

EXHIBIT B

INFRINGEMENT CLAIM CHART FOR '058 PATENT

Jenam Tech, LLC (“Jenam”) provides evidence of infringement of claim 1 of U.S. Patent No. 11,064,058 (hereinafter “the ’058 patent”) by Google LLC (“Google”). In support thereof, Jenam provides the following claim chart.

“Accused Instrumentalities” as used herein refers to at least one or more websites or web addresses including, but not limited to www.google.com, stored and/or hosted on one or more servers owned or under the control of Google. For the sake of clarity, Jenam alleges that all of Google’s various websites, products, and platforms utilizing QUIC and/or gQUIC infringe the ’058 patent, including, but not limited to Google Edge Network, Google Cloud, Chrome Enterprise, G suite, Google Play, Chrome, Android (Android Enterprise, Android Messages (RCS)), Duo, Google Ads, Adwords, Google Analytics, Youtube, Google Mobile apps, Google shopping, and Google Maps. A list of Google’s “products” can be found at https://about.google/intl/en_us/products/. On information and belief, the Accused Instrumentalities referred to in the chart below are representative of Google’s use of QUIC (encompassing gQUIC and IETF QUIC) in Google’s other websites, products and platforms. This claim chart demonstrates Google’s infringement, and provide notice of such infringement, by comparing each element of the asserted claim to corresponding components, aspects, and/or features of the Accused Instrumentalities. These claim charts are not intended to constitute an expert report on infringement. This claim chart includes information provided by way of example, and not by way of limitation.

The analysis set forth below is based only upon information from publicly available resources regarding the Accused Instrumentalities. An analysis of Google’s (or other third parties’) technical documentation and/or software source code may assist in fully identify all infringing features and functionality. Accordingly, Jenam reserves the right to supplement this infringement analysis once such information is made available to Jenam. Furthermore, Jenam reserves the right to revise this infringement analysis, as appropriate, upon issuance of a court order construing any terms recited in any asserted claims. Unless otherwise noted, Jenam contends that Google directly infringes the ’058 patent in violation of 35 U.S.C. § 271(a) by selling, offering to sell, making, using, and/or importing the Accused Instrumentalities. The following exemplary analysis demonstrates that infringement. Unless otherwise noted, Jenam further contends that the evidence below supports a finding of indirect infringement under 35 U.S.C. § 271(b) in conjunction with other evidence of liability under one or more of those subsections. Google makes, uses, sells, imports, or offers for sale in the United States, or has made, used, sold, imported, or offered for sale in the past, without authority, or induces others to make, use, sell, import, or offer for sale in the United States, or has induced others to make, use, sell, import, or offer for sale in the past, without authority products, equipment, or services that infringe claim 1 of the ’058 patent, including without limitation, the Accused Instrumentalities. Google’s infringement includes its own use of the Accused Instrumentalities.

Unless otherwise noted, Jenam believes and contends that each element of claim 1 identified herein is literally met through Google’s provision of the Accused Instrumentalities. However, to the extent that Google attempts to allege that any asserted claim element is not literally met, Jenam believes and contends that such elements are met under the doctrine of equivalents. More specifically, in its investigation and analysis of the Accused Instrumentalities, Jenam did not identify any substantial differences between the elements of the patent claims and the corresponding features of the Accused Instrumentalities, as set forth herein. In each

instance, the identified feature of the Accused Instrumentalities performs at least substantially the same function in substantially the same way to achieve substantially the same result as the corresponding claim element.

To the extent the chart of the asserted claim relies on evidence about certain specifically-identified Accused Instrumentalities, Jenam asserts that, on information and belief, any similarly-functioning instrumentalities also infringe the charted claim. Jenam reserves the right to amend this infringement analysis based on other products made, used, sold, imported, or offered for sale by Google. Jenam also reserves the right to amend this infringement analysis by citing other claims of the '058 patent, not listed in the claim chart, that are infringed by the Accused Instrumentalities. Jenam further reserves the right to amend this infringement analysis by adding, subtracting, or otherwise modifying content in the "Accused Instrumentalities" column of each chart. This infringement analysis is preliminary and is not presented as Jenam's infringement contentions.

Claim 1 Elements	Applicability
An apparatus, comprising: a non-transitory memory storing instructions; and one or more processors in communication with the non-transitory memory, wherein the one or more processors execute the instructions to: identify, at the apparatus, first information on which at least a first duration for detecting a first type of time period is based, the first type of time period being started in response to detecting at least one of a received packet or a sent packet;	Google uses an apparatus (e.g., a platform or device, including a QUIC-compliant server or client, etc.), comprising: a non-transitory memory storing instructions (e.g., server software, etc.), and one or more processors in communication with the non-transitory memory, wherein the one or more processors execute the instructions to: identify, at the apparatus, first information (e.g., first value, etc.) on which at least a first duration (e.g., duration associated with idle timeout in connection with 0-RTT packets, etc.) for detecting a first type of time period (e.g., idle-type timeout, etc.) is based, the first type of time period being started in response to detecting at least one of a received packet or a sent packet; See excerpt(s) below, for example (emphasis added, if any): Google admits that their "Google Cloud Platform" uses QUIC: "Google Cloud Platform Blog... we're happy to be the first major public cloud to offer QUIC support for our HTTPS load balancers." https://cloudplatform.googleblog.com/2018/06/Introducing-QUIC-support-for-HTTPS-load-balancing.html The "Google Cloud Platform" that, as established above, uses QUIC, is also used for ALL of Google's services:

“Google Cloud Platform (GCP), offered by Google (company), is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search, Gmail, file storage, and YouTube”

https://en.wikipedia.org/wiki/Google_Cloud_Platform

At least a portion of the citations herein are made to the QUIC standard found at: <https://tools.ietf.org/html/draft-ietf-quic-transport-22>, and/or <https://tools.ietf.org/html/draft-ietf-quic-transport-27>. Based upon information and belief, the pertinent portions of such version of the QUIC standard may also be found in one or more other versions. For completeness, all of the versions below (and future iterations thereof) are hereby incorporated by reference:

<https://tools.ietf.org/html/draft-ietf-quic-transport-00>
<https://tools.ietf.org/html/draft-ietf-quic-transport-01>
<https://tools.ietf.org/html/draft-ietf-quic-transport-02>
<https://tools.ietf.org/html/draft-ietf-quic-transport-03>
<https://tools.ietf.org/html/draft-ietf-quic-transport-04>
<https://tools.ietf.org/html/draft-ietf-quic-transport-05>
<https://tools.ietf.org/html/draft-ietf-quic-transport-06>
<https://tools.ietf.org/html/draft-ietf-quic-transport-07>
<https://tools.ietf.org/html/draft-ietf-quic-transport-08>
<https://tools.ietf.org/html/draft-ietf-quic-transport-09>
<https://tools.ietf.org/html/draft-ietf-quic-transport-10>
<https://tools.ietf.org/html/draft-ietf-quic-transport-11>
<https://tools.ietf.org/html/draft-ietf-quic-transport-12>
<https://tools.ietf.org/html/draft-ietf-quic-transport-13>
<https://tools.ietf.org/html/draft-ietf-quic-transport-14>
<https://tools.ietf.org/html/draft-ietf-quic-transport-15>
<https://tools.ietf.org/html/draft-ietf-quic-transport-16>
<https://tools.ietf.org/html/draft-ietf-quic-transport-17>
<https://tools.ietf.org/html/draft-ietf-quic-transport-18>
<https://tools.ietf.org/html/draft-ietf-quic-transport-19>

<https://tools.ietf.org/html/draft-ietf-quic-transport-20>
<https://tools.ietf.org/html/draft-ietf-quic-transport-21>
<https://tools.ietf.org/html/draft-ietf-quic-transport-22>
<https://tools.ietf.org/html/draft-ietf-quic-transport-23>
<https://tools.ietf.org/html/draft-ietf-quic-transport-24>
<https://tools.ietf.org/html/draft-ietf-quic-transport-25>
<https://tools.ietf.org/html/draft-ietf-quic-transport-26>
<https://tools.ietf.org/html/draft-ietf-quic-transport-27>

Though these contentions may refer to a specific version of the QUIC standard, the presented evidence is not exclusive to this specific version and is compliant with and/or supported by current versions of the QUIC standard, including QUIC_VERSION_IETF_RFC_V1 (<https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-34>), QUIC_VERSION_IETF_DRAFT_25 (<https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-25>), QUIC_VERSION_IETF_DRAFT_27 (<https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-27>), QUIC_VERSION_IETF_DRAFT_28 (<https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-28>), QUIC_VERSION_IETF_DRAFT_29 (<https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-29>), and/or QUIC_VERSION_50 which relies in part on IETF_QUIC_INVARIANTS_06 (<https://tools.ietf.org/html/draft-ietf-quic-invariants-06>) and/or IETF_QUIC_TRANSPORT_17 (<https://tools.ietf.org/html/draft-ietf-quic-transport-17>), each of which, upon information and belief, has been utilized by Google LLC through its accused instrumentalities.

Note: The following is from current versions of Chromium.

```

115 QUIC_VERSION_46 = 46, // Use IETF draft-17 header format with demultiplexing
116 // bit.
117 // Version 47 added variable-length QUIC server connection IDs.
118 // Version 48 added CRYPTO frames for the handshake.
119 // Version 49 added client connection IDs, long header lengths, and the IETF
120 // header format from draft-ietf-quic-invariants-06
121 QUIC_VERSION_50 = 50, // Header protection and initial obfuscators.
122 // Number 51 was 1051 which used draft-29 features but with GoogleQUIC frames.
123 // Number 70 used to represent draft-ietf-quic-transport-25.
124 // Number 71 used to represent draft-ietf-quic-transport-27.
125 // Number 72 used to represent draft-ietf-quic-transport-28.
126 QUIC_VERSION_IETF_DRAFT_29 = 73, // draft-ietf-quic-transport-29.
127 QUIC_VERSION_IETF_RFC_V1 = 80, // RFC 9000.
128 QUIC_VERSION_IETF_2_DRAFT_01 = 81, // draft-ietf-quic-v2-01.
129 // Version 99 was a dumping ground for IETF QUIC changes which were not yet
130 // ready for production between 2018-02 and 2020-02.

```

https://source.chromium.org/chromium/chromium/src/+main:net/third_party/quiche/src/quiche/quic/core/quic_versions.h;l=127?q=QUIC_VERSION_IETF_RFC_V1&sq=&ss=chromium (emphasis added)


Chromium Code Search

QUIC_VERSION_IETF_RFC

Results 1 - 4 of 4


[chromium > chromium/src > main] net/quic/quic_http_stream.cc

Preview

```

107:     return HttpResponseInfo::CONNECTION_INFO_QUIC_DRAFT_29;
108: case quic::QUIC_VERSION_IETF_RFC_V1:
109:     DCHECK(quic_version.UsesTls());

```


[chromium > chromium/src > main] net/third_party/quiche/src/quiche

Preview

```

126: QUIC_VERSION_IETF_DRAFT_29 = 73,    // draft-ietf-quic-transport-29.
127: QUIC_VERSION_IETF_RFC_V1 = 80,    // RFC 9000.
128: QUIC_VERSION_IETF_2_DRAFT_01 = 81, // draft-ietf-quic-v2-01.

176: QUIC_VERSION_IETF_2_DRAFT_01,
177: QUIC_VERSION_IETF_RFC_V1,
178: QUIC_VERSION_IETF_DRAFT_29,

200: transport_version != QUIC_VERSION_IETF_DRAFT_29 &&
201: transport_version != QUIC_VERSION_IETF_RFC_V1 &&
202: transport_version != QUIC_VERSION_IETF_2_DRAFT_01;

256: static constexpr ParsedQuicVersion RFCv1() {
257:     return ParsedQuicVersion(PROTOCOL_TLS1_3, QUIC_VERSION_IETF_RFC_V1);
258: }

```


[chromium > chromium/src > main] net/third_party/quiche/src/quiche

Preview

```

520: RETURN_STRING_LITERAL(QUIC_VERSION_IETF_DRAFT_29);
521: RETURN_STRING_LITERAL(QUIC_VERSION_IETF_RFC_V1);
522: RETURN_STRING_LITERAL(QUIC_VERSION_IETF_2_DRAFT_01);

```


[chromium > chromium/src > main] net/third_party/quiche/src/quiche

Preview

```

429: EXPECT_EQ(QUIC_VERSION_IETF_DRAFT_29, 73);
430: EXPECT_EQ(QUIC_VERSION_IETF_RFC_V1, 80);
431: EXPECT_EQ(QUIC_VERSION_IETF_2_DRAFT_01, 81);

```

https://source.chromium.org/search?q=QUIC_VERSION_IETF_RFC_V1 (emphasis added)

See also:

- <https://blog.chromium.org/2020/10/chrome-is-deploying-http3-and-ietf-quic.html#:~:text=Chrome%20will%20actively%20support%20both,to%20update%20to%20IETF%20QUIC.>
- <https://groups.google.com/a/chromium.org/g/proto-quic/c/YtdH4MG70Ww>
- <https://chromium.googlesource.com/chromium/src/+/cbfe636a4fe6502dc8a5e869d82bcf6e2e94aa00>

Below is a web page of Google (<https://www.google.com/>).



Name	Status	Protocol	Type	Initiator	Size	Time	Waterfall
google.com	301	http/1.1	text/html	Other	(disk cac...	93 ...	
www.google.com	307	http/1.1	Other	google.com/	0 B	9 ms	
www.google.com	200	http/2+quic/46	document	www.google.com/	61.9 KB	175...	
stayhome-save-lives-april-20...	200	http/2+quic/46	gif	(index)	(disk cac...	24 ...	
KfOwCngUa3fV1MmWUfBbckw...	200	http/2+quic/46	font	(index)106	(disk cac...	12 ...	
KfOwCngUa3fV1MmWUfBbckw...	200	http/2+quic/46	font	(index)106	(disk cac...	12 ...	
data:image/png;base64...	200	data	png	(index)	(memory...	0 ms	
11_1967catla.png	200	http/2+quic/46	png	(index)	(disk cac...	4 ms	
data:image/svg+xml...	200	data	svg+xml	(index)	(memory...	0 ms	
data:image/svg+xml...	200	data	svg+xml	(index)	(memory...	0 ms	
data:image/svg+xml...	200	data	svg+xml	(index)	(memory...	0 ms	
data:image/svg+xml...	200	data	svg+xml	(index)	(memory...	0 ms	
data:image/png;base64...	200	data	png	(index)	(memory...	0 ms	
googlemic_color_24dp.png	200	http/2+quic/46	png	(index)	(disk cac...	10 ...	
desktop_searchbox_sprites302_h...	200	http/2+quic/46	webp	(index)	(disk cac...	11 ...	
gen_2047s=webp&dt=af&utyp...	204	http/2+quic/46	text/html	(index)361	42 B	33 ...	
nav_logo299.webp	200	http/2+quic/46	webp	(index)357	(disk cac...	9 ms	
r=AC190uH0l_R0QhVXUQ9B...	200	http/2+quic/46	script	(index)109	(disk cac...	11 ...	
gen_2047atyp=cs&len=N1G0uU...	204	http/2+quic/46	text/html	rsuAC190uH0l_R0QhVXUQ9B...	17 B	46 ...	

As set forth below, since the `idle_timeout` value sets the duration of idleness, after which the connection is shutdown, a timeout attribute of the connection is necessarily detected based on the value received in the `idle_timeout` field of the connection negotiation packet.

“`idle_timeout` (0x0001): The idle timeout is a value in milliseconds that is encoded as an integer; see (Section 10.2). If this parameter is absent or zero then the idle timeout is disabled.”

	<i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “10. Connection Termination An established QUIC connection can be terminated in one of three ways: • idle timeout (Section 10.2) • immediate close (Section 10.3) • stateless reset (Section 10.4) An endpoint MAY discard connection state if it does not have a validated path on which it can send packets (see Section 8.2).” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “10.2. Idle Timeout If the idle timeout is enabled, a connection is silently closed and the state is discarded when it remains idle for longer than both the advertised idle timeout (see Section 18.1) and three times the current Probe Timeout (PTO). Each endpoint advertises its own idle timeout to its peer. An endpoint restarts any timer it maintains when a packet from its peer is received and processed successfully. The timer is also restarted when sending a packet containing frames other than ACK or PADDING (an ACK-eliciting packet; see [QUIC-RECOVERY]), but only if no other ACK-eliciting packets have been sent since last receiving a packet. Restarting when sending packets ensures that connections do not prematurely time out when initiating new activity. The value for an idle timeout can be asymmetric. The value advertised by an endpoint is only used to determine whether the connection is live at that endpoint. An endpoint that sends packets near the end of the idle timeout period of a peer risks having those packets discarded if its peer enters the draining state before the packets arrive. If a peer could timeout within a Probe Timeout (PTO; see Section 6.3 of [QUIC-RECOVERY]), it is advisable to test for
--	---

	liveness before sending any data that cannot be retried safely. Note that it is likely that only applications or application protocols will know what information can be retried.” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “7.3.1. Values of Transport Parameters for 0-RTT <u>Both endpoints store the value of the server transport parameters from a connection and apply them to any 0-RTT packets that are sent in subsequent connections to that peer, except for transport parameters that are explicitly excluded. Remembered transport parameters apply to the new connection until the handshake completes and the client starts sending 1-RTT packets.</u> Once the handshake completes, the client uses the transport parameters established in the handshake. The definition of new transport parameters (Section 7.3.2) MUST specify whether they MUST, MAY, or MUST NOT be stored for 0-RTT. A client need not store a transport parameter it cannot process. A client MUST NOT use remembered values for the following parameters: <code>original_connection_id</code>, <code>preferred_address</code>, <code>stateless_reset_token</code>, <code>ack_delay_exponent</code> and <code>active_connection_id_limit</code>. The client MUST use the server's new values in the handshake instead, and absent new values from the server, the default value. A client that attempts to send 0-RTT data MUST remember all other transport parameters used by the server. The server can remember these transport parameters, or store an integrity-protected copy of the values in the ticket and recover the information when accepting 0-RTT data. A server uses the transport parameters in determining whether to accept 0-RTT data. If 0-RTT data is accepted by the server, the server MUST NOT reduce any limits or alter any values that might be violated by the client with its 0-RTT data. In particular, a server that accepts 0-RTT data MUST NOT set values for the following parameters (Section 18.1) that are smaller than the remembered value of the parameters.
--	--

	• initial_max_data • initial_max_stream_data_bidi_local • initial_max_stream_data_bidi_remote • initial_max_stream_data_uni • initial_max_streams_bidi • initial_max_streams_uni Omitting or setting a zero value for certain transport parameters can result in 0-RTT data being enabled, but not usable. The applicable subset of transport parameters that permit sending of application data SHOULD be set to non-zero values for 0-RTT. This includes initial_max_data and either initial_max_streams_bidi and initial_max_stream_data_bidi_remote, or initial_max_streams_uni and initial_max_stream_data_uni. A server MUST either reject 0-RTT data or abort a handshake if the implied values for transport parameters cannot be supported. When sending frames in 0-RTT packets, a client MUST only use remembered transport parameters; importantly, it MUST NOT use updated values that it learns from the server's updated transport parameters or from frames received in 1-RTT packets. Updated values of transport parameters from the handshake apply only to 1-RTT packets. For instance, flow control limits from remembered transport parameters apply to all 0-RTT packets even if those values are increased by the handshake or by frames sent in 1-RTT packets. A server MAY treat use of updated transport parameters in 0-RTT as a connection error of type <code>PROTOCOL_VIOLATION</code>.” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “3.3. Session resumption versus Keep-alive Because QUIC is encapsulated in UDP, applications using QUIC must deal with short idle timeouts. Deployed stateful middleboxes will generally establish state for UDP flows on the
--	--

	first packet state, and keep state for much shorter idle periods than for TCP. According to a 2010 study ([Hatonen10]), UDP applications can assume that any NAT binding or other state entry will be expired after just thirty seconds of inactivity. A QUIC application has three strategies to deal with this issue: • Ignore it, if the application-layer protocol consists only of interactions with no or very short idle periods. • Ensure there are no long idle periods. • <u>Resume the session after a long idle period, using 0-RTT resumption when appropriate.</u> ... Alternatively, the client (but not the server) can use session resumption instead of sending keepalive traffic. In this case, a client that wants to send data to a server over a connection idle longer than the server's idle timeout (available from the <code>idle_timeout</code> transport parameter) can simply reconnect. <u>When possible, this reconnection can use 0-RTT session resumption, reducing the latency involved with restarting the connection.</u> This of course only applies in cases in which 0-RTT data is safe, when the client is the restarting peer, and when the data to be sent is idempotent.” https://www.ietf.org/id/draft-ietf-quic-applicability-07.txt
allocate a resource for a first connection;	Google uses an apparatus (e.g., a platform or device, including a QUIC-compliant server or client, etc.), configured to allocate a resource (e.g., service facilitating QUIC connection, etc.) for a first connection (e.g., QUIC connection, etc.); See excerpt(s) above and below, for example (emphasis added, if any): “Google Cloud Platform (GCP), offered by Google (company), is a <u>suite of cloud computing services</u> that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search, Gmail, file storage, and YouTube” https://en.wikipedia.org/wiki/Google_Cloud_Platform

	1. Introduction QUIC is a multiplexed and secure transport protocol that runs on top of UDP. QUIC aims to provide a flexible set of features that allow it to be a general-purpose transport for multiple applications. QUIC implements techniques learned from experience with TCP, SCTP and <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-07
generate a first packet including a first parameter field identifying first metadata for use in determining a second duration for detecting the first type of time period, the first packet for use in a protocol that is different from a transmission control protocol (TCP);	Google uses an apparatus (e.g., a platform or device, including a QUIC-compliant server or client, etc.), configured to generate a first packet (e.g., first QUIC negotiation packet, etc.) including a first parameter field (e.g., first idle_timeout parameter field, etc.) identifying first metadata (e.g., first idle timeout value, etc.) for use in determining a second duration (e.g., duration associated with idle timeout in connection with 1-RTT packets, etc.) for detecting the first type of time period (e.g., idle-type timeout, etc.), the first packet for use in a protocol that is different from a transmission control protocol (TCP) (e.g., QUIC, etc.); See excerpt(s) above and below, for example (emphasis added, if any): As set forth below, a QUIC negotiation packet includes transport parameters that include an idle_timeout parameter that is detected by a recipient of such packet. “7.3. Transport Parameters During connection establishment, <u>both endpoints make authenticated declarations of their transport parameters. These declarations are made unilaterally by each endpoint.</u> Endpoints are required to comply with the restrictions implied by these parameters; the description of each parameter includes rules for its handling.” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22

Value	Parameter Name	Specification
0x0000	original_connection_id	Section 18.1
0x0001	idle_timeout	Section 18.1
0x0002	stateless_reset_token	Section 18.1
0x0003	max_packet_size	Section 18.1
0x0004	initial_max_data	Section 18.1
0x0005	initial_max_stream_data_bidi_local	Section 18.1
0x0006	initial_max_stream_data_bidi_remote	Section 18.1
0x0007	initial_max_stream_data_uni	Section 18.1
0x0008	initial_max_streams_bidi	Section 18.1
0x0009	initial_max_streams_uni	Section 18.1
0x000a	ack_delay_exponent	Section 18.1
0x000b	max_ack_delay	Section 18.1
0x000c	disable_migration	Section 18.1
0x000d	preferred_address	Section 18.1
0x000e	active_connection_id_limit	Section 18.1

Table 6: Initial QUIC Transport Parameters Entries

QUIC: A UDP-Based Multiplexed and Secure Transport <https://tools.ietf.org/html/draft-ietf-quic-transport-22>

“18. Transport Parameter Encoding

The format of the transport parameters is the TransportParameters struct from Figure 15. This is described using the presentation language from Section 3 of [TLS13].

```
enum {
    original_connection_id(0),
    idle_timeout(1),
    stateless_reset_token(2),
    max_packet_size(3),
    initial_max_data(4),
    initial_max_stream_data_bidi_local(5),
    initial_max_stream_data_bidi_remote(6),
    initial_max_stream_data_uni(7),
    initial_max_streams_bidi(8),
    initial_max_streams_uni(9),
    ack_delay_exponent(10),
    max_ack_delay(11),
    disable_migration(12),
    preferred_address(13),
    active_connection_id_limit(14),
    (65535)
} TransportParameterId;
```

QUIC: A UDP-Based Multiplexed and Secure Transport <https://tools.ietf.org/html/draft-ietf-quic-transport-22>

As set forth below, since the `idle_timeout` value sets the duration of idleness, after which the connection is shutdown, a timeout attribute of the connection is necessarily modified based on the value received in the `idle_timeout` field of the connection negotiation packet.

“`idle_timeout` (0x0001): The idle timeout is a value in milliseconds that is encoded as an integer; see (Section 10.2). If this parameter is absent or zero then the idle timeout is disabled.”

QUIC: A UDP-Based Multiplexed and Secure Transport <https://tools.ietf.org/html/draft-ietf-quic-transport-22>

	“QUIC provides reliable, ordered delivery of the cryptographic handshake data. QUIC packet protection is used to encrypt as much of the handshake protocol as possible. The cryptographic handshake MUST provide the following properties: • authenticated key exchange, where ○ a server is always authenticated, ○ a client is optionally authenticated, ○ every connection produces distinct and unrelated keys, ○ <u>keying material is usable for packet protection for both 0-RTT and 1-RTT packets, and</u> ○ <u>1-RTT keys have forward secrecy</u> • authenticated values for the transport parameters of the peer (see Section 7.3) • authenticated negotiation of an application protocol (TLS uses ALPN [RFC7301] for this purpose)” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22
set up the first connection, by sending, from the apparatus to a node, the first packet to provide the first metadata to the node, for use by the node in determining the second duration for detecting the first type of time period;	Google uses an apparatus (e.g., a platform or device, including a QUIC-compliant server or client, etc.), configured to set up the first connection (e.g., QUIC connection, etc.), by sending, from the apparatus to a node (e.g., one of a QUIC-compliant server or client, etc.), the first packet (e.g., first QUIC negotiation packet, etc.) to provide the first metadata (e.g., first idle timeout value, etc.) to the node, for use by the node in determining the second duration (e.g., duration associated with idle timeout in connection with 1-RTT packets, etc.) for detecting the first type of time period (e.g., idle-type timeout, etc.); See excerpt(s) above and below, for example (emphasis added, if any): As set forth below, a QUIC negotiation packet includes transport parameters that include an idle_timeout parameter that is detected by a recipient of such packet. “7.3. Transport Parameters

During connection establishment, both endpoints make authenticated declarations of their transport parameters. These declarations are made unilaterally by each endpoint. Endpoints are required to comply with the restrictions implied by these parameters; the description of each parameter includes rules for its handling.”

QUIC: A UDP-Based Multiplexed and Secure Transport <https://tools.ietf.org/html/draft-ietf-quic-transport-22>

Value	Parameter Name	Specification
0x0000	original_connection_id	Section 18.1
0x0001	idle_timeout	Section 18.1
0x0002	stateless_reset_token	Section 18.1
0x0003	max_packet_size	Section 18.1
0x0004	initial_max_data	Section 18.1
0x0005	initial_max_stream_data_bidi_local	Section 18.1
0x0006	initial_max_stream_data_bidi_remote	Section 18.1
0x0007	initial_max_stream_data_uni	Section 18.1
0x0008	initial_max_streams_bidi	Section 18.1
0x0009	initial_max_streams_uni	Section 18.1
0x000a	ack_delay_exponent	Section 18.1
0x000b	max_ack_delay	Section 18.1
0x000c	disable_migration	Section 18.1
0x000d	preferred_address	Section 18.1
0x000e	active_connection_id_limit	Section 18.1

Table 6: Initial QUIC Transport Parameters Entries

QUIC: A UDP-Based Multiplexed and Secure Transport <https://tools.ietf.org/html/draft-ietf-quic-transport-22>

	“QUIC provides reliable, ordered delivery of the cryptographic handshake data. QUIC packet protection is used to encrypt as much of the handshake protocol as possible. The cryptographic handshake MUST provide the following properties: • authenticated key exchange, where ○ a server is always authenticated, ○ a client is optionally authenticated, ○ every connection produces distinct and unrelated keys, ○ <u>keying material is usable for packet protection for both 0-RTT and 1-RTT packets</u>, and ○ <u>1-RTT keys have forward secrecy</u> • authenticated values for the transport parameters of the peer (see Section 7.3) • authenticated negotiation of an application protocol (TLS uses ALPN [RFC7301] for this purpose)” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22
in response to detecting, based on the first duration and an absence of one or more packets and by the apparatus during at least a portion of the first connection including at least a portion of the first connection set up, a first time period of the first type of time period, at least partially close the first connection, by releasing, by the apparatus, the resource allocated for the first connection;	Google uses an apparatus (e.g., a platform or device, including a QUIC-compliant server or client, etc.), configured to, in response to detecting, based on the first duration (e.g., duration associated with idle timeout in connection with 0-RTT packets, etc.) and an absence of one or more packets and by the apparatus during at least a portion of the first connection (e.g., QUIC connection, etc.) including at least a portion of the first connection set up, a first time period (e.g., idle timeout in connection with 0-RTT packets, etc.) of the first type of time period (e.g., idle-type timeout, etc.), at least partially close (e.g., terminate, etc.) the first connection, by releasing, by the apparatus, the resource (e.g., service facilitating QUIC connection, etc.) allocated for the first connection; See excerpt(s) above and below, for example (emphasis added, if any):

	As set forth below, since the <u>idle_timeout</u> value sets the duration of idleness, after which the <u>connection is shutdown</u>, a timeout attribute of the connection is necessarily detected based on the value received in the <u>idle_timeout</u> field of the connection negotiation packet. “idle_timeout (0x0001): The idle timeout is a value in milliseconds that is encoded as an integer; see (Section 10.2). If this parameter is absent or zero then the idle timeout is disabled.” QUIC: A UDP-Based Multiplexed and Secure Transport https://tools.ietf.org/html/draft-ietf-quic-transport-22 “10. <u>Connection Termination</u> An established QUIC <u>connection can be terminated</u> in one of three ways: • <u>idle timeout</u> (Section 10.2) • immediate close (Section 10.3) • stateless reset (Section 10.4) An endpoint MAY discard connection state if it does not have a validated path on which it can send packets (see Section 8.2).” QUIC: A UDP-Based Multiplexed and Secure Transport https://tools.ietf.org/html/draft-ietf-quic-transport-22 “10.2. Idle Timeout <u>If the idle timeout is enabled, a connection is silently closed and the state is discarded when it remains idle for longer than both the advertised idle timeout</u> (see Section 18.1) and three times the current Probe Timeout (PTO).” QUIC: A UDP-Based Multiplexed and Secure Transport https://tools.ietf.org/html/draft-ietf-quic-transport-22 “7.3.1. Values of Transport Parameters for 0-RTT <u>Both endpoints store the value of the server transport parameters from a connection and apply them to any 0-RTT packets that are sent in subsequent connections to that peer, except for</u>
--	---

	<u>transport parameters that are explicitly excluded. Remembered transport parameters apply to the new connection until the handshake completes and the client starts sending 1-RTT packets.</u> Once the handshake completes, the client uses the transport parameters established in the handshake. The definition of new transport parameters (Section 7.3.2) MUST specify whether they MUST, MAY, or MUST NOT be stored for 0-RTT. A client need not store a transport parameter it cannot process. A client MUST NOT use remembered values for the following parameters: original_connection_id, preferred_address, stateless_reset_token, ack_delay_exponent and active_connection_id_limit. The client MUST use the server's new values in the handshake instead, and absent new values from the server, the default value. A client that attempts to send 0-RTT data MUST remember all other transport parameters used by the server. The server can remember these transport parameters, or store an integrity-protected copy of the values in the ticket and recover the information when accepting 0-RTT data. A server uses the transport parameters in determining whether to accept 0-RTT data. If 0-RTT data is accepted by the server, the server MUST NOT reduce any limits or alter any values that might be violated by the client with its 0-RTT data. In particular, a server that accepts 0-RTT data MUST NOT set values for the following parameters (Section 18.1) that are smaller than the remembered value of the parameters. • initial_max_data • initial_max_stream_data_bidi_local • initial_max_stream_data_bidi_remote • initial_max_stream_data_uni • initial_max_streams_bidi • initial_max_streams_uni
--	---

	Omitting or setting a zero value for certain transport parameters can result in 0-RTT data being enabled, but not usable. The applicable subset of transport parameters that permit sending of application data SHOULD be set to non-zero values for 0-RTT. This includes initial_max_data and either initial_max_streams_bidi and initial_max_stream_data_bidi_remote, or initial_max_streams_uni and initial_max_stream_data_uni. A server MUST either reject 0-RTT data or abort a handshake if the implied values for transport parameters cannot be supported. When sending frames in 0-RTT packets, a client MUST only use remembered transport parameters; importantly, it MUST NOT use updated values that it learns from the server's updated transport parameters or from frames received in 1-RTT packets. Updated values of transport parameters from the handshake apply only to 1-RTT packets. For instance, flow control limits from remembered transport parameters apply to all 0-RTT packets even if those values are increased by the handshake or by frames sent in 1-RTT packets. A server MAY treat use of updated transport parameters in 0-RTT as a connection error of type PROTOCOL_VIOLATION.” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22
in response to detecting, based on the second duration and an absence of at least one packet and by the apparatus after the first connection set up and a use of the first duration is changed to a use of the second duration, a second time period of the first type of time period, at least partially close the first connection, by releasing, by the apparatus, the resource allocated for the first connection, such that, during the	Google uses an apparatus (e.g., a platform or device, including a QUIC-compliant server or client, etc.), configured to, in response to detecting, based on the second duration (e.g., duration associated with idle timeout in connection with 1-RTT packets, etc.) and an absence of at least one packet and by the apparatus after the first connection set up and a use of the first duration (e.g., duration associated with idle timeout in connection with 0-RTT packets, etc.) is changed to a use of the second duration, a second time period (e.g., idle timeout in connection with 1-RTT packets, etc.) of the first type of time period (e.g., idle-type timeout, etc.), at least partially close (e.g., terminate, etc.) the first connection (e.g., QUIC connection, etc.), by releasing, by the apparatus, the resource (e.g., service facilitating QUIC connection, etc.) allocated for the first connection, such that, during the second time period of the first type of time period, a packet including no application data is capable of being sent or received without a second time period timer being reset; and

second time period of the first type of time period, a packet including no application data is capable of being sent or received without a second time period timer being reset; and	See excerpt(s) above and below, for example (emphasis added, if any): As set forth below, since the <code>idle_timeout</code> value sets the duration of idleness, after which the connection is shutdown, a timeout attribute of the connection is necessarily detected based on the value received in the <code>idle_timeout</code> field of the connection negotiation <u>packet</u>. “<code>idle_timeout</code> (0x0001): The idle timeout is a value in milliseconds that is encoded as an integer; see (Section 10.2). If this parameter is absent or zero then the idle timeout is disabled.” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “10. <u>Connection Termination</u> An established QUIC connection can be terminated in one of three ways: • idle timeout (Section 10.2) • immediate close (Section 10.3) • stateless reset (Section 10.4) An endpoint MAY discard connection state if it does not have a validated path on which it can send packets (see Section 8.2).” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “10.2. Idle Timeout If the idle timeout is enabled, a connection is silently closed and the state is discarded when it remains idle for longer than both the advertised idle timeout (see Section 18.1) and three times the current Probe Timeout (PTO). Each endpoint advertises its own idle timeout to its peer. An endpoint restarts any timer it maintains when a packet from its peer is received and processed successfully. The timer is also restarted when sending a packet containing frames other than ACK or PADDING (an
---	--

	ACK-eliciting packet; see [QUIC-RECOVERY]), but only if no other ACK-eliciting packets have been sent since last receiving a packet. Restarting when sending packets ensures that connections do not prematurely time out when initiating new activity. <u>The value for an idle timeout can be asymmetric.</u> The value advertised by an endpoint is only used to determine whether the connection is live at that endpoint. An endpoint that sends packets near the end of the idle timeout period of a peer risks having those packets discarded if its peer enters the draining state before the packets arrive. If a peer could timeout within a Probe Timeout (PTO; see Section 6.3 of [QUIC-RECOVERY]), it is advisable to test for liveness before sending any data that cannot be retried safely. Note that it is likely that only applications or application protocols will know what information can be retried.” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “7.3.1. Values of Transport Parameters for 0-RTT <u>Both endpoints store the value of the server transport parameters from a connection and apply them to any 0-RTT packets that are sent in subsequent connections to that peer, except for transport parameters that are explicitly excluded. Remembered transport parameters apply to the new connection until the handshake completes and the client starts sending 1-RTT packets.</u> Once the handshake completes, the client uses the transport parameters established in the handshake. The definition of new transport parameters (Section 7.3.2) MUST specify whether they MUST, MAY, or MUST NOT be stored for 0-RTT. A client need not store a transport parameter it cannot process. A client MUST NOT use remembered values for the following parameters: original_connection_id, preferred_address, stateless_reset_token, ack_delay_exponent and active_connection_id_limit. The client MUST use the server's new values in the handshake instead, and absent new values from the server, the default value.
--	--

	A client that attempts to send 0-RTT data MUST remember all other transport parameters used by the server. The server can remember these transport parameters or store an integrity-protected copy of the values in the ticket and recover the information when accepting 0-RTT data. A server uses the transport parameters in determining whether to accept 0-RTT data. If 0-RTT data is accepted by the server, the server MUST NOT reduce any limits or alter any values that might be violated by the client with its 0-RTT data. In particular, a server that accepts 0-RTT data MUST NOT set values for the following parameters (Section 18.1) that are smaller than the remembered value of the parameters. • initial_max_data • initial_max_stream_data_bidi_local • initial_max_stream_data_bidi_remote • initial_max_stream_data_uni • initial_max_streams_bidi • initial_max_streams_uni Omitting or setting a zero value for certain transport parameters can result in 0-RTT data being enabled, but not usable. The applicable subset of transport parameters that permit sending of application data SHOULD be set to non-zero values for 0-RTT. This includes initial_max_data and either initial_max_streams_bidi and initial_max_stream_data_bidi_remote, or initial_max_streams_uni and initial_max_stream_data_uni. A server MUST either reject 0-RTT data or abort a handshake if the implied values for transport parameters cannot be supported. When sending frames in 0-RTT packets, a client MUST only use remembered transport parameters; importantly, it MUST NOT use updated values that it learns from the server's updated transport parameters or from frames received in 1-RTT packets. Updated values of transport parameters from the handshake apply only to 1-RTT packets. For instance, flow control limits from remembered transport parameters apply to all 0-RTT packets even if those
--	--

	values are increased by the handshake or by frames sent in 1-RTT packets. A server MAY treat use of updated transport parameters in 0-RTT as a connection error of type <code>PROTOCOL_VIOLATION</code>.” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22
in response to detecting, based on a third duration, a third time period of a second type of time period that is detected based on one or more criteria that is different from that on which detection of the first time period and the second time period of the first type of time period are based, perform at least one action, where the third duration is determined based on a first algorithm that is different from a second algorithm on which a determination of the second duration is based.	Google uses an apparatus (e.g., a platform or device, including a QUIC-compliant server or client, etc.), configured to, in response to detecting, based on a third duration (e.g., duration associated with probe timeout, etc.), a third time period (e.g., probe timeout, etc.) of a second type of time period that is detected based on one or more criteria that is different from that on which detection of the first time period (e.g., idle timeout in connection with 0-RTT packets, etc.) and the second time period (e.g., idle timeout in connection with 1-RTT packets, etc.) of the first type of time period (e.g., idle-type timeout, etc.) are based, perform at least one action, where the third duration is determined based on a first algorithm (e.g., used to determine the probe timeout, etc.) that is different from a second algorithm (e.g., used to negotiate an idle timeout period between nodes, etc.) on which a determination of the second duration (e.g., duration associated with idle timeout in connection with 1-RTT packets, etc.) is based. See excerpt(s) above and below, for example (emphasis added, if any): “10.2. Idle Timeout <u>If the idle timeout is enabled, a connection is silently closed and the state is discarded when it remains idle for longer than both the advertised idle timeout (see Section 18.1) and three times the current Probe Timeout (PTO).</u> Each endpoint advertises its own idle timeout to its peer. An endpoint restarts any timer it maintains when a packet from its peer is received and processed successfully. The timer is also restarted when sending a packet containing frames other than ACK or PADDING (an ACK-eliciting packet; see [QUIC-RECOVERY]), but only if no other ACK-eliciting packets have been sent since last receiving a packet. Restarting when sending packets ensures that connections do not prematurely time out when initiating new activity.

	The value for an idle timeout can be asymmetric. <u>The value advertised by an endpoint is only used to determine whether the connection is live at that endpoint.</u> An endpoint that sends packets near the end of the idle timeout period of a peer risks having those packets discarded if its peer enters the draining state before the packets arrive. If a peer could timeout within a Probe Timeout (PTO; see Section 6.3 of [QUIC-RECOVERY]), it is advisable to test for liveness before sending any data that cannot be retried safely. Note that it is likely that only applications or application protocols will know what information can be retried.” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “18.1 Transport Parameter Definitions ... • max_ack_delay (0x000b): The maximum ACK delay is an integer value indicating the maximum amount of time in milliseconds by which the endpoint will delay sending acknowledgments. This value SHOULD include the receiver's expected delays in alarms firing. For example, if a receiver sets a timer for 5ms and alarms commonly fire up to 1ms late, then it should send a max_ack_delay of 6ms. If this value is absent, a default of 25 milliseconds is assumed. Values of 2^{14} or greater are invalid” <i>QUIC: A UDP-Based Multiplexed and Secure Transport</i> https://tools.ietf.org/html/draft-ietf-quic-transport-22 “6.3. Probe Timeout A Probe Timeout (PTO) triggers a probe packet when ack-eliciting data is in flight but an acknowledgement is not received within the expected period of time. A PTO enables a connection to recover from loss of tail packets or acks. The PTO algorithm used in QUIC implements the reliability functions of Tail Loss Probe [TLP] [RACK], RTO [RFC5681] and F-RTO algorithms for TCP [RFC5682], and the timeout computation is based on TCP's retransmission timeout period [RFC6298]. 6.3.1. Computing PTO
--	--

	When an ack-eliciting packet is transmitted, the sender schedules a timer for the PTO period as follows: $\text{PTO} = \text{smoothed_rtt} + \max(4 * \text{rttvar}, \text{kGranularity}) + \text{max_ack_delay}$ kGranularity, smoothed_rtt, rttvar, and max_ack_delay are defined in Appendix A.2 and Appendix A.3. The PTO period is the amount of time that a sender ought to wait for an acknowledgement of a sent packet. This time period includes the estimated network roundtrip-time (smoothed_rtt), the variance in the estimate (4*rttvar), and max_ack_delay, to account for the maximum time by which a receiver might delay sending an acknowledgement. The PTO value MUST be set to at least kGranularity, to avoid the timer expiring immediately. When a PTO timer expires, the sender probes the network as described in the next section. The PTO period MUST be set to twice its current value. This exponential reduction in the sender's rate is important because the PTOs might be caused by loss of packets or acknowledgements due to severe congestion. A sender computes its PTO timer every time an ack-eliciting packet is sent. A sender might choose to optimize this by setting the timer fewer times if it knows that more ack-eliciting packets will be sent within a short period of time” <i>QUIC Loss Detection and Congestion Control</i> https://datatracker.ietf.org/doc/html/draft-ietf-quic-recovery-22#section-6.3
--	---

Caveat: The notes and/or cited excerpts utilized herein are set forth for illustrative purposes only and are not meant to be limiting in any manner. For example, the notes and/or cited excerpts, may or may not be supplemented or substituted with different excerpt(s) of the relevant reference(s), as appropriate. Further, to the extent any error(s) and/or omission(s) exist herein, all rights are reserved to correct the same.