Packet Too Big Detection and its Integration into QUIC

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

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

***Abstract*—A communication over an Internet Protocol (IP) based network fails if an endpoint sends packets that are too big to reach their destination and if the sender is unable to detect that. The node on the path that drops these packets should respond with a Packet Too Big (PTB) message. However, multiple scenarios exist in which the sender will not receive a PTB message. Even if it does, it refrains from using the information in case it suspects that a potential attacker forged the message. In particular, we are not aware of any implementation of the secure transport protocol QUIC (e.g., used by HTTP/3) that processes PTB messages. In this paper, we present a novel parameterizable PTB detection algorithm for reliable transport protocols that does not depend on PTB messages. We further describe how to integrate our algorithm into QUIC, present results from an evaluation using the algorithm within a QUIC simulation model and, based on these results, suggest concrete parameter values.**

***Index Terms*—PMTUD, PTB, black hole, transport protocol, QUIC, simulation**

I. Introduction

To efficiently use a network path in an Internet Protocol (IP) based network, a sender should know the packet size limit of the network path. Without this knowledge, it might send packets smaller than necessary which is inefficient. Or it might send packets that are too big for the path in which case they will not reach their destination[1]. The connection fails if the sender is unable to detect that and continues to send oversized packets.

The Maximum Transmission Unit (MTU) configured for a network interface limits the size of outgoing IP packets. The packet size limit for a network path is the Path MTU (PMTU), i.e., the smallest MTU of all involved outgoing network interfaces on a path.

A network node that drops a packet, because it is larger than the MTU of its outgoing network interface, should respond with a Packet Too Big (PTB) message (i.e., an Internet Control Message Protocol (ICMP) Destination Unreachable Message with code 4 [2] or an ICMP for IP version 6 (IPv6) Packet Too Big Message [3]). PTB detection is simple for a sender if it receives a PTB message and if it trusts the message. However, because that is not necessarily the case, a PTB detection must function even without a PTB message.

[1]With IP version 4 fragmentation, packets can reach their destination even if they were sent too big. However, a sender should avoid IP fragmentation, as it is considered harmful [1].

[4] describes multiple scenarios where an endpoint sending packets that are too big receives no PTB messages, which is known as PMTU black hole. Measurement results in [5], [6], [7], [8] and [9] show that PMTU black holes exist in the Internet. On the other hand, an attacker can forge a PTB message to force an endpoint to send small packets. A sender receiving a PTB message refrains from using the message for PTB detection if it does not trust the information. As an extreme case, a sender that trusts only cryptographically secured information will not use PTB messages. For example, we are not aware of any implementation of the secure transport protocol QUIC [10] that processes PTB messages.

The IETF recently specified a PMTU Discovery (PMTUD) for transport protocols in [11]. The idea is to prevent an endpoint from unintentionally sending packets that are too big by restricting their size using a PMTU estimation that is equal or smaller than the actual PMTU. The discovery starts with a PMTU search where it successively increases its estimation towards the actual PMTU. A PTB message is not required for this process. However, a PMTU can change. After a PMTU decrease, an endpoint might end up sending packets that are too big. [11] does not describe a way to detect that without a PTB message.

In this paper, we present a novel parameterizable algorithm for reliable transport protocols to detect when sending oversized packets without a PTB message. We choose QUIC [10] as an example to describe how to integrate our algorithm into a transport protocol and use a QUIC simulation model to elaborate it with various parameter values.

After describing PMTUD and the complexity with PMTU change in Section II, Section III presents our PTB detection algorithm. Section IV introduces QUIC and presents results from PMTUD tests with QUIC implementations. The section further describes how to integrate the PTB detection algorithm in QUIC and Section V describes our elaboration of the algorithm with network simulations. Section VI gives a conclusion and an outlook on future work.

II. Path MTU Discovery

The PMTUD as specified in [11] assumes a reliable transport protocol that can choose the packet size and avoids IP fragmentation. It estimates the PMTU with a value equal or

smaller than the actual PMTU. It sends packets larger than its estimation to probe if the network path supports the size. On acknowledgment of receipt of such a PMTU probe packet, it increases its estimation.

A. Phases

The PMTUD consists of the following three main phases.

1) Base: In this initial phase, the PMTUD sets its estimation to a base PMTU (e.g., 1280 B). It then confirms its estimation by sending a PMTU probe packet of this size. After an acknowledgment for the probe packet, it starts the Search phase.

2) Search: During the Search phase, the PMTUD sends PMTU probe packets larger than its current estimation. It increases its estimation when receiving an acknowledgment for a probe packet. The search completes, if it increased its PMTU estimation to the largest considered PMTU candidate (e.g., as reported by a received PTB message) or if it receives repeatedly no acknowledgment for a probe packet with a size of the next larger PMTU candidate than the current PMTU estimation.

3) Complete: After the best PMTU estimation has been found, in the Complete phase, the task is to verify that the current PMTU estimation is still the best one.

If the PMTUD assumes that its estimation is too small, it switches to the Search phase with considering PMTU candidates that are larger than the current estimation only. If it assumes that its estimation is too large, it switches to the Base phase with considering PMTU candidates that are equal or smaller than its current estimation only.

B. PMTU Change

The PMTU can change when the path changes. The Internet, for example, is a complex network that may provide multiple paths from one endpoint to another. This is known as path diversity [12]. Routers decide which path packets use. Usually, they use the same path for packets that belong to a flow [13] (e.g., a transport protocol connection). However, routers sometimes decide to change the path, for example, due to a path outage [14].

1) Increase: A PMTU increase provides the ability to send larger packets. Other than that, it does not affect a data transmission. To detect an increase, [11] suggests using a timer to regularly check if the PMTU has increased by sending a probe packet. [11], and other PMTUD related specifications [15], [16] and [17], recommend a period of ten minutes.

2) Decrease: A PMTU decrease is more severe than an increase. It leads to packet loss for packets larger than the new PMTU. If the network node that drops the packets responds with a PTB message and if the sender processes this message, the detection is simple. Without a PTB message, the detection becomes more complex. In this case, [11] suggests using packet loss as an indication without describing further details.

This paper addresses the question of how to realize a PTB detection in a transport protocol without relying on a PTB message.

III. PTB Detection

Detecting when sending packets that are too big for the network path becomes challenging without a PTB message. We developed a transport protocol agnostic algorithm for that. The key indication for the algorithm is packet loss.

A. Problem

The size of a packet is only one possible reason for its loss. The problem is when to conclude that a packet loss happened due to its size.

The more packets with a specific size or larger get lost, the more likely it becomes that the PMTU is smaller than this size. Finding the right time to conclude that the current PMTU estimation is too large is a trade-off. Conclusions made too early increase the number of false positives. Each false positive unnecessarily reduces the PMTU estimation and produces extra load to the network by sending probe packets. Late conclusions increase the time with a wrong PMTU estimation, during which an endpoint might send packets that are too big for the network path. Consequently, the network drops more packets, which reduces the performance.

B. Algorithm

This algorithm starts after the PMTU search. The main indication is the number of lost packets, but the algorithm does not use all lost packets. It checks which lost packets qualify for the PTB detection.

1) Qualified Lost Packets: The algorithm ignores lost packets sent smaller than a specified minimum PMTU, because the assumption is that each network path at least supports this packet size. It also ignores lost packets sent larger than the current PMTU estimation (e.g., a probe packet), because their loss is an expected result.

After receiving an acknowledgement of receipt for a sent packet, the algorithm considers all packets sent earlier with a size equal or smaller than the acknowledged packet as irrelevant. It does not use these losses to conclude that the PMTU estimation is too large.

The algorithm counts the number of qualified lost packets and uses the parameter n as threshold.

2) Time Distance: Situations in the network may lead to multiple lost packets in a short time (e.g., temporarily path outage). To avoid the potential false conclusion that these lost packets were sent too big, the algorithm uses the time distance of lost packets.

With $n > 1$ qualified lost packets, the algorithm measures the time between the first and last lost packet and uses the parameter t as threshold.

3) Congestion Window Reset: Transport protocols like Transmission Control Protocol (TCP) [18], Stream Control Transmission Protocol (SCTP) [19] or QUIC [20] reset their congestion window (cwnd) to a small value on an event with packet loss (i.e., retransmission timeout or persistent congestion). Since these cwnd-reset events indicate a problem in the network that might relate to the packet size, the algorithm uses them. However, it ignores a cwnd-reset event (and all

previous ones) if a received acknowledgment acknowledges a packet sent after the start of the event with a size of the current PMTU estimation (or larger).

The algorithm counts the number of consecutive cwnd-reset events (e.g., a following cwnd-reset event that resets the cwnd before it has been increased) and uses the parameter c as threshold.

4) Size Reduction: Transport protocols fill packets as large as their PMTU estimation. This leads to continuous packet loss without receiving an acknowledgment if the PMTU estimation is larger than the actual PMTU. However, acknowledgements help transport protocols to declare sent packets as lost.

To trigger acknowledgments even when the PMTU estimation is too large, the algorithm restricts the size of outgoing packets to base PMTU. It starts reducing the size if a sent packet that exceeds base PMTU and triggers an acknowledgment left unacknowledged for a period of at least the time r with no acknowledgement received for a later sent packet. Then, the algorithm reduces the size of outgoing packets until a received acknowledgement acknowledges this or a later sent packet.

Summary: As described in the last section, the algorithm reduces the size of outgoing packets based on the time period r. But it concludes that the current PMTU estimation is too large and, therewith, restarts the PMTU search only if it counts n qualified lost packets, where one of these were sent at least a period t later than another one, or if it counts c consecutive cwnd-reset events without a signal that the network path still supports the current PMTU estimation.

Clearly, the smaller the values for r, n, t and c the faster the algorithm assumes that packets were sent too big. However, choosing larger values prevents false positives.

C. Implementation

Counting the number of qualified lost packets and their time distance seems to us to be the most complex part of the algorithm. For that, we suggest maintaining two lists with the size and sent time of packets.

One list contains acknowledged packets to determine the largest size of an acknowledged packet since a specific time. For this purpose, the algorithm adds newly acknowledged packets to the list only if it contains no larger packet sent later. Consequently, when the algorithm adds a newly acknowledged packet, it removes all equal or smaller packets sent earlier.

The other list contains lost packets. On a newly acknowledged packet, the algorithm removes all equal or smaller packets sent earlier, since their loss become irrelevant for the PTB detection. When the transport protocol declares one or more packets as lost, the algorithm adds each one to the list that is not a PMTU probe packet, larger than minimum PMTU and larger than the largest acknowledged packet since its sent time. After that, the algorithm checks the PTB criterion using both lists.

Figures 1 and 2 show pseudo code to maintain the lists on packet acknowledgment or loss. Figure 3 shows pseudo code to check the conditions using both lists. If the conditions are true, the code indicates how to report which size the path probably still supports. A following PMTU search can use this information to find the new PMTU estimation faster.

```
function onPacketAcked(sentTime, size)
    delEarlierAndSmaller(lostList, sentTime, size)
    if not existsLaterAndLarger(ackedList, sentTime, size)
    then
        delEarlierAndSmaller(ackedList, sentTime, size)
        ackedList.add([sentTime, size])
    end if
end function
```

Figure 1. Pseudo-code that updates the lists of lost and acknowledged packets on a newly acknowledged packet with the given sent time and size

```
function onNonPmtuProbePacketLost(sentTime, size)
    if not (size ≤ minPmtu or
            size ≤ largestAckedSince(sentTime)) then
        lostList.add([sentTime, size])
    end if
end function
```

Figure 2. Pseudo-code that updates the list of lost packets on a packet newly declared lost with the given sent time and size

D. False Positives

Detecting a PMTU estimation as being too large when it does not exceed the actual PMTU is a false positive. Consequently, the PMTUD reduces the PMTU estimation to base PMTU, sends a probe packet for this size and restarts the PMTU search. In case the algorithm reported a packet size still supported by the network path that is equal or larger than base PMTU, PMTUD can skip sending a probe packet for this size and restart PMTU search directly. The PMTU search can use the former PMTU estimation as upper bound of potential PMTU candidates.

```
function checkPtbDetectionCriterion
    for each first in lostList do
        firstAt ← first.sentTime
        largestAcked ← largestAckedSince(firstAt)
        latestAt ← firstAt
        count ← 0
        for each next in lostList do
            if next.sentTime ≥ first.sentTime and
               next.size > largestAcked then
                count ← count+1
                if next.sentTime > latestAt then
                    latestAt ← next.sentTime
                end if
            end if
        end for
        if count ≥ n and (latestAt-firstAt) ≥ t then
            return true (path supports at least largestAcked)
        end if
    end for
    return false
end function
```

Figure 3. Pseudo-code that checks a PTB detection criterion using the lists of lost and acknowledged packets

We consider a search algorithm that starts with probing the upper bound of candidates. With that, the PMTU search starts by sending a probe packet for the former PMTU estimation. In case of a false positive and no further loss of probe packets or their acknowledgements, PMTUD can repair the mistake with one probe packet, or two if it needs to probe for base PMTU first. Thus, it takes about one or two round trip times (RTT).

The negative consequence of one false positive are the extra probe packets and the smaller outgoing non-probe-packets during the time with a smaller PMTU estimation. Note, with further packet loss, the PMTU search may take longer and need to send more probe packets.

IV. Case Study with QUIC

To investigate the PTB detection we need to consider the context, i.e., the transport protocol. We choose QUIC because of its relevance (e.g., as transport protocol for HTTP/3 [21]) and its end-to-end encryption. Conceptionally, the encryption makes it more difficult to use not cryptographically secured information like PTB messages.

A. Specification

[10] specifies QUIC's base protocol and [20] its loss detection and congestion control.

1) Packet: A QUIC packet contains a QUIC header followed by one or more QUIC frames. Multiple types of frames exist. With the STREAM frame, a QUIC sender transmits data in a byte-stream oriented way.

The QUIC header contains a packet number. A sender sets this field. It assigns to each outgoing packet a monotonic increasing number. A receiver acknowledges packets by sending a QUIC packet with an ACK frame that lists the acknowledged packets by its numbers.

QUIC calls packets that trigger an acknowledgment ack-eliciting. For example, a packet containing a STREAM frame is ack-eliciting. A packet containing only ACK frames is not. A receiver sends an acknowledgment for a received ack-eliciting packet immediately if it receives the packet out of order. For an ack-eliciting packet received in order, a receiver delays the acknowledgment until a second ack-eliciting packet arrives, but not longer than the maximum acknowledgment delay (max_ack_delay). [10] recommends setting $max\_ack\_delay = 25\,\mathrm{ms}$. A receiver may further delay the acknowledgment for non-ack-eliciting packets.

2) Loss Detection: QUIC uses an acknowledgment-based loss detection. To know when an acknowledgment to expect, QUIC measures the RTT and calculates a smoothed RTT ($SRTT$) and an RTT variance ($RTTVAR$). It combines these values in the formula $E = SRTT + max(G, 4 \cdot RTTVAR) + max\_ack\_delay$, where G is the timer granularity.

If a sender receives no acknowledgment for a period of E after the last ack-eliciting packet sent, its probe timer expires (i.e., a probe timeout or PTO). On PTO, it sends a loss probe packet containing a STREAM frame with unsent or, if not available, sent but unacknowledged (a.k.a. in flight) data to trigger an acknowledgment. It repeats this, each time with a doubled period (i.e., $2 \cdot E$, $4 \cdot E$ and so on), until it receives an acknowledgment, or its idle timer expires, on which it silently closes the connection.

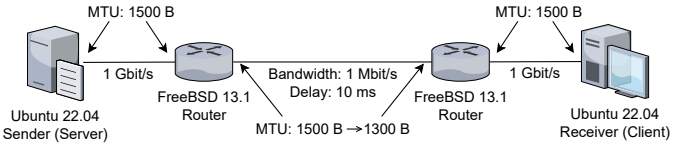

Figure 4. Test Network for QUIC implementations

A sender declares an unacknowledged packet as lost after it receives an acknowledgment for a packet sent $3 \cdot max(SRTT, latest\_rtt)$ later or sent with a packet number equal or larger than $3 + u$, where u is the number of the unacknowledged packet. A sender that declares a packet with STREAM frames as lost, marks the contained data as not in flight (if not still in flight in another packet). When able to send a STREAM frame in a new packet, it selects the oldest data that are not in flight.

3) Congestion Control: QUIC uses a NewReno congestion control [20]. It counts the number of bytes of all packets in flight that contain frames other than ACK frames. The congestion window (cwnd) restricts the number of bytes in flight. The congestion control prohibits sending a packet if the bytes in flight with the packet would exceed the cwnd. As an exception for the loss detection, the congestion control never prohibits sending loss probe packets or packets containing only ACK frames.

A congestion-control-limited sender sends packets clocked by the acknowledgments from the receiver. Each received acknowledgment frees up the cwnd and allows the sender to send further packets.

4) Persistent Congestion: As specified in [20], a sender establishes persistent congestion if it declares all packets sent during a period of at least $3 \cdot E$ as lost. On persistent congestion, it resets its cwnd to the size of two full-sized packets.

5) PMTUD: A QUIC sender must not use IP fragmentation and has to avoid its use by setting the Don't Fragment bit in the IPv4 header. To determine the PMTU, the [10] suggests using PMTUD as described in [11].

B. Implementations

To understand the current state of PMTUD in QUIC, we run tests, especially for the PTB detection, with QUIC implementations. By the time of writing, [22] lists 25 different implementations.

1) Tested Implementations: For our tests, we choose the QUIC implementations picoquic [23] (last commit from 2023-03-23), lsquic [24] (Version 4.0.0), msquic [25] (Version 2.1.8), and s2n-quic [26] (Version 1.17.1). As far as we are aware, only these send packets larger than $1280\,\mathrm{B}$. Hence, we expect a PTB detection only in these implementations.

2) Test Setup: To test the PTB detection in QUIC implementations, we create a small test network, shown in Figure 4. The network consists of a sender and a receiver running Ubuntu 22.04. Each is directly connected by a $1\,\mathrm{Gbit/s}$ link with different routers, which both run FreeBSD 13.1-RELEASE. The routers are directly connected. We use dummynet [27] to emulate a $1\,\mathrm{Mbit/s}$ link with $10\,\mathrm{ms}$ delay between them.

Initially, we set the MTU for all network interfaces to $1500\,\mathrm{B}$. We establish a QUIC connection between the sender and the receiver and let the sender send $1\,\mathrm{GB}$ of data. During transmission we change the MTU of the router's interfaces towards the other router to $1300\,\mathrm{B}$.

3) Result: All tested implementations were able to find the initial PMTU of $1500\,\mathrm{B}$. None of the tested implementations process a PTB message. They ignore the PTB messages the router returns. Only picoquic and lsquic were able to reduce the packet size based on packet loss. The connections of msquic and s2n-quic timed out.

Both, picoquic and lsquic reduce the packet size to $1280\,\mathrm{B}$. picoquic does so after eleven full-sized lost packets. However, other than specified for QUIC, picoquic declares a packet as lost based on a timeout without waiting for an acknowledgment of a later sent packet. lsquic sends two loss probe packets. After that, it also deviates from the specification by starting a retransmission timer. If this timer expires, it reduces the packet size.

Due to the deviations in the loss detection their PTB detections do not describe a solution for a QUIC conform implementation.

C. Simulation Model

To examine the PTB detection algorithm described in Section III, we integrated it in a QUIC simulation model [28].

1) PMTU Search: To determine the PMTU, the QUIC model sends PMTU probe packets in an optimistic binary sequence as described in [29]. It uses $1280\,\mathrm{B}$ as minimum and base PMTU.

With the optimistic binary sequence, the search starts by sending a probe packet for the largest PMTU candidate. If it receives no acknowledgment for this probe packet, it selects the PMTU candidates for the following probe packets as with a binary search.

2) Lost Packets: A sender counts the number of qualified lost packets, measures their time distance and compares it to n and t, respectively, as described in Section III-C.

Note, a QUIC sender that keeps sending oversized packets will not receive acknowledgments from the remote endpoint and is, therefore, unable to declare packets as lost.

3) Packet Size: Generally, a sender builds packets as large as the current PMTU estimation but restricts the size to base PMTU in two cases. If the packet is not ack-eliciting or if an ack-eliciting packet sent before at least a period of r without receiving an acknowledgment for this or a later sent packet (as described in Section III-B4).

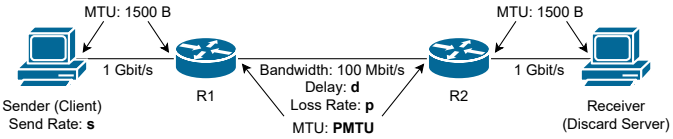

Figure 5. Network in simulation

Since QUIC waits for a period of E for an acknowledgment of an ack-eliciting packet, we set $r \geq E$. With $r = E$, a sender restricts the size of the first loss probe packet. However, the reduced packet is not necessarily a loss probe packet, since the sender restarts the probe timer when sending an ack-eliciting packet. With a send rate of $E/2$, for example, it restricts the size of the third packet, when receiving no acknowledgment.

Since QUIC cannot estimate the time when to receive an acknowledgment for a non-ack-eliciting packet, a sender reduces the packet size based on an outstanding ack-eliciting packet only. Consequently, it restricts the size of non-ack-eliciting packets by base PMTU.

4) Congestion Window Reset: A sender counts the number of consecutive persistent congestions and compares it to the parameter c. However, to establish a persistent congestion a sender needs to receive an acknowledgment. It ignores a persistent congestion (and all previous ones) for the PTB detection if it receives an acknowledgment for a later sent packet whose size corresponds to the current PMTU estimation.

Note, with $r \leq 3 \cdot E$, a sender will not establish persistent congestion if packet loss only happens when sending packets larger than base PMTU.

V. Simulation

We use simulations to investigate the behavior of our PTB detection algorithm in QUIC and to determine adequate values for r, n, t and c.

We start with simulations to measure the time the PTB detection needs to detect a PMTU decrease. Then, we run simulations to measure side effects of the PTB detection with the number of additional sent packets.

A. Setup

The QUIC simulation model (see Section IV-C) operates within the INET network simulation model suite [30] that uses the OMNeT++ simulation library [31].

1) Network: For all simulation runs, we use the network shown in Figure 5. In the network, the links between the hosts and the routers have a bandwidth of $1\,\mathrm{Gbit/s}$ without propagation delay. The link between the routers R1 and R2 is the bottleneck link. For the bottleneck link, we choose a bandwidth of $100\,\mathrm{Mbit/s}$ and a one-way propagation delay of d. The routers use a drop tail queue with a size equal to the bandwidth-delay-product $100\,\mathrm{Mbit/s} \cdot 2 \cdot d$. This bounds the queuing delay to $2 \cdot d$. Depending on the queueing delay, the RTT from one endpoint to the other is between $2 \cdot d$ and $4 \cdot d$.

We configure the MTU of each network interface with $1500\,\mathrm{B}$. However, we reduce the MTU of the router interfaces

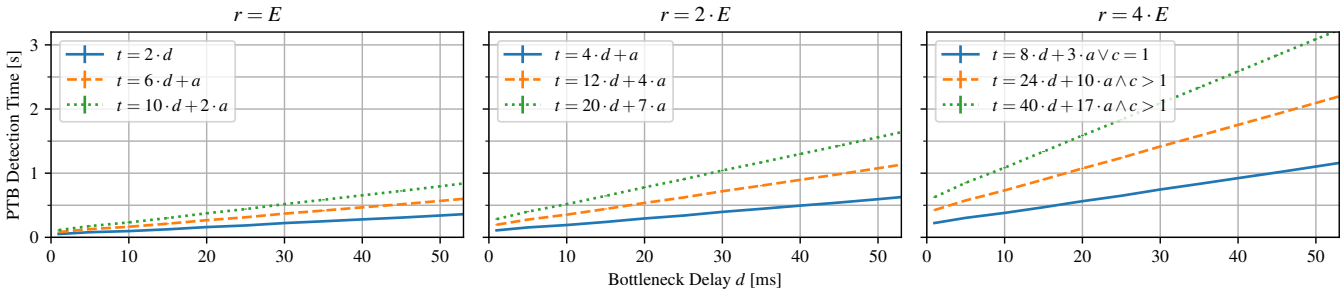

Figure 6. PTB detection time for a congestion-control-limited sender where a denotes $max\_ack\_delay = 25\,\mathrm{ms}$

towards the bottleneck link as required to modify the MTU of the path between sender and receiver. The routers drop packets larger than the MTU of the outgoing interface. We configure the endpoints to ignore PTB messages.

2) Transmissions: The sender starts the QUIC connection and directly searches for the PMTU by sending probe packets.

After the PMTU of 1500 B has been found, the application on the sender starts sending data. The send rate and the size of the packets are relevant factors for the PTB detection algorithm. We examine different cases.

- Congestion-control-limited. The application provides enough data. The sender sends full-sized packets as fast as the congestion control permits.
- Application-limited. The application sends messages with a rate s that is slower than QUIC can send packets. Thus, the message size influences the IP packet size (e.g., because it is equal to the message size plus the size of headers). We use different message sizes.
  - Fixed-sized. The application sends fixed-sized messages. We examine message sizes of 1400 B, 1500 B and 3000 B. With a PMTU estimation of 1500 B, these result in sending one packet, one full-sized packet and one small packet and two full-sized packets and one small packet, respectively.
  - Variable-sized. The application sends messages with a variable size. For each message, it randomly chooses a size between 1042 B and 1442 B. With a PMTU estimation of 1500 B, this results in sending one IP packet that is either smaller or larger than the base PMTU of 1280 B,

We set the send rate s by configuring the time distance between two messages. Since the exact time when the sender sends a packet might influence a result, instead of using a constant distance, we use an exponentially distributed random distance with $1/s$ as mean of the distribution.

To simulate packet loss, the bottleneck link drops packets with a rate p. If $p > 0$, the simulation randomly selects the packet to drop with a universal distribution.

To compensate random effects, we repeat the simulations for each parameter set 1000 times. We present the result with the mean and a 95 % confidence interval (shown by vertical error bars).

B. Detection Time

At first, we use the simulation to measure the time the sender needs to detect that it sent packets that are too big. For this, we set $p = 0$ and reduce the PMTU from 1500 B to 1300 B while the sender transmits data.

In the following, we present the times it takes the sender to detect this PMTU decrease beginning from its first transmission of a packet that is too big.

1) Congestion-Control-Limited: The sender sends packets for about 50 RTTs before the PMTU reduces. We configure the simulation to reduce the PMTU randomly after a period between 49 and 51 RTTs.

Since the sender sends full-sized packets only, after the PMTU decrease, router R1 drops all packets from the sender. The sender's packet send rate is clocked by the acknowledgments from the receiver until one RTT after the decrease. Then, due to missing acknowledgments for the packets sent after the decrease, the congestion control prohibits sending packets. On PTO, the sender sends the loss probe packet. If $r > RTT + E$, it sends a full-sized packet and must wait for the next PTO. Otherwise, it sends a packet with a size of base PMTU, which reaches the receiver and triggers an acknowledgment. When the sender receives the acknowledgement, it declares all outstanding full-sized packets as lost. For $t \leq 2 \cdot d \leq RTT$, the PTB detection resets the PMTU estimation to base PMTU and restarts the PMTU search. Thus, the sender resends the lost data in packets whose size does not exceed the PMTU. For a larger t, the sender keeps sending packets that are too big and the procedure repeats.

Figure 6 shows the result for bottleneck link delays d from 1 to 50 ms. The PTB detection time depends on when the sender reduces the packet size (as specified by r) and whether the time distance of lost packets is sufficient (i.e., at least t). We plot PTB detection times for a sender that reduces the packet size with the first ($r = E$), second ($r = 2 \cdot E$) or third ($r = 4 \cdot E$) loss probe packet. For each case, we set three approximately upper limits for t, such that the sender detects that it sent oversized packets on an acknowledgement after the first, second and third reduced loss probe packet. With $r = E$, for example, choosing a $t < 2 \cdot d$ yields to the same PTB detection time as the blue line shows. We omit the variable n, because we measured the same time for all $n \leq 20$.

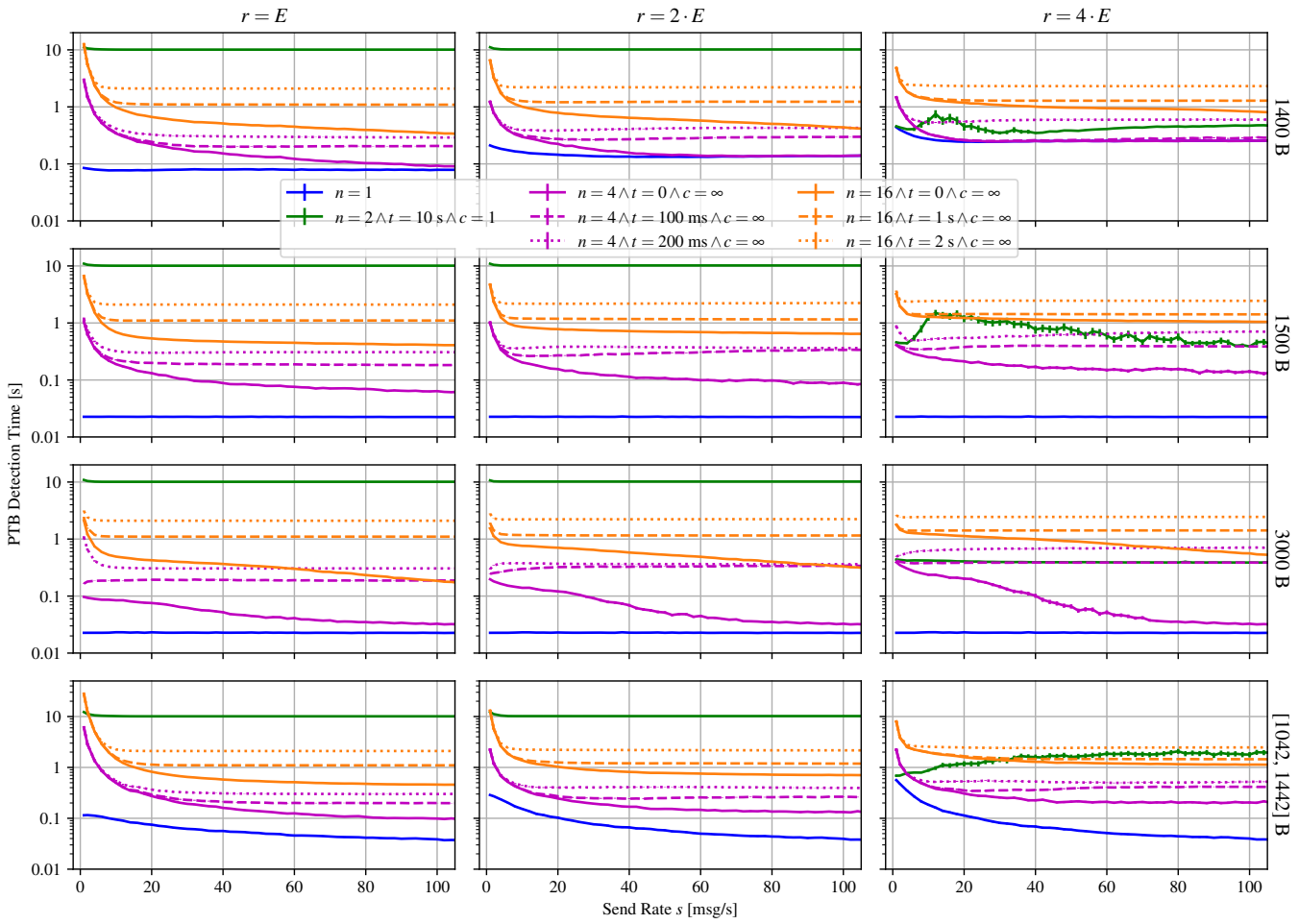

Figure 7. PTB detection time for an application-limited sender

The sender establishes persistent congestion only with $r = 4 \cdot E$ and only on the acknowledgment for the first reduced loss probe packet. Thus, with $r = 4 \cdot E$, setting $t = 8 \cdot d + 3 \cdot max\_ack\_delay$ or $c = 1$ yields to the same detection time as the blue line shows. For $r = E$ or $r = 2 \cdot E$ the parameter c is irrelevant.

2) Application-Limited: The sender sends messages slower than QUIC can send packets. We configure the simulation to reduce the PMTU after one second of sending packets.

Figure 7 shows the PTB detection time for $r = E$, $r = 2 \cdot E$ and $r = 4 \cdot E$ and send rates s between 1 and 100 msg/s. We set the bottleneck link delay to $d = 10\,\text{ms}$. For the plot, we choose values for n, t and c to visualize their influence on the detection time.

As with the congestion-control-limited sender, an application-limited sender sending messages with 1400 B sends only packets that are too big after the PMTU decrease. Therewith, with higher send rates s, the detection time converges faster to the value measured with the congestion-control-limited sender. With message size 1500 B, the acknowledgment for the small packet helps to declare the large packet as lost. This becomes visible in the detection time for $n = 1$. We see the same effect with message size 3000 B, which further shows that the more packets sent at once, the less relevant becomes the parameter n for the detection time. With variable-sized messages, the sender might send multiple messages smaller than the reduced PMTU in a row, which delays the detection time.

The green line shows the detection time when using persistent congestion as signal. Since the sender establishes not in every simulation run a persistent congestion, we also set $n = 2$ and $t = 10\,\text{s}$ to bound the detection time at about 10 s. As the result shows, the sender establishes no persistent congestion with $r = E$ and $r = 2 \cdot E$ and, as visible by the error bars, in some runs not even with $r = 4 \cdot E$. Therefore, using persistent congestion as the only signal is not sufficient.

C. Side Effects

Here, we use the simulation to analyze the effects of the PTB detection without sending packets that are too big. To still trigger the PTB detection we configure random packet loss by a lossy link (with $p > 0$). When triggered, the PTB detection reduces the packet size of outgoing packets on missing acknowledgments and, in case of a false positive, restarts the

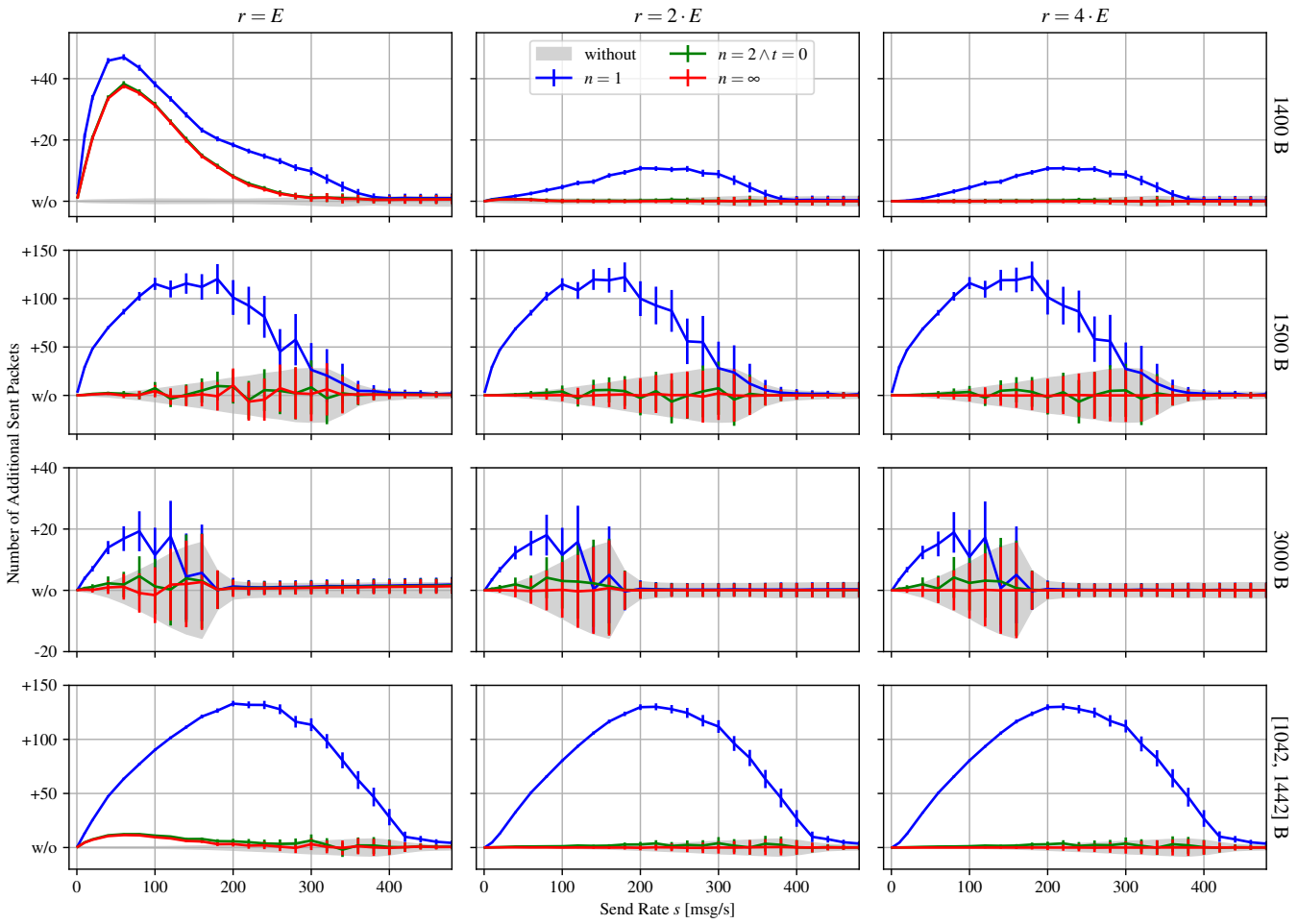

Figure 8. Additional sent packets per minute with PTB detection and a lossy link ($p = 2\,\%$)

PMTU search. Both result in sending additional packets (e.g., two small packets instead of one large packet and PMTU probe packets) that consume extra computing power on all network devices involved. To measure the additional packets, we run the simulations with and without PTB detection enabled and compare the number of packets sent by the sender.

We omit the results for a congestion-control-limited sender here, because they show no statistically relevant number of additional sent packets (i.e., because an acknowledgment for one of the full-sized packets disqualifies any previously sent lost packet for the PTB detection). We considered packet loss due to congestion (caused by other congestion-control-limited senders that share the same bottleneck link) instead of a lossy link. But in order to trigger the PTB detection the application on the sender had to send with such a high rate that itself quickly became congestion-control-limited.

1) Continuously Low Packet Loss: For a continuous packet loss, we set $p = 2\,\%$ and let the sender send packets for one minute.

Figure 8 shows the result. It shows the number of sent packets relative to the mean number of sent packets with PTB detection disabled. The grey area around the base line shows the $95\,\%$ confidence interval. With PTB detection enabled, we choose an extreme case that avoids false positives (i.e., $n = \infty$) to measure the additional packets solely due to the packet size reduction in case of missing acknowledgments.

Due to the low packet loss rate p, multiple lost packets in a short time are improbable. Consequently, the result shows a significant number of additional sent packets only in cases where a single packet loss may trigger the PTB detection. That is the case in the form of packet size reduction with $r = E$ and a sender that sends only one packet per message or in the form of false positives with $n = 1$. Setting $r \geq 2 \cdot E$ and $n > 1$ is sufficient to prevent a statistically relevant number of additional sent packets. The values of t and c are irrelevant for that.

2) Temporarily High Packet Loss: For a temporarily high packet loss, we set p from 0 to $50\,\%$, and after one second back to 0. We identified $50\,\%$ as a high packet loss rate that still transmits enough acknowledgements so that the sender can declare dropped packets as lost quickly which favors false positives. However, with such a high packet loss rate,

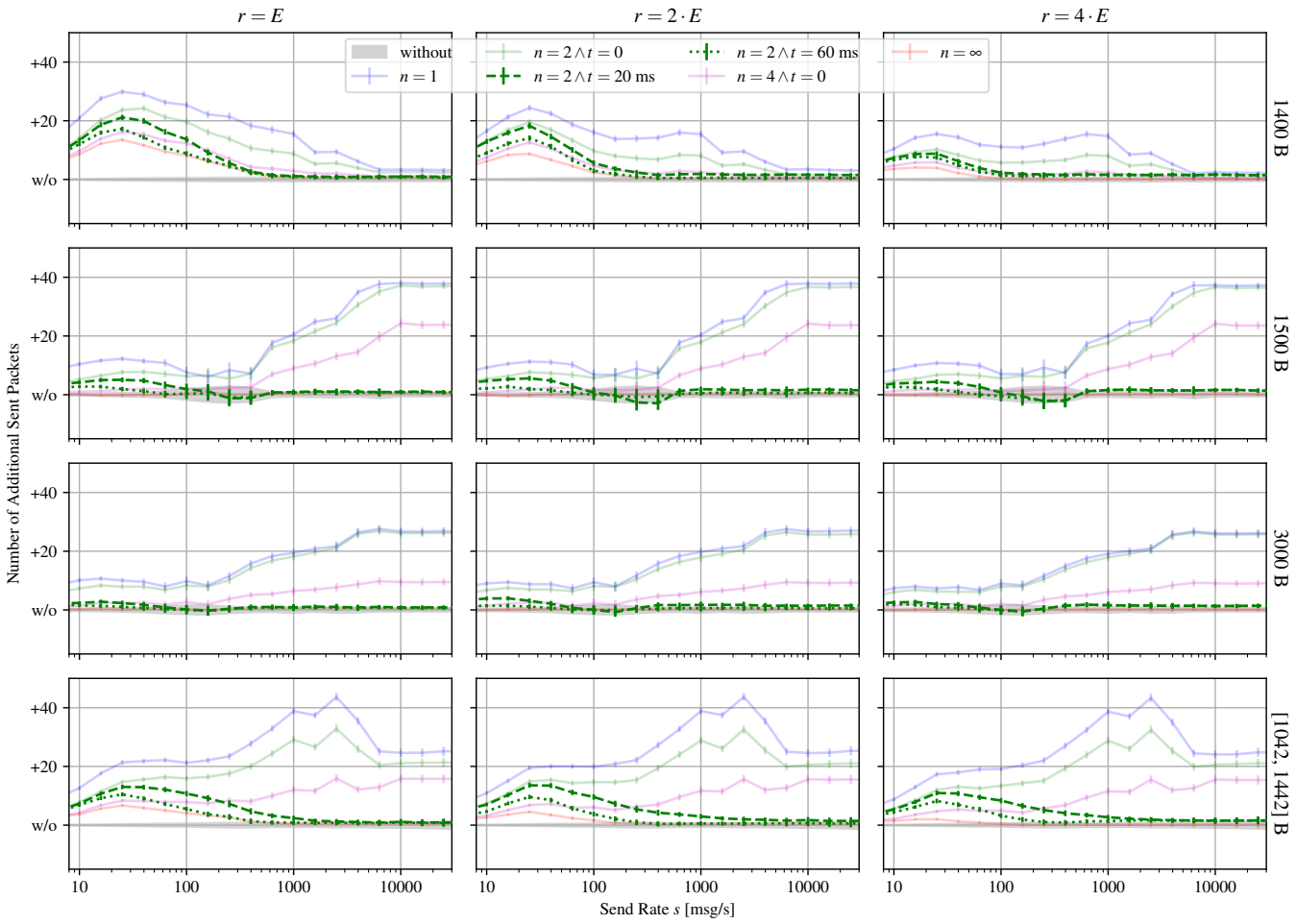

Figure 9. Additional sent packets with PTB detection and a high packet loss rate ($p = 50\,\%$) for one second

the congestion control quickly reduces its cwnd. To avoid an instant congestion-control-limited sending behavior, which prevents false positives, before increasing p, we let the sender send a $50\,\mathrm{Mbit}$ message to increase its cwnd. Even though, the sender becomes congestion-control-limited during the one second. Therefore, using a longer period does not result in substantially more sent packets.

Figure 9 shows the result. Again, it shows the number of sent packets relative to the mean of sent packets with PTB detection disabled and contains the red line for the extreme case with $n = \infty$ to show the number of additional packets without a false positive. The figure shows the result with $c = \infty$. Setting $c = 1$ increases the number of additional sent packets only marginal if at all.

The high packet loss rate p may lead to multiple lost packets in a short time. With $t = 0$, this leads to an early false positive. The earlier the false positive the longer needs the following PMTU search in average due to the longer time with the high packet loss rate. The result shows that setting $t \geq 2 \cdot d = 20\,\mathrm{ms}$ effectively postpones and, for higher send rates s, even prevents a false positive.

D. Result

The simulation results confirm our intuition that the smaller the values for r, n, t and c, the faster the PTB detection. However, given the fact that a PMTU decrease during a connection is a rare case, we believe that it is more important to reduce side effects (e.g., false positives) while still having an adequate detection time.

Based on the results shown in Figure 8, we recommend setting $r \geq 2 \cdot E$ and $n > 1$ to avoid additional packets when experience a continuously low packet loss rate. Based on the results shown in Figure 9, we additionally recommend setting $t \geq SRTT$ to reduce additional packets when experience a short time with a high packet loss rate. Based on the results shown in Figure 7, we suggest setting $n \leq 4$ and, with $r \geq 4 \cdot E$, setting $c = 1$ to achieve an adequate detection time.

As an example, we propose setting $r = 4 \cdot E$, $n = 3$, $t = 3 \cdot SRTT$ and $c = 1$. With that, we measured a detection time of approximately one second or less only and measured a statistically relevant number of additional sent packets only in the case with a high packet loss rate, a large cwnd and an application-limited-sender with a low send rate (see Figure 9).

VI. Conclusion and Outlook

In this paper we presented a novel parameterizable algorithm for reliable transport protocols to detect when sending packets that are too big for the network path (PTB detection) without a PTB message. For a case study, we chose QUIC and presented results from PMTUD tests with QUIC implementations. The results show that their PTB detection is either not function or implemented in a not QUIC conform way. We further described how to integrate our PTB detection algorithm into QUIC and how we used a QUIC simulation model to evaluate the algorithm.

The results from the simulations we presented show the time the algorithm needs to detect that packets were sent too big and the side effects in the number of additional packets sent when packet loss happens independent of the packet size. Based on these results we suggested concrete parameter values for the algorithm with which we were not able to measure a statistically relevant number of additional packets in most cases while still having an adequate detection time.

We expect that the PTB detection algorithm is applicable for other transport protocols, too. Future work is required to proof that. Such an algorithm is especially needed when a sender does not process PTB messages. For example, a sender using SCTP over Datagram Transport Layer Security (DTLS) [32] (as used by WebRTC data channels [33]) will receive no PTB messages, because DTLS does not provide these messages.

References

[1] C. A. Kent and J. C. Mogul, "Fragmentation Considered Harmful," in *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology*, ser. SIGCOMM '87. Association for Computing Machinery, Aug. 1987, p. 390–401. [Online]. Available: https://doi.org/10.1145/55482.55524

[2] J. Postel, "Internet Control Message Protocol," RFC 792, Sep. 1981. [Online]. Available: https://www.rfc-editor.org/info/rfc792

[3] M. Gupta and A. Conta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443, Mar. 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4443

[4] R. Bonica, F. Baker, G. Huston, B. Hinden, O. Trøan, and F. Gont, "IP Fragmentation Considered Fragile," RFC 8900, Sep. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8900

[5] A. Medina, M. Allman, and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, p. 37–52, Apr. 2005. [Online]. Available: https://doi.org/10.1145/1064413.1064418

[6] M. Luckie, K. Cho, and B. Owens, "Inferring and Debugging Path MTU Discovery Failures," in *Internet Measurement Conference 2005*, ser. IMC '05. USENIX Association, Oct. 2005. [Online]. Available: https://www.usenix.org/conference/imc-05/inferring-and-debugging-path-mtu-discovery-failures

[7] M. Luckie and B. Stasiewicz, "Measuring Path MTU Discovery Behaviour," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. Association for Computing Machinery, Nov. 2010, p. 102–108. [Online]. Available: https://doi.org/10.1145/1879141.1879155

[8] M. de Boer and J. Bosma, "Discovering Path MTU Black Holes on the Internet Using the RIPE Atlas," NLnet Labs, Tech. Rep., Jul. 2012, accessed: 2023-05-09. [Online]. Available: https://www.nlnetlabs.nl/downloads/publications/pmtu-black-holes-msc-thesis.pdf

[9] A. Custura, G. Fairhurst, and I. Learmonth, "Exploring Usable Path MTU in the Internet," in *2018 Network Traffic Measurement and Analysis Conference*, ser. TMA '18. IEEE, Oct. 2018, pp. 1–8. [Online]. Available: https://doi.org/10.23919/TMA.2018.8506538

[10] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[11] G. Fairhurst, T. Jones, M. Tüxen, I. Rüngeler, and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports," RFC 8899, Sep. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8899

[12] Y. Schwartz, Y. Shavitt, and U. Weinsberg, "On the Diversity, Stability and Symmetry of End-to-End Internet Routes," in *Conference on Computer Communications Workshops*, ser. INFOCOM '10. IEEE, May 2010, pp. 1–6. [Online]. Available: https://doi.org/10.1109/INFCOMW.2010.5466669

[13] R. Almeida, I. Cunha, R. Teixeira, D. Veitch, and C. Diot, "Classification of Load Balancing in the Internet," in *Conference on Computer Communications*, ser. INFOCOM '20. IEEE, Aug. 2020. [Online]. Available: https://doi.org/10.1109/INFOCOM41043.2020.9155387

[14] Í. Cunha, R. Teixeira, and C. Diot, "Measuring and Characterizing End-to-End Route Dynamics in the Presence of Load Balancing," in *Passive and Active Measurement*, ser. PAM '11. Springer Berlin Heidelberg, Mar. 2011, pp. 235–244. [Online]. Available: https://doi.org/10.1007/978-3-642-19260-9_24

[15] D. S. E. Deering and J. Mogul, "Path MTU discovery," RFC 1191, Nov. 1990. [Online]. Available: https://www.rfc-editor.org/info/rfc1191

[16] J. McCann, S. E. Deering, J. Mogul, and B. Hinden, "Path MTU Discovery for IP version 6," RFC 8201, Jul. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8201

[17] M. Mathis and J. Heffner, "Packetization Layer Path MTU Discovery," RFC 4821, Mar. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4821

[18] W. Eddy, "Transmission Control Protocol (TCP)," RFC 9293, Aug. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9293

[19] R. R. Stewart, M. Tüxen, and K. Nielsen, "Stream Control Transmission Protocol," RFC 9260, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9260

[20] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," RFC 9002, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9002

[21] M. Bishop, "HTTP/3," RFC 9114, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9114

[22] L. Pardue *et al.* QUIC Implementations. Accessed: 2023-05-09. [Online]. Available: https://github.com/quicwg/base-drafts/wiki/Implementations

[23] C. Huitema *et al.* picoquic. Accessed: 2023-05-09. [Online]. Available: https://github.com/private-octopus/picoquic

[24] D. Tikhonov *et al.* lsquic. Accessed: 2023-05-09. [Online]. Available: https://github.com/litespeedtech/lsquic

[25] N. Banks *et al.* msquic. Accessed: 2023-05-09. [Online]. Available: https://github.com/microsoft/msquic

[26] C. Bytheway *et al.* s2n-quic. Accessed: 2023-05-09. [Online]. Available: https://github.com/aws/s2n-quic

[27] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 1, p. 31–41, Jan. 1997. [Online]. Available: https://doi.org/10.1145/251007.251012

[28] T. Völker, E. Volodina, M. Tüxen, and E. P. Rathgeb, "A QUIC Simulation Model for INET and its Application to the Acknowledgment Ratio Issue," in *2020 IFIP Networking Conference (Networking)*, ser. IFIP '20. IEEE, Jul. 2020, p. 737–742. [Online]. Available: https://ieeexplore.ieee.org/document/9142723

[29] T. Völker, M. Tüxen, and E. P. Rathgeb, "The Search of the Path MTU with QUIC," in *Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC*, ser. EPIQ '21. Association for Computing Machinery, Dec. 2021, p. 22–28. [Online]. Available: https://doi.org/10.1145/3488660.3493805

[30] Z. Bojthe, L. Meszaros, G. Szászkő, R. Hornig, A. Varga, and A. Török. INET Framework. Accessed: 2023-05-09. [Online]. Available: https://inet.omnetpp.org/

[31] OpenSim Ltd. OMNeT++. Accessed: 2023-05-09. [Online]. Available: https://omnetpp.org/

[32] M. Tüxen, R. R. Stewart, R. Jesup, and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets," RFC 8261, Nov. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8261

[33] R. Jesup, S. Loreto, and M. Tüxen, "WebRTC Data Channels," RFC 8831, Jan. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8831