Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 17, 2018

M. Kuehlewind
B. Trammell
ETH Zurich
J. Hildebrand

November 13, 2017

Transport-Independent Path Layer State Management
draft-trammell-plus-statefulness-04

Abstract

This document describes a simple state machine for stateful network devices on a path between two endpoints to associate state with traffic traversing them on a per-flow basis, as well as abstract signaling mechanisms for driving the state machine. This state machine is intended to replace the de-facto use of the TCP state machine or incomplete forms thereof by stateful network devices in a transport-independent way, while still allowing for fast state timeout of non-established or undesirable flows.

Table of Contents

1. Introduction

The boundary between the network and transport layers was originally defined to be that between information used (and potentially modified) hop-by-hop, and that used end-to-end. End-to-end information in the transport layer is associated with state at the endpoints, but processing of network-layer information was assumed to be stateless.

The widespread deployment of stateful middleboxes in the Internet, such as network address and port translators (NAPT), firewalls that model the TCP state machine to distinguish packets belonging from desirable flows from backscatter and random attack traffic, and devices which keep per-flow state for reporting and monitoring

purposes (e.g. IPFIX [RFC7011] Metering Processes), has broken this assumption, and made it more difficult to deploy non-TCP transport protocols in the Internet.

The deployment of new transport protocols encapsulated in UDP with encrypted transport headers (such as QUIC [I-D.ietf-quic-transport]) will present a challenge to the operation of these devices, and their ubquity likewise threatens to impair the deployability of these protocols. There are two main causes for this problem: first, stateful devices often use an internal model of the TCP state machine to determine when TCP flows start and end, allowing them to manage state for these flows; for UDP flows, they must rely on timeouts. These timeouts are generally short relative to those for TCP [IMC-GATEWAYS], requiring UDP- encapsulated transports either to generate unproductive keepalive traffic for long-lived sessions, or to tolerate connectivity problems and the necessity of reconnection due to loss of on-path state.

This document presents an abstract solution to this problem by defining a transport-independent state machine to be implemented at per-flow state- keeping middleboxes as a replacement for incomplete TCP state modeling. A key concept behind this approach is that encryption of transport protocol headers allows a transport protocol to separate its wire image - what it looks like to devices on path - from its internal semantics. We advocate the creation of a minimal wire image for these protocols that exposes enough information to drive the state machine presented. Present and future evolution of encrypted transport protocols can then happen behind this wire image, and Middleboxes implementing this state machine can use signals from a UDP encapsulation common to a set of encrypted transport protocols can have equivalent state information to that provided by TCP, reducing the friction between deployed middleboxes and these new transport protocols.

2. Terminology

In this document, the term "flow" is defined to be compatible with the definition given in [RFC7011]: A flow is defined as a set of packets passing a device on the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties. Each property is defined as the result of applying a function to the values of:

1. one or more network layer header fields (e.g., destination IP address) or transport layer header fields (e.g., destination port number) that the device has access to;

2. one or more characteristics of the packet itself (e.g., number of MPLS labels, etc.);

3. one or more of the fields derived from packet treatment at the device (e.g., next-hop IP address, the output interface, etc.).

A packet is defined as belonging to a flow if it completely satisfies all the defined properties of the flow.

A bidirectional flow or biflow is defined as compatible with [RFC5103], by joining the "forward direction" flow with the "reverse direction" flow, derived by reversing the direction of directional fields (ports and IP addresses). Biflows are only relevant at devices positioned so as to see all the packets in both directions of the biflow, generally on the endpoint side of the service demarcation point for either endpoint as defined in the reference path given in [RFC7398].

3. State Machine

A transport-independent state machine for on-path devices is shown in Figure 1. It was designed to have the following properties:

- A device on path that can see traffic in both directions between two endpoints knows that each side of an association wishes that association to continue. This allows firewalls to delegate policy decisions about accepting or continuing an association to the servers they protect.

- A device on path that can see traffic in both directions between two endpoints knows that each device can receive traffic at the source address it provides. This allows firewalls to provide protection against trivially spoofed packets.

Both of these properties hold with current firewalls and network address translation devices observing the flags and sequence/ acknowledgment numbers exposed by TCP.

It relies on six states, three configurable timeouts, and a set of signals defined in Section 4. The states are defined as follows:

- zero: there is no state for a given flow at the device

- uniflow: at least one packet has been seen in one direction

- associating: at least one packet has been seen in one direction, and an indication that the receiving endpoint wishes to continue the association has been seen in the other direction.

- associated: a flow in associating state has further demonstrated that the initial sender can receive packets at its given source address.

- stop-wait: one side of a connection has sent an explicit stop signal, waiting for confirmation

- stopping: stop signal confirmed, association is stopping.

We refer to the zero and uniflow states as "uniflow states", as they are relevant both for truly unidirectional flows, as well as in situations where an on-path device can see only one side of a communication. We refer to the remaining four states as "biflow states", as they are only applicable to true bidirectional flows, where the on-path device can see both sides of the communication.

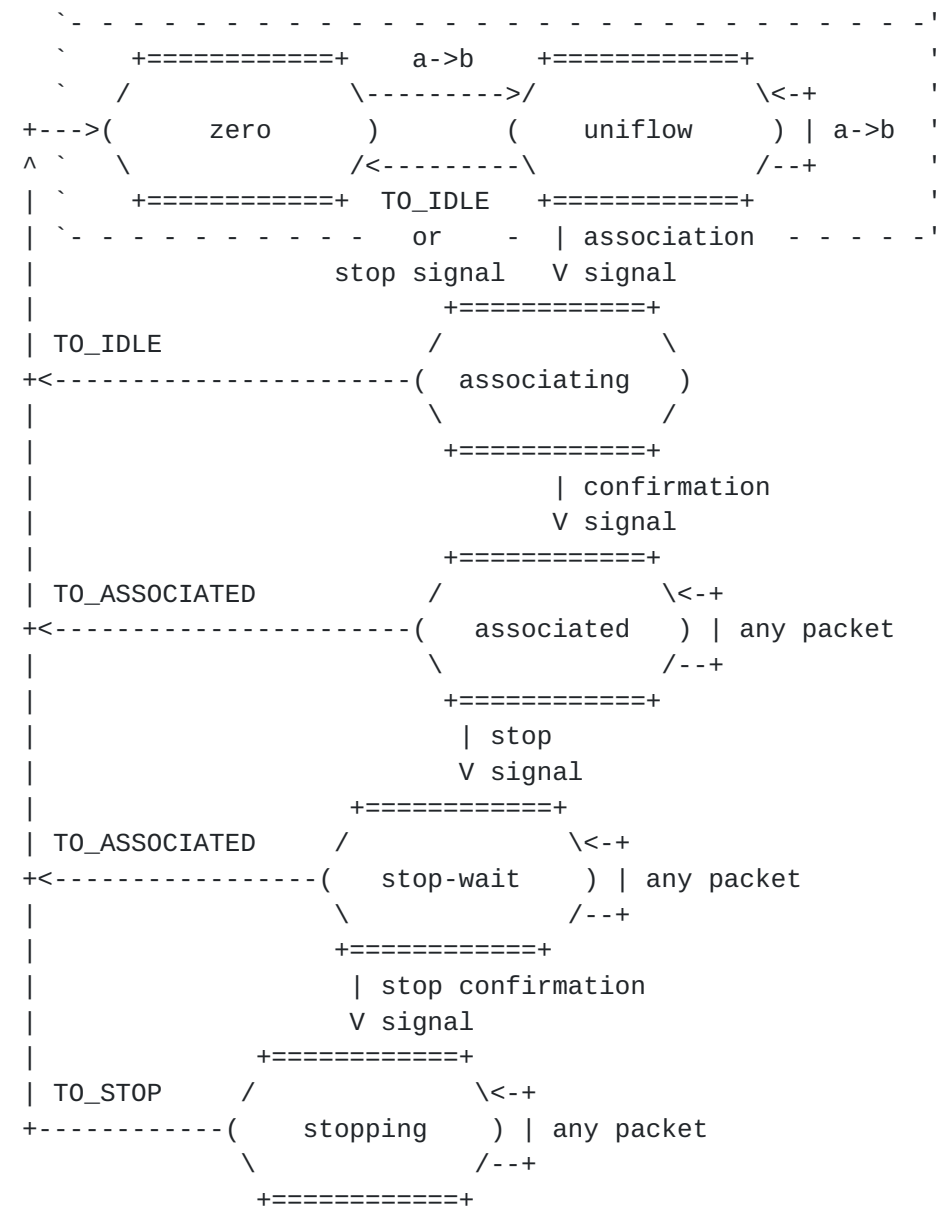

Figure 1: Transport-Independent State Machine for Stateful On-Path Devices

The three timeouts are defined as follows:

- TO_IDLE, the unidirectional idle timeout, can be considered equivalent to the idle timeout for transport protocols where the device has no information about session start and end (e.g. most UDP protocols).

- TO_ASSOCIATED, the bidirectional idle timeout, can be considered equivalent to the timeout for transport protocols where the device has information about session start and end (e.g. TCP).

- o TO_STOP is the teardown timeout: how long the device will account additional packets to a flow after confirming a close signal, ensuring retransmitted and/or reordered close signal don't lead to the spurious creation of new flow state.

Selection of timeouts is a configuration and implementation detail, but generally TO_STOP <= TO_IDLE << TO_ASSOCIATED; see [IMC-GATEWAYS] for an analysis of the magnitudes of these timeouts in presently deployed gateway devices.

3.1. Uniflow States

Every packet received by a device keeping per-flow state must associate that packet with a flow (see Section 4.1). When a device receives a packet associated with a flow it has no state for, and it is configured to forward the packet instead of dropping it, it moves that flow from the zero state into the uniflow state and starts a timer TO_IDLE. It resets this timer for any additional packet it forwards in the same direction as long as the flow remains in the uniflow state. When timer TO_IDLE expires on a flow in the uniflow state, the device drops state for the flow and performs any processing associated with doing so: tearing down NAT bindings, stopping associated firewall pinholes, exporting flow information, and so on. The device may also drop state on a stop signal, if observed.

Some devices will only see one side of a communication, e.g. if they are placed in a portion of a network with asymmetric routing. These devices use only the zero and uniflow states (as marked in Figure 1.) In addition, true uniflows - protocols which are solely unidirectional (e.g. some applications over UDP) - will also use only the uniflow-only states. In either case, current devices generally don't associate much state with observed uniflows, and an idle timeout is generally sufficient to expire this state.

3.2. Biflow States

A uniflow transitions to the associating state when the device observes an association signal, and further to the associated state when the device observes a subsequent confirmation signal; see Section 4.2 for details. If the flow has not transitioned to from the associating to the associated state after TO_IDLE, the device drops state for the flow.

After transitioning to the associated state, the device starts a timer TO_ASSOCIATED. It resets this timer for any packet it forwards in either direction. The associated state represents a fully established bidirectional communication. When timer TO_ASSOCIATED

expires, the device assumes that the flow has shut down without signaling as such, and drops state for the flow, performing any associated processing. When a bidirectional stop signal (see Section 4.3) is confirmed, the flow transitions to the stopping state.

When a flow enters the stopping state, it starts a timer TO_STOP. While the stop signal should be the last packet on a flow, the TO_STOP timer ensures that reordered packets after the stop signal will be accounted to the flow. When this timer expires, the device drops state for the flow, performing any associated processing.

3.3. Additional States and Actions

This document is concerned only with states and transitions common to transport- and function- independent state maintenance. Devices may augment the transitions in this state diagram depending on their function. For example, a firewall that decides based on some information beyond the signals used by this state machine to shut down a flow may transition it directly to a blacklist state on shutdown. Or, a firewall may fail to forward additional packets in the uniflow state until an association signal is observed.

4. Abstract Signaling Mechanisms

The state machine in Section 3 requires four signals: a new flow signal, the first packet observed in a flow in the zero state; an association signal, allowing a device to verify that an endpoint wishes a bidirectional communication to be established or to continue; a confirmation signal, allowing a device to confirm that the initiator of a flow is reachable at its purported source address; and a stop signal, noting that an endpoint wishes to stop a bidirectional communication. Additional related signals may also be useful, depending on the function a device provides. There are a few different ways to implement these signals; here, we explore the properties of some potential implementations.

We assume the following general requirements for these signals; parallel to those given in [draft-trammell-plus-abstract-mech]:

- At least the endpoints can verify the integrity of the signals exposed, and shut down a transport association when that verification fails, in order to reduce the incentive for on-path devices to attempt to spoof these signals.

- Endpoints and devices on path can probabilistically verify that a originator of a signal is on-path.

4.1. Flow Identification

In order to keep per-flow state, each device using this state machine must have a function it can apply to each packet to be able to extract common properties to identify the flow it is associated with. In general, the set of properties used for flow identification on presently deployed devices includes the source and destination IP address, the source and destination transport layer port number, the transport protocol number. The differentiated services field [RFC2474] may also be included in the set of properties defining a flow, since it may indicate different forwarding treatment.

However, other protocols may use additional bits in their own headers for flow identification. In any case, a protocol implementing signaling for this state machine must specify the function used for flow identification.

4.2. Association and Confirmation Signaling

An association signal indicates that the endpoint that received the first packet seen by the device has indeed seen that packet, and is interested in continuing conversation with the sending endpoint. This signal is roughly an in-band analogue to consent signaling in ICE [RFC7675] that is carried to every device along the path.

A confirmation signal indicates that the endpoint that sent the first packet seen by the device is reachable at its purported source address, and is necessary to prevent spoofed or reflected packets from driving the state machine into the associated state. It is roughly equivalent to the final ACK in the TCP three-way handshake.

These two signals are related to each other, in that association requires the receiving endpoint of the first packet to prove it has seen that packet (or a subsequent packet), and to acknowledge it wants to continue the association; while confirmation requires the sending endpoint to prove it has seen the association token.

Transport-independent, path-verifiable association and confirmation signaling can be implemented using three values carried in the packet headers: an association token, a confirmation nonce, and an echo token.

The association token is a cryptographically random value generated by the endpoint initiating a connection, and is carried on packets in the uniflow state. When a receiving endpoint wishes to send an association signal, it generates an echo token from the association token using a well-known, defined function (e.g. a truncated SHA-256 hash), and generates a cryptographically random confirmation nonce.

The initiating endpoint sends a confirmation signal on the next packet it sends after receiving the confirmation nonce, by applying a function to the echo token and the confirmation nonce, and sending the result as a new association token.

Devices on path verify that the echo token corresponds to a previously seen association token to recognize an association signal, and recognize that an association token corresponds to a previously seen echo token and confirmation nonce to recognize an association signal.

If the association token and confirmation nonce are predictable, off-path devices can spoof association and confirmation signals. In choosing the number of bits for an association token, there is a tradeoff between per-packet overhead and state overhead at on-path devices, and assurance that an association token is hard to guess. This tradeoff must be evaluated at protocol design time.

There are a few considerations in choosing a function (or functions) to generate the echo token from the association token, to verify an echo token given an association token, and to derive a next association token from the echo token and confirmation nonce. The functions could be extremely simple (e.g., identity for the echo token and addition for the nonce) for ease of implementation even in extremely constrained environments. Using one-way functions (e.g., truncated SHA-256 hash to derive echo token from association token; XOR followed by truncated SHA-256 hash to derive association token from echo token and confirmation nonce) requires slightly more work from on-path devices, but the primitives will be available at any endpoint using an encrypted transport protocol. In any case, a concrete implementation of association and confirmation signaling must choose a set of functions, or mechanism for unambiguously choosing one, at both endpoints as well as along the path.

4.2.1. Start-of-flow versus continual signaling

There are two possible points in the design space here: these signals could be continually exposed throughout the flow, or could be exposed only on the first few packets of a connection (those corresponding to the cryptographic and/or transport state handshakes in the overlying protocols).

In the former case, an on-path device could re-establish state in the middle of a flow; e.g. due to a reboot of the device, due to a NAT association change without the endpoints' knowledge, or due to idle periods longer than the TO_ESTABLISHED timeout value. The on-path device would receive no special information about which packets were associated with the start of association. In this case, the series

of exposed association tokens, echo tokens, and confirmation nonces can also be observed to derive a running round-trip time estimate for the flow.

In the latter case, an on-path device would need to observe the start of the flow to establish state, and would be able to distinguish connection-start packets from other packets.

4.3. Bidirectional Stop Signaling

The transport-independent state machine uses bidirectional stop signaling to tear down state. This requires a stop signal to be observed in one direction, and a stop confirmation signal to be observed in the other, to complete tearing down an association.

A stop signal is directly carried or otherwise encoded in the protocol header to indicate that a flow is ending, whether normally or abnormally, and that state associated with the flow should be torn down. Upon decoding a stop signal, a device on path should move the flow from uniflow state to zero, or from associated state to stop-wait state, to wait for a confirmation signal in the other direction. While in stop-wait state, state will be maintained until a timer set to TO_ASSOCIATED expires, with any packet forwarded in either direction reseting the timer.

A stop confirmation signal is directly carried or otherwise encoded in the protocol header to indicate that the endpoint receiving the stop signal confirms that the stop signal is valid. The stop confirmation signal contains some assurance that the far endpoint has seen the stop signal. When a stop confirmation signal is observed in the opposite direction from the stop signal, a device on path should move the flow from stop-wait state to stopping state. The flow will then remain in stopping state until a timer set to TO_STOP has expired, after which state for the flow will be dropped. The stopping timeout TO_STOP is intended to ensure that any packets reordered in delivery are accounted to the flow before state for it is dropped.

We assume the encoding of stop and stop confirmation signals into a packet header, as with all other signals, is integrity protected end-to-end. Stop signals, as association signals, could be forged by a single on-path device. However, unless a stop confirmation signal that can be associated with the stop signal is observed in the other direction, the flow remains in stop-wait state, during which state is maintained and packets continue to be forwarded in both directions. So this attack is of limited utility; an attacker wishing to inject state teardown would need to control at least one on-path device on

each side of a target device to spoof both stop and corresponding stop confirmation signals.

4.3.1. Authenticated Stop Signaling

Additionally, the stop and stop confirmation signals could be designed to authenticate themselves. Each endpoint could reveal a stop hash during the initial association, which is the result of a chosen cryptographic hash function applied to a stop token which that endpoint keeps secret. An endpoint wishing to end the association then reveals the stop token, which can be verified both by the far endpoint and devices on path which have cached the stop hash to be authentic. A stop confirmation signal additionally contains information derived from the initiating stop signal's stop token, as further assurance that the stop token was observed by the far endpoint.

4.4. Separate Utility

Although all of these signals are required to drive the state machine described by this document, note that association/confirmation and bidirectional stop signaling have separate utility. A transport protocol may expose the end of a flow without any proof of association or confirmation of return routability of the initiator. Alternately, the transport protocol could rely on short timeouts to clean up stale state on path, while exposing continuous association and confirmation signals to quickly reestablish state.

5. Deployment Considerations

The state machine defined in this document is most useful when implemented in a single instantiation (wire format for signals, and selection of functions for deriving values to be exposed and verified) by multiple transport protocols. It is intended for use with protocols that encrypt their transport- layer headers, and that are encapsulated within UDP, as is the case with QUIC [I-D.ietf-quic-transport]. Definition of that instantiation is out of scope for the present revision of this document.

The following subsections discuss incentives for deployment of this state machine both at middleboxes and at endpoints.

5.1. Middlebox Deployment

The state machine defined herein is designed to replace TCP state-tracking for firewalls and NAT devices. When encrypted transport protocols encapsulated in UDP adopt a set of signals and a wire format for those signals to drive this state machine, these

middleboxes could continue using TCP-like logic to handle those UDP flows. Recognizing the wire format used by those signals would allow these middleboxes to distinguish "UDP with an encrypted transport" from undifferentiated UDP, and to treat the former case more like TCP, providing longer timeouts for established flows, as well as stateful defense against spoofed or reflected garbage traffic.

5.2. Endpoint Deployment

An encrypted, UDP-encapsulated transport protocol has two primary incentives to expose these signals. First, allowing firewalls on networks that generally block UDP (about 3-5% of Internet-connected networks, depending on the study) to distinguish "UDP with an encrypted transport" traffic from other UDP traffic may result in less blocking of that traffic. Second, the difference between the timeouts TO_IDLE and TO_ASSOCIATED, as well as the continuous state establishment possible with some instantiations of the association and confirmation signals, would allow these transport protocols to send less unproductive keepalive traffic for long-lived, sparse flows.

While both of these advantages require middleboxes on path to recognize and use the signals driving this state machine, we note that content providers driving the deployment of this protocols are also operators of their own content provision networks, and that many of the benefits of encrypted- encapsulated transport firewalls will accrue to them, giving these content providers incentives to deploy both endpoints and middleboxes.

6. Signal mappings for transport protocols

We now show how this state machine can be driven by signals available in TCP and QUIC.

6.1. Signal mapping for TCP

A mapping of TCP flags to transitions in to the state machine in Section 3 shows how devices currently using a model of the TCP state machine can be converted to use this state machine.

TCP [RFC0793] provides start-of-flow association only. A packet with the SYN and ACK flags set in the absence of the FIN or RST flags, and an in-window acknowledgment number, is synonymous with the association signal. A packet with the ACK flag set in the absence of the FIN or RST flags after an initial SYN, and an in-window acknowledgment number, is synonymous with the confirmation signal. For a typical TCP flow:

1. The initial SYN places the flow into uniflow state,

2. The SYN-ACK sent in reply acts as a association signal and places the flow into associating state,

3. The ACK sent in reply acts as a confirmation signal and places the flow into associated state,

4. The final FIN is a stop signal, and

5. the ACK of the final FIN is a stop confirmation signal, moving the flow into stopping state.

Note that abormally closed flows (with RST) do not provide stop confirmation, and are therefore not provided for by this state machine. Due to TCP's support for half-closed flows, additional state modeling is necessary to extract a stop signal from the final FIN.

Note also that the association and stop signals derived from the TCP header are not integrity protected, and association and confirmation signals based on in-window ACK are not particularly resistant to off-path attacks [IMC-TCP]. The state machine is therefore more susceptible to manipulation when used with vanilla TCP as when with a transport protocol providing full integrity protection for its headers end-to-end.

6.2. Signal mapping for QUIC

QUIC [I-D.ietf-quic-transport] is a moving target; however, signals for driving this state machine are fundamentally compatible with the protocol's design and could easily be added to the protocol specification.

Specifically, QUIC's handshake is visible to on-path devices, as it begins with an unencrypted version negotiation which exposes a 64-bit connection ID, which can serve as an association and echo token as in Section 4.2. The function of the confirmation nonce is not fully exposed to the path at this point, but could be implemented by exposing information from the proof of source address ownership (section 7.4 of [I-D.ietf-quic-transport]) or via echoing the random initial packet number (as suggested by https://github.com/quicwg/base-drafts/pull/391).

The addition of a public reset signal that would act as a stop signal as in Section 4.3 is presently under discussion within the QUIC working group; the proposal for self-authenticating public reset at

https://github.com/quicwg/base-drafts/pull/20 inspired the addition of Section 4.3.1 to this document.

7. IANA Considerations

This document has no actions for IANA.

8. Security Considerations

This document defines a state machine for transport-independent state management on middleboxes, using in-band signaling, to replace the commonly- implemented current practice of incomplete TCP state modeling on these devices. It defines new signals for state management. While these signals can be spoofed by any device on path that observes traffic in both directions, we presume the presence of end-to-end integrity protection of these signals provided by the upper-layer transport driving them. This allows such spoofing to be detected and countered by endpoints, reducing the threat from on-path devices to connection disruption, which such devices are trivially placed to perform in any case.

9. Acknowledgments

Thanks to Christian Huitema for discussions leading to this document, and to Andrew Yourtchenko for the feedback. The mechanism for using a revealed value to prove ownership of a stop token was inspired by Eric Rescorla's suggestion to use a fundamentally identical mechanism for the QUIC public reset.

This work is partially supported by the European Commission under Horizon 2020 grant agreement no. 688421 Measurement and Architecture for a Middleboxed Internet (MAMI), and by the Swiss State Secretariat for Education, Research, and Innovation under contract no. 15.0268. This support does not imply endorsement.

10. References

10.1. Normative References

[RFC5103] Trammell, B. and E. Boschi, "Bidirectional Flow Export Using IP Flow Information Export (IPFIX)", RFC 5103, DOI 10.17487/RFC5103, January 2008, <https://www.rfc-editor.org/info/rfc5103>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[RFC7398] Bagnulo, M., Burbridge, T., Crawford, S., Eardley, P., and A. Morton, "A Reference Path and Measurement Points for Large-Scale Measurement of Broadband Performance", RFC 7398, DOI 10.17487/RFC7398, February 2015, <https://www.rfc-editor.org/info/rfc7398>.

10.2. Informative References

[draft-trammell-plus-abstract-mech]
Trammell, B., "Abstract Mechanisms for a Cooperative Path Layer under Endpoint Control", September 2016.

[I-D.hardie-path-signals]
Hardie, T., "Path signals", draft-hardie-path-signals-01 (work in progress), May 2017.

[I-D.ietf-quic-tls]
Thomson, M. and S. Turner, "Using Transport Layer Security (TLS) to Secure QUIC", draft-ietf-quic-tls-07 (work in progress), October 2017.

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-07 (work in progress), October 2017.

[IMC-GATEWAYS]
Hatonen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., and M. Kojo, "An experimental study of home gateway characteristics (Proc. ACM IMC 2010)", October 2010.

[IMC-TCP] Luckie, M., Beverly, R., Wu, T., Allman, M., and k. claffy, "Resilience of Deployed TCP to Blind Attacks. (Proc. ACM IMC 2015)", October 2015.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <https://www.rfc-editor.org/info/rfc7675>.

Authors' Addresses

Mirja Kuehlewind
ETH Zurich
Gloriastrasse 35
8092 Zurich
Switzerland

Email: mirja.kuehlewind@tik.ee.ethz.ch

Brian Trammell
ETH Zurich
Gloriastrasse 35
8092 Zurich
Switzerland

Email: ietf@trammell.ch

Joe Hildebrand

Email: hildjj@cursive.net