h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt
- Fachbereich Informatik -

Design and Implementation of a Prototypical Multipath Extension for QUIC

Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt von
Manuel Kieweg - 740319

Referent: Prof. Dr. Martin Stiemerling
Koreferent: Prof. Dr. Stefan Rapp

Ausgabedatum: 3.11.2017
Abgabedatum: 2.2.2018

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Manuel Kieweg
Darmstadt, February 1, 2018

Abstract

Todays computing devices have evolved. They usually provide multiple networking interfaces that can be used simultaneously. However traditional transport protocols only have the ability to make use of one interface per connection. Changing a networking interface – eg whilst leaving the range of a WLAN and switching to mobile data on a smartphone – during a Transmission Control Protocol (TCP) connection causes a shutdown and a re establishment of the connection. Furthermore the internet's grown infrastructure makes it hard for new transport protocols to be established. This so called ossification can be mitigated by using traditional transport protocols like TCP or User Datagram Protocol (UDP) and extend them with a richer set of functionality.

During this work depicts approaches to cope with the constraints caused by the ossification as well as introducing several concepts to multipath transmission. At first a short introduction into the issues the grown structures of the internet is presented. It addresses the issues of today's structure of the internet and explains the troubles that occur whilst deploying new transport protocols. Thereafter as well Quick UDP Internet Connections (QUIC) as different established and experimental approaches to multipath transport are presented. This provides the technological basics for the rest of this work.

It is shown that it is feasible to extend the QUIC protocol with multipath capability. A prototypical concept was designed using QUIC as substrate for the multipath capable expansion QED. A prototypical implementation was developed and evaluated. It was found that QUIC is an easily extendible protocol which allows a comparatively straightforward enhancement. Furthermore this basic research raises new questions that can be addressed in further research.

Contents

1 Introduction

In order to understand the motivation and the need for QUIC's development one has to understand the development of the internet and the changed requirements to internet-based services. The internet is not a homogeneous entity, instead it consists of a large number of autonomous systems[1] that are interconnected. So its structure has grown over time and has thus changed its significance. It evolved from a research project providing best effort packet transport – which it technically still does – to the backbone of our economy and a core asset of our society. Internet speed progressed over the decades – as well as the users' expectations. Nowadays, if a user has to wait too long for his requested content to be delivered, it is to be assumed that the user becomes impatient and leaves the service [KS13, BCK13]. Constantly growing bandwidth [Nie18] solves part of the problem. However, said acceleration brought another problem to the surface: Round trip times cannot be sped up[2] since there is no faster way of transmitting information than the speed of light. This circumstance motivated Google to find a new approach for speeding up connection establishment. Since the delay caused by one round trip cannot feasibly be reduced, it seemed to be a promising approach to reduce the amount of round trips needed instead. Besides that it is cheaper to solve latency related issues using a software solution than changing the hardware infrastructure.

Goggle announced the development of QUIC in 2013 [Ros18] after they had tried to challenge limitations in Hypertext Transfer Protocol (HTTP) with the TCP based SPDY protocol. SPDY was used as a foundation for the development of the HTTP/2 protocol and was designed to overcome certain flaws of HTTP/1.x. Hence SPDY uses multiple transmission streams that are multiplexed over one connection, for instance. This approach was chosen to avoid the establishment of multiple connections for one page request. Said practice was used by HTTP/1.x to load multiple page elements

[1]The term *autonomous system* describes a network under the governance of one entity.
[2]Assuming an ideal transmission medium.

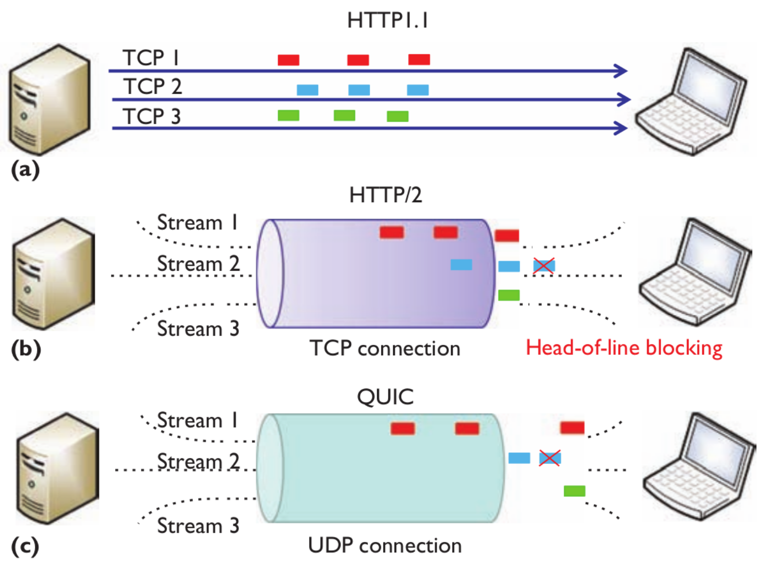

Figure 1.1: The HOL Behaviour of HTTP over TCP and QUIC [CLL+17]

parallel and thus decrease the page load time. However, this approach did not take a fair bandwidth allocation into account. SPDY based protocols counteract this behaviour by using multiple streams on a single connection [BPT15].

Furthermore, TCP-based application protocols face the problem of Head-of-Line blocking (HoL)[3]. For SPDY or HTTP/2 connections this phenomenon causes that a retransmission in one stream affects all streams of this connection. This is where QUIC comes into play with using UDP instead of TCP and a stream level flow control. If a retransmission occurs on one stream, this stream will still be affected by HoL blocking, the other streams, however, will not be affected. This behaviour leads to QUIC performing well in unstable and high latency environments [CMTH17]. It must be noted at this point that QUIC itself is a transport protocol whereas SPDY and the HTTP family are application layer protocols. However, QUIC has been developed with HTTP/2 as payload in mind [Bis17]. A comparison between the HoL behaviour of HTTP over TCP and QUIC is shown in figure 1.1. Remarkably HTTP/2 is more vulnerable to HoL that version 1 because it makes more information dependent on one TCP connection.

At the latest since the Snowden leaks the encryption of communication has had

[3]Head-of-Line blocking describes a performance issue in computer networking in general, as well as a problem in TCP. Considering TCP it occurs if packets have to be retransmitted. Other packets in the send buffer have to await this retransmission which causes a delay.

a much higher importance in the development of communication protocols. This makes QUIC's encryption by design another key feature. Already during the development of SPDY and HTTP/2 there has been a bigger focus on encryption, but it still is not mandatory, even though all major browser developers have stated that their implementations would only work with servers using encryption [Not18]. QUIC, however, has no unencrypted mode of operation. The encryption is realised with Transport Layer Security (TLS) 1.3 [TT17a], which needs less round trips for connection establishment than previous versions. This is one of the mechanisms QUIC uses for faster connection establishment.

Another important aspect to consider in the development of new transport protocols is the grown structure of the internet. Over the years, various technologies called middleboxes have been introduced to the internet, manipulating or evaluating network traffic[4]. Since those middleboxes have usually been designed for a specific application, they can often only handle a limited number of transport protocols. As a result, new transport protocols are hardly able to establish themselves throughout the entire internet but can only be used in homogeneously managed network areas. A prominent example for this is the Stream Control Transmission Protocol (SCTP), which is widely used for signaling in cellular networks, but cannot pass through the middleboxes of the internet. SCTP is introduced in more detail in section 3.2. UDP and TCP are largely excluded from this so-called ossification of the internet [MP15]. Both protocols have become so popular that they are understood by virtually every middlebox. This has resulted in new transport protocols based on TCP or UDP – Multipath TCP or QUIC, for example. This circumstance has the advantage that a middlebox, which cannot handle the "new" transport protocol, sees a UDP or TCP packet. The protocol-specific features only appear as payload.

[4]For instance web proxies, firewalls, network address translators

2 Problem Description

This work addresses two problems of common transport protocols: The utilisation of one single path even if several paths would be available and the difficulties that exist in the roll out of new transport protocols due to the internet's ossification.

2.1 Single Path Usage

Most of the world's internet traffic is transmitted via TCP[1] [LIJM$^+$10] which is a single path transport protocol. This has several shortcomings. For instance it provides no mechanism for bandwidth aggregation in case an endpoint is multihomed or addressed. Another flaw of TCP's single path behaviour becomes obvious if the path that carries a TCP connection gets lost. The reconnection procedure – which is basically the establishment of a new connection – is comparatively expensive due to TCP's 3-way handshake and possibly a following TLS handshake. This behaviour raises problems, since it reduces the Quality of Experience (QoE) of users accessing a single path only service. For example the usage of a given webservice that requires a long lasting connection – assume a download via TCP for the sake of simplicity – with a multihomed mobile device like a smartphone would fail if the user leaves the range of his WLAN even though his phone still has a wireless connection to the mobile network.

One solution for this problem are transport protocols that utilise more than one path for their connections. A variety of approaches is presented in chapter 3. Chapters 4 and 5 present a newly developed prototypical approach to add multipath capability

[1]And UDP which, however, is excluded from this contemplation. Since UDP is connectionless its scope is limited to the one datagram it transmits at a given moment. Any additional semantics for consecutive datagrams have to be defined on a higher OSI layer

to QUIC.

2.2 Ossification

Another problem in protocol design is the ossification of today's internet [FTK17, Han06]. This phenomenon prevents the establishment of new technologies on the internet. The dissemination of IPv6 is a good example. Since its standardisation in 1998 [DH98] it still has not been rolled out extensively [Soc18]. Regarding transport protocols, it is almost impossible to use a transport protocol that is neither TCP nor UDP in the open internet due to middleboxes being in the way [Han06]. This presents major challenges to designers of transport protocols. The issue outlined in Section 2.1 illustrates the implications. Changing requirements to transport protocols cannot be fulfilled by simply designing a new transport protocol like TCP did when the capabilities of UDP became insufficient.

Even though this work cannot present a solution for this problem it shows that using UDP or TCP as a substrate to new transport protocols can mitigate this issue.

3 Backgound

The following chapter presents the technical foundations this work is based on. It contains a QUIC overview, an introduction to multipath data transmission in general as well as a brief insight into several existing concepts of multipath transport on different OSI layers[1].

3.1 The QUIC Architecture

QUIC is a UDP-based transport protocol. This design was chosen to circumvent middleboxes and to be able to establish the 0-RTT behaviour that QUIC incorporates. However, the developers of QUIC wanted their protocol to be reliable, so they added TCP-like flow and congestion control on top of UDP. The result is a resilient and fast transport protocol. Most research investigating QUIC comes to the conclusion that QUIC performs best in unstable networking environments. [Sri17, CMTH17, BG16, MKM16]

This is the result of one of the key aspects of QUIC – the UDP base. QUIC has a similar architecture to HTTP/2. The relationship between a client and a server is one connection. However, there can be several streams of data that are all multiplexed over this connection. The advantage of using UDP instead of TCP is that flow and congestion control can be implemented on stream level. As a result there is no interference with other streams if it comes to retransmissions on one stream of the connection. The Internet Engineering Task Force (IETF) draft proposes the a mechanism TCP also uses; New Reno [HFGN12, IS18]. Furthermore the UDP

[1]As defined in ITU-T X.200 (07/1994). The use of 'OSI layers' in the following refers to this standard.

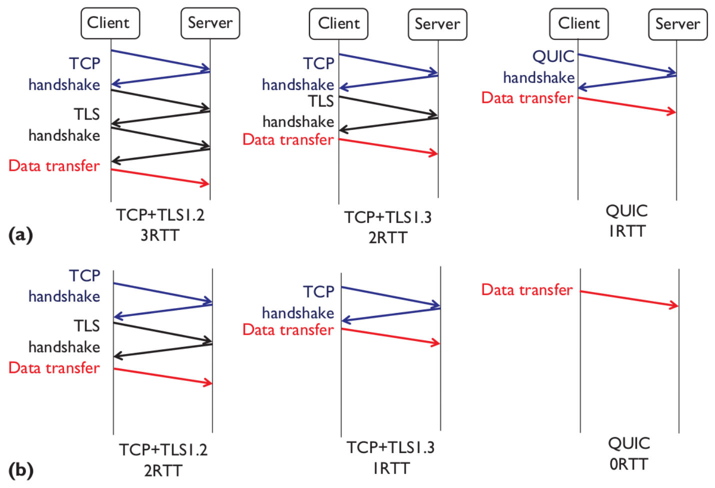

Figure 3.1: The connection establishment of TCP compared to QUIC [CLL+17]

base enables QUIC to reduce the needed amount of round trips for connection establishment. If the server is already known by the client, QUIC is able to establish a connection without any previous handshake messages. If the server is not previously known, one round trip is sufficient. In contrast, a TCP connection with following TLS handshake needs two round trips to be established [IT17]. Figure 3.1 underlines the advantages QUIC brings to connection establishment.

The second key aspect of QUIC is the strict always-on encryption. The IETF draft is proposing the usage of TLS 1.3. The utilisation of TLS 1.3 gives QUIC the ability to shorten the time needed to establish the connection. In contrast to previous versions TLS 1.3 speeds up the connection establishment by needing just one round trip instead of two [Res18a]. QUIC does not only encrypt the payload of a packet, it encrypts – or at least signs – the header of the packets. Following this design decision manufacturers of middleware components have raised concerns that their technologies would not be usable anymore. Because of the authenticated headers and the encryption by default of QUIC packets traffic shaping or traffic analysis is much less practical [TT17b].

QUIC's classification in the classical network stack is not as straightforward as in other transport protocols. It uses another transport protocol – UDP – for the actual transport and incorporates another transport protocol – TLS – for encryption. So it would on the one hand be fair to argue that QUIC is an application protocol. Furthermore, its focus on HTTP/2 delivery could be used as another argument in

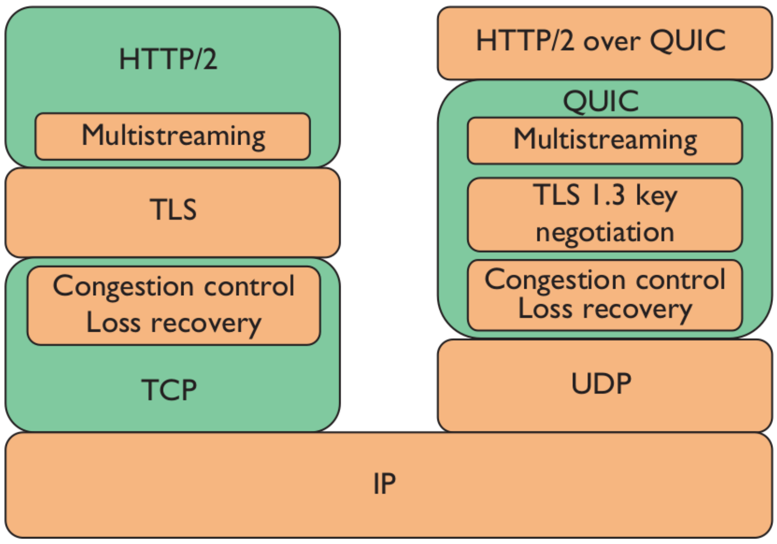

Figure 3.2: The HTTP/2 stack compared to the QUIC stack [CLL+17]

favour of considering QUIC an application protocol. On the other hand, the fact that IETF's transport area is standardising QUIC is a viable argument supporting the theory of QUIC being a transport protocol. Additionally, considering its design and the fact that QUIC has been designed for HTTP/2 transport but will provide the same interfaces as a classic transport protocol, it seems quite reasonable to classify QUIC as a transport protocol [IT17]. Those reflections on QUIC's protocol classifications are further illustrated in the stack graphic (figure 3.2).

3.1.1 QUIC 101

To an application – in QUIC's case mostly HTTP/2 – QUIC appears to be a transport protocol just like any other. Internally it has some very unique features, though. Beginning with the abandonment of the classical 5-tuple[2] for connection identification in favour of a connection ID. This is why every QUIC connection gets its own identification number. This allows QUIC connections to be alive over network changes – or over multiple paths. This feature is the link to the multipath capability of QUIC. Since the connection ID instead of the classical 5-tuple identifies the connection, it does not matter if the connection uses several disjoint IP addresses and therefore one endpoint can use more than one IP address to use the connection.

[2]Source IP and port, destination IP and port, protocol

As mentioned beforehand, QUIC has adopted the stream concept of HTTP/2, thus the multiplexing of several streams over one connection. The usage of streams has been intended to parallelise the access to resources – similar to the parallel establishment of multiple TCP connections for HTTP/1.x transmissions. Every connection has at least two streams: Stream 0 is reserved for signalling, whereas any other stream is used for data transmission. Streams are identified by an unsigned 62-bit integer value. The least significant bit of this value indicates the origin of the stream. Streams with an even numbered identifier – where the LSB is 0 – are initiated by the client, whereas streams with an uneven identifier are initiated by the server. All numeric values are encoded in network byte order, which is big-endian [IT17].

3.2 Multipath Transmission

Network connections are usually limited to one path. This has several disadvantages. A single path connection, for instance, does neither provide redundancy, nor bandwidth aggregation. Multipath transport concepts are challenging these disadvantages. The requirement for using multipath capable protocols is that at least one of the devices' communication has more than one networking interface[3]. They can provide redundancy and to a certain extend load balancing. Multipath transport achieves redundancy by the usage of more than one path. The paths do not necessarily have to be actively used. As an example SCTP uses only one path for the actual transmission of data. The other paths are inactive backup paths. [Ste07] The usage of several parallel paths is part of an extension to SCTP. But many modern multipath protocols have been developed with load balancing in mind. Multipath TCP (MPTCP) for example is one of those protocols. [FRHB13]

The initial design of QUIC had been intended to include multipath capability by default. This changed when the IETF started to standardise QUIC. In the first draft proposed by Google, multipath was planned [HISW16], whereas multipath capability is not even mentioned in the current IETF draft [IT17]. However, the latest IETF roadmap states that a multipath extension document is planned for

[3]This view only considers the transport layer. On the network layer mechanism like Equal-cost mutipath routing (ECMP) can make use of multiple paths themselves

May 2019. Given the delay the standardisation already has, this schedule has to be treated with reservation. There is research for QUIC multipath extensions, though [DCB17, Hui17] which will be discussed later in this work.

QUIC is highly interesting for multipath operations, though. Since most mobile devices have more than one networking interface and QUIC has its biggest advantages on mobile connections [CMTH17], the idea of a multipath adoption is obvious at this point. Advantages could be less issues caused by a cell handover, a better utilisation of the total device bandwidth and redundancy of the connection.

The term multipath transmission describes concepts to transport network traffic over multiple network paths. Multipath transmission is not set to a certain OSI layer. There are concepts for a multipath extension in IPv4 and the ECMP on layer 3, on layer 4 some of the well known protocols are Multipath TCP (MPTCP) or the SCTP.

3.2.1 Equal-cost multipath routing

ECMP is a routing approach to handle multiple routes that tie for the best path. Instead of being a routing protocol ECMP is an approach to deal with the possible hindrances multipath routing comes along with. It is used by different routing protocols like Intermediate System to Intermediate System (IS-IS) or Open Shortest Path First (OSPF).

One of those hindrances is the detection of the Maximum Transmission Unit (MTU). The usage of MTU discovery[4] loses parts of its advantage when the MTU has to be recalculated on a packet-by-packet basis. [TH00] This is an issue since multiple paths can have multiple MTUs.

Another concern are variable latencies on multiple paths. This comes especially into play when order sensitive transport protocols like TCP are used on top of the multipath capable routing. Since TCP is very vulnerable regarding out of order packet arrival, the use of multiple paths with significantly different latency can

[4]MTU discovery is an approach described in [MD90] for IPv4 and in [MDMH17] for IPv6. It determines the smallest MTU on a network path.

trigger TCPs fast retransmit mode due to suspected packet loss [APS99]. This behaviour consumes unnecessary bandwidth and could lead to a further setback concerning the connection quality. [TH00]

ECMP challenges these concerns making use of specific implementations like the one analysed in [Hop00]. Most of ECMP implementations try to route packets belonging to a UDP or TCP stream over the same path. In order to do so flow related information is hashed and used as an identifier for consecutive transmission.

Since ECMP operates on OSI layer 3 it cannot provide end to end multipath capability. This circumstance leads to the conclusion that there is never one consistent multipath state like MPTCP can provide. ECMP furthermore neither has a global scheduling approach nor is it capable of utilising more than one interface an endpoint has. This results in the fact that ECMP is used in routing protocols. Since routing is mostly transparent to the endpoints, the multipath capability of the underlying network is as well. Therefore ECMP-based protocols are neither able to provide a roaming handover between multiple networks for the client nor can they be used to aggregate multiple endpoint links to increase bandwidth. Nevertheless, since routing happens below OSI layer 4 protocols and popular routing protocols like IS-IS and OSPF are using the concept of ECMP, it plays an important role in Internet Protocol (IP) based transmissions. One could define every transmission via ECMP incorporating protocols as a multipath transmission. Since this work has its focus on an OSI-layer 4 protocol extension this issue needs to be addressed in further research.

3.2.2 Multipath IP Transmission

Multipath IP (MPIP) is a conceptual multipath capable IP stack. It was described in [STZ$^+$17]. Just like ECMP it operates on OSI layer 3 and extends IP with multipath capability. The authors of MPIP were influenced by the work of [Lei13] about MPTCP but are representing the opinion that using IP for multipath transmission incorporates several benefits over the TCP usage of MPTCP. They are pointing out that multipath transmissions on OSI layer 3 would be more application independent and thus a reduction of both overhead and the independence of the multipath capability of the actually used transport protocol.

Source Node ID	Session ID	Local IP Address List	CM Flags
Path ID	Feedback Path ID	Packet Timestamp	Path Delay

Table 3.1: MPIP Control Message Structure [STZ+17]

MPIP is divided into six major components: *Signalling Channel, Handshake, Session Management, Path Management, MPIP Routing, and Network Address Translation (NAT) Traversal.*

Signalling Channel

Since IP is a connectionless protocol the MPIP extension needs signalling to establish its multipath capability. For this purpose a signalling mechanism is used. This signalling does not happen on a separate data stream but its information is added to each MPIP packet that is sent. This so called *Control Message* – shown in table 3.1 – contains the necessary information to establish and maintain multipath capability. Since it is added to the users payload it expands the packet size by 25 bytes, which is a comparatively small overhead considering packet sizes of around 1000 bytes. an identifier for the source node which is in most cases the endpoints' Media Access Control (MAC_1) address, identifiers for path and session, a timestamp, path characteristics like the delay and a dedicated feedback path.

Handshake

Because MPIP needs backwards compatibility to IP, each endpoint keeps track of its communication partner's multipath capability using a table . This table contains source IP and port, as well as the multipath capability and a query count that indicates how stable the multipath capability is. If the initial MPIP negotiation succeeded, each endpoint creates a corresponding entry in its mapping table, which contains the opposite endpoint's IP address and port as well as a Node ID. The according three tuple of each subsequent packet is added to that table. An example is shown in table 3.2.

Node ID	Node IP	Node Port
ID_1	IP_1	P_1
ID_1	IP_2	P_2
ID_2	IP_3	P_3
ID_2	IP_4	P_4

Table 3.2: MPIP Mapping Table [STZ+17]

Dst. Node ID	Session ID	Src. IP	Src. Port	Dst. IP	Dst. Port	Protocol	Next Seq#	Update Time
ID_1	SID_1	SIP_1	SP_1	DIP_1	DP_1	TCP	S_1	T_1
ID_1	SID_2	SIP_1	SP_2	DIP_1	DP_2	UDP	N/A	T_2
ID_2	SID_1	SIP_2	SP_3	DIP_2	DP_3	TCP	S_1	T_3
ID_2	SID_2	SIP_2	SP_4	DIP_2	DP_4	UDP	N/A	T_4

Table 3.3: MPIP Session Information Table [STZ+17]

Session Management

The added multipath capability in comparison to IP creates the need for additional session management. Whereas IP is a connection- and stateless protocol, MPIP needs to establish sessions and maintain state. This state is held in a session information table (An example is shown in table 3.3). A distinct session is identified by its Session ID in connection with a Node ID. This behaviour allows MPIP to be resilient to NAT caused IP address changes. Furthermore, the *Update Time* is used to remove inactive sessions from the table if the value exceeds a defined threshold.

Path Management

The core of a multipath enabled transfer mechanism like MPIP is the management of different paths. Every endpoint propagates its local IP addresses in the Control Message (CM). Using these announced addresses is only feasible if all of these addresses are public IP addresses[5]. To circumvent these NAT based restrictions, [STZ+17] are leveraging the combination of IP and port to identify a distinct path. If a packet with a known *Session ID* and *Path ID* arrives from a different IP than advertised,

[5]Opposing to [RMK+96] compliant IP addresses which cannot be routed in the public internet.

the *Session ID* gets mapped to the newly learned address. Thus NAT traversal is possible without using Session Traversal Utilities for NAT (STUN) or Traversal Using Relays around NAT (TURN). Furthermore, the MPIP path management provides a constant path monitoring and a dynamic path addition or removal to the protocol's runtime. The monitoring measures the one way delay of a path using timestamps in the CM. The measured One Way Delay (OWD) is used to determine a path's weight. The weight influences the routing of MPIP packets. As mentioned before, MPIP supports the dynamic addition and deletion of paths. It utilises the *Local IP* parameter sent in the CM, as well as the *Flags_IP_Change* flag to indicate the change of an address or state of an interface.

Routing

MPIP provides different routing modes for different use cases. It has an *All-Paths Mode* and a *User-Defined Mode.* The *All-Paths Mode* makes use of all available paths under the consideration of the paths' weight. Since the transport protocols running on top of MPIP have their own flow and congestion control it would be redundant if not counterproductive to have it on OSI layer 3 as well. The selection of the paths is based on a probabilistic algorithm shown in the following equation:

$$P(k) = \frac{W_{\mathrm{k}}}{\sum_{i=1}^{N} W_{\mathrm{i}}} \tag{3.1}$$

The probability of using a given path (k) is determined by the weight (W). in their prototype [STZ+17] use the one way delay of packets to calculate a dynamic weight for each path in real time. In addition to the *Real-Time Path Delay D* which is calculated using timestamps three values are used for calculation:

- Minimum Path Delay: $\mathrm{D_{min}} = \mathrm{min}\{\mathrm{D_{min}}, \mathrm{D}\}$
- Real-Time Queuing Delay: $\mathrm{Q} = \mathrm{D} - \mathrm{D_{min}}$
- Maximum Queuing Delay: $\mathrm{Q_{max}} = \mathrm{max}\{\mathrm{Q_{max}}, \mathrm{Q}\}$

The other option for routing is the *User Defined Mode.* This mode is meant for applications whose main need is not throughput but a low latency. For example, a live video stream just needs its video bandwidth as throughput whereas the latency must be low to provide a proper QoE. To adapt to the different needs of certain applications MPIP provides special modes of operation:

- selected-paths
- single-path
- protected-path

which are used with the two routing modes throughput-first ($\mathrm{T_f}$) and responsiveness-first ($\mathrm{R_f}$). Whereas in the $\mathrm{T_f}$ mode all available paths are used in all-paths mode, the $\mathrm{R_f}$ mode uses only the path with the lowest latency in single-path mode.

Conclusion

MPIP is an interesting approach to multipath transmission. It provides the ability of multipath transmission regardless of the overlaying transport protocol and application. Thus it may reduce multipath management overhead compared to concurrent multipath enabled applications, which are running separately in user space. But since multipath transport protocols running in the kernel scope only have one multipath management, this reduction of overhead is merely theoretical.

On the other hand MPIP brings several disadvantages into play. First of all, there is no proper signing of address advertisements so that adversaries can propagate their own addresses. Doing so an adversary could easily exploit this behaviour to engage a Man in the Middle (MitM) attack and reroute at least parts of the traffic over his network.

Due to the lack of encryption the MPIP concept exposes all endpoints' IP addresses. This leads to privacy implications because not only the active but even potentially inactive addresses are transmitted.

Furthermore, MPIP adds a lot of complexity to the lean IP. Even though MPIP is backwards compatible to IP, it exceeds the boundaries of OSI layer 3 and establishes kind of a *super protocol* that incorporates some elements normally left to a transport protocol like session handling.

3.2.3 Stream Control Transmission Protocol

The SCTP is a multipath enabled transport protocol. It was standardised by the IETF in 2000 [Ste07]. SCTP is seldomly seen in the open internet but plays a huge role in telecommunication networks. It is part of the SIGTRAN protocol family [ORG+99], a protocol family that handles telecommunications signalling. SCTP is designed to transport Public Switched Telephone Network (PSTN) over IP networks but is capable of a broader scope of tasks. Technically it is perfectly capable of being an internet transport protocol but the internet's ossification prevents deployment on a larger scale.

Its core features as an internet transport protocol are *multipath capability*, *avoidance of HoL* and *partial reliability* [SRX+04]. Its prominent use cases apart from SIGTRAN are the distributed server pooling protocol RSerPool [LOTD08], the successor of the Radius protocol DIAMETER [FALZ12], the Netflow[6] extension IPFIX [Cla08] and WebRTC [JLT15].

Terminology

Almost every transport protocol uses its own terminology. SCTP makes no exception to this. The following terminology as defined in [Ste07] is used throughout the rest of this section.

Association An *association* describes a connection between two endpoints.
Chunk A *chunk* describes the smallest entity that is transmitted over the wire. The QUIC equivalent would be a frame.
Path A *path* is a logical connection between two endpoints.
Stream A *stream* can be described as an unidirectional set of *chunks*. Streams provide in-order delivery of chunks.

[6]Netflow is a technology used for network measurements [Cla04].

Multipath Capability

Even though SCTP is a multipath capable transport protocol, its multipath capability was at first limited to fail over tolerance [Ste07]. SCTP would create as many associations as negotiated during connection establishment but use only the main path for data transfer. The rest were used as stand-by paths. An extension to SCTP called Concurrent Multipath Transfer Extension to SCTP (CMT-SCTP) [Iye06] provides fully concurrent multipath capability.

The establishment of an SCTP association uses a 4-way handshake (figure 3.3). This behaviour makes SCTP resistant to `SYN` flooding[7] attacks. After the `INIT-ACK` reply of the receiving endpoint, the opening endpoint needs to send a valid Message Authentication Code (MAC_2) with its `COOKIE-ECHO` before resources are allocated. That mitigates the adversary's strategy to flood the attacked endpoint with `SYN` messages to use up the attacked endpoints resources.

The address propagation of the initiating host takes place in the `INIT` chunk (figure 3.4). [Ste07] contains three cases of address parameters transmitted in the `INIT` chunk.

No Address Parameters If an `INIT` chunk contains no address parameters the sender's IP address is used as the only destination address for that endpoint.
Host Name If the `INIT` chunk contains a hostname, the receiving endpoint resolves all IP addresses associated with that hostname and uses them as destination addresses.
Only IP Addresses In case of an `INIT` chunk containing only plain IP[8] addresses the receiver must use all of these addresses to derive corresponding transport addresses.

The receiving endpoint applies the same mechanism for address propagation in its `INIT-ACK` chunk. This chunk type has the same structure as the `ACK` chunk shown in

[7] *SYN flooding* describes an attack that exploits a conceptual flaw in the TCP handshakes. An adversary can flood an endpoint with `SYN` messages. The endpoint will reply with an `SYN-ACK`, and will thus accordingly have a half-open connection. If the adversary does not send the final `ACK`, the attacked endpoint stays in this half-open state until the connection times out. Since this behaviour allocates a lot of resources, it can be exploited as a Denial of Service (DoS) attack [Edd07].
[8] IPv4 and/or IPv6

```
+-----------------------------------------------------------------------+
|                                                                       |
| Endpoint A                                           Endpoint Z       |
| {app sets association with Z}                                         |
| (build TCB)                                                           |
| INIT [I+Tag=Tag_A                                                     |
|       & other info]  +-----+\                                         |
| (Start T1+init timer)       \                                         |
| (Enter COOKIE+WAIT state)    \+--^ (compose temp TCB and Cookie_Z)   |
|                                 /++ INIT ACK [Veri Tag=Tag_A,         |
|                                /             I+Tag=Tag_Z,            |
| (Cancel T1+init timer) <-----+/             Cookie_Z, & other info]  |
|                                       (destroy temp TCB)              |
| COOKIE ECHO [Cookie_Z] +-----+\                                       |
| (Start T1+init timer)         \                                       |
| (Enter COOKIE+ECHOED state)    \+--> (build TCB enter ESTABLISHED    |
|                                        state)                         |
|                                /+--+ COOKIE+ACK                       |
|                               /                                       |
| (Cancel T1+init timer, <----+/                                        |
|  Enter ESTABLISHED state)                                             |
+-----------------------------------------------------------------------+
```

Figure 3.3: SCTP Handshake [Ste07]

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 1    |  Chunk Flags  |      Chunk Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Initiate Tag                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Advertised Receiver Window Credit (a_rwnd)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Number of Outbound Streams   |  Number of Inbound Streams    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Initial TSN                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/              Optional/Variable-Length Parameters              /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.4: INIT Chunk [Ste07]

figure 3.4, apart from the different type. The SCTP standard does not include any mechanism to dynamically add or delete endpoint addresses. However, this feature is added by the *AddIP* extension [SXT+07] which adds two new chunk types, the `ASCONF` and the `ASCONF-ACK` chunks, and new parameters including the addition or deletion of an address as well as the definition of a new standard path. To prevent malicious address propagations the `ASCONF` and `ASCONF-ACK` chunks have to be signed as specified in [TSLR07], or they must be silently discarded [SXT+07] to prevent information leakage to a potential adversary. In their `INIT` and `INIT-ACK` chunks server and client can define the number of inbound and outbound streams within the boundaries of one and the physical maximum. The client makes the first offer in its `INIT` chunk, the server can accept or change this offer regarding its limitations. The server should accept the clients number of inbound streams or the client can cancel the establishment of the association [Ste07].

Once initiated, SCTP implementations compliant to the standard [Ste07] are using the primary path[9] to start with their data transmission. Its flow and congestion control mechanisms are very similar to TCP [Ste07]. The backup paths are being monitored with heartbeat chunks during the lifetime of the association [Ste07].

[9]The IETF standard does not specify how this *primary path* is chosen.

The two most significant differences towards TCP – albeit the 4-way handshake – are explained briefly in the following.

Partial Reliability Partial reliability is a concept that allows the sender of a message to discard a message under certain circumstances. The partial reliability extension to SCTP is standardised in [SRX+04]. The extension is backwards compatible to plain SCTP, and its usage is negotiated in the `INIT` and `INIT-ACK` chunks. The added `FORWARD TSN` chunk type is used to forward the sequence number. This dismisses all chunks that are behind the new sequence number.

Head-of-Line Blocking SCTP is less susceptible to the problem of Head-of-Line blocking than TCP, due to its stream oriented architecture [Ste07]. SCTP does provide in order delivery of chunks but only on a stream level. If a transmission delay on a given stream exists, this stream blocks. Other streams, however, are not affected. This behaviour reduces the impact of HoL.

CMT-SCTP

Even though SCTP uses multiple paths in an association, it does not support actual parallel multipath transmissions. Since SCTP's primary design goals were reliability and fault tolerance concurrent multipath transmissions were not in the design scope [ABD+18]. However, the design of SCTP allowed an extension that made it fully multipath capable [Iye06]. During the development of the extension (CMT-SCTP) the authors encountered several hindrances regarding fast retransmissions, conservative congestion window[10], and growth leading to an increased ACK traffic [IAS06]. Since the initial CMT-SCTP research in 2006 [ABD+18] as well as the AddIP extension [SXT+07] are part of the reference implementation of SCTP [Tü18].

[10]The congestion window is a congestion control element. It defines the amount of data the sender can send without receiving an ACK message [Ste07]

Conclusion

SCTP is a protocol initially designed to accomplish a very narrowly defined set of tasks. However, due to its robust design, several extensions were developed for SCTP to broaden its usage possibilities. The concurrent multipath extension defined in [Iye06] and the UDP encapsulation proposed in [TS13] qualify SCTP as a multipath transport protocol to be used in the internet. However, with the development and deployment of the very similarly featured MPTCP by major companies and (MP)QUIC at the starting block, it is unlikely to see SCTP widely deployed in the internet.

3.2.4 Multipath TCP

Another multipath capable transport protocol is MPTCP. It is basically an extension to TCP and specified in [FRHB13]. The reason for using TCP as a basis for MPTCP is TCP's resilience towards middleboxes. Since the major amount of internet traffic (95% in 2010 [LIJM$^+$10]) is TCP or UDP those protocols are mainly unaffected by middleboxes which makes them a good protocol substrate. SCTP states a good example how hard is it to establish new transport protocols in the open internet, even though they might have a larger set of features than existing ones [TH14].

The most obvious approach – striping all TCP packets following a howsoever defined algorithm over the available paths – is not applicable due to middleboxes that most certainly will discard TCP packets with out of order sequence numbers [RPB$^+$12a]. To mitigate this behaviour MPTCP makes use of the TCP options to announce multipath capability and an abstraction layer using a separate set of sequence numbers to provide in order delivery for a multipath enabled flow. Packets on the wire will appear to middleboxes that are unaware of MPTCP as plain TCP packets. If said middleboxes do not follow the recommendation to ignore unknown options and strip the `MP_CAPABLE` option off a `SYN` packet instead, normal TCP communication is still possible [FRHB13]. The removal of the option transmitted in a `SYN/ACK` packet is more of a problem as the following subsections will show, among other things.

Connection Lifecycle

Since MPTCP is a connection oriented protocol there is a connection establishment and a connection tear down. The connection establishment is based on TCP's 3-way handshake and is depicted in figure 3.5. The most significant difference is the transmission of the MP_CAPABLE option in SYN and SYN/ACK messages. As mentioned earlier on the deletion of the MP_CAPABLE option carried in a SYN packet is not a real problem at all. It just prevents the establishment of a multipath connection while the TCP fallback still works. However, the deletion of the MP_CAPABLE option only out of the SYN/ACK packet causes a problem, because the client would assume the server is not multipath capable whereas the server would assume the client was multipath capable. An approach to circumvent this phenomenon would be to append the MP_CAPABLE option to the final ACK message as well. Raiciu et al. [RPB$^+$12a] used this approach. But because an ACK can get lost without affecting the connection establishment fatally, all subsequent data packets have to carry the MP_CAPABLE option until they are ACKed. If a server receives a data packet without MP_CAPABLE option it has to fall back to normal TCP behaviour, even if the client announced multipath capability in the first place [FRHB13].

The tear down of a MPTCP connection also differs from plain TCP. Plain TCP provides two modes of connection tear down: the RST and the FIN version [Pos81b]. RST indicates a connection tear down due to an error and causes an immediate termination of the connection. FIN works similar to the 3-way handshake TCP uses for its connection establishment and is used for a coordinated connection tear down.

Since MPTCP bundles several physical TCP connections to one logical connection the TCP terminology is not applicable to MPTCP. Because RST signals a connection termination due to an erroneous condition this kind of termination should only affect the physical path. Otherwise an error on one path would lead to the termination of the whole MPTCP connection, even if the remaining paths are still fully functional [RPB$^+$12b]. A little bit more subtle is the handling of FIN messages. A FIN message could either have the meaning that the transmitting path has finished its transmission or that the whole MPTCP connection has come to an end. For this reason MPTCP has its own FIN semantics. The FIN message only has the scope of a single path – which is also necessary for TCP compatibility – whereas

```
Host A                                   Host B
------------------------                       ----------
Address A1    Address A2                       Address B1
----------    ----------                       ----------
    |             |                                |
    |            SYN + MP_CAPABLE(Key-A)            |
    |--------------------------------------------->|
    |<---------------------------------------------|
    |          SYN/ACK + MP_CAPABLE(Key-B)         |
    |             |                                |
    |        ACK + MP_CAPABLE(Key-A, Key-B)        |
    |--------------------------------------------->|
    |             |                                |
    |             |   SYN + MP_JOIN(Token-B, R-A)  |
    |             |------------------------------->|
    |             |<-------------------------------|
    |             | SYN/ACK + MP_JOIN(HMAC-B, R-B) |
    |             |                                |
    |             |     ACK + MP_JOIN(HMAC-A)      |
    |             |------------------------------->|
    |             |<-------------------------------|
    |             |             ACK                |
HMAC-A = HMAC(Key=(Key-A+Key-B), Msg=(R-A+R-B))
HMAC-B = HMAC(Key=(Key-B+Key-A), Msg=(R-B+R-A))
```

Figure 3.5: MPTCP Handshake and Subflow Establishment [FRHB13]

the `close()` call on a MPTCP socket causes the transmission of the newly added `DATA_FIN` [FRHB13].

Subflow Handling

After a successfully initiated multipath capable connection MPTCP has to establish multiple new subflows[11] on the available paths. Since subflows are logically represented by TCP connections, every initiation of a new subflow requires a new handshake. Theoretically it would not be necessary to introduce the new subflow with a handshake but instead begin with immediate data transmission, as long as

[11]MPTCP's terminology distinguishes between the term *path* and the term *subflow*. The first describes the physical link between the two hosts identified by a 4-tuple of source and destination address and the port pairs, whereas the latter describes a logical connection that utilises a path and is similar to a TCP connection [FRHB13].

an identifier for the MPTCP connection is present. Due to common middlebox behaviour – dropping data packets without preceding handshake – this approach is not feasible for TCP-based multipath protocols [RPB+12b]. As this work shows in the following, UDP-based multipath protocols do not suffer from that constraint.

Adding new subflows to an existing MPTCP connection presents two challenges. The first one is the identification of connections without the classic 5-tuple[12], which again is an issue caused by middleboxes – more accurately by NAT, since it modifies this 5-tuple. The second challenge is to avoid malicious path announcements. Since MPTCP packets are unencrypted and lacking a proper source validation, the propagation of adversarial subflows needs to be mitigated.

To face this challenges MPTCP incorporates authentication mechanisms for connections and subflows. This authentication procedure happens during the handshake as shown in figure 3.5. Throughout the initial handshake both endpoints are transmitting their `Keys`, which are generated randomly. In the subflow handshake a `Token-B` – derived from `Key-B` using the result of a SHA-1 algorithm truncated to the 32 most significant bits – is used to identify the connection, followed by `HMAC`s for both endpoints consisting of the random keys (`R`) exchanged during the initial connection establishment [FRHB13].

Reliable Delivery

Another problem that occurs during the implementation of multipath transmissions is the reliability of transmissions. To provide a reliable delivery a transport protocol must provide proper flow and congestion control. TCP realises that with receive window mechanisms as the *sliding window* mechanism and the acknowledgment of received data [Pos81b]. However, due to multipath operations, both mechanisms are less efficient [RPB+12b]. Raiciu et al. proposed mitigations to this behaviour.

Even though the usage of TCP provides flow control to every subflow, TCP's flow control scheme cannot be applied to MPTCP. If every subflow maintains its own receive window deadlock can occur. A packet lost on one subflow and a full congestion window on another subflow, could prevent the retransmission of the lost packet.

[12]Source and destination address, port pair and protocol.

To circumvent this, Raiciu et al. propose a single receive window per connection instead of a window per subflow [RPB$^+$12b].

A similar problem occurres by packet acknowledgements. Because subflows most certainly have different OWD's acknowledgements can arrive out of order. This behaviour can lead to an overflow of the receive window and therefore disruptions in the transmission. This is caused because TCP announces its receive window only implicitly [Pos81b]. MPTCP can mitigate this using absolute window size propagations [RPB$^+$12b].

Another possible source of interference are middleboxes that are changing packet properties in favour of other protocols like the File Transfer Protocol (FTP). This can affect the Data Sequence Mapping (DSM), which relies on the position of bytes in the data stream. These positions can be changed by middleboxes – for example if the middlebox re-encodes IP addresses into ASCII, which inevitably changes their length. To mitigate this phenomenon MPTCP includes a checksum in the DSM, which on the downside negatively affects the performance. Therefore Raiciu et al. propose an option to disable the checksumming in controlled high-performance environments where the non-existence of such middleboxes is granted [RPB$^+$12b].

Conclusion

MPTCP has the appearance of a very mature transport protocol. This impression intensifies considering the productive use MPTCP encounters. The two most popular examples of the industrial use of MPTCP are Apple who use MPTCP in their mobile operating system iOS [App18] and the South Korean provider KT who uses MPTCP to accelerate their LTE infrastructure with WLAN [KT18]. This success is not surprising since MPTCP considers many issues the modern internet brings along. It is aware of destructive middlebox behaviour and addresses the issues different Round-trip Time (RTT)'s are bringing to multipath transport protocols.

Nevertheless, even MPTCP has some shortcomings. Despite the security considerations laid out in [BPG$^+$15], another troublemaker is MPTCP's TCP foundation. Not only does MPTCP incorporate TCP's quite long lasting TCP handshake, it requires this handshake also for all subflows, even though it is technically not necessary. It is much more a constraint due to middleboxes expecting this behaviour from TCP packets.

3.2.5 Multipath QUIC

Multipath QUIC (MPQUIC) is a QUIC extension designed and implemented by De Coninck et al. [CB17]. It makes use of certain QUIC design features. It leverages QUIC's connection identification mechanism and its UDP base for faster path establishment and middlebox resilience. Because of the multipath friendly design of QUIC it is much less complex than the similarly featured MPTCP [DCB17]. Probably because both protocols were designed in collaboration with the *ip networking lab*[13] at the Université catholique de Louvain.

Path Handling

After the connection establishment, which remained unchanged, both endpoints are allowed to create new paths they can use. If they have multiple interfaces they implicitly create a new path by using a new interface [CB17]. However, in case of NAT's presence in the path, De Coninck et al. recommend that only the endpoint without a NAT opens multiple paths. To identify paths affected by NAT MPQUIC endpoints propagate their paths with their 4-tuple to the opposite endpoint. This endpoint can detect affected paths by inconsistencies regarding the announced and the actually received 4-tuple.

New addresses are announced via the `ADD_ADDRESS` frame. The sender adds an `Address ID`, which is used for a subsequent deletion using the `DELETE_ADDRESS` frame. Due to the signed and encrypted transmission of the headers, QUIC provides address announcements which cannot be spoofed by adversaries. Since QUIC already implements mechanisms for handover during connection lifetime paths can be migrated over 4-tuples. This keeps a path active over an NAT rebinding for example [CB17].

Since QUIC frames are independent from their transmitting streams and the data

[13]https://inl.info.ucl.ac.be/ visited on January 29th 2018

offset is transmitted on frame level, MPQUIC requires no additional sequence number scope per path as MPTCP does. To cope with packet reordering – which can be expected due to different path RTT's – MPQUIC paths have their own packet number scope and also the ACK messages are expanded by a path identifier. Since the congestion control mechanisms that QUIC implements [IS17] would be unfair whilst used over multiple paths, a similar congestion control scheme as in MPTCP and per path congestion windows were proposed [CB17].

Packet and Frame Changes

The modifications towards multipath required changes in the packet format. The packets now indicate their endpoints' multipath capability and announce the path they were sent on. Without the multipath bit set MPQUIC only instantiates one path. Furthermore, the `ACK` frame was modified. It now transmits a `P` bit which announces a `Path ID` field also present.

For multipath signalling MPQUIC adds new frames to the protocol. The already mentioned `ADD_ADDRESS` and `DELETE_ADDRESS` frames, as well as the `PATHS` frame, which transmits all paths the sender considers active to the opposite endpoint. It contains the amount of active paths the sending endpoint maintains. In addition, it contains the `Path ID` of the sending path and its current local address embedded into a `Path Info` section [CB17].

3.2.6 Conclusion

This chapter introduced the features of different multipath capable transport protocols and extensions. There has been a lot of research done on many different kinds of aspects. SCTP, for example, is specifically designed for reliable transmissions in telecommunication networks. Therefore its multipath architecture is focused more on failover abilities than on actually concurrent multipath transmissions. MPTCP and MPQUIC on the other hand are focusing on every day internet transmissions whilst expanding TCP or QUIC.

MPTCP – still in an experimental state – has already been deployed on industry scale. It has already proved itself in everyday use and appears as a mature protocol that faces the hindrances present day's internet poses to new transport protocols well. MPQUIC on the other hand is still in a very early design phase and comparatively far away from deployment. However, De Coninck et al. have shown that it is possible to extend QUIC with multipath capability without adding much complexity. In its case there is no need to alter the initial connection handshake, which is necessary for MPTCP. The flexible design of QUIC allows it to easily extend it whilst still being backwards compatible. With QUIC on the IETF standards track and announced as *the next transport protocol for the internet* [For18], it is possible that TCP's time comes to an end. And just like QUIC has the potential to become the better TCP, MPQUIC looks promising to become the better MPTCP.

4 Concept

The conceptual multipath extension for QUIC has been developed based on the author's beforehand project work [Kie17]. Since the initial draft of the extension was purely theoretical the original concept showed a few design flaws that were improved during the development process. As shown in chapter 5, a working proof of concept was developed implementing this draft.

4.1 QED – QUIC with Enhanced Distribution

QUIC with Enhanced Distribution (QED) is an extension to the QUIC transport protocol which adds multipath capability to the UDP transport layer. Even though UDP is the extensions' foundation, QED is not UDP-like in its behaviour. Since it extends QUIC it is connection orientated and has to keep state as well. QED is fully IPv6 and IPv4 capable.

Design Assumptions In order to limit the design scope for the implementation, the following assumptions for the environment QED is used in were made:

- One or both endpoints of the connection have more than one networking interface or more than one IP address. Dual-stack endpoints[1] fulfil this requirement. Multiple addresses are necessary to establish multipath functionality.
- Every physical interface has at least one path to the other endpoint.

[1]Endpoints that have as well IPv4 as IPv6 connectivity

QED Concept This paragraph briefly describes the features added by QED and the scope of its extension for QUIC.

- It will behave the same as normal QUIC to an application or middlebox in the path that is not aware of QED.
- A QED connection is a QUIC connection from the start. After connection establishment more paths are negotiated thus multipath functionality is achieved.
- Multiple paths are identified by the existence of multiple endpoint addresses.

4.1.1 QUIC Extension

QED is an extension to QUIC. As such it is closely interwoven with QUIC but cannot make use of full QUIC functionality. QED is not able to perform 0-RTT connections with active multipath transport. The connection is initialised as an unipath QUIC connection. After connection establishment multipath signalling frames are sent over stream 0. Both endpoints advertise their routable addresses using the ADDR_MOD frame. Received addresses are stored and used to establish new subconnections using the newly learned addresses. Whereas MPTCP makes use of TCP options [FRHB13], the signalling of QED has to take place in a set of frames added to the protocol due to the lack of UDP options.

Frames and Frame Types

QED adds new frame types to QUIC. They are used for addition and removal of IP addresses and OWD measurement.

ADDR_MOD Frame QED uses `ADDR_MOD` frames to announce the deletion or the addition of an endpoint address. After connection establishment a packet with one `ADDR_MOD` frame per address is sent to the corresponding endpoint. In the case of an address change an `ADDR_MOD` frame is sent. The frame indicates the type of

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+-----------------------------------------------+
|TV             |
+---------------+-----------------------------------------------+
|                                                               |
|                       IP Address (32/128)                     |
|                                                               |
+---------------------------------------------------------------+
```

Figure 4.1: QED ADDR_MOD Frame

change, the IP version and the address affected by the change. Each frame can only indicate the change of a single address. If multiple addresses have been changed, the equivalent amount of frames has to be sent. (Figure 4.1). Its frame type is 0xD.

The fields are:

Type

- **0** indicates the deletion of an address
- **1** indicates the addition of an address

Version

- **0** indicates IPv4
- **1** indicates IPv6

Address The address affected by the occurring event.

OWD Frame The OWD frame is used by QED to determine the one-way delay. It contains a timestamp and the ID of the outgoing path. Its usage will be presented later on.

The fields are:

Path ID The ID of the measured Path

Time A timestamp in Unix nanosecond format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|                          Path ID (32)                         |
+---------------------------------------------------------------+
|                            Time (64)                          |
|                                                               |
+---------------------------------------------------------------+
```

Figure 4.2: QED OWD Frame

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|                          Path ID (32)                         |
+---------------------------------------------------------------+
|                            OWD (64)                           |
|                                                               |
+---------------------------------------------------------------+
```

Figure 4.3: QED OWD_ACK Frame

OWD_ACK Frame The OWD_ACK frame is used to send the OWD back to the initiating endpoint. It contains the delta of the sent time and the time at frame reception as well as the path ID.

The fields are:
Path ID The ID of the measured Path
Delay The OWD

4.1.2 Subconnection Scheduling

The scheduling happens on a latency based algorithm. Since paths do not necessarily have to be used bidirectional, the delay is only measured one directional. This minimises the effects of routing policies that are transparent to the transport protocol, since every endpoint maintains its own path metrics. QED uses the algorithm 4.4 and the probabilistic approach which was introduced in section 3.2 used by Sun et al. [STZ+17]. The algorithm (figure 4.4) compares the average OWD of all paths Q_{avg} and the OWD Q_i of a given path `i`. Depending on the result of that comparison the weight the value `S` is added or subtracted. `S` is the desired granularity for weight

```
Q_avg = $P(k) = \frac{\sum_{i=1}^{N} Q_i}{N}$
if Q_i < Q_avg then
    W_i = W_i + S;
    if W: i > 1000 then
        W_i = 1000;
    end if
else
    W_i = W_i - S;
    if W: i < 1 then
        W_i = 1;
    end if
end if
return
```

Figure 4.4: Path Weight Adjustment [STZ+17]

changes. Upper and lower bounds of the weight are 1000 and 0.

The periodically measured one-way delay is compiled to a path metric using a weighted moving average. The OWD is measured using the newly added OWD and OWD_ACK frames. In many transport protocols the OWD is approximated by halving the RTT. However, this approach is highly debatable since the topology of today's internet is mostly asymmetrical [VRT08, CY05]. Furthermore using the OWD instead of an RTT based approximation has proved to be more accurate [CY05].

On the other hand, no method able to calculate the OWD by simply by using the delta of transmission and receive time has been discovered yet due to potentially asynchronous clocks [SSP+09, CY05]. As a consequence, existing OWD measurement algorithms are very complex or have a very narrow scope of application [SLK06, SSP+09], as well as still being analytically derived. Another option would be synchronising the clock using GPS receivers or time synchronizing protocols, this, however, can not be considered as feasible for general purpose applications on the internet.

4.1.3 NAT Traversal

There are several approaches to NAT traversal. Sun et al. [STZ+17] are suggesting that an outgoing packet does also transmit its local IP address. A possible discrepancy between the actual source address and the transmitted source address shows the existence of a NAT in the path. As explained in section 3.2, this behaviour poses privacy and security threats in unencrypted protocols. Since QED and QUIC provide fully encrypted transmission, it would be feasible to use that technique in QED as well. However, adding the local IP address to every QED packet would lead to a major design change thus and most likely to a loss of interoperability.

Another possibility are standardised approaches like STUN [RMMW08] and TURN [MMR10]. Both methods rely on a third party server that is resolving the actual source address and returns it to the requesting endpoint. For NAT traversal the usage of TURN was chosen, since TURN is an extended feature set of STUN. Implementation and in the wild testing will have to show if TURN fulfils all requirements needed to provide full functionality of QED.

4.1.4 References

The QED multipath extension is loosely based on the MPTCP architecture [FRH+11], as well as on the MPIP concept [STZ+17], and was adapted to the needs of a multipath capable QUIC implementation.

QED uses a similar approach as MPTCP since both protocols – or more accurate protocol extensions – are using a well known transport protocol as a substrate. Whereas MPTCP is directly based on TCP, QED uses UDP as a substrate but takes QUIC as abstraction layer for the actual connection oriented transport. MPTCP is able to advertise newly added or changed IP addresses during connection lifetime. But whereas MPTCP uses TCP options to advertise addresses, QED makes use of QUIC frames.

QED also takes the scheduling approach of MPIP deeply into consideration. First implementations will show where adoptions to the requirements of QUIC and QED are necessary and if the overall design is applicable to QED. Further considerations will be addressed in section 7.1.

4.2 QED Enhancements

Since the previous versions of the draft some minor and major changes in the protocol design have been defined.

4.2.1 Frame Design

The former draft included a distinction between IPv4 and IPv6 addresses in its `ADDR_ARRAY` frames. For the sake of simplicity QED now only adds one new frame for the address management. Since a QED packet can carry several frames [IT17] the `ADDR_ARRAY` frame was discarded in favour of multiple `ADDR_MOD` frames sent in one packet during initial address propagation, therefore reducing QED's complexity.

4.2.2 Scheduling

The former version of the draft was proposing a scheduling mechanism that was based solely on the path latency. The updated version employs an approach that filters temporary variability in the OWD using a moving average and a weight based scheduling. Since the last draft `OWD` and `OWD_ACK` frames were also added for OWD measurement. It seemed necessary to add new frames instead using QUIC's `PING` and `PONG` frame because [IT17] only allows the `PONG` frame to echo what the `PING` frame previously transmitted. Despite the possibility of measurement a proper way to retransmit this information in the `PONG` frame is not possible within said specification.

The specification seems to be ambiguous at this point. Section 8.9 says

> If the Data field is not empty, the recipient of this frame MUST generate a PONG frame (Section 8.15) containing the same Data [IT17].

whereas section 8.15 states

> If the content of a PONG frame does not match the content of a PING frame previously sent by the endpoint, the endpoint MAY generate a connection error of type UNSOLICITED_PONG [IT17].

The `MAY` in sec 8.15 indicates that it is allowed to receive such a `PONG` frame whereas the `MUST` in sec 8.9 prohibits the creation of such a frame [IT17]. This behaviour however is not meant to provide semantic leeway for the `PONG` frames but to allow the endpoints not having to keep state forever [QUI18].

4.2.3 NAT Traversal

The former draft assumed no techniques for NAT traversal due to scope considerations. After having found TURN an sufficient approach to this problem it was added to the concept.

5 Realisation

For the implementation of the QED concept described in chapter 4, the QUIC implementation *MINQ* by Eric Rescorla [Res18b] was chosen. The reason for this decision was that Rescorla is one of the key developers of the TLS 1.3 standard, which is fundamental for QUIC but also still in development [Res18a]. Using Rescorla's QUIC implementation allows the assumption that the underlying TLS implementation will work without bigger issues. Another prominent QUIC implementation – *QUIC-go* by Google employee Lucas Clemente[1] [Cle18], used by the commercial webserver Caddy – is relying on this TLS implementation which is called *MINT*[2] [bif18] for its experimental IETF QUIC support.

Furthermore, Rescorla takes part in the standardisation of QUIC at the IETF and keeps the design of his implementation close to the IETF draft. Since Google's version of QUIC has not been publicly reviewed and is moving towards the IETF standard anyway [Cle18], using Google's version of QUIC would not be very sustainable since the potential of further works would be limited to the lifetime of Google's version of QUIC.

5.1 Implementation

As stated in section 3.1, QUIC does not rely on the classic 5-tuple[3] to identify a connection like TCP does [Pos81b]. It makes use of a unique session ID [IT17] instead. This has the advantage that a connection is not bound to one IP address. This is the central aspect for the multipath extension approach presented in this

[1]https://github.com/lucas-clemente/quic-go visited on January 27th 2018
[2]https://github.com/bifurcation/mint visited on January 27th 2018
[3]IP address pair, port pair and protocol

work.

The development of the extension required several modifications and additions to the original MINQ code by Rescorla [Res18b] to map the extensions the concept in chapter 4 defined. The two main components were the scheduling and the path management. Also added were new frames like depicted in section 4.1 and an address management mechanism that keeps track of available IP addresses and notifies if changes occur.

5.1.1 Address Management

To make use of multipath capability QED needs to keep track of available addresses and adapt to interface and address changes. The `AddressHelper` class[4] provides this functionality.

Each instance running QED owns one `AddressHelper`. It runs an address inventory on startup collecting all non-link-local addresses [CAG05, HD06] from non-loopback interfaces[5] and storing them for the duration of execution. This process is executed periodically to adapt to address and interface changes. In case of an address or interface change `AddressHelper` notifies all `Scheduler` instances. The implementation is similar to an observer pattern and uses Go's `channel` feature used for communication between concurrent threads. To avoid issues based on concurrent write access the `AddressHelper` class as well as the `Scheduler` class are equipped with mutexes.

The deletion of addresses is vulnerable to issues caused by race conditions as well. Since all addresses handled by the `scheduler` instances are pointers referring to the original object[6] held by the `AddressHelper`, the timing of the deletion is crucial to avoid segmentation violations leading to a crash. Even though networking applica-

[4]Since *Go* is not a object oriented programming language [Tea18], it does not have formal classes. Instead *structs* and *functions* pointing to *structs* are used as a surrogate for classes. Since their semantic and syntactic usage is hardly different from classes the term *class* is used throughout this document.

[5]A loopback interface is a virtual network interface that only has one endpoint [GA07]. Sender and receiver on a loopback interface are necessarily the same. It is used for intra machine communication.

[6]The footnote regarding classes applies on objects as well.

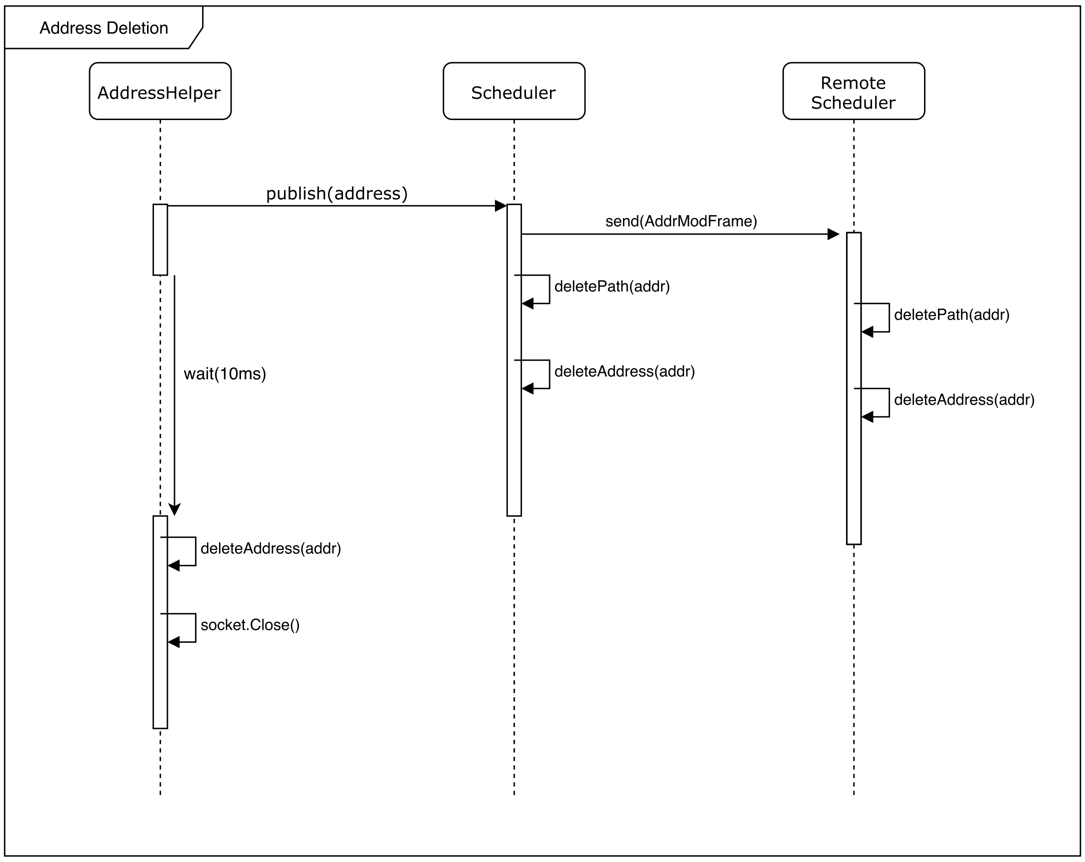

Figure 5.1: Deletion of an Address

tions are usually time sensitive [Tom15] and QUIC is so focussed on speeding up the internet (section 3.1) that it's stated in its acronym, the current implementation uses a short grace period to finish all functions and tear down all objects using this address before deleting the actual object, as shown in figure 5.1.

Additions of addresses can either happen through the `AddressHelper` itself, if it detects new addresses running its checks, or via an incoming `ADDR_MOD` or `ADDR_ARRY` frame. In the latter case the handling `Connection` passes the address on to the `AddressHelper` and in both cases the `AddressHelper` announces address changes to the `Scheduler`.

5.1.2 Path Management and Scheduling

Path management and scheduling are core elements of the QED extension. The `Scheduler` class provides the necessary functionality. The `Scheduler` gets initiated through the `MINQ` connection handling. `MINQ` instantiates a new `Connection` object when a new QUIC connection is incoming or outgoing. `Connection` is a `MINQ` class that represents a QUIC connection and handles outgoing packets and processes incoming ones. Every `Connection` has one `Scheduler` attached which handles `Path` management. A sequence diagram of the initial path establishment is shown in figure 5.2. After its creation, the `Scheduler` instantiates `Path` objects and schedules messages over all paths.

Every newly added address propagated by the `AddressHelper` or an `ADDR_ARRAY` frame triggers the creation of a new `Path`. The structure of a `Path` is shown in figure 5.3. Since the recent changes in the concept regarding the scheduling (See section 5.2 for details) could not be implemented due to code freeze, the actual implementation still uses a rudimentary round robin approach. Therefore RTT and metric are set to default values during construction. The `PathID` is an CRC32 [DG96] checksum using the Institute of Electrical and Electronics Engineers (IEEE) polynomial [Bra75] seeded with the concatenated values of the local and remote IP addresses. The previously used Adler32 algorithm [DG96] was dismissed due to its flaws regarding similar and consecutive seeds [SSO02].

As mentioned before the actually implemented scheduling uses a straightforward algorithm depicted in figure 5.4. Due to its design the scheduling algorithm is biased to IPv6 addresses. Since IPv6 enabled interfaces usually have more than one IP address – due to the IPv6 privacy extensions [NDK07] – QED creates multiple paths for these interfaces. Because of the round robin scheduling each path is utilised equally often. The existence of multiple IPv6 paths in one connection has the effect that IPv6 is used more than IPv4. To mitigate this behaviour it would be necessary to detect IPv6 addresses created because of the privacy extension and the automatically configured address. Furthermore it would have been necessary to implement the detection of manually added addresses. But it is to question anyway if the preference of IPv6 addresses is a negative aspect at all. Since IPv6 has many advantages towards IPv4 [PS18, Lon18] a concentration of traffic in favour of IPv6 is mainly positive. Especially when there is still a full compatibility to single-stack

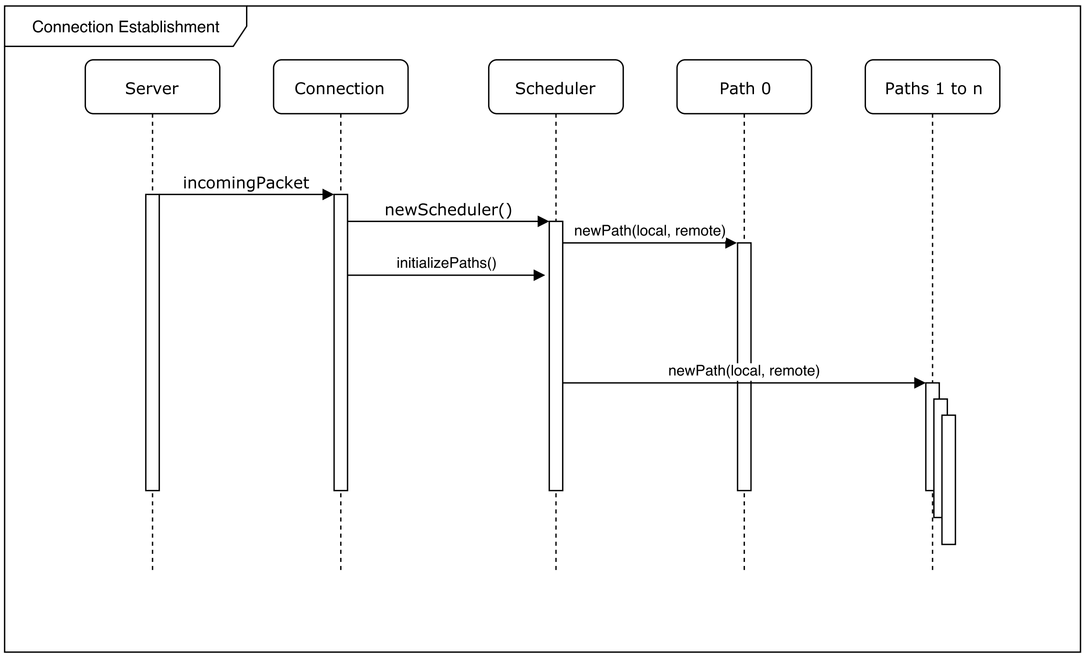

Figure 5.2: Establishment of a Multipath Capable QUIC Connection

IPv4 endpoints.

5.2 Limitations

The current implementation has a few limitations or deviations in comparison with the QED concept. These were mainly caused by unexpected difficulties during implementation and a shortage of time.

The most obvious limitation in QED's design is the heavy focus on the transport aspects of the protocol design. Important features like a fair congestion control are not yet part of the concept but are fields of further research.

Scheduling Even though implementing the scheduling algorithm defined in chapter 4 was successful as the evaluation in section 6.1 shows it could not be properly implemented in QED due to an unsolved dead lock in the OWD estimation mechanism.

Dynamic IP Removal Due to so far unknown reasons the dynamic removal of IP addresses leads to unwanted behaviour. The removal of an active link is not

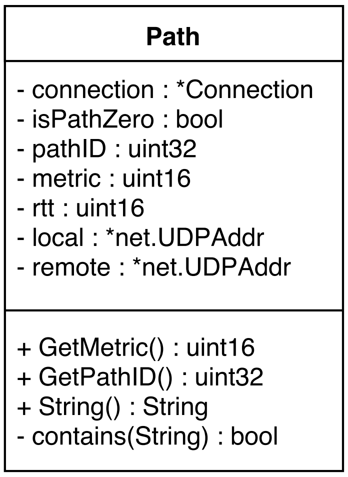

Figure 5.3: Path Structure

```
func (s *Scheduler) Send(p []byte) error {
        s.lockPaths.RLock()
        defer s.lockPaths.RUnlock()
        s.lastPath = (s.lastPath + 1) % uint32(len(s.pathIds))
        tempPath := s.paths[s.pathIds[s.lastPath]]
        err := tempPath.transport.Send(p)
        if err != nil {
                fmt.Println(err, util.Tracer())
                return err
        }
        if s.lastPath == 0 {
                s.connection.log(
                        logTypeMultipath,
                        "Packet sent. Used path zero"
                        )
        } else {
                s.connection.log(
                        logTypeMultipath,
                        "Packet sent. local: %v \n remote: %x",
                        *s.paths[s.pathIds[int(s.lastPath)]].local,
                        *s.paths[s.pathIds[int(s.lastPath)]].remote
                        )
        }
        return nil
}
```

Figure 5.4: QED Scheduling Algorithm

properly caught in QED's mechanics. This causes QED to continually write on the closed socket. A reestablishment of that given link leads to a memory violation. Since dynamic path handling is a core aspect of a robust multipath protocol, these issues will be part of further research.

NAT Traversal Due to the quite narrow development scope is was not feasible to implement a proper NAT traversal in the course of this work. But since today's ossified internet makes the ability to traverse NAT's virtually mandatory further, research has to design and implement a solution for this issue.

Server Address Propagation To ensure multipath capability for both endpoints each one has to announce its addresses. Running the code during tests showed that the endpoint acting as the server did not react to the client's address announcement. Thus the communication from server to client utilised only one path. Since both instances use the same address propagation and scheduling logic it is unknown why this behaviour occurs and will have to be subject of further research.

High Delay Errors While testing the implementation it became obvious that links with a high delay cause massive trouble for QED. Transmission times multiplied in comparison to other tests. Detailed test results will be discussed in chapter 6.

Implementation Repository The implementation is available on GitHub under the following repository: `github.com/boisjacques/minq`. Two branches exist, the `master` which contains the improved scheduling algorithm but is defunct and `latest-running` which was used for general evaluation but schedules following a round-robin approach.

The following commits are the reference for this work:

master	`61a2e29`
latest-running	`1b9dc99`

6 Evaluation and Comparison

This chapter contains an evaluation of the implemented QED components, as well as a comparison between QED on one side and MPTCP and MPQUIC on the other. Focus of the evaluation will be the overall multipath functionality of QED and the quality of the scheduling algorithm. The comparison will concentrate on the approaches each of the protocols have towards multipath transmission and design decisions the protocol developers made.

6.1 Evaluation

In this part QED's performance is evaluated in general, as well as under lossy network links or such that suffer delay. Furthermore, the scheduling algorithm is tested in a simulation.

6.1.1 Multipath Transmission

Even though the implemented prototype does not completely cover all aspects of the QED concept, evaluation shows that it has the capability to transmit ordered data over multiple paths. Two different testbeds were used to evaluate the multipath capability. A physical testbed and a virtualised one (shown in figure 6.1). Both testbeds were dual stack enabled and thus used IPv4 as well as IPv6 on each interface.

The virtual testbed consisted of two virtual machines[1], each of which had four

[1]Virtual Box running Ubuntu 17.10

```
+----+                 +----+     +----+                 +----+
|    |                 |    |     |    |      +---+      |    |
| if0<----------------->if0 |     | if0<------>   <------>if0 |
|    |                 |    |     |    |      |   |      |    |
|    |                 |    |     |    |      |SW0|      |    |
|    |                 |    |     |    |      |   |      |    |
| if1<----------------->if1 |     | if1<------>   <------>if1 |
|    |                 |    |     |    |      +---+      |    |
|    |                 |    |     +----+                 +----+
|    |                 |    |     Host A                 Host B
| if2<----------------->if2 |
|    |                 |    |
|    |                 |    |
|    |                 |    |
| if3<----------------->if3 |
|    |                 |    |
+----+                 +----+
Host A                 Host B
```

Figure 6.1: Left: VM Testbed
Right: Lab Testbed

network interfaces. It was primarily used to test QED in small increments during development. It was shown that QED makes use of all available paths.

The physical testbed consisted of two machines – each with multiple network adapters – as well as two routers in the first place. Each computer was connected to each of both routers with one network interface. It showed during the tests described in the next section that the routers caused significant issues during the tests. It is yet unknown what caused the problems. It is possible that the routers[2] – which were one of the first Cisco routers with IPv6 capability – suffered from a defunct IPv6 implementation.

To circumvent those issues the testbed was restructured and both endpoints were connected to a switch. Disabling the routers turned out to have the desired effect.

[2]Cisco 3600 Series

		if0	**if1**	**if2**	**if3**
VM	Delay	50ms	100ms	200ms	300ms
	Loss	5%	2%	1%	7%
Lab	Delay	50ms	50ms		
	Loss	1%	3%		

Table 6.1: Simulation Parameters

Simulating Networking Delay and Losses

To analyse the behaviour of QED under different network conditions the Linux tool `netem` was used to simulate as well delay as packet loss in both testbed environments. The simulation parameters are shown in table 6.1. The parameters for packet loss were estimated values for a WLAN link based on long term measurements using the traceroute tool `mtr`[3]. The delay parameter was chosen to test QED's performance whilst having paths with high delay differences. The tests' payloads were three randomly generated text files with the size of 2MB, 10MB and 100MB. Those files were transmitted to an echo server via QED which flipped each characters case and transmitted the text back to the client. The payload then was compared to a SHA-256 hash to detect errors. Each test (plain/loss/delay) was executed 100 times with each payload.

Since QED does not extend any of *MINQ*'s flow and congestion control mechanisms it was expected that simulated network errors would negatively affect the transmission, but this assumption turned out to be overly pessimistic. The comparison of tests without network issued and the tests with loss showed almost no differences in success and time of transmission. In contrast to this, however, delay caused major issues as table 6.2 demonstrates. First of all roughly every second test caused the QED server to crash and thus the test to fail. And if the test passed, the rate of transmission was reduced to about 50% to 10% compared to the other two tests as figure 6.2 shows. It was noticed during the tests that failing transmissions always failed within the first second of the transmission. Log files show an error during the creation of a path. A possible explanation could be that an `ADDR_MOD` frame was sent over a delayed link, assumed lost and sent via a less delayed link, resulting a race condition which caused an illegal state in the `AddressHandler` or the `Scheduler`

[3]http://www.bitwizard.nl/mtr/ visited on January 27th 2018

	Plain	**Loss**	**Delay**
	Success Rate		
2 MB	100%	100%	53%
10 MB	100%	100%	50%
100 MB	100%	98%	53%

Table 6.2: QED Multipath Evaluation Results

during path creation.

6.1.2 Scheduling

The scheduling algorithm described in chapter 4 turned out to work as assumed in the concept. Since the implementation of the scheduling algorithm could not be provided, the algorithm was tested in a simulation. For this simulation ten paths with different weight were created[4] and the scheduling algorithm was executed 100,000 times. Figure 6.3 shows a clear dependency between the path weight and the number of selections by the algorithm. Table 6.3 includes the exact figures. As the path weight is dependent on the paths' average OWD this scheduling mechanism leads to a preference of paths with a low OWD.

However, the scheduling neither considers the paths congestion nor are these simulated results representative for a real implementation tested as a part of QED. Section 7.1 will show considerations to circumvent these limitations.

6.1.3 MINQ evolves to QED

During the development of QED the code base of *MINQ* was extended by roughly 1000 lines of Go code, which is an addition of about 20%. The latest version of *MINQ* that was considered for QED has 5,186 lines of code whereas QED in its latest – deadlocking – build has 6,151 lines. The latest working version consists of 5,859 lines of code.

[4]The paths weights were not dynamically set but statically defined due to simulation limitations

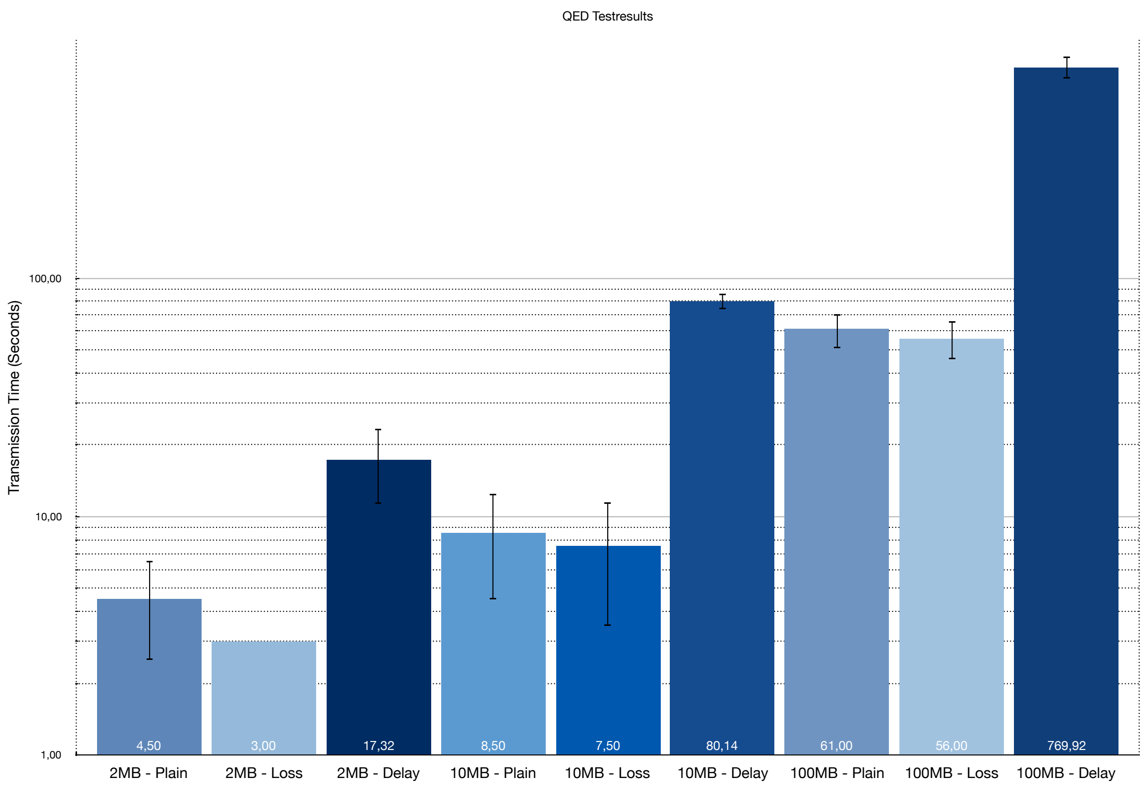

Figure 6.2: QED Multipath Evaluation Results

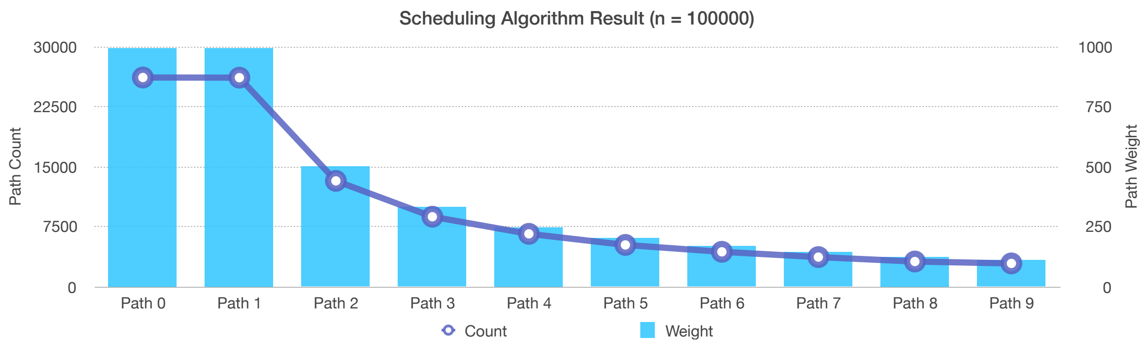

Figure 6.3: QED Scheduling Evaluation Results

	Weight	Count
Path 0	1000	26131
Path 1	1000	26112
Path 2	500	13209
Path 3	333	8726
Path 4	250	6586
Path 5	200	5200
Path 6	166	4339
Path 7	142	3681
Path 8	125	3126
Path 9	111	2890

Table 6.3: Absolute Simulation Results

6.2 Comparison

This section compares QED with MPQUIC and MPTCP. It mostly focuses on a comparison with MPQUIC as both extend the QUIC protocol. But MPTCP – albeit its official experimental status – is relevant enough for a short analysis in comparison to QED.

6.2.1 MPQUIC

The protocol that has the closest relation to QED is MPQUIC, basically because it was designed to fulfil the same purpose: extend QUIC with multipath capability. Both extensions leverage similar design aspects of QUIC to enable multipath capability. Thus, they are layouted very similarly but still differ in detail.

MPQUIC starts a connection with the negotiation of the maximum number of paths every endpoint is willing to open. Both endpoints agree on the smaller `max_path_id`. QED, by contrast, uses all available addresses and opens as many paths as possible without negotiation. Benefit of the `max_path_id` is a better ability to allocate the resources accordingly to the endpoints' need. This modifies the QUIC handshake, however, which could lead to unwanted behaviour if interoperability to unipath QUIC is a design goal.

Another design difference is the handling of packet acknowledgements. Both protocols can choose the path they send the `ACK` messages on independently from the `ACK`ed messages. This has the advantage that endpoints can schedule the acknowledgements for packets that may have arrived on a high latency path back on a low latency path. MPQUIC uses per path `ACK` messages, whereas QED maintains one `ACK` message scope for the entire connection. MPQUIC chooses this approach because it may bring more flexibility. This has to be subject of further investigation.

Both protocols are extending the set of frames QUIC provides. Each protocol uses newly added frames for address announcements. MPQUIC uses the `ADD_ADDRESS` frame, whereas QED uses the `ADDR_MOD` frame. The approaches differ in detail. Whereas MPQUIC uses another frame (the `REMOVE_ADDRESS` frame) to remove addresses, QED provides its `ADDR_MOD` frame with the ability to indicate addition or removal of the address. By using two different frames for address addition and removal MPQUIC's design is more resilient towards flipping bits. Because the QUIC specification provides packet integrity by design [TT17a], this consideration might be less relevant. Another positive effect of MPQUIC's two frame model in combination with the use of an `Address ID` is a downsizing of the overhead. A MPQUIC `REMOVE_ADDRESS` frame only carries a 1-byte `Address ID`, whereas QED's `ADDR_MOD` frame transmits the IP address which in case of IPv6 makes a payload of 16 bytes. But that number seems to be negligible due to the capacity of today's internet. On the downside, the addition of frames always brings more complexity into play, an effect that should possibly be avoided in protocol design.

Another aspect that differs for both protocols is the dependency of paths and addresses on each other. QED permanently ties a path to a pair off addresses. If one of these addresses gets removed, the lifetime of the path is over. If the corresponding endpoint announces a new address, a new path is established. MPQUIC, on the other hand, does not seem to abandon paths whose addresses are flagged as lost, instead it just disables them until a new address can be provided. The exact mechanism remains unclear since the MPQUIC specification just states that paths get flagged as inactive but does not specify any further actions [CB17].

6.2.2 MPTCP

QED and MPTCP have probably more differences than things in common. Both reside in OSI-layer 4 and are multipath capable protocols. But looked at on a lower level many differences are coming to light. This comparison will focus on the most significant ones that are caused by the protocols' differing substrates, one uses the UDP-based QUIC as a foundation whereas MPTCP extends TCP.

The most significant difference is the handling of new paths – or subflows in MPTCPs terminology. MPTCP requires a full TCP connection establishment for each new subflow that is established. This is necessary due to middleboxes, some of which are dropping TCP packets without a preceding connection establishment. This causes the same overhead TCP has for initiating connections for every subflow. QED however does not need a special handshake procedure for new paths. Both endpoints announce their addresses to each other and are opening paths without any further handshake. This is firstly possible due to the UDP substrate which does not need to carry special information for middlebox traversal. Secondly QUICs `Connection ID` feature allows the attribution of incoming packets by changing 5-tuples.

Furthermore, MPTCP needs its packets to have two spaces of sequence numbers. One for MPTCP operation with a connection wide scope and another one for middlebox traversal. The subflow-scoped sequence number is not necessary for multipath transmission but rather a concession for middlebox traversal because middleboxes seeing packets with out of order sequence numbering will most certainly drop this packet. QED can maintain one pool of packet numbers over a connection. The middlebox traversal strategy again relies on QUIC's UDP foundation.

Another key difference is the way TLS is handled by the protocol. While QED incorporates QUIC's always-on encryption MPTCP provides no possibility for native encryption. Even though there is a draft describing MPTLS [Bon14] no further development of neither the draft nor an actual implementation is known.

7 Summary and Outlook

This chapter presents a summary of this work regarding its findings and an outlook over future fields of research, which could not be addressed in this work.

This work introduced several approaches to multipath transmission. It found MPTCP and MPQUIC to be the most promising ones to be widely used on the internet due to their ossification resilient design. SCTP is a very mature transport protocol as well, but because of the structure of today's internet it is unlikely that it will be widely deployed. The recent development of MPQUIC and a recent IETF draft regarding QUIC's multipath requirements [Hui17] show that a lot of fundamental research is carried out in order to make QUIC multipath enabled.

Later in this work QED was introduced and showed how straightforward a prototypical multipath extension can be designed leveraging the flexible and expandable design QUIC incorporates. QUIC's design based on connection identifiers instead of the classical 5-tuple[1] created a layer of indirection that isolates the actual transport from IP addresses. QED provides a partially functioning multipath extension to QUIC. Evaluation of the extension showed an overall stable performance.In the case of delayed links QED suffers transmission failures and a drop of the transmission speed to around 10% of the speed of an unaffected link.

The simulated scheduling algorithm performed as expected but further research has to solve implementation issues. Further evaluation has to show if the algorithm actually provides benefit over the current round-robin implementation or scheduling approaches that use another metric instead of the OWD.

But despite the achievements that were reached, QED showed that there are still many open aspects that have to be addressed by further research.

[1]Source and destination address, the corresponding port pair and the protocol

7.1 Outlook

First and foremost the main goal will be to refine the implementation of QED to a point where it works as intended. To this point address removal as well as path scheduling are faulty due to different reasons. The implementation of the scheduling algorithm is not functional yet due to a suspected deadlock. The working build has issues regarding address modifications, as well as address propagation. Solving these errors will be the base for further development and research.

QED implements OWD measurements to determine the paths weight. The current implementation uses a very simple approach to measure the OWD, which is vulnerable to errors caused by asynchronous clocks. It is up to future research to determine if existing concepts of OWD measurement – for example the work of Choi et al. [CY05] can be usefully implemented by QED.

The current QED design had its focus merely on the transport aspect of protocol development. However to be fully functional QED requires conceptional and implementational research regarding flow and congestion control. The current congestion control scheme QUIC brings along is unfair to concurrent connections on a link [CB17] and therefore needs to be revised. Furthermore, the scheduling algorithm is heavily dependent on the calculated path weight. Further research will have to investigate the influence of different path weighting strategies.

After existing implementation issues have been solved QED needs to be tested on the internet. VM and lab tests can provide a good insight into the overall performance and can even simulate links with a poor performance. None of the variants can test how the protocol reacts to middleboxes and other restrictions on the internet, though.

Another aspect of modern transport protocols – a socket interface – is not yet foreseen by the IETF's standardisation process. Further research has to design a concept for a socket API as an abstraction layer for QUIC and its multipath variants.

Glossary

CM Control Message. A control message is the signalling element for Multipath IP. Its structure is layed out in figure 3.1 14, 15

CMT-SCTP Concurrent Multipath Transfer Extension to SCTP. CMT-SCTP is an extension to SCTP that enables actual multipath transmission [Iye06]. 18, 21

DoS Denial of Service. A Denial of Service attack is an attack that utilises resources of one (or more if it is a Distributed Denial of Service attack) computers to allocate the target's resources. The aim is to make the targeted service unresponsive [DRD04]. 18

ECMP Equal-cost mutipath routing. ECMP is a routing approach in case of multiple routes with equal metrics [TH00]. 10–12

HoL Head-of-Line blocking. Head-of-Line blocking describes a phenomenon that can occur within in-order delivery transport protocols. If such a protocol like TCP needs to retransmit data all following data needs to wait for this retransmission to finish [SFR04]. 2, 17, 21

HTTP Hypertext Transfer Protocol. An application protocol for stateless data transmission [FGM$^+$99, BPT15]. 1–3

IEEE Institute of Electrical and Electronics Engineers. A worldwide professional association of engineers mainly from the fields of electrical engineering and information technology. 42

IETF Internet Engineering Task Force. An organisation that works on the development of internet techniques [HB02]. 7–10, 17, 20, 29, 39, 55, 56

IP Internet Protocol. The Internet Protocol is the core protocol of the internet. It provides magic and AWESOME! [Pos81a] 12–18, 26, 31–33, 36, 39, 40, 42, 43, 53, 55

IS-IS Intermediate System to Intermediate System. IS-IS is a routing protocol that was originally designed for CLNP networks. Later it was adapted for TCP/IP networks as specified in [Cal90]. 11, 12

$\textbf{MAC}_1$ Media Access Control. The Media Access Control address of a network interface is its unique identifier. It consists of six bytes, three of which are the vendor's ID and the other three are the device identifier. The administration of the MAC address space and the assignment is managed by the Institute of Electrical and Electronics Engineers (IEEE) [DMR08]. 13

$\textbf{MAC}_2$ Message Authentication Code. A Message Authentication Code is a mechanism used to verify the integrity and origin of data [MvOV96]. 18

MINQ MINQ. An experimental QUIC implementation by Eric Rescorla [Res18b]. 40

MitM Man in the Middle. A Man in the Middle attack is an attack where an adversary gets into the communications channel of its victims to eavesdrop, withhold or modify information [Tox03]. 16

MPIP Multipath IP. Multipath IP is an experimental concept for a multipath capable IP stack [STZ$^+$17]. 12–16, 36

MPQUIC Multipath QUIC. An experimental QUIC extension that adds multipath functionality developed by De Coninck et al. [DCB17]. vii, 27–29, 47, 52, 53, 55

MPTCP Multipath TCP. MPTCP is a multipath capable extension for TCP [FRHB13]. 11, 12, 22–29, 32, 36, 47, 52, 54, 55

MTU Maximum Transmission Unit. The MTU is the maximum packet size that a given OSI layer 3 link can handle [Pos81a]. If a packet exceeds this size it has to be fragmented into several packets that are matching the MTU. 11

NAT Network Address Translation. To overcome the shortage of public IPv4 addresses Network Address Translation was invented. It maps a public IP address to one or more private ([RMK$^+$96]) addresses [EF94]. 13–15, 25, 27, 36, 38, 45

OSPF Open Shortest Path First. OSPF is an IGP that uses Dijkstra's shortest-path algorithm to route packets accordingly [Moy98]. 11, 12

OWD One Way Delay. The One Way Delay is the time a networking transmission takes from its sender to its receiver [TW04]. 15, 26, 32, 34, 35, 37, 43, 50, 55, 56

PSTN Public Switched Telephone Network. A Public Switched Telephone Network is a signalling network for telephone communication [Hil12]. 17

QED QUIC with Enhanced Distribution. An experimental QUIC extension that adds multipath functionality developed during this work. xv, xvii, 31–34, 36, 37, 39, 40, 42–45, 47–56

QoE Quality of Experience. The term Quality of Experience describes the delight or annoyance a customer made while using a service [MR14]. 5, 16

QUIC Quick UDP Internet Connections. The QUIC protocol is a transport protocol initially developed by Google that is in the IETF standardisation process now [Pöt16]. v, 1–3, 6–8, 11, 17, 27–29, 31, 32, 36, 37, 39, 41, 42, 52–56

RTT Round-trip Time. The Round-trip Time is the time an information takes to its receiver and the receivers answer back to the sender [fTS18]. 26, 28, 35, 42

SCTP Stream Control Transmission Protocol. The Stream Control Transmission

Protocol is a multipath capable transport protocol. It was designed for signalling in cellular networks [Ste07]. 3, 11, 17, 18, 20–22, 28, 55

SPDY SPDY. An experimental successor of HTTP/1.x developed by Google and proprietary version of HTTP/2. 1–3

STUN Session Traversal Utilities for NAT. The Session Traversal Utilites for NAT is a networking protocol that allows the detection of a NAT in a network [RMMW08]. 15, 36

TCP Transmission Control Protocol. TCP is a OSI layer 4 transport protocol. It is connection oriented and provides reliable in order packet transport [Pos81b]. v, 1–3, 5–7, 11, 12, 18, 20–29, 32, 36, 39, 54

TLS Transport Layer Security. A hybrid encryption protocol for encryption of internet traffic [Res18a] 3, 5, 8, 39, 54

TURN Traversal Using Relays around NAT. Traversal Using Relays around NAT is a networking protocol that allows the detection of a NAT in a network [MMR10] 15, 36, 38

UDP User Datagram Protocol The User Datagram Protocol is a simple transport protocol. It provides neither reliable transmission nor congestion or flow control [Pos80]. v, 2, 3, 5, 6, 12, 22, 25, 27, 31, 36, 54

List of Tables

List of Figures

Bibliography

[ABD$^+$18] Professor Paul D. Amer, Martin Becke, Thomas Dreibholz, Nasif Ekiz, Janardhan Iyengar, Preethi Natarajan, Randall R. Stewart, and Michael Tüxen. Load Sharing for the Stream Control Transmission Protocol (SCTP). Internet-Draft draft-tuexen-tsvwg-sctp-multipath-15, Internet Engineering Task Force, January 2018. Work in Progress.

[App18] Apple. Use multipath tcp to create backup connections for ios. `https://support.apple.com/de-de/HT201373`, January 11th 2018.

[APS99] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.

[BCK13] Irene Bertschek, Daniel Cerquera, and Gordon J Klein. More bits–more bucks? measuring the impact of broadband internet on firm performance. *Information Economics and Policy*, 25(3):190–203, 2013.

[BG16] Prasenjeet Biswal and Omprakash Gnawali. Does quic make the web faster? In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–6. IEEE, 2016.

[bif18] bifuraction. A minimal tls 1.3 implementation in go. `https://github.com/bifurcation/mint`, January 27th 2018.

[Bis17] Mike Bishop. Hypertext Transfer Protocol (HTTP) over QUIC. Internet-Draft draft-ietf-quic-http-07, Internet Engineering Task Force, October 2017. Work in Progress.

[Bon14] Olivier Bonaventure. MPTLS : Making TLS and Multipath TCP stronger together. Internet-Draft draft-bonaventure-mptcp-tls-00, Internet Engineering Task Force, October 2014. Work in Progress.

[BPG+15] M. Bagnulo, C. Paasch, F. Gont, O. Bonaventure, and C. Raiciu. Analysis of Residual Threats and Possible Fixes for Multipath TCP (MPTCP). RFC 7430 (Informational), July 2015.

[BPT15] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540 (Proposed Standard), May 2015.

[Bra75] Kenneth Brayer. Evaluation of 32 degree polynomials in error detection on the satin iv autovon error patterns. Technical report, MITRE CORP BEDFORD MASS, 1975.

[CAG05] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927 (Proposed Standard), May 2005.

[Cal90] R.W. Callon. Use of OSI IS-IS for routing in TCP/IP and dual environments. RFC 1195 (Proposed Standard), December 1990. Updated by RFCs 1349, 5302, 5304.

[CB17] Quentin De Coninck and Olivier Bonaventure. Multipath Extension for QUIC. Internet-Draft draft-deconinck-multipath-quic-00, Internet Engineering Task Force, October 2017. Work in Progress.

[Cla04] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004.

[Cla08] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), January 2008. Obsoleted by RFC 7011.

[Cle18] Lucas Clemente. A quic implementation in pure go. `https://github.com/lucas-clemente/quic-go`, January 27th 2018.

[CLL+17] Yong Cui, Tianxiang Li, Cong Liu, Xingwei Wang, and Mirja Kühlewind. Innovating transport with quic: Design approaches and research challenges. *IEEE Internet Computing*, 21(2):72–76, 2017.

[CMTH17] Sarah Cook, Bertrand Mathieu, Patrick Truong, and Isabelle Hamchaoui. Quic: Better for what and for whom? In *IEEE International Conference on Communications (ICC2017)*, 2017.

[CY05] Jin-Hee Choi and Chuck Yoo. One-way delay estimation and its application. *Computer Communications*, 28(7):819–828, 2005.

[DCB17] Quentin De Coninck and Olivier Bonaventure. Multipath quic: Design and evaluation. In *13th International Conference on emerging Networking EXperiments and Technologies (CoNEXT 2017). http://multipath-quic. org*, 2017.

[DG96] P. Deutsch and J-L. Gailly. ZLIB Compressed Data Format Specification version 3.3. RFC 1950 (Informational), May 1996.

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Obsoleted by RFC 8200, updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.

[DMR08] M.A. Dye, R. McDonald, and A.W. Rufi. *Netzwerkgrundlagen: CCNA exploration companion guide*. Pearson Deutschland, 2008.

[DRD04] D. Dittrich, P. Reiher, and S. Dietrich. *Internet Denial of Service: Attack and Defense Mechanisms*. Radia Perlman series in computer networking and security. Pearson Education, 2004.

[Edd07] W. Eddy. TCP SYN Flooding Attacks and Common Mitigations. RFC 4987 (Informational), August 2007.

[EF94] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[FALZ12] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn. Diameter Base Protocol. RFC 6733 (Proposed Standard), October 2012. Updated by RFC 7075.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and

T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[For18] Internet Engineering Task Force. Website of the quic working group. `https://quicwg.github.io/`, January 12th 2018.

[FRH$^+$11] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC 6182 (Informational), March 2011.

[FRHB13] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), January 2013.

[FTK17] G. Fairhurst, B. Trammell, and M. Kuehlewind. Services Provided by IETF Transport Protocols and Congestion Control Mechanisms. RFC 8095 (Informational), March 2017.

[fTS18] The Institute for Telecommunication Sciences. Round-trip delay time. `https://www.its.bldrdoc.gov/fs-1037/dir-031/_4641.htm`, January 9th 2018.

[GA07] P.H. Ganten and W. Alex. *Debian GNU/Linux: Grundlagen, Einrichtung und Betrieb*. X.systems.press. Springer Berlin Heidelberg, 2007.

[Han06] M. Handley. Why the internet only just works. *BT Technology Journal*, 24(3):119–129, Jul 2006.

[HB02] P. Hoffman and S. Bradner. Defining the IETF. RFC 3233 (Best Current Practice), February 2002.

[HD06] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), February 2006. Updated by RFCs 5952, 6052, 7136, 7346, 7371, 8064.

[HFGN12] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno

Modification to TCP's Fast Recovery Algorithm. RFC 6582 (Proposed Standard), April 2012.

[Hil12] G. Hill. *The Cable and Telecommunications Professionals' Reference: PSTN, IP and Cellular Networks, and Mathematical Techniques*. Taylor & Francis, 2012.

[HISW16] Ryan Hamilton, Janardhan Iyengar, Ian Swett, and Alyssa Wilk. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Internet-Draft draft-tsvwg-quic-protocol-02, Internet Engineering Task Force, January 2016. Work in Progress.

[Hop00] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), November 2000.

[Hui17] Christian Huitema. QUIC Multipath Requirements. Internet-Draft draft-huitema-quic-mpath-req-00, Internet Engineering Task Force, December 2017. Work in Progress.

[IAS06] Janardhan R Iyengar, Paul D Amer, and Randall Stewart. Concurrent multipath transfer using sctp multihoming over independent end-to-end paths. *IEEE/ACM Transactions on networking*, 14(5):951–964, 2006.

[IS17] Janardhan Iyengar and Ian Swett. QUIC Loss Detection and Congestion Control. Internet-Draft draft-ietf-quic-recovery-08, Internet Engineering Task Force, December 2017. Work in Progress.

[IS18] Janardhan Iyengar and Ian Swett. QUIC Loss Detection and Congestion Control. Internet-Draft draft-ietf-quic-recovery-09, Internet Engineering Task Force, January 2018. Work in Progress.

[IT17] Janardhan Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-08, Internet Engineering Task Force, December 2017. Work in Progress.

[Iye06] Janardhan R Iyengar. End-to-end concurrent multipath transfer using transport layer multihoming. Technical report, DELAWARE

UNIV NEWARK DEPT OF COMPUTER AND INFORMATION SCIENCES, 2006.

[JLT15] Randell Jesup, Salvatore Loreto, and Michael Tüxen. WebRTC Data Channels. Internet-Draft draft-ietf-rtcweb-data-channel-13, Internet Engineering Task Force, January 2015. Work in Progress.

[Kie17] Manuel Kieweg. Neuartige transportprotokolle am beispiel von quic, 2017.

[KS13] S Shunmuga Krishnan and Ramesh K Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, 2013.

[KT18] KT. Kt's giga lte. `https://www.ietf.org/proceedings/93/slides/slides-93-mptcp-3.pdf`, January 11th 2018.

[Lei13] B. Leiba. Registration of Second-Level URN Namespaces under "ietf". RFC 6924 (Informational), April 2013.

[LIJM$^+$10] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. *SIGCOMM Comput. Commun. Rev.*, 41(4):–, August 2010.

[Lon18] Clive Longbottom. 4 reasons why ipv6 has not taken off – and 3 why it should. `http://blog.silver-peak.com/4-reasons-why-ipv6-has-not-taken-off-and-3-why-it-should`, January 8th 2018.

[LOTD08] P. Lei, L. Ong, M. Tuexen, and T. Dreibholz. An Overview of Reliable Server Pooling Protocols. RFC 5351 (Informational), September 2008.

[MD90] J.C. Mogul and S.E. Deering. Path MTU discovery. RFC 1191 (Draft Standard), November 1990.

[MDMH17] J. McCann, S. Deering, J. Mogul, and R. Hinden. Path MTU Discovery for IP version 6. RFC 8201 (Internet Standard), July 2017.

[MKM16] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. How quick is quic? In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[MMR10] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), April 2010. Updated by RFC 8155.

[Moy98] J. Moy. OSPF Version 2. RFC 2328 (Internet Standard), April 1998. Updated by RFCs 5709, 6549, 6845, 6860, 7474, 8042.

[MP15] Stephen McQuistin and Colin Perkins. Reinterpreting the transport protocol stack to embrace ossification. 2015.

[MR14] Sebastian Möller and Alexander Raake. *Quality of experience: advanced concepts, applications and methods.* Springer, 2014.

[MvOV96] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* Discrete Mathematics and Its Applications. CRC Press, 1996.

[NDK07] T. Narten, R. Draves, and S. Krishnan. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941 (Draft Standard), September 2007.

[Nie18] Jakob Nielsen. Nielsen's law of internet bandwidth. `https://www.nngroup.com/articles/law-of-bandwidth/`, 09.01.2018.

[Not18] Mark Nottingham. Http/2 implementation status. `https://www.mnot.net/blog/2015/06/15/http2_implementation_status`, 09.01.2018.

[ORG$^+$99] L. Ong, I. Rytina, M. Garcia, H. Schwarzbauer, L. Coene, H. Lin, I. Juhasz, M. Holdrege, and C. Sharp. Framework Architecture for Signaling Transport. RFC 2719 (Informational), October 1999.

[Pos80] J. Postel. User Datagram Protocol. RFC 768 (Internet Standard), August 1980.

[Pos81a] J. Postel. Internet Protocol. RFC 791 (Internet Standard), September 1981. Updated by RFCs 1349, 2474, 6864.

[Pos81b] J. Postel. Transmission Control Protocol. RFC 793 (Internet Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[Pöt16] T. Pötsch. *Future Mobile Transport Protocols: Adaptive Congestion Control for Unpredictable Cellular Networks.* Advanced Studies Mobile Research Center Bremen. Springer Fachmedien Wiesbaden, 2016.

[PS18] Tim Pritlove and Clemens Schrimpe. Cre197 ipv6. `https://cre.fm/cre197-ipv6`, January 8th 2018.

[QUI18] QUICWG. Ping/pong ambiguity. `https://github.com/quicwg/base-drafts/issues/1051`, January 15th 2018.

[Res18a] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-tls13-23, Internet Engineering Task Force, January 2018. Work in Progress.

[Res18b] Eric Rescorla. A simple go implementation of quic. `https://github.com/ekr/minq`, January 27th 2018.

[RMK$^+$96] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996. Updated by RFC 6761.

[RMMW08] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), October 2008. Updated by RFC 7350.

[Ros18] Jim Roskind. Experimenting with quic. `https://blog.chromium.org/2013/06/experimenting-with-quic.html`, January 8th 2018.

[RPB$^+$12a] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio

Honda, Fabien Duchene, Olivier Bonaventure, and Mark 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*, pages 29–29. USENIX Association, 2012.

[RPB$^+$12b] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark 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, pages 29–29, Berkeley, CA, USA, 2012. USENIX Association.

[SFR04] W.R. Stevens, B. Fenner, and A.M. Rudoff. *UNIX Network Programming*. Number Bd. 1 in Addison-Wesley professional computing series. Addison-Wesley, 2004.

[SLK06] Jongtae Song, Soon Seok Lee, and Young Sun Kim. Diffprobe: one way delay measurement for asynchronous network and control mechanism in bcn architecture. In *2006 8th International Conference Advanced Communication Technology*, volume 1, pages X000677–X000682, Feb 2006.

[Soc18] The Internet Society. State of ipv6 deployment 2017. `https://www.internetsociety.org/resources/doc/2017/state-of-ipv6-deployment-2017/`, January 11th 2018.

[Sri17] Amit Srivastava. *Performance Analysis of QUIC Protocol under Network Congestion*. PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, 2017.

[SRX$^+$04] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. RFC 3758 (Proposed Standard), May 2004.

[SSO02] J. Stone, R. Stewart, and D. Otis. Stream Control Transmission Protocol (SCTP) Checksum Change. RFC 3309 (Proposed Standard), September 2002. Obsoleted by RFC 4960.

[SSP$^+$09] Barbara Staehle, Dirk Staehle, Rastin Pries, Matthias Hirth, Andreas Kassler, and Peter Dely. Measuring one-way delay in wireless mesh networks - an experimental investigation, 10 2009.

[Ste07] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335, 7053.

[STZ$^+$17] Liyang Sun, Guibin Tian, Guanyu Zhu, Yong Liu, Hang Shi, and David Dai. Multipath IP routing on end devices: Motivation, design, and performance. *CoRR*, abs/1709.05712, 2017.

[SXT$^+$07] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, and M. Kozuka. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. RFC 5061 (Proposed Standard), September 2007.

[Tea18] The Golang Dev Team. The go programming language specification. `https://golang.org/ref/spec`, January 8th 2018.

[TH00] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991 (Informational), November 2000.

[TH14] B. Trammell and J. Hildebrand. Evolving transport in the internet. *IEEE Internet Computing*, 18(5):60–64, Sept 2014.

[Tom15] G. Tomsho. *Guide to Networking Essentials*. Cengage Learning, 2015.

[Tox03] B. Toxen. *Real World Linux Security: Intrusion Prevention, Detection, and Recovery*. Open Source Technology. Prentice Hall PTR, 2003.

[TS13] M. Tuexen and R. Stewart. UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication. RFC 6951 (Proposed Standard), May 2013.

[TSLR07] M. Tuexen, R. Stewart, P. Lei, and E. Rescorla. Authenticated Chunks for the Stream Control Transmission Protocol (SCTP). RFC 4895 (Proposed Standard), August 2007.

[TT17a] Martin Thomson and Sean Turner. Using Transport Layer Security

(TLS) to Secure QUIC. Internet-Draft draft-ietf-quic-tls-08, Internet Engineering Task Force, December 2017. Work in Progress.

[TT17b] Martin Thomson and Sean Turner. Using Transport Layer Security (TLS) to Secure QUIC. Internet-Draft draft-ietf-quic-tls-07, Internet Engineering Task Force, October 2017. Work in Progress.

[TW04] U. Trick and F. Weber. *SIP, TCP/IP und Telekommunikationsnetze: Anforderungen - Protokolle - Architekturen.* De Gruyter, 2004.

[Tü18] Michael Tüxen. A portable userland sctp stack. `https://github.com/sctplab/usrsctp`, January 27th 2018.

[VRT08] L. De Vito, S. Rapuano, and L. Tomaciello. One-way delay measurement: State of the art. *IEEE Transactions on Instrumentation and Measurement*, 57(12):2742–2750, Dec 2008.