RobE: Robust Connection Establishment for Multipath TCP

Markus Amend
Deutsche Telekom, Darmstadt
Darmstadt, Germany
Markus.Amend@telekom.de

Veselin Rakocevic
City University London
London, United Kingdom
Veselin.Rakocevic.1@city.ac.uk

Andreas Philipp Matz
Deutsche Telekom, Darmstadt
Darmstadt, Germany
info@andreasmatz.de

Eckard Bogenfeld
Deutsche Telekom, Darmstadt
Darmstadt, Germany
Eckard.Bogenfeld@telekom.de

ABSTRACT

Over the past decade, the way that devices connect to the Internet has changed drastically. The introduction of high-speed mobile networks and the omnipresence of wireless access points led to a situation where most devices offer multiple network interfaces, such as cellular and WiFi radios. However the most prevalent Layer 4 protocol, TCP, is not multipath capable and thus limited to using one interface at a time [4]. Multipath TCP [6], which is based on the classic TCP, resolves this problem by spreading the data flow across multiple paths. When a new connection is established, an initial subflow is created for backwards compatibility to TCP; afterwards, additional subflows can be created and MPTCP offers resilience against link failures [1]. Depending on the first flow to create a new connection prevents MPTCP from exploiting multi-link capabilities during the establishment phase. Therefore, if the initial subflow is not successful, a connection cannot be established. In this paper an extension to the existing MPTCP standard is proposed: *MPTCP RobE*. This solution extends the inherent resiliency of MPTCP against link failures to the establishment phase by introducing the concept of several potentially initial subflows, which tremendously increases the chance to build a successful initial subflow, and at the same time improves end-to-end latency during the establishment phase significantly. In this paper, three different concepts for MPTCP RobE are proposed and discussed with regard to technical aspects and MPTCP standard integration. Finally, a protoype is used to verify the performance of MPTCP RobE with respect to robustness and latency gains in different scenarios.

CCS CONCEPTS

• **Networks → Transport protocols**; *Network protocol design*; *Network performance analysis*;

ANRW '18, July 16, 2018, Montreal, QC, Canada

ACM ISBN 978-1-4503-5585-8/18/07…$15.00
https://doi.org/10.1145/3232755.3232762

KEYWORDS

Multi-Path TCP, MPTCP, Connection Establishment, Next generation networking, Computer network reliability, Disruption tolerant networking, Measurements.

ACM Reference Format:
Markus Amend, Veselin Rakocevic, Andreas Philipp Matz, and Eckard Bogenfeld. 2018. RobE: Robust Connection Establishment for Multipath TCP. In *ANRW '18: Applied Networking Research Workshop, July 16, 2018, Montreal, QC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3232755.3232762

1 INTRODUCTION

In 2009, a new TCP [5] extension was proposed, which resolved the problem of TCP's 5-tuple binding: *Multipath TCP (MPTCP)* [2]. In contrast to classic TCP, MPTCP is capable of dynamically managing the addresses belonging to one logical end-to-end connection. Depending on the policy set, the MPTCP stack can create multiple *subflows* between pairs of addresses, which resemble classic TCP connections. These subflows can be used for bandwidth aggregation and resilience against network outages [1].

Similar to classic TCP, a new MPTCP connection will be initiated by a three-way handshake. The first subflow in a connection is referred to as the *initial subflow* and is the key factor in retaining backwards compatibility to classic TCP: MPTCP includes a new TCP option[1] `MP_CAPABLE` in order to verify protocol support, and exchange two keys (Key_A and Key_B) that will be used for authentication of any further subflows. If the remote host supports the protocol, it will also include the `MP_CAPABLE` option - both hosts are now aware of each others capability. Figure 1 illustrates the initial MPTCP handshake. If the `SYN/ACK` does not carry the `MP_CAPABLE` option, both hosts will continue and set up a traditional TCP connection.

Once the initial subflow is set up, the connection is ready to exchange data and additional *subsequent subflows* can be created using the same or a different address pair. A special four-way handshake is employed for authenticating the new subflow using the cryptographic information exchanged in the first step. MPTCP signals the creation of a subsequent flow using the `MP_JOIN` TCP option, which

[1] RFC 1323 [3] defines TCP options as the recommended way to extend TCP functionality. When a host receives a TCP option it does not understand, it will silently ignore the option. For MPTCP, a missing `MP_CAPABLE` option in the peer's response indicates the lack of MPTCP support on the remote host.

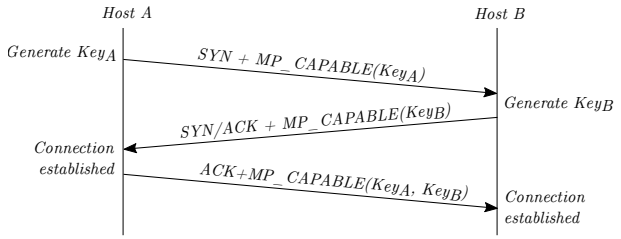

Figure 1: The MPTCP initial subflow handshake

uses cryptographic *HMAC* sums ($HMAC_A$ and $HMAC_B$) for authentication. Additional random numbers ($Random_A$ and $Random_B$) serve as a replay protection. Figure 2 gives an overview on the information exchanged in the subsequent handshake.

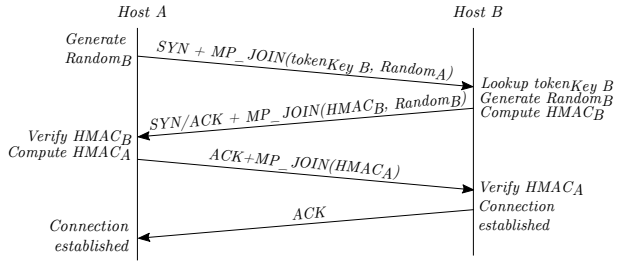

Figure 2: The MPTCP subsequent subflow handshake

At this point, one major flaw of MPTCP becomes obvious: MPTCP uses only one path to establish the initial subflow. If this path is interrupted, the initial subflow cannot be established - and since all subsequent subflows depend on the initial flow, a connection cannot be established, although other paths might provide connectivity to the target host.

In 2015, the idea of a MPTCP extension was born at Deutsche Telekom, which enables the use of all available paths for connection establishment: *Robust Establishment (RobE)*. RobE's design goal is to integrate into the existing MPTCP standard without breaking it; it works mainly by establishing *potentially initial subflows* over all available paths to a target host. The process is similar to establishing an initial subflow in classic MPTCP, but the `MP_CAPABLE` option is used on all paths - the strong separation between initial and subsequent subflows in classic MPTCP is abolished.

Several RobE concepts were designed and will be discussed in this paper. Furthermore, the RobE extension has been implemented, based on the favored concept, as a first prototype allowing to evaluate the solution under different aspects. In this paper, the following question is posed: *How can MPTCP be protected against outages during connection establishment?*

2 THREE PROPOSALS FOR MPTCP ROBE

MPTCP is standardized in RFC 6824 [2], which defines that an initial subflow must be established over one path before any additional flows can be created. However, if there is a network outage along the path from client to server, the initial path may be dysfunctional. With classic MPTCP, the initial subflow cannot be established, and the entire connection will be unsuccessful, although there may be other paths that do not share this point of failure. Depending on the system configuration, the MPTCP stack will retry multiple times [5] before giving up and notifying the user application that the server is not available. The MPTCP RobE approach resolves this problem by exploiting multiple available paths during connection establishment.

In the following sections three different approaches are proposed that expand the current MPTCP standard by a resilient connection establishment. In these approaches, multiple *potentially initial subflows* are created on the available paths. This special kind of flow can take any role found in the current MPTCP standard, i.e. it can be used as an initial or a subsequent subflow. As a result, any connection request arriving a the target host can initiate a new end-to-end connection.

For simplicity reasons, the following sections assume a simple network architecture consisting of a client, referred to as *Host A* and equipped with the interfaces *Int A1* and *Int A2*, and the server *Host B*, with the corresponding interface *Int B1*. Figure 3 reflects this architecture. The following would also apply to any other network structure with additional multiple interfaces on both ends.

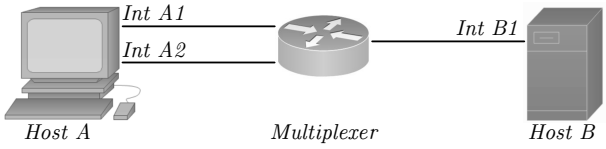

Figure 3: An exemplary MPTCP RobE network architecture

2.1 Proposal 1: Downgrading potential initial subflows

In the first proposal, *potentially initial subflows* are introduced, which are created over any available path simultaneously. The first flow that changes state to `ESTABLISHED` will be declared the initial subflow, while the remaining successful paths are downgraded to subsequent subflows. The key concept of this approach is that any flow can take any role found in the current standard. This process is symmetrical for both client and server side.

During connection establishment, the client sends SYNs along both available paths $IntA1 \leftrightarrow IntB1$ and $IntA2 \leftrightarrow IntB1$ towards the server. The segments are identical apart from the particular source IPs and ports. As a result, any request arriving at the server can trigger the creation of a new end-to-end connection. Related SYNs are identified by using an identical sender key; if a fictional key value `Key A` is unknown, the server will respond with a SYN/ACK containing a newly generated `Key B`. On the other hand, if the comparison against the known keys reveals that `Key A` is already used in an existing connection, the server will respond with the same `Key B` that was used in that connection.

As soon as the first SYN/ACK arrives at the client, the connection changes to `ESTABLISHED`, and this particular flow is declared the *initial subflow*. As more SYN/ACKs arrive, their receiver key is compared against all `ESTABLISHED` keys that are currently in use. If a match is found, the flow is attached to the corresponding connection (i.e. downgraded to a subsequent flow). For every

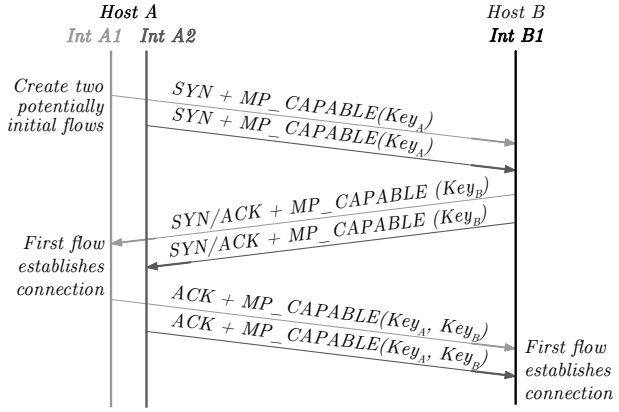

Figure 4: Proposal 1 - Downgrading a potential initial subflow

SYN/ACK received, the client will respond with a final ACK containing both `Key A` and `Key B`, following the MPTCP standard [2]. The server performs similar actions: The first ACK that returns will create the initial flow, while all other responses received will create subsequent subflows. This concept is illustrated in Figure 4.

2.2 Proposal 2: Break Before Make

The second proposal resembles the first one in that the client sends out connection requests on all available interfaces, and the server answers accordingly. However, only one flow is allowed to become fully established, while all other flows are reset. Figure 5 shows the process of establishing a new connection with the second proposal.

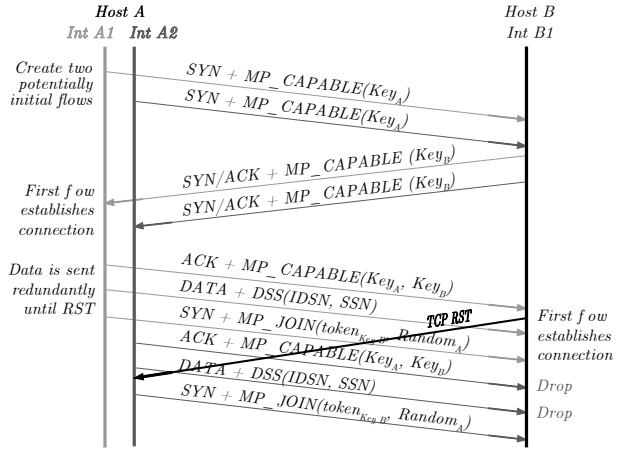

Figure 5: Proposal 2 - Break before make

The process for establishing a connection is widely identical to the first approach: Both client and server will use identical Keys `A` and `B` to identify related subflows. Once the first `ACK` segment arrives at the server, the end-to-end connection is established. All other related flows, that are still in the establishment phase, will now be terminated by the server using a TCP `RST`. Similar to classic MPTCP, data transmission is now possible and the client can create additional subflows as specified in [2]. In this approach, the client immediately tries to create additional flows and send data. As the client does not know which flow is the initial subflow until it has received TCP resets on all paths but one, it will send both `SYN+MP_JOIN` and data segments redundantly on all available paths. This results in one redundant subsequent path along the initial subflow, which needs to be broken down later.

2.3 Proposal 3: The Timer Solution

The third proposal is designed to stay close to MPTCP [2] in that standard functionality is used wherever possible. Resiliency against network outages is achieved by modifying the `SYN` retransmission timer: If one path is defective, another path is used.

A new connection is initiated by sending a `SYN+MP_CAPABLE` along the initial path. If this path is functional, the solution will perform identical to classic MPTCP: the initial flow will be established, and subsequent flows can be created afterwards. If however the initial path is faulty, the retransmission will be triggered on *another path*. This path might circumvent the dysfunctional network, and allow the client to create an initial subflow. The first path is now seen as a subsequent path and the client sends `SYN+MP_JOIN` messages to create a subsequent flow.

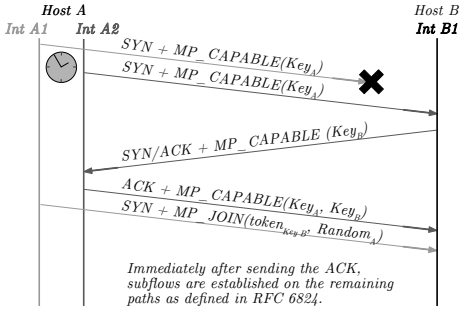

Figure 6: Proposal 3 - The timer solution

In high latency networks, the initial `SYN+MP_CAPABLE` might be delayed until the client retries on another path. Once the second `SYN` arrives at the server, it will try to complete the three-way handshake. If the first `SYN` was delayed by more than the retransmission time plus half a Round Trip Time (RTT) of the second path, it will arrive at the server after the second `SYN`. The server could now either treat the segment as obsolete and drop it or degrade the flow to a subsequent subflow as shown in the first approach (thereby additionally reducing network overhead). The third proposal is illustrated in Figure 6.

2.4 MPTCP RobE Design Decisions

The decision on which proposal to implement was made based on multiple boundary conditions the solution should fulfill.

- **Robustness:** If there is at least one functional path, a connection must be possible.
- **Overhead and Latency:** The solution should not introduce excessive amounts of overhead and latency compared to standard MPTCP.

- **Standard compliance:** Standardized functionalities should be used wherever possible. The extension should respect existing standards and work around limitations. Optionally, a fallback mechanism to classic MPTCP or even TCP could be included.

All three proposals have been designed to meet the first boundary condition. Whenever there is at least one path that provides connectivity, all three approaches will successfully create a new end-to-end connection.

The second approach implements many key concepts of classic MPTCP. However, it introduces a lot of overhead: All but one potentially initial flows must be broken down again, and data must be sent redundantly until the TCP reset has arrived on all but one path, as the client cannot know which path was successful. Additionally, a redundant subsequent subflow is created on the initial path, which should be terminated to avoid interference with the initial subflow. The second approach is not optimal in terms of efficiency.

The third approach makes extensive use of standard MPTCP functionality and is thus possibly the easiest to implement. As a direct result, the overhead generated is very low. However, if the initial path does not provide connectivity, an additional delay is introduced before the end-to-end connection is established which can be a multitude of the RTT of the initial path. In Equation 1, this delay is shown as $\Delta t_{Handshake}$, N is the number of unsuccessful paths tried before a functional path is found and TCP_RTO is the timeout until the next path is tried.

$$\Delta t_{Handshake} = \sum_{i=0}^{N-1} TCP\_RTO \tag{1}$$

This problem could be alleviated by automatically adjusting the initial respectively default route after multiple connection requests over the initial path have failed. Overall, while resolving the issue of robustness, the third approach is not suitable for implementation, because in case of a robustness demand, the connectivity latency increase significantly and may interfere with application behavior.

The only approach which meets all of the above criteria is the first proposal (see Table 1). With this approach, the end-to-end latency is determined by the path with the lowest latency, which could improve the latency until a connection is established. As potentially initial flows are converted to subsequent ones, the overall latency until all flows are available can be significantly reduced. Additionally, network overhead does not increase when compared to classic MPTCP - every flow that is established can be used for data communication afterwards, and additional flows (e.g. connections to different server interfaces) can be established as usual. The first proposal was thus chosen as the baseline for the implementation of MPTCP RobE based on the Linux MPTCP reference implementation. The only remaining uncertainty needing further investigation concerns the integration in the current RFC standard. From the current point of view, most standard functionalities could be reused.

3 EXPERIMENT SETUP

In this section, MPTCP RobE will be evaluated in different situations where the user experience can benefit from establishing multiple flows simultaneously. There are two main benefits of MPTCP RobE:

Table 1: Criteria match of the three proposals

Criteria	Proposal		
	(1) Downgrade	(2) Break before make	(3) Timer
Robustness	✓	✓	✓
No Overhead	✓	✗	✓
No Latency incr.	✓	✓	✗
Latency decrease	✓	✗	✗
Standard compliance	(✓)	(✓)	✓

Robustness and Latency reduction. In a first experiment the robustness gain is analyzed and compared to the standard MPTCP approach. In a second set of experiments, the influence of MPTCP RobE on end-to-end latency shall be examined. Besides improving connection reliability in case of one or multiple faulty paths, MPTCP RobE can significantly improve the latency until an end-to-end connection is established if the initial path does not provide the lowest latency to the target host. Additionally, it profits from having subsequent flows available earlier without needing a `MP_JOIN`. The real-world benefit of the resulting latency decrease is verified by examining the loading times of the top ten most popular websites. The parallel establishment of multiple subflows decreases the time needed for the initial connection establishmentt_i, from $t_{i,RFC6824}$ to $t_{i,RobE}$.

$$t_{i,RobE} = 1.5 * RTT_{RobE} \tag{2}$$

with

RTT_{RobE}:Round Trip Time fastest potential initial path

Compared to standard MPTCP [2]:

$$t_{i,RFC6824} = 1.5 * RTT_{RFC6824} \tag{3}$$

with

$RTT_{RFC6824}$:Round Trip Time initial path according to [2]

The accelleration of the initial establishment can be derived as the gain A_i:

$$A_i = \frac{t_{i,RFC6824}}{t_{i,RobE}} = \frac{RTT_{RFC6824}}{RTT_{RobE}} \tag{4}$$

Another strong driver of the selected MPTCP RobE approach is the downgrade feature, which accelerates subsequent flow establishment by converting successful potential initial flows to subsequent ones. As defined in RFC6824 [2], a *DSS* option must be received before additional flows can be established. If this option could be piggybacked on the *SYN/ACK*, subsequent flows would be available after the time

$$t_{s,RFC6824} = RTT_{RFC6824} + 1.5 * RTT_s \tag{5}$$

Unfortunately, the MPTCP standard prohibits sending the *DSS* during the handshake (see [2, p.41]), which delays the establishment of the subsequent flows by another RTT_s.

$$t_{s,RFC6824} = RTT_{RFC6824} + 2.5 * RTT_s \quad (6)$$

With MPTCP RobE it changes for each subsequent flow to

$$t_{s,RobE} = 1.5 * RTT_s \quad (7)$$

with a time gain of

$$A_s = \frac{t_{s,RFC6824}}{t_{s,RobE}} \quad (8)$$

The overall time gain, until all flows are established, which are available from the very beginning, can be specified by

$$A_{overall} = \frac{t_{i,RFC6824} + \max(t_{s,RFC6824,1}..t_{s,RFC6824,N})}{t_{i,RobE} + \max(t_{s,RobE,1}..t_{s,RobE,N})} \quad (9)$$

with N as the total number of subsequent paths.

In all two experiments both client and server will be running Ubuntu Linux 14.04 with the MPTCP v0.90 kernel implementation [6], enhanced with selectable MPTCP RobE; both will be configured to create a maximum of two connections according to the number of interfaces available (Figure 3).

3.1 Experiment 1: Robustness

The central benefit of establishing multiple flows simultaneously is enhanced resiliency against link failures. A common application for MPTCP is bundling two paths which are heterogenous in their latency and reliability. If an unreliable path is chosen as the default route (e.g. a WLAN link), the initial `SYN` request might get lost, which introduces a large delay - the client will not retry until the retransmission timeout triggers, which is a multitude of the Round Trip Time (RTT) in most cases. If there is a connection interrupt on the default route, and all retransmitted `SYN` segments are lost, the client will be unable to establish a new connection.

The network structure of this experiment corresponds to the one found in Figure 3. The client uses two distinct network interfaces to connect to a server, which is equipped with one Ethernet interface. Client and server machine are located on the same physical host to share the same time base. In this experiment the Linux tool `tc` is used on the client machine in order to simulate a lossy default route and link latencies of 20ms. After each increment, the time span between running a system call *connect()* on a TCP *socket(AF_INET, SOCK_STREAM, 0)* on client side and a successfull *accept()*[2] on server side is measured. This represents nearly the time needed until MPTCP has finished initial flow establishment and is able to transmit payload.

3.2 Experiment 2: Real-world evaluation

In this experiment the client will download the top ten most popular websites[3] using the tool `wget`[4] in an automated procedure. For each website, the time from the first request until the termination of the last connection is measured and stored. The test will be conducted using a filter list for third-party content (commonly referred to as "ad-blocker"). The rationale behind this is that on every access to a web site, ads from different Content Delivery Networks (CDNs) are shown to the user. This behavior results in significant variance of page loading times, as the time for completing the request includes all website elements that would be displayed to a user in the browser.

The network architecture corresponds again to Figure 3. The client's interfaces are limited to 10Mbits/s to avoid any CPU limitation in the measurement. One of the interfaces will simulate a classic DSL-over-WiFi link with low latency (20ms), while the other link will resemble a LTE link with high latency (40ms), with the simulated LTE link chosen as the initial path. The measurement is performed in one minute intervals until 30 data points have been collected for each website; this reduces the impact of temporary network outages.

Since most of the servers on the Internet are not yet MPTCP capable, a translation to classic TCP is needed if aggregation via MPTCP shall be used with these endpoints. The host responsible for this translation is a server in a public data center and is referred to as the *Hybrid Access Aggregation Point (HAAP)*. On the HAAP a transparent TCP proxy is used for both terminating the incoming MPTCP connection and creating a new classic TCP connection towards the destination host. From the client's perspective, there is no difference between a response from the HAAP and the real target host. Once both connections are established, the HAAP server will forward data packets in both directions. The network architecture used in this experiment is exhibited in Figure 7. The measurement results will be discussed relative to each other, which compensates for any effects that could possibly be introduced by the HAAP server.

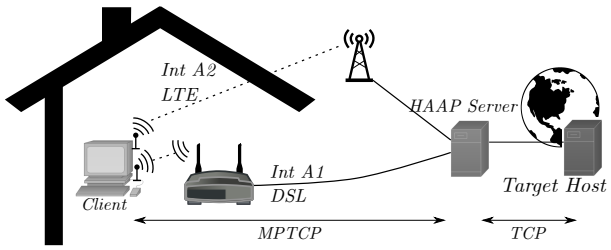

Figure 7: The targeted network architecture used in experiment 2.2

4 RESULTS AND DISCUSSION

The MPTCP RobE extension was evaluated in different scenarios where it can be beneficial to the overall user experience. These scenarios include the robust establishment of connections and improved latency under certain circumstances. In this section, the measurement results will be presented and discussed.

4.1 Results of Experiment 1: Robustness

With MPTCP the initial subflow needed for connection establishment will always use a certain initial path. This path is not adapted

[2]The aforementioned syscalls are not limited to the TCP or MPTCP protocol. For more information, see the respective `manual pages` [8–10].

[3]The top ten websites as of November 2016 were obtained from SimilarWeb [7]. The ranking was adjusted to avoid multiple entries of the same website (e.g. `google.com`, `google.co.uk`, `google.com.br`). Additionally, `amazon.com` uses techniques that actively prevent the automatic retrieval of the website, so this entry was removed from the list.

[4]`wget` was configured to "mirror" the websites, i.e. images and scripts were included in the download, similar to the behavior of a web browser.

according to external circumstances, but is usually equal to the system default route. Introducing packet loss on the default route thus leads to the loss of SYN segments, which introduces delays until the client retransmits the lost segment. This delay is referred to as *Retransmission Timeout (RTO)*; in most cases, it is a multitude of the latency of the initial path. With increasing packet loss it is more likely that multiple SYNs in a row are lost, eventually triggering the Linux *Exponential Back-Off Mechanism*. After multiple failed connection attempts the TCP stack will assume that the target host is overloaded and increase the SYN retransmission interval in order to avoid further stress on the server. A substantial increase in the delay until a connection is successfully established can thus serve as a reliable indicator if a packet loss has occurred. Figure 8 reflects these increasing timeouts intervals with classic MPTCP: The avg. loading time rise in multiple steps according to the number of lost SYN requests and the retransmission times chosen by the MPTCP stack. Also the overall effect of the exponential backoff mechanism becomes obvious when examining the avg. loading times, which increase exponentially with higher values of packet loss. If the initial path does not provide connectivity at all, MPTCP will not be able to establish new connections. This is the reason why no data is plotted for classic MPTCP and 100% packet loss.

As MPTCP RobE does not share the limitation of using only a single (possibly defective) path for connection establishment, a systematic increase of loading times cannot be found. Instead, the average loading time remains nearly constant under arbitrary conditions of the initial path.

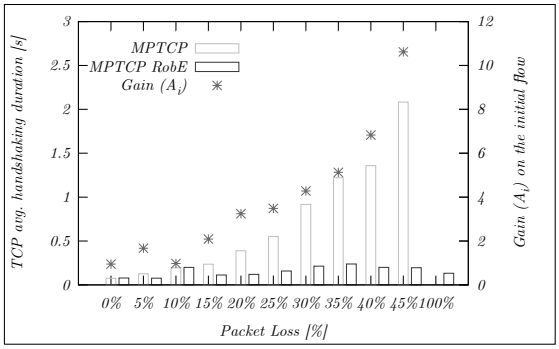

Figure 8: Average handshaking time for different packet loss on the initial path

4.2 Results of Experiment 2: Real-world evaluation

The real-world test expands the latency experiment to a practical example. Websites consist of many different elements which may be distributed over distinct hosts and even locations; retrieving a website thus requires creating many connections, transmitting small amounts of data and closing the connection again. As most web servers are not yet MPTCP capable, a proxy setup according to Figure 7 is necessary to use aggregation of multiple paths when retrieving public websites. The high-latency link was chosen as the default route; MPTCP thus receives additional latency on every new connection made. In contrast, MPTCP RobE can use the quickest path for connection establishment, which leads to loading time improvements between 50% and 100% in most cases, as shown in Figure 9.

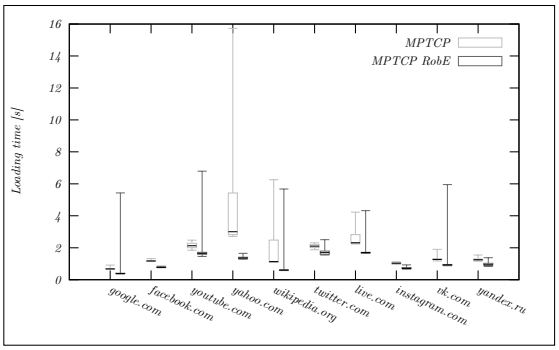

Figure 9: Comparison of website loading times

5 CONCLUSION AND FUTURE WORK

The MPTCP standard is in a phase of permanent innovation and improvement. The idea of aggregation of different flows and the resulting resiliency against link failures offers benefits for data throughput and a reliable user experience. In the present paper, a major flaw of the current MPTCP standard was examined: if a default path is impacted by packet loss or latency, new MPTCP connections will be affected, although there might be multiple paths that do not share this point of failure.

MPTCP RobE increases reliability and may improve end-to-end latency by using all available paths for establishing new connections. These benefits were verified in two experiments, which were conducted in a laboratory and a real-world environment. MPTCP RobE was able to eliminate the latency spikes introduced by increasing amounts of packet loss on the default path; additionally, the path providing the lowest latency determined the overall connection latency. In a real-world setup, MPTCP RobE was able to accelerate the average loading times of the ten most popular websites, while drastically reducing latency spikes due to network issues along the path to the target host. MPTCP RobE is currently implemented as a first prototype, which is why there are occasional latency spikes which do not seem to correlate to external factors.

In a future version of MPTCP RobE there might be an extension which enables the reliable connection establishment feature without explicit support from the peer site; the MPTCP RobE client could establish connections on all paths, and as soon as it recognizes that the peer does not support the extension, it might react accordingly and break down redundant flows. In the current implementation of MPTCP RobE connections to classic TCP and MPTCP hosts are possible, yet the solution does not terminate superfluous paths.

Overall, the findings in these experiments are very promising. MPTCP RobE can protect MPTCP against network outages during connection establishment. As a result, it can improve the user experience in terms of both reliability and latency, which directly corresponds to shorter loading times of websites and other small, short-period data transfers. Combined with a functional fallback mechanism, it might be a valuable addition to the current MPTCP standard as defined in RFC 6824 [2].

REFERENCES

[1] Christoph Paasch. October 30, 2014. *Improving Multipath TCP*. Dissertation. Louvain School of Engineering, Louvain. http://inl.info.ucl.ac.be/system/files/phd-thesis_1.pdf
[2] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. 2013. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824. RFC Editor. http://www.rfc-editor.org/rfc/rfc6824.txt http://www.rfc-editor.org/rfc/rfc6824.txt.
[3] Internet Engineering Task Force. 1992-05. TCP Extensions for High Performance. http://tools.ietf.org/html/rfc1323
[4] Olivier Bonaventure. 2013. Decoupling TCP from IP with Multipath TCP. http://multipath-tcp.org/data/MultipathTCP-netsys.pdf
[5] Jon Postel. 1981. *Transmission Control Protocol*. STD 7. RFC Editor. http://www.rfc-editor.org/rfc/rfc793.txt http://www.rfc-editor.org/rfc/rfc793.txt.
[6] S. Barre et al. 2016. The Linux/MPTCP v0.90 Implementation. http://multipath-tcp.org/pmwiki.php?n=Main.Release90
[7] SimilarWeb Team. 2016. The top 10 websites as of November 2016. https://pro.similarweb.com/#/industry/topsites/All/999/1m?webSource=Total
[8] The Linux networking team. 2013. The accept() Linux manual page. https://linux.die.net/man/2/accept
[9] The Linux networking team. 2013. The connect() Linux manual page. http://linux.die.net/man/2/connect
[10] The Linux networking team. 2013. The socket() Linux manual page. http://linux.die.net/man/2/socket