Poster: Evaluating Android Applications with Multipath TCP

Quentin De Coninck *, Matthieu Baerts *, Benjamin Hesmans †, Olivier Bonaventure †
Université catholique de Louvain, Louvain-la-Neuve, Belgium
*first.last@student.uclouvain.be
†first.last@uclouvain.be

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—Wireless Communication; C.2.3 [**Computer-Communication Networks**]: Network Operations—Network Management; C.4 [**Performance of System**]: Measurement techniques, Performance attributes

Keywords

Multipath TCP; smartphones; measurement; automation; Android

1. INTRODUCTION

Smartphones are the most popular mobile multihomed devices. End-user expects that thanks to their WiFi and cellular interfaces, they are able to seamlessly use all available networks. Unfortunately, reality tells us that seamless coexistence between cellular and WiFi is not as simple as what the user expect.

Several cellular/WiFi coexistence technologies have been proposed during the last years [4]. Some of them have been deployed. Recently, Multipath TCP [3] received a lot of attention when it was selected by Apple to support its voice recognition (Siri) application. As of this writing, Siri is the only deployed smartphone application that uses Multipath TCP. and there is no public information about the benefits of using Multipath TCP with it.

Multipath TCP is a TCP extension that allows to send data from one end-to-end connection over different paths. On a smartphone, Multipath TCP allows the applications to simultaneously send and receive data over both WiFi and cellular interfaces. It achieves this objective by establishing one TCP connection, called subflow in [3], over each interface. Once the subflows have been established, data can be sent over any of the subflows. Researchers have analyzed the performance of Multipath TCP in such hybrid networks [7, 1, 5, 2]. However, these analyses have been performed with bulk transfers between laptops and servers. As of this writing, no detailed analysis of the performance of real smartphone applications with Multipath TCP has been published.

We fill this gap in this paper by proposing a framework that automates user actions on Android smartphone applications to perform network measurements. We use it to analyze how eight popular smartphone applications interact with Multipath TCP.

MobiCom'15, September 7–11, 2015, Paris, France.
ACM ISBN 978-1-4503-3543-0/15/09.
DOI: http://dx.doi.org/10.1145/2789168.2795165.

2. AUTOMATING MEASUREMENTS

In order to collect a large number of measurements, we developed an automated tests framework that automates the interactions with these applications[1]. It contains more than 4000 lines of code and is divided into two distinct parts: one to control the smartphone and one to mimic user interactions in a reproducible way. It was designed to be reusable, modular using parameters and to cope with unexpected situations caused by the limited reliability of this kind of device.

2.1 Test scenarios

Our test scenarios can be divided into two categories: *upload intensive* and *download intensive* scenarios. Each test takes less than 120 seconds.

Upload intensive We first consider two interactive applications: Facebook and Messenger. With the Facebook application, our test first updates the news feed, then writes a new status, takes and shares a new photo with a description and finally performs a new check out status. With Messenger, it sends a text message, then puts a smiley and finally sends a new photo. Then we consider two cloud storage applications: Dropbox and Google Drive. For both, we create a fresh file containing 20 MB of purely random data and upload it.

Download intensive First, we use Firefox to browse the main page of the top 12 Alexa web sites with an empty cache. Our second application is Spotify. This is a music delivery application. The test plays a new music (shuffle play feature) for 75 seconds. Finally, we consider two popular video streaming applications: Dailymotion and Youtube. For both applications, we play three different videos in the same order and watch them for 25 seconds. Those videos are available in HD and we fetch the best possible quality even when using cellular networks.

2.2 Using Multipath TCP on smartphones

Several backports of the Multipath TCP kernel on Android smartphones have been released in the last years. However, these ports were often based on older versions of the

[1]See: `github.com/MPTCP-smartphone-thesis/uitests`

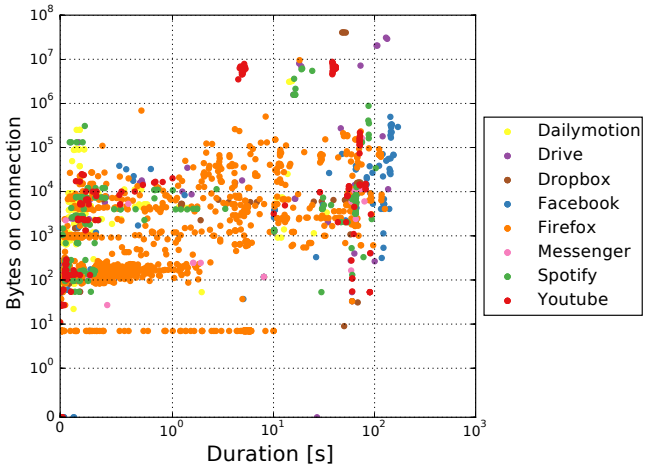

Figure 1: The smartphone applications, each point correspond to a connection for both upload and download traffic.

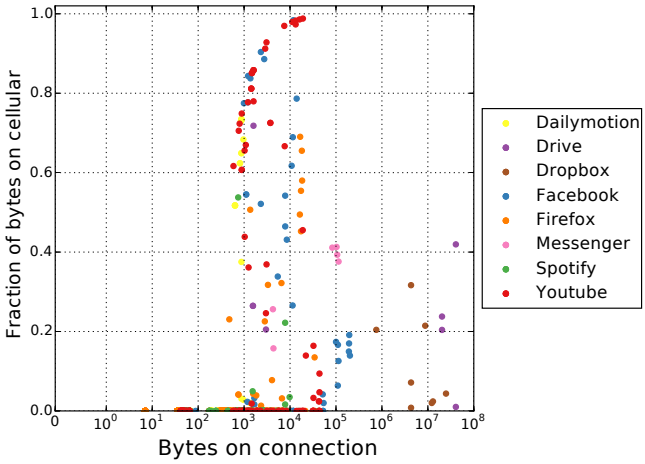

Figure 2: When the default route points to the WiFi interface, Multipath TCP mainly uses WiFi in upload for the short connections

Multipath TCP kernel. Measurements with such ports are not always representative of the current state of the implementation. For this work, we use the latest version 0.89v5 of the Multipath TCP Linux kernel that was backported on a Nexus 5 running Android 4.4.4. We also modified Android to enable the smartphone to use both interfaces simultaneously.

All the popular smartphone applications use TCP to interact with servers managed by the application developers. As of this writing, it is impossible to convince them to install Multipath TCP on their servers. To overcome this issue, we configured the smartphone to use a Multipath TCP capable SOCKS proxy server for all its connections. Each connection initiated by the smartphone is thus redirected to, and terminated at, the proxy server. The proxy server then establishes a regular TCP connection to the server. Thanks to this setup, the smartphone can use Multipath TCP over the cellular and WiFi interfaces while interacting with legacy servers. The SOCKS server itself is ShadowSocks and is configured to use the simplest encryption scheme to reduce the overhead. On the smartphone, we use the standard ShadowSocks client.

This setup allows us to capture all the packets sent by both the smartphone and the SOCKS server. We captured more than 85,000 connections over about 1200 tests conducted in February and March 2015 carrying more than 15 GBytes of data. The entire dataset is publicly available[2].

3. MEASUREMENTS

We first analyze the characteristics of the TCP connections that are used by smartphone applications. Figure 1 plots the TCP connections created by our studied applications when the smartphone uses only its WiFi interface. Each point on this figure represents one captured TCP connection. The x-axis (in logarithmic scale) is the connection duration in seconds while the y-axis is the number of bytes exchanged on the connection. Firefox is clearly the application that uses the largest number of connections (63.9 % of all connections) which is not surprising given that our Firefox scenario contacts the 12 top Alexa web sites. Dropbox (31.75%), Youtube (29.7%), Drive (19.9%), Dailymotion (9.6 %) and Spotify (4.96 %) are the applications that exchange the largest volume. On the other hand, our Facebook scenario generates long TCP connections that do not exchange too many bytes.

Some of the connections that we observed are caused by the utilization of a SOCKS proxy. We observe hundreds connections that last up to tens of seconds but only transfer seven bytes of data. After investigation, Firefox preventively opens new TCP connections but sometimes never uses them. The seven exchanged bytes correspond to the command sent by the SOCKS client. Most of the short connections that only transfer about 100 bytes are the DNS requests that are sent over TCP by the SOCKS client.

In our experiments, the connections can be categorized in 3 types: *(i)* the short ones with around 75% of all connections that last less than 1 second, *(ii)* the long ones that carry 98.5% of the overall volume, and *(iii)* long-lived ones with around 18% of all connections that do not carry a lot data (less than 10 KB).

Another important factor for TCP is the measured round-trip-time. In the upstream direction, we measured a median round-trip-time of 42.6 msec (50 msec for the mean). In the downstream direction, the median RTT is only 38.1 msec. On WiFi, 60% of the connections have a RTT shorter than 15.4 msec. There is some bufferbloat on the cellular network, mainly in the upstream direction, but bufferbloat remains reasonable compared to other networks.

3.1 Multipath measurements

We now enable Multipath TCP on our smartphone and perform the same measurements to understand how our eight applications interact with Multipath TCP. The first, but important, point to be noted is that we did not observe any incompatibility between the applications and Multipath TCP.

When a cellular and a WiFi interface are pooled together it is interesting to analyze how the smartphone balances its sending traffic over which interface. With the Multipath TCP implementation in the Linux kernel [6], this balancing

[2]See: `multipath-tcp.org/data/Mobicom15`

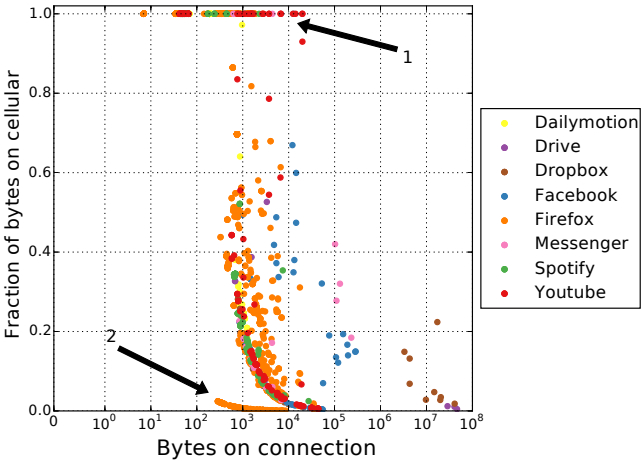

Figure 3: When the default route points to the cellular interface, lot of connections are aspired by Wifi in upload traffic.

depends on the interactions between the congestion control scheme, the packet scheduler, the underlying networks and the application.

On figure 2, we observe that 92.4% of all the connections only use the WiFi interface, but these connections only carry 1.1% of all data bytes. There are several factors that explain why Multipath TCP does not use the cellular network for these connections. The first one is the configured `default` route. When an application initiates a connection, Multipath TCP sends the `SYN` over the interface with the `default` route, in our case over WiFi. Preferring WiFi is the standard configuration of Android smartphones. If the Multipath TCP connection is short and only transfers a few KB, then most of the data fits inside the initial congestion window and can be sent over the WiFi interface while the second subflow is established over the cellular interface. Furthermore, the RTT over the WiFi interface is shorter than over the cellular interface. Thus, as long as the congestion window is open over the WiFi interface, Multipath TCP's RTT-based scheduler [8] prefers to send packets over WiFi. Indeed, 94.88% of connections in that case have a better average RTT on WiFi than 4G (this percentage is 75.44% if the maximal RTT seen is considered).

When setting the default route via the cellular interface, Figure 3 shows that most of short connections still use this one (as annotated as 1), but it concerns only 53.22% of all connections. It seems that even if the cellular interface is the default one, connections still mainly use WiFi, even for connections exchanging less than 1 KB. This occurs for connections that do not push data as fast as possible data. If the connection lasts more than two RTT, Multipath TCP has enough time to establish the second subflow. The packet scheduler will select the flow with the lowest RTT — 82.74% of all connections have a WiFi flow with a lower maximal RTT than the cellular one.

This explains the bottom of figure 3 (annotated as 2): a set of Firefox connections transferring fewer than 10 KB uses nearly exclusively the WiFi interface. A closer look at the packet trace reveals that these connections are part of the connection pool managed by Firefox. This behavior does not happen with other applications. When Firefox creates a connection in the pool, the initial handshake and the SOCKS command to our SOCKS server are sent. These packets are exchanged over the cellular interface and Firefox does not immediately send data over these established connections. This leaves enough time to Multipath TCP to create the subflow over the WiFi interface and measure its RTT. When Firefox starts to transmit data over such a connection, the RTT-based scheduler used by Multipath TCP prefers the WiFi subflow and no data (except the initial SOCKS command) is sent over the cellular one.

When the applications push more data over the Multipath TCP connection, the distribution of the traffic between the cellular and the WiFi interface also depends on the evolution of the congestion windows over the two subflows. If the application pushes data at a low rate, then the packet scheduler will send it over the lowest-RTT interface (WiFi in our case). However, this can be fragile. If one packet is lost, then the congestion window is reduced and the next data might be sent over the other interface. If the application pushes data at a higher rate, then the congestion window over the lowest-RTT interface is not large enough and the packet scheduler will send data over the second subflow.

4. DISCUSSION

In this paper, we have proposed and implemented[3] a measurement testing framework that enables researchers to conduct reproducible experiments with traffic generated by real applications. We have used it to evaluate how these applications interact with Multipath TCP. Our initial results show that smartphone applications work well with Multipath TCP. However, they also show that Multipath TCP uses too many subflows for short connections and that the default route has an impact on the traffic distribution. Our measurement framework will enable researchers to assess the impact of proposed modifications to Multipath TCP on real applications.

5. REFERENCES

[1] Chen, Y.-C., et al. A Measurement-based Study of Multipath TCP Performance over Wireless Networks. In *IMC'13* (2013).

[2] Deng, S., et al. WiFi, LTE, or both?: Measuring multi-homed wireless internet performance. In *IMC '14* (2014), ACM, pp. 181–194.

[3] Ford, A., et al. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, January 2013.

[4] Lee, K., et al. Mobile data offloading: How much can wifi deliver? In *SIGCOMM '10* (2010).

[5] Lim, Y.-s., et al. How green is Multipath TCP for mobile devices? In *AllThingsCellular '14* (2014), ACM.

[6] Paasch, C., Barre, S., et al. Multipath TCP in the Linux kernel. `http://www.multipath-tcp.org`.

[7] Paasch, C., et al. Exploring Mobile/WiFi Handover with Multipath TCP. In *ACM SIGCOMM CellNet workshop* (2012), pp. 31–36.

[8] Paasch, C., et al. Experimental evaluation of Multipath TCP schedulers. In *CSWS '14* (2014).

[3] All our code has been released on `multipath-tcp.org`