# Observing real Multipath TCP traffic

Viet-Hoang Tran, Quentin De Coninck, Benjamin Hesmans,
Ramin Sadre, Olivier Bonaventure
{hoang.tran,quentin.deconinck,benjamin.hesmans,
ramin.sadre,olivier.bonaventure}@uclouvain.be

*ICTEAM, Université catholique de Louvain*
*Louvain-la-Neuve, Belgium*

**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 extension, little is known about its behavior in real networks. We analyze a five-month trace collected on `multipath-tcp.org` using Multipath TCP. This first detailed study of real Multipath TCP traffic reveals several interesting points about its behavior in the wild. With packets from thousands of hosts using IPv4 and/or IPv6, we confirm that Multipath TCP correctly passes through a wide range of Internet paths. We observe long Multipath TCP connections that benefit from handovers and also connections composed of subflows having very different round-trip-times. We also analyze some inefficiencies in the current Multipath TCP implementations and quantify the importance of reinjections, i.e. the transmission of the same data over two or more subflows.

*Keywords:* networking, transport layer, measurement, performance

## 1. Introduction

The Transmission Control Protocol (TCP) is the dominant transport protocol in today's fixed and wireless networks. The design of TCP dates from the 1970s. Over the years, the protocol and its implementations have significantly evolved. Although today's TCP implementations still use the packet format proposed forty years ago, they include various optimizations [1] that are key in today's networks. These optimizations and extensions include TCP congestion control schemes [2], TCP timestamp option, large windows extensions, improved round-trip-time estimations and several forms of selective acknowledgements for better retransmission.

Multipath TCP [3, 4] is the latest significant TCP extension approved by the Internet Engineering Task Force (IETF). It enables hosts to exchange the packets that belong to one connection over different interfaces or paths. This is a major change to TCP that assumes that a connection is identified by the source

and destination addresses and the source and destination ports. Multipath TCP breaks this assumption by combining several TCP connections, called subflows, into a single Multipath TCP connection. Multipath TCP includes a special congestion control scheme that couples the congestion window among the different subflows [5].

Several use cases for Multipath TCP have already been identified in the scientific literature [6]. The initial motivation for Multipath TCP were the multihomed hosts like today's smartphones that are equipped with several network interfaces. Such hosts did not exist when TCP was designed. Users of smartphones expect that multiple available interfaces could be used simultaneously to increase the bandwidth [7] or successively to support mobility [8, 9]. With regular TCP, this form of resource pooling is not possible since each interface uses a different IP address. A second use case are the datacenters [10] where Multipath TCP improves the network utilization with Equal Cost Multipath (ECMP). A third use case are single-homed but dual-stack hosts that support IPv4 and IPv6. On such hosts, Multipath TCP can use both network layer paths simultaneously for increased performance [11].

In September 2013, Apple took the networking community by surprise by implementing and enabling Multipath TCP on all its iOS devices (including smartphones and tablets). Within a few months, hundreds of millions of devices started to use this TCP extension. On iOS, Multipath TCP does not replace regular TCP. As of this writing, the Multipath TCP API on iOS is not public and only Apple's Siri voice recognition application uses Multipath TCP to exchange small amounts of data with a few servers managed by Apple. Given the difficulty of implementing a complex protocol like Multipath TCP [12], it can be expected that other applications will leverage Multipath TCP in the coming years. It is now included in the latest MacOS releases and there are plans to upstream the Linux implementation in the official Linux kernel [13].

In this paper, we provide the first detailed analysis of the operation of Multipath TCP in real networks by analyzing long packet traces collected on a Multipath TCP server often used by researchers and implementers. More precisely, this paper provides the following contributions that improve our understanding of the dynamics of this new protocol.

We first collected a five-month-long packet trace that contains 190,451 Multipath TCP connections originated from more than 7,000 different hosts across the Internet. This is the largest publicly available dataset about this new protocol[1]. To analyze this long trace, we improve the open-source `mptcptrace` software [14] and develop custom scripts. Our analysis reveals various important information about the current usage of Multipath TCP. Multipath TCP works well over many Internet paths and we observe very few fallbacks caused by middleboxes. Multipath TCP connections can last thousands of seconds or more and we observe handovers on such connections. From the performance viewpoint, our analysis reveals that Multipath TCP connections are often com-

[1]This dataset is available at `http://www.multipath-tcp.org/data/COMCOM16`

posed of subflows having very different round-trip-times. We also analyze some inefficiencies of the current Multipath TCP implementations and study in more details the reinjections, i.e. the transmissions of the same data over several paths, and how they affect the performance.

This paper is organized as follows. We first provide a brief overview of Multipath TCP and discuss related work in Sec. 2. We describe the dataset that we have collected in Sec. 3 and our extensions to `mptcptrace` that enabled us to process them efficiently in Sec. 4. In Sec. 5, we analyze how the Multipath TCP connections use their subflows. We discuss in Sec. 6 main lessons that we learned from this first detailed analysis of Multipath TCP packet trace.

## 2. Multipath TCP

This section provides a brief introduction to the Multipath TCP protocol [3] and the key characteristics of its implementation in the Linux kernel. A more complete introduction to Multipath TCP may be found in [15].

A Multipath TCP connection is a logical association between a client and a server. To understand the operation of Multipath TCP, let us consider the simple but common case of a smartphone using WiFi and cellular and a single-homed server. To request the utilization of Multipath TCP, the smartphone adds the `MP_CAPABLE` option in the TCP's `SYN` segment sent over its cellular interface. This option contains several flags and a random key [3]. If the server supports Multipath TCP, it also replies with the `MP_CAPABLE` option in the `SYN+ACK` segment. This option also contains a random key chosen by the server. To cope with middleboxes that could remove the `MP_CAPABLE` option from the `SYN+ACK` segment, an `MP_CAPABLE` option containing the two keys is also placed in the third acknowledgement returned by the client. According to the Multipath TCP terminology, this initial TCP connection is called a subflow [3]. Thanks to this subflow, the smartphone and the server can exchange data over the cellular interface. The smartphone or the server must then send some data over this subflow to verify that the Multipath TCP options placed in data segments are not modified by middleboxes. Once this first data has been correctly acknowledged, the subflow is considered to be valid and other subflows can be created to use different paths.

If the smartphone wants to send data also 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 and allows the server to attach the new subflow to the existing Multipath TCP connection. The server replies with an `MP_JOIN` and the subflow is established. The smartphone can also inform the server of the other addresses that it uses through the `ADD_ADDR` option.

At this stage, the Multipath TCP connection is composed of two subflows. This number of subflows is not fixed. A subflow can stop at any time and other subflows can be established during the lifetime of the Multipath TCP connection. Data can be sent over any of the available subflows. To enable the receiver

to reorder the received data, Multipath TCP relies on two levels of sequence numbers: the regular TCP sequence number and the Data Sequence Number (DSN) which corresponds to the application-level bytestream. When data is sent over a subflow, its DSN is mapped to the regular sequence numbers within the `Data Sequence Signal (DSS)` option. The `DSS` option also contains DSN acknowledgements. Thanks to this option, the same data can be transmitted over more than one subflow. This is called a reinjection [4]. A reinjection can occur if data transmitted over one subflow has not been acknowledged quickly enough. For example, a smartphone could send some data over its WiFi interface while moving out of reach of the access point. This data will have to be retransmitted later over the cellular interface to reach the server.

Several independent implementations of Multipath TCP have been written [12]. Apple's implementation is the most widely deployed, but it is only used by one application. The Multipath TCP implementation in the Linux kernel [16] is the most complete one. It includes several algorithms that are not part of the protocol specification [3] but influence its performance and the transmission of packets. There are three main modular components that affect the performance of the Linux implementation: *(i)* the path manager, *(ii)* the congestion control scheme and *(iii)* the scheduler.

The *path manager* defines the strategy that is used by the Multipath TCP stack to create subflows. Two path managers are included in the Linux implementation: `full-mesh` and `ndiffports`. They decide when and how subflows are created. As of this writing, subflows are only created by the clients. The server does not attempt to create subflows because the clients are often behind NATs or firewalls that would block these subflows [12]. The `full-mesh` path manager creates a subflow from each address owned by the client to each address advertised by the server. These subflows are created at the beginning of the connection, as soon as the initial subflow has been validated. If the host learns a new address, e.g., a smartphone attaches to a new WiFi access point, the `full-mesh` path manager automatically creates a new subflow over this interface. The `ndiffports` path manager was designed for single-homed hosts in datacenters [10]. With this path manager, a Multipath TCP connection is composed of `n` subflows that use different source ports. The `full-mesh` path manager is the default path manager in the Linux implementation, but some researchers use the `ndiffports` path manager.

The second important algorithm is the *congestion controller*. Several congestion control algorithms have been proposed for Multipath TCP. These congestion control schemes couple the congestion windows of the different subflows to preserve fairness with the regular TCP congestion control scheme. Four of them are included in the Linux kernel implementation: LIA [5], OLIA [17], BALIA [18] and wVegas [19]. LIA is the default congestion control scheme, but users can opt for other congestion control schemes. Some users have reported that they used non-coupled congestion control schemes like CUBIC which is the default congestion control scheme in Linux.

The third important algorithm of the Linux kernel is the *packet scheduler* [20]. This algorithm is used every time a data segment needs to be sent. It

selects, among the active subflows that have an open congestion window, the subflow that will be used to send the data. The default scheduler tries to send data over the subflow having the lowest round-trip-time.

### *2.1. Related work*

Various researchers have analyzed the performance of Multipath TCP through measurements. Raiciu et al. [8] discuss how Multipath TCP can be used to support mobile devices and provide early measurement results. Pluntke et al. [21] analyze whether Multipath TCP could reduce energy consumption by using several interfaces simultaneously. Paasch et al. [9] propose three modes for the operation of Multipath TCP in wireless networks and describe measurements of handovers. Chen et al. [22] analyze the performance of Multipath TCP in WiFi/cellular networks by using bulk transfer applications running on laptops. Ferlin et al. [23] 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. Livadariu et al. explore the performance of Multipath TCP on dual-stack hosts [11] and show that performance over IPv6 and IPv4 paths differ. Ferlin et al. [24] propose a probing technique to detect low performing paths and evaluate it in wireless networks. Lim et al. [25] propose a technique to reduce the energy consumption of smartphones that are using Multipath TCP and evaluate it experimentally. Williams et al. analyze in [26] the performance of Multipath TCP on moving vehicles. Deng et al. [27] compare the performance of single-path TCP over WiFi and LTE networks with the performance of Multipath TCP on multi-homed devices by using active measurements and replaying HTTP traffic observed on mobile applications. Their measurements show that Multipath TCP provides benefits for long flows but not for short ones. For short flows, they show that the selection of interface for the initial subflow is important from a performance viewpoint. Hesmans et al. analyze in [28] a one-week-long subset of the trace that we use in this paper. In this paper, we use a much larger dataset (5 months), apply new analyses based on concept of data burst and subflow switch, as well as look at the amount of data acknowledged on TCP and MPTCP level. This work and its earlier version [28] are the first ones that collect and analyze a large real-world Multipath TCP dataset, revealing some new aspects of Multipath TCP traffic.

## 3. Dataset

Although Multipath TCP is already used by hundreds of millions of Apple smartphones to support the Siri voice recognition application, our analysis instead focus on the Multipath TCP implementation in the Linux kernel [16]. This implementation is distributed from `multipath-tcp.org` and thousands of users have downloaded and installed it on their computers. We use `tcpdump` on the server side to collect all the packets sent and received by `multipath-tcp.org`. At the time of writing, one limitation is that we have a single server only with one interface, but dual IPv4/IPv6 stack is available to clients. In addition to

providing source code and binaries for the Multipath TCP patch, this host also supports other web servers, an `FTP` server and uses an `Iperf` daemon to enable researchers and users to perform various tests. We extract from the trace the packets that correspond to Multipath TCP connections. They are identified by looking at the utilization of the Multipath TCP options during the three-way handshake. In the remaining of this section, we provide statistics about our trace.

We first analyzed the main characteristics of the Multipath TCP connections in the dataset, which has been collected during 5 months from November $25^{\text{th}}$, 2014 to April $29^{\text{th}}$, 2015. We observed 10300 IP addresses have been used by our clients, including 7770 IPv4 addresses. While users could be biased toward MPTCP developers and early adopters, we still observed a very diverse users set based on their Autonomous Systems (AS). According to the IP-to-ASNumber database of Team Cymru [29], these client IP addresses correspond to 588 unique ASes.

We identified 190,451 Multipath TCP connections, which used several destination ports. Most (86.17 %) of the connections used port 80. The remaining connections were on ports 21 (FTP, 5.33 %), 5001 (Iperf, 1.68 %) and other high port numbers linked to passive-mode FTP connections and a private HTTP proxy (port 3128) used for some tests.

| Port | MPTCP connections | % Connections | % Traffic volume |
|---|---|---|---|
| 20 | 82 | 0.04 | 1.04 |
| 21 | 10158 | 5.33 | 0.002 |
| 80 | 164123 | 86.17 | 26.80 |
| 3128 | 1215 | 0.63 | 0.50 |
| 5001 | 3211 | 1.68 | 22.14 |
| Others | 11662 | 6.12 | 49.52 |

Table 1: Popular TCP server ports used in the dataset

The observed connections are diverse in terms of both duration and size. Figure 1 plots the durations of the Multipath TCP connections. We observe that most of the Multipath TCP connections last in the range of 1 to 100 seconds. There are only 1.51% of the connections lasting more than 100 seconds.

In terms of connection size, as shown in Fig. 2, we could identify some major clusters of connections: a group of exactly 2 Bytes (account for 12% of all the connections), large groups around 1 KByte (account for 15%) and 10 KBytes (account for 65%), smaller groups around 100 KBytes and 1 MBytes. The connections containing exactly two bytes are empty connections with one-byte space for DATA_FIN in both directions.

This is expected since today's web browsers usually pre-open connections on client side to reduce the latency perceived by users. Web traffic and the FTP control channel are responsible for the majority of the small connections which carry around 400 Bytes and 20 KBytes (account for around 80% of all the connections). The remaining large connections are likely FTP data traffic

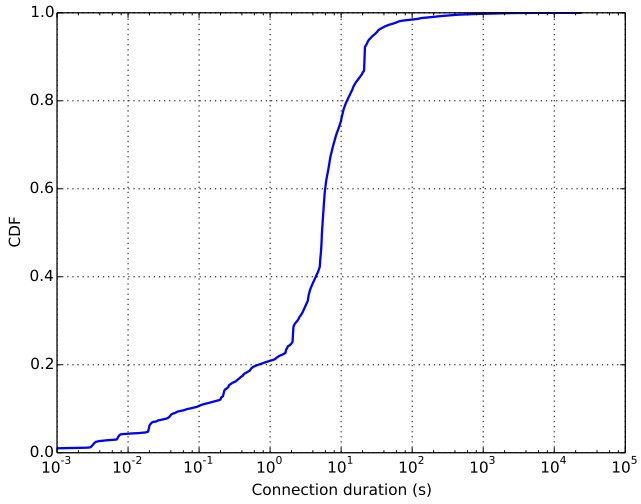


Figure 1: Duration of the Multipath TCP connections

and `iperf` test transfers.

## 4. Tracing Multipath TCP

While many software tools are designed to extract information about TCP connections from a packet trace, such as `tcptrace` [30] or `tstat` [31], these tools do not understand the specificities of Multipath TCP. To analyze our dataset, we developed and extended the `mptcptrace` software [14]. The first version of `mptcptrace` [14] was designed to process short packet traces.

We have significantly improved `mptcptrace` implementation to process the long packet traces that we analyze in this paper. Memory management has been improved to consume less memory. The tool now recognizes three ways to finish a connection. First, the `DATA_FIN` has been exchanged from both end hosts. Second, one of the two end host has sent the `FAST_CLOSE mptcp` option. Third, no activity of the connection has been detected within the last $s$ seconds, which can be set as a parameter. Another option has been added to limit the size of the queue that keeps in memory the mapping between Multipath TCP (or data-level) sequence numbers and the subflow on which it has been sent initially. This queue is used to detect reinjections but can become very large if a connection is long. Moreover, if the queue is long enough, it is unlikely that `mptcptrace` will miss any reinjection. The internal structure used to store the Multipath TCP connections and subflows has also be changed to hash table to process long traces faster. With real traffic traces, new corners cases have been detected and are now correctly handled by `mptcptrace`. We have also added new statistics in addition to the ones described in [14]. New per-file statistics,

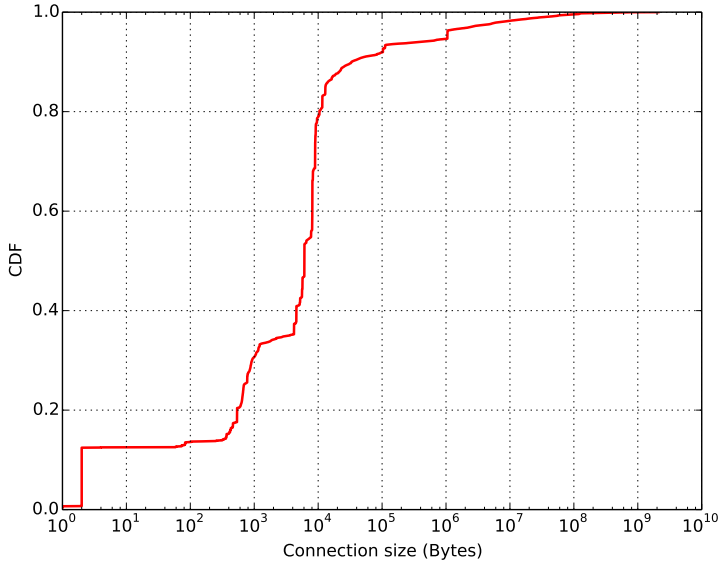


Figure 2: Traffic volume (in Bytes) carried by each connection

rather than per-connection statistics have also been added to ease the analysis. Anomalies detected during the analysis are also better reported by `mptcptrace`.

`mptcptrace` understands the semantics of the Multipath TCP protocol and can extract the keys that are exchanged during the establishment of the initial subflow from the `MP_CAPABLE` option. Thanks to these keys, `mptcptrace` computes the tokens that identify the Multipath TCP connection on both the client and the server. With these keys, `mptcptrace` can link the different subflows that compose each Multipath TCP connection.

Once the subflows that are associated to one Multipath TCP connection have been grouped together, `mptcptrace` produces detailed plots that provide a graphical representation of the Multipath TCP connection and its subflows. In contrast with `tstat`, `mptcptrace` exploits mainly the contents of the `DSS` option instead of the `SEQ`, `ACK` and other fields of the TCP header. For example, `mptcptrace` can plot the instantaneous throughput at the Multipath TCP level. It can also plot the evolution of the receive window and the amount of data that are in flight. `mptcptrace` also computes global statistics like `tstat` but at Multipath TCP level. For example, `mptcptrace` computes the duration of each Multipath TCP connection. This is defined to be the delay between the first `SYN` that carries the `MP_CAPABLE` option and the last packet sent over this connection, possibly on another subflow. `mptcptrace` also computes the fractions of the bytes that are sent over each subflow and also detects reinjections (i.e. the transmission of the same data over two or more subflows).

`mptcptrace` is publicly available[2]. In addition, we also rely on a set of Python scripts that post-process the outputs of `tstat`, `tcptrace` and `mptcptrace`. These scripts are also publicly available [3] [4].

## 5. Analysis

This section provides the most important results about both behavior and performance of the protocol, including the interferences caused by middleboxes (Sec. 5.1), how fast the hosts establish the additional subflows (Sec. 5.2), the round-trip-time pattern of Multipath TCP paths (Sec. 5.3), how data is acknowledged at TCP and Multipath TCP level (Sec. 5.4), how data is spread over different subflows (Sec. 5.5), and the overhead introduced by Multipath TCP compared to regular TCP (Sec. 5.6 and 5.7).

### *5.1. Middlebox interference*

Multipath TCP was designed to cope with a wide range of middlebox interferences [3, 32]. If a middlebox interferes too much with the operation of Multipath TCP, the connection should fallback to regular TCP. More concretely, if a middlebox modifies the payload, the mapping between Multipath TCP-level and subflow-level sequence numbers would likely be invalid and corrupt the transferred data. Multipath TCP uses its own checksum to detect any modification of payload, ensuring that the mappings are always correct. If the checksum fails, the receiver will inform the remote peer, close all but one subflow and switch to regular TCP.

Among 190,451 Multipath TCP connections, we observe only 125 of them falling back to regular TCP, which happened with 28 distinct client IP addresses. These include 91 HTTP connections and 34 FTP connections. The FTP interference is expected and due to Application Level Gateways running on NAT boxes which try to "correct" the port number in FTP command to match with forwarded port number. The HTTP interference appeared only on the direction from server to client and could have been caused by transparent proxies deployed in cellular and enterprise networks [33]. The rate of fallbacks that we observe is lower than that of earlier work [34]. Note that since we collect data on the server, we cannot detect the removal of the `MP_CAPABLE` option by a middlebox in the SYN.

### *5.2. Establishment of the subflows*

With Multipath TCP, a host can send data over different subflows. The number of subflows that a host creates depends on various factors including the number of interfaces that it has and its path-manager as explained in section 2. As mentioned earlier, the server has a single interface but with dual IPv4/IPv6

[2] See `https://bitbucket.org/bhesmans/mptcptrace`
[3] See `https://github.com/multipath-tcp/mptcp-analysis-scripts`
[4] See `https://bitbucket.org/hoang-tranviet/mpmine`

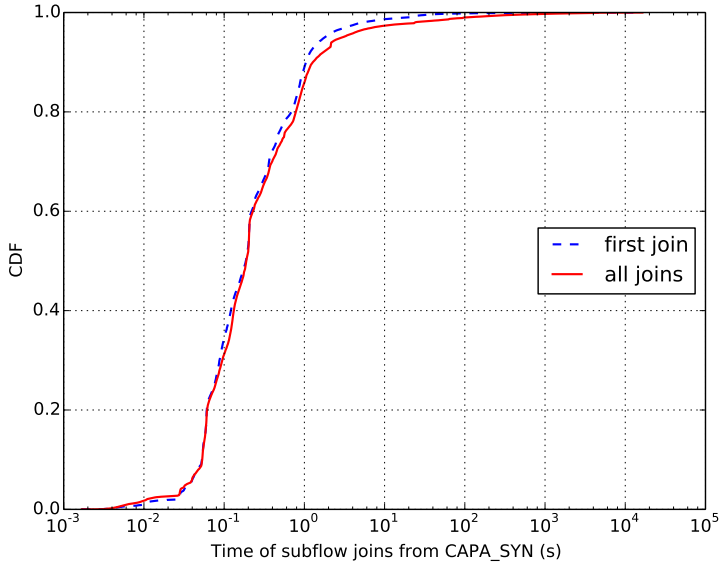


Figure 3: Delay from the creation of MPTCP connection to the establishment of a subflow

stack. Table 2 reports the number of (not necessarily concurrent) subflows per connection. We see that 61.56% of the connections contain only one subflow. Among the connections that have at least two subflows, 49% of them originated from different IP addresses and were likely created by the `full-mesh` path manager[5]. The other 51% of the connections originated from a single address, which would correspond to the `ndiffports` path manager.

| Number of subflows | 1 | 2 | more than 2 |
|---|---|---|---|
| | 61.56% | 23.3% | 15.14% |

Table 2: Number of subflows per Multipath TCP connection

The maximum number of subflows per connection is as large as 68. Notice that this is the number of all subflows during the lifetime, not necessarily active at the same time, so this is not inconsistent with the fact that the Multipath TCP implementation in Linux kernel supports only 32 concurrent subflows. This unexpected large number was probably due to researchers who performed tests with the monitored server.

Another interesting point is the delay between the establishment of the con-

[5]We expect that most of the packets in the dataset were generated by Linux clients. Although other implementations exist, they are not usable for regular traffic (e.g., Apple or Citrix) or not yet mature enough (e.g., FreeBSD). We cannot however infer the version of the Multipath TCP implementation used by a client from the packet trace.

nection (i.e. the first subflow) and the establishment of the other subflows. This metric can be used for two purposes: showing how quickly Multipath TCP utilizes additional subflows, and suggesting handovers. The process to establish the subflows is called `join`. With the Linux implementation, path managers try to set up subflows shortly after the creation of the Multipath TCP connection and as soon as a new IP address is learned by the client. To quantify this effect, we plot in Fig. 3 the CDF of the delays between the creation of each Multipath TCP connection and all the subflows that are linked to it.

We observe that 86% of the additional subflows are established during the first second of the Multipath TCP connections. Comparing the two lines, we can see the difference with longer duration. After one minute, most of the first joins had been done, but there is still 1.79% of all subflows joined after one minute and 0.22% of that after one hour. These joins happening longly after the connection establishment suggest that those connections were likely experiencing a handover.

*5.3. Subflows round-trip-time*

A subflow is established through a handshaking process, as explained in section 2. Thanks to this exchange, the communicating hosts measure the initial value of the round-trip-time (RTT) for the subflow. For the Linux implementation, the round-trip-time measurement is an important performance metric because the default packet scheduler prefers the subflow having the lowest RTT.

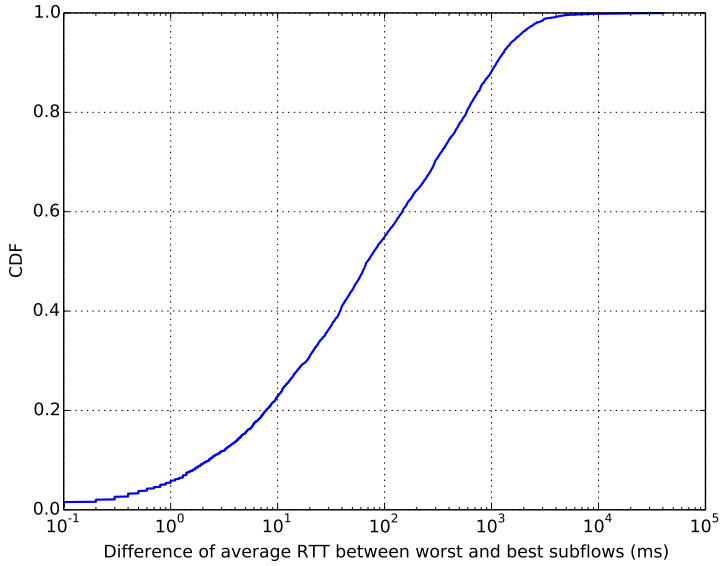


Figure 4: Difference of average RTT between worst and best subflows from server to clients of the same Multipath TCP connection (at least 100 KB)

We evaluate the RTTs heterogeneity of Multipath TCP connections. For this, we first extract from the trace the connections that carry at least 100 KBytes.

For each connection, we use `tcptrace` to compute the average round-trip-time of all the subflows that it contains, and then extract the minimum and the maximum of these values. Figure 4 plots the CDF of the average RTT difference between the fastest and the slowest subflows over all connections. Only 22.6% of the connections have subflows whose RTT difference is within 10 ms.

We observe as many as 12.5% of the considered connections whose RTTs differ by more than one second, which might be the result of the bufferbloat issue. Exceptionally, for some connections, this RTT difference is as high as 22 seconds.

Since the default packet scheduler prefers low RTT subflows, we focus on the two best subflows of each connection. Figure 5 shows the boxplot of the average RTT of the second-best subflow in relation with that of the best subflow of the same Multipath TCP connection. Except the first box depicting the connections having best subflow's RTT smaller than 40 ms, the figure shows an observable difference between the RTT of second-best subflow and RTT of best subflow. While the median value of second-best subflow's RTT is not far away from the best one, the tail values may be pretty high.

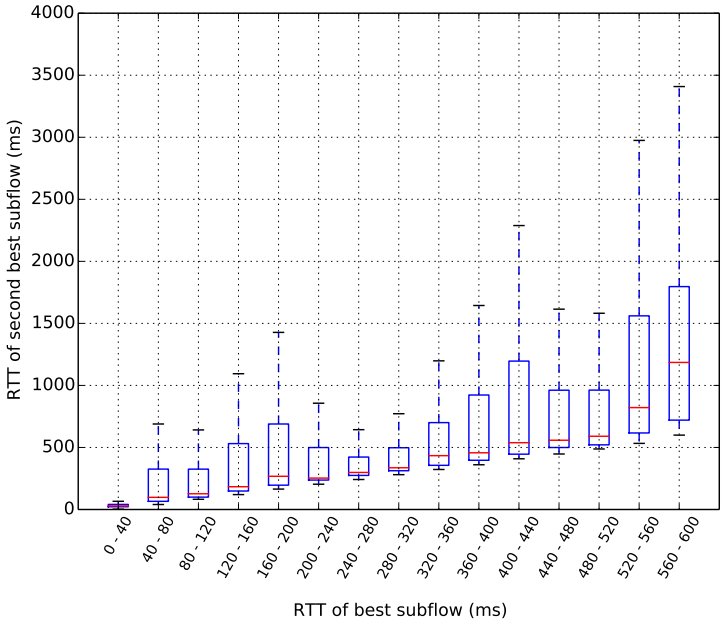


Figure 5: Average RTT of the second-best subflow vs. average RTT of the best subflow of the same Multipath TCP connection (at least 100 KB)

### *5.4. Multipath TCP acknowledgements*

As explained in Sec. 2, Multipath TCP uses two levels of acknowledgement: the regular TCP acknowledgements at the subflow level and the cumulative

Multipath TCP acknowledgements at the connection level. By comparing the amount of data acknowledged at these two levels, we can infer about the degree of out-of-order issue as well as the memory footprint pressure in general.

Over the years, TCP has been tuned to pack acknowledgements and minimize their overhead [1]. When there are no loss and no reordering, a TCP receiver should acknowledge every second packet that it receives. We clearly observe this in Fig. 6 that shows the number of bytes that are acknowledged by the non-duplicate TCP acknowledgements on Multipath TCP connections with at least two subflows. This plot is a weighted CDF where the contribution of each acknowledgement is weighted by the number of bytes that it acknowledges. A big step at around 2,856 Bytes shows the effect of `delayed acknowledgement` feature of TCP hosts which acknowledge every two packets. We observe the TCP acks that acknowledge one (max. 1428 bytes) or two packet (max. 2856 bytes) account for as many as 72% of the bytes acknowledgement.

We perform a similar analysis with the Multipath TCP Data acknowledgements. The solid line in Fig. 6 shows the weighted cumulative distribution of the number of bytes acked per data acknowledgement. Compared with the regular TCP acknowledgements, the Multipath TCP acknowledgements cover more bytes. 50.5% of the Data-acks acknowledge two segments of data or less, while for regular TCP this number is 72%. Furthermore, 9% of the total traffic are acknowledged as data chunks larger than 100 KBytes at MPTCP level, compared to 4% at TCP level.

The difference between the regular TCP acknowledgements and the Data acknowledgements is caused by the reordering that occurs when data is sent over several subflows. Since the Data acks are cumulative they can only be updated once all the previous data have been received from all subflows. If a subflow with a long round-trip-time is used, it will cause reordering and data will remain in the reordering queue on the receiver for a long period. Reordering is less frequent with regular TCP.

### *5.5. Data distribution*

Another important point is the distribution of the data among the available subflows. Observing Multipath TCP connections that contain two active subflows or more, we see that data can be distributed in very different ways among the subflows. At one extreme, almost all bytes are sent over a single subflow. At the other extreme, packets can be distributed in a round-robin fashion.

#### *5.5.1. Bias of data distribution towards initial subflow*

A basic question about data distribution is whether the scheduler has bias towards the initial subflow, like the capture effect mentioned in [35]. For this purpose, we extract from the the trace all the connections having exactly 2 subflows and compute the percentage of data that has been sent on initial subflow. Based on the connection size distribution (Fig. 2), we examine on three different groups of connections: (1) smaller than 50 KBytes, (2) between 50 KBytes and 5 MBytes, and (3) larger than 5 MBytes.

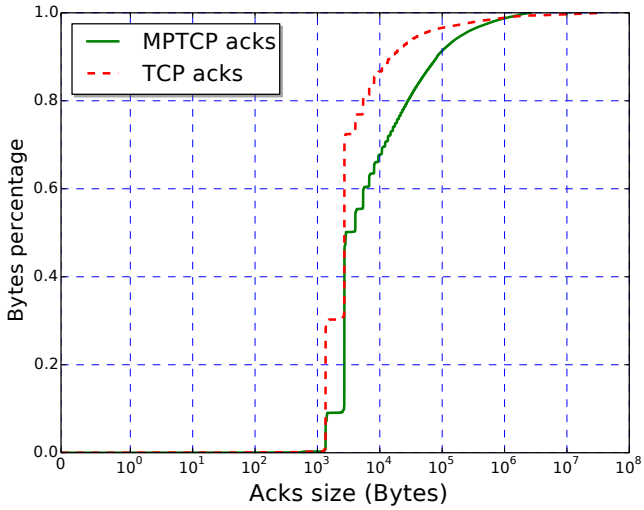


Figure 6: Weighted CDF of the size of acknowledged data chunks for traffic from server to clients, from connections having at least two subflows

As shown in Fig. 7, it turns out that the data distribution of two-subflow connections suggests some interesting points. Looking at the right part of the graph, there are as many as 29.6% of sub-50KB MPTCP connections which send all traffic over only the initial subflow. This proportion is smaller for groups of larger connections (13.8% in group 2 and 6.7% in group 3). Moreover, there are around 65.5% of the MPTCP connections larger than 5 MB sending more than half of their traffic over the initial subflow. This bias is even more significant (76.6%) if we consider the MPTCP connections between 50 KBytes and 5 MBytes. Interestingly, only a minority (47.5%) of the connections smaller than 50 KBytes send more traffic over their initial subflow. After checking these connections in details, we observe that the initial subflow has the RTT worse than that of second subflow in most cases. Usually, the initial subflow carries payload while the second subflow is establishing. The RTT of the initial subflow when carrying payload is likely higher than that of second subflow when handshaking without payload. Since the Linux's scheduler prefers subflow with smallest RTT, traffic would be sent through the second subflow. When the connection is very small, the scheduler would not switch back to initial subflow before finishing the transfer.

*5.5.2. Subflow switching frequency*

To further evaluate how data is spread among the established subflows, we propose to use the notion of *subflow switch* and *burst*. A burst is a series of consecutive bytes that are sent over a given subflow. The sizes of the bursts sent by the monitored server are shown in Fig. 8. We observe that most of bursts (

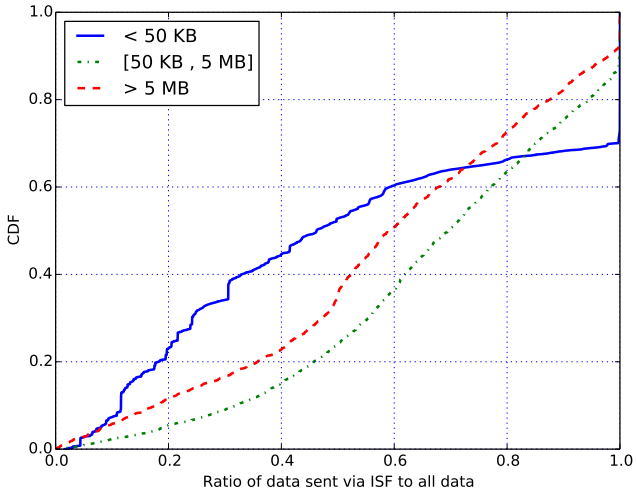


Figure 7: Ratio of data sent via initial subflow to all data sent via both subflows.

about 90%) are smaller than 10 KBytes.

For a given Multipath TCP connection, a *subflow switch* is when the sender changes the subflow to send out data. The number of *subflow switches* per MByte data is called *subflow switch frequency*. By measuring the *subflow switch frequency*, we can compare the behavior of connections which have different rates and durations. A high *subflow switch frequency* might suggest a connection that sends the packets in a round-robin fashion among the active subflows. A low frequency indicates a connection that sends long bursts of data over each subflow.

Figure 9 provides the CDF of the subflow switch frequency computed over all Multipath TCP connections that carry at least one MBytes of data and contain at least two subflows. Each point in the curves corresponds to one Multipath TCP connection and indicates the number of subflow switches normalized on its size in bytes. Nearly 15% of the connections do not switch, which means that they use only one subflow. On the other hand, we observe that 13% of the connections in this trace have a subflow switching frequency larger than 200 times/MByte. The high subflow switching frequencies are likely the result of the RTT similarities between different subflows. As mentioned in Sec. 5.2, half of the connections in this trace used the `ndiffports` path manager. The subflows on these connections usually use the same path and thus have almost the same round-trip-times. Since the default scheduler prefers the lowest-RTT subflow, the sender switches very frequently between the different subflows.

Given that the typical maximum segment size (MSS) is between 1400 and 1500 Bytes, it is unexpected that the switch frequency is larger than 700 times/MByte. By looking at the trace in details, we figured out two main reasons for this: (1) some connections have MTU smaller than typical value, around 1000

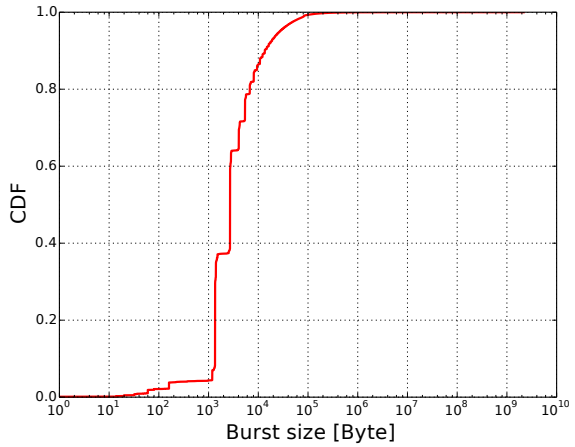


Figure 8: Size of bursts of all multi-subflow connections from server

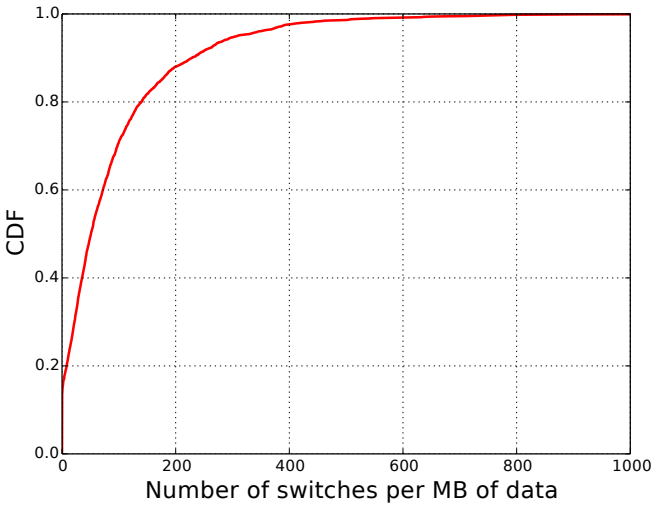


Figure 9: Subflow switch frequency, from multi-subflow connections

Bytes or even 500 Bytes, and (2) in some case the scheduler sends small packets frequently even if the standard MSS is used.

### *5.6. Unused and underused subflows*

A potential imperfection of Multipath TCP is that subflows can be established without being used to transport data. Creating subflows that are not used consumes resources and energy [36, 37] on smartphones since the interface over which these subflows are established is kept active.

There are three reasons explaining those unused subflows. Firstly, for the small connections, all data exchange can be finished before new subflows are fully established. Secondly, the subflow can be established as a backup subflow. Thirdly, the difference in round-trip-times between the two subflows can be so large that the subflow with the highest RTT is never selected by the packet scheduler. Though, we noticed that on the unused additional subflows, 43.15% show a better RTT with the newly-established subflow. In fact, 75% of the connections containing such subflows carry less than 1000 bytes, and 95% less than 20 KBytes.

By immediately creating subflows, the `full-mesh` path manager is responsible for some Multipath TCP inefficiencies. An improved path manager [36, 37] would prevent some of these inefficiencies.

### *5.7. Reinjection overhead*

A *reinjection* [4], is the retransmission of the same data over two or more subflows. Reinjections can occur for several reasons: *(i)* handover, *(ii)* excessive losses over one subflow or *(iii)* limited windows due to the Opportunistic Retransmission and Penalization (ORP) heuristic proposed in [4] and enhanced in [38].

Multipath TCP reinjections are closely linked with regular TCP retransmissions. Since reinjections, by definition, can only occur on connections that contain at least two subflows, we consider only the connections composed of at least two subflows and carrying at least one byte. Figure 10 shows the CDF of the reinjections and retransmissions. The number of retransmitted and reinjected bytes are normalized with the number of bytes exchanged over the connection. Since the same data can be sent over several subflows, this fraction can be larger than 1.0 for connections that carry a small amount of data that is retransmitted or reinjected.

We observe that reinjections occur but are less frequent than regular TCP retransmissions. While 25% of the multi-subflow connections do experience retransmissions, only 4.1% of them experience reinjections. 923 MBytes are reinjections and 1323 MBytes are retransmissions, out of a total of 113 GBytes transmitted.

## 6. Conclusion

TCP is probably one of the most studied networking protocols. Hundreds of papers have analyzed its performance and behavior in a wide range of conditions.

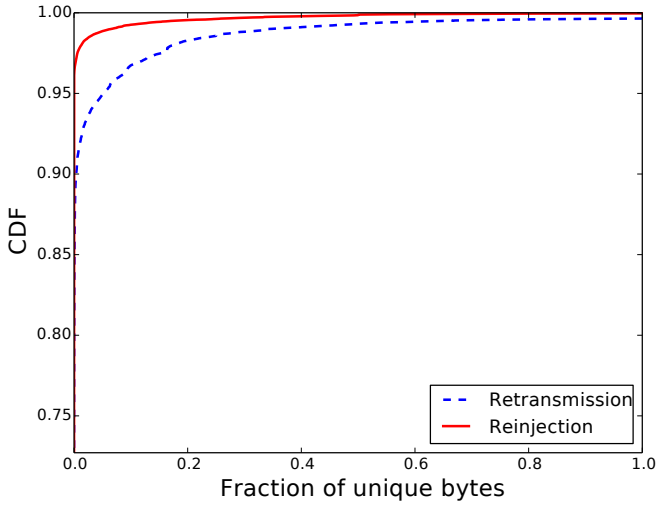


Figure 10: CDF of the fraction of bytes that are reinjected/retransmitted on the Multipath TCP connections composed of at least two subflows

All this work has led to various improvements that are used in widely deployed implementations [1]. Thanks to the utilization of multiple flows, Multipath TCP brings another dimension to the problem of reliably transmitting data between two hosts. With regular TCP, a host adapts its transmission rate to avoid congestion thanks to its congestion control scheme. With Multipath TCP, when several concurrent subflows are active, the host can spread the data over different paths. Furthermore, the number of subflows that are associated to a connection can vary over time.

Our detailed analysis of a five-month-long packet trace has led to several interesting observations. First, Multipath TCP works correctly over a wide range of Internet paths. This demonstrates the deployability of the protocol extension in the global Internet. Second, the trace reveals two different usages of Multipath TCP. About 86% of the connections are composed of subflows that are created almost immediately. These connections expect improved performance from the utilization of several subflows. In the remaining 14% of the connections, one or more subflows are created more than one second after the initial subflow. This is likely corresponding to the second use case for Multipath TCP that enables applications to continue to use existing connections after a handover. Looking at the round-trip-times of the subflows that Multipath TCP uses, our analysis reveals that there can be a huge difference between the fastest and the slowest subflow inside each Multipath TCP connection. This large delay difference must be taken into account by Multipath TCP implementers who propose solutions to improve the performance of the protocol.

We have also studied in details how retransmissions and reinjections occur.

Our analysis reveals that reinjections, i.e. retransmitting data over more than one subflow, are common but less frequent than regular retransmissions.

**Acknowledgements**

This work was supported by the Marie Curie ITN METRICS project (grant agreement No. 607728) and the FP7 TRILOGY 2 project funded by the European Commission and by the BESTCOM IAP.

**References**

[1] E. Blanton, M. Duke, R. Braden, W. Eddy, A. Zimmermann, A roadmap for Transmission Control Protocol (TCP) specification documents, RFC7414.

[2] A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, Host-to-host congestion control for TCP, Communications Surveys & Tutorials, IEEE 12 (3) (2010) 304–342. doi:10.1109/SURV.2010.042710.00114.

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

[4] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, 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, NSDI'12, USENIX Association, Berkeley, CA, USA, 2012, pp. 29–29.

[5] D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley, Design, implementation and evaluation of congestion control for Multipath TCP, in: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, USENIX Association, Berkeley, CA, USA, 2011, pp. 99–112.

[6] O. Bonaventure, C. Paasch, G. Detal, Experience with Multipath TCP, Internet-Draft draft-ietf-mptcp-experience-01, IETF Secretariat, i-D Exists (Mar. 2015) [cited 10-Jan-2015].

[7] D. Wischik, M. Handley, M. B. Braun, The resource pooling principle, SIGCOMM Comput. Commun. Rev. 38 (5) (2008) 47–52. doi:10.1145/1452335.1452342.

[8] C. Raiciu, D. Niculescu, M. Bagnulo, M. J. Handley, Opportunistic mobility with Multipath TCP, in: Proceedings of the Sixth International Workshop on MobiArch, MobiArch '11, ACM, New York, NY, USA, 2011, pp. 7–12. doi:10.1145/1999916.1999919.

[9] C. Paasch, G. Detal, F. Duchene, C. Raiciu, O. Bonaventure, Exploring Mobile/WiFi Handover with Multipath TCP, in: ACM SIGCOMM CellNet workshop, 2012, pp. 31–36. doi:10.1145/2342468.2342476.

[10] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving Datacenter Performance and Robustness with Multipath TCP, in: ACM SIGCOMM 2011, 2011. doi:10.1145/2018436.2018467.

[11] I. Livadariu, S. Ferlin, O. Alay, T. Dreibholz, A. Dhamdhere, A. M. Elmokashfi, Leveraging the IPv4/IPv6 Identity Duality by using Multi-Path Transport, in: Proceedings of the 18th IEEE Global Internet Symposium (GI), Hong Kong/People's Republic of China, 2015.

[12] P. Eardley, Survey of MPTCP Implementations, Internet-Draft draft-eardley-mptcp-implementations-survey-02, IETF Secretariat (Jul. 2013) [cited 10-Jan-2015].

[13] D.-C. Gucea, O. Purdila, Shaping the Linux kernel MPTCP implementation towards upstream acceptance, presented at Netdev01 (March 2015).

[14] B. Hesmans, O. Bonaventure, Tracing Multipath TCP connections, SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 361–362. doi:10.1145/2740070.2631453.

[15] C. Paasch, O. Bonaventure, Multipath TCP, Commun. ACM 57 (4) (2014) 51–57. doi:10.1145/2578901.

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

[17] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, J.-Y. Le Boudec, MPTCP is not pareto-optimal: Performance issues and a possible solution, in: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, ACM, New York, NY, USA, 2012, pp. 1–12. doi:10.1145/2413176.2413178.

[18] Q. Peng, A. Walid, J. Hwang, S. Low, Multipath TCP: Analysis, design, and implementation, Networking, IEEE/ACM Transactions on PP (99) (2015) 1–1. doi:10.1109/TNET.2014.2379698.

[19] Y. Cao, M. Xu, X. Fu, Delay-based congestion control for Multipath TCP, in: Network Protocols (ICNP), 2012 20th IEEE International Conference on, 2012, pp. 1–10. doi:10.1109/ICNP.2012.6459978.

[20] C. Paasch, S. Ferlin, O. Alay, O. Bonaventure, Experimental evaluation of Multipath TCP schedulers, in: Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop, CSWS '14, ACM, New York, NY, USA, 2014, pp. 27–32. doi:10.1145/2630088.2631977.

[21] C. Pluntke, L. Eggert, N. Kiukkonen, Saving mobile device energy with Multipath TCP, in: Proceedings of the Sixth International Workshop on MobiArch, MobiArch '11, ACM, New York, NY, USA, 2011, pp. 1–6. doi:10.1145/1999916.1999918.

[22] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, D. Towsley, A measurement-based study of MultiPath TCP performance over wireless networks, in: Proceedings of the 2013 Conference on Internet Measurement Conference, IMC '13, ACM, New York, NY, USA, 2013, pp. 455–468. doi:10.1145/2504730.2504751.

[23] S. Ferlin-Oliveira, T. Dreibholz, 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, 2014, pp. 123–128. doi:10.1109/IWQoS.2014.6914310.

[24] S. Ferlin, T. Dreibholz, Ö. Alay, Multi-Path Transport Over Heterogeneous Wireless Networks: Does It Really Pay Off?, in: Proceedings of the IEEE GLOBECOM, IEEE, Austin, Texas/U.S.A., 2014. doi:10.1109/GLOCOM.2014.7037567.

[25] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, 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. doi:10.1145/2627585.2627596.

[26] N. Williams, P. Abeysekera, N. Dyer, H. Vu, G. Armitage, Multipath TCP in Vehicular to Infrastructure Communications, Tech. Rep. 140828A, CAIA, Swinburne University of Technology (August 2014).

[27] S. Deng, R. Netravali, A. Sivaraman, H. Balakrishnan, WiFi, LTE, or both?: Measuring multi-homed wireless internet performance, in: Proceedings of the 2014 Conference on Internet Measurement Conference, IMC '14, ACM, New York, NY, USA, 2014, pp. 181–194. doi:10.1145/2663716.2663727.

[28] B. Hesmans, H. Tran-Viet, R. Sadre, O. Bonaventure, A first look at real Multipath TCP traffic, in: M. Steiner, P. Barlet-Ros, O. Bonaventure (Eds.), Traffic Monitoring and Analysis, Vol. 9053 of Lecture Notes in Computer Science, Springer International Publishing, 2015, pp. 233–246. doi:10.1007/978-3-319-17172-2_16.

[29] T. Cymru, IP to ASN mapping, http://www.team-cymru.org/IP-ASN-mapping.html (January 2016).

[30] S. Ostermann, Tcptrace, http://tcptrace.org (2005).

[31] A. Finamore, M. Mellia, M. Meo, M. Munafo, D. Rossi, Experiences of Internet traffic monitoring with tstat, Network, IEEE 25 (3) (2011) 8–14. doi:10.1109/MNET.2011.5772055.

[32] B. Hesmans, F. Duchene, C. Paasch, G. Detal, O. Bonaventure, Are TCP extensions middlebox-proof?, in: Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox

'13, ACM, New York, NY, USA, 2013, pp. 37–42. doi:10.1145/2535828.2535830.

[33] N. Weaver, C. Kreibich, M. Dam, V. Paxson, Here Be Web Proxies, in: Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362, PAM 2014, Springer-Verlag New York, Inc., New York, NY, USA, 2014, pp. 183–192. doi:10.1007/978-3-319-04918-2_18.

[34] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, H. Tokuda, Is it still possible to extend TCP?, in: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, IMC '11, ACM, New York, NY, USA, 2011, pp. 181–194. doi:10.1145/2068816.2068834.

[35] B. Arzani, A. Gurney, S. Cheng, R. Guerin, B. T. Loo, Impact of path characteristics and scheduling policies on MPTCP performance, in: Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on, 2014, pp. 743–748. doi:10.1109/WAINA.2014.121.

[36] B. Hesmans, G. Detal, R. Bauduin, O. Bonaventure, et al., SMAPP: Towards Smart Multipath TCP-enabled APPlications, in: CoNEXT15, 2015.

[37] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, R. J. Gibbens, E. Cecchet, Design, Implementation, and Evaluation of Energy-Aware Multi-Path TCP, in: CoNEXT15, 2015.

[38] C. Paasch, R. Khalili, 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, CoNEXT '13, ACM, New York, NY, USA, 2013, pp. 393–398. doi:10.1145/2535372.2535403.