A First Analysis of Multipath TCP on Smartphones

Quentin De Coninck[1], Matthieu Baerts[2], Benjamin Hesmans[1], and Olivier Bonaventure[1]

[1] Université catholique de Louvain, Louvain-la-Neuve, Belgium
{quentin.deconinck, benjamin.hesmans, olivier.bonaventure}@uclouvain.be
[2] Tessares SA, Louvain-la-Neuve, Belgium
matthieu.baerts@tessares.net
smartphone.multipath-tcp.org

Abstract. Multipath TCP is a recent TCP extension that enables multihomed hosts like smartphones to send and receive data over multiple interfaces. Despite the growing interest in this new TCP extension, little is known about its behavior with real applications in wireless networks. This paper analyzes a trace from a SOCKS proxy serving smartphones using Multipath TCP. This first detailed study of real Multipath TCP smartphone traffic reveals several interesting points about its behavior in the wild. It confirms the heterogeneity of wireless and cellular networks which influences the scheduling of Multipath TCP. The analysis shows that most of the additional subflows are never used to send data. The amount of reinjections is also quantified and shows that they are not a major issue for the deployment of Multipath TCP. With our methodology to detect handovers, around a quarter of the connections using several subflows experience data handovers.

1 Introduction

TCP is the dominant transport protocol, both on the wired Internet and in wireless networks. Over the years, TCP has evolved and included various optimizations. Multipath TCP is the last major extension to TCP [9], [20]. It enables a multihomed host to exchange data for a single connection over different interfaces.

Multipath TCP was standardized in early 2013. Although the extension is still young, it is already used to support several commercial services. In September 2013, Apple has deployed Multipath TCP on hundreds of millions of smartphones and tablets to improve the user experience for the Siri voice recognition application. In July 2015, Korean Telecom announced that they have enabled Multipath TCP on Android smartphones to bond WiFi and LTE together. These smartphones reach download speeds of 800 Mbps and more. In September 2015, OVH, a French ISP and hosting provider, announced their OverTheBox service that uses Multipath TCP to enable SMEs to bond several DSL over cable links together. Other use cases are being explored and it can be expected that Multipath TCP traffic will grow in the coming years.

Despite the important role that Multipath TCP could play on smartphones, little is known about its behavior with real applications. Most of the articles on Multipath TCP performance relied on lab measurements [15], [19] or were carried out with test applications [1], [7, 8].

This paper provides the first detailed analysis of the operation of Multipath TCP on smartphones used by real users. Since Multipath TCP is not yet deployed on Internet and cloud servers, installing a Multipath TCP kernel is not sufficient to automatically generate Multipath TCP traffic. To benefit from Multipath TCP, a SOCKS proxy had to be installed on a server supporting Multipath TCP and the smartphones were configured to use the SOCKS server as their relay for all connections. This is the same setup as KT's commercial deployment. By sharing the studied trace, the measurement tools and the analysis, this paper improves our understanding of the dynamics of this new protocol.

This paper is organized as follows. It first provides a brief overview of Multipath TCP and discusses related work in Sect. 2. It describes the collected dataset in Sect. 3 and gives first characteristics in Sect. 4. In Sect. 5, it takes a closer look at the performances of Multipath TCP. It concludes in Sect. 6 with the main lessons that we learned from this first detailed analysis of Multipath TCP packet traces.

2 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 [9]. A Multipath TCP connection is a logical association that provides a bytestream service. To understand its operation, let us see briefly an example on how a smartphone could use Multipath TCP. To request the utilization of Multipath TCP, the smartphone adds the `MP_CAPABLE` option in `SYN` segment sent over its cellular interface. This option contains some flags and a key [9]. 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 [9]. The smartphone can use it to exchange data over the cellular interface. If the smartphone wants to also send data for this connection over its WiFi 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. At this stage, the Multipath TCP connection contains two subflows, but this number is not fixed. The WiFi subflow can stop when the smartphone goes away from access point. At this point, the smartphone can advertise the proxy that it lost one address through a `REMOVE_ADDR` sent unreliably in TCP options. Another subflow can be created when another IP address is learned from a different access point. 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 corresponds to the Multipath TCP bytestream and 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. This operation is called a reinjection [20].

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. The smartphones use the `full-mesh` path manager that creates one subflow over each pair of interfaces as soon as the initial subflow has been fully established or as soon as a new address has been learned. Second, the *packet scheduler* [16] selects, among the active subflows that have an open congestion window, the subflow that will be used to send the data. The smartphones and the proxy used the default Multipath TCP scheduler in the Linux kernel that prefers the subflow with the smallest RTT. Third, the *congestion controller*. Here, the standard one (LIA) was used.

Various researchers have analyzed the performance of Multipath TCP through measurements. Raiciu et al. [19] discuss how Multipath TCP can be used to support mobile devices and provide early measurement results. Paasch et al. [15] propose three modes for the operation of Multipath TCP in wireless networks and analyse measurements of handovers. Chen et al. [1] analyze the performance of Multipath TCP in WiFi/cellular networks by using bulk transfer applications running on laptops. Ferlin et al. [8] analyze how Multipath TCP reacts to bufferbloat and propose a mitigation technique. As of this writing, this mitigation technique has not been included in the Linux Multipath TCP implementation. Ferlin et al. [7] propose a probing technique to detect low performing paths and evaluates it in wireless networks. Deng et al. [4] 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 interface for the initial subflow is important from a performance viewpoint. Hesmans et al. [11] analyze a one week-long server trace supporting Multipath TCP.

3 Dataset

Although Multipath TCP is already used by hundred of millions of Apple smartphones to support the Siri voice recognition application, it is difficult to collect both WiFi and cellular traces without cooperation from an ISP. Instead, a Multipath TCP capable SOCKS proxy was set up (like KT) and this analysis focuses on the Multipath TCP implementation in the Linux kernel [14]. This implementation is distributed from `http://multipath-tcp.org` and can be integrated in Android.

The dataset covers the traffic produced by a dozen of users using Nexus 5 smartphones running Android 4.4 with a modified Linux kernel that includes Multipath TCP v0.89.5. These users were either professors, PhD or Master stu-

dents at Université catholique de Louvain. While some of them used their device to go only on the Internet, others are still using them as their main phone. However, installing Multipath TCP on the smartphones is not sufficient to use it for all connections established by applications. As of this writing, there are probably only a few dozens of Multipath TCP enabled servers on the Internet and these are rarely accessed by real smartphone applications. To force these applications to use Multipath TCP, `ShadowSocks`[1] was installed on each smartphone and configured to use a SOCKS server that supports Multipath TCP for all TCP connections. Note that since `ShadowSocks` does not support IPv6, this trace only contains IPv4 packets. The smartphones thus use Multipath TCP over their WiFi and cellular interfaces to reach the SOCKS server and this server uses regular TCP to interact with the final destinations. From the server side, all the connections from the dozen smartphones appear as coming from the SOCKS server. This implies that the external (cellular or WiFi) IP address of the smartphone is not visible to the servers that it contacts. This might affect the operation of some servers that adapt their behavior (e.g. the initial congestion window) in function of the client IP address. Moreover, note that the `ShadowSocks` client sends DNS requests over TCP.

A special Android application [3] managing the utilization of the cellular and WiFi interfaces was also installed on each smartphone. Smartphones with Android 4.4 assume that only one wireless interface is active at a time. When such a smartphone switches from cellular to WiFi, it automatically resets all existing TCP connections by using Android specific functions. This application enables the cellular and WiFi interfaces simultaneously. It also controls the routing tables and updates the policy routes that are required for Multipath TCP every time the smartphone connects to a wireless network. Thanks to this application, the modified Nexus 5 can be used by any user since it does not require any networking knowledge.

The SOCKS proxy ran `tcpdump` to collect all the packets exchanged with the smartphones. Measurements were performed in Belgium from March 8th to April 28th 2015. Over this period of 7 weeks, more than 71 millions Multipath TCP packets were collected for a total of 25.4 GBytes over 390,782 Multipath TCP connections.[2] To our knowledge, there is no equivalent public dataset. The analysis scripts are also open-sourced [2, 3].

4 Characterization of the Trace

The main characteristics of the Multipath TCP connections in the dataset are first analyzed. The destination ports of the captured packets are not sufficient to identify the application level protocol. Since the smartphone connects through a SOCKS proxy, all the packets are sent towards the destination port used by the proxy (443 to prevent middlebox interferences). The real destination port is extracted from the SOCKS command sent by the `ShadowSocks` client at the

[1] Available at `http://shadowsocks.org`.
[2] Anonymized traces available: `http://crawdad.org/uclouvain/mptcp_smartphone`.

Table 1. Statistics about destination port fetched by smartphones.

Port	# connections	% connections	Bytes	% bytes
53	107,012	27.4	17.4 MB	< 0.1
80	103,597	26.5	14,943 MB	58.8
443	104,223	26.7	9,253 MB	36.4
4070	571	0.1	91.7 MB	0.4
5228	10,602	2.7	27.3 MB	0.1
8009	10,765	2.8	0.97 MB	< 0.1
Others	54,012	13.8	1,090 MB	4.3

beginning of each connection. As shown on Tab. 1, most of the connections and data bytes are related to Web traffic. Since `ShadowSocks` sends DNS requests over TCP, it is expected to have a large fraction of the connections using port 53. Among other popular port numbers, there are ports 4070 — e.g., used by Spotify —, Google Services (5228) and Google Chromecast (8009).

65% of the observed connections last less than 10 seconds. In particular, 4.3% are failed connections, i.e. the first SYN was received and answered by the proxy, but the third ACK was lost (or a RST occurred). 20.8% of the connections last more than 100 seconds. Six of them last for more than one entire day (up to nearly two days).

Looking at the bytes carried by each connection, most (86.9%) of them carry less than 10 KBytes. In particular, 3.1% of the connections carry between 9 and 11 bytes. Actually, those are empty connections, since the SOCKS command are 7 bytes long, two bytes are consumed by the SYNs and the use of the remaining two bytes depend on how the connections were closed (RST or FIN). The longest connection in terms of bytes transported around 450 MBytes and was spread over five subflows.

5 Analysis

In the following, the analysis will focus on relevant subsets of the trace such as connections with at least two subflows, connections using at least two subflows or connections experiencing handover. Table 2 gives the characteristics of these subsets. They are used to analyze how Multipath TCP subflows are created (Sect. 5.1), study the heterogeneity of the available networks in terms of round-trip-times (Sect. 5.2), estimate the packet reordering of Multipath TCP (Sect. 5.3), study how subflows are used (Sect. 5.4), quantify the reinjection overhead (Sect. 5.5) and identify connections experiencing handovers (Sect. 5.6).

5.1 Establishment of the subflows

With Multipath TCP, a smartphone can send data over various paths. The number of subflows that a smartphone creates depends on the number of active interfaces that it has and on the availability of the wireless networks.

Table 2. The different (sub)traces analyzed in this section.

Name	Description	# connections	Bytes to proxy	Bytes from proxy
$\mathcal{T}_0$	Full trace	390,782	652 MB	24,771 MB
$\mathcal{T}_1$	At least 2 established subflows	126,040	238 MB	13,496 MB
$\mathcal{T}_2$	At least 2 used subflows	32,889	152 MB	11,856 MB
$\mathcal{T}_3$	With handover	8,461	36.7 MB	4,626 MB

Table 3. Number of subflows per Multipath TCP connection.

Number of subflows	1	2	3	4	5	>5
Percentage of connections	67.75%	29.96%	1.07%	0.48%	0.26%	0.48%

Table 3 reports the number of (not necessarily concurrent) subflows that are observed in $\mathcal{T}_0$. Most of the connections only have one subflow. On another side, 2.29% of the connections have more than two subflows. Having more subflows than the number of network interfaces is a sign of mobility over different WiFi and/or cellular access points since IPv6 was not used. A connection establishing 42 different subflows was observed.

Another interesting point is the delay between the establishment of the connection (i.e. the first subflow) and the establishment of the other subflows. The smartphone tries to create subflows shortly after the creation of the Multipath TCP connection and as soon as a new interface gets an IP address. Late joins can mainly be expected when a smartphone moves from one network access point to another. To quantify this effect, Fig. 1 plots the CDF of the delays between the creation of each Multipath TCP connection and all the additional subflows that are linked to it. 57.4% of all the additional subflows are established within 200 ms. This percentage increases to 72.2% if this limit is set to one second. If the analysis is restricted to the first additional subflow, these percentages are respectively 61.7% and 77.5%. Joins can occur much after the connection is established. Indeed, 13.5% of the additional subflows were established one minute after the establishment of the connection, and 1.5% of them were added one hour later. The maximal observed delay is 134,563 seconds (more than 37 hours) and this connection was related to the Google Services. Those late joins suggests network handovers, and late second subflow establishments can be explained by smartphones having one network interface unavailable.

5.2 Subflows round-trip-times

From now, we focus on the subtrace $\mathcal{T}_1$ that includes all the connections with at least two subflows. A subflow is established through a three-way handshake like a TCP connection. Thanks to this exchange, the communicating hosts agree on the sequence numbers, TCP options and also measure the initial value of the round-trip-time for the subflow. For the used Linux implementation of Multipath TCP,

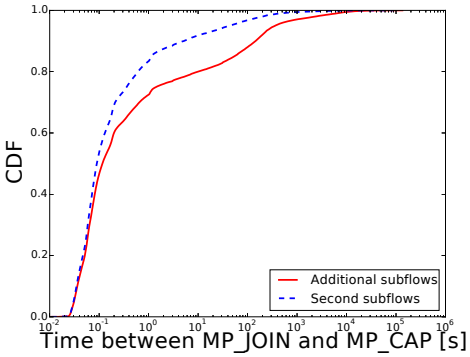

Fig. 1. Delay between the creation of the Multipath TCP connection and the establishment of a subflow.

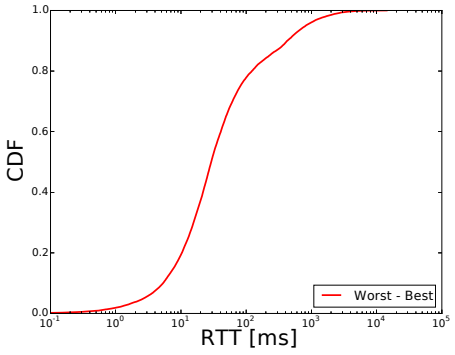

Fig. 2. Difference of average RTT seen by the proxy between the worst and the best subflows with at least 3 RTT samples.

the round-trip-time measurement is an important performance metric because the default packet scheduler prefers the subflows having the lowest round-trip-times.

To evaluate the round-trip-time heterogeneity of the Multipath TCP connections, the analysis uses `tstat` [13] to compute the average round-trip-time over all the subflows that a connection contains. Then, it extracts for each connection the minimum and the maximum of these average round-trip-times. To have consistent values, it only takes into account the subflows having at least 3 RTT estimation samples. Fig. 2 plots the CDF of the difference in the average RTT between the subflows having the largest and the smallest RTTs over all connections in $\mathcal{T}_1$. Only 19.4% of the connections are composed of subflows whose round-trip-times are within 10 ms or less whereas 77.9% have RTTs within 100 ms or less. 3.9 % of the connections experience subflows having 1 seconds or more of difference in their average RTT. With such network heterogeneity, if a packet is sent on a low-bandwidth and high-delay subflow s_0 and following packets are sent on another high-bandwidth low-delay one s_1, the sender may encounter head-of-line blocking.

5.3 Multipath TCP acknowledgements

As explained in section 2, Multipath TCP uses two ACK levels: the regular TCP ACKs at the subflow level and the cumulative Multipath TCP ACKs at the connection level. It is possible to have some data acknowledged at TCP level but not at Multipath TCP one, typically if previous data was sent on another subflow but not yet acknowledged. Fig. 3 plots in red-dotted curve the CDF of the number of bytes sent by the proxy that are acknowledged by non-duplicate TCP ACKs. This plot is a weighted CDF where the contribution of each ACK is weighted by the number of bytes that it acknowledges. In TCP, ACKs of 1428 bytes or less cover 50.7% of all acknowledged bytes and considering ACKs of 20 KB or less the percentage is 91.1%.

The same analysis is now performed by looking at the DSS option that carries the Multipath TCP Data ACKs with `mptcptrace` [10]. The green curve in Fig. 3 shows the weighted cumulative distribution of the number of bytes acked per Data ACK. Compared with the regular TCP ACKs, the Multipath TCP ACKs cover more bytes. Indeed, 51% of all bytes acknowledged by Multipath TCP are covered with Data ACKs of 2856 bytes or less, and this percentage increases to 70.6% considering Data ACKs of 20 KB or less.

The difference between the regular TCP ACKs and the Data ACKs is caused by the reordering that occurs when data is sent over different subflows. Since the Data ACKs are cumulative they can only be updated once all the previous data have been received on all subflows. If subflows with very different round-trip-times are used, it will cause reordering and data will possibly filling the receiver's window during a long period. This can also change the way applications read data which would be more by large bursts instead of small frequent reads. On mobile devices, such memory footprints should be minimized.

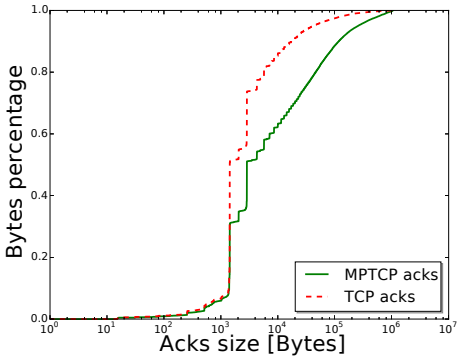

Fig. 3. Size of the Multipath TCP and TCP ACKs received by the proxy.

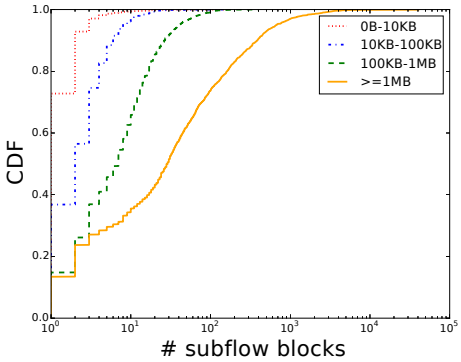

Fig. 4. Size of the subflow blocks from proxy to smartphones on $\mathcal{T}_1$.

5.4 Utilization of the subflows

The next question is how data is spread among the subflows. Does Multipath TCP alternates packets between the different subflows or does it send bursts of packets ? Again, to be relevant, the subtrace $\mathcal{T}_1$ is considered.

To quantify the spread of data, this paper introduces the notion of *subflow block*. Intuitively, a *subflow block* is a sequence of packets from a connection sent over a given subflow without any packet transmitted over another subflow. Consider a connection where a host sends N data packets. Number them as $0, ..., N-1$ with 0 the first data packet sent and $N-1$ the last one. Let f_i denote the subflow on which packet i was sent. The n^{th} subflow blocks b_n is defined as $b_n = \{\max(b_{n-1}) + 1\} \cup \{i \mid i - 1 \in b_n \text{ and } f_i = f_{i-1}\}$, with $b_0 = \{-1\}$ and

$f_{-1} = \perp$. As an example, if the proxy sends two data packets on s_0, then three on s_1, retransmits the second packet on s_0 and sends the last two packets on s_1, we will have $b_1 = \{0, 1\}$, $b_2 = \{2, 3, 4\}$, $b_3 = \{5\}$ and $b_4 = \{6, 7\}$. This notion is implemented in our analysis scripts [2]. A connection balancing the traffic with several subflows will produce lot of small subflow blocks whereas a connection sending all its data over one single subflow will have only one subflow block containing all the connection's packets. Figure 4 shows the number of subflow blocks that each connection contains. Each curve contains connections carrying their labeled amount of total bytes from proxy to smartphones. For most of the large connections, Multipath TCP balances well the packets over different subflows. In particular, 26.4% of connections carrying more than 1 MB have more than 100 subflow blocks. As expected, the shorter the connection is, more the subflow blocks tend to contain most of the connection traffic. For short connections carrying less than 10 KBytes, 72.8% of them contain only one subflow block, and therefore they only use one subflow. This number raises concerns about unused subflows. If connections having at least two subflows are considered, over their 276,133 subflows, 41.2% of them are unused in both directions. It is worth noting that nearly all of these unused subflows are actually additional subflows, leading to 75.6% of unused additional subflows. This is clearly an overhead, since creating subflows that are not used consumes bytes and energy [18] on smartphones since the interface over which these subflows are established is kept active.

There are three reasons that explain those unused subflows. Firstly, a subflow can become active after all the data has been exchanged. This happens frequently since 62.9% of the connections carry less than 2000 bytes of data. In practice, for 21% of the unused additional subflows the proxy received their third ACK after that it had finished to send data. Secondly, as suggested in Sect. 5.2, the difference in round-trip-times between the two available subflows can be so large that the subflow with the highest RTT is never selected by the packet scheduler. If the server does not transmit too much data, the congestion window on the lowest-RTT subflow remains open and the second subflow is not used. Though, 36.2% of the unused additional subflows have a better RTT for the newly-established subflow than the other available one. However, 59.9% of these subflows belong to connections carrying less than 1000 bytes (90.1% less than 10 KBytes). Thirdly, a subflow can be established as a backup subflow [9]. Indeed, a user can set the cellular subflow as a backup one, e.g., for cost purpose. 2.1% of the unused additional subflows were backup subflows.

5.5 Reinjections and retransmissions

In addition to unused subflows, another Multipath TCP specific overhead is the reinjections. A *reinjection* [20] is the transmission of the same data over two or more subflows. Since by definition, reinjections can only occur on connections that use at least two subflows, this analysis considers the subtrace $\mathcal{T}_2$. A reinjection can be detected by looking at the Multipath TCP Data Sequence Number (DSN). If a packet A with DSN x is sent first on the subflow 1 and after another packet B with the same DSN x is sent on the subflow 2, then B is a reinjection

of A. `mptcptrace` [10] was extended to detect them. A reinjection can occur for several reasons: *(i)* handover, *(ii)* excessive losses over one subflow or *(iii)* the utilization of the Opportunistic Retransmission and Penalization (ORP) algorithm [17], [20]. This phenomenon has been shown to limit the performance of Multipath TCP in some wireless networks [21]. Typically, Multipath TCP reinjections are closely coupled with regular TCP retransmissions. Figure 5 shows the CDF of the reinjections and retransmissions sent by the proxy. The number of retransmitted and reinjected bytes are normalized with the number of unique bytes sent by the proxy over each connection. 52.7% of the connections using at least two subflows experience retransmissions on one of their subflows whereas reinjections occur on 29.3% of them. This percentage of retransmissions tends to match previous analysis of TCP on smartphones [6], [12]. 68.7% of $\mathcal{T}_2$ connections have less than 1% of their unique bytes retransmitted, and 85% less than 10%. 79.7% of the connections have less than 1% of their unique bytes reinjected, and 89.8% less than 10%. Observing more retransmissions than reinjections is expected since retransmissions can trigger reinjections. In the studied trace, the impact of reinjections remains limited since over more than 11.8 GBytes of unique data sent by proxy, there are only 86.8 MB of retransmissions and 65 MB of reinjections. On some small connections, we observe more retransmitted and reinjected bytes than the unique bytes. This is because all the data sent over the connection was retransmitted several times. On Fig. 5 the thousand of connections having a fraction of retransmitted bytes over unique bytes greater or equal to 1 carried fewer than 10 KB of unique data, and 83.3% of them fewer than 1 KB. Concerning the reinjections, the few hundred of connections in such case carried less than 14 KB, 63.4% of them carried less than 1 KB and 76.1% of them less than 1428 bytes.

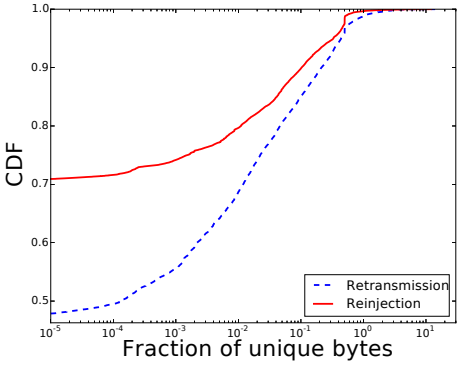

Fig. 5. Fraction of bytes that are reinjected/retransmitted by the proxy on $\mathcal{T}_2$.

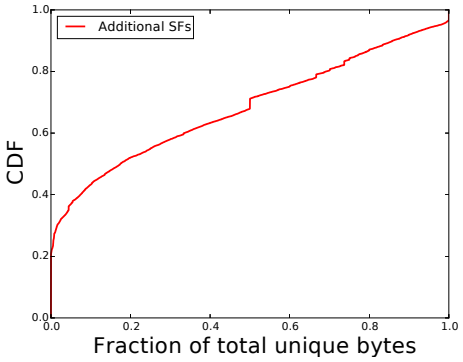

Fig. 6. Fraction of total data bytes on non-initial subflows sent by the proxy on $\mathcal{T}_3$.

5.6 Handovers

One of the main benefits of Multipath TCP is that it supports seamless handovers which enables mobility scenarios [9], [15]. A handover is here defined as a recovery of a failed subflow by another one. A naive solution is to rely on `REMOVE_ADDR`s to detect handover. However, this TCP option is sent unreliably. Indeed, 22.1% of the connections experiencing handover have no `REMOVE_ADDR`.

This paper proposes an alternative methodology implemented in [2] that relies on the TCP segments exchanged. Let LA_i be the time of the last (non-RST) ACK sent by the smartphone seen on subflow i (that was used to send data) and LP_j the time of the last (non-retransmitted) segment containing data on subflow j. If $\exists\ k, l \mid k \neq l$, no FIN seen from the smartphone on subflow k, $LA_l > LA_k$ and $LP_l > LA_k$, then the connection experiences handover. Notice that only handovers on the subflows carrying data are detected. Among the connections that use at least two subflows, 25.7% experience handover. It has also the advantage to be implementation independent since it does not use the `ADD_ADDR`s or `REMOVE_ADDR`s options that are not supported by all implementations [5].

Based on the subtrace $\mathcal{T}_3$, Fig. 6 shows the fraction of unique bytes that were sent by the proxy on the additional subflows on connections experiencing handover. This illustrates the connections that could not be possible if regular TCP was used on these mobile devices. Indeed, an handover is typically related to the mobility of the user who can go out of the reachability of a network. Notice that this methodology can also detect handover in the smartphone to proxy flow. Indeed, 20.4% of connections experience handover with all data sent by the proxy on the initial subflow because the smartphone sent data on another subflow after having lost the initial one.

6 Conclusion

This work brings the first results about real Multipath TCP traffic on smartphones. In addition to analyzing the released trace, this paper proposes techniques to quantify the utilization of the subflows and presents a simple implementation independent methodology to detect handover. The analysis tools are also available for the community [2]. The results shows that Multipath TCP offers benefits for long connections, since it allows seamless handovers. However, with the default algorithms, the protocol brings some overheads, in particular with the establishment of unused subflows. This opens new areas of improvements to adapt Multipath TCP with the smartphone case, in particular the path manager.

Acknowledgements This work was partially supported by the EC within the FP7 Trilogy2 project. We would like to thank Gregory Detal and Sébastien Barré 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] Chen, Y.-C., et al. A measurement-based study of MultiPath TCP performance over wireless networks. In *IMC '13* (New York, NY, USA), ACM, pp. 455–468.

[2] De Coninck, Q., and Baerts, M. Analysis scripts. `http://github.com/multipath-tcp/mptcp-analysis-scripts`, 2015.

[3] De Coninck, Q., et al. Poster: Evaluating android applications with multipath tcp. In MOBICOM 2015, ACM, pp. 230–232.

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

[5] Eardley, P. Survey of MPTCP Implementations. Internet-Draft draft-eardley-mptcp-implementations-survey-02, IETF Secretariat, July 2013.

[6] Falaki, H., et al. A first look at traffic on smartphones. In *IMC '10* (Melbourne, Australia), ACM, pp. 281–287.

[7] Ferlin, S., Dreibholz, T., and Alay, Ö. Multi-path transport over heterogeneous wireless networks: Does it really pay off? In *Proceedings of the IEEE GLOBECOM* (Austin, Texas/U.S.A., December 2014), IEEE.

[8] Ferlin-Oliveira, S., et al. Tackling the challenge of bufferbloat in multi-path transport over heterogeneous wireless networks. In *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of* (May 2014), pp. 123–128.

[9] Ford, A., Raiciu, C., Handley, M., and Bonaventure, O. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, January 2013.

[10] Hesmans, B., and Bonaventure, O. Tracing Multipath TCP connections. *SIGCOMM Comput. Commun. Rev. 44*, 4 (Aug. 2014), 361–362.

[11] Hesmans, B., et al. A first look at real Multipath TCP traffic. In *TMA '15*, vol. 9053 of *LNCS*. Springer International Publishing, 2015, pp. 233–246.

[12] Huang, J., et al. Anatomizing application performance differences on smartphones. In *MobiSys '10*, ACM, pp. 165–178.

[13] Mellia, M., et al. Tstat: Tcp statistic and analysis tool. In *Quality of Service in Multiservice IP Networks*. Springer, 2003, pp. 145–157.

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

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

[16] Paasch, C., et al. Experimental evaluation of Multipath TCP schedulers. In *CSWS '14* (New York, NY, USA), ACM, pp. 27–32.

[17] Paasch, C., et al. On the benefits of applying experimental design to improve Multipath TCP. In *CoNEXT '13* (New York, NY, USA), ACM, pp. 393–398.

[18] Peng, Q., et al. Energy efficient Multipath TCP for mobile devices. In *MobiHoc '14* (New York, NY, USA), ACM, pp. 257–266.

[19] Raiciu, C., et al. Opportunistic mobility with Multipath TCP. In *MobiArch '11* (New York, NY, USA), ACM, pp. 7–12.

[20] Raiciu, C., et al. How hard can it be? Designing and implementing a deployable Multipath TCP. In *NSDI'12* (Berkeley, CA, USA), USENIX Assoc., pp. 29–29.

[21] sup Lim, Y., et al. Cross-layer path management in multi-path transport protocol for mobile devices. In *INFOCOM 2014* (April 2014), IEEE, pp. 1815–1823.