Workgroup: Internet Engineering Task Force
Internet-Draft:
draft-kuhn-quic-careful-resume-02
Published: 10 July 2022
Intended Status: Informational
Expires: 11 January 2023
Authors: N. Kuhn
Thales Alenia Space
E. Stephan
Orange
G. Fairhurst
University of Aberdeen
T. Jones
University of Aberdeen
C. Huitema
Private Octopus Inc.

Careful resumption of congestion control from retained state with QUIC

Abstract

This document discusses careful resumption of congestion control parameters in QUIC with a cautious method that enables faster startup of new connections.

The method uses a set of computed congestion control parameters that are based on the previously observed path characteristics, such as the bottleneck bandwidth, available capacity, or the RTT. These parameters are stored and can then used to modify the congestion control behaviour of a subsequent connection. The draft discusses assumptions around how a server ought to utilise these parameters to provide opportunities for a new connection to more quickly get up to speed (i.e. utilise available capacity). It discusses how these changes impact the capacity at a shared network bottleneck and the response that is needed after any indication that the new rate is inappropriate.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

All Internet transports are required to use a CC method. In 2010, RFC 5783 provided a survey of alternative CC methods, and noted that there are challenges when a CC operates across an Internet path with a high and/or variable bandwidth-delay product (BDP) [RFC5783].

A CC algorithm typically takes time to ramp-up the packet rate, called the "slow-start phase", informally known as the time to "Get up to speed". The slow start phase is a period in which a sender intentionally uses less capacity than might be available with the intention to avoid overshooting the actual capacity at a bottleneck, which would result in increased queueing (latency/jitter) and/or congestion packet loss. An overshoot in the capacity can have a detrimental effect on other flows sharing a common bottleneck. In the extreme case, persistent congestion can result in unwanted starvation of other flows [RFC8867] (i.e. Preventing other flows from successfully sharing a common bottleneck).

In Reno, the slow-start phase consists of a sequence of increases in the congestion window (cwnd) starting from the Initial Window (IW). Each step lasts approximately one path RTT, until the sender estimates that the capacity at the bottleneck for the path has been (or is nearing) reached.

To fully-utilise the capacity along a path with a certain RTT, the transport needs to determine an appropriate volume of bytes in flight, based on the product of the available capacity and the path RTT. [RFC6349] defines the BDP as follows: "Derived from Round-Trip Time (RTT) and network Bottleneck Bandwidth (BB), the Bandwidth-Delay Product (BDP) determines the Send and Received Socket buffer sizes required to achieve the maximum TCP Throughput." The BDP estimated by a server includes all buffering experienced along a network path. Various approaches are possible to determine the BDP, based on measurements of the path characteristics. [RFC6349] specifies one procedure for TCP. CC for QUIC is specified in [RFC9002] and does not specify a required method to measure the BDP, allowing the sender to implement an appropriate method.

The specification for the QUIC transport protocol [RFC9000] notes "Generally, implementations are advised to be cautious when using previous values on a new path." The method uses a set of computed Congestion Control (CC) parameters that are based on the previously observed path characteristics, such as the bottleneck bandwidth, available capacity, or the Round Trip Time (RTT). These parameters are stored and can then used to modify the CC behaviour of a subsequent connection.

This document specifies a method that can improve throughput by reducing the time to get up to speed, and hence the total duration of a transfer. It introduces an alternative method to select initial CC parameters, including a way to more rapidly and safely grow the cwnd.

There are scenarios where temporal sharing of previously parameters relating to observed path characteristics, such as the bottleneck bandwidth or RTT, can help to save round-trip times at the start of a new connection. For example:

1. To optimize applications that use a series of short connections over the same path, each of which needs to individually learn the available capacity/rtt;

2. After a pause in transmission (e.g., when transmission pauses, and then the transport protocol wishes to connect over the same path);

3. To connect after a service disruption where the network service was temporarily reduced (e.g. due to a link propagation impairment, or where a user on a train journey travels through different areas of connectivity before the user returns to a path with the original characteristics).

In these cases, specific characteristics of the path may have been learned, including CC information. This information might be expected to be similar when a new connection is made between the same local and remote endpoints.

While a server could take optimization decisions without considering the client's preference, in some cases a client could have information that is not available at the server. A client may provide hints, for example: (1) an indication that the path/local interface has changed; (2) information related to current hardware limitations of the client or (3) an understanding about the capacity needs of other concurrent flows that would compete for shared capacity. As a result, a client could explicitely ask for tuning the slow start when the application continues transmission, or to inhibit tuning. This is discussed further later in the document.

There are also cases where using the parameters of a previous connection are not appropriate, and a need to evaluate the potential for malicious use of the method.

The remainder of this document:

1. discusses use-cases where carefully resuming QUIC connections is expected to have benefit;

2. proposes guidelines for how to carefully utilise the previously stored CC information;

3. describes implementation considerations for the proposed method using QUIC;

4. discusses the trade-offs associated with the different implementation solutions.

2. Language, notations and terms

This section provides a brief summary of key terms and the requirements language that is used. The document uses language drawn from a range of IETF RFCs.

2.1. Requirements Language

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Notations and Terms

This document defines current, and saved values for a set of CC parameters:

*IW: Initial Window [RFC9002];

*current_iw: Current IW;

*recom_iw: Recommended IW;

*current_bb : Current estimated bottleneck bandwidth;

*saved_bb: Estimated bottleneck bandwidth preserved from a previous connection;

*current_rtt: Current RTT;

*saved_rtt: RTT measure RTT preserved from a previous connection;

*client_ip : IP address of the client;

*current_client_ip : Current IP address of the client;

*saved_client_ip : IP address of a previous connection by the client;

*remembered BDP parameters: a combination of saved_rtt and saved_bb

Congestion controllers, such as CUBIC or RENO, could estimate the saved_bb and current_bb values by utilizing a combination of the cwnd/flight_size and the minimum RTT. A different method could be used to estimate the same values when using a rate-based congestion controller, such as BBR [I-D.cardwell-iccrg-bbr-congestion-control]. It is important to consider whether the methods could result in over-estimating the bottleneck bandwidth, and the preserved values ought to be used with caution.

3. Scenarios of Interest

3.1. Large BDP Scenarios

QUIC introduces the concept of transport parameters (section 4 of [RFC9000]). This document notes that a new connection can utilise a set of key transport parameters from a previous connection to reduce the completion time for a transfer with a size much larger than the IW over paths where the available capacity is also significantly larger than the IW. This benefit is particularly evident for a path where the RTT is much larger than for typical Internet paths.

For example, a satellite access network, a 5.3 MB transfer takes up to 9 seconds using standard congestion control, whereas using the specified method this could reduce to 4 seconds [IJSCN]; and the time to complete a 1 MB transfer could be reduced by 62 % [MAPRG111]. Benefits is also expected for other sizes of transfer and for different path characteristics that also result in a higher BDP.

3.2. Accomodating from a Known Reduction in Capacity

A transport protocol is not able to assume that the path characteristics remain the same. Variation can arise from a combination of various factors:

*Competing network traffic sharing a common bottleneck can result in short or long term variation;

*Changes in the forward path can change the set of links/routers over forming the path (from routing/mobility/circuit restoration/ interface change), resulting in a change in the bandwidth and the other traffic that shares a bottleneck on the path;

*Link conditions can change, resulting in a change of the bottleneck bandwidth (e.g., as a result of changes in propagation conditions or sharing of a medium);

*Application/endpoint behavior can change the capacity available to a flow.

Although a transport protocol can have information about a previously used path, the path characteristics can change, and previous information may not be appropriate when a new connection uses the path.

In some cases (e.g., after a change in the interface used by the local endpoint), a client may be aware of such a change, and might be able to infer that a previously available path has again become available. However, to safetly utilise the previous information, the client would need assurance that the path was to the same endpoint, and that the characteristics have not significantly changed from those previously measured. When the path is expected to be the same, there is then an opportunity to reduce the time to get up to speed by utilising saved CC information for the path.

3.3. Optimizing Client Requests

3.4. Sharing Transport Information across Multiple Connections

There can be benefit in sharing transport information across multiple concurrent connections. [RFC9040] considers the sharing of transport parameters between TCP connections that originate from a host. The proposal in this document has the advantage of storing server-generated information at the client and not requiring the server to retain additional state for each client.

3.5. Connection Establishment, Client and Server

In the previously detailed scenarios, the application data transfer was unidirectional towards the client, i.e., the main flow of data was from a server to a client (e.g., downloading a file or web page). This is the focus of the current version of the document.

In a different example, the application data transfer can be unidirectional towards the server, e.g., uploading an image/video is a server.

There are also use cases where a client initiates a connection for a bidirectional service where both endpoints send data to each other, such as to support a remote executing application, or a video conference call.

In general, the guidelines proposed in this document apply when a congestion controller is sending data to a remote peer and that remote endpoint resumes the connection. Both endpoints can assume the role of a client or a server.

4. The Phases of CC

This document defines a series of different phases through which the CC algorithm moves as a connection gets up to speed. The phases are labelled as follows:

1. Observe: During a previous connection, the current RTT (current_rtt), bottleneck bandwidth (current_bb) and current client IP (current_client_ip) are stored as saved_rtt, saved_bb and saved_client_ip;

2. Reconnaissance: When an application resumes between the same pair of IP addresses, the server measures the path characteristics of a new connection to confirm the path appears to be similar to that observed previously (e.g., a similar RTT). The server also seeks assurance that initial data is not lost, to avoid resuming under congested conditions.

3. Unvalidated: Utilise the saved path characteristics to send at a rate higher than allowed by slow start. The convergence towards the previous rate is expected be faster than when using traditional slow-start mechanisms, but should not be instantaneous, to avoid adding congestion to an already congested bottleneck.

    1. If the unvalidated rate was used without inducing noticeable congestion to the path, the sender is permitted to continue at this rate in the 'Normal' phase.

    2. If the validation phase determines that previous parameters are not valid (due to a change) or congestion was experienced, the sender must withdraw rapidly to a safe rate, before it enters the 'Normal' phase.

4. Normal: Resume using the normal CC method.

5. Safe Jump

This section introduces the rationale behind safety guidelines related to the usage of previous values on a new path: variable network conditions and malicious client.

The "variable network conditions" related to the fact that previously measured values may not remaib relevant and should be exploited cautiously by a CC algorithm.

The "malicious client" relates to the fact that a malicious client could try to send malicious information to a server. Three approaches are then introduced and compared : either (1) all the information related to previous connections is stored at the server

and never send to a client ("Local storage"), (2) some information is transmited to a client that can use it when reconnecting but the client cannot read the information received from the server ("NEW TOKEN"), or (3) some information is transmitted to a client that can use it when reconnecting and the client can read it to accept or not the exploitation of previous congestion information (a.k.a. "BDP extension").

5.1. Rationale behind the Safety Guidelines

NOTE: The sender ought not to re-utilise all the capacity it previously used, to avoid starving other flows that started or increased their capacity after the last measurement. How strong should this be stated: ... MUST or SHOULD ... What safety factor is appropriate for the resuming sender? If using slow-start it would anyway double the rate on the next RTT, so is capacity/2 appropriate to initially try?

A new connection MUST NOT use the previously measured saved_rtt and saved_bb to simply initialise a new flow to resume sending at the same rate.

- Rationale #1: Bottleneck bandwidth and network traffic can change at any time. An Internet method needs to be robust to network conditions that can differ from one connection to the next, due to variations in the forwarding path, reconfiguration of equipment or changes in the link conditions. An Internet method needs to be robust to changes in network traffic, including the arrival of new traffic flows that compete for the bottleneck capacity. Behaviours need to be designed that avoid sending excessive data into a congestion bottleneck because this can have a material impact on any flows using that bottleneck, and the ability of those flows to control their own sending rate.

- Rationale #2: Information sent by a malicious client is not relevant. A client could request a server to use a cwnd higher than appropriate, to gain an unfair share of capacity for itself or to induce congestion for other flows. A server might anyway decide whether to fully use the new allowed rate.

5.2. Rationale #1: Variable Network Conditions

The server MUST check the validity of any received saved_rtt and saved_bb parameters, whether these are sent by a client or are stored at the server. The following events indicates cases where the use of these parameters is inappropriate:

- IP address change: If the client changes its local IP address (i.e., the saved_client_ip is different from the current_client_ip), the different source address is a assumed an

indication of a different network path. This new path does not necessarily exhibit the same characteristics as the old one. If the server changes its IP address after a migration, it would not be safe to exploit previously estimated parameters.

*RTT change: A significant change in RTT might be an indication that the network conditions have changed. Since the CC information is directly impacted by the RTT, a significant change in the RTT is a strong indication that the previously estimated BDP parameters are likely to not be valid for the current path. NOTE: This document needs to define a significant change.

*Lifetime of the information: The CC information is temporal. Frequent connections to the same IP address are likely to track changes, but long-term use of previous values is not appropriate. NOTE: This document needs to define how long.

*BB over-estimation: There are cases where using a measured cwnd would inflate the bottleneck bandwidth. At the end of the CC slow start phase, the value of cwnd can be significantly larger than the minimum value needed to utilise the path (i.e., cwnd overshoot). In most case, the cwnd finally converges to a stable value after a few more RTTs. It would be inappropriate to use an overshoot in the cwnd as a basis for estimating the bottleneck bandwidth. NOTE: One mitigation could be to further restrict to only a fraction (e.g., 1/2) of the previously used cwnd; another mitigation might be to calculate the bottleneck bandwidth based on the flight_size or an averaged cwnd.

*Preventing Starvation of New Flows: It would not be appropriate to fully use a bottleneck bandwidth estimate based on a previous measurement of capacity, because new flows might have started using the available capacity since that measurement was made. The mitigation could be to restrict to only a fraction (e.g., 1/2) of the previously used cwnd.

There are several solutions to mitigate the impact of changes in network conditions:

*Rationale #1 - Solution #1 : When resuming, restore the current_bb and current_rtt from the saved_bb and saved_rtt parameters estimated from a previous connection.

*Rationale #1 - Solution #2 : When resuming, implement a safety check to measure avoid using the saved_bb and saved_rtt parameters to cause congestion over the path. In this case, the current_bb and current_rtt might not be set directly to the saved_bb and saved_rtt: the server might wait for the completion of the safety check before this is done.

Section 6 describes various approaches for Rationale #1 - Solution #2.

5.3. Rationale #2: Malicious clients

The server MUST check the integrity of the saved_rtt and saved_bb parameters received from a client.

There are several solutions to avoid attacks by malicious clients:

- Rationale #2 - Solution #1 : The server stores a local estimate of the bottleneck bandwidth and RTT parameters as the saved_bb and saved_rtt.

- Rationale #2 - Solution #2 : The server sends the estimate of the bottleneck bandwidth and RTT parameters to the client as the saved_bb and saved_rtt in a block of information that is authenticated. This information also could be encrypted by the server. The client resends the same information for a new connection. The server can use its local key information to authenticate the information, without needing to keep a local copy.

- Rationale #2 - Solution #3 : This approach is the same as above, except that the server sends an estimate of the saved_rtt and saved_bb parameters in a form that may be read by the client. The information might not be encrypted, or the information might be duplicated outside of the encrypted block. This allows a client to read, but not modify, the saved_rtt and saved_bb parameters and could enable a client to decide whether the new parameters are thought appropriate, based on client-side information about the network conditions, connectivity, or needs of the new connection.

Appendix A describes various implementation approaches for each of these solutions using local storage (Appendix A.2 for Rationale #2 - Solution #1), NEW_TOKEN Frame (Appendix A.3 for Rationale #2 - Solution #2), BDP extension Frame (Appendix A.4 for Rationale #2 - Solution #3).

5.4. Trade-off between the different solutions

This section provides a description of several implementation options and discusses their respective advantages and drawbacks.

While there are some discussions for the solutions regarding Rationale #2, the server MUST consider Rationale #1 - Solution #2 and avoid Rationale #1 - Solution #1: the server MUST implement a safety check to measure whether the saved BDP parameters (i.e.

saved_rtt and saved_bb) are relevant or check that their usage would not cause excessive congestion over the path.

Security consideration are discussed in Section 9 .

5.4.1. Interoperability and Use Cases

A server that stores a resumption ticket for each client to protect against replay on a third party IP, it could also store the IP address (i.e., saved_client_ip) and BDP parameters (i.e., saved_rtt and saved_bb) of a previous connection.

When the BDP Frame extension is used, locally stored BDP parameters at the server can provide a cross-check of the BDP parameters sent by a client. The server can anyway enable a safe jump, but without the BDP Frame extension. However, using the parameters enables a client to choose whether to request this or not, enabling it to utilize local knowledge of the network conditions, connectivity, or connection requirements.

XXX-Editor-note: Text to be improved: Storing local values related to the BDP would help improve the ingress for new connections, however, not using a BDP Frame extension could reduce the interest of the approach where (1) the client knows the BDP estimation at the server, (2) the client decides to accept or reject ingress optimization, (3) the client tunes application level requests.

5.4.2. Summary

Local storage of values can be secure and the BDP Frame extension provides more information to the client and more interoperability. The Figure 1 provides a summary of the advantages and drawbacks of each approach.

Rationale	Solution	Advantage	Drawback	Comment
#1 Variable Network	#1 set current_* to saved_*	Ingress optim.	Risk of adding congestion	MUST NOT implement
	#2 Implement safety check	Reduce risk of adding congestion	Negative impact on ingress optim.	MUST implement Section 3
#2 Malicious client	#1 Local storage	Enforced security	Client unable to decide to reject Malicious server could fill client's buffer Limited use-cases	Section 4.2
	#2 NEW_TOKEN	Save resource at server Opaque token protected	Malicious client could change token even if protected Malicious server could fill client's buffer Server may not trust client	Section 4.3
	#3 BDP extension	Extended use-cases Save resource at server Client can read and decide to reject BDP extension protected	Malicious client could change BDP even if protected Server may not trust client	Section 4.4

XXX-Editor-Note: Need to clarify the text around changing the authenticated token.

Figure 1: Comparing solutions

6. Safety Guidelines

The following safety guidelines refer to the labelling defined in Section 4.

The safety guidelines are designed to mitigate the risk that a server adds excessive congestion to an already congested path. The following mechanisms help in fulfilling this objective:

* (observation phase) The server SHOULD NOT store and/or send information related to a previously estimated bottleneck bandwidth (saved_bb) (see Section 2.2 for more details on bottleneck bandwidth definition), if the cwnd is not at least four times larger than the IW.

* (reconnaissance phase) The server MUST NOT send more than the recommended maximum IW (recom_iw) in the first RTT of transmitting data [RFC9000]. (When used in a controlled network, additional information about local path characteristics could be known that might be used to configure a non-standard IW).

* (reconnaissance phase) The server MUST compare the measured transport parameters (in particular current_rtt) of the 0-RTT connection with those of the 1-RTT connection (in particular saved_rtt). The method MUST NOT be used when the path fails to be validated;

* (unvalidated phase) The server MUST NOT use the parameters unless the first IW packets when packets are detected as lost or acknowledgements indicate the packets were ECN CE-marked. These are indication of potential congestion and therefore the method MUST NOT be used;

* (unvalidated phase) The server MUST implement the retreat method when packets are detected as lost or acknowledgements indicate the packets were ECN CE-marked. These are indication of potential congestion and therefore the method MUST NOT be used.

The proposed mechanisms SHOULD be limited by any rate-limitation mechanisms of QUIC, such as flow control mechanisms or amplification attack prevention. In particular, it may be necessary to issue proactive MAX_DATA frames to increase the flow control limits of a connection. In particular, the maximum number of packets that can be sent without acknowledgements needs to be chosen to avoid the creation and the increase of congestion for the path.

This extension MUST NOT provide an opportunity for the current connection to be a vector of an amplification attack. The address

validation process, used to prevent amplification attacks, SHOULD be performed [RFC9000].

XXX-Editor-note: This probbaly should be a range rather than an inequality (current_rtt < 1.2*saved_rtt).

The following mechanisms could be implemented:

*Exploit a standard IW:

1. The server sends the first data packet using the IW - this is a safe starting point for any path where there is no path information or congestion control information. This avoids adding excessive congestion to a path;

2. The sender monitors the reception of the IW data. If the path characteristics resemble those of a recent previous connection from to the same server (i.e., current_rtt < 1.2*saved_rtt) and all data was acknowledged without reported congestion), the method permits the sender to utilise the saved_bb as an input to adapt current_bb to rapidly determine a new safe rate;

3. The sender needs to avoid a burst of packets resulting from a step-increase in the congestion window [RFC9000]. Pacing the packets as a function of the current_rtt can provide this additional safety during the period in which the CWND is increased by the method.

*Identify a relevant pacing rhythm:

-The server estimates the pacing rhythm using saved_rtt and saved_bb. The Inter-packet Transmission Time (ITT) is determined by the ratio between the current Maximum Message Size (MMS) and the ratio between the saved_bb and saved_rtt. A tunable safety margin can avoid sending more than a recommended maximum IW (recom_iw):

ocurrent_iw = min(recom_iw,saved_bb)

oITT = MSS/(current_iw/saved_rtt)

-When the successful receipt of the IW data is acknowledged, the server returns to a standard slow-start mechanism.

*Tune slow-start mechanisms: After transport parameters are set to a previously estimated bottleneck bandwidth, if the slow-start mechanisms continue, the sender can then overshoot the bottleneck

capacity. This can occur even when using the safety check described in this section.

- For NewReno and CUBIC, it is recommended to exit slow-start and enter the congestion avoidance phase.

- For BBR, it is recommended to enter the "probe bandwidth" state.

This follows the idea presented in [RFC4782], [I-D.irtf-iccrg-sallantin-initial-spreading] and [CONEXT15].

7. Acknowledgments

The authors would like to thank Gabriel Montenegro, Patrick McManus, Ian Swett, Igor Lubashev, Robin Marx, Roland Bless and Franklin Simo for their fruitful comments on earlier versions of this document.

8. IANA Considerations

TBD: Text is required to register the BDP Frame and the enable_bdp transport parameter. Parameters are registered using the procedure defined in [RFC9000].

9. Security Considerations

Security considerations for QUIC are discussed in Section 6

The client can send information related to the saved_rtt and saved_bb to the server with the BDP Frame extension using either Rationale #2 - Solution #2 or Rationale #2 - Solution #3. However, the server SHOULD NOT trust the client. Indeed, even if 0-RTT packets containing the BDP Frame are encrypted, a client could modify the values within the extension and encrypt the 0-RTT packet. Authentication mechanisms might not guarantee that the values are safe. It is not an easy operation for a client to modify authenticated or encrypted data without this being detected by a server. Modification could be realized by malicious clients. One way to avoid this is for a server to also store the saved_rtt and saved_bb parameters.

A malicious client might modify the saved_bb parameter to convince the server to use a larger CWND than appropriate. Using the algorithms proposed in Section 6, the server may reduce any intended harm and can check that part of the information provided by the client are valid.

Storing the BDP parameters locally at the server reduces the associated risks by allowing the client to transmit information

related to the BDP of the path in the case of a malicious client trying to break the encryption mechanism that it had received.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4782] Floyd, S., Allman, M., Jain, A., and P. Sarolahti, "Quick-Start for TCP and IP", RFC 4782, DOI 10.17487/RFC4782, January 2007, <https://www.rfc-editor.org/info/rfc4782>.

[RFC6349] Constantine, B., Forget, G., Geib, R., and R. Schrage, "Framework for TCP Throughput Testing", RFC 6349, DOI 10.17487/RFC6349, August 2011, <https://www.rfc-editor.org/info/rfc6349>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/info/rfc9002>.

10.2. Informative References

[CONEXT15] Li, Q., Dong, M., and P B. Godfrey, "Halfback: Running Short Flows Quickly and Safely", ACM CoNEXT , 2015.

[I-D.cardwell-iccrg-bbr-congestion-control] Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrg-bbr-congestion-control-02, 7 March 2022, <https://www.ietf.org/archive/id/draft-cardwell-iccrg-bbr-congestion-control-02.txt>.

[I-D.irtf-iccrg-sallantin-initial-spreading]
Sallantin, R., Baudoin, C., Arnal, F., Dubois, E., Chaput, E., and A. Beylot, "Safe increase of the TCP's Initial Window Using Initial Spreading", Work in Progress, Internet-Draft, draft-irtf-iccrg-sallantin-initial-spreading-00, 15 January 2014, <https://www.ietf.org/archive/id/draft-irtf-iccrg-sallantin-initial-spreading-00.txt>.

[IJSCN] Thomas, L., Dubois, E., Kuhn, N., and E. Lochin, "Google QUIC performance over a public SATCOM access", International Journal of Satellite Communications and Networking 10.1002/sat.1301, 2019.

[MAPRG111] Kuhn, N., Stephan, E., Fairhurst, G., Jones, T., and C. Huitema, "Feedback from using QUIC's 0-RTT-BDP extension over SATCOM public access", IETF 111 - MAPRG meeting , 2022.

[RFC5783] Welzl, M. and W. Eddy, "Congestion Control in the RFC Series", RFC 5783, DOI 10.17487/RFC5783, February 2010, <https://www.rfc-editor.org/info/rfc5783>.

[RFC8867] Sarker, Z., Singh, V., Zhu, X., and M. Ramalho, "Test Cases for Evaluating Congestion Control for Interactive Real-Time Media", RFC 8867, DOI 10.17487/RFC8867, January 2021, <https://www.rfc-editor.org/info/rfc8867>.

[RFC9040] Touch, J., Welzl, M., and S. Islam, "TCP Control Block Interdependence", RFC 9040, DOI 10.17487/RFC9040, July 2021, <https://www.rfc-editor.org/info/rfc9040>.

Appendix A. Implementation Considerations

A.1. Rationale behind the different implementation options

The NewSessionTickets message of TLS can offer a solution. The proposal is to add a 'bdp_metada' field in the NewSessionTickets, which the client is able to read. The only extension currently defined in TLS1.3 that can be seen by the client is max_early_data_size (see Section 4.6.1 of [RFC8446]). However, in the general design of QUIC, TLS sessions are managed by a TLS stack.

Three distinct approaches are presented: sending an opaque blob to the client that the client may return to the server when establishing a future new connection (see Appendix A.3), enabling local storage of the BDP infromation (see Appendix A.2) and a BDP Frame extension (see Appendix A.4).

A.2. Independent Local Storage of Values

This approach independently lets both a client and a server store their BDP parameters:

- During a 1-RTT session, the endpoint stores the RTT (as the saved_rtt) and bottleneck bandwidth (as the saved_bb) together in the session resume ticket. The client can also store the IP address of the server;

- The server maintains a table of previously issued tickets, indexed by the random ticket identifier that is used to guarantee uniqueness of the Authenticated Encryption with Associated Data (AEAD) encryption. Old tokens are removed from the table using the Least Recently Used (LRU) logic. For each ticket identifier, the table holds the RTT and bottleneck bandwidth (i.e. saved_rtt and saved_bb), and also the IP address of the client (i.e. saved_client_ip).

During the 0-RTT session, the local endpoint waits for the first RTT measurement from the remote endpoint IP address. This is used to verify that the current_rtt has not significantly changed from the saved_rtt (used as an indication that the BDP information is appropriate for the current path).

If this RTT is confirmed, the endpoint also verifies that an IW of data has been acknowledged without requiring retransmission or resulting in an ECN CE-mark. This second check detects whether a path is experiencing significant congestion (i.e., where it would not be safe to update the cwnd based on the saved_bb). In practice, this could be realized by a proportional increase in the cwnd, where the increase is (saved_bb/IW)*proportion_of_IW_currently-ACKed.

This solution does not allow a client to request the server not to use the BDP parameters. If the server does not want to store the metrics from previous connections, an equivalent of the tcp_no_metrics_save for QUIC may be necessary. This option could be negotiated that allows a client to choose whether to use the saved information.

A.3. Using NEW_TOKEN frames

A server can send a NEW_TOKEN Frame to the client. The token is an opaque (encrypyted) blob and the client can not read its content (see section 19.7 of [RFC9000]). The client sends the received token in the header of an Initial packet of a later connection.

A.4. BDP Frame

Using BDP Frames, the server could send information relating to the path characteristics to the client. The use of the BDP Frame is negotiated with the client. The client can read its content. If the client agrees with the usage of previous parameters, it can send the BDP Frame back to the server in an Initial packet of a later connection.

Authors' Addresses

Nicolas Kuhn
Thales Alenia Space

Email: nicolas.kuhn.ietf@gmail.com

Emile Stephan
Orange

Email: emile.stephan@orange.com

Godred Fairhurst
University of Aberdeen
Department of Engineering
Fraser Noble Building
Aberdeen

Email: gorry@erg.abdn.ac.uk

Tom Jones
University of Aberdeen
Department of Engineering
Fraser Noble Building
Aberdeen

Email: tom@erg.abdn.ac.uk

Christian Huitema
Private Octopus Inc.

Email: huitema@huitema.net