Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

BACHELOR'S THESIS IN INFORMATICS

Evaluation of the QUIC Spin Bit for RTT Estimation

Marcel Kempf

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

Evaluation of the QUIC Spin Bit for RTT Estimation

Evaluierung des QUIC Spin Bits zur RTT-Berechnung

Author:	Marcel Kempf
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Benedikt Jaeger
	Johannes Zirngibl
Date:	March 15, 2020

I confirm that this Bachelor's Thesis is my own work and I have documented all sources and material used.

Garching, March 15, 2020	
Location, Date	Signature

Abstract

This thesis evaluates the QUIC spin bit for passive RTT measurements. The spin bit is a single bit inside the QUIC header that toggles once per RTT. The signal created by the changing spin bit can be captured by an observer to estimate the RTT. A test environment was built to emulate a network with given conditions. This test environment was used to perform RTT measurements which were then compared to ground truth values.

It was shown that the observer position in regards to delay or the bottleneck link does not seem to affect measurement accuracy. The influence of bandwidth on measurement quality does not seem to be significant either. However, with higher latency the number of RTT samples and the measurement accuracy decreases. In comparison with the TCP Timestamps option, the spin bit method produces fewer RTT samples, but the effect on the measurement error is not meaningful. It has been found that pacing in combination with large bandwidth or RTT values can result in a higher deviation between spin bit RTT estimations and the ground truth. The spin bit estimation, which is normally higher than the ground truth, is lower and stays mostly constant in this situation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The transport layer protocol TCP, which has been in use for over 40 years, is the most widely used transport layer protocol. Like many other protocols it has received few changes and updates over the years [2]. Due to the continued development of the Internet in the past decade and the resulting changes in requirements, further development has become increasingly necessary. With the amount of data rapidly increasing, network throughput is getting increasingly important. However, the TCP/TLS stack, often used for web traffic, introduces Head-of-Line blocking that reduces throughput, amongst other issues, for instance a handshake that takes too long.
The transport layer protocol QUIC (Quick UDP Internet Connections) has been introduced and announced as the successor to TCP in terms of web traffic [16]. QUIC is built on top of UDP to make sure that it runs on nearly all Internet devices. It features swift connection establishment as well as improved transport layer functionality implemented at the layer of QUIC. One of the main aspects is that QUIC encrypts the entire payload of a packet and only a small fraction of data in the header is transmitted without encryption.
The usage of QUIC has already increased in the last few years, even though it has not yet been standardized. Rüth et al. published statistics on QUIC usage [17]. Google uses QUIC for around 40 % of their traffic, mostly for its apps and content delivery. Therefore, Google accounts for around 98 % of all QUIC traffic worldwide. In October 2017, the traffic share of QUIC was around 10 %. The number of hosts supporting QUIC is also quickly increasing. As an official standard is still missing, QUIC is currently only

an IETF[1] draft.
To measure the quality of a network, passive measurements of the round-trip time (RTT) are usually performed, amongst others. Passive measurements have several advantages, for instance no data overhead or an arbitrary position inside the network. When using TCP, the TCP Timestamps option is often utilized for performing passive measurements. This method provides a large number of RTT samples throughout the entire connection. For a long time, QUIC did not offer the possibility to do a passive RTT measurement. Many methods that were used to estimate the RTT with other protocols such as TCP are no longer possible in QUIC due to the limited amount of publicly available information within the QUIC header. The encryption of a large part of the entire QUIC packet made passive measurements throughout the entire connection impossible. In 2018, the so-called spin bit has been introduced, which should enable passive measurements of the RTT [22]. One single bit in the header spins once per RTT and hereby allows observers to estimate the RTT. The performance of the spin bit in QUIC under various network conditions has been studied sparsely. There are a few papers that propose methods to make the spin bit more robust against bad network conditions by extending it by one or two bits. However, until now, only the one-bit method is contained in the IETF draft.
A transport layer protocol cannot rely on perfect network conditions like no loss or no reordering. Packet loss can have many different reasons. For example, interference or line disturbances can damage packets to such an extent that the contents can no longer be used. Packets can also be lost due to software issues, such as a TTL value dropping to zero or a packet filter dropping the packet. Packet reordering can occur if different packets get routed on multiple different paths.

1.1 Goals

Because of the importance of network latency monitoring, general knowledge of how the spin bit behaves when specific network parameters change is needed to use it properly. Furthermore, the influence of the bandwidth on measurement accuracy is important. In addition to that, the influence of the observer position could be interesting for the accuracy of the RTT measurements.
The objective of this thesis is to analyze and evaluate the method of RTT measurement using the spin bit. We would like to compare and evaluate the accuracy under different

[1] Internet Engineering Task Force

conditions and influences. For this purpose, a test environment is built, in which the emulation of various network parameters such as latency, bandwidth, loss, jitter and cross-traffic is possible. The test environment should be extensible and also be suitable for different QUIC implementations if needed.
We want to execute measurements in this environment by transferring data between client and server and measuring the RTT at the same time. Primarily, we want to vary the latency, bandwidth and position of the passive observer. The results will then be analyzed and compared to those of a measurement with TCP. At the end, we want to discover which network parameters have how much influence on the accuracy of the measurements and if the method using the QUIC spin bit provides comparable accuracy to the TCP Timestamps option method.

1.2 Outline

In Chapter 2, basic knowledge about QUIC and passive measurements is conveyed, as well as the functionality of the spin bit. In Chapter 3, related work is presented, which also deals with the spin bit or RTT measurement with TCP Timestamps option. The measurement setup and implementation is described in Chapter 4. A short explanation on how we analyze the captured traffic is given. Chapter 5 evaluates the results we obtained in the measurement environment. After looking at spin bit measurements in general, the influence of several network parameters is analyzed. In the last chapter, a conclusion of the results is given as well as an overview of possible future work.

Chapter 2

Background

This chapter provides basic knowledge about QUIC and passive RTT measurements. The different methods of passive RTT measurement when using QUIC and TCP are briefly explained.

2.1 QUIC

QUIC is a transport layer network protocol based on UDP[1]. QUIC was originally developed by Google to replace TCP as a transport protocol for HTTP [16]. QUIC provides the following main features [10]:

- Encryption and Authentication

  QUIC uses authentication and encryption to protect header and payload. Thus only the header information is publicly accessible. Apart for the handshake, the short header is used throughout the connection, containing just the content shown in Figure 2.2.

- Stream Multiplexing

  Another QUIC feature is stream multiplexing. A stream is a bidirectional flow of data within an existing connection. Since retransmissions are performed at stream-level, packet loss only impacts the corresponding stream. The use of multiple streams prevents Head-of-Line blocking. Head-of-Line blocking is a phenomenon

[1] User Datagram Protocol. a transport layer network protocol.

that may occur when using TCP. If packet loss occurs, TCP halts all streams. Only after this packet is re-transmitted and successfully received, the transmission can go on. When multiple files are transferred, this can prevent TCP from using fully received files.

- Congestion Control, Flow Control & Loss Detection

  QUIC features credit-based flow control to limit the amount of data a server receives. Congestion control in QUIC is mostly similar to TCP Reno. The used algorithm can be chosen unilaterally by the endpoints. Acknowledgments are used to detect packet loss. In general, those three mechanisms are inspired by different techniques used by TCP. IETF document `draft-ietf-quic-recovery` describes congestion control, flow control and loss detection in detail [9].

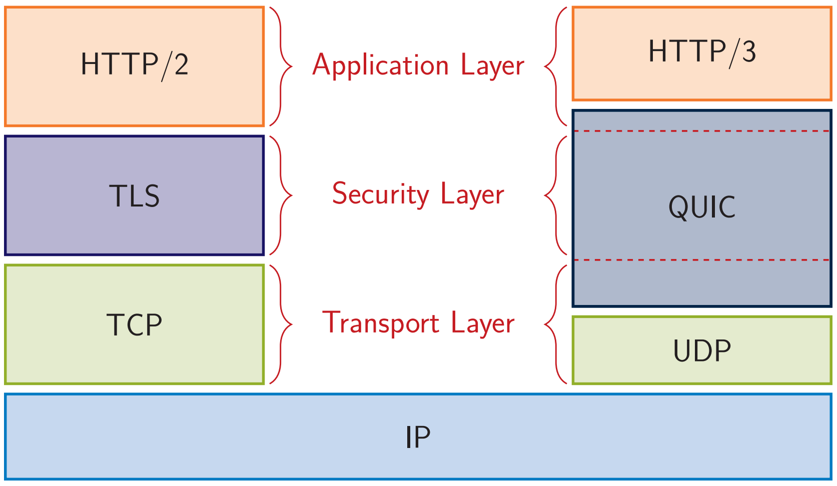


Figure 2.1: QUIC and TCP protocol stacks for HTTP

- Low Latency Connection Establishment

  An essential feature of QUIC is the combination of transport handshake and cryptographic handshake. An initial QUIC handshake needs only one RTT. This feature, in combination with the TLS 0-RTT handshake, enables QUIC to perform an actual 0-RTT connection establishment in some cases. Hence, application data can be sent already in the first roundtrip before even receiving the first packet by the server. This 0-RTT handshake cannot be used as initial handshake since it already requires some preshared information.

- User Space Implementation

  As shown in Figure 2.1, QUIC is built on top of UDP. This enables it to be implemented completely in the user space. Two big advantages of this strategy are that QUIC can run on many existing devices without any additional kernel application and it can be updated more often, since it is easier to replace an application running only in user space. Since QUIC has transport layer functionality as well as security and application layer functions implemented, it can be seen as a cross-layer protocol.

QUIC is currently still an IETF draft and has been in IETF standardization process since 2016. The draft is currently in its 27th version. The QUIC working group currently plans to submit the documents needed for standardization to the Internet Engineering Steering Group (IESG) in July 2020. The IESG is responsible for the approval of Internet standards.

2.1.1 Header

QUIC uses two different header forms, the long and short header. The long header is used to establish the connection, while the short header is used after version negotiation and exchange of the encryption keys. The reason for those two header forms is the fact that more information needs to be exchanged during connection establishment. Since this phase is only a short fraction of the entire connection, the short header helps to reduce the overhead of unnecessary information. In this work, only the short header will be considered because it is predominantly relevant for our measurements.

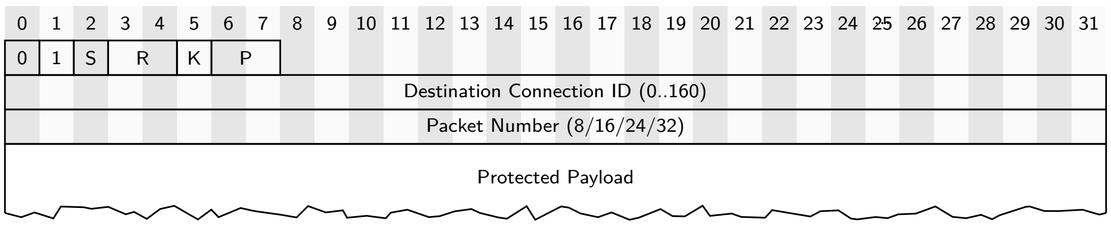


Figure 2.2: QUIC short header

The short header with its size between 2 and 25 bytes contains the following fields: The two most significant bits of a QUIC header are always `01` as shown in Figure 2.2. The third most significant bit of the header is the spin bit, which is described in detail within Section 2.2.1. The following two bits are reserved bits. The next bit indicates the key phase and is used for packet protection. The least significant two bits contain the length of the packet number. The length of the Destination Connection ID (DCID) has to be known by client and server from the time of connection establishment. After the DCID and the packet number, the encrypted payload follows.

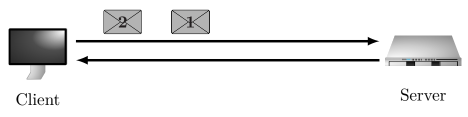
2.2 Passive RTT Measurements

Measurements of latency as well as RTT are fundamental to assess the quality of a network. The difference between active and passive measurements of the RTT is that for passive measurements the productive traffic is used whereas for active measurements extra traffic is generated. ICMP (ping) can be used for this purpose. Passive measurements do not generate overhead and ensure that no erroneous values result from other processing of the measurement traffic [21]. In some cases, measurement traffic may be preferred.
Since passive measurements can be performed at any point between transmitter and receiver, they are particularly popular among companies or network access providers. They can perform the measurements on gateway routers, for example to help customers with network problems like a slow connection or to do large measurements for research purposes. Intra- and inter-network health monitoring is also done with RTT measurement [20].

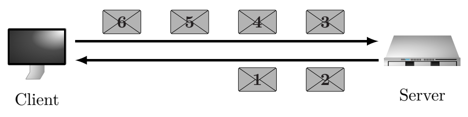
2.2.1 Latency Spin Bit

The latency spin bit, also just called spin bit, is a single bit only present in the short header of a QUIC packet. It was added to allow passive latency monitoring. The spin bit is an optional feature and must be deactivatable according to the IETF draft. It only works if both client and server have it enabled. Also, it should be randomly disabled by both client and server on one of 16 connections, so that the spin bit is effectively disabled on roughly one of eight connections [10]. A reason for that policy is not mentioned inside the QUIC IETF draft, however, another IETF document specifies privacy as the main reason [14].

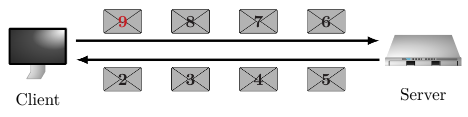
An example connection is presented in Figure 2.3. In (a), the client starts sending packets to the server. For all outgoing packets, the spin bit is not set. The server, which receives the first packet from the client in (b) replies to it having the spin bit not set for outgoing packets. After the client in (c) has received the first packet from the server, it flips the spin bit on all outgoing packets. The server, which just received the first packet with a set spin bit in (d), responds to this packet with the spin bit set. Finally, this packet in (e) arrives back at the client, and the client flips the spin bit on outgoing packets again. The mechanism described above lets the spin bit value change once per RTT. While the client inverts the incoming spin bit, the server reflects it.
Observers between client and server can now measure the time between two edges to estimate the RTT. Figure 2.4 shows a possible curve of the spin bit as an observer can



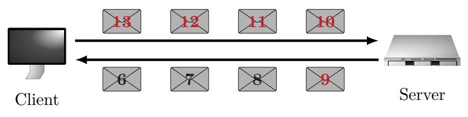
(a) The client starts sending packets



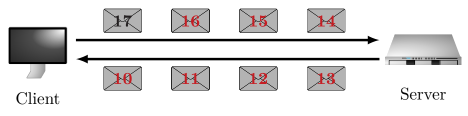
(b) The server starts responding to the received packets



(c) As soon as the first packet is received by the client the Spin Bit is inverted for outgoing packets



(d) The server inverts the Spin Bit for outgoing packets as soon as it changes on incoming packets



(e) The client also inverts the Spin Bit as soon as it changes on incoming packets

Figure 2.3: Spin bit explanation (The color (red/black) represents the spin bit (set/unset))

capture it. Every point stands for a packet with the respective spin bit value set. This can be captured both upstream and downstream, shifted by the observer upstream delay. It should be noted that the packet flows in both directions must be observed separately so that the two spin bit courses are not merged.

There is another method to measure just up- or downstream delay. For this purpose the time between spin bit transitions in both flow directions is measured, instead of focusing on just one direction. The time difference between the moment when a spin bit transition is detected in the upstream and downstream direction is naturally the time it takes for a packet to travel from the observer to the server and back. Analogous to this, the time difference between a spin bit transition downstream and upstream is the time a packet needs to travel from the observer to the client and back. This technique can be used to measure delay between two on-path observers.

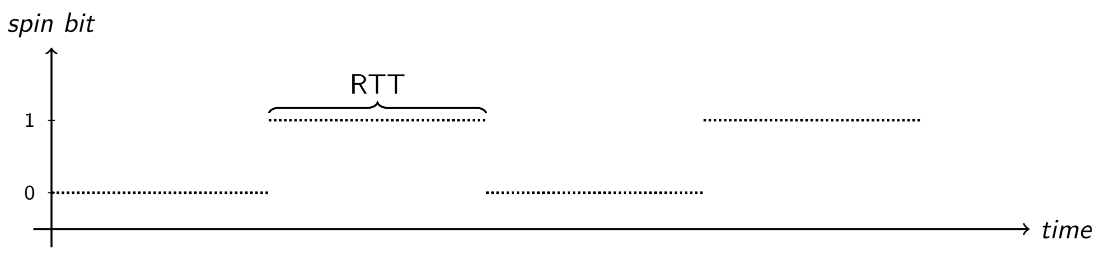


Figure 2.4: Spin bit as seen by an observer

2.2.2 TCP Timestamps Option

With TCP there are several methods to measure the RTT passively. The unencrypted header facilitates the extraction of connection information as an external observer. There are a few methods that use special packets during connection setup. These include SYN-ACK estimation and slow-start estimation. To estimate the RTT with SYN-ACK estimation, the time interval between the last SYN and the first ACK packet is measured. Those two packets are both sent from client to server. When none of the initial packets get lost or delayed, the measured time interval equals the RTT. For a slow-start RTT estimation, data must be sent from the server to the client. Additionally, if the flow does not start with at least five data packets with the first four being MSS[1], this method does not work. Since these methods only provide a minimal number of samples (one per connection) and do not work for the entire connection, it is not possible to make an accurate RTT estimate for the whole connection duration [12]. Since the RTT can vary during the connection, e.g. due to congestion, a RTT estimate from the beginning of the connection says little about the entire connection.
It is also possible to obtain the RTT by measuring the time difference from a packet and the corresponding acknowledgment. The sequence number can be used to match those two packets. However, the sequence number of a packet remains the same in case of a retransmission. When dealing with bad network conditions and a large number of retransmissions, the measurement gets inaccurate. In this situation, Phil Karn et al. showed that estimated RTT values are significantly higher than the ground truth [13].

Another method uses the TCP Timestamps option to calculate the RTT. This method provides a large number of measurements over the entire connection. Figure 2.5 illustrates its mechanism of action. We simply capture all packets and save the timestamp as well as both values TSval and TSecr. Server and client both send increasing TSval in their packets, just like a timestamp. A virtual clock is generating those values. If server or client receive a TSval they have not seen before, they send this value in the TSecr field on outgoing packets. As soon as the next TSval reaches server or client, they echo this new value on outgoing packets [7].
To measure the upstream delay Δt_1, the timestamp is stored for each TSval seen in upstream direction. Once this TSval is observed in the opposite direction as TSecr, the difference between the two timestamps Δt_1 can be calculated. The same procedure can be used to determine the downstream delay Δt_2. To calculate the entire RTT, two consecutive up- and downstream delays are added together.

[1] Maximum Segment Size

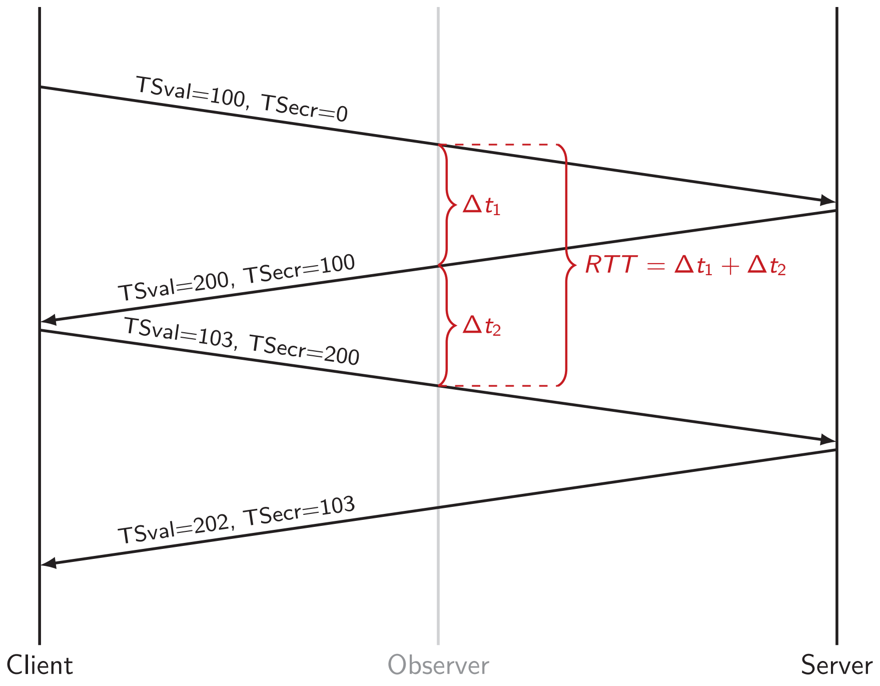


Figure 2.5: RTT measurement with TCP Timestamps option

Chapter 3

Related Work

As the QUIC protocol is quite new and has not been standardized by the IETF, there is not a lot of work on passive RTT measurement with QUIC.
De Vaere et al. fundamentally investigated the spin bit and its performance [4]. Since this was done before the spin bit was introduced into the IETF draft, they implemented the spin bit by themselves. They used further methods of spin bit implementations, including two or three bits. A theoretical analysis was done as well as measurements to determine the influence of different network conditions. In this work, we use an IETF implementation of the spin bit to evaluate the technique included into the protocol by the IETF.
Another paper by De Vaere et al. evaluates the three-bit variant of the spin bit in more detail. This variant contains a so-called valid edge counter (VEC) to simplify identification of invalid measurements. It was built into both the QUIC and TCP header [5]. The main purpose of the additional bits is to improve the detection of faulty measurements caused by reordering. They show that their three-bit variant is resistant to large amounts of loss or reordering. They focus on packet loss and reordering simulation, to evaluate the improved behavior of other spin bit variants in comparison to the one-bit solution. In this work, we also want to compare the QUIC spin bit with TCP, but we used the Timestamps option already integrated in TCP, which have been used for years for passive RTT measurements with TCP. We also want to focus on the influence of link latency and bandwidth first.
Bulgarella et al. also compared the spin bit to the three-bit variant from De Vaere et al. and another two-bit variant introduced by them, containing an additional delay bit [3]. They showed that their two-bit variant is comparable to the three-bit variant, but if packet loss occurs, it generates less valid RTT samples than the three-bit solution.

An article by Stephen D. Strowes takes a closer look on RTT measurements using the TCP Timestamps option [18]. He experimented with changing network conditions like latency, bandwidth or loss and used ICMP to retrieve his RTT ground truth. His conclusion is that the TCP Timestamps option performs well in most cases. However, the article does not contain any information on how large the influence of bandwith or observer positioning really is.
Veal et al. explain and evaluate RTT measurements with the help of the TCP Timestamps option. They performed tests in virtual and real networks to evaluate their measurement accuracy. The values were compared with server RTT values and RTT values retrieved with a self-clocking method. This self-clocking method works for symetric and asymetric routes [23]. Since the RTT measurement method with the TCP Timestamps option is by far more popular, we decided to use it as comparison in our work.

Chapter 4

Design

4.1 Measurement Setup

All measurements were performed in a test environment created in Mininet. Mininet is a widespread network emulation tool, which was used in a part of the related work [4][3]. Our network topology consists of one client, one server and four switches in between, as illustrated in Figure 4.1. NetEm[1] is used to add delay for outgoing packets at the interfaces $\texttt{delay}_1$ to $\texttt{delay}_4$. We decided to use NetEm manually with commands rather than setting the delay with Mininet for each link. This allows us to set flexible delays in the future. On the `tbf` interface, a token bucket filter is used. This filter uses the token bucket algorithm to limit the bandwidth in the client-server direction. For this purpose, virtual tokens are given to packets so that they are transmitted further. New tokens are created at a certain interval. The bandwidth is determined by this interval and the amount of tokens one packet needs. This is often coupled to the packet size. A token bucket contains all unused tokens but has a fixed maximum size (buffer size). If no tokens are available for a packet, it waits. If a packet waits longer than the buffer latency, it is dropped by the filter. This causes the token bucket filter to drop packets when the total RTT reaches the sum of the buffer latency and link latency. The link latency is the sum of all four delay parameters. We use a fixed buffer size of 1600 B and 100 ms buffer latency. The buffer size was chosen to be slightly larger than one packet to prevent bursts. We have chosen a buffer latency of 100 ms because it was also chosen by others for similar measurements.

[1] Network Emulator [8]

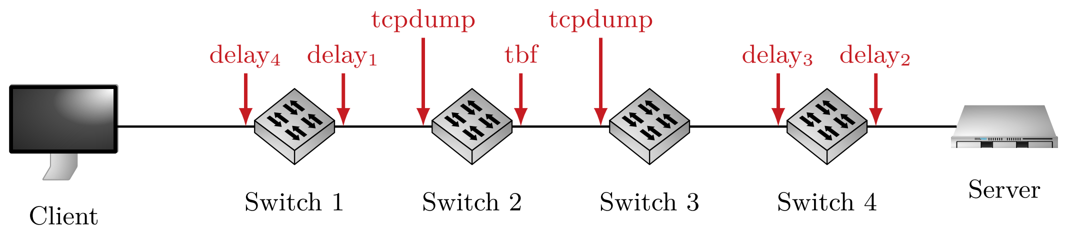


Figure 4.1: Mininet test environment setup

On Switch 2 and 3 we capture all packets with tcpdump[1] at the interface named `tcpdump`. We run tcpdump both before and after the token bucket filter to detect potential differences in measurements depending on whether the bottleneck link is between observer and client or observer and server.
An additional script on Switch 2 logs the current size of the token bucket continuously. Another script on the client is pinging the server in small time intervals (0.02 ms to 0.05 ms). Those two scripts help to quickly detect if there was an error during the measurements. Furthermore the ping is used as ground truth for RTT measurements. With the size of the token bucket filter we can use Equation 4.1 to calculate the current delay caused by the token bucket filter.

$$\text{Filter\_Delay} = \frac{\text{Bucket\_Size} \cdot 8}{\text{Bandwidth} \cdot 10^3} \tag{4.1}$$

The fraction is expanded with $\frac{8}{10^3}$ to allow the use of the usual units. The Bucket_Size is given in Bytes while the Bandwidth is noted in Mbit/s.

The test environment can be run with different parameters to vary the protocol being used (TCP or QUIC), the bandwidth and the RTT. The RTT can be modified at all four points as decribed above to place the observer (Switch 2) closer to the client or the server.

To compare measurements done with the QUIC spin bit with others using TCP Timestamps option, one single script is used to create the topology described in Section 4.1. After all link parameters are set and all scripts to capture data are started, the respective server and client applications are started too.

[1] https://www.tcpdump.org

4.1.1 QUIC Client/Server

We took a look on a few different open source QUIC implementations, as seen in Table 4.1 [6]. The implementation quic-go which was used by Bulgarella et al. does not implement the spin bit [3]. We decided to use lsquic, due to its active maintenance, its age and the spin bit support. Another advantage is that lsquic does not disable the spin bit in 1 of 16 connections until release 2.10.1 (2020-01-29). Due to this fact we can use every successful connection for spin bit measurements.

Name	**Language**	**Version**[1]	**Created**[2]	**Commits**	**Changed**[3]	**Spin Bit**
aioquic	Python	25	02/2019	764	02/2020	✗
lsquic	C	25	09/2017	235	02/2020	✓
mvfst	C++	24	04/2018	1462	02/2020	✗
ngtcp2	C	25	06/2017	1917	02/2020	✗
picoquic	C	25	06/2017	2409	02/2020	✓
quant	C11	25	12/2016	2720	02/2020	✓
quiche	Rust	25	09/2018	930	02/2020	✗
QUICker	TypeScript	20	09/2017	578	07/2019	✓
quicly	C	25	06/2017	1178	02/2020	✗
Quinn	Rust	23	04/2018	1978	02/2020	✓
quic-go	Go	22	04/2016	4210	02/2020	✗

[1] IETF draft version [2] Git repository creation date [3] Latest commit on Github

Table 4.1: QUIC implementations, retrieved from Github on 2020-02-12

For our test environment, the md5 server and client provided by lsquic are used. After the server has been started, it waits for incoming connections. As soon as a client has established a connection, it sends a file of arbitrary size to the server. When the file is transferred completely, the server calculates the md5 hash value of the file and sends it back to the client, which then terminates [19]. Those given client and server applications support various parameters to change the lsquic behavior. Several test functions to simulate packet loss or send failure are built directly into the client. A SSL certificate is required by the lsquic md5 server and therefore created by our script before the server is started. The file we send is created with the GNU/linux tool `truncate`. The size of the testfile is chosen depending on the bandwidth to reach comparable transmission durations.

4.1.2 TCP Client/Server

The TCP implementation inside the test environment is quite similar. A random file, created as for the QUIC transmission described above, is sent via netcat from client to server. After the file is completely received by the server, it terminates. We use port 443 for this connection. The traffic is unencrypted.

4.2 Analyzer

To process the pcap files from our test environment, we implemented a parser in Python. A whole connection including multiple pcap files can be analyzed by the script. We use the Python module dpkt for parsing pcap files. After detecting an UDP packet, we automatically assume that it is a QUIC packet and use our own class to retrieve QUIC header information.
For QUIC measurements, we filter all packets for short header packets and save all edge transitions in a table. The data we store in the table are the following:

- timestamp
- packet direction
- transition direction (rising/falling)
- time difference from last corresponding transition (RTT)
- number of packets since last corresponding transition

We only use the timestamp and RTT values for further processing. The other data is stored for debugging purposes.
For TCP measurements, we create two tables, one for each part of the RTT. The RTT is split into two parts, as described in Figure 2.5. In each table, we save any new TSval we see in a packet with the corresponding timestamp. For each packet, we also check if the TSecr value is already inside one of our tables as TSval. If an entry exists, we add the timestamp of this packet to the table, as well as the difference between the current time and the timestamp the value was seen first as TSval (one part if the RTT). To calculate the whole RTT, we add two consecutive samples from different tables.
Our analyzer also parses the file created by the ping script containing timestamps and RTT samples. These values are used as ground truth to calculate the RTT deviation with the following equations:

$$Deviation(t) = \frac{RTT_{spin}(t) - RTT_{ping}(t)}{RTT_{ping}(t)} \tag{4.2}$$

$$Deviation = \sum_t \left| \frac{RTT_{spin}(t) - RTT_{ping}(t)}{RTT_{ping}(t)} \right| \tag{4.3}$$

However, two measured values (spin and ping) with exactly the same timestamp are rarely available, since the Ping script delivers RTT values in a fixed interval while the interval between two spin bit RTT estimations is not controllable. This is why the closest Ping value one is used. To compare measurements better, we don't sum over all values in Equation 4.3 but skip some at the start of the measurement. In Chapter 5, we will explain in more depth why skipping samples from the first few seconds helps us to get more accurate results.
The functionality to parse the buffer size of the token bucket filter and calculate the buffer delay is also implemented for debugging purposes but is disabled by default.

Chapter 5

Evaluation

Using the measurement setup described in detail in Chapter 4, we performed measurements with different setups. In the following sections, we will present the results and compare the spin bit with TCP Timestamps option. We always used RTT measurements from the Ping script as ground truth.

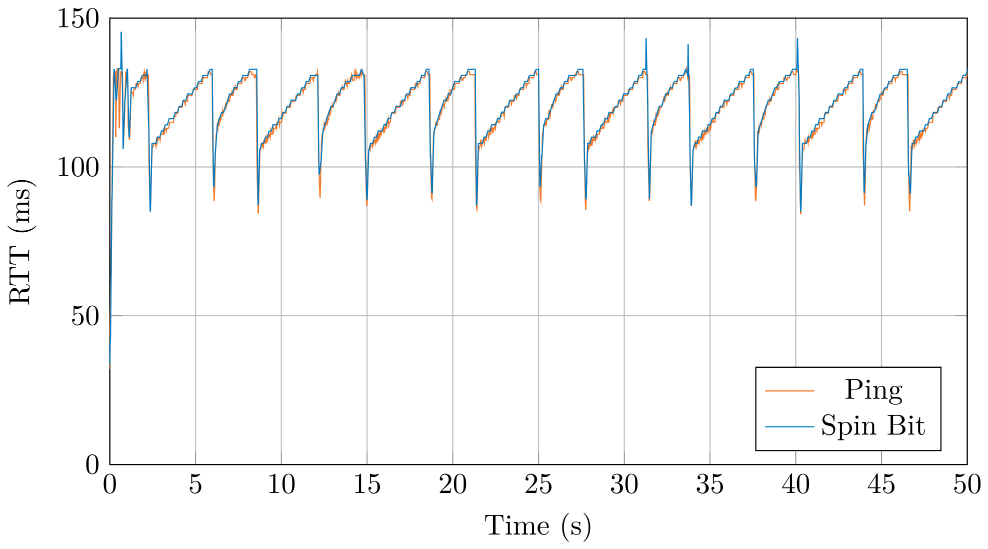


Figure 5.1: Measurement with 10 Mbit/s bandwidth, 32 ms RTT and observer placed in the middle between client and server

In Figure 5.1 a sample measurement is shown. The course of RTT, which can be noted in many of the following graphs, is characterized by repeated rapid drops and slower rises. These shapes are created by the CUBIC congestion control algorithm. The lsquic implementation uses CUBIC by default.

It can be seen immediately that there is a fundamentally larger deviation at the beginning of the connection, as there occur large changes in RTT in a short time. This is caused by the slow-start algorithm trying to avoid congestion. Throughout the rest of the connection, the measurement of the RTT using the spin bit remains close to the ground truth. Only at the peaks, where the RTT drops down, a larger deviation can be observed. The measurement using the spin bit delivers values that are repeatedly higher than the ground truth at peaks. Even at low points it is noticeable that the measurements with the spin bit produce larger RTT values than Ping. Since CUBIC is loss-based, it reduces the congestion window if loss occurs. The moment this happens is a high point in Figure 5.1. The larger deviation between spin bit and Ping RTT values at some peaks is therefore caused by loss. The loss affects the measurement only in some cases, depending on the phase of the spin bit transition.

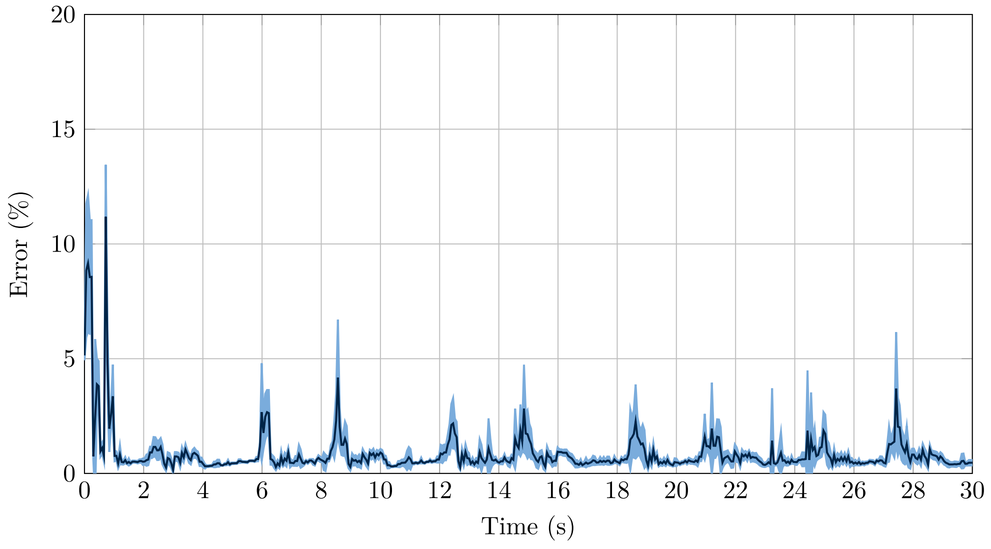


Figure 5.2: Mean average error with standard deviation of RTT measurement with spin bit (20 repetitions, 10 Mbit/s Bandwidth, 32 ms RTT)

Figure 5.2 shows the mean average error as well as the standard deviation of 10 measurements using the same parameters. Due to the fact that the measurement always are inaccurate at the beginning, we skip the first 10-30 samples in our following comparisons to achieve equal conditions. For measurements performed in real networks, a visualization as given in this figure would not be practical. Since we have nearly the same conditions in every measurement, spikes appear at the same time. In real networks, the aggregation of multiple measurements with spikes at different positions would lead to higher error throughout the entire connection. The same applies to Figure 5.6.

5.1 Influence of Observer Positioning

Since in reality an observer is never placed exactly in the middle between client and server (concerning RTT), we tested five different positions for each total RTT value, distributed equally between client and server.

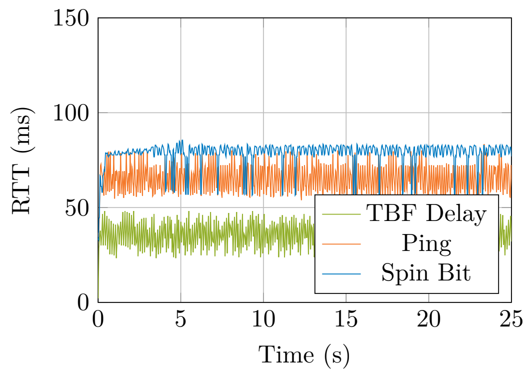
#	$\text{Delay}_{1\ \&\ 4}$ [1]	$\text{Delay}_{2\ \&\ 3}$ [2]	Average Error	Standard Deviation
0	0 ms	16 ms	15.227 %	10.50 %
1	1 ms	15 ms	0.426 %	0.72 %
2	4 ms	12 ms	0.396 %	0.72 %
3	8 ms	8 ms	0.403 %	0.79 %
4	12 ms	4 ms	0.447 %	1.04 %
5	16 ms	0 ms	0.458 %	1.11 %

[1] Delay between Client and Observer [2] Delay between Observer and Server

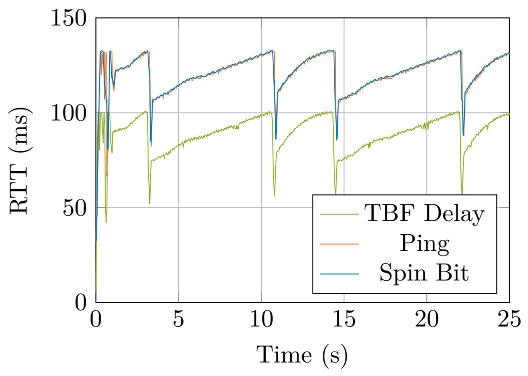
Table 5.1: Comparison of different observer positions.
10 measurements with a RTT of 32 ms were done for each position. The Observer is at switch 2.

We performed 10 measurements for each position to ensure our results are reproducible. A fixed bandwidth of 20 Mbit/s and a total RTT of 32 ms was used for all measurements. It turned out that problems with the measurement occur if the observer is located directly at the client, which is probably caused by a problem with NetEm. For this reason, we tried another sixth positioning, where the observer is 1 ms distanced from the client. The values we got from this positioning were more realistic. We performed some TCP measurements at the position directly at the client which failed as well. Our debug scripts showed that the course of the token bucket filter size and accordingly the filter delay fluctuated during the whole connection. Therefore, the RTT values from Ping as well as those estimated with the spin bit are fluctuating. This can be seen in Figure 5.3 (a). The mentioned figure shows one sample measurement for each of the 6 positions. While the orange and blue line show the RTT measured with Ping and the spin bit, the green line displays the queuing delay added to a packet by the token bucket filter, emulating the bandwidth limitation. At each moment, the sum of the filter delay and the link latency (here 32 ms) should be equal to the RTT.

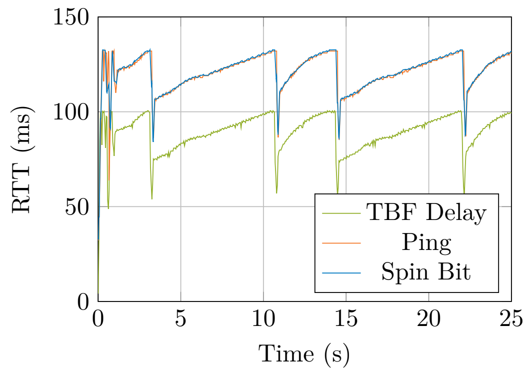
Figure 5.4 compares the accuracy of different observer positions. For each box, we considered all datapoints (about 500 per measurement) from all 10 measurements. The top whisker showing the maximum value is not displayed. The maximum value for all positions is between 10 % and 25 %. We consider the maximum as less important, since it is mostly caused by a spin bit transition happening close to a packet loss event due to congestion and between two Ping measurements. Our Ping script sampling rate may



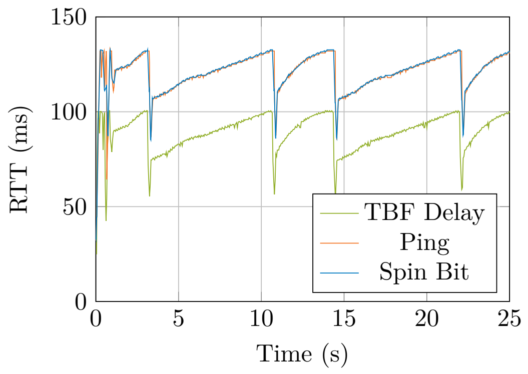
(a) Position 0



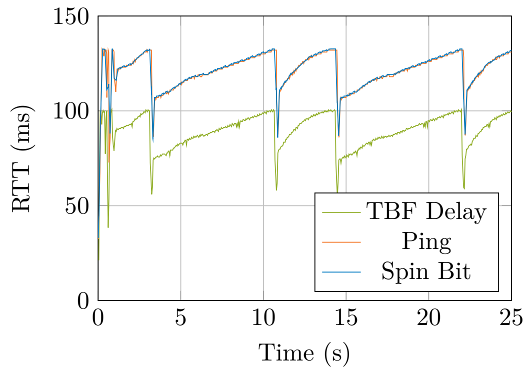
(b) Position 1



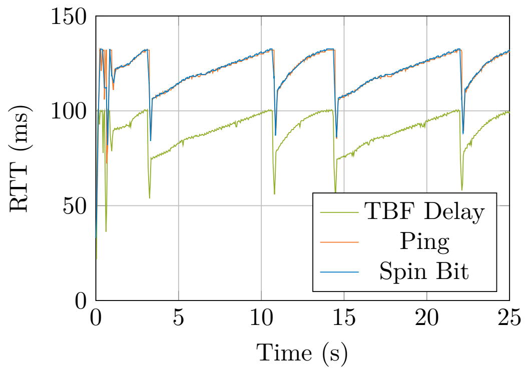
(c) Position 2



(d) Position 3

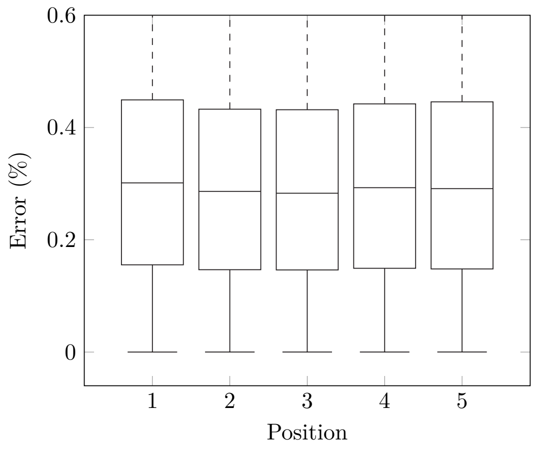


(e) Position 4

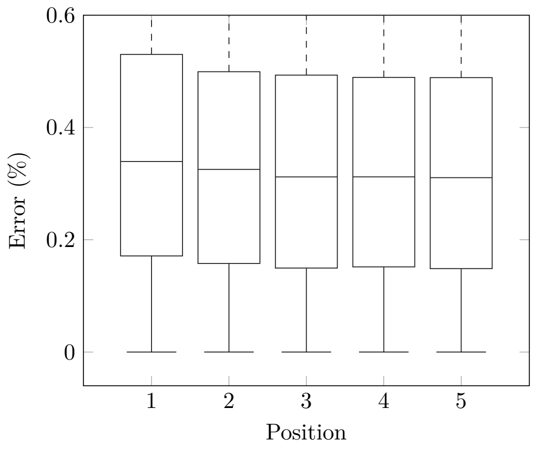


(f) Position 5

Figure 5.3: 6 sample measurements with parameters set as described in Section 5.1.



(a) Observer on Switch 2



(b) Observer on Switch 3

Figure 5.4: Distribution of measurements with different observer position.

not be high enough for these events, where the RTT drops down fast in a short time interval. A continuous ground truth could lead to a much lower maximum error.
When looking at Figure 5.3, all different positions 1-5 produce about the same error. Table 5.1 shows that the mean average deviation as well as the standard deviation is slightly increasing the closer the observer is to the server.

#	**Delay$_{1\ \&\ 4}$** [1]	**Delay$_{2\ \&\ 3}$** [2]	**Average Error**	**Standard Deviation**
0	0 ms	16 ms	15.053 %	10.38 %
1	1 ms	15 ms	0.578 %	1.50 %
2	4 ms	12 ms	0.531 %	1.43 %
3	8 ms	8 ms	0.520 %	1.47 %
4	12 ms	4 ms	0.490 %	1.21 %
5	16 ms	0 ms	0.473 %	1.11 %

[1] Delay between Client and Observer [2] Delay between Observer and Server

Table 5.2: Comparison of different observer positions.
10 measurements with a RTT of 32 ms were done for each position. The observer is at switch 3.

Since the difference is smaller than 0.2 %, we compare the results with the ones from the second tcpdump script running on Switch 3. The result of those measurements can be seen in Table 5.2. When using the data from Switch 3, the error is decreasing with the observer moving closer to the server. However, the difference is still smaller than 0.2 %. This shows that there is no influence of the observer positioning concerning delay. If we ignore position 0, the difference between left-most and right-most positioning for both observers is less than 0.1 %.

5.2 Influence of Bandwidth

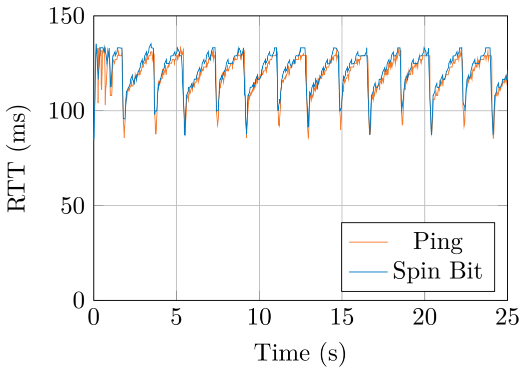
We tested different values for the bandwidth between 5 Mbit/s and 30 Mbit/s. For measurements with higher bandwidth, we noticed that Mininet does not generate reproducible results consistently. Due to experiences from other projects, we knew that Mininet may have problems with very high bandwidth values. Although 30 Mbit/s is not a high bandwidth, we decided to use only the range mentioned above to provide precise results. In Section 5.5 we will illuminate those problems in more detail.

As in the tests with the positioning, we performed 10 measurements for each combination of parameters. A fixed RTT of 32 ms was used for all these measurements. The observer was always placed centered in between client and server having all four delay parameters are set to 8 ms. We decided to use this position because the error related to positioning is comparably small. We have observed that when varying the bandwidth, it does not matter whether the bandwidth bottleneck is located before or after the observer. Only minimal differences could be detected. Therefore, the observer at switch 3 is used for the following measurements.

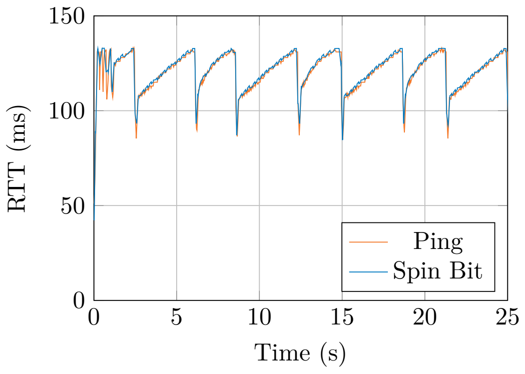
Bandwidth	**Average Error**	**Standard Deviation**
5 Mbit/s	1.823 %	1.71 %
10 Mbit/s	0.865 %	1.14 %
15 Mbit/s	0.616 %	1.41 %
20 Mbit/s	0.505 %	1.30 %
25 Mbit/s	0.640 %	2.61 %
30 Mbit/s	0.723 %	3.10 %

Table 5.3: Comparison of different bandwidths. 10 measurements were done for each bandwidth.

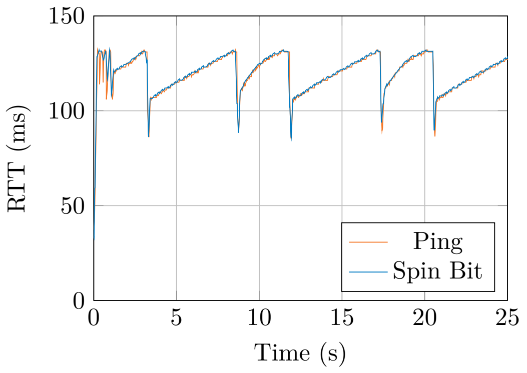
As shown in Figure 5.5, the number of peaks in a fixed period decreases with increasing bandwidth. Since the measurement accuracy at the position of a high or low point is usually significantly lower than in a phase of increase, the errors should intuitively decrease with higher bandwidth. In Table 5.3 it can indeed be seen that the error at first decreases with increasing bandwidth. In Figure 5.6, the average error for measurements with the respective bandwidth is shown. It can be noted that with increasing bandwidth, the baseline of the average error is getting lower. However, the height of peaks is increasing. The distribution in Figure 5.7 shows that for a bandwidth greater than 10 Mbit/s, the error for most values is lower than 0.5 %. A heuristic filtering out these errors at points where RTT drops down fast due to loss would help a lot to improve measurement quality.



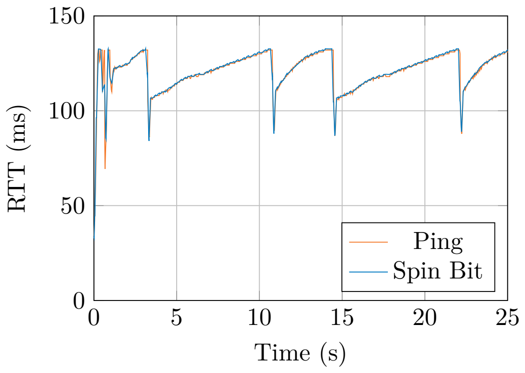
(a) Bandwidth: 5 Mbit/s



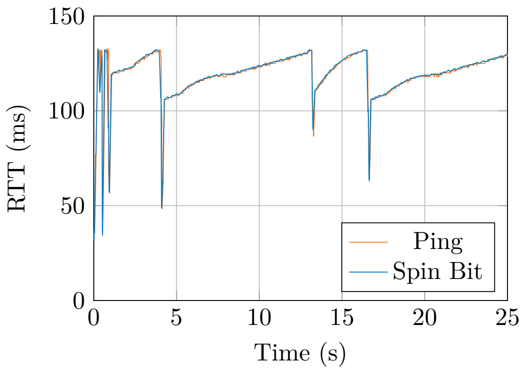
(b) Bandwidth: 10 Mbit/s



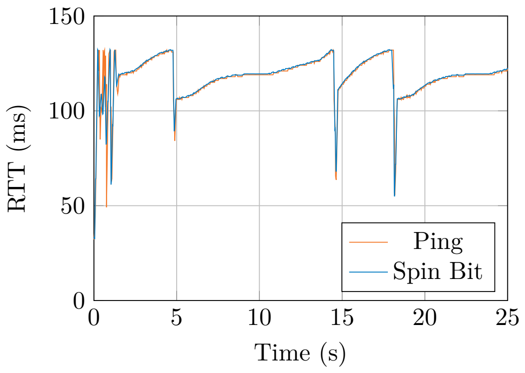
(c) Bandwidth: 15 Mbit/s



(d) Bandwidth: 20 Mbit/s

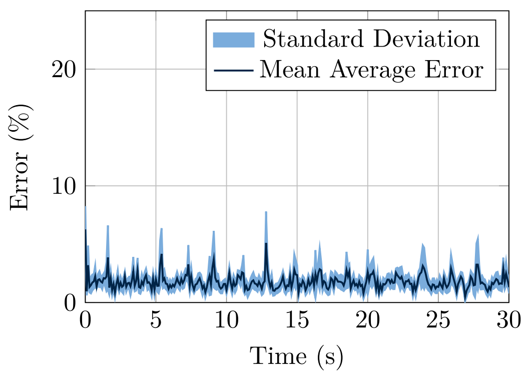


(e) Bandwidth: 25 Mbit/s

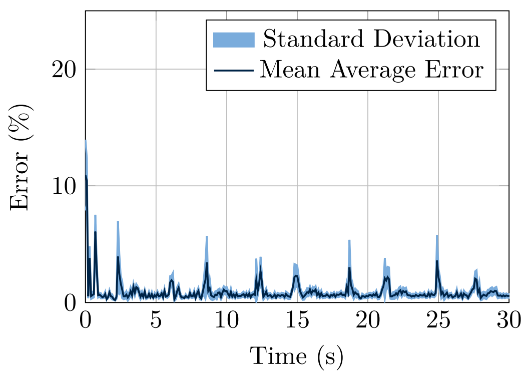


(f) Bandwidth: 30 Mbit/s

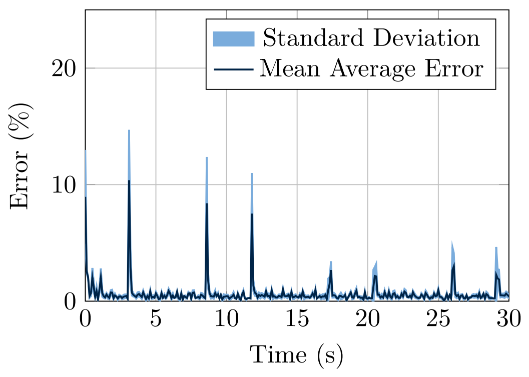
Figure 5.5: 6 sample measurements with parameters set as described in Section 5.2.



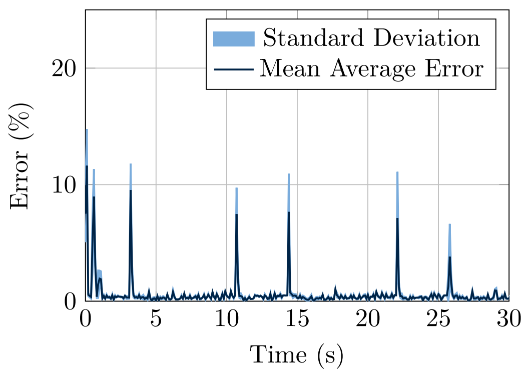
(a) Bandwidth: 5 Mbit/s



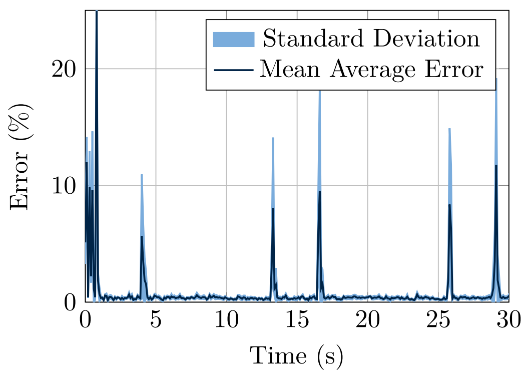
(b) Bandwidth: 10 Mbit/s



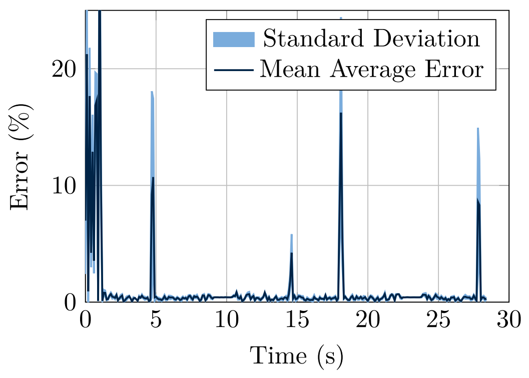
(c) Bandwidth: 15 Mbit/s



(d) Bandwidth: 20 Mbit/s



(e) Bandwidth: 25 Mbit/s



(f) Bandwidth: 30 Mbit/s

FIGURE 5.6: Average error over time with standard deviation.

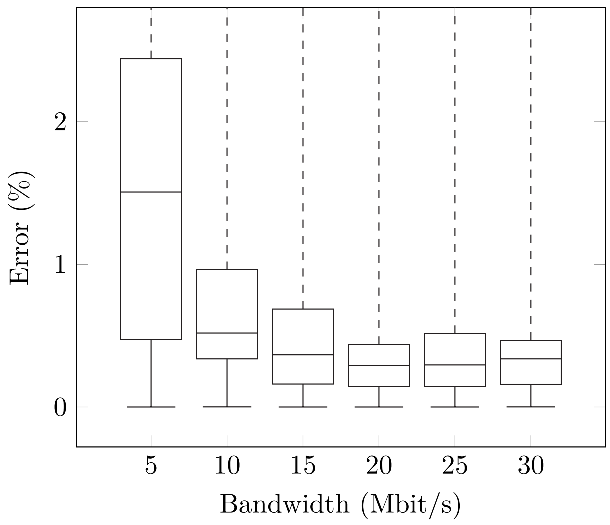


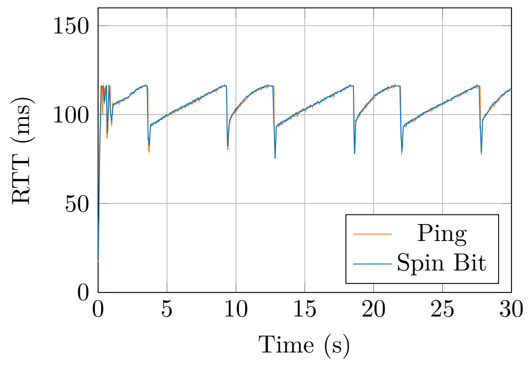
Figure 5.7: Distribution of measurements with different bandwidth

The larger deviation at the peaks may be related to the higher transmission rate resulting from the higher bandwidth. As more packets are on the path, more packets are lost when the queue at the bottleneck link reaches the maximum value. Because it is currently not possible to detect or measure packet loss in a passive measurement with QUIC, we cannot make any precise statement about this.
Our measurements showed that from a bandwidth of 10 Mbit/s upwards, the error is less than one percent on average. Thus, the bandwidth, as well as the position of the observer, does not have much impact. A bandwidth that is too low (smaller than 10 Mbit/s in our case) can cause the measurements to become inaccurate.

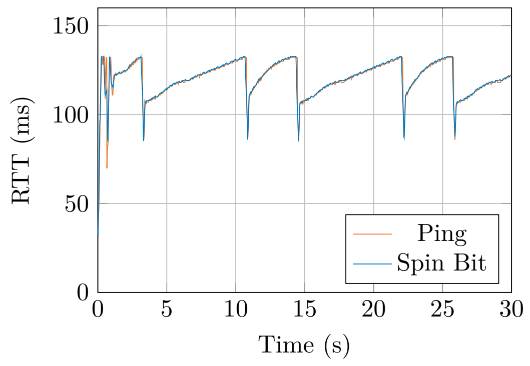
5.3 Influence of Latency

In real networks, link latency varies from values smaller than 1 ms (intranets or fiber-optic connection) to arbitrarily high values (satellite or cellular connection). In comparison to the TCP Timestamps option, the QUIC spin bit only produces two RTT samples per RTT, while the TCP Timestamps option produces one sample per TSval, e.g. one per acknowledgment. When dealing with a high link latency, this reduced amount of samples might affect the overall accuracy. In this section, we will take a closer look at the influence of latency.

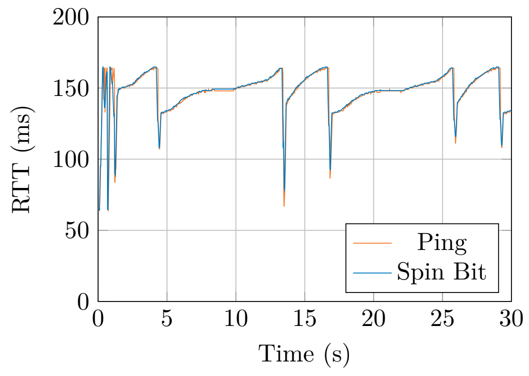
We tested different RTT values from 16 ms to 128 ms. The observer was always placed in the middle between client and server. We decided to use a bandwidth of 20 Mbit/s



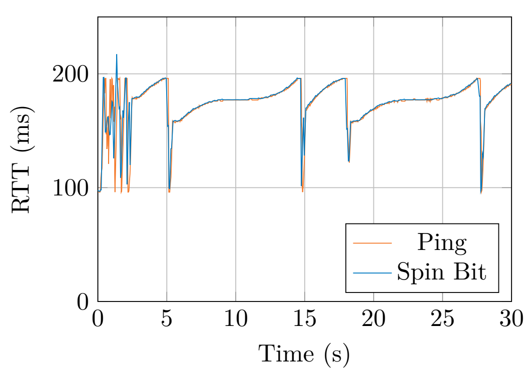
(a) RTT: 16 ms



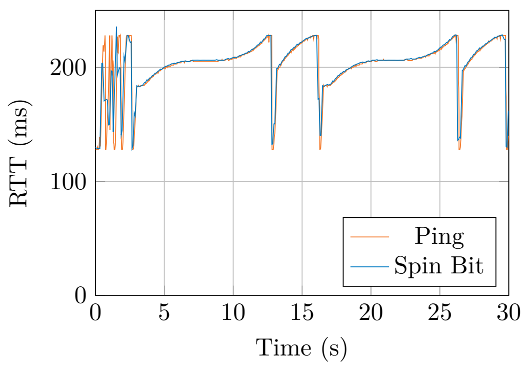
(b) RTT: 32 ms



(c) RTT: 64 ms



(d) RTT: 96 ms



(e) RTT: 128 ms

Figure 5.8: 5 sample measurements with different link latency.

RTT	Average Error	Standard Deviation
16 ms	0.448 %	1.73 %
32 ms	0.520 %	1.44 %
64 ms	0.643 %	1.94 %
96 ms	1.286 %	5.24 %
128 ms	1.625 %	6.23 %

Table 5.4: Comparison of different delay parameters. 10 measurements per row were done.

for these measurements since this was the value we got the most reproducible results with.

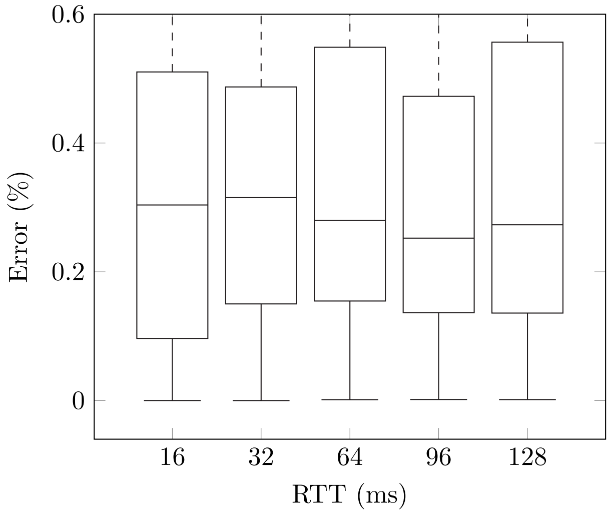


Figure 5.9: Distribution of measurements with different RTT

It is noticeable that the variation is greater at the beginning with larger RTT. This is again caused by the slow-start algorithm, which depends on acknowledgments and therefore takes longer with higher RTT [1]. However, since we do not take the deviation at the beginning into account, this should not change the average error. It can be clearly seen from the data in Table 5.4 that as RTT increases, the average error increases as well. If we compare the data with the distribution shown in Figure 5.9, it can be observed that the median is not varying more than 0.2 % over all measurements. We learned that the latency does not significantly influence the accuracy of RTT measurements using the spin bit. There are changes to the measurement caused by a higher link latency, but the observer is still able to obtain a RTT value close to the ground truth.

5.4 Comparison to the TCP Timestamps Option

In this chapter, we compare the accuracy of spin bit measurements with TCP Timestamps option measurements in selected cases.

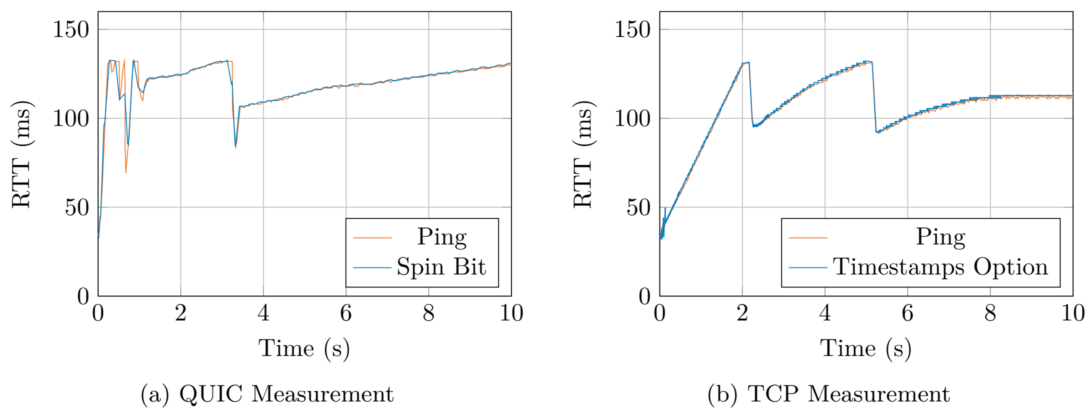
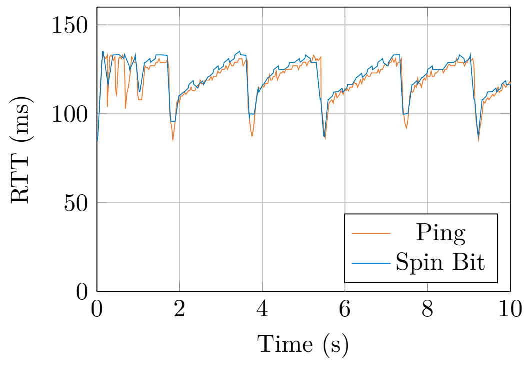


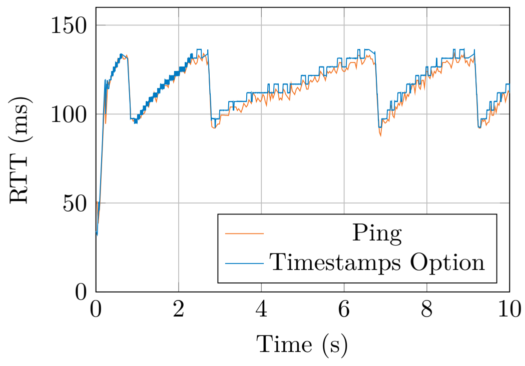
Figure 5.10: QUIC spin bit and TCP Timestamps option comparison

First, we look at a measurement with parameters that have produced an accurate and consistent measurement with QUIC. We decided to use a bandwidth of 20 Mbit/s and an RTT of 32 ms with 8 ms delay for each delay parameter. Figure 5.10 shows this measurement in comparison with a QUIC measurement performed with the same parameters. It can be observed that both measurements have a small error. The mean average error of the QUIC measurement is 0.46 %, while it is 0.47 % for the TCP measurement. The difference of both median errors is also smaller than 0.05 %. Both measurements are very accurate when conditions like these are given. In the first 10 seconds, we obtained 170 samples with our QUIC measurement and 7600 samples from TCP. As it was mentioned before, the TCP Timestamps option provides a larger number of samples compared to the QUIC spin bit.

In Figure 5.11, we take a closer look on measurements with low bandwidth. For measurements with QUIC, we experienced higher error rates during the whole connection compared to a bandwidth between 10 Mbit/s and 30 Mbit/s. It can be seen that both QUIC and TCP measurements seem to have a larger error compared to the measurements made with the parameters in Figure 5.10. Both mean average errors have increased to about 1.8 %. We also noticed that the number of samples decreased to 160 samples with QUIC and 2000 samples with TCP. However, the median of the QUIC measurement error is 1.54 %, while it is 1.86 % for TCP. These measurements show that



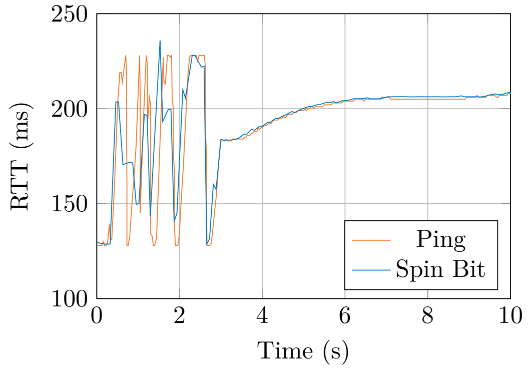
(a) QUIC Measurement



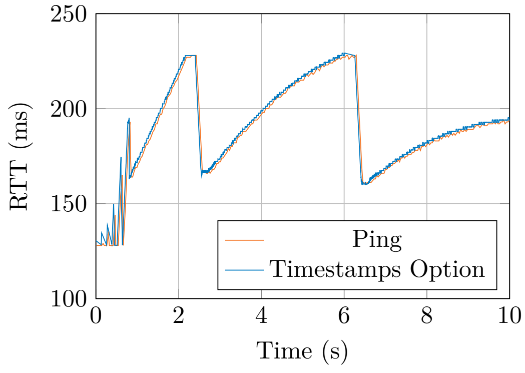
(b) TCP Measurement

Figure 5.11: Comparison of two measurements with a bandwidth of 5 Mbit/s.

the higher error with low bandwidths is not only a spin bit issue, but exists with the TCP Timestamps option method as well.



(a) QUIC Measurement



(b) TCP Measurement

Figure 5.12: Comparison of two measurements with a RTT of 128 ms.

Figure 5.12 compares QUIC and TCP in regard to a measurements with high latency. As in the chapters before, the values in the very beginning were ignored.

The data from Table 5.5 shows that our measurements with the TCP Timestamps option were more accurate than the ones performed with the QUIC spin bit. As we expected, the number of samples with QUIC drops to only 100 due to the high RTT. The number of TCP RTT samples is still at 6900. We can therefore conclude that the number of QUIC RTT samples decreases with a higher RTT, while the TCP RTT samples seem to decrease with a lower bandwidth. In the measurements performed, the accuracy of RTT estimations with the QUIC spin bit is worse than the accuracy

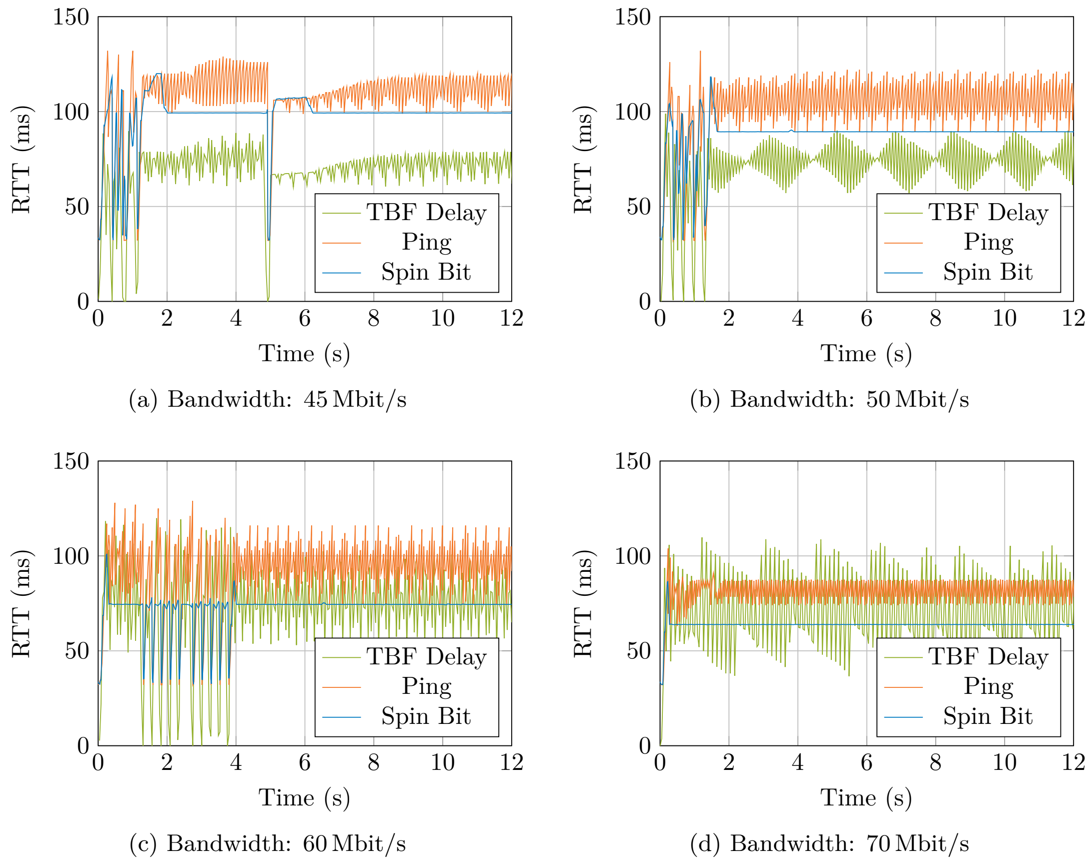
Protocol	**Average Error**	**Standard Deviation**	**Median**
QUIC	1.50 %	6.07 %	0.29 %
TCP	0.26 %	0.35 %	0.17 %

Table 5.5: Comparison of measurements with high latency.

of TCP RTT estimations. The smaller amount of samples could be a reason for this observation.

5.5 Problems with Spin Bit Measurements

During the measurements we encountered a number of problems. In this section we take a closer look at these problems and why they might occur.



(a) Bandwidth: 45 Mbit/s

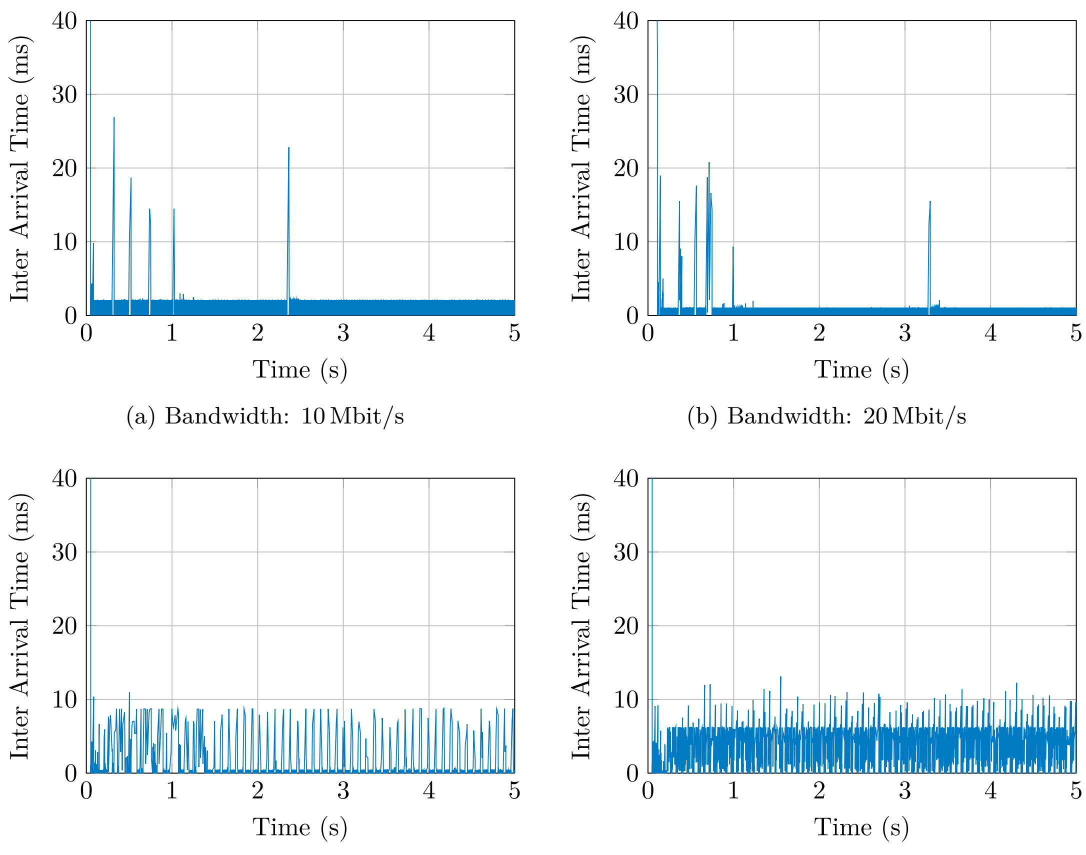
(b) Bandwidth: 50 Mbit/s

(c) Bandwidth: 60 Mbit/s

(d) Bandwidth: 70 Mbit/s

Figure 5.13: 4 sample measurements with higher bandwidth values.

In Figure 5.13 it can be clearly seen that the spin bit does not produce valid samples, when the RTT is 45 Mbit/s or higher. After a specific point of time, the RTT drops down to a constant value.
The delay caused by the token bucket filter, which is directly related to the size of the filter bucket, is fluctuating a lot in those measurements. This is caused by the token bucket filter periodically becoming fuller and emptier at small intervals. Since this is related to the sending behavior of the client, we have taken a closer look at the client.
Another instance of tcpdump running on the client provides a pcap file with the original sending interval of the packets. Since the traffic may get paced or shaped before it reaches the observer, the client was required to capture the packets to provide the data as precisely as possible. We took a closer look at the time the client needs between sending two packets. This time interval, also called inter-arrival time, is visualized over the first 5 seconds of a connection in Figure 5.14.



(a) Bandwidth: 10 Mbit/s

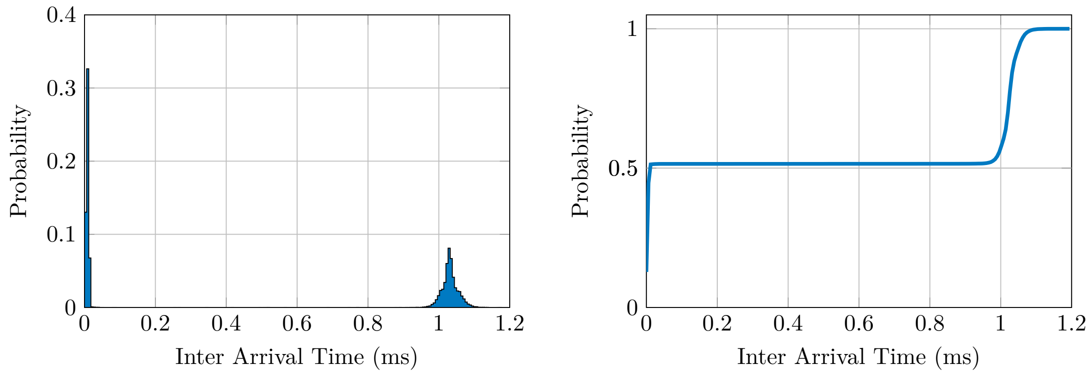
(b) Bandwidth: 20 Mbit/s

(c) Bandwidth: 50 Mbit/s

(d) Bandwidth: 70 Mbit/s

Figure 5.14: Inter-Arrival Time of QUIC Packets

Figure 5.14 shows that most of the packets are sent immediately after one another if the bandwidth is 10 or 20 Mbit/s. Only a few times the client waits for around 20 ms. This happens either during the slow-start phase or when packet loss occurs.
At bandwidths where our spin bit measurements differ from those with Ping, the inter-arrival time fluctuates much more.
A quick scan showed that for the 20 Mbit/s measurement, 99.9 % of the values are below 1.2 ms. To be able to make more precise statements, we visualized all inter-arrival times between 0 and 1.2 ms in a histogram.



(a) Probability Distribution (b) Cumulative Probability Distribution

Figure 5.15: Histogram and cumulative distribution function showing inter-arrival times between 0 and 1.2 ms. The measurement was performed with a bandwith of 20 Mbit/s and a delay of 8 ms for all four delay parameters.

Figure 5.15 shows that the inter-arrival time is either around 0 ms or 1 ms. Both values occur with a probability of about 50 %.

With the bandwidth of 10 Mbit/s, the distribution is comparable, but the inter-arrival time is either 0 or 2 ms. The likelihood of both intervals is also 50 % each. We have discovered that the two inter-arrival times alternate almost exclusively. This implies that with a bandwidth of 10 Mbit/s (or 20 Mbit/s), the client sends two packets directly after one another and then waits one (or two) milliseconds.

We also analyzed bandwidths where our spin bit measurements differ from those with Ping. 99.9 % of all inter-arrival times of the measurement with 70 Mbit/s were below 10 ms. We noticed that 97 % of all inter-arrival times are even below 0.5 ms. The distribution of inter-arrival times from this measurement is shown in Figure 5.16. The number of bins of the histogram was decreased from 200 to 20. The cumulative distribution clearly shows that the packets are usually sent directly after one another.
Unlike the low-bandwidth measurements, the inter-arrival time distribution over the packets is different here. The inter-arrival time stays close to 1 ms in the range of 100

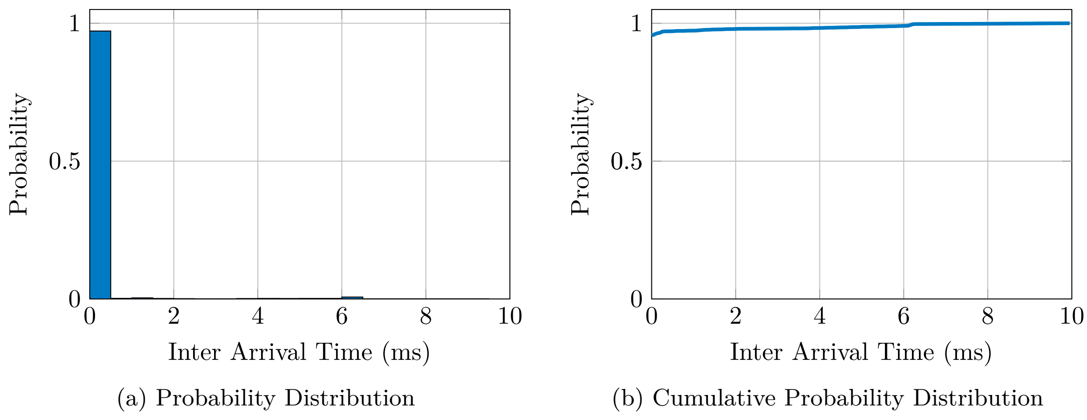


Figure 5.16: Histogram and cumulative distribution function showing inter-arrival times between 0 and 10 ms. The measurement was performed with a bandwith of 70 Mbit/s and a delay of 8 ms for all four delay parameters.

to 400 packets. After that, the inter-arrival time increases to around 6 ms for a few packets. This behavior repeats throughout the entire connection.

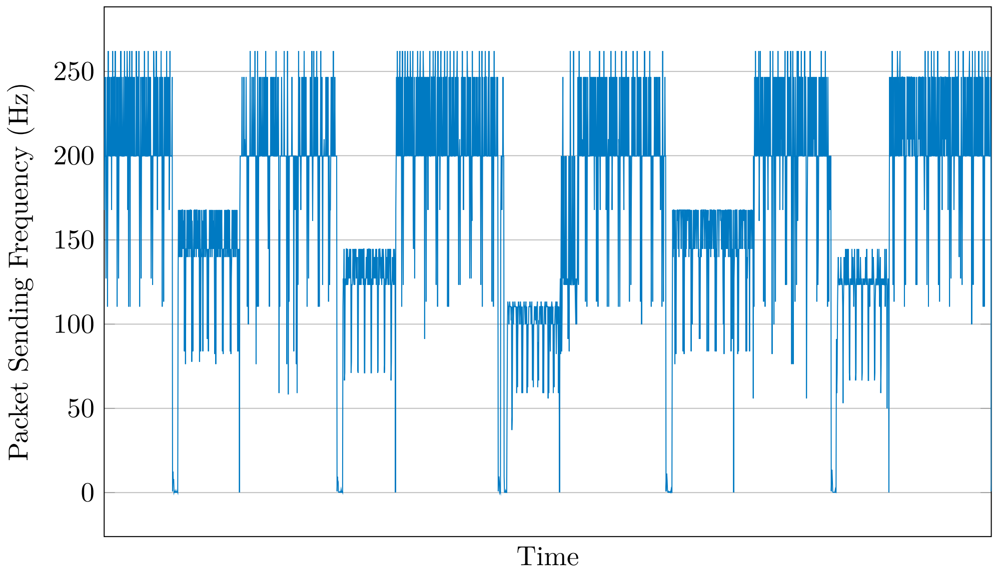


Figure 5.17: Packet sending frequency. The time interval equals the time it took to send 2400 packets. This graph shows only a clip of the whole connection.

Figure 5.17 visualizes those so-called packet trains [11]. While the Y-axis shows the transmission frequency of the packets, the X-axis shows the time progression. To make the packet trains visible, the time axis is proportional to the amount of sent packets during a connection instead of the time itself. Therefore, the X-axis does not have ticks. A packet train starts where the packet sending frequency rises from a value close to

zero. The end of a packet train is defined by the moment the frequency goes back to a value close to zero. In between that, the frequency stays at a value between 100 and 250 Hz. There are some smaller drops during packet trains where the frequency shortly drops down a bit. Those smaller drops divide the train into several parts which are sometimes called *cars*.
We expect the differences between the RTT from our spin bit measurements and from Ping to be caused by this phenomenon. Since QUIC performs pacing in the user space, every implementation can handle this differently. We assume that the pacing that lsquic implements may cause issues with the spin bit measurements when dealing with bandwidths higher than 35 Mbit/s.

Chapter 6

Conclusion

This thesis evaluates the QUIC latency spin bit in its use for passive RTT measurements. The spin bit is a bit within the QUIC header flipping once per RTT to allow passive observers estimation of the RTT. A test environment for passive measurements with QUIC or TCP was designed and implemented. The test environment has several parameters to allow modification of network conditions. It captures all the necessary data to analyze the RTT afterwards. Finally, we performed measurements inside the test environment with different simulated network conditions. We looked at several scenarios and analyzed the performance of the QUIC spin bit for RTT estimation.

We showed that the positioning of the observer between client and server has no significant influence on the measurement accuracy. The bottleneck link position, as well as the allocation of a fixed delay to multiple links before and after the observer, does not affect measurement quality.

The bandwidth does not influence the spin bit measurement either if it is not too low. In our tests, bandwidths above 10 Mbit/s led to a measurement error of less than 1 %.

In our measurements, we observed that with higher latency the measurement error increases. This higher error is likely to be caused by the fact that the number of measurement samples decreases with increasing RTT.

The general accuracy of RTT measurements with the TCP Timestamps option and spin bit estimations under good network conditions with a bandwidth of 20 Mbit/s and a RTT of 32 ms is comparable. The difference between mean average error and median error is not significant, even though the method with the TCP Timestamps option provides far more samples. Only with a delay of 32 ms per delay parameter, we noted that the spin bit delivers worse estimations than the TCP Timestamps option.

Measurements with a high deviation between the spin bit estimation and the ground

truth were analyzed in detail. We found that there are differences in pacing and inter-arrival times for measurements with a higher deviation. The assumption was made that the rapid sending of many packets in succession and the subsequent waiting leads to problems with spin bit measurements.
As a future work, more parameters can be integrated into the test environment to investigate packet loss or reordering in more detail.
It would also be possible to replace the ground truth that currently comes from the ping script. For this purpose the QUIC client can be modified to output the internally computed RTT samples, which are used for loss recovery and congestion control. These samples calculated by the client may be closer to the actual application RTT since the client can make use of encrypted packet numbers to calculate the RTT.
Another task for future work is the replacement of netcat as TCP sender and receiver with another application that uses encryption. An HTTP server that features TLS could be used instead. This would make the conditions for the comparison more equal, as we cannot ensure that TCP connections with and without TLS provide the same accuracy of RTT estimations.
At last the QUIC library could be exchanged for another one. Since we have found that the transmission behavior of lsquic causes a measurement inaccuracy in some cases, comparisons with other implementations are interesting. This way it can be determined if other QUIC implementations use different pacing methods. Analysis and modification of the lsquic code to lower the measurement error are also possible.

Chapter A

List of acronyms

HTTP	Hypertext transfer protocol. Application layer protocol for data transfer in distributed information systems. Mostly used for the transmission of web pages.
IETF	Internet Engineering Task Force. An open standards organization, dealing with technical development of the Internet.
NetEm	Network Emulator. Utility to add delay, packet loss and other characteristics to packets outgoing from a selected network interface.
pcap	Packet capture. An interface to capture network traffic.
QUIC	Quick UDP Internet Connections. General-purpose network protocol initially designed at Google.
RTT	Round trip time. The time it takes for a signal to travel from sender to receiver and back.
TCP	Transmission control protocol. Stream-oriented, reliable, transport layer protocol.
UDP	User datagram protocol. Datagram-oriented, unreliable transport layer protocol.

Bibliography

[1] Dr.Ghassan Abed, Mahamod Ismail, and Kasmiran Jumari. „Exploration and Evaluation of Traditional TCP Congestion Control Techniques“. In: *Journal of King Saud University - Computer and Information Sciences* 24 (July 2012), 145–155.

[2] Center for Applied Internet Data Analysis. *Analyzing UDP usage in Internet traffic.* `https://www.caida.org/research/traffic-analysis/tcpudpratio/`. Accessed: 2020-02-17.

[3] Fabio Bulgarella, Mauro Cociglio, Giuseppe Fioccola, Guido Marchetto, and Riccardo Sisto. „Performance Measurements of QUIC Communications“. In: *Proceedings of the Applied Networking Research Workshop.* ANRW '19. New York, NY, USA: Association for Computing Machinery, 2019, 8–14.

[4] Piet De Vaere and Mirja Kühlewind. „Adding Passive Measurability to QUIC“. MA thesis. ETH Zürich, 2018.

[5] Piet De Vaere, Tobias Bühler, Mirja Kühlewind, and Brian Trammell. „Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP“. In: *Proceedings of the Internet Measurement Conference 2018.* IMC '18. Boston, MA, USA: Association for Computing Machinery, 2018, pp. 22–28.

[6] IETF QUIC Working Group. *QUIC Implementations.* `https://github.com/quicwg/base-drafts/wiki/Implementations`. Accessed: 2020-03-12. 2019.

[7] Haijin Yan, Kang Li, S. Watterson, and D. Lowenthal. „Improving Passive Estimation of TCP Round-Trip Times Using TCP Timestamps“. In: *2004 IEEE International Workshop on IP Operations and Management.* Oct. 2004, pp. 181–185.

[8] Stephen Hemminger. „Network emulation with NetEm“. In: *Linux Conf Au* (May 2005).

[9] Jana Iyengar and Ian Swett. *QUIC Loss Detection and Congestion Control.* Internet-Draft draft-ietf-quic-recovery-25. Work in Progress. Internet Engineering

Task Force, Jan. 2020. 41 pp. URL: `https://datatracker.ietf.org/doc/html/draft-ietf-quic-recovery-25`.

[10] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport.* Internet-Draft draft-ietf-quic-transport-25. Work in Progress. Internet Engineering Task Force, Jan. 2020. 175 pp. URL: `https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-25`.

[11] Raj Jain and Shawn Routhier. „Packet Trains - Measurements and a New Model for Computer Network Traffic“. In: *IEEE Journal on Selected Areas in Communications* 4.6 (Sept. 1986), pp. 986–995.

[12] Hao Jiang and Constantinos Dovrolis. „Passive Estimation of TCP Round-Trip Times“. In: *SIGCOMM Comput. Commun. Rev.* 32.3 (July 2002), 75–88.

[13] Phil Karn and Craig Partridge. „Improving Round-Trip Time Estimates in Reliable Transport Protocols“. In: *ACM Transactions on Computer Systems* 9 (Feb. 2001).

[14] Mirja Kühlewind and Brian Trammell. *Manageability of the QUIC Transport Protocol.* Internet-Draft draft-ietf-quic-manageability-06. Work in Progress. Internet Engineering Task Force, Jan. 2020. 22 pp. URL: `https://datatracker.ietf.org/doc/html/draft-ietf-quic-manageability-06`.

[15] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, and et al. „The QUIC Transport Protocol: Design and Internet-Scale Deployment“. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication.* SIGCOMM ’17. Los Angeles, CA, USA: Association for Computing Machinery, 2017, 183–196.

[16] Jim Roskind. *QUIC Versions.* `https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34`. Accessed: 2020-03-12. 2012.

[17] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. „A First Look at QUIC in the Wild“. In: *Passive and Active Measurement.* Springer International Publishing, 2018, pp. 255–268.

[18] Stephen D. Strowes. „Passively Measuring TCP Round-Trip Times“. In: *Commun. ACM* 56.10 (Oct. 2013), 57–64.

[19] LiteSpeed Technologies. *LiteSpeed QUIC (LSQUIC) and HTTP/3 Library.* `https://github.com/litespeedtech/lsquic`. Accessed: 2020-03-12. 2019.

[20] Brian Trammell. *On The Passive Measurability of QUIC.* `https://ripe75.ripe.net/presentations/62-quic-rtt-ripe.pdf`. Accessed: 2020-03-12. 2019.

[21] Brian Trammell. *Why do we need passive measurement of round trip time?* Internet-Draft draft-trammell-why-measure-rtt-00. Work in Progress. Internet

Engineering Task Force, Aug. 2018. URL: `https://tools.ietf.org/id/draft-trammell-why-measure-rtt-00.html`.

[22] Brian Trammell and Mirja Kühlewind. *The QUIC Latency Spin Bit.* Internet-Draft draft-ietf-quic-spin-exp-00. Work in Progress. Internet Engineering Task Force, Apr. 2018. 8 pp. URL: `https://datatracker.ietf.org/doc/html/draft-ietf-quic-spin-exp-00`.

[23] Bryan Veal, Kang Li, and David Lowenthal. „New Methods for Passive Estimation of TCP Round-Trip Times“. In: *Passive and Active Network Measurement.* Ed. by Constantinos Dovrolis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 121–134.