Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 14, 2019

T. Pauly
E. Kinnear
Apple Inc.

September 10, 2018

An Interface to the QUIC Transport Protocol
draft-pauly-quic-interface-00

Abstract

This document defines the abstract application interface to the QUIC transport protocol. This allows applications to use a standard interface to directly interact with the QUIC protocol for cases that may not be using an HTTP mapping.

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 March 14, 2019.

Table of Contents

1. Introduction

The QUIC Transport Protocol [I-D.ietf-quic-transport] defines a mechanism for allowing applications to communicate as clients or servers to securely send and receive data over multiplexed streams associated with a single cryptographic state. While some applications may not need to directly interact with QUIC as a transport if they use HTTP over QUIC, others will need to send and recieve data directly over the transport.

Defining a standard application interface to QUIC as a transport has several benefits to applications as they adopt the protocol. These benefits are expressed in the following requirements for a transport interface to QUIC:

- Many of the benefits of QUIC, such as reducing head-of-line blocking or being able to send zero-RTT data, can be lost if the transport API does not provide adequate support. The interface SHOULD allow such features to be accessed in a reliable fashion.

- Various implementations of the QUIC protocol SHOULD provide similar transport interfaces in order to allow applications to easily adopt them across different platforms and deployments.

- The interface to configure QUIC security properties SHOULD be restricted to a standard set of functionality to ensure that applications cannot easily diminish the security properties of the protocol, while still retaining control over the configuration.

1.1. Specification of Requirements

The key words "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. Background

2.1. QUIC Streams

QUIC defines the concept of "streams" of data. Streams may be either bidirectional or unidirectional. For both cases, data in each direction is treated as a reliable in-order sequence of bytes. Streams are transmitted without head-of-line blocking between one another, although data frames for multiple streams may be consolidated into single packets.

2.2. Transport Connection Interface

The design of this abstract interface is intended to be compatible with the Transport Services Abstract Interface [I-D.ietf-taps-interface], although it can be used independently. The Transport Services interface defines a Connection as an active transport protocol instance that can send and/or receive Messages between a Local Endpoint and a Remote Endpoint.

To avoid confusion with the notion of a QUIC Connection, this definition of a Connection will be referred to as a "Transport Connection Object" in this document.

The portions of data that are transferred over the Transport Connection are referred to as Messages. Messages that are sent and received may be associated with metadata and properties in addition to their raw bytes.

2.3. Transferring Messages over Streams

The QUIC stream architecture provides several possible mappings of application data into separate streams. Consider a case in which an application wants to send ten separate messages to a peer, and some of these messages expect replies from the remote host. There are two high-level strategies for mapping these messages over QUIC streams:

1. Each message sent is represented as a new stream, and consumes a Stream ID. Any message that expects a reply can use a bidirectional stream, and data sent back on that stream will be

interpreted as the reply. This stategy is descibed in Section 3.1.

2. The messages are sent all over a single bidirectional stream, in order. This requires that the messages are able to encode their boundaries within the byte stream, as well as some message identifier or ordering guarantee to allow correlation of replies with outbound messages. This stategy is descibed in Section 3.2.

Strategy (1) relies on QUIC for message delineation and correlation; this may be a benefit if an application does not already define its own message framing. However, if messages already define message boundaries and semantics, strategy (2) may be less redundant.

Strategy (1) allows all messages to be delivered without head-of-line blocking, which may be beneficial if there are delays on one stream. However, this approach does not provide any ordering guarantees. If an application will only be able to process messages in a strict order, strategy (2) may be preferable.

3. API Mappings

3.1. Transport Connection as QUIC Connection

The mapping of a Transport Connection Object onto an entire QUIC Connection SHOULD be exposed to applications as the "quic-connection" protocol.

When this protocol is part of a Protocol Stack being used for a Transport Connection, either as the top-level protocol (the one that the application directly interacts with) or as a protocol in the middle of the stack, each Message object corresponds to a QUIC stream. The description of this mapping will refer to the interation model as if the quic-connection protocol is being used as a top-level protocol.

The following API mappings are defined:

Initiate: When an application calls Initiate on an outbound connection, and the quic-connection protocol is being used, QUIC MUST initiate its handshake with the Remote Endpoint. The Ready event will be delivered once the handshake is complete and 1-RTT keys have been established.

Close: When the application calls Close on its connection, QUIC MUST send a CONNECTION_CLOSE frame to the endpoint if it is currently active.

Send: When the application sends a new Message, QUIC MUST allocate a new stream ID. A metadata option SHOULD be exposed to allow the application to specify whether or not it expects a reply. If it does, the QUIC stream will be bidirectional; if not, the QUIC stream will be unidirectional. The API SHOULD allow the application to configure a default directionality setting on the connection to apply to default Messages. Any data sent associated with the Message Send should be sent in a QUIC stream frame for the new stream ID.

Send Idempotent: When the application sends a Message, it may mark it as idempotent, which makes the data eligible for sending under 0-RTT keys.

Send End-of-Message: When the application marks the end of a Message, which may be done as part of the first call to Send, or a subsequent call, the associated QUIC stream MUST deliver a FIN.

Receive: A call to receive new data from a Connection will invoke the Received event upon receiving new stream data. If a new stream is received from the peer, the Received event will be associated with a new Message object. If the stream data is marked with a FIN, the Received event will indicate that the Message is complete; otherwise, it will indicate that the Message has received partial data. As new data arrives on various streams, Received events will be delivered for various streams, and may result in partial receives be interleaved with one another. If an application does not wish to ever receive partial Messages, it can indicate that in the call to Receive; this means that data will only be delivered on behalf of a QUIC stream once the FIN bit has been received.

3.2. Transport Connection as QUIC Stream

The mapping of each Transport Connection Object onto an single QUIC Stream SHOULD be exposed to applications as the "quic-stream" protocol.

Messages in this mapping are transferred as in-order chunks of data over a stream represented by the Connection. This Connection is equivalent in contract to a TLS or TCP stream in many ways. The description of this mapping will refer to the interation model as if the quic-stream protocol is being used as a top-level protocol.

The following API mappings are defined:

Initiate: When an application calls Initiate on an outbound connection, QUIC MUST both start a new handshake with the remote

endpoint and also allocate a new stream ID to be associated with the Transport Connection Object.

Clone: When an application calls Clone on an existing outbound Transport Connection Object, and the QUIC connection is not already closed, QUIC MUST allocate a new stream ID and associate that stream with a new Transport Connection Object.

Close: If the application calls Close on a connection, QUIC MUST send a FIN on the associated stream if it it has not been marked previously.

Send: When the application sends a new Message, QUIC MUST send that data on the associated stream. If the application is using a framing protocol on top of quic-stream, then the message boundaries may be interpreted by the framing protocol. Otherwise, the end of a Message will have no impact on the frames being send by QUIC, unless that Message is also marked Final, in which case the QUIC stream MUST send a FIN.

Send Idempotent: When the application sends a Message, it may mark it as idempotent, which makes the data eligible for sending under 0-RTT keys.

Receive: A call to Receive will enqueue a request to receive data on the associated QUIC stream only. Once new data is available on the stream, or the stream is remotely closed, the Received event MUST be invoked. If the stream is not allowed to receive data, since it is unidirectional, the Receive call MUST result in a Received event delivering an error.

4. Racing QUIC Connections

When a QUIC hostname endpoint is resolved using DNS, a client may want to use the Happy Eyeballs algorithm [RFC8305] to race connections to the various IPv6 and IPv4 addresses returned by the DNS resolver.

If multiple connection attempts are run in parallel, the end of the "race" can be determined in one of two ways:

1. The race ends at the completion of the QUIC handshake, once 1-RTT keys are established.

2. The race ends upon successful reception the first Handshake Packet received from the server.

The first strategy results in potentially longer overlap of connection attempts, but guarantees that the chosen connection instance completes authentication. Thus, the first option SHOULD be used when possible. This also means that the API for QUIC as a transport SHOULD support multiple handshakes running in parallel for the duration of the Happy Eyeballs race. If the application needs to be involved in Identity Valdiation, then it may need to validate identities multiple times for a process that results in a single transport connection.

5. Security Considerations

The security interface exposed for QUIC as a transport SHOULD be expressed in terms of minimal interactions required for correct behavior. Functionality that MUST be exposed includes Identity Validation (to allow the application to validate a certificate). Functionality that SHOULD NOT be exposed include direct key export for negotiated keys.

6. IANA Considerations

This document has no request to IANA.

7. Acknowledgments

Thanks to members of the TAPS working group who helped design and review these mappings.

8. Informative References

[I-D.ietf-quic-applicability]
Kuehlewind, M. and B. Trammell, "Applicability of the QUIC Transport Protocol", draft-ietf-quic-applicability-02 (work in progress), July 2018.

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-14 (work in progress), August 2018.

[I-D.ietf-taps-interface]
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P., and C. Wood, "An Abstract Application Layer Interface to Transport Services", draft-ietf-taps-interface-01 (work in progress), July 2018.

[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>.

[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>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: tpauly@apple.com

Eric Kinnear
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: ekinnear@apple.com