Observing Real Smartphone Applications over Multipath TCP

Quentin De Coninck [1], Matthieu Baerts [2], Benjamin Hesmans [1], Olivier Bonaventure [1]
[1]ICTEAM, Universite catholique de Louvain, Louvain-la-Neuve, Belgium
[2]Tessares, Louvain-la-Neuve, Belgium
[1] `first.last@uclouvain.be`
[2] `matthieu.baerts@tessares.net`

***Abstract*—A large fraction of the smartphones have both cellular and WiFi interfaces. Despite of this, smartphones rarely use them simultaneously because most of their data traffic is controlled by TCP which can only use one interface at a time. Multipath TCP is a recently standardized TCP extension that solves this problem. Smartphone vendors have started to deploy Multipath TCP, but the performance of Multipath TCP with real smartphone applications has not been studied in details yet. To fill this gap, we port Multipath TCP on Android smartphones and propose a framework to analyze the interactions between real network-heavy applications and this new protocol. We use eight popular Android applications and analyze their usage of the WiFi and cellular networks (especially 4G/LTE).**

I. Introduction

SMARTPHONES are the most popular mobile multi-homed devices. Many users expect that their smartphones will be able to seamlessly use all available WiFi and cellular networks. Unfortunately, reality tells us that seamless coexistence between cellular and WiFi is not as simple as what users would expect despite the huge investments in both cellular and WiFi networks by large network operators.

Several cellular/WiFi coexistence technologies have been proposed during the last years [1]. Some of them have been deployed. Recently, Multipath TCP [2] received a lot of attention when it was selected by Apple to support its voice recognition (Siri) application. Siri leverages Multipath TCP to send voice samples over both WiFi and cellular interfaces to cope with various failure scenarios. As of this writing, Siri is the only deployed smartphone application that explicitly uses Multipath TCP. But there is no public information about the benefits of using Multipath TCP with it. In July 2015, Korea Telecom announced at IETF 93 that they use Multipath TCP on the Samsung Galaxy S6 smartphones to provide their users a higher bandwidth.

Multipath TCP is a TCP extension that allows sending 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 [2], over each interface. Once the subflows are established, data can be sent over any of the subflows thanks to the Multipath TCP *scheduler*. Researchers have analyzed the performance of Multipath TCP in such hybrid networks [3], [4], [5], [6]. Their measurements show that Multipath TCP can indeed provide benefits by pooling network resources or enabling seamless handovers. However, these analyses were 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 presenting two main contributions that improve our understanding of the interactions between smartphone applications and the protocol stack. After a brief overview of Multipath TCP, we first propose a measurement methodology that automates user actions on Android smartphone applications. These actions trigger the creation of real connections. We then analyze how eight popular smartphone applications interact with Multipath TCP under different network conditions with both WiFi and cellular networks. Our measurements indicate that Multipath TCP works well with existing smartphone applications. Finally, we summarize the key lessons learned from this analysis.

II. Multipath TCP and Related Work

Multipath TCP is a recent TCP extension that enables the transmission of the data belonging to one connection over different paths or interfaces [2]. A Multipath TCP connection is a logical association that provides a bytestream service. Compared to other multi-path transport layer solutions such as SCTP, Multipath TCP can be deployed on TCP-compatible networks. To request the utilization of Multipath TCP, the smartphone adds the `MP_CAPABLE` option in `SYN` segment sent over its `default` interface (for instance, WiFi). This option contains some flags and a key [2]. If the server supports Multipath TCP, it includes its key in the `MP_CAPABLE` option sent in the

SYN+ACK. According to the Multipath TCP terminology, this TCP connection is called the initial subflow [2]. The smartphone can use it to exchange data over the WiFi interface. If the smartphone also wants to send data for this connection over its cellular interface, it sends a new SYN segment with the MP_JOIN option over this interface. This option contains a token derived from the key announced by the server in the MP_CAPABLE option. This token identifies the Multipath TCP connection on the server side. The server replies with a SYN+ACK containing the MP_JOIN option and the second subflow is established. Multipath TCP sends data over any of the available subflows. Two levels of sequence numbers are used by Multipath TCP : the regular TCP sequence number and the Data Sequence Number (DSN). The DSN is associated to the bytestream. When data is sent over a subflow, its DSN is mapped to the regular sequence numbers with the DSS option that also contains DSN acknowledgements. When losses occur, Multipath TCP can retransmit data over a different subflow. To achieve that, the device sends on another subflow a packet with the same DSN containing data. This operation is called a reinjection [7]. Although at subflow level it looks like a new packet, a reinjection can be detected by looking at its DSN to see if it was previously sent on another subflow.

The operation of a Multipath TCP implementation depends on several algorithms that are not standardized by the IETF. First, the *path manager* defines the strategy used to create and delete subflows. Second, the *packet scheduler* [8] selects, among the active subflows that have an open congestion window, the subflow that will be used to send the data.

Various researchers have analyzed the performance of Multipath TCP through measurements. Raiciu et al. [9] discuss how Multipath TCP can be used to support mobile devices and provide early measurement results. Chen et al. [4] analyze the performance of Multipath TCP in WiFi/cellular networks by using bulk transfer applications running on laptops. Deng et al. [6] compare the performance of single-path TCP over WiFi and LTE networks with Multipath TCP on multi-homed devices by using active measurements and replaying HTTP traffic observed on mobile applications. They show that Multipath TCP provides benefits for long flows but not for short ones, for which the selection of the interface for the initial subflow is important from a performance viewpoint.

A. Multipath TCP on Android smartphones

Several backports of the Multipath TCP kernel on Android smartphones were released in the last years. However, these ports were often based on old versions of the Multipath TCP kernel. For this work, we rely on a backport of the latest version 0.89v5 of the Multipath TCP Linux kernel[1] on a Nexus 5 running Android 4.4.4. It should be noted that the Linux kernel used on such Android devices is tweaked to use only one interface at a time. We disable this function and configure the kernel to be able to simultaneously use two interfaces. The Multipath TCP kernel controls the utilization of the available interfaces thanks to a path manager. We use the Full Mesh path manager that creates a subflow over all network interfaces for each established TCP connection. To spread packets over the available paths, we use the default RTT-based scheduler [8] which sends packets over the available path with the lowest Round-Trip-Time.

[1]Available from www.multipath-tcp.org

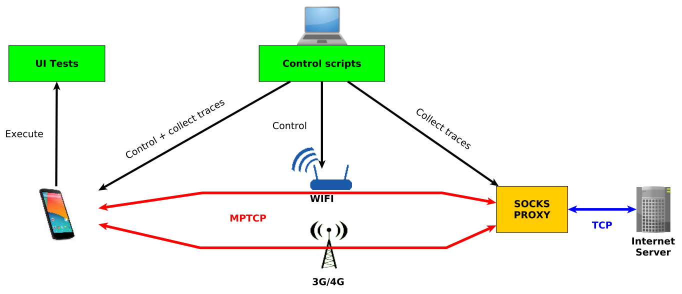


Figure 1. High-level view of the test framework.

Most popular smartphone applications use TCP to interact with servers managed by the application developers. As of this writing, it has not been possible to convince them to install Multipath TCP on their servers. To overcome this issue, we configure the smartphone to use a Multipath TCP capable SOCKS proxy server for all its connections as shown in Fig. 1. This is exactly the same setup as the one that was launched commercially in Korea in June 2015. Each (Multipath) TCP 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 via the proxy. The SOCKS server itself uses ShadowSocks and is configured to use the minimum encryption scheme to reduce the overhead. The other settings are set to the recommended values[2]. On the smartphone, we use the standard Android ShadowSocks client.

III. Automating measurements

In order to collect a large number of measurements, we developed a test framework that automates the interactions with these applications[3]. A high-level overview is shown on Fig. 1. On this basis, we identify two main tasks: *controlling devices* and *mimicking user interaction.*

The devices are controlled by Python and shell scripts (3100 lines split into different modules). Our controller checks their availability of the smartphones and the wireless networks, collects packet traces and modifies settings such as the protocol (either TCP or Multipath TCP) or the interfaces (WiFi, cellular or both) used by the smartphone. It was designed to be reusable, modular using

[2]See http://shadowsocks.org/en/config/advanced.html

[3]Results are reproducible, instructions are publicly available. See http://github.com/MPTCP-smartphone-thesis/uitests

parameters and to cope with unexpected situations caused by unreliability of this kind of devices.

User interactions are simulated through application UI tests to produce each high-level scenario. Each of the eight selected applications has its own UI test. These UI tests are implemented by using the MonkeyRunner Android UI testing tool. Each unit test is implemented as a new program and all of them use a shared `Utils` class. Thanks to this class, our framework allows to build a scenario with less than a few hundred lines of code. Each test was designed to resist different unusual situations, such as the failure of the smartphone, the failure of one of the wireless networks or unexpected reaction of the application. The measurements presented in this article were performed with the versions of the applications released on November 15th, 2014. To avoid network optimization and have repeatable measures, cached files are deleted when launching our tests.

All the tests described in this paper were performed during the night to reduce interferences with other users on the networks. The WiFi network was provided by a controlled router with an `802.11n` interface on the 5 GHz frequency band with a bit rate of 65 to 72 Mbps. The router was connected with a 100 Mbps link to the university network. We ensured that no other WiFi network was emitting in this frequency band in the building. The cellular network is a commercial one and we configured either 3G or 4G on the smartphone. The test scenarios were run in a random order each day to limit correlation of the results with the time at which they were launched.

A. *Test scenarios*

Now, we provide an overview of the scenarios used to generate network traffic. Our test scenarios can be split into two categories: *upload intensive* scenarios and *download intensive* scenarios. Each test takes less than 120 seconds.

1) 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.

2) 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.

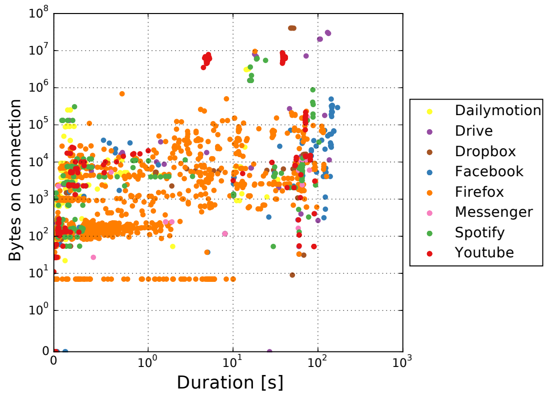


Figure 2. Duration and data transferred by the smartphone applications.

We used these applications on the testbed shown in Fig. 1. This setup allows us to capture all the packets sent by both the smartphone and the SOCKS server. We captured more than 110000 connections over about 1400 different tests conducted in February and March 2015 carrying more than 15 GBytes of data. The entire dataset is publicly available[4].

IV. Measurements

We use our test framework to analyze the interactions between smartphone applications and the network under various conditions. We first observe our applications over regular TCP, then we study how they behave over Multipath TCP. We use `tstat` [10] and `mptcptrace` [11] to extract information from packet traces.

A. *Single-path measurements*

The selected applications interact in different ways with the underlying transport protocol. An important factor that influences the performance of TCP is the lifetime of the connections and the number of bytes that are exchanged. To study this factor, we analyze the TCP connections established by our studied applications. Figure 2 shows that they create different types of TCP connections. 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. Unsurprisingly, streaming and cloud storage scenarios with Dropbox (31.75%), Youtube (29.7%), Drive (19.9%), Dailymotion (9.6%) and Spotify (5%) are the applications that exchange the largest volume in bytes. On the other hand, our Facebook scenario generates long TCP connections that do not exchange too many bytes.

Some of the connections that we observe are caused by the utilization of a SOCKS proxy. There are hundreds

[4]See http://multipath-tcp.org/data/IEEEComMag16

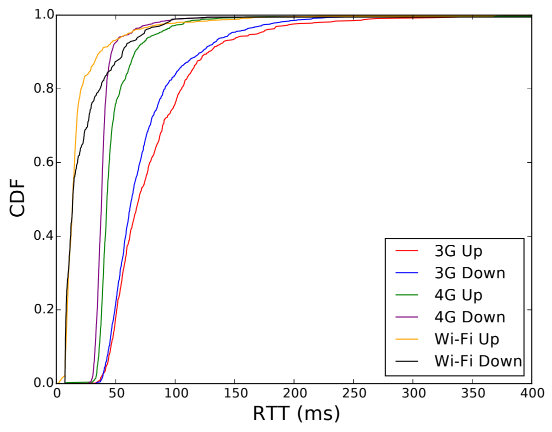


Figure 3. Average Round-Trip-Time of the TCP connections over the WiFi, 3G and 4G networks.

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. This command contains the IPv4 address and destination port used by the SOCKS proxy to establish the regular TCP connection to the remote servers. Most of the short connections that only transfer about 100 bytes are the DNS requests that are sent over TCP by the SOCKS client.

Our connections can be categorized in 3 types : *(i)* the short connections carrying a relatively small amount of data, *(ii)* the long connections carrying most of the data, and *(iii)* long-lived connections carrying a small amount of data. In our tests, 74% of the connections last less than 1 second. Among the connections that last more than 1 second, 32% carry more than 10 KB and represent 98.6% of the overall volume. Finally, the remaining 68% of the connections that last more than 1 second exchange less than 10 KB of data. This tends to match many measurement studies which identified that most TCP connections are short and most of the traffic is carried by a small fraction of all TCP connections [12].

The Round-Trip-Time is one of the key factors that influence the performance of TCP connections. We used `tstat` to compute the average RTT for each of the captured TCP connections. Figure 3 provides the CDF of the RTT measures among all the TCP connections used in the upstream (data sent by the smartphone) and downstream directions. The 4G network exhibits an RTT in upstream with a median of 42.6 msec and a mean of 50 msec. In the downstream direction, the median RTT increases up to 38.1 msec. On the WiFi network, 60% of the connections have an RTT shorter than 15.4 msec. Unsurprisingly, there is some bufferbloat on the 3G network, mainly in the upstream direction, but the bufferbloat remains reasonable compared to other networks [13].

B. Multipath measurements

The previous section showed that our measurement scenarios cover different utilizations of TCP. 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.

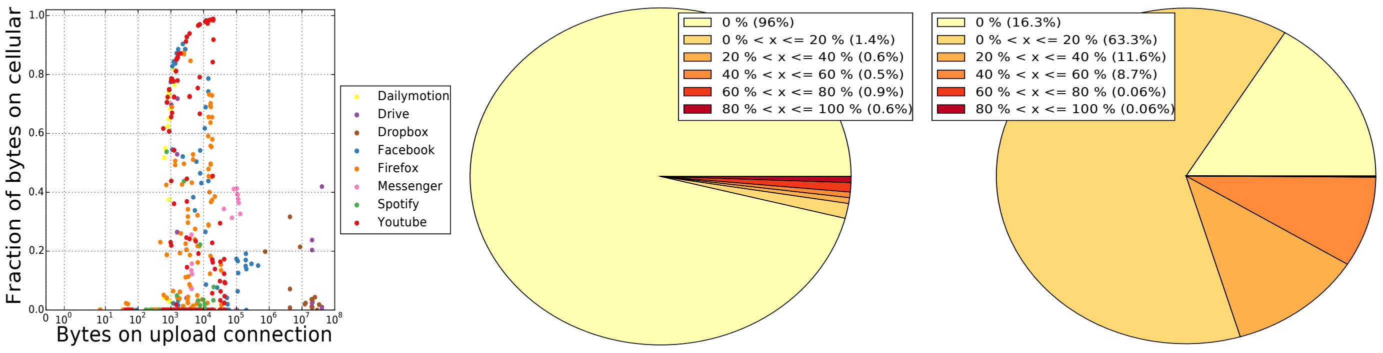
Multipath TCP can be used in different modes [3] on smartphones. For our measurements, we focus on a configuration where Multipath TCP tries to pool the resources of the cellular and the WiFi interfaces simultaneously since the handover and backup performance has already been studied in [3].

When a 4G and a WiFi interface are pooled together it is interesting to analyze which fraction of the traffic is sent over which interface. With the Multipath TCP implementation in the Linux kernel, this fraction depends on the interactions between the congestion control scheme, the packet scheduler, the underlying networks and the application.

We first consider Multipath TCP connections using WiFi and 4G interfaces, with WiFi set as the `default` interface. In Fig. 4a, each point corresponds to one Multipath TCP connection, and the x axis indicates the number of bytes transfered by this connection from the smartphone to servers. Although we observe connections using both WiFi and cellular interfaces, Fig. 4b shows that 96% of the connections only use the WiFi interface. However, Fig. 4c indicates that those connections are small since they carry only 16.3% of all the data bytes contained in the considered connections.

Several factors explain why Multipath TCP does not use the cellular network for these short connections. The first factor 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 the WiFi interface. This is the standard configuration of Android smartphones that prefer the WiFi interface when it is active. If the Multipath TCP connection is short and only transfers a few KiloBytes or less, 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. 71% of the connections sending only on WiFi interface are in this case. Furthermore, the RTT over the WiFi interface is shorter than over the cellular interface. This implies that most of the time, as long as the congestion window is open over the WiFi interface, Multipath TCP's RTT-based scheduler [8] prefers to send packets over the WiFi interface. Indeed, 84% of the connections with both subflows established have a smaller average RTT on WiFi than on 4G.

Those factors explain why data on the short connections are exchanged only over the WiFi interface. We experimentally verified this by performing the same set of measurements with the `default` route pointed to the



a) Fraction of data bytes sent on a connection depending of its data size.

b) Connections classified by the percentage of data sent on cellular interface.

c) Data bytes classified by the percentage of data sent on cellular interface on the connection it belongs to.

Figure 4. When the default route points to the WiFi interface, Multipath TCP mainly uses this interface for the short connections.

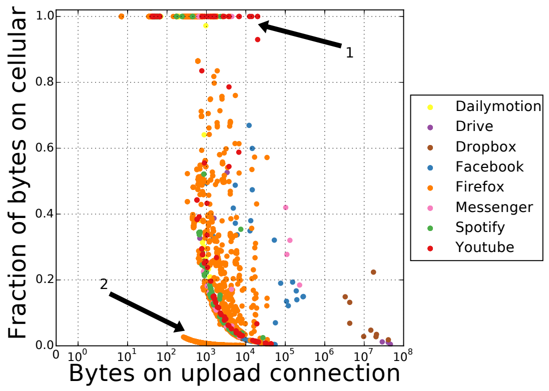


Figure 5. When the default route points to the cellular interface, many connections are aspired by the WiFi interface.

4G interface. Figure 5 shows that with this configuration most short connections still exclusively use the 4G network (see label 1 on Fig. 5), but this concerns only 65% of all connections. It seems that even if cellular is the `default` interface, many 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. If the connection lasts more than two RTTs, Multipath TCP has enough time to establish the second subflow. The packet scheduler will then select the subflow with the lowest RTT — 88% of connections using both subflows have a WiFi subflow with a lower average RTT than the cellular one.

This explains the bottom of Fig. 5 (annotated as 2): a group of Firefox connections that transfer less than 10 KB nearly exclusively use 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 this 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 the established connection. This leaves enough time for Multipath TCP to create the subflow over the WiFi interface and measure its RTT. When Firefox starts transmitting 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 subflow.

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 this case). However, this distribution 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.

In some cases, data transfered by Multipath TCP on one flow may be retransmitted again on the other flow. This phenomenon is called reinjection [7] and might limit the performance of Multipath TCP in some circumstances [14]. We used `mptcptrace` to compute the reinjections over all observed Multipath TCP connections. In our experiments (WiFi and 4G), reinjections in the upstream direction were rare (less than half one percent of all connections include a reinjection) and short (no more than 5 KB are reinjected on a connection). Looking at the proxy traces in the downstream direction, reinjections are observed on only 2% of all connections, and the largest observed reinjection is 30 KB on a 5 MB connection. This overhead is thus low.

An important benefit of the resource pooling capabilities of Multipath TCP is its ability to adapt to various networking conditions. When a smartphone moves, the performance of the WiFi and cellular interfaces often vary. Previous work with bulk transfer applications has shown that Multipath TCP can adapt to heterogeneous networks having different bandwidths and delays [15]. Our measurement framework also allows exploring the performance of smartphone applications under various network conditions. As an illustration, we analyze the packet traces collected when the smartphone is uploading a file with Dropbox. We first consider a WiFi access point attached to a DSL router having 1 Mbps of upstream bandwidth and 15 Mbps of downstream bandwidth. When the smartphone is attached to both this WiFi access point and the 4G network, it sends on average 91% of the data over the 4G network. This is expected because although the WiFi has better RTT, the congestion window of this path is quickly full and it slowly empties. In that case, the Multipath TCP scheduler selects the next available subflow with lowest RTT — here the cellular interface. Since the cellular network offers a larger bandwidth, Multipath TCP can take advantage of it and thus avoids being trapped in a low performance network for big connections.

As a second test case, we consider our standard WiFi access (around 70 Mbps in both streams) and the 4G network whose bandwidth is limited down to a few hundred kilobits per second. This is the shaping enforced by our cellular network once we reach the monthly traffic volume quota. In this case, 98.8% of the bytes are sent over the WiFi interface.

V. Conclusion

Multipath TCP is a new TCP extension that has a strong potential on smartphones as shown by its recent adoption by Apple and Korea Telecom. By enabling TCP connections to exchange data over cellular and WiFi interfaces, it brings new possibilities to improve the user experience. Apple's deployment focused on a single use case and little is currently known about the interactions between real smartphone applications and Multipath TCP. In this article, we have proposed and implemented a measurement testing framework that enables researchers to conduct reproducible experiments with traffic generated by real applications.

We have used our measurement framework to study the interactions between 8 very different smartphone applications covering several smartphone use cases and the latest version of the Multipath TCP implementation in the Linux kernel. Several lessons have already been learned from a first analysis of the packet traces that were captured. First, all the studied applications work without any modification with Multipath TCP. This confirms that Multipath TCP is compatible with existing applications. Second, for the short connections, that are often used by the studied applications, Multipath TCP uses the `default` route to forward the data for most connections. As suggested in [6], we confirm that the selection of this `default` route is thus an important decision on the smartphone. Third, for long connections, Multipath TCP enables the applications to pool the bandwidth on the cellular and WiFi interfaces and maintains good performance when one of them has bandwidth restrictions. This is important for the user's experience given that smartphones often associate to wireless networks by relying on metrics like signal-to-noise ratio.

We expect that our framework and the collected packet traces will be beneficial to Multipath TCP researchers and implementers by enabling them to study how improvements to the implementation would affect real applications in a reproducible manner. Moreover, this framework could be also used to measure the energy consumption impact of Multipath TCP on mobile devices like smartphones.

Acknowledgment

This work was partially supported by the EC within the FP7 Trilogy2 project. We would like to thank Gregory Detal and Sebastien Barre for the port of the latest Multipath TCP Linux kernel on the Nexus 5 and Patrick Delcoigne and his team for the cellular measurements.

References

[1] F. Rebecchi, M. Dias de Amorim, V. Conan, A. Passarella, R. Bruno, and M. Conti, "Data offloading techniques in cellular networks: A survey," *Communications Surveys Tutorials, IEEE*, vol. 17, no. 2, pp. 580–603, Secondquarter 2015.

[2] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, Internet Engineering Task Force, January 2013.

[3] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, "Exploring Mobile/WiFi Handover with Multipath TCP," in *ACM SIGCOMM CellNet workshop*, 2012, pp. 31–36.

[4] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath tcp performance over wireless networks," in *Proceedings of the 2013 conference on Internet measurement conference.* ACM, 2013, pp. 455–468.

[5] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and R. J. Gibbens, "How green is multipath tcp for mobile devices?" in *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges.* ACM, 2014, pp. 3–8.

[6] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "Wifi, lte, or both?: measuring multi-homed wireless internet performance," in *Proceedings of the 2014 Conference on Internet Measurement Conference.* ACM, 2014, pp. 181–194.

[7] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath tcp," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation.* USENIX Association, 2012, pp. 29–42.

[8] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath tcp schedulers," in *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop.* ACM, 2014, pp. 27–32.

[9] C. Raiciu, D. Niculescu, M. Bagnulo, and M. Handley, "Opportunistic Mobility with Multipath TCP," in *ACM MobiArch 2011*, 2011.

[10] M. Mellia, A. Carpani, and R. L. Cigno, "Tstat: Tcp statistic and analysis tool," in *Quality of Service in Multiservice IP Networks.* Springer, 2003, pp. 145–157.

[11] B. Hesmans and O. Bonaventure, "Tracing multipath tcp connections," in *Proceedings of the 2014 ACM conference on SIGCOMM.* ACM, 2014, pp. 361–362.

[12] A. Gember, A. Anand, and A. Akella, "A comparative study of handheld and non-handheld traffic in campus wi-fi networks," in *Passive and Active Measurement.* Springer, 2011, pp. 173–183.

[13] S. Ferlin-Oliveira, T. Dreibholz, and O. Alay, "Tackling the challenge of bufferbloat in multi-path transport over heterogeneous wireless networks," in *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of.* IEEE, 2014, pp. 123–128.

[14] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and K.-W. Lee, "Cross-layer path management in multi-path transport protocol for mobile devices," in *INFOCOM, 2014 Proceedings IEEE.* IEEE, 2014, pp. 1815–1823.

[15] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath tcp," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies.* ACM, 2013, pp. 393–398.

Quentin De Coninck (quentin.deconinck@uclouvain.be) received his B. Eng. and M. Eng. degrees in computer engineering from Université Catholique de Louvain, Belgium, in 2013 and 2015, respectively. He is now pursuing a Ph. D. in the same institution. His research interests include applications of mobility in computer networks and low-level system architecture.

Matthieu Baerts (matthieu.baerts@tessares.net) received his M. Sc. degree in computer science in 2015 from Université Catholique de Louvain, Belgium. He is now working with Tessares s.a. which is currently developing hybrid Internet access bonding solutions using Multipath TCP protocol. Beside that he is involved in Free Software communities hoping contributing to technologies available for everyone.

Benjamin Hesmans (benjamin.hesmans@uclouvain.be) is a third year PhD student at Université Catholique de Louvain (Louvain-La-Neuve, Belgium). His main interest is Multipath TCP performance understanding and its improvements. He has written open source tools such as MPTCPTrace that provides detailed graphical and statistical information on MPTCP connections based on packet traces.

Olivier Bonaventure (olivier.bonaventure@uclouvain.be) is professor at Université catholique de Louvain (UCL) in Louvain-la-Neuve, Belgium, where he leads the IP Networking Lab (INL) (http://inl.info.ucl.ac.be). His research focuses on improving Internet protocols. His recent work includes various improvements to routing protocols, Multipath TCP and Segment Routing. He currently serves as Editor for SIGCOMM Computer Communication Review.