Workgroup: Internet Engineering Task Force
Internet-Draft: draft-kuhn-quic-0rtt-bdp-11
Published: 23 October 2021
Intended Status: Informational
Expires: 26 April 2022
Authors: N. Kuhn E. Stephan G. Fairhurst
CNES Orange University of Aberdeen
T. Jones C. Huitema
University of Aberdeen Private Octopus Inc.

Transport parameters for 0-RTT connections

Abstract

QUIC 0-RTT transport features currently focuses on egress traffic optimization. This draft describes a QUIC extension that can be used to improve the performance of ingress traffic.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 April 2022.

Table of Contents

1. Introduction

QUIC 0-RTT transport features currently focus on egress traffic optimization. This draft describes a QUIC extension that can be used to improve the performance of ingress traffic.

[RFC9000] mentions that "Generally, implementations are advised to be cautious when using previous values on a new path." This draft proposes a discussion on how using previous values can be achieved in a interoperable manner and how it can be done safely.

When clients resume a session to download a large object, the congestion control algorithms will require time to ramp-up the packet rate as a sequence of Round-Trip Time (RTT)-based increases.

This document specifies a method that can improve traffic delivery by allowing a QUIC connection to avoid a the slow process to discover key path parameters including a way to more rapidly grow the congestion window (cwnd):

1. During a previous session, 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. When resuming a session to the same IP address, the server can then utilize the current_rtt and the current_bb to the saved_rtt and saved_bb of a previous connection.

This method applies to any resumed QUIC session: both saved_session and recon_session can be a 0-RTT QUIC connection or a 1-RTT QUIC connection.

The current version of this draft considers several possible solutions: (1) the saved parameters are stored at the server; they are not sent to the client; (2) the saved parameters are sent to the client as an encrypted opaque blob; although the client is unable to read the parameters can include this opaque blob in a subsequent request to the server; (3) the saved parameters are sent to the client and the client is notified of their value, but the parameters also include a cryptographic integrity check; the client can include both the parameters and the integrity check in a subsequent request to the server.

None of these possible solutions allow q client to modify the parameters that will be used by the server.

There are several cases where the parameters of a previous session are not appropriate. These include:

(1) the network conditions have changed and the current capacity is less than the previously estimated bottleneck bandwidth. Using the saved congestion control state would increase congestion;

(2) the network path has changed and the new path is different. Using the saved congestion control state could increase congestion. This case might be accompanied by a change in the RTT or IP address.

(3) a client uses parameters that are no longer appropriate, e.g., to intentionally try to use a CWND larger than appropriate.

This document:

1. proposes guidelines for how to safely apply the previously computed parameters to new sessions;

2. describes different implementation considerations for the proposed method using QUIC;

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

1.1. Notations and terms

*IW: Initial Window (e.g., from [RFC6928]);

*current_iw: Current Initial Window

*recom_iw: Recommended Initial Window

*BDP: defined below

*CWND: the congestion window used by server (maximum number of bytes allowed in flight by the CC)

*current_bb : Current estimated bottleneck bandwidth

*saved_bb: Estimated bottleneck bandwidth preserved from a previous connection

*RTT: Round-Trip Time

*current_rtt: Current RTT

*saved_rtt: 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 the client preserved from a previous connection

*remembered BDP parameters: a combination of saved_rtt and saved_bb

*ITT : Interpacket Transmission Time

*MSS : Maximum Message Size

*AEAD : Authenticated Encryption with Associated Data

*LRU : Least Recently Used

[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." This draft considers the BDP estimated by a server that includes all buffering along the network path. In that sense, the BDP estimated is related to the amount of bytes in flight.

A QUIC connection might not reproduce the procedure detailed in [RFC6349] to measure the BDP. A server might be able to exploit an internal evaluation of the Bottleneck Bandwidth to estimate the BDP.

This document refers to the saved_bb and current_bb for the previously estimated bottleneck bandwidth. This value can be easilly estimated when using a rate-based congestion controller, such as BBR. Other congestion controllers, such as CUBIC or RENO, could estimate the bottleneck bandwidth by utilizing a combinatioin of the cwnd and the minimum RTT. This approach could result in over estimating the bottleneck bandwidth and ought to be used with caution.

1.2. 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. Safe jump start

2.1. Rationale behind the safety guidelines

The previously measured saved_rtt and saved_bb SHOULD NOT be used as-is, to avoid potential congestion collapse:

*Rationale #1: Internet path capacity can change at any time. An Internet method needs to be robust to network conditions that can differ from one session to the next.

*Rationale #2: Information sent by a malicious client is not relevant. A client could try to convince 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.

2.2. Rationale #1: Variable network conditions

The server MUST check the validity of the 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 use of these parameters is inappropriate:

- IP address changed: If the client changes its IP address (i.e. the saved_client_ip is different from the current_client_ip), the different address is to be takne as 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 changed: A significant change in RTT might be an indication that the the network conditions changed. Since the CC information is directly impacted by the RTT, a significant change in RTT is a strong indication that the previously estimated BDP parameters are likely to not be valid for the current path.

- Lifetime of the extension: The CC information is temporal. Frequent connections to the same IP address are likely to track changes, but long-term use of previous values are not appropriate.

- BB over estimation: There are cases where using the cwnd would infralte the bottleneck bandwidth. However, at the end of a CC slow start, the value of cwnd can be significantly larger than the value, that the CC finally converges to (after a few more rounds). Directly exploiting such value for the bottleneck bandwidth estimation may be inappropriate. One mitigation could be to 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 flightsize.

There are different solutions for the variable network conditions:

- Rationale #1 - Solution #1 : When resuming a session, 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 a session, 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 doing so.

Section 3 describes various approaches for Rationale #1 - Solution #2.

2.3. 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. This information is encrypted by the server. The client resends the same encrypted information when resuming a connection. The client can neither read nor modify the saved_rtt and saved_bb parameters.

- Rationale #2 - Solution #3 : The server sends an estimate of the saved_rtt and saved_bb parameters to the client. The information includes an integrity protection check. The client can resend the information when resuming a connection. This allows a client to read, but not modify, the saved_rtt and saved_bb parameters. This might enable a client to decide whether the new parameters are appropriate, based on client-side information about the network conidtions or connectivity.

Section 4 describes various implementation approaches for each of these solutions using local storage (Section 4.2 for Rationale #2 - Solution #1), NEW_TOKEN Frame (Section 4.3 for Rationale #2 - Solution #2), BDP extension Frame (Section 4.4 for Rationale #2 - Solution #3).

2.4. Trade-off between the different solutions

This section provides a description of different 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.

2.4.1. Security aspects

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 3, 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.

2.4.2. Interoperability and use-cases

If the server 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 the previous session of the client.

In cases where the BDP Frame extension is exploited, the approach of storing the BDP parameters locally at the server can provide a cross-check of the BDP parameters sent by a client. The server can anyway enable a safe jumpstart, but without the BDP Frame extension. However, the client does not have the choice of accepting to use this or not, and is unable to utilize local knowledge of the network conditions or conenctivity.

Storing local values related to the BDP would help in improving the ingress for 0-RTT connections, however, not using a BDP Frame extension could reduce the interest of the approach where (1) the client knows the BDP estimations done at the server, (2) the client decides to accept or reject ingress optimization, (3) the client tunes application level requests.

2.4.3. Summary

As a summary, the approach of 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.	Risks of adding congestion	MUST NOT implement
	#2 Implement safety check	Reduce risks 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

Figure 1: Comparing solutions

3. Safety guidelines

The safety guidelines are designed to avoid a server adding excessive congestion to an already congested path. The following mechanisms help in fulfilling this objective:

- The server SHOULD 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 server SHOULD NOT consider the saved_bb parameter when there is any indicated congestion (e.g., loss of packet during the first transmission of data or ECN-CE mark);

- The server MUST NOT send more than the recommended maximum IW (recom_iw) in the first transmission of data. This value could be based on a local understanding of the path characteristics. Knowing the congestion status of the network in closed environments may help in increasing the recommended maximum IW.

- The server SHOULD NOT store and/or send information related to the previously estimated bottleneck bandwidth (saved_bb) (see Section 1.1 for more details on bottleneck bandwidth definition), if this estimation has not been computed after some rounds during the 1-RTT connection. At least, the 1-RTT connection should have reached the congestion avoidance phase.

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 acknowledgment needs to be chosen to avoid the creation and the increase of congestion for the path.

This extension should 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].

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 where there is no congestion state. This avoids adding excessive congestion to the path;

2. If the reception of IW exhibits characteristics that resemble those of a recent previous session from the client (i.e. current_rtt < 1.2*saved_rtt and all data was acknowledged without reported congestion), the method permits the sender to consider 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 Interpacket Transmission Time (ITT) is determined by the ratio between the current Maximum Message Size (MSS) for packets and the ratio between the saved_bb and saved_rtt. A tunable safety margin might be introduced to 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 IW is acknowledged, the server falls back to a standard slow-start mechanism.

*Tune slow-start mechanisms: After transport parameters are set to a previously estimated bottleneck bandwidth, if slow-start mechanisms continue, the sender can overshoot the bottleneck capacity. This can occur even if the safety check described in this section is implemented.

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

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

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

4. Implementation considerations

4.1. Rationale behind the different implementation options

The NewSessionTickets messages of TLS offer a solution. The idea would have been to add a 'bdp_metada' field in the NewSessionTickets

that the client could read. The sole 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 the TLS stacks.

Three distinct approaches are presented: sending an opaque blob to the client that it may return to the server for a future connection (see Section 4.3), enable a local storage of BDP related values (see Section 4.2) and a BDP Frame extension (see Section 4.4).

4.2. Independent local storage of values

This approach independently lets both a client and a server remember 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 with 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 endpoint waits for the first RTT measurement from the peer's IP address. This is used to verify that the current_rtt has not significantly changed from the saved_rtt, and hence is an indication that the BDP information is appropriate to the path that is currently being used.

If this RTT is confirmed (e.g. current_rtt < 1.2*saved_rtt, the endpoint also verifies that an initial window of data has been acknowledged without requiring retransmission. This second check detects a path with significant incipient 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 the client to refuse the exploitation of 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 negociated that alows a client to choose whether to use the saved information.

4.3. Using NEW_TOKEN frames

Using NEW_TOKEN Frames, the server could send a token to the client through a NEW_TOKEN Frame. The token is an opaque 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 for a later connection.

4.4. BDP Frame

This section describes the use of a new Frame, the BDP Frame. The BDP Frame MUST be contained in 0-RTT packets, if sent by the client. The BDP Frame MUST be contained in 1-RTT packets, if sent by the server. The BDP Frame MUST be considered by congestion control and its data is not be limited by flow control limits. The server MAY send multiple BDP Frames in both 1-RTT and 0-RTT connections. The client can send BDP Frames during 1-RTT and 0-RTT connections.

4.4.1. BDP Frame Format

A BDP Frame is formatted as shown in Figure 2.

```
BDP Frame {
  Type (i) = 0xXXX,
  Lifetime (i),
  Saved BB (i),
  Saved RTT (i),
  Saved IP length (i),
  Saved IP (...)
}
```

Figure 2: BDP Frame Format

A BDP Frame contains the following fields:

- Lifetime (extension_lifetime): The extension_lifetime is a value in milliseconds, encoded as a variable length integer. This follows the idea of NewSessionTicket of TLS [RFC8446]. This represents the validity in time of this extension.

- Saved BB (saved_bb): The saved_bb is a value in bytes, encoded as a variable length integer. The bottleneck bandwidth estimated for the previous connection by the server. Using the previous values of bytes_in_flight defined in [RFC9002] can result in overshoot of the bottleneck capacity and is not advised.

- Saved RTT (saved_rtt): The saved_rtt is a value in milliseconds, encoded as a variable length integer. This could be set to the minimum RTT (min_rtt). The saved_rtt can be set to min_rtt. NOTE:

The min_rtt defined in [RFC9002], does not track a decreasing RTT: therefore min_rtt reported might be larger than the actual minimum RTT measured during the 1-RTT connection.

*Saved IP length (saved_ip_length) : The length of the IP address set to either 4 (IPv4) or 16 (IPv6).

*Saved IP (saved_client_ip) : The saved_client_ip could be set to the IP address of the client.

4.4.2. Extension activation

The client can accept the transmission of BDP Frames from the server by using the enable_bdp transport extension.

enable_bdp (0xTBD): in the 1-RTT connection, the client indicates to the server that it wishes to receive BDP extension Frames for improving ingress of 0-RTT connection. The default value is 0. Values strictly above 3 are invalid, and receipt of these values MUST be treated as a connection error of type TRANSPORT_PARAMETER_ERROR.

*0: Default value. If the client does not send this parameter, the server considers that the client does not support or does not wish to activate the BDP extension.

*1: The client indicates to the server that it wishes to receive BDP Frame and activates the ingress optimization for the 0-RTT connection.

*2: The client indicates that it does not wish to receive BDP Frames but activates ingress optimization.

*3: The client indicates that it wishes to receive BDP Frames but does not activate ingress optimization.

This Transport Parameter is encoded as per Section 18 of [RFC9000].

5. Discussion

5.1. BDP extension protected as much as initial_max_data

The BDP metadata parameters are measured by the server during a previous connection. The BDP extension is protected by the mechanism that protects the exchange of the 0-RTT transport parameters. For version 1 of QUIC, the BDP extension is protected using the mechanism that already protects the "initial_max_data" parameter. This is defined in sections 4.5 to 4.7 of [RFC9001]. This provides a way for the server to verify that the parameters proposed by the

client are the same as those that the server sent to the client during the previous connection.

5.2. Other use-cases

5.2.1. Optimizing client's requests

When using Dynamic Adaptive Streaming over HTTPS (DASH), clients might encounter issues in knowing the available path capacity or DASH can encounter issues in reaching the best available video playback quality. The client requests could then be adapted and specific traffic could utilize information from the path characteristics (such as encouraging the client to increase the quality of video chunks, to fill the buffers and avoid video blocking or to send high quality adds).

In other cases, applications could provide additional services if clients can know the server estimation of the path characteristics.

5.2.2. Sharing transport information across multiple connections

There can be benefit in sharing transport information across multiple connections. [I-D.ietf-tcpm-2140bis] considers the sharing of transport parameters between TCP connections originating from the same 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.

6. 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.

7. 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].

8. Security Considerations

Security considerations are discussed in Section 5 and in Section 3.

9. References

9.1. Normative References

[I-D.ietf-tcpm-2140bis] Touch, J., Welzl, M., and S. Islam, "TCP Control Block Interdependence", Work in Progress,

Internet-Draft, draft-ietf-tcpm-2140bis-11, 12 April 2021, <https://www.ietf.org/archive/id/draft-ietf-tcpm-2140bis-11.txt>.

[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>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[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>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/info/rfc9001>.

[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>.

9.2. Informative References

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

[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>.

Authors' Addresses

Nicolas Kuhn
CNES

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