Network Working Group
Internet-Draft
Intended status: Experimental
Expires: June 26, 2021

N. Banks
Microsoft Corporation

December 23, 2020

QUIC Performance
draft-banks-quic-performance-00

Abstract

The QUIC performance protocol provides a simple, general-purpose protocol for testing the performance characteristics of a QUIC implementation. With this protocol a generic server can support any number of client-driven performance tests and configurations. Standardizing the performance protocol allows for easy comparisons across different QUIC implementations.

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 June 26, 2021.

Table of Contents

1. Introduction

The various QUIC implementations are still quite young and not exhaustively tested for many different performance heavy scenarios. Some have done their own testing, but many are just starting this process. Additionally, most only test the performance between their own client and server. The QUIC performance protocol aims to standardize the performance testing mechanisms. This will hopefully achieve the following:

- Remove the need to redesign a performance test for each QUIC implementation.
- Provide standard test cases that can produce performance metrics that can be easily compared across different configurations and implementations.
- Allow for easy cross-implementation performance testing.

1.1. Terms and Definitions

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. Specification

The sections below describe the mechanisms used by a client to connect to a QUIC perf server and execute various performance scenarios.

2.1. Protocol Negotiation

The ALPN used by the QUIC performance protocol is "perf". It can be used on any UDP port, but UDP port 443 is used by default, if no other port is specified. No SNI is required to connect, but may be optionally provided if the client wishes.

2.2. Configuration

TODO - Possible options: use the first stream to exchange configurations data OR use a custom transport parameter.

2.3. Streams

The performance protocol is primarily centered around sending and receiving data. Streams are the primary vehicle for this. All performance tests are client-driven:

- The client opens a stream.
- The client encodes the size of the requested server response.
- The client sends any data it wishes to.
- The client cleanly closes the stream with a FIN.

When a server receives a stream does the following:

- The server accepts the new stream.
- The server processes the encoded response size.
- The server drains the rest of the client data.

o The server then sends any response payload that was requested.

Note - Should the server wait for FIN before replying?

2.3.1. Encoding Server Response Size

Every stream opened by the client uses the first 8 bytes of the stream data to encode a 64-bit unsigned integer in network byte order to indicate the length of data the client wishes the server to respond with. An encoded value of zero is perfectly legal, and a value of MAX_UINT64 (0xFFFFFFFFFFFFFFFF) is practically used to indicate an unlimited server response. The client may then cancel the transfer at its convenience with a STOP_SENDING frame.

On the server side, any stream that is closed before all 8 bytes are received should just be ignored, and gracefully closed on its end (if applicable).

2.3.2. Bidirectional vs Unidirectional Streams

When a client uses a bidirectional stream to request a response payload from the server, the server sends the requested data on the same stream. If no data is requested by the client, the server merely closes its side of the stream.

When a client uses a unidirectional stream to request a response payload from the server, the server opens a new unidirectional stream to send the requested data. If no data is requested by the client, the server need take no action.

3. Example Performance Scenarios

All stream payload based tests below can be achieved either with bidirectional or unidirectional streams. Generally, the goal of all these performance tests is to measure the maximum load that can be achieved with the given QUIC implementation and hardware configuration. To that end, the network is not expected to be the bottleneck in any of these tests. To achieve that, the appropriate network hardware must be used so as to not limit throughput.

3.1. Single Connection Bulk Throughput

Bulk data throughput on a single QUIC connection is probably the most common metric when first discussing the performance of a QUIC implementation. It uses only a single QUIC connection. It may be either an upload or download. It can be of any desired length.

For an upload test, the client need only open a single stream, encodes a zero server response size, sends the upload payload and then closes (FIN) the stream.

For a download test, the client again opens a single stream, encodes the server's response size (N bytes) and then closes the stream.

The total throughput rate is measured by the client, and is calculated by dividing the total bytes sent or received by difference in time from when the client created its initial stream to the time the client received the server's FIN.

3.2. Requests Per Second

Another very common performance metric is calculating the maximum requests per second that a QUIC server can handle. Unlike the bulk throughput test above, this test generally requires many parallel connections (possibly from multiple client machines) in order to saturate the server properly. There are several variables that tend to directly affect the results of this test:

- The number of parallel connections.
- The size of the client's request.
- The size of the server's response.

All of the above variables may be changed to measure the maximum RPS in the given scenario.

The test starts with the client connecting all parallel connections and waiting for them to be connected. It's recommended to wait an additional couple of seconds for things to settle down.

The client then starts sending "requests" on each connection. Specifically, the client should keep at least one request pending (preferrably at least two) on each connection at all times. When a request completes (receive server's FIN) the client should immediately queue another request.

The client continues to do this for a configured period of time. From my testing, ten seconds seems to be a good amount of time to reach the steady state.

Finally, the client measures the maximum requests per second rate as the total number of requests completed divided by the total execution time of the requests phase of the connection (not including the handshake and wait period).

3.3. Handshakes Per Second

Another metric that may reveal the connection setup efficiency is handshakes per second. It lets multiple clients (possibly from multiple machines) setup QUIC connections (then close them by CONNECTION_CLOSE) with a single server. Variables that may potentially affect the results are:

- The number of client machines.
- The number of connections a client can initialize in a second.
- The size of ClientHello (long list of supported ciphers, versions, etc.).

All the variables may be changed to measure the maximum handshakes per second in a given scenario.

The test starts with the multiple clients initializing connections and waiting for them to be connected with the single server on the other machine. It's recommended to wait an additional couple of seconds for connections to settle down.

The clients will initialize as many connections as possible to saturate the server. Once the client receive the handshake from the server, it terminates the connection by sending a CONNECTION_CLOSE to the server. The total handshakes per second are calculated by dividing the time period by the total number of connections that have successfully established during that time.

3.4. Throughput Fairness Index

Connection fairness is able to help us reveal how the throughput is allocated among each connection. A way of doing it is to establish multiple hundreds or thousands of concurrent connections and request the same data block from a single server. Variables that have potential impact on the results are:

- the size of the data being requested.
- the number of the concurrent connections.

The test starts with establishing several hundreds or thousands of concurrent connections and downloading the same data block from the server simultaneously.

The index of fairness is calculated using the complete time of each connection and the size of the data block in [Jain's manner] (https://www.cse.wustl.edu/~jain/atmf/ftp/af_fair.pdf).

Be noted that the relationship between fairness and whether the link is saturated is uncertain before any test. Thus it is recommended that both cases are covered in the test.

TODO: is it necessary that we also provide tests on latency fairness in the multi-connection case?

3.5. Maximum Number of Idle Connections

TODO

4. Things to Note

There are a few important things to note when doing performance testing.

4.1. What Data Should be Sent?

Since the goal here is to measure the efficiency of the QUIC implementation and not any application protocol, the performance application layer should be as light-weight as possible. To this end, the client and server application layer may use a single preallocated and initialized buffer that it queues to send when any payload needs to be sent out.

4.2. Ramp up Congestion Control or Not?

When running CPU limited, and not network limited, performance tests ideally we don't care too much about the congestion control state. That being said, assuming the tests run for enough time, generally congestion control should ramp up very quickly and not be a measureable factor in the measurements that result.

4.3. Disabling Encryption

A common topic when talking about QUIC performance is the effect that its encryption has. The draft-banks-quic-disable-encryption draft specifies a way for encryption to be mutually negotiated to be disabled so that an A:B test can be made to measure the "cost of encryption" in QUIC.

5. Security Considerations

Since the performance protocol allows for a client to trivially request the server to do a significant amount of work, it's generally advisable not to deploy a server running this protocol on the open internet.

One possible mitigation for unauthenticated clients generating an unacceptable amount of work on the server would be to use client certificates to authenticate the client first.

6. IANA Considerations

None

7. 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>.

[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>.

Author's Address

Nick Banks
Microsoft Corporation

Email: nibanks@microsoft.com