

" Use Cases and Operational Experience with Multipath TCP"

Bonaventure, Olivier ; Paasch, Christoph ; Detal, Gregory

Abstract

This document discusses both use cases and operational experience with Multipath TCP in real world networks. It lists several prominent use cases for which Multipath TCP has been considered and is being used. It also gives insight to some heuristics and decisions that have helped to realize these use cases.

Document type : *Rapport (Report)*

Référence bibliographique

Bonaventure, Olivier ; Paasch, Christoph ; Detal, Gregory. *Use Cases and Operational Experience with Multipath TCP*. (2015) 24 pages

Available at:

<http://hdl.handle.net/2078.1/162835>

[Downloaded 2019/04/19 at 04:05:40]

MPTCP Working Group
Internet-Draft
Intended status: Informational
Expires: January 7, 2016

O. Bonaventure
C. Paasch
UCLouvain
G. Detal
UCLouvain and Tessares
July 06, 2015

Use Cases and Operational Experience with Multipath TCP
draft-ietf-mptcp-experience-02

Abstract

This document discusses both use cases and operational experience with Multipath TCP in real world networks. It lists several prominent use cases for which Multipath TCP has been considered and is being used. It also gives insight to some heuristics and decisions that have helped to realize these use cases.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Use cases	3
2.1. Datacenters	3
2.2. Cellular/WiFi Offload	4
2.3. Multipath TCP proxies	7
3. Operational Experience	8
3.1. Middlebox interference	8
3.2. Congestion control	10
3.3. Subflow management	11
3.4. Implemented subflow managers	11
3.5. Subflow destination port	13
3.6. Closing subflows	14
4. Packet schedulers	15
5. Segment size selection	16
6. Interactions with the Domain Name System	16
7. Captive portals	17
8. Conclusion	18
9. Acknowledgements	18
10. Informative References	18
Appendix A. Changelog	23
Authors' Addresses	23

1. Introduction

Multipath TCP was standardized in [RFC6824] and four implementations have been developed [[I-D.eardley-mptcp-implementations-survey](#)]. Since the publication of [RFC6824], experience has been gathered by various network researchers and users about the operational issues that arise when Multipath TCP is used in today's Internet.

When the MPTCP working group was created, several use cases for Multipath TCP were identified [RFC6182]. Since then, over use cases have been proposed and some have been tested and even deployed. We describe these use cases in section [Section 2](#).

The second part of the document focuses on the operational experience with Multipath TCP. Most of this experience comes from the utilisation of the Multipath TCP implementation in the Linux kernel [[MultipathTCP-Linux](#)]. This open-source implementation has been downloaded and is used by thousands of users all over the world. Many of these users have provided direct or indirect feedback by writing documents (scientific articles or blog messages) or posting to the mptcp-dev mailing list (see <https://listes->

2.sipr.ucl.ac.be/sympa/arc/mptcp-dev). This Multipath TCP implementation is actively maintained and continuously improved. It is used on various types of hosts, ranging from smartphones or embedded routers to high-end servers.

The Multipath TCP implementation in the Linux kernel is not, by far, the most widespread deployment of Multipath TCP. Since September 2013, Multipath TCP is also supported on smartphones and tablets running iOS7 [IOS7]. There are likely hundreds of millions of Multipath TCP enabled devices. However, this particular Multipath TCP implementation is currently only used to support a single application. Unfortunately, there is no public information about the lessons learned from this large scale deployment.

The second part of this document is organized as follows. Supporting the middleboxes was one of the difficult issues in designing the Multipath TCP protocol. We explain in [Section 3.1](#) which types of middleboxes the Linux Kernel implementation of Multipath TCP supports and how it reacts upon encountering these. [Section 3.2](#) summarises the MPTCP specific congestion controls that have been implemented. [Section 3.3](#) and [Section 4](#) discuss heuristics and issues with respect to subflow management as well as the scheduling across the subflows. [Section 5](#) explains some problems that occurred with subflows having different MSS values. [Section 6](#) presents issues with respect to content delivery networks and suggests a solution to this issue. Finally, [Section 7](#) documents an issue with captive portals where MPTCP will behave suboptimal.

2. Use cases

Multipath TCP has been tested in several use cases. There is already an abundant scientific literature on Multipath TCP [MPTCPBIB]. Several of the papers published in the scientific literature have identified possible improvements that are worth being discussed here.

2.1. Datacenters

A first, although initially unexpected, documented use case for Multipath TCP has been the datacenters [HotNets][SIGCOMM11]. Today's datacenters are designed to provide several paths between single-homed servers. The multiplicity of these paths comes from the utilization of Equal Cost Multipath (ECMP) and other load balancing techniques inside the datacenter. Most of the deployed load balancing techniques in these datacenters rely on hashes computed on the five tuple to ensure that all packets from the same TCP connection will follow the same path to prevent packet reordering. The results presented in [HotNets] demonstrate by simulations that

Multipath TCP can achieve a better utilization of the available network by using multiple subflows for each Multipath TCP session. Although [RFC6182] assumes that at least one of the communicating hosts has several IP addresses, [HotNets] demonstrates that there are also benefits when both hosts are single-homed. This idea was pursued further in [SIGCOMM11] where the Multipath TCP implementation in the Linux kernel was modified to be able to use several subflows from the same IP address. Measurements performed in a public datacenter showed performance improvements with Multipath TCP [SIGCOMM11].

Although ECMP is widely used inside datacenters, this is not the only environment where there are different paths between a pair of hosts. ECMP and other load balancing techniques such as LAG are widely used in today's network and having multiple paths between a pair of single-homed hosts is becoming the norm instead of the exception. Although these multiple paths have often the same cost (from an IGP metrics viewpoint), they do not necessarily have the same performance. For example, [IMC13c] reports the results of a long measurement study showing that load balanced Internet paths between that same pair of hosts can have huge delay differences.

2.2. Cellular/WiFi Offload

A second use case that has been explored by several network researchers is the cellular/WiFi offload use case. Smartphones or other mobile devices equipped with two wireless interfaces are a very common use case for Multipath TCP. As of this writing, this is also the largest deployment of Multipath-TCP enabled devices [IOS7]. Unfortunately, as there are no public measurements about this deployment, we can only rely on published papers that have mainly used the Multipath TCP implementation in the Linux kernel for their experiments.

The performance of Multipath TCP in wireless networks was briefly evaluated in [NSDI12]. One experiment analyzes the performance of Multipath TCP on a client with two wireless interfaces. This evaluation shows that when the receive window is large, Multipath TCP can efficiently use the two available links. However, if the window becomes smaller, then packets sent on a slow path can block the transmission of packets on a faster path. In some cases, the performance of Multipath TCP over two paths can become lower than the performance of regular TCP over the best performing path. Two heuristics, reinjection and penalization, are proposed in [NSDI12] to solve this identified performance problem. These two heuristics have since been used in the Multipath TCP implementation in the Linux kernel. [CONEXT13] explored the problem in more details and revealed some other scenarios where Multipath TCP can have difficulties in

efficiently pooling the available paths. Improvements to the Multipath TCP implementation in the Linux kernel are proposed in [CONEXT13] to cope with some of these problems.

The first experimental analysis of Multipath TCP in a public wireless environment was presented in [Cellnet12]. These measurements explore the ability of Multipath TCP to use two wireless networks (real WiFi and 3G networks). Three modes of operation are compared. The first mode of operation is the simultaneous use of the two wireless networks. In this mode, Multipath TCP pools the available resources and uses both wireless interfaces. This mode provides fast handover from WiFi to cellular or the opposite when the user moves. Measurements presented in [CACM14] show that the handover from one wireless network to another is not an abrupt process. When a host moves, it does not experience either excellent connectivity or no connectivity at all. Instead, there are regions where the quality of one of the wireless networks is weaker than the other, but the host considers this wireless network to still be up. When a mobile host enters such regions, its ability to send packets over another wireless network is important to ensure a smooth handover. This is clearly illustrated from the packet trace discussed in [CACM14].

Many cellular networks use volume-based pricing and users often prefer to use unmetered WiFi networks when available instead of metered cellular networks. [Cellnet12] implements the support for the MP_PRIO option to explore two other modes of operation.

In the backup mode, Multipath TCP opens a TCP subflow over each interface, but the cellular interface is configured in backup mode. This implies that data only flows over the WiFi interface when both interfaces are considered to be active. If the WiFi interface fails, then the traffic switches quickly to the cellular interface, ensuring a smooth handover from the user's viewpoint [Cellnet12]. The cost of this approach is that the WiFi and cellular interfaces likely remain active all the time since all subflows are established over the two interfaces.

The single-path mode is slightly different. This mode benefits from the break-before-make capability of Multipath TCP. When an MPTCP session is established, a subflow is created over the WiFi interface. No packet is sent over the cellular interface as long as the WiFi interface remains up [Cellnet12]. This implies that the cellular interface can remain idle and battery capacity is preserved. When the WiFi interface fails, new subflows are established over the cellular interface in order to preserve the established Multipath TCP sessions. Compared to the backup mode described earlier, this mode of operation is characterised by a throughput drop while the cellular

interface is brought up and the subflows are reestablished. During this time, no data packet is transmitted.

From a protocol viewpoint, [Cellnet12] discusses the problem posed by the unreliability of the ADD_ADDR option and proposes a small protocol extension to allow hosts to reliably exchange this option. It would be useful to analyze packet traces to understand whether the unreliability of the REMOVE_ADDR option poses an operational problem in real deployments.

Another study of the performance of Multipath TCP in wireless networks was reported in [IMC13b]. This study uses laptops connected to various cellular ISPs and WiFi hotspots. It compares various file transfer scenarios and concludes based on measurements with the Multipath TCP implementation in the Linux kernel that "MPTCP provides a robust data transport and reduces variations in download latencies".

A different study of the performance of Multipath TCP with two wireless networks is presented in [INFOCOM14]. In this study the two networks had different qualities : a good network and a lossy network. When using two paths with different packet loss ratios, the Multipath TCP congestion control scheme moves traffic away from the lossy link that is considered to be congested. However, [INFOCOM14] documents an interesting scenario that is summarised in the Figure 1.

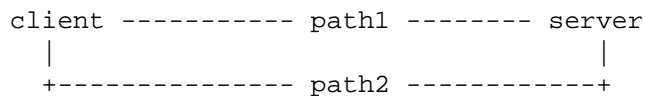


Figure 1: Simple network topology

Initially, the two paths have the same quality and Multipath TCP distributes the load over both of them. During the transfer, the second path becomes lossy, e.g. because the client moves. Multipath TCP detects the packet losses and they are retransmitted over the first path. This enables the data transfer to continue over the first path. However, the subflow over the second path is still up and transmits one packet from time to time. Although the N packets have been acknowledged over the first subflow (at the MPTCP level), they have not been acknowledged at the TCP level over the second subflow. To preserve the continuity of the sequence numbers over the second subflow, TCP will continue to retransmit these segments until either they are acknowledged or the maximum number of retransmissions is reached. This behavior is clearly inefficient and may lead to blocking since the second subflow will consume window space to be able to retransmit these packets. [INFOCOM14] proposes a new

Multipath TCP option to solve this problem. In practice, a new TCP option is probably not required. When the client detects that the data transmitted over the second subflow has been acknowledged over the first subflow, it could decide to terminate the second subflow by sending a RST segment. If the interface associated to this subflow is still up, a new subflow could be immediately reestablished. It would then be immediately usable to send new data and would not be forced to first retransmit the previously transmitted data. As of this writing, this dynamic management of the subflows is not yet implemented in the Multipath TCP implementation in the Linux kernel.

2.3. Multipath TCP proxies

As Multipath TCP is not yet widely deployed on both clients and servers, several deployments have used various forms of proxies. Two families solutions are currently being used or tested [[I-D.deng-mptcp-proxy](#)].

A first use case is when a Multipath TCP enabled client wants to use several interfaces to reach a regular TCP server. A typical use case is a smartphone that needs to use both its WiFi and its cellular interface to transfer data. Several types of proxies are possible for this use case. An HTTP proxy deployed on a Multipath TCP capable server would enable the smartphone to use Multipath TCP to access regular web servers. Obviously, this solution only works for applications that rely on HTTP. Another possibility is to use a proxy that can convert any Multipath TCP connection into a regular TCP connection. The SOCKS protocol [[RFC1928](#)] is an example of such a protocol. Other proxies have been proposed [[I-D.wei-mptcp-proxy-mechanism](#)] [[HotMiddlebox13b](#)]. Measurements performed with smartphones [[Mobicom15](#)] show that popular applications work correctly through a SOCKS proxy and Multipath TCP enabled smartphones. Thanks to Multipath TCP, long connections can be spread over the two available interfaces. However, for short connections, most of the data is sent over the initial subflow that is created over the interface corresponding to the default route and the second subflow is almost not used.

A second use case is when Multipath TCP is used by middleboxes, typically inside access networks. Various network operators are discussing and evaluating solutions for hybrid access networks [[BBF-WT348](#)]. Such networks arise when a network operator controls two different access network technologies, e.g. DSL and LTE, and wants to combine them to improve the bandwidth offered to the endusers [[I-D.lhwz-hybrid-access-network-architecture](#)]. Several solutions are currently investigated for such networks [[BBF-WT348](#)]. Figure 2 shows the organisation of such a network. When a client creates a normal TCP connection, it is intercepted by the Hybrid CPE

(HPCE) that converts it in a Multipath TCP connection so that it can use the available access networks (DSL and LTE in the example). The Hybrid Access Gateway (HAG) does the opposite to ensure that the regular server see a normal TCP connection. Some of the solutions that are currently discussed for those hybrid networks use Multipath TCP on the HCPE and the HAG. Other solutions rely on tunnels between the HCPE and the HAG [[I-D.lhwxyz-gre-notifications-hybrid-access](#)].

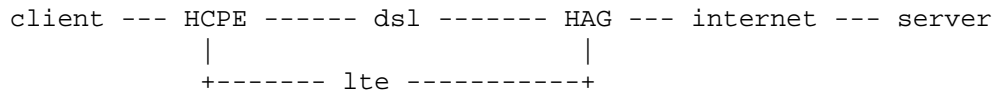


Figure 2: Hybrid Access Network

3. Operational Experience

3.1. Middlebox interference

The interference caused by various types of middleboxes has been an important concern during the design of the Multipath TCP protocol. Three studies on the interactions between Multipath TCP and middleboxes are worth being discussed.

The first analysis was described in [[IMC11](#)]. This paper was the main motivation for including inside Multipath TCP various techniques to cope with middlebox interference. More specifically, Multipath TCP has been designed to cope with middleboxes that :

- o change source or destination addresses
- o change source or destination port numbers
- o change TCP sequence numbers
- o split or coalesce segments
- o remove TCP options
- o modify the payload of TCP segments

These middlebox interferences have all been included in the MBtest suite [[MBTest](#)]. This test suite has been used [[HotMiddlebox13](#)] to verify the reaction of the Multipath TCP implementation in the Linux kernel when faced with middlebox interference. The test environment used for this evaluation is a dual-homed client connected to a single-homed server. The middlebox behavior can be activated on any of the paths. The main results of this analysis are :

- o the Multipath TCP implementation in the Linux kernel is not affected by a middlebox that performs NAT or modifies TCP sequence numbers
- o when a middlebox removes the MP_CAPABLE option from the initial SYN segment, the Multipath TCP implementation in the Linux kernel falls back correctly to regular TCP
- o when a middlebox removes the DSS option from all data segments, the Multipath TCP implementation in the Linux kernel falls back correctly to regular TCP
- o when a middlebox performs segment coalescing, the Multipath TCP implementation in the Linux kernel is still able to accurately extract the data corresponding to the indicated mapping
- o when a middlebox performs segment splitting, the Multipath TCP implementation in the Linux kernel correctly reassembles the data corresponding to the indicated mapping. [HotMiddlebox13] shows on figure 4 in section 3.3 a corner case with segment splitting that may lead to a desynchronisation between the two hosts.

The interactions between Multipath TCP and real deployed middleboxes is also analyzed in [HotMiddlebox13] and a particular scenario with the FTP application level gateway running on a NAT is described.

From an operational viewpoint, knowing that Multipath TCP can cope with various types of middlebox interference is important. However, there are situations where the network operators need to gather information about where a particular middlebox interference occurs. The tracebox software [tracebox] described in [IMC13a] is an extension of the popular traceroute software that enables network operators to check at which hop a particular field of the TCP header (including options) is modified. It has been used by several network operators to debug various middlebox interference problems. tracebox includes a scripting language that enables its user to specify precisely which packet is sent by the source. tracebox sends packets with an increasing TTL/HopLimit and compares the information returned in the ICMP messages with the packet that it sends. This enables tracebox to detect any interference caused by middleboxes on a given path. tracebox works better when routers implement the ICMP extension defined in [RFC1812].

A closer look at the packets received on the multipath-tcp.org server showed that among the 184 thousands Multipath TCP connections in the trace, we observed only 125 of them falling back to regular TCP, which happened with 28 different 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. The HTTP interference appeared only on the direction from server to client and could have been caused by transparent proxies deployed in cellular or enterprise networks.

Users of the Multipath TCP implementation have reported some experience with middlebox interference. The strangest scenario has been a middlebox that accepts the Multipath TCP options in the SYN segment but later replaces Multipath TCP options with a TCP EOL option [[StrangeMbox](#)]. This causes Multipath TCP to perform a fallback to regular TCP without any impact on the application.

3.2. Congestion control

Congestion control has been an important problem for Multipath TCP. The standardised congestion control scheme for Multipath TCP is defined in [[RFC6356](#)] and [[NSDI11](#)]. This congestion control scheme has been implemented in the Linux implementation of Multipath TCP. Linux uses a modular architecture to support various congestion control schemes. This architecture is applicable for both regular TCP and Multipath TCP. While the coupled congestion control scheme defined in [[RFC6356](#)] is the default congestion control scheme in the Linux implementation, other congestion control schemes have been added. The second congestion control scheme is OLIA [[CONEXT12](#)]. This congestion control scheme is also an adaptation of the NewReno single path congestion control scheme to support multiple paths. Simulations and measurements have shown that it provides some performance benefits compared to the the default congestion control scheme [[CONEXT12](#)]. Measurement over a wide range of parameters reported in [[CONEXT13](#)] also indicate some benefits with the OLIA congestion control scheme. Recently, a delay-based congestion control scheme has been ported to the Multipath TCP implementation in the Linux kernel. This congestion control scheme has been evaluated by using simulations in [[ICNP12](#)]. The fourth congestion control scheme that has been included in the Linux implementation of Multipath TCP is the BALIA scheme [[I-D.walid-mptcp-congestion-control](#)].

These different congestion control schemes have been compared in several articles. [[CONEXT13](#)] and [[PaaschPhD](#)] apply an experimental design approach to compare these algorithms in an emulated environment. The evaluation showed that the delay-based congestion control scheme is less able to efficiently use the available links than the three other schemes. Reports from some users indicate that they seem to favor OLIA.

3.3. Subflow management

The multipath capability of Multipath TCP comes from the utilisation of one subflow per path. The Multipath TCP architecture [[RFC6182](#)] and the protocol specification [[RFC6824](#)] define the basic usage of the subflows and the protocol mechanisms that are required to create and terminate them. However, there are no guidelines on how subflows are used during the lifetime of a Multipath TCP session. Most of the experiments with Multipath TCP have been performed in controlled environments. Still, based on the experience running them and discussions on the `mptcp-dev` mailing list, interesting lessons have been learned about the management of these subflows.

From a subflow viewpoint, the Multipath TCP protocol is completely symmetrical. Both the clients and the server have the capability to create subflows. However in practice the existing Multipath TCP implementations [[I-D.eardley-mptcp-implementations-survey](#)] have opted for a strategy where only the client creates new subflows. The main motivation for this strategy is that often the client resides behind a NAT or a firewall, preventing passive subflow openings on the client. Although there are environments such as datacenters where this problem does not occur, as of this writing, no precise requirement has emerged for allowing the server to create new subflows.

3.4. Implemented subflow managers

The Multipath TCP implementation in the Linux kernel includes several strategies to manage the subflows that compose a Multipath TCP session. The basic subflow manager is the full-mesh. As the name implies, it creates a full-mesh of subflows between the communicating hosts.

The most frequent use case for this subflow manager is a multihomed client connected to a single-homed server. In this case, one subflow is created for each interface on the client. The current implementation of the full-mesh subflow manager is static. The subflows are created immediately after the creation of the initial subflow. If one subflow fails during the lifetime of the Multipath TCP session (e.g. due to excessive retransmissions, or the loss of the corresponding interface), it is not always reestablished. There is ongoing work to enhance the full-mesh path manager to deal with such events.

When the server is multihomed, using the full-mesh subflow manager may lead to a large number of subflows being established. For example, consider a dual-homed client connected to a server with three interfaces. In this case, even if the subflows are only

created by the client, 6 subflows will be established. This may be excessive in some environments, in particular when the client and/or the server have a large number of interfaces. A recent draft has proposed a Multipath TCP option to negotiate the maximum number of subflows. However, it should be noted that there have been reports on the `mptcp-dev` mailing indicating that users rely on Multipath TCP to aggregate more than four different interfaces. Thus, there is a need for supporting many interfaces efficiently.

Creating subflows between multihomed clients and servers may sometimes lead to operational issues as observed by discussions on the `mptcp-dev` mailing list. In some cases the network operators would like to have a better control on how the subflows are created by Multipath TCP [[I-D.boucadair-mptcp-max-subflow](#)]. This might require the definition of policy rules to control the operation of the subflow manager. The two scenarios below illustrate some of these requirements.

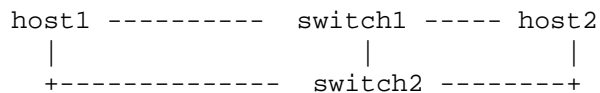


Figure 3: Simple switched network topology

Consider the simple network topology shown in Figure 3. From an operational viewpoint, a network operator could want to create two subflows between the communicating hosts. From a bandwidth utilization viewpoint, the most natural paths are `host1-switch1-host2` and `host1-switch2-host2`. However, a Multipath TCP implementation running on these two hosts may sometimes have difficulties to obtain this result.

To understand the difficulty, let us consider different allocation strategies for the IP addresses. A first strategy is to assign two subnets : `subnetA` (resp. `subnetB`) contains the IP addresses of `host1`'s interface to `switch1` (resp. `switch2`) and `host2`'s interface to `switch1` (resp. `switch2`). In this case, a Multipath TCP subflow manager should only create one subflow per subnet. To enforce the utilization of these paths, the network operator would have to specify a policy that prefers the subflows in the same subnet over subflows between addresses in different subnets. It should be noted that the policy should probably also specify how the subflow manager should react when an interface or subflow fails.

A second strategy is to use a single subnet for all IP addresses. In this case, it becomes more difficult to specify a policy that indicates which subflows should be established.

The second subflow manager that is currently supported by the Multipath TCP implementation in the Linux kernel is the `ndiffport` subflow manager. This manager was initially created to exploit the path diversity that exists between single-homed hosts due to the utilization of flow-based load balancing techniques. This subflow manager creates N subflows between the same pair of IP addresses. The N subflows are created by the client and differ only in the source port selected by the client. It was not designed to be used on multihomed hosts.

3.5. Subflow destination port

The Multipath TCP protocol relies on the token contained in the `MP_JOIN` option to associate a subflow to an existing Multipath TCP session. This implies that there is no restriction on the source address, destination address and source or destination ports used for the new subflow. The ability to use different source and destination addresses is key to support multihomed servers and clients. The ability to use different destination port numbers is worth being discussed because it has operational implications.

For illustration, consider a dual-homed client that creates a second subflow to reach a single-homed server as illustrated in Figure 4.

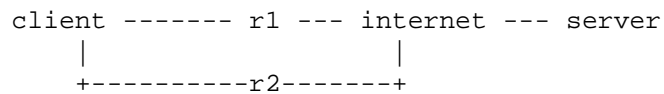


Figure 4: Multihomed-client connected to single-homed server

When the Multipath TCP implementation in the Linux kernel creates the second subflow it uses the same destination port as the initial subflow. This choice is motivated by the fact that the server might be protected by a firewall and only accept TCP connections (including subflows) on the official port number. Using the same destination port for all subflows is also useful for operators that rely on the port numbers to track application usage in their network.

There have been suggestions from Multipath TCP users to modify the implementation to allow the client to use different destination ports to reach the server. This suggestion seems mainly motivated by traffic shaping middleboxes that are used in some wireless networks. In networks where different shaping rates are associated to different destination port numbers, this could allow Multipath TCP to reach a higher performance. As of this writing, we are not aware of any implementation of this kind of tweaking.

However, from an implementation point-of-view supporting different destination ports for the same Multipath TCP connection introduces a new performance issue. A legacy implementation of a TCP stack creates a listening socket to react upon incoming SYN segments. The listening socket is handling the SYN segments that are sent on a specific port number. Demultiplexing incoming segments can thus be done solely by looking at the IP addresses and the port numbers. With Multipath TCP however, incoming SYN segments may have an MP_JOIN option with a different destination port. This means, that all incoming segments that did not match on an existing listening-socket or an already established socket must be parsed for an eventual MP_JOIN option. This imposes an additional cost on servers, previously not existent on legacy TCP implementations.

3.6. Closing subflows

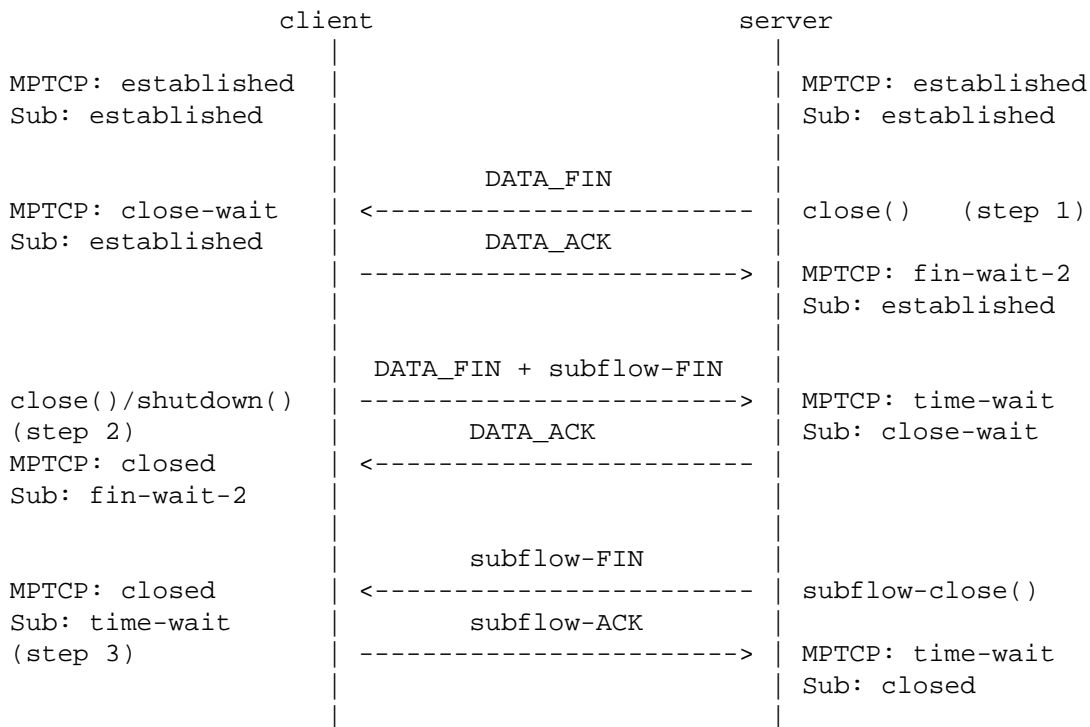


Figure 5: Multipath TCP may not be able to avoid time-wait state (even if enforced by the application).

Figure 5 shows a very particular issue within Multipath TCP. Many high-performance applications try to avoid Time-Wait state by deferring the closure of the connection until the peer has sent a FIN. That way, the client on the left of Figure 5 does a passive

closure of the connection, transitioning from Close-Wait to Last-ACK and finally freeing the resources after reception of the ACK of the FIN. An application running on top of a Multipath TCP enabled Linux kernel might also use this approach. The difference here is that the `close()` of the connection (Step 1 in Figure 5) only triggers the sending of a `DATA_FIN`. Nothing guarantees that the kernel is ready to combine the `DATA_FIN` with a subflow-FIN. The reception of the `DATA_FIN` will make the application trigger the closure of the connection (step 2), trying to avoid Time-Wait state with this late closure. This time, the kernel might decide to combine the `DATA_FIN` with a subflow-FIN. This decision will be fatal, as the subflow's state machine will not transition from Close-Wait to Last-Ack, but rather go through Fin-Wait-2 into Time-Wait state. The Time-Wait state will consume resources on the host for at least 2 MSL (Maximum Segment Lifetime). Thus, a smart application, that tries to avoid Time-Wait state by doing late closure of the connection actually ends up with one of its subflows in Time-Wait state. A high-performance Multipath TCP kernel implementation should honor the desire of the application to do passive closure of the connection and successfully avoid Time-Wait state - even on the subflows.

The solution to this problem lies in an optimistic assumption that a host doing active-closure of a Multipath TCP connection by sending a `DATA_FIN` will soon also send a FIN on all its in subflows. Thus, the passive closer of the connection can simply wait for the peer to send exactly this FIN - enforcing passive closure even on the subflows. Of course, to avoid consuming resources indefinitely, a timer must limit the time our implementation waits for the FIN.

4. Packet schedulers

In a Multipath TCP implementation, the packet scheduler is the algorithm that is executed when transmitting each packet to decide on which subflow it needs to be transmitted. The packet scheduler itself does not have any impact on the interoperability of Multipath TCP implementations. However, it may clearly impact the performance of Multipath TCP sessions. The Multipath TCP implementation in the Linux kernel supports a pluggable architecture for the packet scheduler [[PaaschPhD](#)]. As of this writing, two schedules have been implemented: round-robin and lowest-rtt-first. They are compared in [[CSWS14](#)]. The experiments and measurements described in [[CSWS14](#)] show that the lowest-rtt-first scheduler appears to be the best compromise from a performance viewpoint. Another study of the packet schedulers is presented in [[PAMS2014](#)]. This study relies on simulations with the Multipath TCP implementation in the Linux kernel. These simulations confirm the impact of the packet scheduler on the performance of Multipath TCP.

5. Segment size selection

When an application performs a write/send system call, the kernel allocates a packet buffer (`sk_buff` in Linux) to store the data the application wants to send. The kernel will store at most one MSS (Maximum Segment Size) of data per buffer. As MSS can differ amongst subflows, an MPTCP implementation must select carefully the MSS used to generate application data. The Linux kernel implementation had various ways of selecting the MSS: minimum or maximum amongst the different subflows. However, these heuristics of MSS selection can cause significant performances issues in some environment. Consider the following example. An MPTCP connection has two established subflows that respectively use a MSS of 1420 and 1428 bytes. If MPTCP selects the maximum, then the application will generate segments of 1428 bytes of data. An MPTCP implementation will have to split the segment in two (a 1420-byte and 8-byte segments) when pushing on the subflow with the smallest MSS. The latter segment will introduce a large overhead as for a single data segment 2 slots will be used in the congestion window (in packets) therefore reducing by ~ 2 the potential throughput (in bytes/s) of this subflow. Taking the smallest MSS does not solve the issue as there might be a case where the subflow with the smallest MSS will only participate marginally to the overall performance therefore reducing the potential throughput of the other subflows.

The Linux implementation recently took another approach [[DetailMSS](#)]. Instead of selecting the minimum and maximum values, it now dynamically adapts the MSS based on the contribution of all the subflows to the connection's throughput. For this it computes, for each subflow, the potential throughput achieved by selecting each MSS value and by taking into account the lost space in the cwnd. It then selects the MSS that allows to achieve the highest potential throughput.

6. Interactions with the Domain Name System

Multihomed clients such as smartphones can send DNS queries over any of their interfaces. When a single-homed client performs a DNS query, it receives from its local resolver the best answer for its request. If the client is multihomed, the answer returned to the DNS query may vary with the interface over which it has been sent.

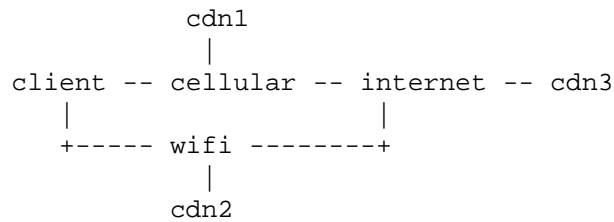


Figure 6: Simple network topology

If the client sends a DNS query over the WiFi interface, the answer will point to the cdn2 server while the same request sent over the cellular interface will point to the cdn1 server. This might cause problems for CDN providers that locate their servers inside ISP networks and have contracts that specify that the CDN server will only be accessed from within this particular ISP. Assume now that both the client and the CDN servers support Multipath TCP. In this case, a Multipath TCP session from cdn1 or cdn2 would potentially use both the cellular network and the WiFi network. This would violate the contract between the CDN provider and the network operators. A possible solution to prevent this problem would be to modify the DNS resolution on the client. The client subnet EDNS extension defined in [I-D.vandergaast-edns-client-subnet] could be used for this purpose. When the client sends a DNS query from its WiFi interface, it should also send the client subnet corresponding to the cellular interface in this request. This would indicate to the resolver that the answer should be valid for both the WiFi and the cellular interfaces (e.g., the cdn3 server).

7. Captive portals

Multipath TCP enables a host to use different interfaces to reach a server. In theory, this should ensure connectivity when at least one of the interfaces is active. In practice however, there are some particular scenarios with captive portals that may cause operational problems. The reference environment is shown in Figure 7.

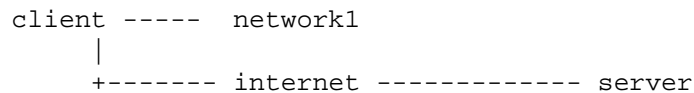


Figure 7: Issue with captive portal

The client is attached to two networks : network1 that provides limited connectivity and the entire Internet through the second network interface. In practice, this scenario corresponds to an open

WiFi network with a captive portal for network1 and a cellular service for the second interface. On many smartphones, the WiFi interface is preferred over the cellular interface. If the smartphone learns a default route via both interfaces, it will typically prefer to use the WiFi interface to send its DNS request and create the first subflow. This is not optimal with Multipath TCP. A better approach would probably be to try a few attempts on the WiFi interface and then try to use the second interface for the initial subflow as well.

8. Conclusion

In this document, we have documented a few years of experience with Multipath TCP. The information presented in this document was gathered from scientific publications and discussions with various users of the Multipath TCP implementation in the Linux kernel.

9. Acknowledgements

This work was partially supported by the FP7-Trilogy2 project. We would like to thank all the implementers and users of the Multipath TCP implementation in the Linux kernel. This document has benefited from the comments of John Ronan, Yoshifumi Nishida, Phil Eardley and Jaehyun Hwang.

10. Informative References

[BBF-WT348]

Fabregas (Ed), G., "WT-348 - Hybrid Access for Broadband Networks", Broadband Forum, contribution bbf2014.1139.04 , June 2015.

[CACM14]

Paasch, C. and O. Bonaventure, "Multipath TCP", Communications of the ACM, 57(4):51-57 , April 2014, <<http://inl.info.ucl.ac.be/publications/multipath-tcp>>.

[CONEXT12]

Khalili, R., Gast, N., Popovic, M., Upadhyay, U., and J. Leboudec, "MPTCP is not pareto-optimal performance issues and a possible solution", Proceedings of the 8th international conference on Emerging networking experiments and technologies (CoNEXT12) , 2012.

- [CONEXT13] Paasch, C., Khalili, R., and O. Bonaventure, "On the Benefits of Applying Experimental Design to Improve Multipath TCP", Conference on emerging Networking EXperiments and Technologies (CoNEXT) , December 2013, <<http://inl.info.ucl.ac.be/publications/benefits-applying-experimental-design-improve-multipath-tcp>>.
- [CSWS14] Paasch, C., Ferlin, S., Alay, O., and O. Bonaventure, "Experimental Evaluation of Multipath TCP Schedulers", SIGCOMM CSWS2014 workshop , August 2014.
- [Cellnet12] Paasch, C., Detal, G., Duchene, F., Raiciu, C., and O. Bonaventure, "Exploring Mobile/WiFi Handover with Multipath TCP", ACM SIGCOMM workshop on Cellular Networks (Cellnet12) , 2012, <<http://inl.info.ucl.ac.be/publications/exploring-mobilewifi-handover-multipath-tcp>>.
- [DetalMSS] Detal, G., "Adaptive MSS value", Post on the mptcp-dev mailing list , September 2014, <<https://listes-2.sipr.ucl.ac.be/sympa/arc/mptcp-dev/2014-09/msg00130.html>>.
- [HotMiddlebox13] Hesmans, B., Duchene, F., Paasch, C., Detal, G., and O. Bonaventure, "Are TCP Extensions Middlebox-proof?", CoNEXT workshop HotMiddlebox , December 2013, <<http://inl.info.ucl.ac.be/publications/are-tcp-extensions-middlebox-proof>>.
- [HotMiddlebox13b] Detal, G., Paasch, C., and O. Bonaventure, "Multipath in the Middle(Box)", HotMiddlebox'13 , December 2013, <<http://inl.info.ucl.ac.be/publications/multipath-middlebox>>.
- [HotNets] Raiciu, C., Pluntke, C., Barre, S., Greenhalgh, A., Wischik, D., and M. Handley, "Data center networking with multipath TCP", Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX) , 2010, <<http://doi.acm.org/10.1145/1868447.1868457>>.

- [I-D.boucadair-mptcp-max-subflow]
Boucadair, M. and C. Jacquenet, "Negotiating the Maximum Number of MPTCP Subflows", [draft-boucadair-mptcp-max-subflow-00](#) (work in progress), June 2015.
- [I-D.deng-mptcp-proxy]
Lingli, D., Liu, D., Sun, T., Boucadair, M., and G. Cauchie, "Use-cases and Requirements for MPTCP Proxy in ISP Networks", [draft-deng-mptcp-proxy-01](#) (work in progress), October 2014.
- [I-D.eardley-mptcp-implementations-survey]
Eardley, P., "Survey of MPTCP Implementations", [draft-eardley-mptcp-implementations-survey-02](#) (work in progress), July 2013.
- [I-D.lhwxz-gre-notifications-hybrid-access]
Leymann, N., Heidemann, C., Wasserman, M., Xue, L., and M. Zhang, "GRE Notifications for Hybrid Access", [draft-lhwxz-gre-notifications-hybrid-access-01](#) (work in progress), January 2015.
- [I-D.lhwxz-hybrid-access-network-architecture]
Leymann, N., Heidemann, C., Wasserman, M., Xue, L., and M. Zhang, "Hybrid Access Network Architecture", [draft-lhwxz-hybrid-access-network-architecture-02](#) (work in progress), January 2015.
- [I-D.vandergaast-edns-client-subnet]
Contavalli, C., Gaast, W., Leach, S., and E. Lewis, "Client Subnet in DNS Requests", [draft-vandergaast-edns-client-subnet-02](#) (work in progress), July 2013.
- [I-D.walid-mptcp-congestion-control]
Walid, A., Peng, Q., Hwang, J., and S. Low, "Balanced Linked Adaptation Congestion Control Algorithm for MPTCP", [draft-walid-mptcp-congestion-control-02](#) (work in progress), January 2015.
- [I-D.wei-mptcp-proxy-mechanism]
Wei, X., Xiong, C., and E. Ed, "MPTCP proxy mechanisms", [draft-wei-mptcp-proxy-mechanism-02](#) (work in progress), June 2015.
- [ICNP12] Cao, Y., Xu, M., and X. Fu, "Delay-based congestion control for multipath TCP", 20th IEEE International Conference on Network Protocols (ICNP) , 2012.

- [IMC11] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it still possible to extend TCP?", Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference (IMC '11) , 2011, <<http://doi.acm.org/10.1145/2068816.2068834>>.
- [IMC13a] Detal, G., Hesmans, B., Bonaventure, O., Vanaubel, Y., and B. Donnet, "Revealing Middlebox Interference with Tracebox", Proceedings of the 2013 ACM SIGCOMM conference on Internet measurement conference , 2013, <<http://inl.info.ucl.ac.be/publications/revealing-middlebox-interference-tracebox>>.
- [IMC13b] Chen, Y., Lim, Y., Gibbens, R., Nahum, E., Khalili, R., and D. Towsley, "A measurement-based study of MultiPath TCP performance over wireless network", Proceedings of the 2013 conference on Internet measurement conference (IMC '13) , n.d., <<http://doi.acm.org/10.1145/2504730.2504751>>.
- [IMC13c] Pelsser, C., Cittadini, L., Vissicchio, S., and R. Bush, "From Paris to Tokyo on the suitability of ping to measure latency", Proceedings of the 2013 conference on Internet measurement conference (IMC '13) , 2013, <<http://doi.acm.org/10.1145/2504730.2504765>>.
- [INFOCOM14] Lim, Y., Chen, Y., Nahum, E., Towsley, D., and K. Lee, "Cross-Layer Path Management in Multi-path Transport Protocol for Mobile Devices", IEEE INFOCOM'14 , 2014.
- [IOS7] "Multipath TCP Support in iOS 7", January 2014, <<http://support.apple.com/kb/HT5977>>.
- [MBTest] Hesmans, B., "MBTest", 2013, <<https://bitbucket.org/bhesmans/mbtest>>.
- [MPTCPBIB] Bonaventure, O., "Multipath TCP - An annotated bibliography", Technical report , April 2015, <<https://github.com/obonaventure/mptcp-bib>>.
- [Mobicom15] De Coninck, Q., Baerts, M., Hesmans, B., and O. Bonaventure, "Poster - Evaluating Android Applications with Multipath TCP", Mobicom 2015 (Poster) , September 2015.

- [MultipathTCP-Linux]
Paasch, C., Barre, S., and . et al, "Multipath TCP implementation in the Linux kernel", n.d., <<http://www.multipath-tcp.org>>.
- [NSDI11] Wischik, D., Raiciu, C., Greenhalgh, A., and 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 (NSDI11) , 2011.
- [NSDI12] Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP", USENIX Symposium of Networked Systems Design and Implementation (NSDI12) , April 2012, <<http://inl.info.ucl.ac.be/publications/how-hard-can-it-be-designing-and-implementing-deployable-multipath-tcp>>.
- [PAMS2014] Arzani, B., Gurney, A., Cheng, S., Guerin, R., and B. Loo, "Impact of Path Selection and Scheduling Policies on MPTCP Performance", PAMS2014 , 2014.
- [PaaschPhD] Paasch, C., "Improving Multipath TCP", Ph.D. Thesis , November 2014, <<http://inl.info.ucl.ac.be/publications/improving-multipath-tcp>>.
- [RFC1812] Baker, F., "Requirements for IP Version 4 Routers", [RFC 1812](#), June 1995.
- [RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", [RFC 1928](#), March 1996.
- [RFC6182] Ford, A., Raiciu, C., Handley, M., Barre, S., and J. Iyengar, "Architectural Guidelines for Multipath TCP Development", [RFC 6182](#), March 2011.
- [RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols", [RFC 6356](#), October 2011.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", [RFC 6824](#), January 2013.

[SIGCOMM11]

Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and M. Handley, "Improving datacenter performance and robustness with multipath TCP", Proceedings of the ACM SIGCOMM 2011 conference , n.d., <<http://doi.acm.org/10.1145/2018436.2018467>>.

[StrangeMbox]

Bonaventure, O., "Multipath TCP through a strange middlebox", Blog post , January 2015, <http://blog.multipath-tcp.org/blog/html/2015/01/30/multipath_tcp_through_a_strange_middlebox.html>.

[tracebox]

Detal, G., "tracebox", 2013, <<http://www.tracebox.org>>.

Appendix A. Changelog

- o initial version : September 16th, 2014 : Added section [Section 5](#) that discusses some performance problems that appeared with the Linux implementation when using subflows having different MSS values
- o update with a description of the middlebox that replaces an unknown TCP option with EOL [[StrangeMbox](#)]
- o version ietf-02 : July 2015, answer to last call comments
 - * Reorganised text to better separate use cases and operational experience
 - * New use case on Multipath TCP proxies in [Section 2.3](#)
 - * Added some text on middleboxes in [Section 3.1](#)
 - * Removed the discussion on SDN
 - * Restructured text and improved writing in some parts

Authors' Addresses

Olivier Bonaventure
UCLouvain

Email: Olivier.Bonaventure@uclouvain.be

Christoph Paasch
UCLouvain

Email: Christoph.Paasch@gmail.com

Gregory Detal
UCLouvain and Tessaes

Email: Gregory.Detal@tessaes.net