The Search of the Path MTU with QUIC

Timo Völker
timo.voelker@fh-muenster.de
FH Münster Uni. of Appl. Sciences
Dep. of Electrical Engineering and Computer Science
Steinfurt, NRW, Germany

Michael Tüxen
tuexen@fh-muenster.de
FH Münster Uni. of Appl. Sciences
Dep. of Electrical Engineering and Computer Science
Steinfurt, NRW, Germany

Erwin P. Rathgeb
erwin.rathgeb@uni-due.de
University of Duisburg-Essen
Computer Networking Technology Group
Essen, NRW, Germany

ABSTRACT

A data sender in an IP based network is only capable to efficiently use a network path if it knows the packet size limit of the path, i.e., the Path Maximum Transmission Unit (PMTU). The IETF recently specified a PMTU discovery framework for transport protocols like QUIC. This paper complements this specification by presenting a search algorithm. In addition, it defines several metrics and shows results of analyses for the algorithm with various PMTU candidate sequences using these metrics. We integrated the PMTU discovery with our algorithm into a QUIC simulation model. This paper describes the integration and presents measurements obtained by simulations.

CCS CONCEPTS

• **General and reference** → *Metrics*; *Evaluation*; • **Networks** → *Transport protocols*; *Network simulations.*

KEYWORDS

Path MTU Discovery, Search Algorithm, Evaluation, Transport Protocol, QUIC, Simulation

ACM Reference Format:
Timo Völker, Michael Tüxen, and Erwin P. Rathgeb. 2021. The Search of the Path MTU with QUIC. In *Workshop on the Evolution, Performance and Interoperability of QUIC (EPIQ'21), December 7, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3488660.3493805

1 INTRODUCTION

A data sender in an Internet Protocol (IP) based network is only capable to efficiently use a network path if it knows the packet size limit of the path. Without this knowledge, a sender could rely on the network fragmenting IP packets that exceed the limit, but this works only with IP version 4 (IPv4) [23] and is considered harmful [15] (IPv6 [7] allows only the sender to fragment packets). Alternatively, a sender could generate only packets not exceeding the largest size that, by specification, each network node must be able to forward without fragmentation. But, sending smaller and therewith more packets increases the overhead and consumes more processing power (e.g., [22] shows impact on throughput).

EPIQ'21, December 7, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-9135-1/21/12…$15.00
https://doi.org/10.1145/3488660.3493805

The Maximum Transmission Unit (MTU) configured for a network interface limits the size of IP packets that can be sent over it. For a network path, the smallest MTU of all involved network interfaces, a.k.a. the Path MTU (PMTU), limits the size of IP packets.

The Internet Engineering Task Force recently specified a PMTU Discovery (PMTUD) framework for transport protocols like QUIC in RFC8899 [10]. The specification of QUIC in RFC9000 [14] recommends to use this framework. However, [10] does not contain details about how to search for a PMTU.

We developed a search algorithm and defined metrics to evaluate the algorithm with various PMTU candidate sequences. We implemented the PMTUD with this algorithm for QUIC and used it to examine its behavior in simulations.

After discussing related work, this paper describes how to use the PMTUD framework with QUIC in Section 3. Section 4 describes the search algorithm and examples for PMTU candidate sequences. Then, this paper defines metrics, presents the evaluation of the algorithm with each candidate sequence using these metrics and presents measurements from simulations using the search algorithm within QUIC in Section 5. Lastly, it discusses the usage of PMTUD in QUIC in Section 6 and gives a conclusion in Section 7.

2 RELATED WORK

A PMTUD mechanism for the network layer is specified for IPv4 in [8] and for IPv6 in [20]. This mechanism expects to receive a Packet Too Big (PTB) message, when a packet too large for the path was sent. As described in [3], there are multiple reasons why a PTB message might not arrive at the sender. Measurement results in [21], [17], [18], [6] and [5] show that this is a problem in the Internet. [11] even suggests to stop using this mechanism to protect against a potential attack where a spoofed PTB message is used to pretend a smaller PMTU.

A PMTUD mechanism for the layer that selects the size of IP packets, a.k.a. the Packetization Layer (PL), is specified in RFC4821 [19] and, recently, in RFC8899 [10]. Both work without a signal from the network. But, [19] has a focus on the Transmission Control Protocol (TCP), while [10] covers generic PL protocols (e.g., QUIC). However, both specifications do not contain details about how to search for a PMTU.

3 PMTUD FRAMEWORK AND QUIC

This section describes a way to use the PMTUD framework as described in RFC8899 [10] for a QUIC endpoint.

3.1 PMTU Probe

As described by RFC8899 [10], the endpoint probes whether the network path supports a specific packet size by sending an IP packet

with that size (a.k.a. a probe packet). As specified in RFC9000 [14], it builds a probe packet with a short packet header, a PING frame and PAD frames. It controls the size of the probe packet by the number of PAD frames, whose size is one byte each. The PING frame makes the packet ack-eliciting.

The following three subsections describe the events handled after sending a probe packet.

3.1.1 Acknowledgement. The receipt of an acknowledgment (ack) for a probe packet confirms that the network path supports the probed size and, therefore, lets the probe succeed. The result is known after approximately one round trip time (RTT). Note that the remote endpoint may delay the sending of the ack by up to the `max_ack_delay`.

3.1.2 Loss. To be robust against loss of probe packets that happen independent of the packet's size, the endpoint resends a lost probe packet. However, analogous to RFC8899 [10], after `MAX_PROBES` lost probe packets, it considers the probe as failed. Note, since QUIC's loss detection covers all sent packet, especially ack-eliciting packets like probe packets, the endpoint does not need the `PROBE_TIMER` as used in RFC8899 [10].

3.1.3 PTB. The receipt of a valid PTB message as response to a probe packet (validation described in RFC8899 [10]), lets the probe fail. The result is usually known in less than one RTT.

3.2 Phases

As described in RFC8899 [10], the endpoint holds a PMTU estimation. It sets the estimation initially to a safe but small value. It then increases the estimation successively until it has found the best one. RFC8899 [10] describes this in multiple phases. The main phases are Base, Search and Search Complete.

3.2.1 Base. In the Base phase, the endpoint probes for a basic PMTU size. On success, it enters the Search phase. In case of a failure, the endpoint enters an Error phase. However, since our focus is on the PMTU search, we do not consider this case.

RFC8899 [10] describes to start in the Base phase. However, as specified in RFC9000 [14], QUIC must send QUIC packets with a size of at least 1200 B when validating a path during connection initiation or migration. Since the size of the resulting IP packet is similar to what RFC8899 [10] considers a basic PMTU size, it allows to start directly in the Search phase.

3.2.2 Search. In the Search phase, the endpoint uses a search algorithm that repeatedly chooses PMTU candidates to probe. A successful probe increases the PMTU estimation to the size of the probed candidate. When the algorithm determines that it has found the best PMTU estimation, the endpoint enters the Search Complete phase.

Note, RFC8899 [10] does not describe a search algorithm. This paper presents one in Section 4.2.

3.2.3 Search Complete. The network path might change during a connection. With that, the PMTU might change as well. In the Search Complete phase, the endpoint checks whether the current PMTU estimation is still the best one. Since that is out of scope of this paper, we do not consider it further.

3.3 Congestion Control

In QUIC, all ack-eliciting packets, like probe packets, are congestion controlled. Sending such a packet uses the congestion window (cwnd) until its ack or loss. In order to save the cwnd for application data, the endpoint waits for an ack or loss signal before sending the next probe packet.

Since the loss of a probe packet is an expected result, the endpoint does not interpret it as congestion signal [14] and ignores it for the detection of a persistent congestion.

4 PMTU SEARCH

4.1 Possible PMTU Candidates

The set of possible PMTU candidates is finite.

4.1.1 Lower Bound. The size probed in the Base phase or by QUIC's path validation. Since, as specified in RFC9000 [14], QUIC assumes network paths that support an IP packet size of at least 1280 B, we use this size as lower bound.

4.1.2 Upper Bound. Due to the length field in the header, an IPv4 or IPv6 packet cannot be larger[1] than 65,535 B or 65,575 B, respectively. However, often, it is possible to determine a smaller upper bound than the maximum IP packet size. Three indications help to find a smaller upper bound.

(1) The MTU specified for the local network interface used for the network path.
(2) The MTU learned from another network device. An IPv6 endpoint could learn an MTU from a router sending a Router Advertisement [24]. An IPv4 endpoint could learn an MTU from a Dynamic Host Configuration Protocol server using the Interface MTU Option [9].
(3) The Maximum Receive Unit (MRU) declared by the remote endpoint. For instance, a QUIC endpoint may specify the maximum UDP payload size it is willing to receive during connection setup. This can be used to deduce the MRU.

The minimum of these three values is the upper bound.

4.1.3 Precision. It is possible to reduce the number of candidates between these bounds by accepting a lower precision.

Measurements showed that the most common PMTU value is 1500 B [5], which is the default MTU specified for Ethernet [12]. Other common PMTU values are based on that but reduced by the overhead of a tunnel technology. Many of these overheads have a multiple of four bytes in size (e.g., Point-to-Point Protocol over Ethernet, IPv4, IPv6). Thus, considering only PMTU candidates that are multiples of four is a reduction without losing precision in most cases.

4.1.4 Candidate Set. Practically, it might be a good choice to consider only a set of common PMTU values. However, which PMTU values are common changes over time. This means, regular measurements, as done in [5] and [1], are needed to update the set. A PMTUD analysis loses its relevance if it is based on a set of PMTU values that are no longer common. Therefore, we refrain from using only a set of common PMTU values.

[1] With IPv6 Jumbograms [4] packets can be larger than 65,575 bytes. However, we are not aware of its usage in the current Internet and therefore ignored it.

4.2 Algorithm

A simple search algorithm is one that probes one PMTU candidate after the other. This means, it starts the probe for the next candidate not before the probe for the current candidate either succeeded or failed (as described in Section 3.1).

However, when searching for the PMTU, usually an algorithm needs to consider many PMTU candidates and the probe for some candidates might fail. Detecting that a probe fails is time-consuming. To accelerate the search, we deviated from the simple algorithm as described in the following.

4.2.1 Prefer Smaller Candidate. In case of a probe packet loss, our algorithm postpones its retransmission and instead starts a probe for a smaller candidate, if available. This avoids investing too much time in a single PMTU candidate. If no smaller candidate is available, the algorithm resends the lost probe packet.

The loss of a probe packet indicates that the PMTU is smaller than the probed PMTU candidate. Therefore, the algorithm does not start a probe for a larger PMTU candidate.

4.2.2 Failed Probe. In absence of a PTB message, the algorithm considers a probe for a PMTU candidate as failed, only if `MAX_PROBES` probe packets of the size of the candidate were detected as lost. This means, the algorithm uses `MAX_PROBES` not as a total maximum of all sent probe packets as described in RFC8899 [10], but only as a maximum of probe packets sent for one PMTU candidate.

A probe for a PMTU candidate that fails, lets all other probes for larger candidates fail as well. The receipt of a validated PTB message lets all probes for candidates larger than the reported MTU within the PTB message fail. If not already probed, the algorithm continues with probing the reported MTU.

If the probe for the smallest of the currently probed PMTU candidates failed and no smaller candidates are available, the algorithm terminates with the largest successfully probed candidate as result.

4.2.3 Successful Probe. A probe for a PMTU candidate that succeeds, lets all other probes for smaller candidates succeed as well. To control the rate of probe packets sent, the algorithm continues with probing only if no probe packet of a larger candidate is outstanding.

If the probe for the largest of the currently probed PMTU candidates succeeded and no larger one is available, the algorithm terminates with this succeeded candidate as result.

4.3 Candidate Sequence

The candidate sequence specifies the order in which the algorithm probes PMTU candidates. It must choose a candidate larger than the largest successfully probed candidate and smaller than any other probed candidate with a lost probe packet. The following subsections describe some examples.

4.3.1 Linear Upward. The linear upward sequence (Up) selects one candidate after the other from a list of candidates in ascending order, starting with the second one (the first one was probed before, e.g., in the Base phase).

4.3.2 Linear Downward. The linear downward sequence (Down) selects one candidate after the other from a list of candidates in descending order, starting with the first one.

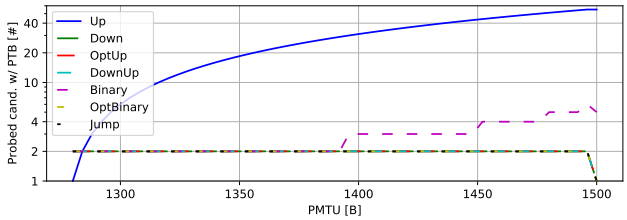


Figure 1: Number of probed PMTU candidates (PTB)

4.3.3 Optimistic Linear Upward. The optimistic linear upward sequence (OptUp) selects the largest candidate first. After that, it continues as Up.

4.3.4 Linear Down-Upward. The linear down-upward sequence (DownUp) is a combination of Down and Up. It switches between these two. It starts by selecting the largest candidate, followed by the second smallest one, followed by the second largest one, and so on.

4.3.5 Binary. The binary sequence (Binary) selects candidates from a balanced binary tree over the list of currently allowed PMTU candidates in descending order. This means, the middle element (choosing the larger one if there is no one distinct middle element) is at the root with larger candidates left from the root and smaller candidates right from the root. It selects the first candidate not selected before while traversing the binary tree in a breadth-first order.

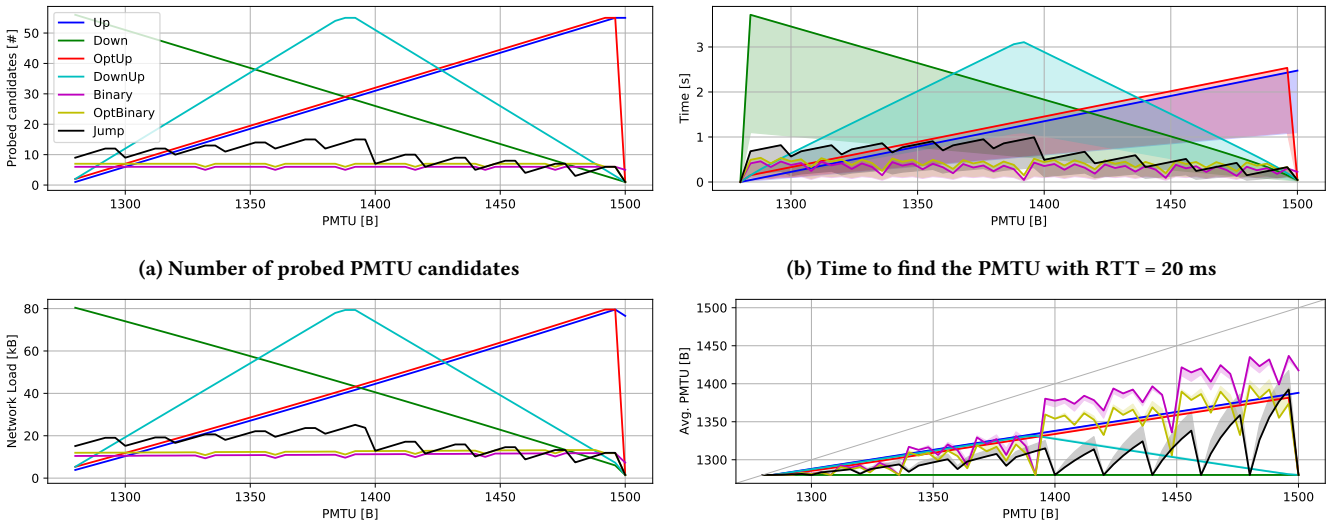
4.3.6 Optimistic Binary. The optimistic binary sequence (short: OptBinary) selects the largest candidate first. After that, it continues as Binary.

4.3.7 Jump. The jump sequence (Jump) selects candidates from a list with a reduced number of PMTU candidates. It adds further candidates to the list successively.

Without using a set of common PMTU candidates, we choose the initial list of PMTU candidates based on a distance. However, we use a smaller distance in regions where it is more likely to find the PMTU. As initial list, we use the lower and upper bound of the PMTU candidates and, within these bounds, each of the following candidates: 1300, 1400, 1420, 1440, 1460, 1480, 1500, 1520, 4000, 6500, 9000, 21,500, 34,000, 46,500 and 59,000 bytes.

Jump uses Down in the first round until a probe for a PMTU candidate succeeds. For each following round, it uses Up until the loss of a probe packet. Between two rounds, it adds new PMTU candidates if the distance d between the last successfully probed candidate and the next larger one is larger than 4. In-between these two candidates, it adds all multiples of a jump size j as new candidates in ascending order. Basically[2], we chose $j = d/5$, because it suits well to the chosen set of initial PMTU candidates where almost every candidate has a distance of $d = 4 \cdot 5^x$ to its next larger one for some $x > 0$. Up starts in each round with the next candidate after the last successfully probed one (i.e., the smallest added candidate).

[2]more precisely, we chose $j = 4 \cdot 5^{\lceil log_5(d/20) \rceil}$ to cover the cases where d is not a multiple of $4 \cdot 5$



(a) Number of probed PMTU candidates

(b) Time to find the PMTU with RTT = 20 ms

(c) Network Load by probe packets with `MAX_PROBES` = 3

(d) Average PMTU estimation during the search

Figure 2: Results from analyses without PTB

5 EVALUATION

The PMTU candidate sequence completes the search algorithm. In the following, we often omit the separation and, for example, write linear upward search algorithm or short Up to refer to the search algorithm that uses the linear upward sequence.

This section evaluates the algorithms first analytical and then by simulation and concludes with a result.

5.1 Analytical

To compare the algorithms analytical, we define four metrics and analyze the algorithms for each metric. For the analyses, we consider a network endpoint that searches the MTU of a path to another endpoint using the 55 PMTU candidates between 1280 B and 1500 B that are a multiple of four. For that, we assume no packet loss due to any reason other than the packet size.

Each of the following subsections first describes a metric and then uses it to evaluate the algorithms. They present the results using line graphs that have the actual PMTU on the x-axis and the metric value on the y-axis.

5.1.1 Number of Probed PMTU Candidates. The number of probed PMTU candidates by an algorithm already gives a rough picture of the performance.

At first, we analyze the number of probed PMTU candidates per algorithm with the assumption that the endpoint receives a PTB message reporting the correct PMTU when it sends a probe packet larger than the PMTU. Figure 1 depicts the result. It shows that all algorithms except Up and Binary either find the PMTU or trigger a PTB message with its first candidate and thus require only one or two probes. Up triggers a PTB message only with its last selected candidate. Binary triggers a PTB message with its n^{th} selected candidate for a PMTU that is smaller than $1/2^n$ of all candidates. Since the case with a PTB message is only relevant for these two algorithms and this in a predictable way, we omit it for the following metrics.

Here and for the following metrics, we present results of analyses without the assumption that the endpoint receives a PTB message. Figure 2a depicts the result for the number of probed PMTU candidates. The graph shows that the linear algorithms have a relatively high peak with 55 probed PMTU candidates.

5.1.2 Time. To be effective, a search algorithm should determine the best PMTU estimation and terminate as fast as possible. Here, we focus on the time the algorithm needs to find the best PMTU estimation. This is less than the time an algorithm needs to terminate if it requires extra steps to check that the found PMTU estimation is the best one. Note, this makes our time analysis independent of the value for MAX_PROBES.

The time to receive an ack or get a loss indication for a probe packet depends on whether the endpoint sends other ack-eliciting packets.

The receiver of a probe packet might delay the ack by up to `max_ack_delay`. However, if, for example, it received other ack-eliciting packets before, the probe packet could trigger an immediate ack. Therefore, we assume an ack time between `RTT` and `RTT + max_ack_delay`.

Without receiving an ack, the probe timeout (PTO) triggers QUIC to send an ack probe packet to request an ack (we precede the word ack here, to distinguish these probe packets from probe packets used by the PMTUD). Without assuming further packet loss, an ack that acknowledges the ack probe packet, but not the probe packet, arrives one RTT later and lets QUIC declare the probe packet as lost. However, if the endpoint sends a probe packet as the first one among other ack-eliciting packets, an ack could arrive one RTT later that acknowledges only these other ack-eliciting packets and lets QUIC consider the probe packet as lost. Thus, we assume a loss detection time between `RTT` and `RTT + PTO`.

For the PTO, we evolve the values for `smoothed_rtt` and `rttvar` with every new RTT sample as described in RFC9002 [13]. We assume a constant RTT. With $n > 0$ RTT samples, this gives

$$\texttt{smoothed\_rtt} = \texttt{RTT}$$
$$\texttt{rttvar}(n) = (3/4)^{n-1} \times \texttt{RTT}/2$$
$$\Rightarrow \texttt{PTO}(n) = (1 + 2(3/4)^{n-1}) \times \texttt{RTT} + \texttt{max\_ack\_delay}$$

We assume to have at least one RTT sample when the PMTU search starts (e.g., from the Base phase) and to have another one with each probe packet sent (the ack of either the probe packet or the later sent ack probe packet allows the RTT measurement).

Figure 2b shows the result of the analysis with `RTT` = 20 ms and `max_ack_delay` = 25 ms. The color filled area visualize possible time values. The result confirms that the high number of probed candidates of the linear search algorithms has a negative effect on the time these algorithms need.

5.1.3 Network Load. Sending probe packets produces extra load on the network. A search algorithm should produce as less load as possible. We consider the bytes of the IP packets the algorithm sends until it terminates.

Figure 2c shows the result of the analysis with `MAX_PROBES` = 3. The result corresponds to the result from the number of probed PMTU candidates.

5.1.4 Average PMTU Estimation. Data transmission can benefit from a search algorithm that increases the PMTU estimation while searching. We look at the average PMTU estimation weighted by the time an algorithm uses each estimation before it has found the best one. For example, if an algorithm estimated the PMTU with 1280 B for 20 ms and after that with 1400 B for 40 ms before it has found the best PMTU estimation, the average PMTU estimation is $(1280\ \text{B} \cdot 20\ \text{ms} + 1400\ \text{B} \cdot 40\ \text{ms})/(20\ \text{ms} + 40\ \text{ms}) = 1360\ \text{B}$. Care must be taken when interpreting this metric, because algorithms that need a long time have a better chance to achieve a high value here.

Figure 2d shows the result of the analysis. Since this metric depends on the time an algorithm needs, similar as in Section 5.1.2, the color filled area visualizes possible values for the average PMTU estimation. As the intermediate PMTU estimations cannot be equal or larger than the actual PMTU value, the result spectrum is below a diagonal line from the lower left to the upper right. This graph outlines that Down does not change its PMTU estimation before it finds the best one, not even during the long time it needs for small PMTU values.

5.2 Simulation

We use a simulation model to investigate the PMTU search inside QUIC where external events influence the search. For that we focus on the two most promising algorithms: OptBinary and Jump (see Section 5.3 for a discussion).

5.2.1 Environment. For the simulation we use the OMNeT++ simulation library [16] together with the INET network simulation model suite [2]. We implemented the PMTUD for the QUIC model [25] as described in Section 3.

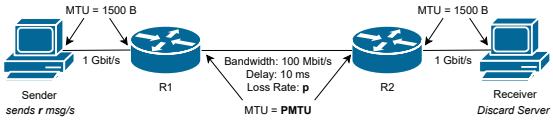


Figure 3: Network in Simulation

For all simulation runs, we use the network shown in Figure 3. In the network, the links between the hosts and the routers have a bandwidth of 1 Gbit/s without propagation delay. The link between the routers R1 and R2 is the bottleneck link. For the bottleneck link, we set a bandwidth of 100 Mbit/s and a one-way propagation delay of 10 ms. Thus, the RTT from one endpoint to the other is approximately 20 ms.

The MTU of each network interface is 1500 B. However, we reduce the MTU of the router interfaces towards the bottleneck link as required to modify the MTU of the path between the sender and the receiver. The routers drop packets larger than the MTU of the outgoing interface without notifying the sender with a PTB message.

For the simulations, we configured the sender to discover the PMTU with using 1280 B as lower bound of the PMTU candidates. The sender determines 1500 B as upper bound due to the MTU of its network interface.

5.2.2 Data Transmission. As shown by the color filled areas in Figure 2b, the time the PMTU search needs to find the best PMTU estimation varies as described in Section 5.1.2. To measure this dependence, we use the simulation.

The Search phase starts with one RTT sample from the Base phase. We repeat the simulation with various values for the message send rate of the application on the sender r between 0 and 2000 messages per second each with 1 kB in size.

Since the exact time when the sender sends a packet might influences the result, instead of using a constant distance between two messages, we use an exponentially distributed random distance with r as mean. To compensate random effects, we repeat the simulations for each r 100 times.

Note that r sets only the send rate of the application on the sender. QUIC decides when to send out a packet. We configured QUIC to send packets as soon as possible even if the available application data do not fill the packet completely to get the same packet send rate for a small r. For a $r \geq 500$ QUIC's New Reno congestion control blocks outgoing packets for a short period at the start of the connection where the cwnd is low. This leads to a smaller packet send rate compared to r, but, however, influences the result only marginal.

We measure the time the sender needs to determine the PMTU. To show the decreasing loss detection time, we run the simulation with a PMTU of 1284 B. To also show the decreasing ack delay, additionally, we run the simulation with a PMTU of 1496 B for OptBinary and a PMTU of 1396 B for Jump.

Figure 4 shows the result. The dashed horizontal lines mark the largest and the smallest time as determined by the analysis described in Section 5.1.2. The solid lines show the mean of the time the algorithms need to find the PMTU over all 100 runs for each r with a 95 % confidence interval (shown by vertical error bars).

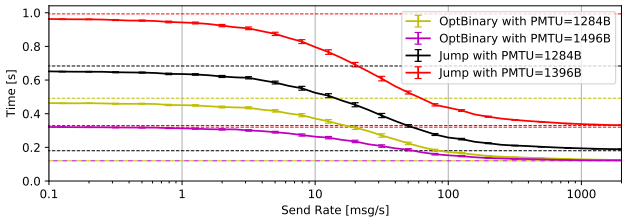


Figure 4: Simulation with Data Transmission

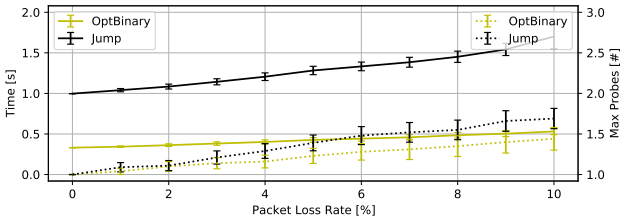


Figure 5: Simulation with Packet Loss

These lines show how the time decreases between the bounds due to the reduced loss detection time or ack delay for increasing r.

5.2.3 Packet Loss. Since the PMTUD uses the loss of a probe packet as indication that the path does not support its size, the loss of a probe packet not larger than the PMTU leads to a false negative indication. We use the simulation to measure the influence of packet loss to the time needed to find the PMTU and to the number of probe packets required to send before receiving an ack.

We set the PMTU so that both algorithms send the most probe packets that may result in a false negative. This is 1496 B for OptBinary and 1396 B for Jump. We repeat the simulation with various values for the packet loss rate p between 0 and 10 %. To compensate random effects, we repeat the simulations for each p 100 times.

Figure 5 shows the result. Shown by the solid lines, the time the algorithms need increases only moderate for larger p. Shown by the dashed lines, the required number of probe packets keep below 2, which shows that 3 is a reasonable value for `MAX_PROBES`.

5.3 Result

Looking at the results of the four metrics in Section 5.1, Binary shows the best performance overall followed by OptBinary and Jump. We prefer the optimistic variant of Binary, because it selects the upper bound of PMTU candidates first and, therewith, has the potential to immediately either find the best PMTU estimation or trigger a PTB message. The results of the simulations show that both algorithms perform well even with external events like data transmission or packet loss.

In general, we recommend to use OptBinary. But, if a small but still accurate set of currently common PMTU candidates is known, Jump configured to initially use this set potentially outperforms Binary.

For our analyses we limited the number of PMTU candidates by an upper bound of 1500 B. We did this to have a presentable small number of candidates and because most of the endpoints in the current Internet advertise a TCP Maximum Segment Size (MSS) that correspond to an MTU equal or smaller than 1500 B [5]. However, we believe that our results apply to cases with larger upper bounds as well.

6 DISCUSSION

This section provides a brief discussion about the usage of PMTUD in QUIC.

6.1 When to use PMTUD in QUIC?

QUIC should cache a discovered PMTU. With that, QUIC can benefit from using PMTUD in connections that last long enough for the search algorithm to terminate. In this case, QUIC can cache the discovered PMTU and use it for future connections to the same endpoint.

Since QUIC itself cannot know how long the connection lasts, the application should decide whether to use PMTUD.

6.2 How to use a cached PMTU estimation?

A cached PMTU estimation is a good indication for the PMTU. However, QUIC should not use it blindly in a new connection. For example, the network could route the new connection on another path with a PMTU less than the cached value. Therefore, QUIC should still use PMTUD.

The PMTUD can utilize the cached PMTU estimation by using it temporarily as upper bound of the PMTU candidates. With a search algorithm that probes the largest candidate first, the PMTUD sets the PMTU estimation to the cached size and terminates after approximately one RTT if the network path supports this cached size. Even if it does not, the algorithm continues with the benefit of a potential smaller upper bound. The PMTUD should reset the upper bound to the initially determined value when entering the Search Complete phase. This allows to detect an increased PMTU.

7 CONCLUSION AND OUTLOOK

This paper complements the PMTUD framework as specified in RFC8899 [10] by presenting a search algorithm. In addition, it defines metrics to evaluate the algorithm with different candidate sequences in an analytical way.

We implemented PMTUD with the search algorithm in a simulation model for QUIC. Results from simulations were presented showing that the algorithm with the suggested candidate sequences performs well even with external events.

We are planning to publish work about the validation of the PMTU estimation in the Search Complete phase. Another field for future work are the consequences of having probe packets congestion controlled. For example, it allows to send additional probe packets (note that our algorithm supports this). However, having multiple probe packets or, as another example, one large probe packet outstanding uses more of the cwnd and, therewith, increases the risk of delaying the transmission of application data.

REFERENCES

[1] Shane Alcock and Richard Nelson. 2010. *An Analysis of TCP Maximum Segment Sizes.* Technical Report. Retrieved October 20, 2021 from https://wand.net.nz/sites/default/files/mss_ict11.pdf

[2] Zoltan Bojthe, Levente Meszaros, György Szászkő, Rudolf Hornig, Andras Varga, and Attila Török. 2021. *INET Framework.* Retrieved June 26, 2021 from https://inet.omnetpp.org/

[3] Ron Bonica, Fred Baker, Geoff Huston, Bob Hinden, Ole Trøan, and Fernando Gont. 2020. IP Fragmentation Considered Fragile. RFC 8900. https://doi.org/10.17487/RFC8900

[4] David A. Borman, Dr. Steve E. Deering, and Bob Hinden. 1999. IPv6 Jumbograms. RFC 2675. https://doi.org/10.17487/RFC2675

[5] Ana Custura, Gorry Fairhurst, and Iain Learmonth. 2018. Exploring Usable Path MTU in the Internet. In *2018 Network Traffic Measurement and Analysis Conference (TMA)* (Vienna, Austria) *(TMA '18)*. IEEE, 1–8. https://doi.org/10.23919/TMA.2018.8506538

[6] Maikel de Boer and Jeffrey Bosma. 2012. *Discovering Path MTU Black Holes on the Internet Using the RIPE Atlas.* Technical Report. Retrieved October 21, 2021 from https://www.nlnetlabs.nl/downloads/publications/pmtu-black-holes-msc-thesis.pdf

[7] Dr. Steve E. Deering and Bob Hinden. 2017. Internet Protocol, Version 6 (IPv6) Specification. RFC 8200. https://doi.org/10.17487/RFC8200

[8] Dr. Steve E. Deering and Jeffrey Mogul. 1990. Path MTU discovery. RFC 1191. https://doi.org/10.17487/RFC1191

[9] Ralph Droms and Steve Alexander. 1997. DHCP Options and BOOTP Vendor Extensions. RFC 2132. https://doi.org/10.17487/RFC2132

[10] Gorry Fairhurst, Tom Jones, Michael Tüxen, Irene Rüngeler, and Timo Völker. 2020. Packetization Layer Path MTU Discovery for Datagram Transports. RFC 8899. https://doi.org/10.17487/RFC8899

[11] Matthias Göhring, Haya Shulman, and Michael Waidner. 2018. Path MTU Discovery Considered Harmful. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 866–874. https://doi.org/10.1109/ICDCS.2018.00088

[12] IEEE. 2018. IEEE Standard for Ethernet. *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015* (August 2018).

[13] Jana Iyengar and Ian Swett. 2021. QUIC Loss Detection and Congestion Control. RFC 9002. https://doi.org/10.17487/RFC9002

[14] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. https://doi.org/10.17487/RFC9000

[15] Christopher A. Kent and Jeffrey C. Mogul. 1987. Fragmentation Considered Harmful. In *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology* (Stowe, Vermont, USA) *(SIGCOMM '87)*. Association for Computing Machinery, New York, NY, USA, 390–401. https://doi.org/10.1145/55482.55524

[16] OpenSim Ltd. 2021. *OMNeT++.* Retrieved June 26, 2021 from https://omnetpp.org/

[17] Matthew Luckie, Kenjiro Cho, and Bill Owens. 2005. Inferring and Debugging Path MTU Discovery Failures. In *Internet Measurement Conference 2005 (IMC 05)*. USENIX Association, Berkeley, CA. https://www.usenix.org/conference/imc-05/inferring-and-debugging-path-mtu-discovery-failures

[18] Matthew Luckie and Ben Stasiewicz. 2010. Measuring Path MTU Discovery Behaviour. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (Melbourne, Australia) *(IMC '10)*. Association for Computing Machinery, New York, NY, USA, 102–108. https://doi.org/10.1145/1879141.1879155

[19] Matt Mathis and John Heffner. 2007. Packetization Layer Path MTU Discovery. RFC 4821. https://doi.org/10.17487/RFC4821

[20] Jack McCann, Stephen E. Deering, Jeffrey Mogul, and Bob Hinden. 2017. Path MTU Discovery for IP version 6. RFC 8201. https://doi.org/10.17487/RFC8201

[21] Alberto Medina, Mark Allman, and Sally Floyd. 2005. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM Comput. Commun. Rev.* 35, 2 (April 2005), 37–52. https://doi.org/10.1145/1064413.1064418

[22] Kazuho Oku and Jana Iyengar. 2020. *Can QUIC match TCP's computational efficiency?* Retrieved September 23, 2021 from https://www.fastly.com/blog/measuring-quic-vs-tcp-computational-efficiency

[23] Jon Postel. 1981. Internet Protocol. RFC 791. https://doi.org/10.17487/RFC0791

[24] William A. Simpson, Dr. Thomas Narten, Erik Nordmark, and Hesham Soliman. 2007. Neighbor Discovery for IP version 6 (IPv6). RFC 4861. https://doi.org/10.17487/RFC4861

[25] Timo Völker, Ekaterina Volodina, Michael Tüxen, and Erwin P. Rathgeb. 2020. A QUIC Simulation Model for INET and its Application to the Acknowledgment Ratio Issue. In *2020 IFIP Networking Conference (Networking)* (Paris, France) *(IFIP '20)*. IEEE, 737–742.