

64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.055 ms

^C

--- localhost ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 999ms

rtt min/avg/max/mdev = 0.051/0.053/0.055/0.002 ms

Iperf

iperf is a tool for performing network throughput measurements. It can test either TCP or UDP throughput. To perform an iperf test the user must establish both a server (to discard traffic) and a client (to generate traffic).

Syntax:

iperf -s [options]

iperf -c server [options]

iperf -u -s [options]

iperf -u -c server [options]

Iperf

Example (server-side):

#iperf -s -u -i 1

Output:

```
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[904] local 10.1.1.1 port 5001 connected with 10.6.2.5 port 32781
[ ID]   Interval       Transfer       Bandwidth        Jitter      Lost/Total Datagrams
[904]   0.0- 1.0 sec   1.17 MBytes   9.84 Mbits/sec   1.830 ms   0/ 837   (0%)
[904]   1.0- 2.0 sec   1.18 MBytes   9.94 Mbits/sec   1.846 ms   5/ 850   (0.59%)
[904]   2.0- 3.0 sec   1.19 MBytes   9.98 Mbits/sec   1.802 ms   2/ 851   (0.24%)
[904]   3.0- 4.0 sec   1.19 MBytes   10.0 Mbits/sec   1.830 ms   0/ 850   (0%)
[904]   4.0- 5.0 sec   1.19 MBytes   9.98 Mbits/sec   1.846 ms   1/ 850   (0.12%)
[904]   5.0- 6.0 sec   1.19 MBytes   10.0 Mbits/sec   1.806 ms   0/ 851   (0%)
[904]   6.0- 7.0 sec   1.06 MBytes   8.87 Mbits/sec   1.803 ms   1/ 755   (0.13%)
[904]   7.0- 8.0 sec   1.19 MBytes   10.0 Mbits/sec   1.831 ms   0/ 850   (0%)
[904]   8.0- 9.0 sec   1.19 MBytes   10.0 Mbits/sec   1.841 ms   0/ 850   (0%)
[904]   9.0-10.0 sec   1.19 MBytes   10.0 Mbits/sec   1.801 ms   0/ 851   (0%)
[904]   0.0-10.0 sec   11.8 MBytes   9.86 Mbits/sec   2.618 ms   9/ 8409  (0.11%)
```

Iperf

Example (client-side):

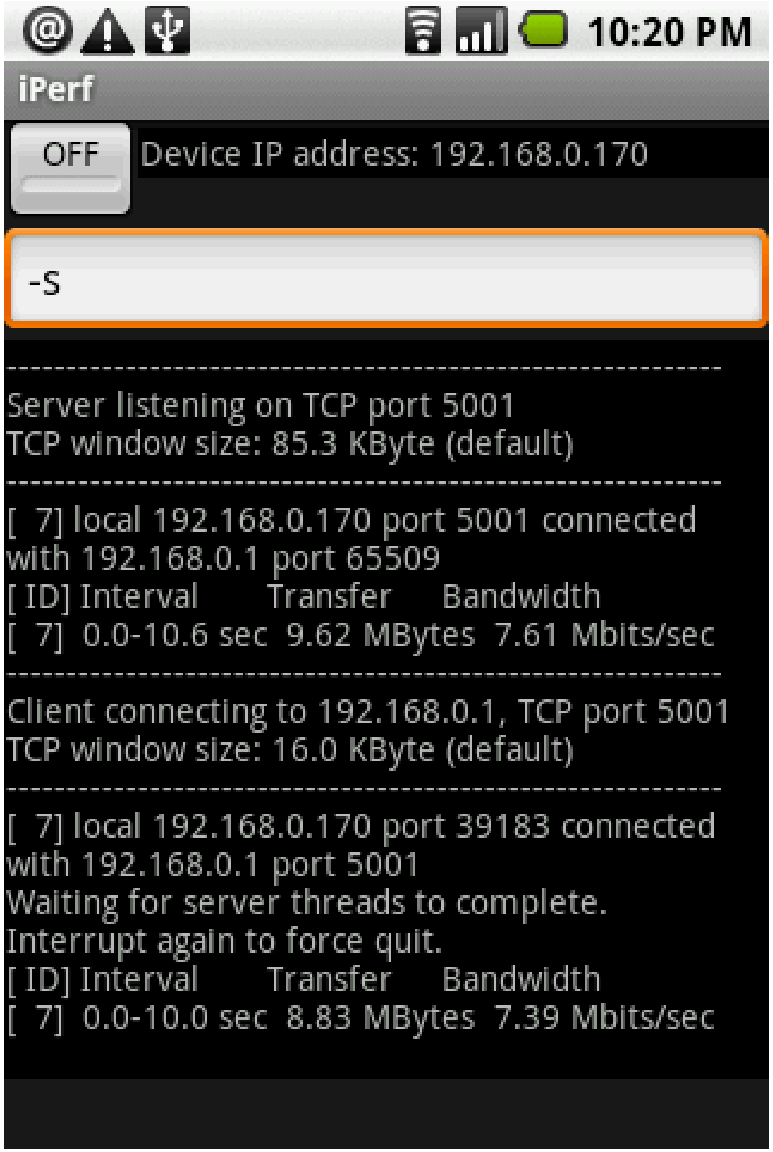
#iperf -c 10.1.1.1 -u -b 10m

Output:

```
------------------------------------------------------------
Client connecting to 10.1.1.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 108 KByte (default)
------------------------------------------------------------
[ 3] local 10.6.2.5 port 32781 connected with 10.1.1.1 port 5001
[ 3]   0.0-10.0 sec   11.8 MBytes   9.89 Mbits/sec
[ 3] Sent 8409 datagrams
[ 3] Server Report:
[ 3]   0.0-10.0 sec   11.8 MBytes   9.86 Mbits/sec   2.617 ms   9/ 8409   (0.11%)
```

Iperf

Iperf for Android:

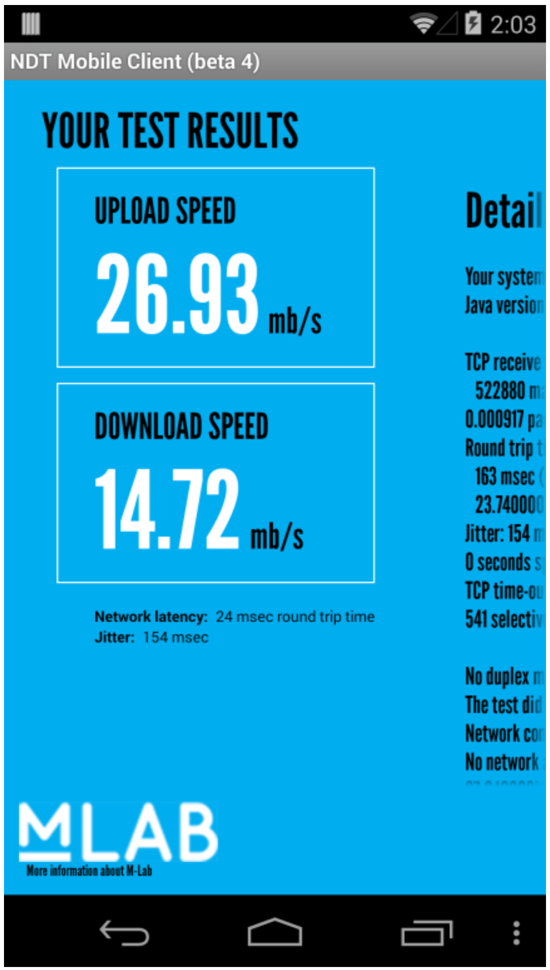


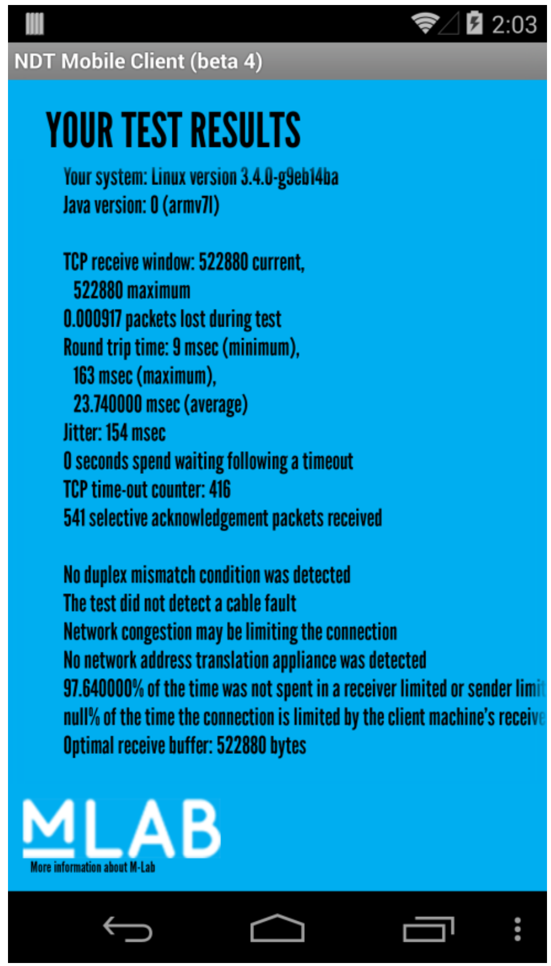
NDT

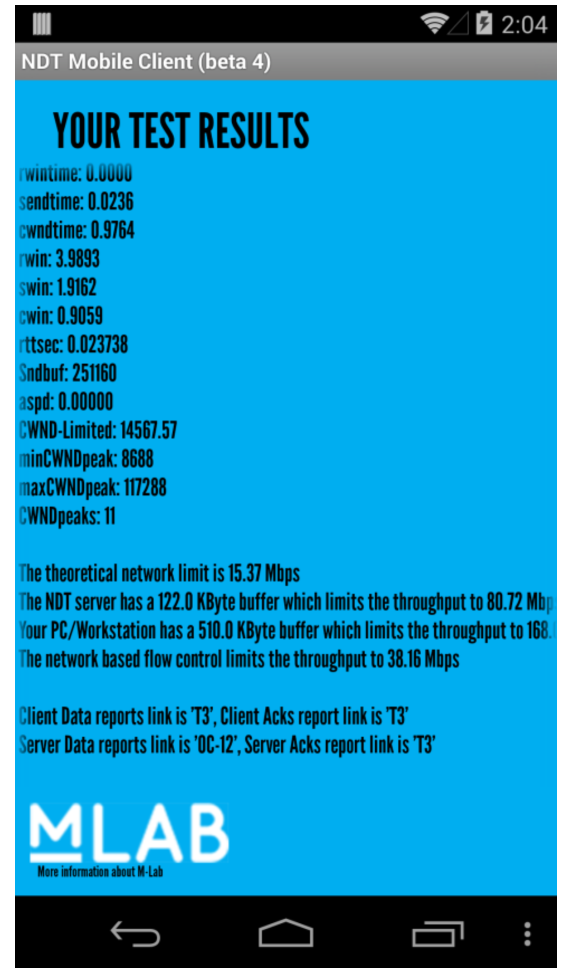
NDT (Network Diagnostic Tool) provides a sophisticated speed and diagnostic test. An NDT test reports more than just the upload and download speeds. It also attempts to determine what, if any, problems limited these speeds, differentiating between computer configuration and network infrastructure problems. While the diagnostic messages are most useful for expert users, they can also help novice users by allowing them to provide detailed trouble reports to their network administrator.

NDT

NDT Android client screenshots:







Monitoring the network QoS