HTTP/3 explained

README

HTTP/3 explained is a collaborative effort to document the HTTP/3 and the QUIC protocols. Join in and help!

Get the Web or PDF versions on gitbook.com.

The contents get updated automatically on every commit to this git repository.

HTTP/3 explained cover

English

This book effort was started in March 2018. The plan is to document HTTP/3 and its underlying protocol: QUIC. Why, how they work, protocol details, the implementations and more.

The book is entirely free and is meant to be a collaborative effort involving anyone and everyone who wants to help out.

Prerequisites

A reader of this book is presumed to have a basic understanding of TCP/IP networking, the fundamentals of HTTP and the web. For further insights and specifics about HTTP/2 we recommend first reading up the details in http2 explained.

Author

This book is created and the work is started by Daniel Stenberg. Daniel is the founder and lead developer of curl, the world's most widely used HTTP client software. Daniel has worked with and on HTTP and internet protocols for over two decades and is the author of http2 explained.

Home

The home page for this book is found at daniel.haxx.se/http3-explained.

Help out

If you find mistakes, omissions, errors or blatant lies in this document, please send us a refreshed version of the affected paragraph and we will make amended versions. We will give proper credits to everyone who helps out. I hope to make this document better over time.

Preferably, you submit errors or pull requests on the book's GitHub page.

License

Why QUIC

QUIC is a name, not an acronym. It is pronounced exactly like the English word "quick".

QUIC is a new reliable and secure transport protocol that is suitable for a protocol like HTTP and that can address some of the known shortcomings of doing HTTP/2 over TCP and TLS. The logical next step in the web transport evolution.

QUIC is not limited to just transporting HTTP. The desire to make the web and data in general delivered faster to end users is probably the largest reason and push that initially triggered the creation of this new transport protocol.

So why create a new transport protocol and why do it on top of UDP?

QUIC logo

Remember HTTP/2

The HTTP/2 specification RFC 7540 was published in May 2015 and the protocol has since then been implemented and deployed widely across the Internet and the World Wide Web.

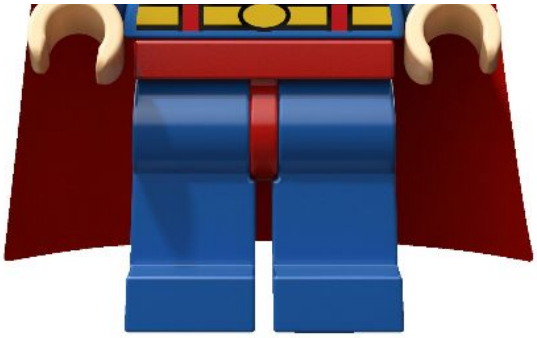
In early 2018, almost 40% of the top-1000 web sites run HTTP/2, around 70% of all HTTPS requests Firefox issues get HTTP/2 responses back and all major browsers, servers and proxies support it.

HTTP/2 addresses a whole slew of shortcomings in HTTP/1 and with the introduction of the second version of HTTP users can stop using a bunch of work-arounds. Some of which are pretty burdensome on web developers.

One of the primary features of HTTP/2 is that it makes use of multiplexing, so that many logical streams are sent over the same physical TCP connection. This makes a lot of things better and faster. It makes congestion control work much better, it lets users use TCP much better and thus properly saturate the bandwidth, makes the TCP connections more long-lived - which is good so that they get up to full speed more frequently than before. Header compression makes it use less bandwidth.

With HTTP/2, browsers typically use *one* TCP connection to each host instead of the previous *six*. In fact, connection coalescing and "desharding" techniques used with HTTP/2 may actually even reduce the number of connections much more than so.

HTTP/2 fixed the HTTP head of line blocking problem, where clients had to wait for the first request in line to finish before the next one could go out.

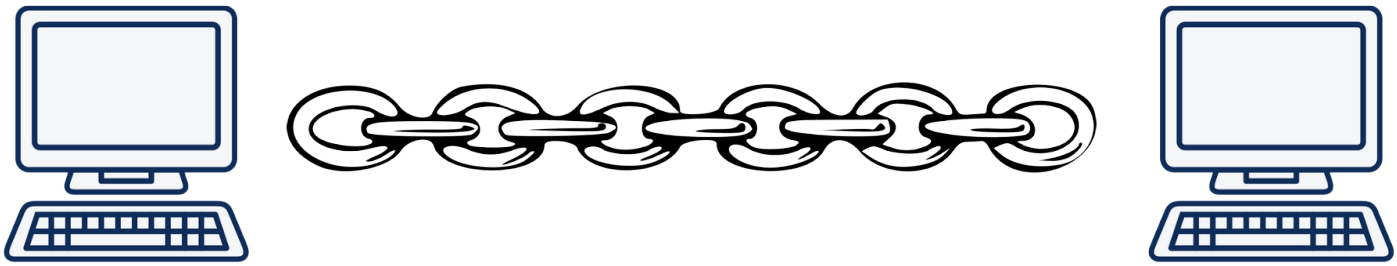


http2 man

TCP head of line blocking

TCP head of line blocking

HTTP/2 is done over TCP and with much fewer TCP connections than when using earlier HTTP versions. TCP is a protocol for reliable transfers and you can basically think of it as an imaginary chain between two machines. What is being put out on the network in one end will end up in the other end, in the same order - eventually. (Or the connection breaks.)

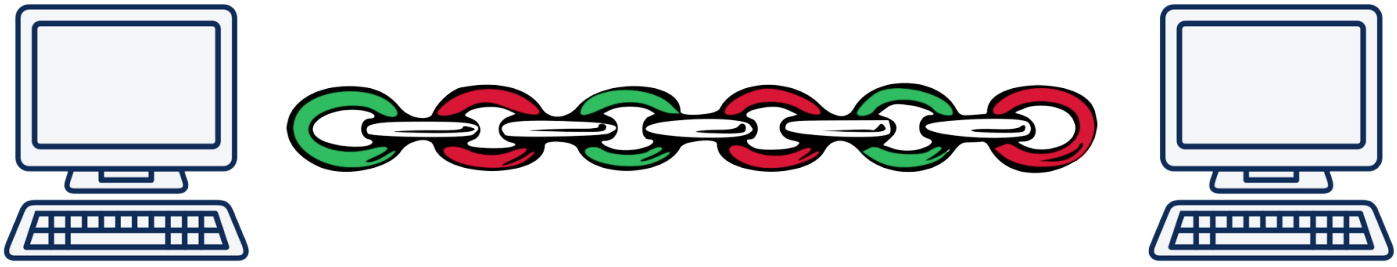


a TCP chain between two computers

With HTTP/2, typical browsers do tens or hundreds of parallel transfers over a single TCP connection.

If a single packet is dropped, or lost in the network somewhere between two endpoints that speak HTTP/2, it means the entire TCP connection is brought to a halt while the lost packet is re-transmitted and finds its way to the destination. Since TCP is this "chain", it means that if one link is suddenly missing, everything that would come after the lost link needs to wait.

An illustration using the chain metaphor when sending two streams over this connection. A red stream and a green stream:



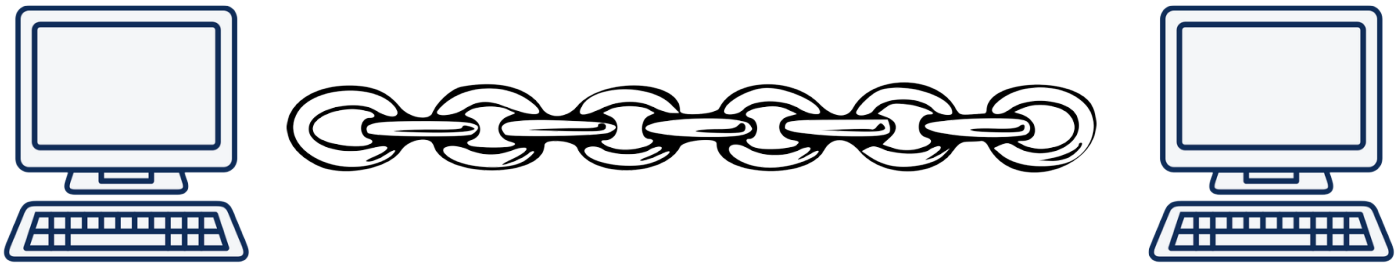
the chain showing links in different colors

It becomes a TCP-based head of line block!
As the packet loss rate increases, HTTP/2 performs less and less well. At 2% packet loss (which is a terrible network quality, mind you), tests have proven that HTTP/1 users are usually better off - because they typically have up to six TCP connections to distribute lost packets over. This means for every lost packet the other connections can still continue.

Fixing this issue is not easy, if at all possible, with TCP.

Independent streams avoids the block

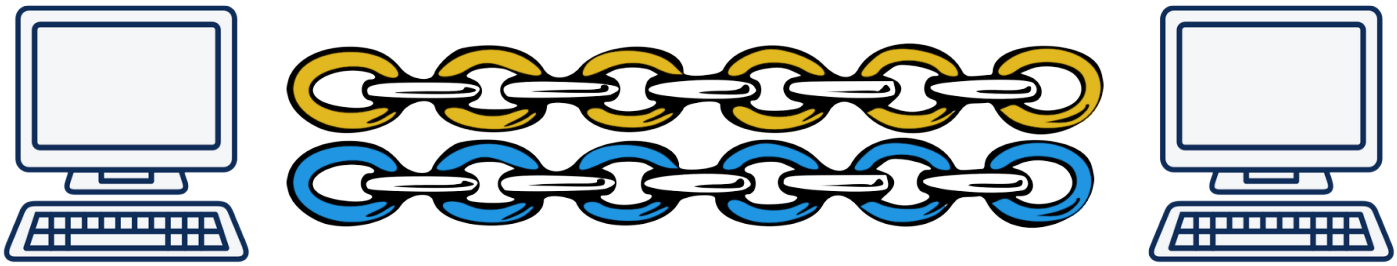
With QUIC there is still a connection setup between the two end-points that makes the connection secure and the data delivery reliable.



a QUIC chain between two computers

When setting up two different streams over this connection, they are treated independently so that if any link goes missing for one of the streams, only that stream, that particular chain, has to pause and wait for the missing link to get retransmitted.

Illustrated here with one yellow and one blue stream sent between two end-points.



two QUIC streams between two computers

TCP or UDP

TCP or UDP

If we can't fix head-of-line blocking within TCP, then in theory we should be able to make a new transport protocol next to UDP and TCP in the network stack. Or perhaps even use SCTP which is a transport protocol standardized by the IETF in RFC 4960 with several of the desired characteristics.

However, in recent years efforts to create new transport protocols have almost entirely been halted because

of the difficulties in deploying them on the Internet. Deployment of new protocols is hampered by many firewalls, NATs, routers and other middle-boxes that only allow TCP or UDP are deployed between users and the servers they need to reach. Introducing another transport protocol makes N% of the connections fail because they are being blocked by boxes that see it not being UDP or TCP and thus evil or wrong somehow. The N% failure rate is often deemed too high to be worth the effort.

Additionally, changing things in the transport protocol layer of the network stack typically means protocols implemented by operating system kernels. Updating and deploying new operating system kernels is a slow process that requires significant effort. Many TCP improvements standardized by the IETF are not widely deployed or used because they are not broadly supported.

Why not SCTP-over-UDP

SCTP is a reliable transport protocol with streams, and for WebRTC there are even existing implementations using it over UDP.

This was not deemed good enough as a QUIC alternative due to several reasons, including:

- SCTP does not fix the head-of-line-blocking problem for streams
- SCTP requires the number of streams to be decided at connection setup
- SCTP does not have a solid TLS/security story
- SCTP has a 4-way handshake, QUIC offers 0-RTT
- QUIC is a bytestream like TCP, SCTP is message-based
- QUIC connections can migrate between IP addresses but SCTP cannot

For more details on the differences, see A Comparison between SCTP and QUIC.

Ossification

The internet is a network of networks. There is equipment set up on the Internet in many different places along the way to make sure this network of networks works as it is supposed to. These devices, the boxes that are distributed out in the network, are what we sometimes refer to as middle-boxes. Boxes that sit somewhere between the two end-points that are the primary parties involved in a traditional network data transfer.

These boxes serve many different specific purposes but I think we can say that universally they are put there because someone thinks they must be there to make things work.

Middle-boxes route IP packets between networks, they block malicious traffic, they do NAT (Network Address Translation), they improve performance, some try to spy on the passing traffic and more.

In order to perform their duties these boxes must know about networking and the protocols that are monitored or modified by them. They run software for this purpose. Software that is not always upgraded frequently.

While they are glue components that keep the Internet together they are also often not keeping up with the latest technology. The middle of the network typically does not move as fast as the edges, as the clients and the servers of the world.

The network protocols that these boxes might want to inspect, and have ideas about what is okay and what is not then have this problem: these boxes were deployed some time ago when the protocols had a feature set of that time. Introducing new features or changes in behavior that were not known before risks ending up considered bad or illegal by such boxes. Such traffic may well just be dropped or delayed to the degree that users really do not want to use those features.

This is called "protocol ossification".

Changes to TCP also suffer from ossification: some boxes between a client and the remote server will spot unknown new TCP options and block such connections since they do not know what the options are. If allowed to detect protocol details, systems learn how protocols typically behave and over time it becomes impossible to change them.

The only truly effective way to "combat" ossification is to encrypt as much of the communication as possible in order to prevent middle-boxes from seeing much of the protocol passing through.

Secure

QUIC is always secure. There is no clear-text version of the protocol so to negotiate a QUIC connection means doing cryptography and security with TLS 1.3. As mentioned above, this prevents ossification as well as other sorts of blocks and special treatments, as well as making sure QUIC has all the secure properties of HTTPS that web users have come to expect and want.

There are only a few initial handshake packets that are sent in the clear before the encryption protocols have been negotiated.

Reduced latency

QUIC offers both 0-RTT and 1-RTT handshakes that reduce the time it takes to negotiate and setup a new connection. Compare with the 3-way handshake of TCP.

In addition to that, QUIC offers "early data" support from the get-go which is done to allow more data and it is used more easily than TCP Fast Open.

With the stream concept, another logical connection to the same host can be done at once without having to wait for the existing one to end first.

TCP Fast Open is problematic

TCP Fast Open was published as RFC 7413 in December 2014 and that specification describes how

applications can pass data to the server to be delivered already in the first TCP SYN packet.
Actual support for this feature in the wild has taken time and is riddled with problems even today in 2018. The TCP stack implementors have had issues and so have applications trying to take advantage of this feature - both in knowing in which OS version to try to activate it but also in figuring out how to gracefully back down and deal when problems arise. Several networks have been identified to interfere with TFO traffic and they have thus actively ruined such TCP handshakes.

Process

The initial QUIC protocol was designed by Jim Roskind at Google and was initially implemented in 2012, announced publicly to the world in 2013 when Google's experimentation broadened.

Back then, QUIC was still claimed to be an acronym for "Quick UDP Internet Connections", but that has been dropped since then.

Google implemented the protocol and subsequently deployed it both in their widely used browser (Chrome) and in their widely used server-side services (Google search, gmail, youtube and more). They iterated protocol versions fairly quickly and over time they proved the concept to work reliably for a vast portion of users.

In June 2015, the first internet draft for QUIC was sent to the IETF for standardization, but it took until late 2016 for a QUIC working group to get approved and started. But then it took off immediately with a high degree of interest from many parties.

In 2017, numbers quoted by QUIC engineers at Google mentioned that around 7% of *all* Internet traffic were already using this protocol. The Google version of the protocol.

IETF

The QUIC working group that was established to standardize the protocol within the IETF quickly decided that the QUIC protocol should be able to transfer other protocols than "just" HTTP. Google-QUIC only ever transported HTTP - in practice it transported what was effectively HTTP/2 frames, using the HTTP/2 frame syntax.

It was also stated that IETF-QUIC should base its encryption and security on TLS 1.3 instead of the "custom" approach used by Google-QUIC.

In order to satisfy the send-more-than-HTTP demand, the IETF QUIC protocol architecture was split in two separate layers: the transport QUIC and the "HTTP over QUIC" layer - the latter of which was renamed to HTTP/3 in November 2018.

This layer split, while it may sound innocuous, has caused the IETF-QUIC to differ quite a lot from the original Google-QUIC.

The working group did however soon decide that in order to get the proper focus and ability to deliver QUIC version 1 on time, it would focus on delivering HTTP, leaving non-HTTP transports to later work.

In March 2018 when we started working on this book, the plan was to ship the final specification for QUIC version 1 in November 2018 but this has been postponed a number of times and, at the time of writing (June 2020), is entering final stages.

While the work on IETF-QUIC has progressed, the Google team has incorporated details from the IETF version and has started to slowly progress their version of the protocol towards what the IETF version might become. Google has continued using their version of QUIC in their browser and services.

Most new implementations under development have decided to focus on the IETF version and are not compatible with the Google version.

Experience from HTTP/2

The HTTP/2 specification RFC 7540 was published in May 2015, just a month before QUIC was brought to IETF for the first time.

With HTTP/2, the foundation for changing HTTP over the wire was laid out and the working group that created HTTP/2 was already of the mindset that this would help iterating to new HTTP versions much faster than it had taken to go to version 2 from version 1 (about 16 years).

With HTTP/2, users and software stacks got used to the idea that HTTP can no longer be assumed to be done with a text-based protocol in a serial manner.

HTTP-over-QUIC (HTTP/3) builds upon HTTP/2 and follows many of the same concepts but moves some of the specifics from the HTTP layer as they are covered by QUIC.

Status

The QUIC working group has worked fiercely since late 2016 on specifying the protocols and is approaching final stages at the time of writing (June 2020).

During 2019 and 2020 there has been an increasing number of interoperability tests with HTTP/3 and CDNs and Browsers have started launching initial support - though often behind flags.

There are a number of different QUIC implementations listed in the QUIC working groups' wiki pages.

Implementing QUIC is not easy and the protocol has kept moving and changing even up to this date.

Servers

NGINX support for QUIC and HTTP/3 is under development and a preview version has been announced.

There have been no public statement in terms of support for QUIC from Apache.

Clients

None of the larger browser vendors have yet shipped any version, at any state, that can run the IETF version of QUIC or HTTP/3.

Google Chrome has shipped with a working implementation of Google's own QUIC version since many years and has recently started supporting the IETF version behind a flag. Firefox similarly supports this behind a flag.

curl shipped the first experimental HTTP/3 support (draft-22) in the 7.66.0 release on September 11, 2019. curl uses either the Quiche library from Cloudflare or the ngtcp2 family of libraries to get the work done.

Implementation Obstacles

QUIC decided to use TLS 1.3 as the foundation for the crypto and security layer to avoid inventing something new and instead lean on a trustworthy and existing protocol. However, while doing this, the working group also decided that to really streamline the use of TLS in QUIC, it should only use "TLS messages" and not "TLS records" for the protocol.

This might sound like an innocuous change, but this has actually caused a significant hurdle for many QUIC stack implementors. Existing TLS libraries that support TLS 1.3 simply do not have APIs enough to expose this functionality and allow QUIC to access it. While several QUIC implementors come from larger organizations who work on their own TLS stack in parallel, this is not true for everyone.

The dominant open source heavyweight OpenSSL for example, does not have any API for this. The plan to address this seems to happen in their PR 8797 that aims to introduce an API that is very similar to the one of BoringSSL.

This will eventually also lead to deployment obstacles since QUIC stacks will need to either base themselves on other TLS libraries, use a separate patched OpenSSL build or require an update to a future OpenSSL version.

Kernels and CPU load

Both Google and Facebook have mentioned that their wide scale deployments of QUIC require roughly twice the amount of CPU than the same traffic load does when serving HTTP/2 over TLS.

Some explanations for this include

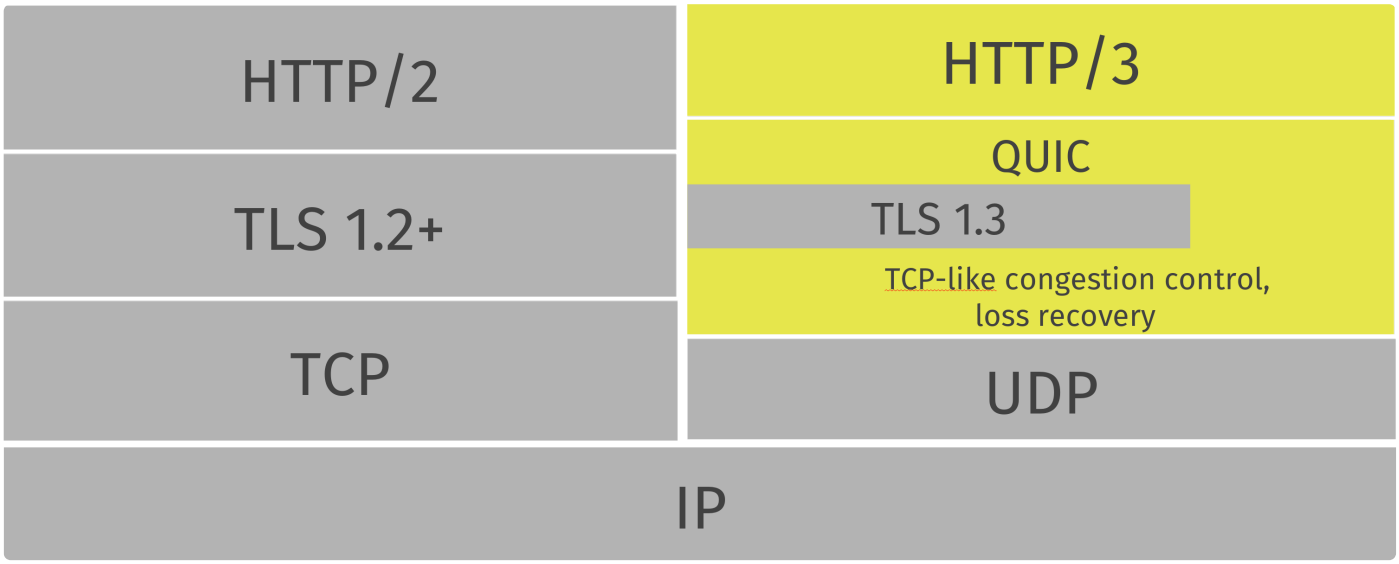
- the UDP parts in primarily Linux is not at all as optimized as the TCP stack is, since it has not traditionally been used for high speed transfers like this.
- TCP and TLS offloading to hardware exist, but that is much rarer for UDP and basically non-existing for QUIC.

There are reasons to believe that performance and CPU requirements will improve over time

Protocol features

The QUIC protocol from a high level.

Illustrated below is the HTTP/2 network stack on the left and the QUIC network stack on the right, when used as HTTP transport.



HTTP/3 over QUIC stack overview

UDP

Transfer protocol over UDP

QUIC is a transfer protocol implemented on top of UDP. If you watch your network traffic casually, you will see QUIC appear as UDP packets.

Based on UDP it also then uses UDP port numbers to identify specific network services on a given IP address.

All known QUIC implementations are currently in user-space, which allows for more rapid evolution than kernel-space implementations typically allow.

Will it work?

There are enterprises and other network setups that block UDP traffic on other ports than 53 (used for DNS). Others throttle such data in ways that makes QUIC perform worse than TCP based protocols. There is no end to what some operators may do.

For the foreseeable future, all use of QUIC-based transports will probably have to be able to gracefully fall-

back to another (TCP-based) alternative. Google engineers have previously mentioned measured failure rates in the low single-digit percentages.

Will it improve?

Chances are that if QUIC proves to be a valuable addition to the Internet world, people will want to use it and they will want it to function in their networks and then companies may start to reconsider their obstacles. During the years the development of QUIC has progressed, the success rate for establishing and using QUIC connections across the Internet has increased.

Reliable

While UDP is not a reliable transport, QUIC adds a layer on top of UDP that introduces reliability. It offers re-transmissions of packets, congestion control, pacing and the other features otherwise present in TCP.

Data sent over QUIC from one end-point will appear in the other end sooner or later, as long as the connection is maintained.

Streams

Similar to SCTP, SSH and HTTP/2, QUIC features separate logical streams within the physical connections. A number of parallel streams that can transfer data simultaneously over a single connection without affecting the other streams.

A connection is a negotiated setup between two end-points similar to how a TCP connection works. A QUIC connection is made to a UDP port and IP address, but once established the connection is associated by its "connection ID".

Over an established connection, either side can create streams and send data to the other end. Streams are delivered in-order and they are reliable, but different streams may be delivered out-of-order.

QUIC offers flow control on both connection and streams.

See further details in connections and streams sections

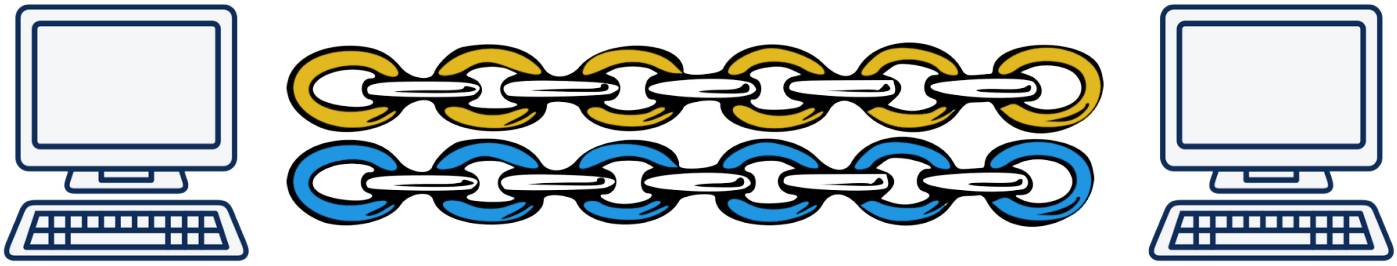
In Order

QUIC guarantees in-order delivery of streams, but not between streams. This means that each stream will send data and maintain data order, but each stream may reach the destination in a different order than the application sent it!

For example: stream A and B are transferred from a server to a client. Stream A is started first and then stream B. In QUIC, a lost packet only affects the stream to which the lost packet belongs. If stream A loses a

packet but stream B does not, stream B may continue its transfers and complete while stream A's lost packet is re-transmitted. This was not possible with HTTP/2.

Illustrated here with one yellow and one blue stream sent between two QUIC end-points over a single connection. They are independent and may arrive in a different order, but each stream is reliably delivered to the application in order.



two QUIC streams between two computers

Fast handshakes

QUIC offers both 0-RTT and 1-RTT connection setups, meaning that at best QUIC needs no extra round-trips at all when setting up a new connection. The faster of those two, the 0-RTT handshake, only works if there has been a previous connection established to a host and a secret from that connection has been cached.

Early data

QUIC allows a client to include data already in the 0-RTT handshake. This feature allows a client to deliver data to the peer as fast as it possibly can, and that then of course allows the server to respond and send data back even sooner.

TLS 1.3

The transport security used in QUIC is using TLS 1.3 (RFC 8446) and there are never any unencrypted QUIC connections.

TLS 1.3 has several advantages compared to older TLS versions but a primary reason for using it in QUIC is that 1.3 changed the handshake to require fewer roundtrips. It reduces protocol latency.

The Google legacy version of QUIC used a custom crypto.

Transport and application

The IETF QUIC protocol is a transport protocol, on top of which other application protocols can be used. The

initial application layer protocol is HTTP/3 (h3).

The transport layer supports connections and streams.

The legacy Google version of QUIC had transport and HTTP glued together into one single do-it-all and was a more special-purpose send-http/2-frames-over-udp protocol.

HTTP/3 over QUIC

The HTTP layer, called HTTP/3, does HTTP-style transports, including HTTP header compression using QPACK - which is similar to the HTTP/2 compression named HPACK.

The HPACK algorithm depends on an *ordered* delivery of streams so it was not possible to reuse it for HTTP/3 without modifications since QUIC offers streams that can be delivered out of order. QPACK can be seen as the QUIC-adapted version of HPACK.

Non-HTTP over QUIC

The work on sending protocols other than HTTP over QUIC has been postponed until after QUIC version 1 has shipped.

How QUIC works

Without explaining the exact bits and bytes on the wire, this section describes how the fundamental building blocks of the QUIC transport protocol work. If you want to implement your own QUIC stack, this description should give you a general understanding, but for all the details, refer to the actual IETF Internet Drafts and RFCs.

1. Set up a connection
2. ... that includes TLS security
3. Then use streams

Connections

A QUIC connection is a single conversation between two QUIC endpoints. QUIC's connection establishment combines version negotiation with the cryptographic and transport handshakes to reduce connection establishment latency.

To actually send data over such a connection, one or more streams have to be created and used.

Connection ID

Each connection possesses a set of connection identifiers, or connection IDs, each of which can be used to identify the connection. Connection IDs are independently selected by endpoints; each endpoint selects the connection IDs that its peer uses.

The primary function of these connection IDs is to ensure that changes in addressing at lower protocol layers (UDP, IP, and below) do not cause packets for a QUIC connection to be delivered to the wrong endpoint.

By taking advantage of the connection ID, connections can thus migrate between IP addresses and network interfaces in ways TCP never could. For instance, migration allows an in-progress download to move from a cellular network connection to a faster wifi connection when the user moves their device into a location offering wifi. Similarly, the download can proceed over the cellular connection if wifi becomes unavailable.

Port numbers

QUIC is built atop UDP, so a 16 bit port number field is used to differentiate incoming connections.

Version negotiation

A QUIC connection request originating from a client will tell the server which QUIC protocol version it wants to speak, and the server will respond with a list of supported versions for the client to select from.

Connections use TLS

Immediately after the initial packet setting up a connection, the initiator sends a crypto frame that starts setting up the secure layer handshake. The security layer uses TLS 1.3 security.

There is no way to opt-out or avoid using TLS for a QUIC connection. The protocol is designed to be hard for middle-boxes to tamper with, in order to help prevent ossification of the protocol.

Streams

Streams in QUIC provide a lightweight, ordered byte-stream abstraction.

There are two basic types of stream in QUIC:

- Unidirectional streams carry data in one direction: from the initiator of the stream to its peer.
- Bidirectional streams allow for data to be sent in both directions.

Either type of stream can be created by either endpoint, can concurrently send data interleaved with other streams, and can be canceled.

To send data over a QUIC connection, one or more streams are used.

Flow control

Streams are individually flow controlled, allowing an endpoint to limit memory commitment and to apply back pressure. The creation of streams is also flow controlled, with each peer declaring the maximum stream ID it is willing to accept at a given time.

Stream Identifiers

Streams are identified by an unsigned 62-bit integer, referred to as the Stream ID. The least significant two bits of the Stream ID are used to identify the type of stream (unidirectional or bidirectional) and the initiator of the stream.

The least significant bit (0x1) of the Stream ID identifies the initiator of the stream. Clients initiate even-numbered streams (those with the least significant bit set to 0); servers initiate odd-numbered streams (with the bit set to 1).

The second least significant bit (0x2) of the Stream ID differentiates between unidirectional streams and bidirectional streams. Unidirectional streams always have this bit set to 1 and bidirectional streams have this bit set to 0.

Stream concurrency

QUIC allows for an arbitrary number of streams to operate concurrently. An endpoint limits the number of concurrently active incoming streams by limiting the maximum stream ID.

The maximum stream ID is specific to each endpoint and applies only to the peer that receives the setting.

Sending and Receiving Data

Endpoints use streams to send and receive data. That is after all their ultimate purpose. Streams are an ordered byte-stream abstraction. Separate streams are however not necessarily delivered in the original order.

Stream Prioritization

Stream multiplexing has a significant effect on application performance if resources allocated to streams are correctly prioritized. Experience with other multiplexed protocols, such as HTTP/2, shows that effective prioritization strategies have a significant positive impact on performance.

QUIC itself does not provide frames for exchanging prioritization information. Instead it relies on receiving priority information from the application that uses QUIC. Protocols that use QUIC are able to define any prioritization scheme that suits their application semantics.

There have been criticisms of the HTTP/2 prioritisation model, and concerns it is overly complex and not used and implemented by many HTTP/2 servers. For now prioritisation in HTTP/3 has been removed from the main HTTP/3 specification and is being worked on as a separate specification.

0-RTT

To reduce the time required to establish a new connection, a client that has previously connected to a server may cache certain parameters from that connection and subsequently set up a **0-RTT** connection with the server. This allows the client to send data immediately, without waiting for a handshake to complete.

Spin Bit

One of the perhaps longest design discussions within the QUIC working group that has been the subject of several hundred mails and hours of discussions concerns a single bit: the Spin Bit.

The proponents of this bit insist that there is a need for operators and people on the path between two QUIC endpoints to be able to measure latency.

The opponents to this feature do not like the potential information leak.

Spinning a bit

Both endpoints, the client and the server, maintain a spin value, 0 or 1, for each QUIC connection, and they set the spin bit on packets it sends for that connection to the appropriate value.

Both sides then send out packets with that spin bit set to the same value for as long as one round trip lasts and then it toggles the value. The effect is then a pulse of ones and zeroes in that bitfield that observers can monitor.

This measuring only works when the sender is neither application nor flow control limited and packet reordering over the network can also make the data noisy.

User space

Implementing a transport protocol in user-space helps enable quick iteration of the protocol, as it is comparatively easy to evolve the protocol without necessitating that clients and servers update their operating system kernel to deploy new versions.

Nothing inherent in QUIC prevents it from being implemented and offered by operating system kernels in the future, should someone find that a good idea.

Many implementations

One obvious effect of implementing a new transport protocol in user-space is that we can expect to see many independent implementations.

Different applications are likely to include (or layer atop) different HTTP/3 and QUIC implementations for the foreseeable future.

API

One of the success factors for regular TCP and programs using that, is the standardized socket API. It has well defined functionality and using this API you can move programs between many different operating systems as TCP works the same.

QUIC is not there. There is no standard API for QUIC.

With QUIC, you need to pick one of the existing library implementations and stick with its API. It makes applications "locked in" to a single library to some extent. Changing to another library means another API and that might involve a lot of work.

Also, since QUIC is typically implemented in user-space, it can't easily just extend the socket API or appear similar to how existing TCP and UDP functionality do. Using QUIC will mean using another API than the socket API.

HTTP/3

As mentioned previously, the first and primary protocol to transport over QUIC is HTTP.

Much like HTTP/2 was once introduced to transport HTTP over the wire in a completely new way, HTTP/3 is yet again introducing a new way to send HTTP over the network.

HTTP still maintains the same paradigms and concepts like before. There are headers and a body, there is a request and a response. There are verbs, cookies and caching. What primarily changes with HTTP/3 is how the bits gets sent over to the other side of the communication.

In order to do HTTP over QUIC, changes were required and the results of this is what we now call HTTP/3.

These changes were required because of the different nature that QUIC provides as opposed to TCP. These changes include:

- In QUIC the streams are provided by the transport itself, while in HTTP/2 the streams were done within the HTTP layer.
- Due to the streams being independent of each other, the header compression protocol used for HTTP/2 could not be used without it causing a head of block situation.
- QUIC streams are slightly different than HTTP/2 streams. The HTTP/3 section will detail this somewhat.

HTTPS:// URLs

HTTP/3 will be performed using `HTTPS://` URLs. The world is full of these URLs and it has been deemed impractical and downright unreasonable to introduce another URL scheme for the new protocol. Much like HTTP/2 did not need a new scheme, neither will HTTP/3.

The added complexity in the HTTP/3 situation is however that where HTTP/2 was a completely new way of transporting HTTP over the wire, it was still based on TLS and TCP like HTTP/1 was. The fact that HTTP/3 is done over QUIC changes things in a few important aspects.

Legacy, clear-text, `HTTP://` URLs will be left as-is and as we proceed further into a future with more secure transfers they will probably become less and less frequently used. Requests to such URLs will simply not be upgraded to use HTTP/3. In reality they rarely upgrade to HTTP/2 either, but for other reasons.

Initial connection

The first connection to a fresh, not previously visited host for a HTTPS:// URL probably has to be done over TCP (possibly in addition to a parallel attempt to connect via QUIC). The host might be a legacy server without QUIC support or there might be a middle box in between setting up obstacles preventing a QUIC connection from succeeding.

A modern client and server would presumably negotiate HTTP/2 in the first handshake. When the connection has been setup and the server responds to a client HTTP request, the server can tell the client about its support of and preference for HTTP/3.

Bootstrap with Alt-svc

The alternate service (Alt-svc:) header and its corresponding `ALT-SVC` HTTP/2 frame are not specifically created for QUIC or HTTP/3. They are part of an already designed and created mechanism for a server to tell a client: *"look, I run the same service on THIS HOST using THIS PROTOCOL on THIS PORT"*. See details in RFC 7838.

A client that receives such an Alt-svc response is then advised to, if it supports and wants to, connect to that given other host in parallel in the background - using the specified protocol - and if it is successful switch its

operations over to that instead of the initial connection.

If the initial connection uses HTTP/2 or even HTTP/1, the server can respond and tell the client that it can connect back and try HTTP/3. It could be to the same host or to another one that knows how to serve that origin. The information given in such an Alt-svc response has an expiry timer, making clients able to direct subsequent connections and requests directly to the alternative host using the suggested alternative protocol, for a certain period of time.

Example

An HTTP server includes an `Alt-Svc:` header in its response:

```
1 Alt-Svc: h3=":50781"
```

This indicates that HTTP/3 is available on UDP port 50781 at the same host name that was used to get this response.

A client can then attempt to setup a QUIC connection to that destination and if successful, continue communicating with the origin like that instead of the initial HTTP version.

QUIC streams and HTTP/3

HTTP/3 is made for QUIC so it takes full advantage of QUIC's streams, where HTTP/2 had to design its entire stream and multiplexing concept of its own on top of TCP.

HTTP requests done over HTTP/3 use a specific set of streams.

HTTP/3 frames

HTTP/3 means setting up QUIC streams and sending over a set of frames to the other end. There's but a small fixed number (actually nine on December 18th, 2018!) of known frames in HTTP/3. The most important ones are probably:

- HEADERS, that sends compressed HTTP headers
- DATA, sends binary data contents
- GOAWAY, please shutdown this connection

HTTP Request

The client sends its HTTP request on a client-initiated *bidirectional* QUIC stream.

A request consists of a single HEADERS frame and might optionally be followed by one or two other frames: a series of DATA frames and possibly a final HEADERS frame for trailers.

After sending a request, a client closes the stream for sending.

HTTP Response

The server sends back its HTTP response on the bidirectional stream. A HEADERS frame, a series of DATA frames and possibly a trailing HEADERS frame.

QPACK headers

The HEADERS frames contain HTTP headers compressed using the QPACK algorithm. QPACK is similar in style to the HTTP/2 compression called HPACK (RFC 7541), but modified to work with streams delivered out of order.

QPACK itself uses two additional unidirectional QUIC streams between the two end-points. They are used to carry dynamic table information in either direction.

Prioritization

As mentioned previously, priorisation between streams has been removed from the main HTTP/3 specification to be worked on separately.

This was due to learnings from the HTTP/2 prioritisation model and its implementation (or lack there of) in the real world.

A simpler prioritisation model than HTTP/2 has been proposed using HTTP header fields with a limited number of priority settings. This is a key change from the Dependency and Weight flags in the HTTP/2 Header frames and allows better understanding at the application layer.

Reprioritisation, and whether to support this, is still being discussed. HTTP/2 had Prioritisation frames to handle this but the true independent streams in QUIC and HTTP/3 makes this more complicated so the benefits versus the complexity are still being debated.

When (or if!) a better prioritisation model is agreed for HTTP/3 it is hoped to be able to backport this to HTTP/2 too, to address the complexity and implementation concerns there.

Server push

HTTP/3 server push is similar to what is described in HTTP/2 (RFC 7540), but uses different mechanisms.

A server push is effectively the response to a request that the client never sent!

Server pushes are only allowed to happen if the client side has agreed to them. In HTTP/3 the client even sets a limit for how many pushes it accepts by informing the server what the max push stream ID is. Going over that limit will cause a connection error.

If the server deems it likely that the client wants an extra resource that it hasn't asked for but ought to have anyway, it can send a `PUSH_PROMISE` frame (over the request stream) showing what the request looks like that the push is a response to, and then send that actual response over a new stream.

Even when pushes have been said to be acceptable by the client before-hand, each individual pushed stream can still be canceled at any time if the client deems that suitable. It then sends a `CANCEL_PUSH` frame to the server.

Problematic

Ever since this feature was first discussed in the HTTP/2 development and later after the protocol shipped and has been deployed over the Internet, this feature has been discussed, disliked and pounded up in countless different ways in order to get it to become useful.

Pushing is never "free", since while it saves a half round-trip it still uses bandwidth. It is often hard or impossible for the server-side to actually know with a high level of certainty if a resource should be pushed or not.

Comparison with HTTP/2

HTTP/3 is designed for QUIC, which is a transport protocol that handles streams by itself.

HTTP/2 is designed for TCP, and therefore handles streams in the HTTP layer.

Similarities

The two protocols offer clients virtually identical feature sets.

- Both protocols offer server push support
- Both protocols have header compression, and QPACK and HPACK are similar in design.
- Both protocols offer multiplexing over a single connection using streams

---

Differences

The differences are in the details and primarily there thanks to HTTP/3's use of QUIC:

- HTTP/3 has better and more likely to work early data support thanks to QUIC's 0-RTT handshakes, while TCP Fast Open and TLS usually sends less data and often faces problems.
- HTTP/3 has much faster handshakes thanks to QUIC vs TCP + TLS.
- HTTP/3 does not exist in an insecure or unencrypted version. HTTP/2 can be implemented and used without HTTPS - even if this is rare on the Internet.
- HTTP/2 can be negotiated directly in a TLS handshake with the ALPN extension, while HTTP/3 is over QUIC so it needs an `Alt-Svc:` header response first to inform the client about this fact.
- HTTP/3 has no prioritization. The HTTP/2 approach to prioritization has been deemed too complicated, or even a downright failure, and there's work on creating a simpler take. This planned simpler scheme is also planned to be able so backport to run over HTTP/2 using HTTP/2's extension mechanism.

Common criticism

UDP will never work

A lot of enterprises, operators and organizations block or rate-limit UDP traffic outside of port 53 (used for DNS) since it has in recent days mostly been abused for attacks. In particular, some of the existing UDP protocols and popular server implementations for them have been vulnerable for amplification attacks where one attacker can make a huge amount of outgoing traffic to target innocent victims.

QUIC has built-in mitigation against amplification attacks by requiring that the initial packet must be at least 1200 bytes and by restriction in the protocol that says that a server must not send more than three times the size of the request in response without receiving a packet from the client in response.

UDP is slow in kernels

This seems to be the truth, at least initially. We can of course not tell how this will develop and how much of this is simply the result of UDP transfer performance not having been in developers' focus for many years.

For most clients, this "slowness" is probably never even noticeable.

QUIC takes too much CPU

Similar to the "UDP is slow" remark above, this is partly because the TCP and TLS usage of the world has had a longer time to mature, improve and get hardware assistance.

There are reasons to expect this to improve over time, and already we are seeing some improvements in this space. The question is how much this extra CPU usage will hurt deployers.

This is just Google

No it is not. Google brought the initial spec to the IETF after having proved, on a large Internet-wide scale, that deploying this style of protocol over UDP actually works and performs well.

Since then, individuals from a large number of companies and organizations have worked in the vendor-neutral organization IETF to put together a standard transport protocol out of it. In that work, Google employees have of course been participating, but so have employees from a large number of other companies that are interested in furthering the state of transport protocols on the Internet, including Mozilla, Fastly, Cloudflare, Akamai, Microsoft, Facebook and Apple.

This is too small of an improvement

That is not really a critique but an opinion. Maybe it is, and maybe it is too little of an improvement so close in time since HTTP/2 was shipped.

HTTP/3 is likely to perform much better in packet loss-ridden networks, it offers faster handshakes so it will improve latency both as perceived and actual. But is that enough of benefits to motivate people to deploy HTTP/3 support on their servers and for their services? Time and future performance measurements will surely tell!

The specifications

Here is a collection of the latest official drafts for the various parts and components of QUIC and HTTP/3.

Invariants

Version-Independent Properties of QUIC

Transport

QUIC: A UDP-Based Multiplexed and Secure Transport

Recovery

QUIC Loss Detection and Congestion Control

TLS

Using TLS to Secure QUIC

HTTP

Hypertext Transfer Protocol Version 3 (HTTP/3)

QPACK

QPACK: Header Compression for HTTP/3

QUIC v2

In order to get the most possibly focus on the core QUIC protocol and be able to ship it on time, several features that were originally planned to be part of the core protocol have been postponed and are now planned to instead get done in a subsequent QUIC version. QUIC version 2 or beyond.

The author of this document does however have a rather faulty crystal ball so we can not tell for sure exactly what features will or will not appear in version 2. We can however mention some of the features and things that explicitly have been removed from the v1 work to be "worked on later" and that then possibly might appear in a version 2.

Forward Error Correction

Forward error correction (FEC) is a method of obtaining error control in data transmission in which the transmitter sends redundant data and the receiver recognizes only the portion of the data that contains no apparent errors.

Google experimented with this in their original QUIC work but it was subsequently removed again since the experiments did not turn out well. This feature is subject for discussion for QUIC v2 but probably takes for someone to prove that it actually can be a useful addition without too much penalty.

Multipath

Multipath means that the transport can by itself use multiple network paths to maximize resource usage and increase redundancy.

The SCTP proponents of the world like to mention that SCTP already features multipath and so does modern TCP.

Unreliable data

It has been discussed to offer "unreliable" streams as an option, that would then allow QUIC to also replace UDP-style applications.

Non-HTTP adaptions

DNS over QUIC was one of the early mentioned non-HTTP protocols that just might get some attention once QUIC v1 and HTTP/3 ship. But with a new transport like this having been brought on to the world I cannot imagine that it will stop there.

Deutsch

Die Arbeit an diesem Buch wurde im März 2018 gestartet. Es dient der Dokumentation von HTTP/3 und dessen zugrundeliegendem Protokoll: QUIC. Warum, wie sie funktionieren, Protokolldetails, die Implementierung und mehr.

Dieses Buch ist zur Gänze frei zugänglich und wird als kollaborative Angstrengung von jedem getragen, der aushelfen will.

Voraussetzungen

Einer Leserin oder Leser wird ein Grundverständnis von TCP/IP, den Grundlagen von HTTP und dem Web vorausgesetzt. Für weitere Einblicke und Details über HTTP/2 empfehlen wir: http2 explained.

Autor

Dieses Buch wurde geschrieben von Daniel Stenberg. Daniel ist der Gründer und Lead Developer von curl, der meistgenutzten HTTP Client Software der Welt. Daniel hat über zwei Jahrzehnte lang mit und an HTTP sowie Internetprotokollen gearbeitet und ist der Autor von http2 explained.

Website

Die Website von diesem Buch kann hier gefunden werden: daniel.haxx.se/http3-explained.

Aushelfen

Solltest du Irrtümer, fehlende Details, Fehler oder offensichtliche Lügen in diesem Dokument finden, dann sende uns bitte eine ausgebesserte Version des betroffenen Paragrafen und wir werden diese als neue Version veröffentlichen. Natürlich führen wir alle an, die aushelfen. Ich hoffe, dass wir das Dokument im Laufe der Zeit verbessern können.

Fehler sollten als Issues oder als Pull Requests auf der GitHub Seite des Buches übermittelt werden.

Lizenz

Warum QUIC

QUIC ist ein Name, kein Akronym. Es wird genauso wie das englische Wort "quick" ausgesprochen.

Aus vielen verschiedenen Blickwinkeln kann QUIC als Lösung verstanden werden, wie ein neues, zuverlässiges und sicheres Transportprotokoll umgesetzt wird, das für ein Protokoll wie HTTP adäquat ist und die durch die Umsetzung von HTTP/2 über TCP und TLS entstandenen Mängel behebt. Der logische nächste Schritt in der Web-Transport Evolution.

QUIC ist nicht nur auf den Transport von HTTP limitiert. Der Wunsch nach einer schnelleren Auslieferung von Daten zum Endnutzer über das Web ist wahrscheinlich der größte Grund, der die Erschaffung dieses neuen Transportprotokolls angestoßen hat.

Warum also ein neues Transportprotokoll erschaffen und warum aufbauend auf UDP?

QUIC logo

Erinnerst du dich an HTTP/2?

Die HTTP/2 Spezifikation RFC 7540 wurde im Mai 2015 veröffentlicht und seitdem flächendeckend im Internet und World Wide Web implementiert sowie ausgeliefert.

Anfang 2018 wurden fast 40% der Top 1000 Websites mit HTTP/2 ausgeliefert. Ungefähr 70% aller HTTPS Anfragen, die über Firefox verschickt wurden, bekamen eine HTTP/2 Antwort zurück. Die meistgenutzten Browser, Server und Proxies unterstützen die Spezifikation.

HTTP/2 behebt eine Fülle an Mängel von HTTP/1, weshalb HTTP-Nutzer nicht mehr zu Übergangslösungen greifen müssen - welche gerade Web Entwickler oft belastet haben.

Eines der Hauptmerkmale von HTTP/2 ist Multiplexing, die Möglichkeit viele logische Streams über die gleiche physische TCP Verbindung zu schicken. Das macht viele Dinge besser und schneller; es verbessert die Überlastungskontrolle wesentlich, es lässt User TCP besser nutzen um die Bandbreite besser auszunutzen, die TCP Verbindungen sind langlebiger - was gut ist, weil Verbindungen öfter die volle Geschwindigkeit aufbauen können. Header-Komprimierung nutzt dabei weniger Bandbreite.

Mit HTTP/2 nutzen Browser typischerweise *eine* TCP Verbindung zu jedem Host, anstatt der vorherigen *sechs*. Tatsächlich verringert sich die Anzahl der Verbindungen wegen der Verbindungsvereinigung und "Desharding" Techniken von HTTP/2 noch viel wesentlicher.

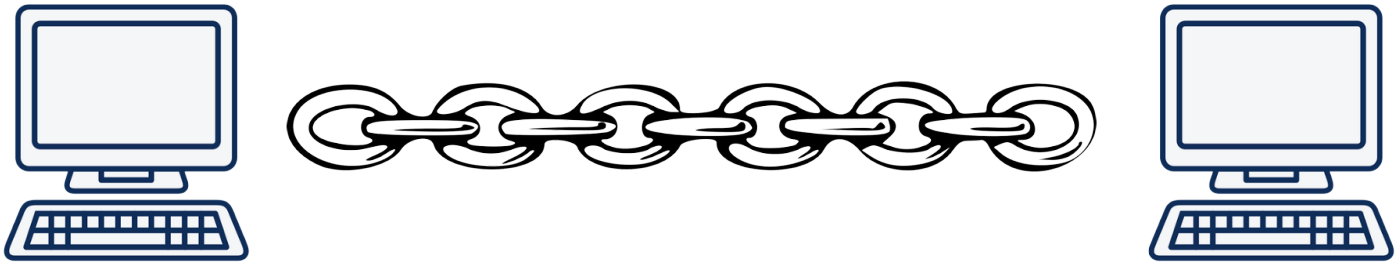
HTTP/2 löst das Problem des sogenannten HTTP Head-of-line blockings, bei dem Clients so lange warten müssen, bis eine gestellte Anfrage fertig ist, bevor die nächste gestellt werden kann.

http2 man

TCP Head-of-line blocking

TCP Head-of-line blocking

HTTP/2 nutzt TCP, wenngleich es im Gegensatz zu früheren HTTP Versionen weniger TCP Verbindungen benötigt. TCP ist ein Protokoll für eine zuverlässige Übertragung und kann als imaginäre Kette zwischen zwei Computern gesehen werden. Was vom ersten Computer über das Netzwerk übermittelt wird, wird schließlich beim zweiten Computer in der gleichen Reihenfolge ankommen (oder die Verbindung ist unterbrochen).

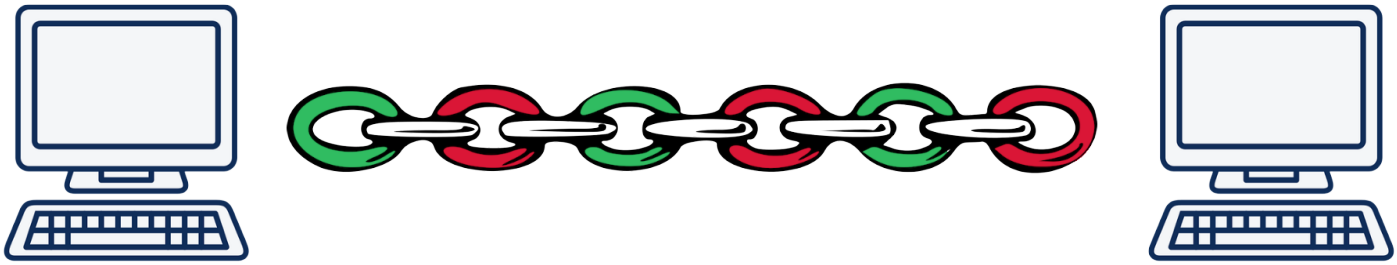


Eine TCP Kette zwischen zwei Computern

Mit HTTP/2 machen Browser zehn oder hunderte Übertragungen gleichzeitig über eine einzige TCP Verbindung.

Wenn ein einziges Paket fallen gelassen wird oder im Netzwerk irgendwo zwischen den beiden Endpunkten - welche über HTTP/2 kommunizieren - verloren geht, dann wird die gesamte TCP Verbindung solange gestoppt, bis das verlorene Paket wieder übertragen wurde und ihren Weg zum Ziel gefunden hat. Weil TCP diese "Kette" darstellt, muss alles - sollte eine Verbindung plötzlich fehlen - was nach der verlorenen Verbindung kommen würde, warten.

Eine Illustration der Ketten-Metapher, wenn zwei Streams über diese Verbindung versendet werden. Ein roter und ein grüner Stream:



Die Kette zeigt zwei Verbindungen in unterschiedlichen Farben

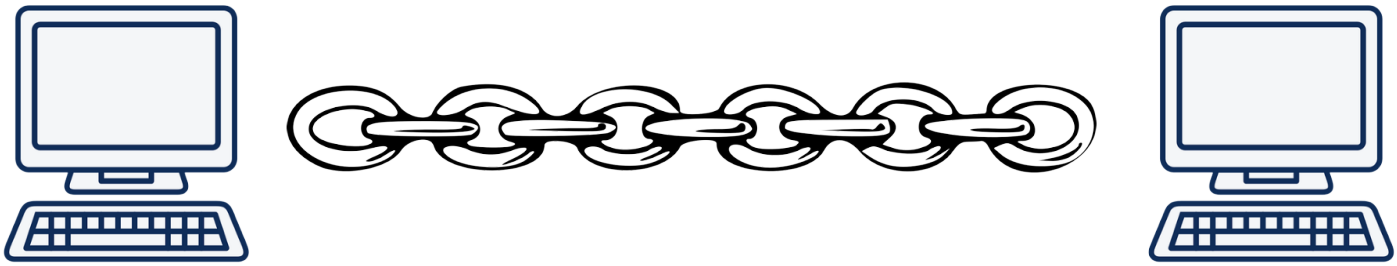
Es resultiert ein TCP basierender Head-of-line block!

Steigt die Verlustrate von Paketen, liefert HTTP/2 eine immer schlechtere Performance. Bei einer 2%-igen Verlustrate (was wohlgemerkt eine schlechte Netzwerkqualität ist) haben Tests gezeigt, dass HTTP/1 User normalerweise besser aussteigen - weil diese bis zu sechs TCP-Verbindungen haben, auf welche verlorene Pakete aufgeteilt werden können. Das bedeutet, dass für jedes verlorene Paket eine andere Verbindung weiterarbeiten kann.

Dieses Problem zu beheben ist nicht einfach - wenn überhaupt mit TCP möglich.

Unabhängige Streams vermeiden den Block

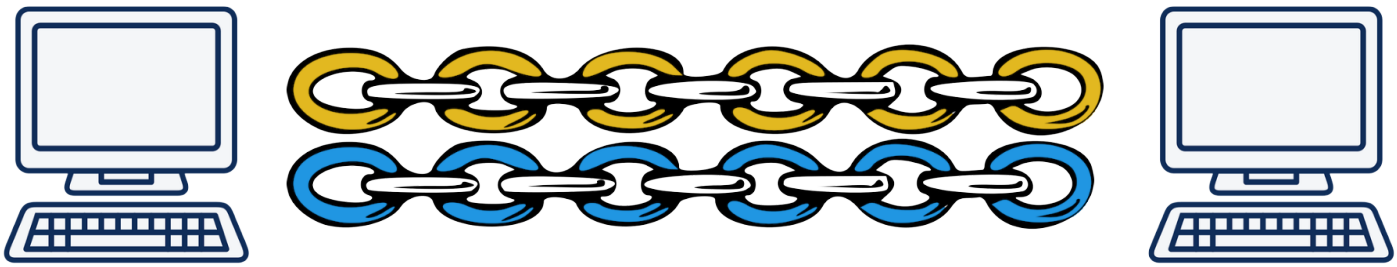
Mit QUIC gibt es noch immer einen Verbindungsaufbau zwischen den beiden Endpunkten, welcher die Verbindung sicher und den Datentransport zuverlässig macht.



Eine QUIC Kette zwischen zwei Computern

Wenn zwei unterschiedliche Streams über diese Verbindung aufgebaut werden, werden diese unabhängig voneinander behandelt, sodass - sollte eine Verbindung für einen der Streams verloren gehen - nur der eine Stream, diese eine Kette pausieren und darauf warten muss, dass die fehlende Verbindung wieder versendet wird.

Hier wird dies mit einem gelben und einem blauen Stream illustriert, die zwischen zwei Endpunkten versendet werden.



Zwei QUIC Streams zwischen zwei Computern

TCP oder UDP

TCP oder UDP

Wenn wir Head-of-line blocking innerhalb von TCP nicht reparieren können, dann sollten wir theoretisch ein

neues Transportprotokoll neben UDP und TCP im Netzwerkstack erstellen können. Oder vielleicht sogar SCTP nutzen; ein Transportprotokoll mit etlichen der von uns gewünschten Charakteristiken, welches von IETF in RFC 4960 standardisiert wurde.

Jedoch ist die Entwicklung von neuen Transportprotokollen in den letzten Jahren nahezu vollständig eingestellt worden, weil diese nur unter schwierigen Umständen im Internet eingesetzt werden können. Der Einsatz neuer Protokolle wird durch viele Firewalls, NATs, Router und andere Netzwerk-Appliances verhindert, welche lediglich TCP oder UDP in der Kommunikation zwischen User und Server zulassen. Ein weiteres Transportprotokoll einzuführen hat die Auswirkung, dass N% der Verbindungen fehlschlagen, weil diese durch Appliances geblockt werden, welche die Verbindungen wegen der Nutzung eines anderen Protokolls als UDP oder TCP als böswillig einstufen. Die N% Fehlerrate wird oft als zu hoch erachtet, um den Aufwand zu rechtfertigen.

Werden Änderungen auf der Ebene des Transportprotokolls im Netzwerkstack vorgenommen, bedeutet das, dass Protokolle über den Kernel des Betriebssystems implementiert werden. Kernel des Betriebssystems zu aktualisieren und auszuliefern ist ein langsamer Prozess, der signifikanten Aufwand mit sich bringt. Viele Verbesserungen von TCP, welche durch IETF standardisiert wurden, sind nicht flächendeckend eingesetzt, weil sie nicht umfassend unterstützt werden.

Warum nicht SCTP-over-UDP

SCTP ist ein zuverlässiges Transportprotokoll mit Streams. Für WebRTC gibt es sogar existierende Implementierungen von SCTP über UDP.

Aus folgenden Gründen wurde SCTP-over-UDP als nicht gut genug befunden, um QUIC zu ersetzen:

- SCTP löst das Head-of-line-blocking Problem für Streams nicht
- SCTP hat keine solide TLS/Security Geschichte
- SCTP hat einen 4-way handshake, QUIC bietet 0-RTT
- QUIC ist ein Bytestream wie TCP, SCTP basiert auf Nachrichten
- QUIC Verbindungen können zwischen IP Adressen migrieren, was SCTP nicht kann

  Für weitere Details zu den Unterschieden empfiehlt sich folgende Übersicht: A Comparison between SCTP and QUIC.

Ossifikation

Das Internet ist ein Netzwerk der Netzwerke. Um sicherzustellen, dass dieses Netzwerk der Netzwerke so funktioniert, wie es soll, gibt es Infrastruktur an vielen verschiedenen Plätzen. Diese Geräte - jene Boxen, die über das Netzwerk verteilt sind - bezeichnen wir manchmal als "Middle-Boxes". Boxen, die irgendwo zwischen den beiden Endpunkten sitzen und die primären Beteiligten in einem traditionellen Netzwerk-Datentransfer sind.

Diese Boxen dienen vielen verschiedenen spezifischen Zwecken - aber ich glaube sagen zu können, dass sie dort genutzt werden, weil jemand glaubt, dass diese dort sein müssen, damit alles funktioniert.

Middle-Boxes leiten IP-Pakete zwischen Netzwerken, sie blockieren böswilligen Traffic, führen die NAT (Network Address Translation, zu Deutsch "Netzwerkadressübersetzung") durch, verbessern die Leistung, manche versuchen den Traffic auszuspionieren und vieles mehr.

Um ihren Pflichten gerecht zu werden, müssen diese Boxen Networking und jene Protokolle kennen, die sie überwachen oder modifizieren. Dazu nutzen sie Software - die aber nicht oft aufgerüstet wird.

Auch wenn sie wie Klebstoff das Internet zusammenhalten, halten sie oft nicht mit der neuesten Technologie mit. Die Mitte eines Netzwerks verändert sich nicht so schnell wie die Enden, wie die Clients und Server dieser Welt.

Die Netzwerkprotokolle, die diese Boxen inspizieren wollen und kennen, haben dann folgendes Problem: die Boxen wurden vor einiger Zeit ausgeliefert, als die Protokolle gewisse Features hatten. Neue Features oder Veränderungen in der Verhaltensweise riskieren, dass Boxen diese als schlecht oder illegal interpretieren. Solcher Traffic könnte fallen gelassen oder verzögert werden - bis hin zu einem Ausmaß, dass User diese Features nicht nutzen wollen.

Das wird "Protokoll-Ossifikation" genannt.

Änderung an TCP leiden auch an Ossifikation: einige Boxen zwischen einem Client und einem entfernten Server erkennen neue TCP Optionen und blockieren solche Verbindungen, weil diese die Optionen nicht kennen. Wenn sie Protokolldetails erkennen dürfen, werden Systeme lernen, wie sich Protokolle normalerweise verhalten und im Laufe der Zeit wird es unmöglich sie zu ändern.

Der einzig effektive Weg um Ossifikation zu bekämpfen, ist die Verschlüsselung von so viel Kommunikation wie möglich. Damit werden Middle-Boxes daran gehindert, von durchgeleiteten Protokollen viel einsehen zu können.

Sicherheit

QUIC ist immer sicher. Es gibt keine Klartext-Version des Protokolls, daher involviert der Verbindungsaufbau einer QUIC Verbindung Kryptographie und Sicherheit mit TLS 1.3. Wie vorhergehend angemerkt, verhindert dies eine Ossifikation sowie andere Blockierungen und spezielle Handhabung - und sorgt dafür, dass QUIC alle sicheren Eigenschaften von HTTPS hat, die User des Webs erwarten und sich wünschen.

Es gibt nur wenige initiale Handshake-Pakete, die im Klaren geschickt werden, bevor die Verschlüsselungsprotokolle verhandelt wurden.

Reduzierte Latenz

QUIC bietet 0-RTT und 1-RTT Handshakes, welche die Zeit des Verhandelns sowie den Aufbau einer neuen Verbindung verkürzen. Vergleiche diesen Prozess mit dem 3-Schritt Handshake von TCP.

Zusätzlich bietet QUIC "Daten im Voraus" Support, welcher größeren Datenfluss erlaubt und im Vergleich zu TCP Fast Open benutzerfreundlicher ist.

Mit dem Stream-Konzept kann sofort eine weitere logische Verbindung zum gleichen Host aufgebaut werden, ohne auf das Ende einer bereits bestehenden Verbindung warten zu müssen.

TCP Fast Open ist problematisch

TCP Fast Open wurde als RFC 7413 im Dezember 2014 veröffentlicht. Diese Spezifikation beschreibt, wie Applikationen Daten zum Server bereits im ersten TCP SYN Paket übermitteln können.

Der tatsächliche Support für dieses Feature hat lange gedauert und ist bis ins heutige Jahr 2018 mit Problemen gespickt. Diejenigen, die den TCP-Stack implementiert haben, hatten Probleme und somit auch Applikationen, die die Vorteile dieses Features nutzen wollten. Dies gilt sowohl für das Verständnis der zu aktivierenden Betriebssystemversionen als auch für das Lösen von Problemen im Zusammenhang mit der mangelnden Clientseitigen Unterstützung (backdown). Viele Netzwerke konnten identifiziert werden, die TFO-Traffic stören und daher den reibungslosen Ablauf von TCP-Handshakes beeinträchtigen.

Prozess

Das ursprüngliche QUIC Protokoll wurde von Jim Roskind bei Google entworfen, im Jahr 2012 erstmals implementiert und der Welt 2013 vorgestellt, als Google's Experiment erweitert wurde.

Damals galt QUIC noch als Akronym für "Quick UDP Internet Connections", aber das wurde seitdem eingestellt.

Google implementierte das Protokoll und nutzte es anschließend sowohl in seinem weit verbreiteten Browser (Chrome) als auch in seinen weit verbreiteten serverseitigen Diensten (Google-Suche, Gmail, YouTube und mehr). Sie haben ziemlich schnell zwischen Protokollversionen iteriert und im Laufe der Zeit bewiesen, dass das Konzept für einen großen Teil der Nutzer zuverlässig funktioniert.

Im Juni 2015 wurde der erste Internetentwurf von QUIC zur Standardisierung an die IETF übermittelt. Es dauerte jedoch bis Ende 2016, bis eine QUIC-Arbeitsgruppe genehmigt und gestartet wurde. Aber dann ist es mit großem Interesse von vielen Parteien sofort losgegangen.

Im Jahr 2017 gaben die QUIC-Ingenieuren bei Google an, dass rund 7% des *gesamten* Internetverkehrs bereits das Protokoll verwenden. Die Google-Version des Protokolls.

IETF

Die zur Standardisierung des Protokolls innerhalb der IETF eingerichteten QUIC-Arbeitsgruppe entschied bald, dass das QUIC-Protokoll auch andere Protokolle als "nur" HTTP übertragen können soll. Google-QUIC hatte lediglich HTTP transportiert - in der Praxis wurden HTTP/2 Frames mithilfe des HTTP/2 Frame Syntax transportiert.

Es wurde auch angegeben, dass IETF-QUIC seine Verschlüsselung und Sicherheit auf TLS 1.3 anstelle des für Google-QUIC eigens entwickelten Ansatzes stützen sollte.

Um die Anforderung zu erfüllen, mehr als nur HTTP zu transportieren, wurde die IETF-QUIC-Protokollarchitektur in zwei separate Schichten aufgeteilt: die Transport-QUIC-Schicht und die "HTTP-over-QUIC"-Schicht (letztere wird manchmal als "hq" bezeichnet).
Diese Teilung in Schichten - auch wenn sie harmlos klingt - hat dazu geführt, dass sich IETF-QUIC erheblich vom ursprünglichen Google-QUIC unterscheidet.

Die Arbeitsgruppe hat jedoch bald entschieden, dass sie sich auf die Bereitstellung von HTTP konzentriert, um eine rechtzeitige Auslieferung der ersten QUIC Version gewährleisten zu können. Damit wurde die Implementierung des Nicht-HTTP-Transports auf später verschoben.

Als wir im März 2018 mit der Arbeit an diesem Buch begonnen haben, war geplant, die endgültige Spezifikation für die erste QUIC Version im November 2018 auszuliefern. Dies wurde später auf Juli 2019 verschoben.

Während die Arbeit an IETF-QUIC vorangeschritten ist, hat das Google-Team Details aus der IETF-Version übernommen und langsam damit begonnen, ihre Version des Protokolls dahingehend weiterzuentwickeln, dass diese wie die IETF-Version aussieht. Google hat die eigene QUIC-Version weiterhin in ihrem Browser und ihren Diensten im Einsatz.

[Die meisten neuen Implementierungen, die sich aktuell in Entwicklung befinden,](https://github.com/quicwg/base-drafts/wiki/Implementations) haben beschlossen, sich auf die IETF-Version zu konzentrieren und sind nicht mit der Google-Version kompatibel.

Erfahrungen von HTTP/2

Die HTTP/2 Spezifikation RFC 7540 wurde im Mai 2015 veröffentlicht, nur einen Monat bevor QUIC zum ersten Mal bei der IETF eingebracht wurde.

Mit HTTP/2 wurde die Grundlage für zukünftige Änderungen von HTTP gelegt. Die Arbeitsgruppe, die HTTP/2 kreiert hatte, war der Ansicht, dass diese Grundlage Iterationen zu neuen HTTP Versionen beschleunigen wird - viel schneller als der Wechsel von Version 1 zu Version 2 (ca. 16 Jahre).

Mit HTTP/2 haben Nutzer und Software-Stacks erkannt, dass es nicht mehr zeitgemäß ist, HTTP mit einem textbasierten Protokoll seriell abzuwickeln.

HTTP-over-QUIC wurde im November 2018 in HTTP/3 umbenannt.

Status

Die QUIC-Arbeitsgruppe hat seit Ende 2016 intensiv an der Festlegung der Protokolle gearbeitet. Es ist geplant, die Spezifikation bis Juli 2019 zu finalisieren.

Bis November 2018 gab es noch keine größeren Interoperabilitätstests mit HTTP/3 - nur mit den beiden vorhandenen Implementierungen, nicht aber mit einem Browser oder einer beliebten Open-Server Lösungen.

In den Wiki-Seiten der QUIC-Arbeitsgruppe werden rund fünfzehn verschiedene QUIC Implementierungen

aufgelistet - jedoch sind bei weitem nicht alle dieser Implementierungen mit den letzten Änderungen der Entwurfsspezifikation kompatibel.
QUIC zu implementieren ist nicht einfach. Das Protokoll hat sich bis zu diesem Zeitpunkt ständig weiterentwickelt und verändert.

Server

Es wurden keine öffentlichen Erklärungen hinsichtlich der Unterstützung von QUIC durch Apache oder Nginx abgegeben.

Clients

Keiner der größeren Browser-Anbieter hat bisher eine Version ausgeliefert, mit der die IETF-Version von QUIC oder HTTP/3 ausgeführt werden kann.

Google Chrome wird seit vielen Jahren mit einer funktionsfähigen Implementierung von Googles eigener QUIC-Version ausgeliefert, die jedoch nicht mit dem IETF-QUIC-Protokoll kompatibel ist und deren HTTP-Implementierung sich von HTTP/3 unterscheidet.

Implementierungshindernisse

QUIC hat sich dazu entschlossen, auf TLS 1.3 als Grundlage der Krypto- und Sicherheitsebene zu setzen, um sich auf ein zuverlässiges und bestehendes Protokoll stützen zu können. Jedoch hat sich die Arbeitsgruppe auch dazu entschieden, TLS in QUIC wirkungsvoller zu gestalten: es sollen lediglich "TLS messages" und nicht "TLS records" genutzt werden.

Dies mag nach einer harmlosen Änderung klingen, hat jedoch für viele QUIC-Stack-Implementierer eine erhebliche Hürde bedeutet. Bestehende TLS-Bibliotheken, die TLS 1.3 unterstützen, verfügen einfach nicht über ausreichende APIs, um diese Funktionalität verfügbar zu machen und QUIC den Zugriff darauf zu ermöglichen. Während mehrere QUIC-Implementierer von größeren Organisationen stammen, die parallel an ihrem eigenen TLS-Stack arbeiten, gilt dies nicht für alle.

Das dominierende Open-Source-Schwergewicht OpenSSL zum Beispiel, hat keine API dafür - und hat bisher auch keinen Wunsch geäußert, eine solche bald zur Verfügung zu stellen (Stand November 2018).

Dies wird letztendlich auch zu Hindernissen bei der Bereitstellung führen, da sich QUIC-Stacks entweder auf andere TLS-Bibliotheken stützen müssen, einen separat gepatchten OpenSSL-Build verwenden müssen oder ein Update auf eine zukünftige OpenSSL-Version benötigen.

Kernel und CPU-Last

Sowohl Google als auch Facebook haben angemerkt, dass die Bereitstellung von QUIC in großem Umfang ungefähr die doppelte Menge an CPU-Rechenleistung erfordert als wenn die gleiche Traffic-Last via HTTP/2 über TLS abgewickelt würde.

Einige Erklärungen hierfür sind:

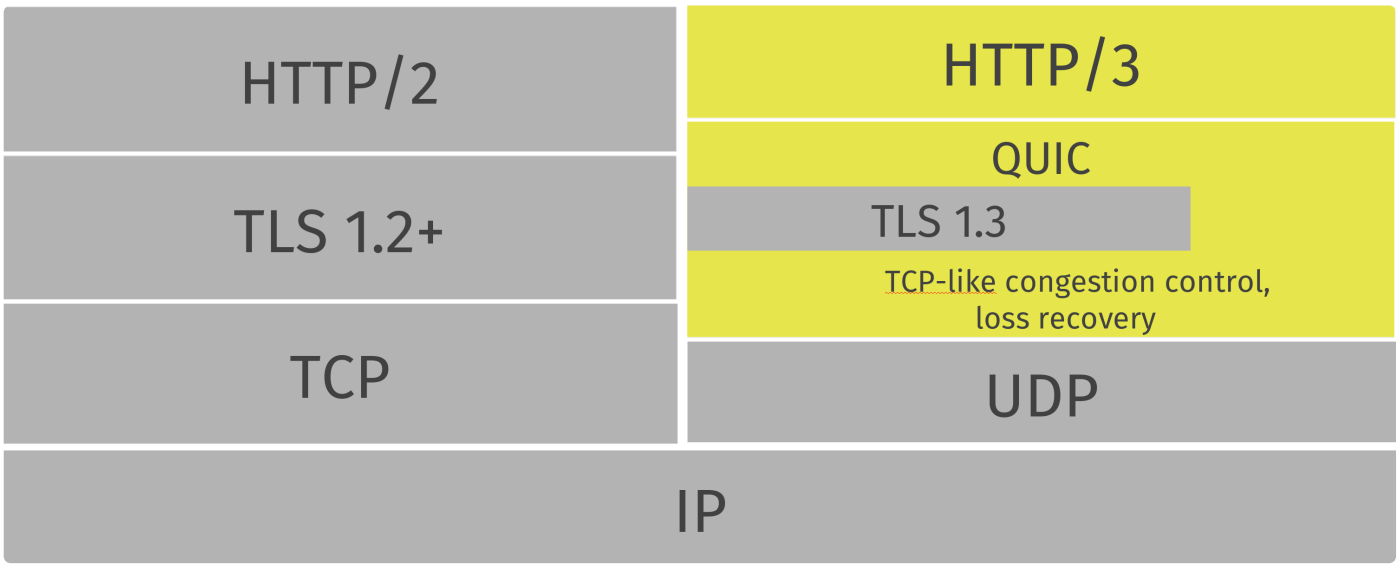
- Teile von UDP sind vor allem unter Linux nicht so optimiert wie der TCP-Stack, da UDP traditionell nicht für solche Hochgeschwindigkeitsübertragungen verwendet wurde.
- TCP- und TLS-Offloading auf Hardware ist vorhanden, bei UDP jedoch viel seltener und bei QUIC im Grunde nicht vorhanden.

Daher gibt es Gründe zur Annahme, dass sich die Leistung und die CPU-Anforderungen im Laufe der Zeit verbessern werden.

Eigenschaften des Protokolls

Das QUIC-Protokoll aus der Ferne betrachtet.

Folgend ist der HTTP/2-Netzwerkstack links und der QUIC-Netzwerkstack rechts dargestellt, wenn als HTTP-Transport verwendet.



QUIC Logo

UDP

Transportprotokoll über UDP

QUIC ist ein Transportprotokoll, das basierend auf UDP implementiert wurde. Wenn Sie Ihren Netzwerkverkehr gelegentlich beobachten, wird QUIC als UDP-Pakete angezeigt.

Basierend auf UDP werden dann auch UDP-Portnummern verwendet, um bestimmte Netzwerkdienste unter

einer bestimmten IP-Adresse zu identifizieren.
Alle bekannten QUIC-Implementierungen befinden sich derzeit im User-Space, was eine schnellere Entwicklung ermöglicht, als dies bei Kernel-Space-Implementierungen normalerweise möglich ist.

Wird es funktionieren?

Es gibt Unternehmen und andere Netzwerkkonfigurationen, die den UDP-Verkehr an anderen Ports als 53 blockieren (für DNS verwendet). Andere drosseln solche Daten auf eine Weise, die die Leistung von QUIC schlechter macht als jene von TCP-basierten Protokollen. Es gibt kein Ende für das, was manche Betreiber tun könnten.

Auf absehbare Zeit muss jeder Einsatz von QUIC-basierten Transporten möglicherweise auf eine andere (TCP-basierte) Alternative zurückgreifen können. Google-Ingenieure haben zuvor gemessene Fehlerraten im niedrigen einstelligen Prozentbereich angegeben.

Wird es besser?

Wenn sich QUIC als eine wertvolle Bereicherung für das Internet herausstellt, möchten Leute es nutzen und das es in ihren Netzwerken funktioniert - dann können Unternehmen damit beginnen, ihre Hindernisse zu überdenken. Im Laufe der Jahre hat die Entwicklung von QUIC Fortschritte gemacht und die Erfolgsquote beim Aufbau sowie Nutzung von QUIC-Verbindungen über das Internet ist gestiegen.

Zuverlässige Datenübertragung

Während UDP kein zuverlässiger Transport ist, fügt QUIC über UDP eine Schicht hinzu, die Zuverlässigkeit einführt. Es bietet Neuübertragungen von Paketen, Überlastungskontrolle, Pacing und andere Funktionen, die sonst in TCP vorhanden sind.

Daten, die von einem Endpunkt über QUIC gesendet werden, erscheinen früher oder später am anderen Ende - solange die Verbindung besteht.

Streams

Ähnlich wie SCTP, SSH und HTTP/2 verfügt QUIC über separate logische Streams innerhalb von physischen Verbindungen. Eine Anzahl paralleler Streams, die Daten gleichzeitig über eine einzelne Verbindung übertragen können, ohne die anderen Streams zu beeinträchtigen.

Eine Verbindung ist ein ausgehandelter Aufbau zwischen zwei Endpunkten, die ähnlich wie eine TCP-Verbindung funktioniert. Eine QUIC-Verbindung wird zu einem UDP-Port und einer IP-Adresse hergestellt, aber sobald die Verbindung hergestellt ist, wird sie durch ihre "Verbindungs-ID" verknüpft.

Über eine bestehende Verbindung kann jede Seite Streams erstellen und Daten an das andere Ende senden. Streams werden in richtiger Reihenfolge (in-order) und zuverlässig geliefert - andere Streams können aber in nicht richtiger Reihenfolge (out-of-order) geliefert werden.
QUIC bietet eine Datenflusssteuerung sowohl für Verbindungen als auch für Streams.

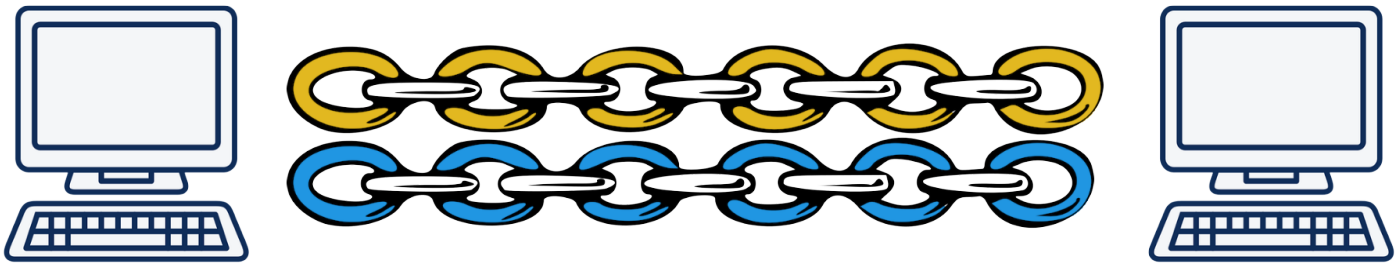
Weitere Details findest du in den Bereichen Verbindungen und Streams.

In-Order Ausliefung

QUIC garantiert die richtige Reihenfolge (in-order) der Zustellung von Streams, jedoch nicht zwischen Streams. Dies bedeutet, dass jeder Stream Daten sendet und die Datenreihenfolge beibehält, aber die Streams das Ziel möglicherweise in einer anderen Reihenfolge erreichen, als wie die Anwendung diese gesendet hat.

Beispiel: Stream A und B werden von einem Server zu einem Client übertragen. Zuerst wird Stream A und dann Stream B gestartet. In QUIC wirkt sich ein verlorenes Paket nur auf jenen Stream aus, zu dem das verlorene Paket gehört. Wenn Stream A ein Paket verliert, Stream B jedoch nicht, setzt Stream B die Übertragung möglicherweise fort und wird abgeschlossen, während das verlorene Paket von Stream A erneut übertragen wird. Dies war unter HTTP/2 nicht möglich.

Hier dargestellt mit einem gelben und einem blauen Stream, die über eine einzige Verbindung zwischen zwei QUIC-Endpunkten gesendet werden. Sie sind unabhängig und können in einer anderen Reihenfolge eintreffen, aber jeder Stream wird zuverlässig und in der richtigen Reihenfolge (in-order) an die Anwendung geliefert.



zwei QUIC-Streams zwischen zwei Computern

Schnelle Handshakes

QUIC bietet einen Verbindungsaufbau sowohl mit 0-RTT als auch 1-RTT. Das bedeutet, dass QUIC beim Aufbau einer neuen Verbindung bestenfalls keine zusätzlichen Roundtrips benötigt. Der Schnellere von beiden, der 0-RTT-Handshake, funktioniert nur, wenn zuvor eine Verbindung zu einem Host hergestellt und ein Geheimnis dieser Verbindung zwischengespeichert wurde.

Früher Daten versenden

Mit QUIC kann ein Client Daten bereits zu 0-RTT-Handshakes hinzufügen. Diese Funktion ermöglicht es einem Client, Daten so schnell wie möglich an den Peer zu übermitteln. Auf diese Weise kann der Server natürlich noch früher antworten und Daten zurücksenden.

TLS 1.3

QUIC verwendete TLS 1.3 (RFC 8446) zur Transportsicherheit und es gibt unter keinen Umständen unverschlüsselte QUIC-Verbindungen.

TLS 1.3 hat einige Vorteile gegenüber älteren TLS-Versionen - ein Hauptgrund für die Verwendung in QUIC ist, dass 1.3 den Handshake so geändert hat, dass weniger Roundtrips erforderlich sind. Dieses Vorgehen reduziert die Protokolllatenz.

In der Google-Version von QUIC wurde eine eigens entwickelte Lösung verwendet.

Transport und Anwendung

Das IETF-QUIC-Protokoll ist ein Transportprotokoll, über welches andere Anwendungsprotokolle genutzt werden können. Das erste Protokoll der Anwendungsschicht ist HTTP/3 (h3).

Die Transportschicht unterstützt Verbindungen und Streams.

In der Google-Version von QUIC waren Transport und HTTP zu einem Allheilmittel gebündelt worden und war eher ein spezielles Sende-http/2-Frames-über-UDP-Protokoll.

HTTP/3 über QUIC

Die HTTP-Schicht namens HTTP/3 führt Transporte im HTTP-Stil durch, einschließlich HTTP-Header-Komprimierung mit QPACK - ähnlich der HTTP/2-Komprimierung HPACK.

Der HPACK-Algorithmus hängt von einer *geordneten* Übermittlung von Streams ab, weshalb es nicht möglich war, ihn ohne Änderungen in HTTP/3 wiederzuverwenden; vor allem, weil QUIC Streams anbietet, die nicht in der richtigen Reihenfolge übermittelt werden können. QPACK kann als die QUIC-angepasste Version von HPACK gesehen werden.

Nicht-HTTP über QUIC

Die Arbeit am Senden anderer Protokolle als HTTP über QUIC wurde solange verschoben, bis die erste Version von QUIC ausgeliefert wurde.

Wie QUIC funktioniert

In diesem Abschnitt wird beschrieben, wie die grundlegenden Bausteine des QUIC-Transportprotokolls funktionieren, ohne dabei auf alle Bits und Bytes einzugehen. Wenn du deinen eigenen QUIC-Stack implementieren möchtest, sollte dir diese Beschreibung ein allgemeines Verständnis vermitteln. Alle Details findest du in den aktuellen IETF Internet-Entwürfen und RFCs.

1. Eine Verbindung aufbauen
2. ... die TLS inkludiert
3. und dann Streams verwendet

Verbindungen

Eine QUIC-Verbindung ist eine einzelne Konversation zwischen zwei QUIC-Endpunkten. Der Verbindungsaufbau von QUIC kombiniert die Versionsaushandlung mit den kryptografischen und Transport-Handshakes, um die Latenz beim Verbindungsaufbau zu verringern.

Um tatsächlich Daten über eine solche Verbindung zu senden, müssen ein oder mehrere Streams erstellt und verwendet werden.

Verbindungs-ID

Jede Verbindung verfügt über eine Reihe von Verbindungskennungen oder Verbindungs-IDs, von denen jede zur Identifizierung der Verbindung verwendet werden kann. Verbindungs-IDs werden unabhängig von den Endpunkten ausgewählt; jeder Endpunkt wählt die Verbindungs-IDs aus, die sein Peer verwendet.

Die Hauptfunktion dieser Verbindungs-IDs besteht darin, sicherzustellen, dass Änderungen in der Adressierung auf niedrigeren Protokollschichten (UDP, IP und darunter) nicht dazu führen, dass Pakete einer QUIC-Verbindung an den falschen Endpunkt gesendet werden.

Durch die Nutzung der Verbindungs-ID können Verbindungen auf eine Weise zwischen IP-Adressen und Netzwerkschnittstellen migrieren, wie dies TCP niemals könnte. Durch die Migration kann beispielsweise ein laufender Download von einer Mobilfunkverbindung zu einer schnelleren WLAN-Verbindung geändert werden, wenn der Benutzer sein Gerät an einen Standort mit WLAN-Verbindung bringt. Ebenso kann der Download über die Mobilfunkverbindung erfolgen, wenn das WLAN nicht mehr verfügbar ist.

Portnummern

QUIC ist auf UDP aufgebaut, daher wird ein 16-Bit-Portnummernfeld verwendet, um eingehende Verbindungen zu unterscheiden.

Versionsaushandlung

Eine von einem Client stammende QUIC-Verbindungsanforderung teilt dem Server mit, welche QUIC-Protokollversion er sprechen möchte. Der Server antwortet mit einer Liste der unterstützten Versionen, aus denen der Client auswählen kann.

Verbindungen verwenden TLS

Unmittelbar nach dem ersten Aufbau einer Verbindung sendet der Initiator einen Krypto-Frame, der mit dem Einrichten des Handshakes der Sicherheitsschicht beginnt. Die Sicherheitsschicht verwendet TLS 1.3.

Es gibt keine Möglichkeit, TLS bei eine QUIC-Verbindung zu deaktivieren oder zu vermeiden. Das Protokoll ist so konzipiert, dass es für Middleboxen schwer zu manipulieren ist, um eine Ossifikation des Protokolls zu verhindern.

Streams

QUIC Streams bieten eine einfache, geordnete Byte-Stream-Abstraktion.

In QUIC gibt es zwei grundlegende Arten von Streams:

- Unidirektionale Streams übertragen Daten in eine Richtung: vom Initiator des Streams zu seinem Peer.
- Bidirektionale Streams ermöglichen das Senden von Daten in beide Richtungen.

Jeder Stream-Typ kann von jedem Endpunkt erstellt werden, gleichzeitig mit anderen Streams verschachtelte Daten senden und abgebrochen werden.

Um Daten über eine QUIC-Verbindung zu senden, werden ein oder mehrere Streams verwendet.

Ablaufsteuerung

Streams werden individuell ablaufgesteuert, sodass ein Endpunkt die Speicherbindung begrenzen und Gegendruck ausüben kann. Die Erstellung von Streams wird ebenfalls ablaufgesteuert, wobei jeder Peer die maximale Stream-ID angibt, die er zu einem bestimmten Zeitpunkt akzeptieren möchte.

Stream Identifikatoren

Streams werden durch eine vorzeichenlose 62-Bit-Ganzzahl identifiziert, die als Stream-ID bezeichnet wird. Die niedrigstwertigen zwei Bits der Stream-ID werden verwendet, um den Stream-Typ (unidirektional oder bidirektional) und den Initiator des Streams zu identifizieren.

Das niedrigstwertige Bit (0x1) der Stream-ID identifiziert den Initiator des Streams. Clients initiieren

geradzahlige Streams (solche mit dem niedrigstwertigen Bit auf 0); Server initiieren ungeradzahlige Streams (wobei das Bit auf 1 gesetzt ist).

Das zweitniedrigste Bit (0x2) der Stream-ID unterscheidet zwischen unidirektionalen und bidirektionalen Streams. Bei unidirektionalen Streams ist dieses Bit immer auf 1 gesetzt, und bei bidirektionalen Streams ist dieses Bit auf 0 gesetzt.

Gleichzeitigkeit von Streams

Mit QUIC kann eine beliebige Anzahl von Streams gleichzeitig betrieben werden. Ein Endpunkt begrenzt die Anzahl der gleichzeitig aktiven eingehenden Streams, indem die maximale Stream-ID begrenzt wird.

Die maximale Stream-ID ist für jeden Endpunkt spezifisch und gilt nur für den Peer, der die Einstellung erhält.

Daten senden und empfangen

Endpunkte verwenden Streams zum Senden und Empfangen von Daten - das ist immerhin ihr letztendlicher Zweck. Streams sind eine geordnete Byte-Stream-Abstraktion. Separate Streams werden jedoch nicht unbedingt in der ursprünglichen Reihung ausgeliefert.

Stream Priorisierung

Stream-Multiplexing hat erhebliche Auswirkungen auf die Anwendungsleistung, wenn den Streams zugewiesene Ressourcen korrekt priorisiert werden. Die Erfahrung mit anderen Multiplex-Protokollen - wie HTTP/2 - zeigt, dass effektive Priorisierungsstrategien einen signifikanten positiven Einfluss auf die Leistung haben.

QUIC selbst bietet keine Frames für den Austausch von Priorisierungsinformationen. Stattdessen müssen Prioritätsinformationen von der Anwendung empfangen werden, die QUIC verwendet. Protokolle, die QUIC verwenden, können jedes Priorisierungsschema definieren, das ihrer Anwendungssemantik entspricht.

Wenn HTTP/3 über QUIC verwendet wird, erfolgt die Priorisierung in der HTTP-Schicht.

0-RTT

Um die zum Herstellen einer neuen Verbindung erforderliche Zeit zu verkürzen, kann ein Client, der zuvor eine Verbindung zu einem Server hergestellt hat, bestimmte Parameter dieser Verbindung zwischenspeichern und anschließend eine **0-RTT**-Verbindung mit dem Server herstellen. Auf diese Weise kann der Client Daten sofort senden, ohne auf den Abschluss eines Handshakes warten zu müssen.

Spin Bit

Eine der vielleicht längsten Designdiskussionen innerhalb der QUIC-Arbeitsgruppe - Gegenstand von mehreren hundert Mails und stundenlangen Diskussionen - betrifft ein einziges Bit: das Spin Bit.

Die Befürworter dieses Bits bestehen darauf, dass Operatoren und Personen zwischen zwei QUIC-Endpunkten die Latenz messen können.

Die Gegner dieser Funktion mögen das potenzielle Informationsleck nicht.

Ein Bit drehen

Beide Endpunkte, der Client und der Server, behalten für jede QUIC-Verbindung den Spin-Wert 0 oder 1 bei und setzen das Spin Bit für Pakete, die für diese Verbindung gesendet werden, auf den entsprechenden Wert.

Beide Seiten senden dann Pakete aus, bei denen das Spin Bit auf den gleichen Wert gesetzt ist solange ein Roundtrip dauert, und schalten dann den Wert um. Der Effekt ist dann ein Impuls von Einsen und Nullen in dem Bitfeld, das Beobachter überwachen können.

Diese Messung funktioniert nur, wenn der Absender weder auf die Anwendung noch auf die Flusskontrolle beschränkt ist. Die Neuordnung von Paketen über das Netzwerk kann auch zu unscharfen Daten führen.

User-space

Die Implementierung eines Transportprotokolls im User-space ermöglicht eine schnelle Iteration des Protokolls, da das Protokoll vergleichsweise einfach weiterentwickelt werden kann, ohne dass Clients und Server ihren Betriebssystemkern aktualisieren müssen, um neue Versionen bereitzustellen.

QUIC gibt nichts vor, dass eine zukünftige Implementierung in Betriebssystemkernen verhindern könnte - sollte jemand dies für eine gute Idee halten.

Viele Implementierungen

Ein offensichtlicher Effekt der Implementierung eines neuen Transportprotokolls im User-space besteht darin, dass wir viele unabhängige Implementierungen erwarten können.

Verschiedene Anwendungen werden wahrscheinlich auf absehbare Zeit unterschiedliche HTTP/3- und QUIC-Implementierungen inkludieren, oder auf solchen aufbauen.

API

Einer der Erfolgsfaktoren von regulärem TCP und Programmen, die dieses verwenden, ist die standardisierte Socket-API. Sie verfügt über gut definierte Funktionen und ermöglicht ein Verschieben von Programmen zwischen vielen verschiedenen Betriebssystemen, da TCP überall gleich funktioniert.

QUIC ist nicht so weit. Es gibt keine Standard-API für QUIC.

Bei QUIC muss eine der vorhandenen Implementierungen ausgewählt werden, bei deren API man bleiben muss. Dadurch werden Anwendungen bis zu einem gewissen Grad an eine einzelne Bibliothek "gebunden". Das Wechseln zu einer anderen Bibliothek bedeutet eine andere API und kann viel Arbeit erfordern.

Da QUIC normalerweise im User-space implementiert wird, kann es die Socket-API nicht einfach erweitern oder der vorhandenen TCP- und UDP-Funktionalität ähneln. Die Verwendung von QUIC bedeutet die Verwendung einer anderen API als der Socket-API.

فارسی

نوشتن این کتاب در ماه مارس سال ۲۰۱۸ آغاز شده و هدف مستندسازی HTTP/3 و پروتکل آن است: QUIC. چرایی‌ها، نحوهٔ کارکرد، جزئیات پروتکل، راه‌اندازی‌ها و غیره.

این کتاب به‌طور کامل رایگان و مبتنی بر همکاری است، بدین صورت که هر کسی مایل باشد می‌تواند یاری رساند.

پیش‌نیازها

از خوانندهٔ این کتاب انتظار می‌رود درک پایه از شبکهٔ TCP/IP، اصول اولیهٔ HTTP و وِب داشته باشد. برای درک و بدست آوردن دیدی بهتر در خصوص HTTP/2 مطالعهٔ http2 explained پیشنهاد می‌شود.

نویسنده

این کتاب توسط دنیل استنبرگ تهیه و آغاز شده است. دنیل پایه‌گذار و توسعه‌دهندهٔ ارشد curl، رایج‌ترین ابزارِ استفاده شده به‌عنوان HTTP client است. دنیل بیش از دو دهه با، و بر روی، پروتکل‌های HTTP و Internet کار کرده و همچنین نویسندهٔ کتاب http2 explained است.

خانه

صفحهٔ اصلی این کتاب در نشانی daniel.haxx.se/http3-explained قابل دسترس است.

یاری‌رسانی

در صورت مشاهدهٔ هر گونه اشتباه، سهو و از قلم‌افتادگی، مشکل و یا موردی در نحوهٔ نوشتار و یا نگارش، خواهشمند است نسخه‌ای اصلاح‌شده از بند مربوطه را برای ما ارسال کنید تا آن را اعمال و به‌روز کنیم. همچنین از هرکس که یاری رساند به‌صورت رسمی تقدیر خواهد شد. به امید بهتر کردن و کامل‌تر ساختن این دبیزه.

ترجیحاً، یا مشکل مربوطه را در errors و یا به‌عنوان pull requests بر روی صفحهٔ GitHub کتاب ثبت کنید.

پروانه

چرا QUIC

واژهٔ QUIC یک اسم است، سرنام نیست. این واژه دقیقاً مانند واژهٔ انگلیسی «quick» تلفظ می‌شود.

پروتکل QUIC یک پروتکل انتقالِ دادهٔ امن و قابل اعتماد جدید است که برای پروتکلی مانند HTTP مناسب بوده و می‌تواند برخی از کاستی‌های استفاده از HTTP/2 بر روی TCP و TLS را برطرف کند. گام منطقی بعدی در تکامل حمل و نقل وب.

پروتکل QUIC تنها به انتقال HTTP محدود نمی‌شود. تمایل به سرعت بخشیدن به وب و تحویل داده‌ها به کاربران احتمالاً بزرگترین انگیزه و دلیلی است که در ابتدا باعث ایجاد این پروتکل حمل و نقل جدید شد.

پس چرا ایجاد یک پروتکل جدید و چرا بر روی بستر UDP؟

QUIC

QUIC logo

آیا HTTP/2 را به یاد دارید؟

مشخصات فنی پروتکل RFC7540 ،HTTP/2، در ماه مِه سال ۲۰۱۵ منتشر شد و پروتکل HTTP/2 از آن زمان به صورت گسترده در سراسر اینترنت و شبکه جهانیِ وِب راه‌اندازی و مورد استفاده قرار گرفته است.

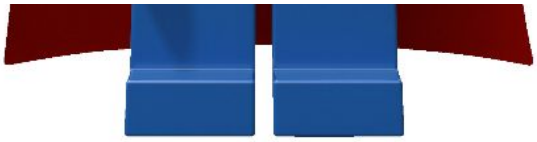
در اوایل سال ۲۰۱۸، حدود ۴۰٪ از ۱۰۰۰ وب سایت برتر از HTTP/2 استفاده می‌کردند، حدود ۷۰٪ از تمام درخواست‌های HTTPS ارسال شده توسط Firefox پاسخ‌های HTTP/2 دریافت کرده‌اند و تمام مرورگرهای اصلی، سرورها و پروکسی‌ها این پروتکل را پشتیبانی می‌کنند.

پروتکل HTTP/2 بسیاری از مشکلات موجود در پروتکل HTTP/1 را برطرف می‌کند و با ارائه نسخه دوم پروتکل HTTP کاربران قادر به کنارگذاشتن راه‌کارهای موقت بسیاری هستند که برخی از آن راه‌کارهای موقت برای توسعه دهندگان وب بسیار دشوار است.

یکی از ویژگی‌های اصلی پروتکل HTTP/2 استفاده از روش تسهیم است، روش تسهیم این‌گونه است که بسیاری از جریان‌های منطقی بر روی یک ارتباط فیزیکیِ TCP ارسال می‌شوند. این روش باعث تسریع و بهبود در خیلی از مسائل می‌شود. این روش باعث بهبود کنترل ازدحام، استفاده بهتر کاربر از ارتباط TCP و در نتیجه استفاده بهینه از پهنای باند، و همچنین افزایش تداوم ارتباطات TCP که نتیجه آن استفاده‌ی کامل از سرعت به نسبت قبل است می‌شود. فشرده‌سازی سرآیندها نیز منجر به استفاده‌ی کمتر از پهنای باند می‌شود.

با استفاده از HTTP/2 مرورگرها معمولاً از یک ارتباط TCP با هر میزبان به جای شش ارتباط که در گذشته رایج بود، استفاده می‌کنند. در واقع، تکنیک‌هایی مانند 'desharding' و 'coalescing' که همراه HTTP/2 استفاده می‌شود، می‌تواند به مراتب تعداد ارتباطات را کاهش دهد.

همچنین HTTP/2 مشکل 'Head-of-line (HOL) blocking' که باعث می‌شد تا کاربر مجبور به انتظار برای اتمام درخواست اولِ در صف پیش از ارسال درخواست‌های بعدی باشد را حل کرد.

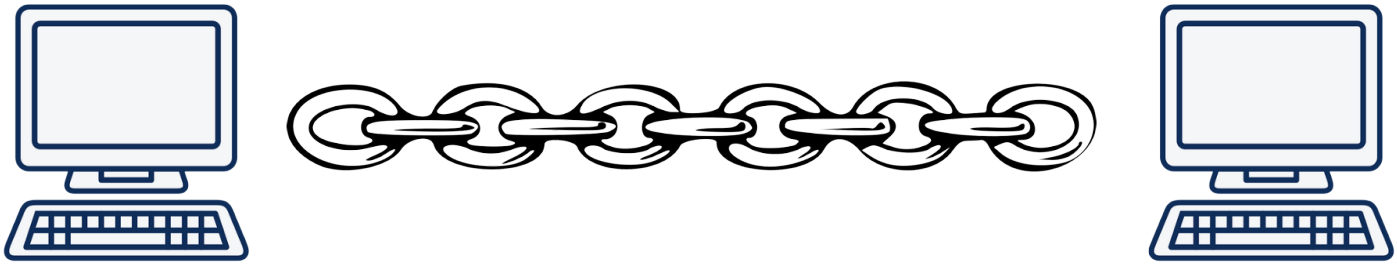


http2 man

مسدود کننده سرِ صف TCP

مسدود کننده سرِ صف TCP

ارتباطات HTTP/2 بر روی بستر TCP بنا شده و به نسبت نُسَخ قبلی از تعداد اتصالات بسیار کمتری استفاده می‌شود. TCP پروتکلی برای انتقالات قابل اعتماد است و می‌توان آن را اساسا زنجیره‌ای خیالی میان دو ماشین در نظر گرفت. آنچه در انتهایی از شبکه قرار داده شده است به همان ترتیب به انتهای دیگر در شبکه می‌رسد - در نهایت (وگرنه ارتباط قطع می‌شود).

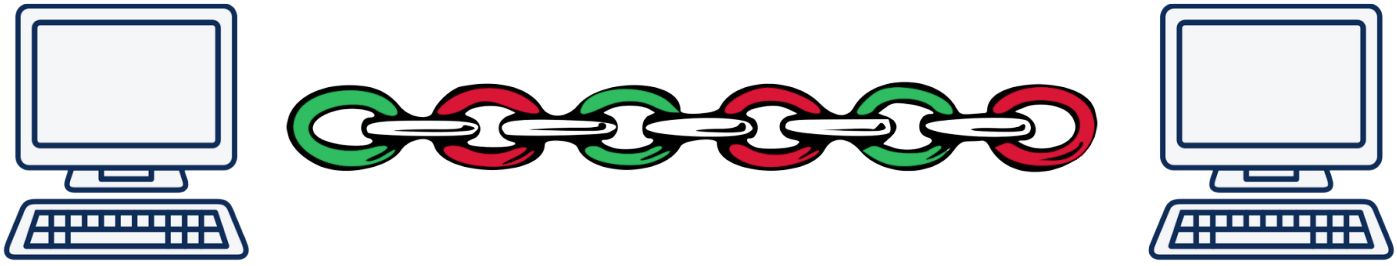


a TCP chain between two computers

به کمک HTTP/2، مرورگرهای عادی ده‌ها و صدها انتقال موازی را بر روی یک اتصال TCP انجام می‌دهند.

در صورتی که یک بسته از بین رفته باشد، و یا جایی در شبکه بین دو نقطه که توسط HTTP/2 صحبت می‌کنند گم شده باشد، به این معناست که کل اتصال TCP متوقف می‌شود تا زمانی که بسته‌ی گم شده دوباره ارسال شود و راه خود را به سمت مقصد نهایی پیدا کند. از آنجایی که TCP این 'زنجیره' است، به این معنی است که اگر یک پیوند به طور ناگهانی از دست رفته باشد، همه چیز پس از آن پیوندِ از دست رفته باید در حالت انتظار باقی بماند.

یک تصویر با استفاده از استعاره زنجیره‌ای هنگام ارسال دو جریان بر روی این اتصال. جریان قرمز و جریان سبز:



the chain showing links in different colors

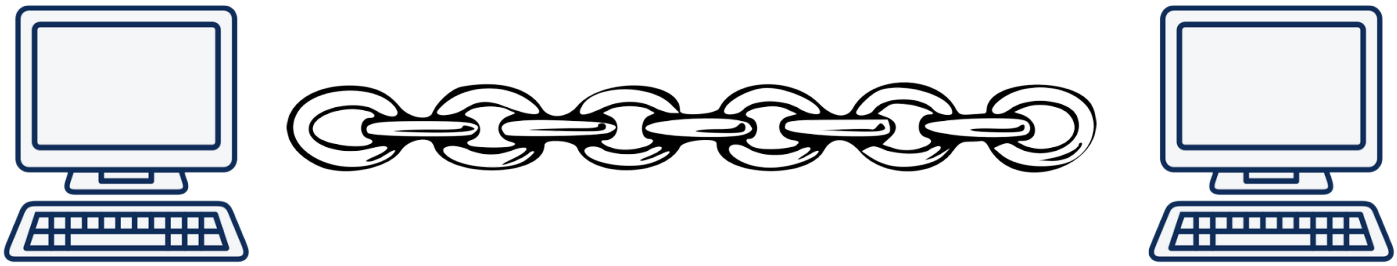
این یک head-of-line block یا به اختصار HOL block مبتنی بر TCP (مسدود کننده سرِ صفِ TCP) می‌شود.

با افزایش درصد از دست رفتن بسته ها، HTTP/2 ضعیف و ضعیف‌تر عمل می‌کند. در شرایطی با ۲٪

packet loss (که نشان‌دهنده‌ی کیفیت بسیار پایین و وحشتناکی از شبکه است) آزمایش ثابت کرده است که کاربران HTTP/1 معمولاً از عملکرد بهتری برخوردار خواهند بود - چرا که آنان ۶ اتصال برای توزیع بسته‌های گم شده دارند. که البته همچنین به این معنی است که در صورت از دست رفتن بسته‌ای، اتصال دیگر می‌تواند همچنان به کار خود ادامه دهد.
حل این مشکل با TCP اصلاً ساده نخواهد بود، البته اگر ممکن باشد.

جریان‌های مستقل از انسداد جلوگیری می‌کنند

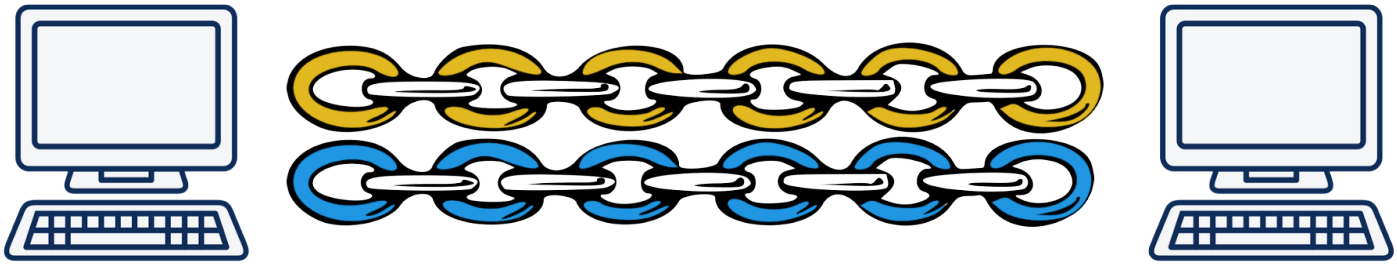
به کمک QUIC همچنان اتصالی بین دو نقطه وجود دارد که ارتباط را ایمن و ارسال اطلاعات را معتبر و قابل اطمینان می‌سازد.



a QUIC chain between two computers

زمانی که دو جریان مختلف بر روی این اتصال ایجاد شود، آن دو به صورت مجزا دیده می‌شوند، به گونه‌ای که اگر پیوندی برای یکی از دو جریان از دست رفته باشد، تنها آن جریان، آن زنجیره‌ی مشخص، باید منتظر بماند تا پیوند گم شده دوباره برای انتقال ارسال شود.

در اینجا به واسطه‌ی تصویری با یک جریان زرد و یک جریان آبیِ ارسال شده بین دو نقطه توضیح داده شده است.



two QUIC streams between two computers

TCP یا UDP

TCP یا UDP

اگر نتوانیم مشکلِ مسدودکننده سرِ صف را در TCP حل کنیم، در اصول نظری باید قادر به تولید یک لایه انتقال جدید در کنار UDP و TCP در پُشته‌ی شبکه باشیم. و یا حتی از SCTP استفاده کنیم که یک پروتکل انتقالِ استاندارد شده توسط IEEE در RFC 4960 و دارای چندی از مشخصات مورد نیاز است.

هر چند، در سالیان اخیر تلاش‌های موبوط به ساخت و تولید لایه انتقال جدید به دلیل سختی کار برای قرار دادن آن بر روی بستر اینترنت تقریباً به صورت کامل متوقف شده است. استفاده‌ی پروتکل‌های جدید توسط بسیاری از firewall ها، NAT ها، router ها و بسیاری از تجهیزات میانی‌ای که برای ارتباط بین کاربر و سرور عقب داشته و مختل شده است. معرفی یک پروتکل انتقالِ دیگر باعث رد شدن N% از اتصالات خواهد شد، چرا که آنان توسط تجهیزات میانی به دلیل ناشناس بودن و در نتیجه تصور به مخرب و یا اشتباه بودنشان رد خواهند شد. و این N% خطا گاهی بسیار بالاتر از آنی که ارزش پیاده‌سازی و متحمل شدن را داشته باشد تلقی می‌شود.
همچنین، تغییر مسائل در لایه‌ی پروتکل انتقالِ پشته‌ی شبکه معمولاً به معنی پروتوکل‌های راه‌اندازی شده در هسته های سیستم عامل (kernel) است. به روز رسانی و به‌کارگیری هسته های سیستم عامل روندی آهسته‌ست که مستلزم تلاش و تغییرات قابل توجهی است. بسیاری از بهسازی‌های TCP استاندارد شده توسط IEEE به دلیل عدم پشتیبانی رایج به صورت گسترده استفاده و به کار گرفته نشده‌اند.

چرا SCTP-over-UDP نه

پروتکل انتقال کنترل جریان (SCTP) یک پروتکل لایه انتقال قابل‌اعتماد از جریان‌ها تست، و همچنین برای WebRTC هم پیاده‌سازی‌هایی از این پروتکل بر روی UDP موجود است.

به دلایل متعددی این روش به خوبی QUIC نبود، از جلمه:

- پروتکل انتقال کنترل جریان (SCTP) مشکل مسدود کننده‌ی سر صف را برای جریان‌ها حل نمی‌کند
- پروتکل انتقال کنترل جریان (SCTP) برای مرحله‌ی راه‌اندازی نیازمند معین شدن تعداد جریان‌ها است
- پروتکل انتقال کنترل جریان (SCTP) سابقه محکمی از TLS و امنیت ندارد
- پروتکل انتقال کنترل جریان (SCTP) مبتنی بر دست‌دهی چهار مرحله‌ای است در حالی که QUIC روش RTT-0 را ارائه می‌دهد
- پروتکل QUIC همانند TCP یک bytestream است، در حالی که SCTP پروتکلی message-based است
- اتصالات QUIC می‌توانند بین آدرس‌های IP جابجا شوند در حالی که اتصالات SCTP نمی‌توانند

برای مطالعه‌ی بیشتر در خصوص تفاوت، به مقایسه‌ای بین SCTP و QUIC رجوع شود.

استخوان‌سازی (fa/Ossification)

اینترنت شبکه‌ای از شبکه‌هاست. در طول مسیر تجهیزاتی در جاهای مختلف بر روی اینترنت راه اندازی شده‌اند تا از کارکرد درست این شبکه‌ای از شبکه‌ها اطمینان حاصل کنند. این دستگاه‌ها، دستگاه‌هایی که در شبکه توزیع شده‌اند، همان‌هایی هستند که ما اغلب به عنوان دستگاه‌های میانی از آنها یاد می‌کنیم. دستگاه‌هایی که در جایی بین دو پایانه قرار گرفته‌اند و بخش‌های اصلی انتقال داده‌های شبکه مرسوم را تشکیل می‌دهند.

این دستگاه‌ها در خدمت اهداف مشخصی هستند اما من فکر می‌کنم که می‌توان گفت بطور کلی به دلیل آنکه شخصی فکر می‌کند آنها باید در آنجا باشند تا همه چیز درست کار کند آنجا قرار گرفته‌اند.

دستگاه‌های میانی بسته‌های IP را بین شبکه‌ها مسیریابی می‌کنند، آن‌ها ترافیک مخرب را مسدود می‌کنند، عمل NAT (برگردان نشانی شبکه) را انجام می‌دهند، کارایی را بهبود می‌بخشند، برخی تلاش می‌کنند تا ترافیک در حال عبور را مورد بررسی و جاسوسی قرار دهند، و بیشتر.

برای آنکه این دستگاه‌ها وظیفه خود را انجام دهند ملزم هستند تا درباره‌ی شبکه و پروتکل‌هایی که توسط آن‌ها نظارت شده و یا تغییر یافته است آگاه باشند. آنها نرم افزار را به همین هدف اجرا می‌کنند. نرم افزاری که پیاپی بروزرسانی نمی‌شود.

گرچه این‌ها اجزای متصل کننده‌ای هستند که اینترنت را در کنار هم نگاه می‌دارند معمولاً با آخرین تکنولوژی همراه نیستند. میانه‌ی شبکه معمولاً به سرعتِ لبه‌ها، کارخواه‌ها و کارگزارهای سراسر جهان حرکت نمی‌کند.

پروتکل‌هایی که این دستگاه‌ها ممکن است بخواهند بررسی کنند و ببینند تا که چه چیز درست هست و چه چیز خیر، چنین مشکلاتی را دارند: این دستگاه‌ها مدت‌ها قبل هنگامی که این پروتکل‌ها امکانات آن زمان را داشتند کار گذاشته شده‌اند. معرفی امکانات جدیدتر یا تغییر در عملکرد به گونه‌ای که قبل‌تر شناخته نشده است می‌تواند خطرِ بد و یا غیرمجاز تلقی شدن توسط چنین دستگاه‌هایی به همراه داشته باشد. چنین ترافیکی می‌تواند افت یا تأخیر داشته باشد تا درجه‌ای که کاربران به راستی دیگر نخواهند از آن امکانات استفاده کنند.

چنین مشکلی را "سختی پروتکل" و یا "استخوان‌سازی پروتکل" می‌نامند.

تغییرات در TCP نیز دچار سختی می‌شوند: برخی دستگاه‌ها مابین یک کارخواه و کارگزار موارد ناشناخته‌ی جدید TCP را تشخیص می‌دهند و از آنجا که آنها نمی‌دانند این موادر چیستند چنین اتصالاتی را مسدود می‌کنند. سیستم‌ها اگر مجاز به بررسی جزئیات پروتکل باشند، نحوه برخورد همیشگی آنها را یاد می‌گیرند و به مرور تغییر آنها ناممکن می‌شود.

تنها راهِ به‌راستی مؤثر برای "مقابله" با استخوان‌سازی، این است که ارتباطات به جهت جلوگیری از دیده شدن بیشتر محتوای در حال گذر پروتکل توسط دستگاه‌های میانی تا جای ممکن رمزگذاری شوند.

امن

پروتکل QUIC همیشه امن است. هیچ نسخهٔ متن‌آشکاری از این پروتکل وجود ندارد، پس برای مذاکرهٔ یک اتصال QUIC باید رمزنگاری و امنیت همراه با TLS 1.3 صورت بگیرد. همانطور که در بالاتر آمد، این امر مانع از استخوان‌سازی (به معنای از بین رفتن انعطاف پذیری و امکان توسعه) و همینطور دیگر موانع و رفتارهای خاص می‌شود و همچنین اطمینان حاصل می‌کند تا QUIC تمام عوامل ایمنی HTTPS که کاربران وب انتظارش را دارند دارا باشد.

تنها تعداد کمی بستهٔ دست‌دهی اولیه وجود دارد که به صورت آشکارا پیش از آغاز مذاکرهٔ پروتکل‌های رمزنگاری ارسال می‌شوند.

تأخیرِ کاهش یافته

پروتکل QUIC هم دست‌دهی RTT-0 و هم دست‌دهی RTT-1 را ارائه می‌کند که باعث کاهش زمان لازم برای

مذاکره و برقراری اتصال جدید می‌شود. مقایسه شود با دست‌دهی سه‌گانهٔ TCP.
علاوه بر آن، QUIC از همان ابتدا پشتیبانی «داده‌های اولیه» را که برای پذیرفتن داده‌های بیشتر است ارائه می‌دهد و ساده‌تر از TCP Fast Open استفاده می‌شود.

با استفاده از مفهموم جریان، یک اتصال به میزبان می‌تواند یک‌باره و بدون آنکه ابتدا نیازی به انتظار کشیدن برای پایانِ اتصالِ موجود داشته باشد صورت پذیرد.

قابلیت TCP Fast Open مشکل ساز است

قابلیت TCP Fast Open برای اولین بار در تاریخ دسامبر ۲۰۱۴ به عنوان RFC 7413 منتشر شد. این مشخصات بیان می‌کند که برنامه‌های کاربردی چگونه می‌توانند داده را به سمت کارساز، به‌طوری که در همان اولین بستهٔ TCP SYN دریافت شوند، ارسال کنند.

پشتیبانی فعلی برای این ویژگی زمان بسیار برده است و حتی امروزه در ۲۰۱۸ نیز با مشکلاتی همراه است. پیاده‌سازانِ پشتهٔ TCP و همچنین برنامه‌هایی که سعی در بهره‌وری از این ویژگی دارند مشکلاتی به همراه داشته‌اند - هردو در تشخیص آنکه در کدام نسخه از سامانه عامل سعی بر فعالسازی آن داشته باشند، و همچنین در فهم آنکه چگونگه به‌هنگام بروز مشکلات با ملایمت از این روش کنار بکشند و با آن مقابله کنند. شبکه‌های متعددی به مداخله در ترافیک TFO شناخته شده‌اند و آنها بدین ترتیب چنین دست‌دهی‌های TCP را عمداً منهدم کرده‌اند.

پردازش

پروتکل QUIC نخست توسط Jim Roskind در Google طراحی شد و ابتدا در سال ۲۰۱۲ پیاده سازی شد، و در سال ۲۰۱۳ به طور عمومی انتشار یافت هنگامیکه آزمایشات Google گسترش یافت.

در آن زمان، کماکان مدعی بودند که QUIC مخففی برای "Quick UDP Internet Connections" است، گرچه امروزه دیگر اینطور نیست و از همان زمان رد شد.

شرکت Google این پروتکل را پیاده‌سازی کرد و سپس آنرا در مرورگر متداول خود (Chrome) و همچنین سرویس‌های مرسوم و پُرکاربَرِ خود (همچو Google Search، Gmail، YouTube و غیره) گسترش داد. آنها نسخه‌های پروتکل را به طور مداوم تکرار کردند و با گذر زمان ثابت کردند که این پروتکل برای تعداد زیادی از کاربران به طرزی قابل اعتماد مناسب بوده و درست کار می‌کند.

در ژوئن ۲۰۱۵، اولین پیش نویس اینترنتی QUIC برای استانداردسازی به IETF فرستاده شد، اما تا اواخر سال ۲۰۱۶ به طول انجامید تا گروهی از متخصصانِ QUIC مور تأیید قرار گیرد و آغاز به کار کند. اما بعد از آن سریعاً رشد کرد و مورد توجه تعداد زیادی از گروه‌ها قرار گرفت.

در سال ۲۰۱۷، به نقل از مهندسین QUIC در Google تقریباً حدود ۷٪ از کل ترافیک اینترنت از این پروتکل استفاده می‌کرده‌اند. نسخه‌ی Google این پروتکل.

کارگروه مهندسی اینترنت (IETF/fa)

کارگروه QUIC که برای استانداردسازی پروتکل در IETF پایه گذاری شد سریعاً تصمیم گرفت که

پروتکل QUIC باید قادر باشد تا پروتکل‌هایی به غیر از "فقط" HTTP را هم انتقال دهد. پروتکل Google-QUIC تنها HTTP را انتقال داده است- در عمل آنچه را که بطور موثر قاب‌های HTTP/2 بوده است انتقال داده است، با استفاده از دستور و قواعد قاب HTTP/2.
همچنین اعلام شد که IETF-QUIC باید رمزگذاری و امنیت خود را به جای نگرش "دستی-سفارشی" بکار گرفته شده توسط Google-QUIC روی TLS 1.3 بنا کند.

به منظور برآوردن تقاضای بیش-از-HTTP- بفرست، معماری پروتکلِ QUIC کارگروه مهندسی اینترنت در دو لایه‌ی جدا تقسیم شد: انتقال QUIC و لایه‌ی "HTTP روی QUIC" (که دومی گاهی با عنوان "hq" یاد می‌شود).

این جدایی لایه ها، در حالی که ممکن است بی‌ضرر به نظر رسد، باعث شده است که IETF-QUIC تفاوت بسیاری نسبت به Google-QUIC اصلی پیدا کند.

با این حال طولی نکشید که کارگروه تصمیم گرفت تا به منظور پیدا کردن تمرکز و توان مناسب برای تحویل به موقع QUIC نسخه ۱، تمرکزش را بر روی تحویل HTTP قرار دهد و انتقال‌های غیر HTTP را برای بعد بگذارد.

در مارس ۲۰۱۸، هنگامی که ما کار بر روی این کتاب را شروع کردیم، برنامه آن بود که جزئیات نهایی برای QUIC نسخه ۱ را در نوامبر ۲۰۱۸ تحویل دهیم؛ بعدتر به تاریخ جولای ۲۰۱۹ موکول شد.

در حالیکه کار بر روی IETF-QUIC پیش می‌رفت، تیم گوگل جزئیاتی از نسخه‌ی IETF را استفاده کرده و به آرامی در جهت آنچه نسخه‌ی IETF ممکن است تبدیل شود شروع به توسعه دادن نسخه پروتکل خود کرد. گوگل به استفاده از نسخه‌ی QUIC خود در مرورگر و سرویس‌های خود ادامه داده است.

اکثر پیاده‌سازی‌های تحت توسعه تصمیم گرفته اند تا تمرکزشان را روی نسخه‌ی IETF قرار دهند و با نسخه گوگل سازگار نیستند.

تجربه از HTTP/2

مشخصات HTTP/2 توسط RFC 7540 در ماه مِه سال ۲۰۱۵، یک ماه قبل از آنکه QUIC برای اولین بار به IETF آورده شود انتشار یافت.

همراه با HTTP/2، مبانی تغییرات HTTP وضع شد و گروهی که HTTP/2 را ساختند این ذهنیت را داشتند که این امر به فرایند نوسازی نسخه های جدیدتر HTTP کمک می‌کند تا سریعتر از زمانی باشند که برای رفتن از نسخه ۱ به ۲ به طول انجامید (حدود ۱۶ سال).

با استفاده از HTTP / 2، کاربران و بسته های نرم افزاری به این ایده رسیدند که دیگر نمی‌توان انتظار داشت که HTTP با پروتکلی مبتنی بر متن به صورت زنجیره‌ای اجرا شود.

نام "HTTP بر روی بستر QUIC" در نوامبر ۲۰۱۸ به HTTP/3 تغییر یافت.

وضعیت

کارگروه QUIC از اواخر ۲۰۱۶ سخت مشغول تبیین و به کارگیری پروتکل‌ها بوده‌اند و اکنون قرار بر این است که این امر تا تاریخ جولای ۲۰۱۹ به پایان رسیده باشد.

از نوامبر ۲۰۱۸، هنوز آزمایش هم‌کنش‌پذیری بزرگتری در خصوص HTTP/3 انجام نگرفته است - تنها با دو پیاده سازی موجود که هیچ کدام توسط یک مرورگر یا یک نرم افزارِ باز سمتِ کارساز صورت نگرفته اند.
در حال حاضر حدود ۱۵ پیاده سازی QUIC در صفحات ویکی کارگروه QUIC فهرست شده است، اما توانایی هم‌کنش‌پذیری با آخرین نسخه‌ی پیش‌نویسِ تجدید شده راه طولانی‌ای پیش‌رو دارد.

پیاده سازی QUIC به این سادگی نیست و خود پروتکل دائم تغییر کرده است.

کارگزارها

پشتیبانی NGINX از QUIC و HTTP/3 در دست توسعه است و بنا است که در حین چرخه‌ی توسعه NGINX 1.17 توزیع گردد.

هیچ اعلامیه رسمی از طرف Apache در ارتباط با پشتیبانی از QUIC وجود ندارد.

کارخواه‌ها

هیچ یک از عرضه کنندگانِ مرورگرهای بزرگ نسخه‌ای که بتواند از نسخه‌ی QUIC کارگروه مهندسی اینترنت پشتیبانی کند را ارائه نکرده‌اند.

گوگل کروم سال‌هاست که همراه با نسخه‌ی توسعه داده شده‌ی خودش از QUIC عرضه شده است، اما در تعامل با نسخه‌ی کارگروه مهندسی اینترنت مشکل دار و پیاده سازیِ HTTP متفاوتی هم نسبت به HTTP/3 دارد.

شرکت Mozilla در حال توسعه‌ی Neqo است - یک پیاده سازیِ QUIC و HTTP/3 نوشته شده با زبانِ Rust. بناست که Neqo داخل Necko (که یک کتابخانه‌ی شبکه‌ی مورد استفاده در بسیاری از برنامه های سمت کارخواهِ مبتنی بر Mozilla است - از جمله Firefox) کار گذاشته شود.

نرم‌افزار curl اولین پشتیبانی از نسخه‌ی آزمایشی HTTP/3 (پیش‌نویسِ ۲۲) را در نشرِ 7.66.0 در تاریخ ۱۱ سپتامبرِ سال ۲۰۱۹ عرضه کرد. برای انجام کار، curl یا از کتابخانه‌ی Quiche توسط Cloudflare یا خانواده‌ی کتابخانه های ngtcp2 استفاده می‌کند.

موانع پیاده‌سازی

برای پروتکل QUIC تصمیم گرفته شده تا از TLS 1.3 به عنوان زیربنای رمزگذاری و لایه‌ی امنیت استفاده شود تا از ساخت و تولید یک چیز جدید اجتناب شود و در عوض بر روی یک پروتکل موجود و قابل اعتماد تأکید شود. گرچه، در همین هنگام، کارگروه تصمیم گرفت تا کاربرد TLS (پروتکل امنیتی لایه‌ی انتقال) در QUIC را بهینه سازد، بدین شکل که فقط باید از "پیغام های TLS" و نه "رکوردهای TLS" برای پروتکل استفاده شود.

این تغییر شاید به ظاهر بی‌ضرر باشد، اما در واقع یک مانع قابل توجهی برای بسیاری از پیاده‌سازانِ پشته‌ی QUIC ایجاد کرده است. کتابخانه های TLS موجود که از TLS 1.3 پشتیبانی می‌کنند از رابط برنامه نویسی کاربردیِ کافی برای دسترسی به این کارکرد و همچنین ایجاد امکان دسترسی به آن برای QUIC برخوردار نیستند. تعدادی از پیاده‌سازانِ QUIC از سازمان‌های

بزرگتری می‌آیند که به موازات در حال کار بر روی پشته‌ی امنیت لایه‌ی انتقالِ (TLS) خود هستند، اما به هر حال این برای همه صادق نیست.

بطور مثال OpenSSL متن باز بزرگ، هیچ API (رابط برنامه نویسی کاربردی) برای این منظور ندارد. به نظر می‌رسد که رسیدگی به این موضوع در درخواستِ انجامِ PR 8797 اتفاق می‌افتد که مقصود معرفی یک رابط برنامه نویسی کاربردی بسیار شبیه به برای BoringSSL است.

این موضوع در نهایت موجب موانعِ راه اندازی می‌شود زیرا که پشته های QUIC نیاز خواهند داشت که یا خود را بر روی کتابخانه های امنیت لایه‌ی انتقالِ دیگر بنا کنند، یا از یک نسخه‌ی جدا و تصحیح شده‌ی OpenSSL استفاده کنند و یا نیازمند بروزرسانی‌ای در نسخه ای از OpenSSL درآینده باشند.

بارِ هسته و پردازنده‌ی مرکزی

شرکت‌های Google و Facebook گفته اند که آنها برای راه اندازی QUIC در مقیاس گسترده تقریباً به دو برابر میزان CPU ای که برای همان‌مقدار ترافیک به هنگام ارائه‌ی HTTP/2 بر روی TLS استفاده می‌شود نیاز دارند.

برخی توضیحات در این مورد شامل موادر ذیل می‌شود:

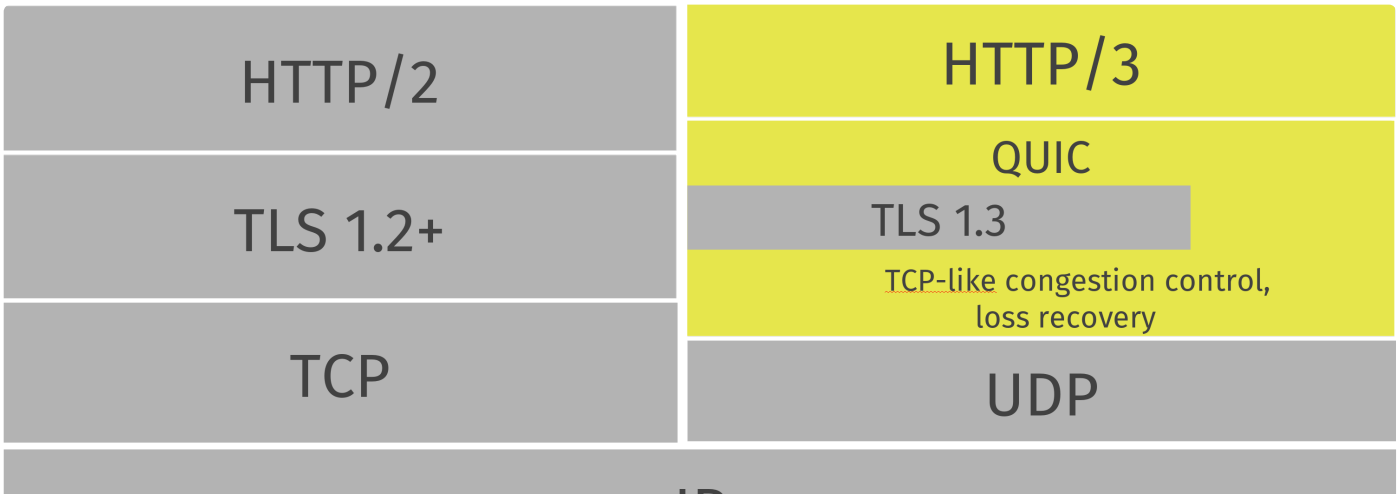
- بخش‌های UDP در لینوکس از آنجا که بطور معمول برای انتقال‌های اینچنین پرسرعت استفاده نشده است اساساً به هیچ وجه به اندازه‌ی پشته‌ی TCP بهینه نشده است.
- تخلیه‌ی بارِ TCP و TLS به سخت افزار موجود است، اما برای UDP بسیار نادرتر و برای QUIC اساساً ناموجود است.

باور بر این است که کارایی و نیازمندی‌های CPU به مرور زمان بهبود خواهند یافت.

ویژگی‌های پروتکل

پروتکل QUIC از سطحی بالا.

تصویر پایین پشتهٔ شبکهٔ HTTP/2 را در سمت چپ و پشتهٔ شبکهٔ QUIC را در سمت راست، زمانی که به‌عنوان انتقالِ HTTP استفاده شوند، نمایش می‌دهد.



IP

QUIC logo

UDP

پروتکل انتقال روی UDP

پروتکل QUIC یک پروتکل انتقالِ پیاده‌سازی شده بر روی UDP است. اگر گاهی ترافیک شبکهٔ خود را زیر نظر بگیرید، می‌بینید که QUIC به شکل بسته‌های UDP نمایان می‌شود.

بر اساس UDP، این پروتکل نیز از شماره درگاه‌های UDP برای شناسایی خدمات خاص شبکه در یک نشانی IP داده شده استفاده می کند.

تمام پیاده‌سازی‌های شناخته شدهٔ QUIC در حال حاضر در فضای کاربری هستند چرا که این امر منجر به بالا رفتن سرعت توسعه نسبت به آن چیزی خواهد بود که فضای هسته برای پیاده‌سازی اجازه می‌دهد.

آیا کارساز خواهد بود؟

شرکت‌ها و نهادهای شبکه‌ای دیگری وجود دارند که ترافیک UDP روی درگاه‌های غیر از ۵۳ (که برای DNS استفاده می‌شود) را مسدود می‌کنند. دیگری‌ها نیز داده‌ها را به گونه‌ای محدود می‌کنند که باعث می‌شود QUIC از پروتکل‌های مبتنی بر TCP هم بدتر عمل کند. برای کارهای احتمالی برخی از اپراتورها پایانی نیست.

تا آن‌جایی که می‌توان آینده را پیش‌بینی کرد، احتمالاً تمامی استفاده از انتقال‌های مبتنی بر QUIC باید بتوانند به‌طور خوشایندی به دیگر جایگزین‌ها (مبتنی بر TCP) بازگردند. مهندس‌های گوگل پیشتر نرخِ شکست را در درصدهای تک‌رقمیِ پایین پیش‌بینی کرده‌اند.

آیا پیشرفت خواهد کرد؟

اگر ثابت شود QUIC چیزی با ارزش به دنیای اینترنت اضافه می‌کند، آن‌وقت احتمال دارد افراد بخواهند از آن استفاده کنند و در نتیجه بخواهند که آن در شبکه‌هایشان کار کند و بدین ترتیب ممکن است شرکت‌ها شروع به بازنگری و بررسی موانع موجودِ خود کنند. طی سال‌های پیشرفت در توسعهٔ QUIC، نرخ موفقیت برای ایجاد و استفاده از اتصالات QUIC در اینترنت افزایش یافته است.

مطمئن

در حالی که UDP روش انتقال مطمئنی نیست، QUIC لایه‌ای بالای UDP قرار می‌دهد که موجب قابلیت

اطمینان می‌شود. انتقال مجدد بسته‌ها، نظارت تراکم، pacing و امکانات دیگری که در TCP موجود است بدین جهت ارائه می‌شود.

داده‌ای که از طریق QUIC از یک پایانه ارسال شده است، مادامی که اتصال برقرار باشد، دیر یا زود در پایانهٔ دیگر ظاهر خواهد شد.

جریان‌ها

همانند SSH، SCTP و HTTP/2، پروتکل QUIC شامل جریان‌های منطقی جداگانه‌ای درون اتصال‌های فیزیکی است. تعدادی از جریان‌های موازی که می‌توانند بدون آنکه جریان‌های دیگر را تحت تاثیر قرار دهند بطور همزمان انتقالِ داده بر روی یک اتصال واحد را انجام دهند.

یک اتصال یک چیدمانِ مذاکره شده بین دو پایانه است همانند حالتی که یک اتصال TCP کار می‌کند. یک اتصال QUIC، به یک پورت UDP و آدرس IP انجام می‌گیرد، اما پس از برقراری ارتباط آن اتصال توسط "شناسهٔ اتصال" شناخته می‌شود.

بر روی یک اتصال برقرار شده، هردو طرف می‌توانند جریانی بسازند و داده را به پایانهٔ دیگر انتقال دهند. جریان‌ها به شکلی مرتب دریافت می‌شوند و قابل اطمینان هستند، اما جریان‌های مختلف می‌توانند نامنظم دریافت شوند.

پروتکل QUIC کنترل جریان بر روی هر دوی اتصال و جریان‌ها را ارائه می‌دهد.

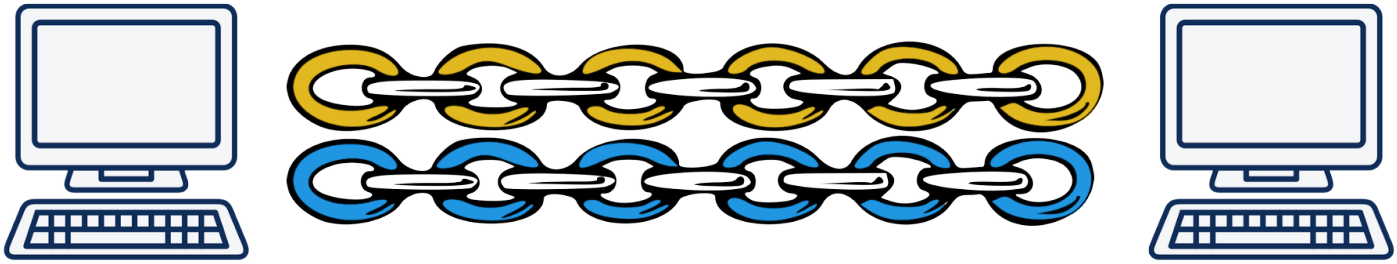
جزئیات بیشتر را در بخش‌های اتصالات و جریان‌ها ببینید.

منظم

پروتکل QUIC تحویل/توزیع منظم و کامل جریان‌ها را تضمین می‌کند، اما نه مابین جریان‌ها. بدین معنا که هر جریان داده را با حفظ ترتیب آن ارسال می‌کند، اما هر جریان ممکن است که با ترتیبی متفاوت از آنچه که نرم‌افزار فرستاده است به مقصد رسد!

به‌طور مثال: جریان‌های A و B از یک سرور به یک کارخواه منتقل شده‌اند. جریان A نسبت به جریان B تقدم دارد. در QUIC، یک بستهٔ گمشده تنها همان جریانی را که به آن تعلق دارد تحت تأثیر قرار می‌دهد. اگر جریان A یک بسته را گم کند اما جریان B خیر، جریان B می‌تواند انتقال خود را ادامه دهد و تکمیل کند در حالی که بستهٔ گمشدهٔ جریان A دوباره فرستاده می‌شود. چنین چیزی در HTTP/2 مقدور نبود.

در تصویر زیر یک جریان زرد و یک جریان آبی مابین دو پایانهٔ QUIC بر روی یک اتصال فرستاده شده است. آنها مستقل هستند و می‌توانند با ترتیبی متفاوت برسند، اما هر جریان دقیقاً منظم به نرم‌افزار تحویل داده می‌شود.



دست‌دهی‌های سریع

پروتکل QUIC هر دو نحوه اتصال RTT-0 و RTT-1 را ارائه می‌دهد، بدین معنا که در بهترین حالت QUIC نیازی به round-trip اضافی به هنگام برقراری اتصال ندارد. روش سریع‌تر، دست‌دهی RTT-0، تنها زمانی کار می‌کند که یک اتصالِ از پیش برقرار شده با یک میزبان وجود داشته باشد و یک secret از آن اتصال cache شده باشد.

دادهٔ اولیه

پروتکل QUIC به کارخواه اجازه می‌دهد تا دادهٔ موجود از دست‌دهی RTT-0 را شامل شود. این ویژگی به کارخواه این امکان را می‌دهد تا داده را هر چه سریع‌تر به جفت خود برساند، و البته از این رو همچنین این امکان را به سرور می‌دهد تا حتی پاسخ‌دهی و ارسال داده در مرحلهٔ برگشت را زودتر انجام دهد.

TLS 1.3

امنیت انتقالِ استفاده شده در QUIC از TLS 1.3 استفاده می‌کند (RFC 8446) و هرگز هیچ اتصال رمزگذاری نشده‌ای در QUIC وجود ندارد.

پروتکل TLS 1.3 نسبت به نسخه‌های پیشین TLS برتری‌هایی دارد که البته یک دلیل اصلی استفاده از آن در QUIC این است که نسخهٔ 1.3 دست‌دهی را به نحوی تغییر داده است تا شامل roundtrip های کمتری باشد. این امر تأخیر در پروتکل را کاهش می‌دهد.

نسخهٔ پیشین QUIC گوگل از یک رمزنگاری سفارشی استفاده می‌کرد.

انتقال و کاربرد

پروتکل IETF QUIC یک پروتکل انتقال است، که در بالای آن پروتکل‌های کاربردی دیگر می‌توانند استفاده شوند. پروتکل اصلی لایهٔ کاربرد HTTP/3 (یا h3) است.

لایهٔ انتقال از اتصال‌ها و جریان‌ها پشتیبانی می‌کند.

نسخهٔ پیشین QUIC گوگل، انتقال و HTTP را همراه با یکدیگر به شکلی همه‌کاره دارا بود و بیشتر یک پروتکل ارسال برای قاب‌های (frame) پروتکل HTTP/2 بر روی بستر UDP با هدف خاص بود.

پروتکل HTTP/3 بر روی QUIC

لایهٔ HTTP، به نام HTTP/3، انتقال را به‌شیوهٔ HTTP مدیریت می‌کند، از جمله فشرده‌سازی

سرایندها با استفاده از QPACK - که مشابه فشرده‌سازی سرایندها در HTTP/2 به نام HPACK است.

الگوریتم HPACK وابسته به توزیع مرتبِ جریان‌هاست و در نتیجه امکان استفادهٔ دوباره از آن در HTTP/3 بدون اعمال اصلاحات مقدور نبود، چرا که QUIC جریان‌هایی را ارائه می‌دهد که می‌توانند به صورت نامنظم تحویل داده شوند. QPACK را می‌توان نسخهٔ سازگار با QUIC از HPACK در نظر گرفت.

غیر HTTP بر روی QUIC

کار بر روی انتقال پروتکل‌هایی به غیر از HTTP بر روی بستر QUIC تا بعد از توزیع اولین نسخهٔ QUIC به تعویق افتاده است.

پروتکل QUIC چطور کار می‌کند

این بخش، بدون توضیح دقیق بیت‌ها و بایت‌ها، نحوهٔ کارکرد عناصر اصلی ساخت پروتکل انتقال QUIC را شرح می‌دهد. اگر قصد دارید تا پشتهٔ QUIC خودتان را پیاده‌سازی کنید، این توضیحات باید یک درک کلی به شما بدهد، اما برای تمامی جزئیات، به پیش‌نویس‌ها و RFC های کارگروه مهندسی اینترنت (IETF) مراجعه کنید.

۱. یک اتصال برقرار کنید

۲. ... که شامل امنیت TLS باشد

۳. سپس از جریان‌ها استفاده کنید

اتصال‌ها

یک اتصال QUIC یک مکالمه واحد بین دو پایانه‌ی QUIC است. برقراری ارتباطِ QUIC، در جهت کاهش تأخیر در برقراری ارتباط، مذاکره‌ی نسخه را با رمزنگاری و دست‌دهی‌های انتقال ترکیب می‌کند.

در واقع برای آنکه بتوان داده را از طریق چنین اتصالی ارسال کرد، یک یا تعداد بیشتری جریان باید ساخته و در نتیجه استفاده شود.

شناسه اتصال

هر اتصال یک مجموع از شناسه های اتصال را در اختیار می‌گیرد که هر کدام می‌تواند برای شناسایی اتصال استفاده شود. شناسه های اتصال بطور مستقل توسط پایانه ها انتخاب می‌شوند؛ هر پایانه شناسه اتصالی را که همتایش استفاده می‌کند انتخاب می‌کند.

کارکرد اصلی این شناسه اتصال‌ها برای اطمینان حاصل کردن از آن است که تغییرهای اعمال شده

در آدرس‌دهی در لایه‌های پروتکلِ پایین‌تر (IP، UDP، و پایین) باعث دریافت شدنِ بسته‌های اتصالِ QUIC به پایانه اشتباه نشوند.
از این قرار با بهره بردن از شناسه اتصال، اتصال‌ها می‌توانند بین آدرس IP ها و رابط‌های شبکه به طرقی که TCP هرگز نمی‌توانست حرکت کنند. بطور مثال، این جا به‌جایی اجازه می‌دهد تا یک بارگیری در حال انجام از یک اتصال تلفنی به یک اتصال سریع‌تر وای-فای انتقال پیدا کند - به هنگامی که کاربر دستگاهش را به مکانی ببرد که وای-فای داشته. به همین نحو، بارگیری می‌تواند از طریق اتصال تلفن همراه ادامه پیدا کند اگر وای-فای از دسترس خارج شود.

شماره درگاه

ساخته شده است، پس یک جایگاه شماره درگاه ۱۶ بیتی برای تمیز دادن UDP در بالای QUIC اتصال‌های دریافتی استفاده می‌شود.

مذاکره‌ی نسخه

یک درخواستِ اتصال QUIC آغاز شده از سمت یک کارخواه به کارساز می‌گوید که با کدام نسخه‌ی پروتکل QUIC می‌خواهد صحبت کند، و کارساز با فهرستی از نسخه‌های پشتیبانی شده پاسخ می‌دهد

اتصال‌ها از TLS استفاده می‌کنند

بلافاصله بعد از اولین بسته‌ای که اتصال را برقرار می‌کند، آغازگر یک قاب رمزنگاری ارسال می‌کند که شروع به تنظیم دست‌دهی لایهٔ امن می‌کند. لایهٔ امنیت از امنیت TLS 1.3 استفاده می‌کند.

هیچ راهی برای انصراف یا جلوگیری از استفادهٔ TLS برای یک اتصال QUIC وجود ندارد. این پروتکل، جهت جلوگیری از استخوان‌سازی، به نحوی طراحی شده است تا دستکاری در آن برای دستگاه‌های میانی دشوار باشد.

جریان‌ها

جریان‌ها در QUIC یک جریان-بایتِ سبکِ انتزاعی و متوالی را ارائه می‌کنند.

دو نوع ساده از جریان‌ها در QUIC وجود دارند:

- جریان‌های یک طرفه که داده‌ها را در یک جهت حمل می‌کنند: از آغازکننده جریان به سمت همتای خود.
- جریان‌های دو طرفه که به داده‌ها اجازه می‌دهند تا در هر دو جهت ارسال شوند.

هر دو نوع جریان‌ها می‌توانند توسط هرکدام از پایانه‌ها ساخته شوند، می‌توانند همزمان داده‌ها را متناوب با جریان‌های دیگر ارسال نماید، و همچنین می‌توانند لغو شوند.

برای ارسال داده‌ها بر روی یک اتصال QUIC، یک یا تعداد بیشتری جریان استفاده می‌شود.

نظارت جریان

جریان‌ها بطور انفرادی نظارت می‌شوند، بدین شکل که به یک پایانه اجازه‌ی محدود کردن و مدیریت تخصیص حافظه و اعمال دوباره‌ی فشار داده می‌شود. ساخت جریان‌ها نیز با اعلام حداکثر شناسه جریانِ مورد نظر برای پذیرش در یک زمان معین توسط هر همتا نیز نظارت شده است.

معرف‌های جریان

جریان‌ها توسط یک عدد صحیح بدون علامت ۶۴ بیتی، به عنوان شناسه جریان، شناسایی می‌شوند. آخرین دو بیتِ شناسه جریان برای شناسایی نوع جریان (یک طرفه یا دو طرفه) و آغازکننده‌ی جریان استفاده می‌شوند.

اولین بیت از سمت راست (0x1) از شناسه جریان آغازکننده‌ی جریان را شناسایی می‌کند. کارخواه‌ها جریان‌های زوج را آغاز می‌کنند (آنهایی که اولین بیت‌شان از سمت راست برابر صفر قرار گرفته است)؛ کارگزارها جریان‌های فرد را آغاز می‌کنند (آنهایی که اولین بیت‌شان از سمت راست برابر یک است).

دومین بیت از سمت راست (0x2) از شناسه جریان، جریان‌های یک طرفه و دو طرفه را از هم جدا می‌سازد. جریان‌های یک طرفه همیشه این بیت را روی یک قرار می‌دهند و جریان های دو طرفه این بیت را روی صفر قرار می‌دهند.

هم‌زمانی جریان

پروتکل QUIC به شماری از جریان‌های تصادفی اجازه می‌دهد که به شکل همزمان کار کنند. یک پایانه تعداد جریان‌های همزمان و فعال پیش‌رو را با محدود کردن حداکثر شناسه جریان محدود و مشخص می‌سازد.

حداکثر شناسه جریان مختص هر پایانه است و تنها شامل آن همتایی است که تنظیم را دریافت می‌کند.

ارسال و دریافت داده‌ها

پایانه‌ها از جریان‌ها برای ارسال و دریافت داده‌ها استفاده می‌کنند. هرچه باشد مقصود نهایی آنها همین است. جریان‌ها یک جریان-بایتِ انتزاعی و متوالی هستند. گرچه جریان‌های جداگانه لزوماً به ترتیب اصلی تحویل داده نمی‌شوند.

الویت بندی جریان

اگر منابع تخصیص داده شده به جریان‌ها بطور صحیح اولویت بندی شده باشند تسهیم جریان تاثیر بسزایی بر روی عملکرد برنامه می‌گذارد. تجربه با دیگر پروتکل‌های توزیع شده، همانند HTTP/2 نشان می‌دهد که استراتژی‌های اولویت بندیِ مؤثر اثر مثبتِ بسزایی بر روی عملکرد دارند.

پروتکل QUIC خودش قابهایی برای تبادل اطلاعات اولویت بندی را ارائه نمی‌دهد. در عوض به دریافت اطلاعاتِ اولویت بندی توسط برنامه‌ای که از QUIC استفاده می‌کند اتکا می‌کند. پروتکل‌هایی که از QUIC استفاده می‌کنند این امکان و قابلیت را دارند که هر ترتیب اولویت بندی که متناسب با معنای برنامه آنها باشد تعریف کنند.

هنگامی‌که HTTP/3 از طریق QUIC انجام می‌گیرد، اولویت بندی در لایه‌ی HTTP صورت می‌پذیرد.

زمان تأخیر چرخشی صفر (fa/0-RTT)

برای کاهش زمان مورد نیاز جهت برقراری یک اتصال جدید، کارخواهی که قبل‌تر به یک کارگزار متصل شده است ممکن است برخی پارامترها را از آن اتصال ذخیره کند و سپس یک اتصال **RTT-0** با کارگزار برقرار کند. این به کارخواه اجازه می‌دهد تا داده را بدون آنکه برای تکمیل یک دست‌دهی منتظر بماند بلافاصله ارسال کند.

بیت چرخشی

احتمالاً یکی از طولانی‌ترین بحث‌های طراحی در بین کارگروه QUIC که موضوع چند صد پیغام و ساعت‌ها مناظره بوده است مربوط به تنها یک بیت است: بیتِ چرخش (spin bit).

حامیان این بیت به نیازِ وجود امکان اندازه‌گیری تأخیر برای اپراتورها و مردمی که در مسیرِ میان دو پایانه‌ی QUIC هستند پافشاری می‌کنند.

مخالفان این ویژگی علاقه‌ای به فاش شدن احتمالی اطلاعات ندارند.

چرخاندن یک بیت

هر دو پایانه‌ها، کارخواه و کارگزار، برای هر اتصالِ QUIC یک مقدار چرخش (spin) را حفظ می‌کنند، ۰ یا ۱، و آن بیت چرخش را روی بسته‌هایی که برای آن اتصال ارسال می‌کنند به مقدار مناسب معین می‌کنند.

سپس هردو طرف بسته‌ها را با آن بیت چرخش در طرل یک round trip با مقداری یکسان ارسال می‌کنند و پس از آن، مقدار را تغییر می‌دهند. سپس نتیجه ضربانی از صفر و یک‌ها در آن بخش بیت است که ناظران می‌توانند مشاهده کنند.

این اندازه‌گیری تنها هنگامی کار می‌کند که ارسال‌کننده محدود به نرم‌فزار یا کنترل جریان نباشد و همچنین سازمان‌دهی مجدد بسته‌ها برروی شبکه می‌تواند داده را اغوا کند.

فضای کاربر

پیاده‌سازی پروتکل انتقال در فضای کاربری به نسخه‌دهی و توسعهٔ سریع‌تر پروتکل کمک می‌کند، چرا که توسعهٔ آن بدون ناگزیر ساختن کارخواه و کارگزار به به‌روزرسانی هستهٔ سیستم‌عامل‌هایشان برای استفادهٔ نسخه‌های جدید به مراتب ساده‌تر خواهد بود.

هیچ چیز در QUIC مانع پیاده‌سازی و ارائهٔ توسط هستهٔ سیستم‌عامل نمی‌شود، حتی ممکن است کسی آن را ایدهٔ خوبی بداند.

پیاده‌سازی‌های فراوان

یک تأثیر آشکار از پیاده‌سازی یک پروتکلِ انتقال در فضای کاربری این است که می‌توانیم انتظار مشاهدهٔ پیاده‌سازی‌های مستقل فراوانی را داشته باشیم.

در آینده نرم‌افزارهای مختلف احتمالاً شامل (یا سوار) پیاده‌سازی‌های مختلف HTTP/3 و QUIC باشند.

رابط برنامه‌نویسی کاربردی (fa/API)

یکی از شاخص‌های موفقیت برای TCP و برنامه‌هایی که از آن استفاده می‌کنند، کانالِ رابط برنامه‌نویسی استاندارد شده است. کانال رابط برنامه‌نویسی از قابلیت خوبی برخوردار است و با استفاده از این رابط برنامه‌نویسی می‌توانید برنامه‌ها را بین سیستم‌عامل‌های مختلف جابه‌جا کنید همانگونه که TCP کار می‌کند.

پروتکل QUIC به آنجا نرسیده است. هیچ رابط برنامه‌نویسی استانداردی برای QUIC وجود ندارد.

با QUIC، شما لازم است یکی از کتابخانه‌های پیاده‌سازی شده‌ی موجود را انتخاب کنید و با رابط برنامه‌نویسی آن بمانید. این باعث می‌شود نرم‌افزارها تا حدی بر روی یک کتابخانه "قفل" شوند. تغییر کتابخانه به معنای تغییر رابط برنامه‌نویسی خواهد بود و بنا بر این می‌تواند مستلزم زحمت‌های فراوان باشد.

همچنین، از آنجا که QUIC بطور معمول در فضای کاربر پیاده سازی شده است، نمی‌تواند به همین سادگی کانال رابط برنامه‌نویسی را بسط دهد یا مانند عملکرد TCP و UDP موجود ظاهر شود. استفاده از QUIC به معنای استفاده از رابط برنامه‌نویسی دیگری است تا کانال رابط برنامه‌نویسی.

پروتکل انتقال ابرمتن نگارش ۳ (fa/HTTP/3)

همانطور که پیش‌تر اشاره شد، HTTP اولین و مهم‌ترین پروتکل برای انتقال بر روی بستر QUIC است.

بسیار شبیه به هنگامیکه HTTP/2 معرفی شده بود تا HTTP را به شیوه‌ای کاملاً جدید انتقال‌دهی کند، HTTP/3 دارد دوباره راهی جدید را برای ارسال HTTP از طریق شبکه معرفی می‌کند.

پروتکل HTTP هنوز هم همچون گذشته دارد همان الگوها و نمونه‌ها را حفظ می‌کند. سرایندها

هستند و یک بدنه، یک درخواست هست و یک پاسخ. دستورالعمل‌ها، کوکی‌ها و حافظه نهان وجود دارد. آنچه که اساساً تغییر می‌کند با HTTP/3 این است که چگونه بیت‌ها به آن سوی ارتباط ارسال می‌شوند.

به منظورِ انجامِ HTTP بر روی QUIC، تغییراتی لازم بود و حاصل آن چیزیست که امروزه HTTP/3 می‌نامیم. این تغییرات نیاز بود، بدلیل طبیعت متفاوت QUIC در مقابل TCP. این تغییرات به شرح زیر هستند:

- در QUIC جریان‌ها توسط خودِ انتقال فرامم می‌گردند، حال آنکه در HTTP/2 جریان‌ها داخل لایه‌ی HTTP صورت میگرفتند.
- بخاطر آنکه جریان‌ها از یکدیگر مستقل‌اند، پروتکلِ فشرده‌سازیِ سرایندِ استفاده شده برای HTTP/2 نمی‌توانست بدون آنکه موجبِ یک موقعیت مسدود کننده‌ی سر شود استفاده گردد.
- جریان‌های QUIC کمی نسبت به جریان‌های HTTP/2 متفاوت هستند. بخش HTTP/3 تا حدی این جزئیات را پوشش خواهد داد.

نشانی وب‌های HTTPS://

پروتکل HTTP/3 با استفاده از URL های HTTPS:// اجرا می‌شود. جهان پر است از این نشانی وب‌ها، و معرفی کردنِ طرح URL جدیدی برای پروتکل جدید، غیر عملی و کاملاً غیر منطقی شمرده می‌شود. همانطور که HTTP/2 نیاز به طرحی جدید نداشت، HTTP/3 نیز نیازی نخواهد داشت.

پیچیدگی اضافه شده در وضعیت HTTP/3 این است که بر خلاف آن زمان که HTTP/2 روشی جدید برای انتقال HTTP بود و هنوز همانند HTTP/1 بر اساس TLS و TCP، پروتکل HTTP/3 بر روی بستر QUIC بنا شده است، و این موضوع از چند جنبهٔ مهم امور را تغییر می‌دهد.

نشانی وب‌های قدیمی، متن‌آشکار، و `HTTP://` همانگونه که هستند باقی می‌مانند، و همینطور که با انتقال‌های امن‌تر به‌سمت آینده پیش می‌رویم، احتمالاً آنها کمتر و کمتر مورد استفاده قرار می‌گیرند. درخواست چنین URL هایی برای استفاده از HTTP/3 ارتقا نخواهند یافت. در حقیقت، آنها برای HTTP/2 هم به‌ندرت ارتقا پیدا می‌کنند، اما از برای دلایلی دیگر.

اتصال اولیه

اولین اتصال به یک میزبان جدید تا به‌حال بازدید نشده برای یک نشانی `HTTPS://`، احتمالاً باید بر روی TCP صورت بگیرد (احتمالاً علاوه بر تلاش موازی برای اتصال از طریق QUIC). میزبان ممکن است کارسازی قدیمی و بدون پشتیبانی QUIC باشد، یا ممکن است در مسیر واسطی قرار گرفته باشد که مانع موفقیت اتصال QUIC شود.

یک کارخواه و کارساز امروزی احتمالاً در اولین مصافحه بر سر HTTP/2 مذاکره می‌کنند. هنگامی که اتصال برقرار شد و کارساز به درخواست HTTP کارخواه پاسخ داد، کارساز می‌تواند در مورد پشتیبانی خود از و ترجیحش برای HTTP/3 به کارخواه بگوید.

بوت اِسترپ با سرایندِ Alt-svc

سرایندِ خدمات جایگزین (Alt-svc:) و قالبِ متناظر `ALT-SVC` آن در HTTP/2 منحصراً برای QUIC یا HTTP/3 ساخته نشده‌اند. آنها بخشی از یک مکانیزم از پیش‌طراحی و ساخته شده هستند تا که یک سرور بتواند به کارخواه بگوید: *"نگاه کن، من دارم همین سرویس مشابه را روی این میزبان با استفاده از این پروتکل بر روی این درگاه اجرا می‌کنم"*. جزئیات را در RFC 7838 ببینید.

به کارخواهی که چنین پاسخ Alt-svc را دریافت می‌کند پیشنهاد می‌شود که با استفاده از پروتکل مشخص‌شده، در صورت پشتیبانی و تمایل، به آن میزبان معلوم به‌طور موازی در پس‌زمینه متصل شود و چنانکه این امر موفقیت‌آمیز باشد عملیاتش را به‌طور کامل از روی اتصال اولیه به آن تغییر دهد.

اگر اتصال اولیه از HTTP/2 یا حتی HTTP/1 استفاده کند، سرور می‌تواند پاسخ بدهد و به کارخواه بگوید که می‌تواند دوباره متصل شود و HTTP/3 را امتحان کند. این می‌تواند به سمت همان میزبان یا دیگر میزبانی که می‌داند چگونه به آن خاستگاه خدمت کند باشد. اطلاعاتی که در چنین پاسخ Alt-svc داده شده است دارای زمان‌سنجِ انقضا هستند که باعث می‌شود کارخواه‌ها بتوانند اتصال‌ها و درخواست‌های سپسین را با استفاده از پروتکل جایگزینِ پیشنهادی مستقیماً به سمت آن میزبان جایگزین، برای زمانی مشخص، هدایت کنند.

نمونه

یک سرور HTTP در پاسخش شامل یک سرایند `Alt-Svc:` است:

```
1 Alt-Svc: h3=":50781"
```

این نشانگر این است که HTTP/3 بر روی درگاه UDP شمارهٔ ۵۰۷۸۱ در همان میزبانی که برای دریافت این پاسخ استفاده شده در دسترس است.

یک کارخواه سپس می‌تواند اقدام به برقراری یک اتصال QUIC به‌سمت آن مقصد کند و اگر موفقیت آمیز بود، به‌جای ادامهٔ ارتباط اولیهٔ نسخهٔ HTTP، بدان شکل با خاستگاه به ارتباط خود ادامه دهد.

جریان‌های QUIC و HTTP/3

پروتکل HTTP/3 برای QUIC ساخته شده است، در نتیجه از مزایای جریان‌ها در QUIC بهره کامل می‌برد، در حالی که HTTP/2 مجبور بود تا کل جریان و مفهوم multiplexing خود را بر روی TCP طراحی کند.

درخواست‌های انجام شده‌ی HTTP بر روی HTTP/3 از یک مجموعه‌ی بخصوص از جریان‌ها استفاده می‌کنند.

قاب‌های HTTP/3

پروتکل HTTP/3 به معنای راه اندازی جریان‌های QUIC و ارسال دسته ای از قاب‌ها به پایانه‌ی دیگر است. گرچه تعداد ثابت کمی (در واقع ۹ عدد در ۱۸ دسامبر ۲۰۱۸!) از قاب‌های شناخته شده در HTTP/3 وجود دارد. که مهم‌ترین آنها احتمالاً به شرح ذیل‌اند:

- قاب HEADERS، که سرایندهای فشرده‌ی HTTP را ارسال می‌کند
- قاب DATA، محتوای داده‌ی دودویی ارسال می‌کند
- قاب GOAWAY، خواهشمند است این اتصال را پایان دهید

درخواست پروتکل انتقال ابرمتن

کارخواه درخواست HTTP خود را روی یک جریان دوطرفه‌ی QUIC آغاز شده از سمت خود ارسال می‌کند.

یک درخواست از یک قاب HEADERS تشکیل می‌شود و ممکن است بطور اختیاری با یک یا دو قاب دیگر نیز همراه باشد: یک مجموعه از قاب‌های DATA و احتمالاً یک قاب HEADERS نهایی برای پشت‌بندهایش.

کارخواه پس از پس از ارسال یک درخواست، جریان را برای ارسال می‌بندد.

پاسخ پروتکل انتقال ابرمتن

سرور پاسخ HTTP خود را روی جریان دوطرفه برمی‌گرداند. یک قاب HEADERS، مجموعه‌ای از قاب‌های DATA و احتمالاً پشت‌بندش یک قابِ HEADERS.

سرایندهای QPACK

قاب‌های HEADERS شامل سرایندهای فشرده شده‌ی HTTP با استفاده از الگوریتم QPACK هستند. پروتکل QPACK در روش شبیه به فشرده‌سازی HPACK (RFC 7541) در HTTP/2 است، اما اصلاح شده برای کار با جریان‌های دریافت شده‌ی خارج از ترتیب.

لازم به ذکر که QPACK خودش از دو جریان اضافی QUIC یک‌طرفه مابین دو پایانه استفاده می‌کند. آنها، در هر دو جهت، برای حمل کردن اطلاعات جدول پویا استفاده می‌شوند.

اولویت بندی

همانطور که پیشتر اشاره شد، اولویت‌بندی بین جریان‌ها از مشخصات اصلی HTTP/3 حذف شده است تا که به‌طور جداگانه روی آن کار شود.

این به‌دلیل آموخته‌ها از مدل اولویت‌بندی HTTP/2 و پیاده‌سازی آن (یا عدم وجود آن) در دنیای واقعی بود.

یک مدل اولویت‌بندی ساده‌تر از HTTP/2 با استفاده از بخش‌های سرامد HTTP به همراه تعداد محدودی تنظیمات اولویت‌بندی، پیشنهاد شده است. این یک تغییر اساسی نسبت به پرچم‌های وابستگی و وزن در قاب‌های سرامد HTTP/2 است و به شما امکان درک بهتر لایهٔ کاربرد را می‌دهد. امکان اولویت‌بندی دوباره و پشتیبانی از آن هنوز مورد بحث و گفتگو است. HTTP/2 برای مدیریت این موضوع قاب‌های اولویت‌بندی را داشت، اما جریان‌های مستقل در QUIC و HTTP/3 این مسئله را به مراتب پیچیده‌تر می‌سازد و ازین‌رو مزایا و پیچیدگی آن هنوز در مرحلهٔ مناظره است.

زمانی که (یا اگر!) مدل اولویت‌بندی بهتری برای HTTP/3 مورد توافق قرار گرفت، به امکان پیش‌انتقال آن به HTTP/2 جهت رفع پیچیدگی و نگرانی‌های پیاده‌سازی آن سمت نیز امید است.

فشارِ سرور

فشار سرور HTTP/3 شبیه به آن چیزیست که در (RFC 7540) HTTP/2 توصیف شده است، اما از ساز و کار متفاوتی بهره می‌گیرد.

یک فشار سرور (server push) در واقع پاسخ به درخواستی است که کارخواه هرگز ارسال نکرد!

فشارهای سرور تنها در صورتی ایجاد می‌شوند که از سمت کارخواه با آنها موافقت شده باشد. در HTTP/3 کارخواه حتی محدودیتی برای تعداد فشارهایی که قبول می‌کند با اعلام بیشترین شناسه‌ی جریانِ فشار برای کارساز ایجاد می‌کند. و از آن حد بالاتر رفتن موجب خطا در اتصال می‌گردد.

حتی هنگامی که که از پیش‌گفته شده باشد که فشارها توسط کارخواه مورد قبول هستند، هر جریانِ فشار می‌تواند در هر زمان که کارخواه صلاح بداند لغو گردد. که در این صورت یک قاب CANCEL_PUSH به سمت سرور فرستاده می‌شود.

دشواری

از همان زمانی که این ویژگی برای نخستین بار در توسعه‌ی HTTP/2 مطرح شد و بعدتر که پروتکل بر روی بستر اینترنت توسعه پیدا کرد و توزیع شد، در خصوص این ویژگی بحث شد، انتقاد شد و به کرّات به روش‌های گوناگون مورد حمله قرار گرفت تا که به حالتی قابل استفاده درآید.

فشار هرگز "رایگان" نیست چرا که در ذخیره و نگاه‌داشت یک نیم round-trip هنوز هم از پهنای باند استفاده می‌کند. این معمولاً برای سمت کارساز سخت یا ناممکن است که به درستی و با سطح بالایی از اطمینان بداند که یک منبع آیا به فشار نیاز دارد یا خیر.

مقایسه با HTTP/2

پروتکل HTTP/3 برای QUIC طراحی شده است، که پروتکل انتقالی است که به خودی خود جریان‌ها را مدیریت می‌کند.

پروتکل HTTP/2 برای TCP طراحی شده است، و در نتیجه جریان‌ها را در لایه‌ی HTTP مدیریت می‌کند.

شباهت‌ها

هر دوی پروتکل‌ها به کارخواهان مجموعه‌ای از ویژگی‌های کمابیش مشابه را ارائه می‌دهند.

- هر دو پروتکل پشتیبانی از server push را ارائه می‌دهند
- هر دو پروتکل دارای فشرده سازی سرایند هستند، و QPACK و HPACK در طراحی شبیه هستند.
- هر دو پروتکل توزیع بر روی یک تک-اتصال با استفاده از جریان‌ها را ارائه می‌دهند

تفاوت‌ها

تفاوت‌ها در جزئیات و عمدتاً در این حیطه هستند به لطف استفاده‌ی HTTP/3 از QUIC:

- پروتکل HTTP/3 به لطف دست‌دهی RTT-0 پروتکلِ QUIC از داده اولیه پشتیبانی بهتری می‌کند، هنگامیکه TCP Fast Open و TLS هماره داده کمتری ارسال می‌کنند و با مشکل مواجه می‌شوند.
- پروتکل HTTP/3 به لطف QUIC در مقایسه با TCP + TLS از دست‌دهی‌های به مراتب سریع‌تری برخوردار است.
- پروتکل HTTP/3 در نسخه‌ی نا-امن و بدون رمزگذاری وجود ندارد. پروتکل HTTP/2 می‌تواند بدون HTTPS پیاده‌سازی و استفاده شود - اگرچه در اینترنت کمتر بدین شکل دیده می‌شود.
- پروتکل HTTP/2 می‌تواند مستقیم داخل یک دست‌دهی TLS با افزونه ALPN قرار بگیرد، حال آنکه HTTP/3 بر روی QUIC است و ازینرو ابتدا به یک پاسخ سرایند `Alt-Svc:` نیاز دارد تا کارخواه را از این عامل آگاه سازد.
- پروتکل HTTP/3 اولویت‌بندی ندارد. رویکرد HTTP/2 در اولویت‌بندی پیچیده تلقی می‌شود، و یا حتی صرفاً یک شکست، لذا کار بر روی ساخت موردی ساده‌تر در جریان است. لین طرح ساده‌تر هم برنامه‌ریزی شده تا پیش‌انتقال بتواند با استفاده از مکانیزم پسوند HTTP/2 بر روی HTTP/2 اجرا شود.

نقدگریِ مرسوم

پروتکل UDP مرگز کار نخواهد کرد

بسیاری از شرکت‌ها، اپراتورها و سازمان‌ها ترافیک UDP خارج از پورت ۵۳ (استفاده شده برای DNS) را از آنجایی که امروزه بیشتر در جهت حمله‌ها مورد سوءاستفاده قرار می‌گیرد مسدود و یا محدود می‌کنند. همچنین به‌طور خاص، برخی از پروتکل‌های UDP موجود و سرورهای پیاده‌سازی شدهٔ حاضر برای آنها، در برابر حملات تقویت شده که در آن مهاجم می‌تواند ترافیک سنگینی را به‌سمت یک هدف بی‌گناه ارسال کند، آسیب‌پذیر بوده‌اند.

پروتکل QUIC یک کاهش‌دهندهٔ داخلی در برابر چنین حملات تقویت شده دارد، بدین ترتیب که بستهٔ اولیه باید حداقل ۱۲۰۰ بایت باشد و با این محدودیت که یک سرور اجازه ندارد در ازای هر پاسخ

بیش از سه برابر حجم درخواست را، بدون آنکه ابتدا در پاسخ از سمت کاربر بسته‌ای دریافت کند، ارسال کند.

پروتکل UDP در کرنل کند است

که به نظر درست می‌رسد، حداقل در وهلهٔ نخست. ما نمی‌توانیم به ضرس قاطع بگوییم که این مسئله چگونه توسعه می‌یابد و چقدر از آن نتیجهٔ سالیان سهو و از نظر افتادگی کیفیت انتقال در UDP توسط توسعه دهندگان است.

برای اکثر کاربران اما، این کندی هرگز محسوس نیست.

پروتکل QUIC مقدار قابل توجهی CPU مصرف می‌کند

همانند قسمت «پروتکل UDP کند است» در بالا، تا حدی علت آن است که TCP و TLS زمان بیشتری برای بالغ شدن، بهبود یافتن، و به پشتیبانی سخت افزاری رسیدن داشته است.

دلایلی وجود دارد که انتظار می‌رود این موضوع با گذشت زمان بهبود یابد، و در حال حاضر نیز شاهد برخی پیشرفت‌ها در این فضا هستیم. سؤال این است که این استفادهٔ مازاد CPU تا چه اندازه به راه‌اندازان آسیب وارد می‌کند.

این فقط Google است

نه اینطور نیست. Google مشخصات اولیه را، پس از اثبات کارآییِ این نوع راه‌اندازی از پروتکل بر روی بستر UDP در مقیاس بزرگ، به IETF (کارگروه مهندسی اینترنت) آورد.

از آن موقع، افراد از شرکت‌ها و سازمان‌های متعددی در سازمان تأمین کننده بی‌طرفِ IETF گرد هم آمدند تا یک پروتکل انتقال استاندارد ایجاد کنند. در این کار، کارمندانِ Google بی‌شک حضور داشتند اما در کنارشان تعداد زیادی از کارمندان شرکت‌های دیگر نیز نقش داشتند که تلاششان بر این بود تا جایگاه پروتکل‌های انتقال در اینترنت را به جلو ببرند، از جلمه Mozilla، Facebook، Microsoft، Akamai، Cloudflare، Fastly و Apple.

این پیشرفت بسیار کوچک است

این در واقع یک نقد نیست، یک نظر است. شاید همینطور باشد، شاید این یک پیشرفت خیلی کوچک در مقایسه با زمانی که HTTP/2 ارائه شد باشد.

احتمالاً HTTP/3 در شبکه‌هایی که از دست رفتگی بسته زیاد است بهتر عمل می‌کند، همچنین handshake سریع‌تری را پیشنهاد می‌کند و در نتیجه تأخیر را چه در تشخیص و چه در حقیقت بهبود خواهد داد. اما آیا این امکانات و مزایا برای ترغیب و تهییج مردم کافیست که پشتیبانی این پروتکل را روی سرورها و سرویس‌های خود اضافه کنند؟ بی‌شک زمان و اندازه‌گیری کیفی در آینده به ما خواهد گفت!

مشخصات

در اینجا مجموعه‌ای از آخرین پیش‌نویس‌های رسمی برای بخش‌های مختلف QUIC و HTTP/3 آورده شده است.

نامتغیرها

دارایی‌های مستقل و غیروابسته به نسخهٔ پروتکل QUIC

انتقال

پروتکل QUIC: یک انتقال امن و چندگانهٔ UDP محور

بازیابی

تشخیص اتلاف و کنترل تراکم پروتکل QUIC

امنیت لایهٔ انتقال

استفاده از TLS برای امن‌سازی پروتکل QUIC

پروتکل انتقال ابرمتن

پروتکل انتقال ابرمتن نسخهٔ ۳ (HTTP/3)

روش فشرده‌سازی سرایند QPACK

روش QPACK: فشرده‌سازی سرایند برای HTTP/3

پروتکل QUIC نسخه ۲

بمنظور آنکه بیشترین تمرکز ممکن بر روی هسته‌ی پروتکل QUIC دریافت شود و بتوان آنرا به

موقع تحویل داد، برخی از ویژگی‌هایی که در اصل قرار بود تا بخشی از پروتکل مرکزی باشند به QUIC تعویق افتادند و اکنون بنا بر این است تا در نسخ بعدی QUIC قرار داده شوند. نسخه‌ی دوم QUIC یا فراتر.

گوی سحرآمیز نویسنده‌ی این کتاب خراب است و ما نمی‌توانیم با اطمینان بگوییم کدام قابلیت‌ها در نسخه‌ی ۲ می‌آیند یا نمی‌آیند. هرچند می‌توانیم به برخی از قابلیت‌هایی که بطور صریح از نسخه ۱ حذف شده اند تا "بعدتر بر روی آن‌ها کار شود" و احتمالاً در نسخه ۲ پدیدار شوند اشاره کنیم.

تصحیح خطای مستقیم (Forward Error Correction)

تصحیح خطای مستقیم (FEC) یا "کدگذاری کانال" روش به دست آوردن کنترل خطا در انتقال داده‌هاست که در آن فرستنده داده‌هایی زائد را ارسال می‌کند (برای مقابله با بروز خطا به هنگام انتقال) و دریافت‌کننده تنها قسمتی از داده که هیچ خطای مشهودی نداشته باشد را تشخیص می‌دهد.

گوگل این قابلیت را در کارهای اصلی خود با QUIC مورد آزمایش قرار داد اما این قابلیت بعدتر دوباره حذف شد چرا که آزمایش‌ها پاسخ مثبتی به همراه نداشتند. این قابلیت موضوع بحث QUIC نسخه ۲ است اما احتمالاً نیاز دارد تا شخصی ثابت کند که این در واقع یک افزودنی مفید و بدون ضرر و زیان بیش از اندازه است.

چند مسیره

چند مسیره بدین معناست که انتقال می‌تواند به خودی خود از مسیرهای شبکه گوناگون استفاده کند تا استفاده از منابع را بالا ببرد و افزونگی (redundancy) را افزایش دهد.

حامیان SCTP در جهان علاقه دارند تا به این موضوع اشاره کنند که SCTP هم اکنون از قابلیت چند مسیره برخوردار است و TCP مدرن نیز به همچنین.

داده‌های نامعتبر

ارائه‌ی جریان‌های نامعتبر به عنوان یک امکان مورد بحث قرار گرفته است، که بعد به QUIC این اجازه را می‌دهد که نرم‌افزارهای به سبک UDP را نیز جایگزین کند.

سازگاری‌های غیر HTTP

قابلیت DNS روی QUIC یکی از اولین پروتکل‌های غیر HTTP اشاره شده است که احتمال می‌رود زمانی که QUIC نسخه ۱ و HTTP/3 عرضه شوند توجهات رو به خود جلب کند. اما با آورده شدن چنین انتقال جدیدی به دنیا نمی‌توانم تصور کنم که همینجا توقف پیدا کند.

Français

Ce livre a été lancé en mars 2018. Il est prévu de documenter HTTP/3 et son protocole sous-jacent: QUIC. Pourquoi, comment fonctionnent-ils, les détails du protocole, les implémentations et plus.

Le livre est entièrement gratuit et se veut être un effort de collaboration impliquant tous ceux qui veulent aider.

Prérequis

Un lecteur de ce livre est censé avoir une compréhension de base du réseau TCP/IP, des principes fondamentaux de HTTP et du Web. Pour plus d'informations et de détails sur HTTP/2, nous vous recommandons tout d'abord de lire les détails dans http2 expliqué.

Auteur

Ce livre est créé et le travail est démarré par Daniel Stenberg. Daniel est le fondateur et le développeur principal de curl, le client HTTP le plus utilisé au monde. Daniel travaille avec et sur les protocoles HTTP et Internet depuis plus de deux décennies et est l'auteur de http2 expliqué.

Accueil

La page d'accueil de ce livre se trouve à l'adresse daniel.haxx.se/http3-explained.

Contribuer

Si vous trouvez des fautes, des omissions, des erreurs ou des mensonges flagrants dans ce document, veuillez nous envoyer une version mise à jour du paragraphe concerné et nous en ferons des versions corrigés. Nous allons donner des crédits appropriés à tous ceux qui aident. J'espère améliorer ce document avec le temps.

De préférence, vous soumettez des erreurs ou des demandes de tirage sur la page GitHub du livre.

Licence

Pourquoi QUIC

QUIC est un nom, pas un acronyme. Il se prononce exactement comme le mot anglais "quick".

QUIC est à bien des égards ce qui pourrait être perçu comme un moyen de créer un nouveau protocole de transport fiable et sécurisé, adapté à un protocole comme HTTP et pouvant résoudre certains des inconvénients connus de HTTP/2 sur TCP et TLS. La prochaine étape logique dans l'évolution du transport Web.

QUIC n'est pas limité au seul transport de HTTP. La volonté de rendre le web et les données en général plus rapides aux utilisateurs finaux est probablement la raison principale et la poussée qui a initialement déclenché la création de ce nouveau protocole de transport.

Alors pourquoi créer un nouveau protocole de transport et pourquoi le faire par dessus UDP?

QUIC logo

Souvenez-vous de HTTP/2 ?

La spécification HTTP/2 RFC 7540 a été publiée en mai 2015 et le protocole a depuis été mis en œuvre et déployé largement sur Internet et sur le World Wide Web.

Début 2018, près de 40% des 1 000 meilleurs sites Web utilisaient HTTP/2, environ 70% de toutes les demandes HTTPS de Firefox recevaient des réponses HTTP/2 et tous les principaux navigateurs, serveurs et proxies le prenaient en charge.

HTTP/2 corrige toute une série de lacunes presentes dans HTTP/1 et avec l'introduction de la deuxième

version de HTTP, les utilisateurs peuvent cesser d'utiliser toute une série de solutions de contournement. Certaines sont assez pénibles pour les développeurs Web.

L'une des principales caractéristiques de HTTP/2 est qu'il utilise le multiplexage, de sorte que de nombreux flux logiques soient envoyés sur la même connexion TCP physique. Cela rend beaucoup de choses meilleures et plus rapides. Le contrôle de congestion fonctionne bien mieux, il permet aux utilisateurs d'utiliser bien mieux le protocole TCP et ainsi de saturer correctement la bande passante, de rendre les connexions TCP plus durables - ce qui est bien pour qu'ils atteignent la vitesse maximale plus souvent qu'avant. La compression d'en-tête lui fait utiliser moins de bande passante.

Avec HTTP/2, les navigateurs utilisent généralement *une* connexion TCP avec chaque hôte au lieu des précédents *six*. En fait, les techniques de fusion et de "désarchivage" des connexions utilisées avec HTTP/2 peuvent même réduire beaucoup plus que ça le nombre de connexions.

HTTP/2 a corrigé le problème de blocage de tête de ligne HTTP, dans lequel les clients devaient attendre la fin de la première requête en ligne avant que la suivante ne puisse être envoyée.

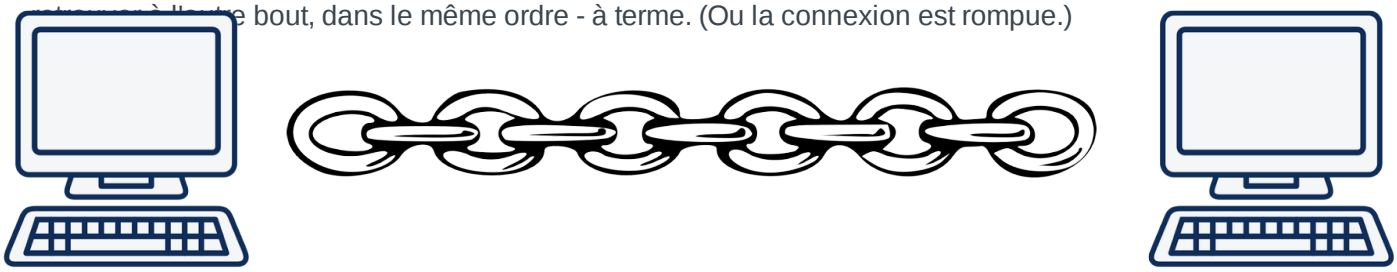
http2 man

Blocage de tête de ligne TCP

Blocage de tête de ligne TCP

HTTP/2 est réalisé sur TCP et avec beaucoup moins de connexions TCP que lors de l'utilisation de versions HTTP antérieures. TCP est un protocole pour des transferts fiables et vous pouvez le considérer

comme une chaîne imaginaire entre deux machines. Ce qui est mis sur le réseau d'un côté finira par se retrouver à l'autre bout, dans le même ordre - à terme. (Ou la connexion est rompue.)

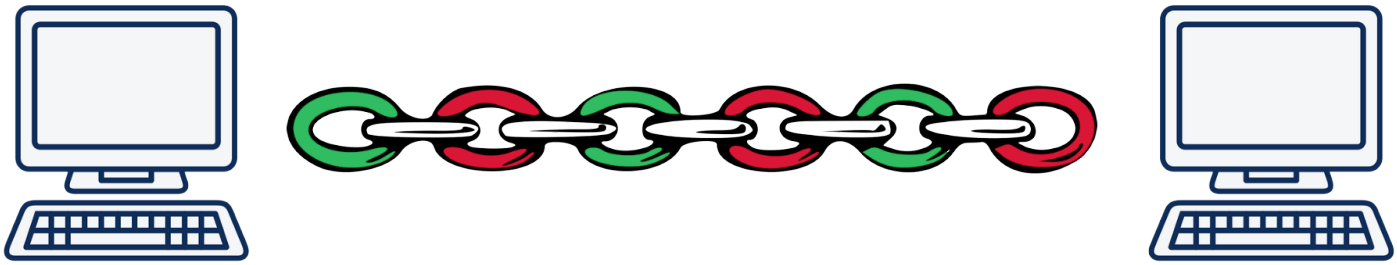


une chaîne TCP entre deux ordinateurs

Avec HTTP/2, les navigateurs classiques effectuent des dizaines, voire des centaines de transferts parallèles sur cette seule connexion TCP.

Si un seul paquet est abandonné ou perdu sur le réseau quelque part entre deux terminaisons qui parlent HTTP/2, cela signifie que toute la connexion TCP est interrompue et que le paquet perdu doit être retransmis et doit retrouver son chemin jusqu'à la destination. Puisque TCP est cette "chaîne", cela signifie que si un lien manque soudainement, tout ce qui viendrait après le lien perdu doit attendre.

Illustration utilisant la métaphore de la chaîne lors de l'envoi de deux flux sur cette connexion. Un flux rouge et un flux vert:



la chaîne montrant des liens de différentes couleurs

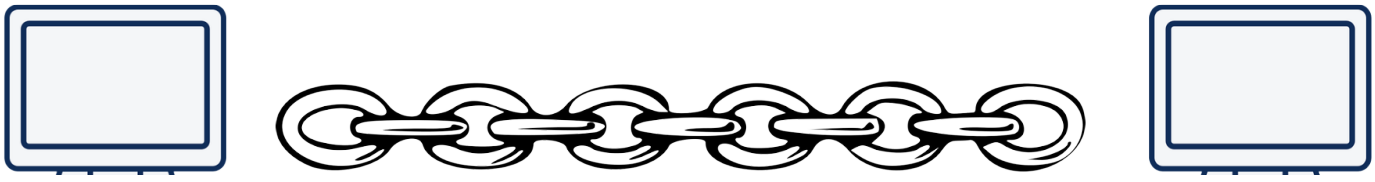
Cela devient un bloc de début de ligne basé sur TCP!

À mesure que le taux de perte de paquets augmente, HTTP/2 est de moins en moins performant. Avec 2% de perte de paquets (ce qui est une qualité de réseau épouvantable, remarquez-vous bien), des tests ont montré que les utilisateurs de HTTP/1 sont généralement mieux lotis - car ils disposent généralement de six connexions TCP pour répartir le paquet perdu donc pour chaque paquet perdu, les autres connexions sans perte peuvent toujours continuer.

Résoudre ce problème n’est pas facile, et si tout de même possible, à faire avec TCP.

Les flux indépendants évitent le blocage

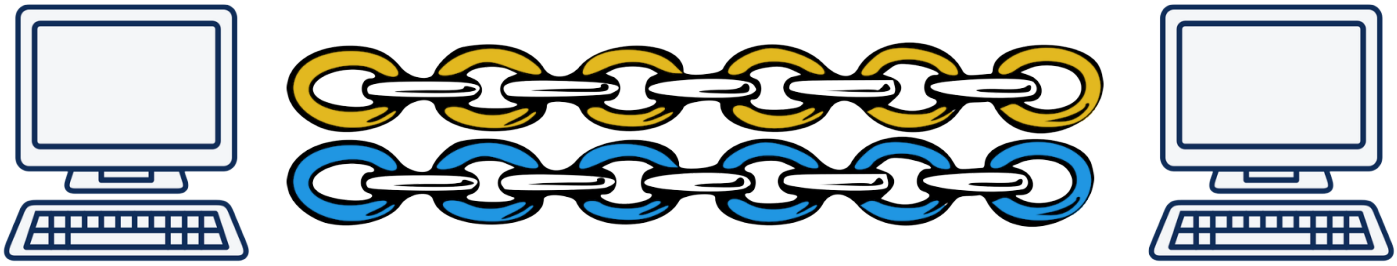
Avec QUIC, il existe toujours une connexion configurée entre les deux terminaisons qui sécurise la connexion et la livraison des données.



une chaîne QUIC entre deux ordinateurs

Lors de la configuration de deux flux différents sur cette connexion, ils sont traités indépendamment. Ainsi, si un lien manque à l'un des flux, seul ce flux, cette chaîne particulière, doit s'interrompre et attend que le lien manquant soit retransmis.

Illustré ici avec un flux jaune et un flux bleu envoyés entre deux terminaisons.



deux flux QUIC entre deux ordinateurs

TCP ou UDP

TCP ou UDP

Si nous ne pouvons pas résoudre le blocage de la tête de ligne dans TCP, nous devrions théoriquement pouvoir créer un nouveau protocole de transport à côté de UDP et TCP dans la stack réseau. Ou peut-être même utilisez-vous SCTP qui est un protocole de transport normalisé par l'IETF dans la RFC 4960 avec plusieurs des caractéristiques souhaitées.

Cependant, au cours des dernières années, les efforts pour créer de nouveaux protocoles de transport ont été presque complètement arrêtés en raison de la difficulté de les déployer sur Internet. Le déploiement de nouveaux protocoles est entravé par de nombreux pare-feu, NATs, routeurs et autres boîtes centrales qui autorisent uniquement le protocole TCP ou UDP déployés entre les utilisateurs et les serveurs qu'ils doivent atteindre. L'introduction d'un autre protocole de transport provoque l'échec de N% des connexions car elles sont bloquées par des boîtes ne la voyant pas comme étant UDP ou TCP et donc malfaisantes ou erronés d'une manière ou d'une autre. Le taux d'échec de N% est souvent jugé trop élevé pour en valoir la peine.

De plus, les modifications apportées à la couche de protocole de transport de la stack réseau impliquent généralement des protocoles implémentés par les noyaux de système d'exploitation. La mise à jour et le déploiement de nouveaux noyaux de système d'exploitation est un processus lent qui nécessite des efforts importants. De nombreuses améliorations TCP normalisées par l'IETF ne sont pas beaucoup déployées ou utilisées car elles ne sont pas énormément prises en charge.

Pourquoi pas SCTP sur UDP

SCTP est un protocole de transport fiable avec des flux, et pour WebRTC, il existe même des implémentations existantes qui l'utilisent via UDP.

Cela n’a pas été jugé suffisant comme alternative à QUIC pour plusieurs raisons, notamment:

- SCTP ne résout pas le problème de blocage de tête de ligne pour les flux
- SCTP exige que le nombre de flux soit déterminé lors de l'établissement de la connexion
- SCTP n'a pas un solide concept TLS/de sécurité
- SCTP a un handshake à 4 voies, QUIC offre 0-RTT
- QUIC est un flux par octet comme TCP, SCTP est basé sur des messages
- Les connexions QUIC peuvent migrer entre les adresses IP, mais SCTP ne peut pas

Pour plus de détails sur les différences, voir A Comparison between SCTP and QUIC.

Ossification

Internet est un réseau de réseaux. Il y a des équipements installés sur Internet dans de nombreux endroits pour assurer le fonctionnement de ce réseau de réseaux. Ces périphériques, les boîtiers distribués sur le réseau, sont ce que nous appelons parfois des boîtiers centraux. Les zones situées quelque part entre les points terminaisons sont l'une des deux parties principales impliquées dans un transfert de données réseau traditionnel.

Ces boîtes servent à de nombreuses fins spécifiques, mais je pense que nous pouvons dire qu'universellement, elles sont placées là parce que quelqu'un pense qu'elles doivent être là pour que les choses fonctionnent.

Les boîtiers centraux routent les paquets IP entre les réseaux, bloquent le trafic malveillant, effectuent la traduction d'adresses réseau (en anglais Network Address Translation (NAT)), améliorent les performances, tentent parfois d'espionner le trafic en passant, etc...

Afin de s'acquitter de leurs tâches, ces boîtiers doivent connaître la mise en réseau et les protocoles qu'ils surveillent ou modifient. Ils exécutent des logiciels à cette fin. Logiciels qui ne sont pas toujours mis à jour fréquemment.

Bien qu'ils soient des composants essentiels qui maintiennent Internet attaché, ils ne sont pas souvent en phase avec les dernières technologies. Le milieu du réseau ne se déplace généralement pas aussi vite que les bords, comme les clients et les serveurs du monde.

Tous les protocoles de réseau que ces boîtes pourraient vouloir inspecter et qui ont des idées sur ce qui est ok et ce qui ne l'est pas alors ont ce problème: ces boîtes ont été déployées il y a quelque temps, alors que les protocoles avaient un ensemble de fonctionnalités de cette époque. L'introduction de nouvelles fonctionnalités ou de changements de comportement inconnus auparavant risquerait d'être considérée comme mauvaise ou illégale par de telles boîtes. Ce trafic peut tout simplement être supprimé ou retardé dans la mesure où les utilisateurs ne souhaitent vraiment pas utiliser ces fonctionnalités.

C'est ce qu'on appelle "l'ossification du protocole".

Les modifications apportées au protocole TCP souffrent également d'ossification: certaines boîtes entre un client et le serveur distant détectent de nouvelles options TCP inconnues et bloquent ces connexions car ils ne savent pas quelles sont les options. S'ils sont autorisés à détecter les détails de protocole, les systèmes apprennent le comportement typique des protocoles et, avec le temps, il devient impossible de les modifier.

Le seul moyen véritablement efficace de "combattre" l'ossification consiste à chiffrer le plus possible la communication afin d'empêcher les boîtes moyennes de voir beaucoup du protocole la traversant.

Sécurisé

QUIC est toujours sécurisé. Il n'y a pas de version en texte clair clair du protocole, donc négocier une connexion QUIC signifie faire de la cryptographie et de la sécurité avec TLS 1.3. Comme mentionné ci-dessus, cela empêche l'ossification ainsi que d'autres types de blocages et traitements spéciaux, et garantit que QUIC possède toutes les propriétés sécurisées de HTTPS auxquelles les utilisateurs Web s'attendent et souhaitent.

Il n'y a que quelques paquets handshake initiaux qui sont envoyés en clair, avant que les protocoles de cryptage aient été négociés.

Latence réduite

QUIC offre à la fois des handshakes 0-RTT et 1-RTT qui réduisent le temps nécessaire pour négocier et établir une nouvelle connexion. Comparez avec le handshake à 3 voies du TCP.

En plus de ça, QUIC offre une prise en charge des "données antérieures" dès le départ, ce qui est fait pour autoriser plus de données et est utilisé plus facilement que TCP Fast Open.

Avec le concept de flux, vous pouvez établir une autre connexion logique avec le même hôte sans avoir à d'abord attendre la fin de la connexion existante.

TCP Fast Open est problématique

TCP Fast Open a été publié en décembre 2014 en tant que la RFC 7413 et cette spécification explique comment les applications peuvent transmettre des données au serveur afin qu'elles soient déjà livrées dans le premier paquet TCP SYN.

La prise en charge effective de cette fonctionnalité a pris du temps et pose encore de nombreux problèmes, même aujourd'hui en 2018. Les responsables de la mise en œuvre de la stack TCP ont rencontré des problèmes, tout comme les applications qui ont essayé de tirer parti de cette fonctionnalité, en sachant dans quelle version d'OS essayer pour l'activer mais également pour savoir comment revenir en arrière avec élégance et régler les problèmes qui surviennent. Plusieurs réseaux ont été identifiés pour interférer avec le trafic TFO et ont donc activement ruiné de telles handshakes TCP.

La procédure

Le protocole QUIC originel a été conçu par Jim Roskind chez Google et a été initialement mis en œuvre en 2012, puis annoncé publiquement au monde en 2013 lorsque l'expérimentation de Google s'est élargie.

À l'époque, il était toujours prétendu que QUIC était l'acronyme de "Quick UDP Internet Connections", mais il a été abandonné depuis.

Google a mis en œuvre le protocole et l'a ensuite déployé à la fois dans son navigateur beaucoup utilisé (Chrome) et dans ses services côté serveur beaucoup utilisés (Google search, gmail, youtube, etc...). Ils ont répété les versions du protocole assez rapidement et, avec le temps, ils ont prouvé que le concept fonctionnait de manière fiable pour une grande partie des utilisateurs.

En juin 2015, le premier brouillon Internet pour QUIC a été envoyé à l'IETF pour une standardisation, mais il a fallu attendre la fin de l'année 2016 pour qu'un groupe de travail QUIC soit approuvé et lancé. Mais ensuite, il a immédiatement décollé avec beaucoup d'intérêt de la part de nombreux partis.

En 2017, des chiffres cités par les ingénieurs de QUIC chez Google ont indiqué qu'environ 7% de *tout* le trafic Internet utilisait déjà ce protocole. La version de Google du protocole.

IETF

Le groupe de travail QUIC mis en place pour standardiser le protocole au sein de l'IETF a rapidement décidé que le protocole QUIC devait pouvoir transférer des protocoles autres que "simplement" HTTP. Google-QUIC ne transportait jamais que HTTP - en pratique, il transportait ce qui était en réalité des trames HTTP/2, en utilisant la syntaxe HTTP/2.

Il a également été indiqué que l'IETF-QUIC devrait baser son cryptage et sa sécurité sur TLS 1.3 au lieu de l'approche "personnalisée" utilisée par Google-QUIC.

Afin de satisfaire la demande d'envoi-plus-que-HTTP, l'architecture de protocole IETF QUIC a été scindée en deux couches distinctes: la couche de transport QUIC et la couche "HTTP sur QUIC" (cette dernière parfois appelée "hq").

Cette division de couche, bien que cela puisse paraître inoffensif, a entraîné une grande différence entre l'IETF-QUIC et l'original de Google-QUIC.

Cependant, le groupe de travail a rapidement décidé que pour se concentrer correctement et délivrer la version 1 de QUIC à temps, il se concentrerait sur la livraison de HTTP, laissant les transports non-HTTP à un travail ultérieur.

En mars 2018, lorsque nous avons commencé à travailler sur ce livre, le plan était d'expédier la spécification finale de la version 1 de QUIC en novembre 2018; cela a ensuite été reporté à juillet 2019.

Alors que les travaux sur l'IETF-QUIC ont progressé, l'équipe de Google a incorporé les détails de la version de l'IETF et a commencé à faire progresser lentement sa version du protocole vers ce que pourrait devenir la version de l'IETF. Google a continué d'utiliser sa version de QUIC dans son navigateur et ses services.

La plupart des nouvelles implémentations en cours de développement ont décidé de se concentrer sur la version de l'IETF et ne sont pas compatibles avec la version de Google.

Expérience depuis HTTP/2

La spécification HTTP/2 RFC 7540 a été publiée en mai 2015, juste un mois avant que QUIC ne soit présenté pour la première fois à l'IETF.

Avec HTTP/2, les bases de la modification de HTTP sur le réseau ont été aménagées et le groupe de travail qui a créé HTTP/2 était déjà convaincu que cela aiderait à effectuer une itération sur de nouvelles versions HTTP plus rapidement qu'il a fallu pour passer de la version 1 à la version 2 (environ 16 ans).

Avec HTTP/2, les utilisateurs et les stacks logiciels se sont habitués à l'idée que le protocole HTTP ne peut plus être supposé être exécuté en série avec un protocole texte.

HTTP-over-QUIC a été renommé HTTP/3 en novembre 2018.

Statut

Le groupe de travail de QUIC a travaillé d'arrache-pied depuis fin 2016 pour spécifier les protocoles et le plan est maintenant de le faire d'ici juillet 2019.

En novembre 2018, il n'y avait toujours pas de tests d'interopérabilité plus grands avec HTTP/3 - uniquement avec les deux implémentations existantes et aucun d'entre eux n'est effectué par un navigateur ou un logiciel populaire de serveur ouvert.

Il y a une quinzaine de différentes implémentations de QUIC répertoriées dans les pages de wiki des groupes de travail de QUIC, mais toutes ne peuvent pas interagir sur les dernières revisions des spécifications.

L'implémentation de QUIC n'est pas facile et le protocole a continué à évoluer, même à cette date.

Les serveurs

Le support NGINX pour QUIC et HTTP/3 est en cours de développement. Il est prévu qu'il soit déployé durant le cycle de développement NGINX 1.17.

Il n'y a eu aucune déclaration publique en termes de support QUIC d'Apache.

Les clients

Aucun des plus gros éditeurs de navigateurs n'a encore fourni de version, quel que soit son état, pouvant

exécuter la version IETF de QUIC ou de HTTP/3.
Google Chrome est livré avec une implémentation fonctionnelle de la version QUIC de Google depuis de nombreuses années, mais cela n'interagit pas avec le protocole QUIC officiel et son implémentation HTTP est différente de HTTP/3.

Mozilla est en train de développer Neqo - une implémentation de QUIC et HTTP/3 écrit en Rust. Neqo est prévu d'être intégré dans Necko (qui est une biliothèque réseau utilisé dans plein d'applications clientes basés sur Mozilla - Firefox inclus).

Obstacles d'Implémentation

QUIC a décidé d'utiliser TLS 1.3 comme base pour la couche de chiffrement et de sécurité pour éviter d'inventer quelque chose de nouveau et plutôt s'appuyer sur un protocole fiable et existant. Cependant, ce faisant, le groupe de travail a également décidé que, pour rationaliser réellement l'utilisation de TLS dans QUIC, il ne devrait utiliser que des "messages TLS" et non des "enregistrements TLS" pour le protocole.

Cela peut sembler être un changement anodin, mais cela a en fait créé un obstacle considérable pour de nombreux développeurs de stack QUIC. Les bibliothèques TLS existantes qui prennent en charge TLS 1.3 n'ont tout simplement pas assez d'API pour exposer cette fonctionnalité et permettre à QUIC d'y accéder. Bien que plusieurs développeurs QUIC proviennent de grandes organisations qui travaillent sur leur propre stack TLS en parallèle, cela n’est pas vrai pour tout le monde.

OpenSSL, le poids lourd open source dominant, par exemple, n’a pas d’API et n’a manifesté aucun désir de la fournir dans les meilleurs délais (à partir de novembre 2018).

Cela entraînera éventuellement des obstacles de déploiement puisque les stacks QUIC devront se baser sur d'autres bibliothèques TLS, utiliser une version OpenSSL corrigée ou nécessiter une mise à jour d'une version future d'OpenSSL.

Noyaux et charge du processeur

Google et Facebook ont tous deux mentionné que leurs déploiements à grande échelle de QUIC requièrent environ deux fois plus de ressources processeur que la même charge de trafic lorsque HTTP/2 est utilisé via TLS.

Quelques explications à cela incluent

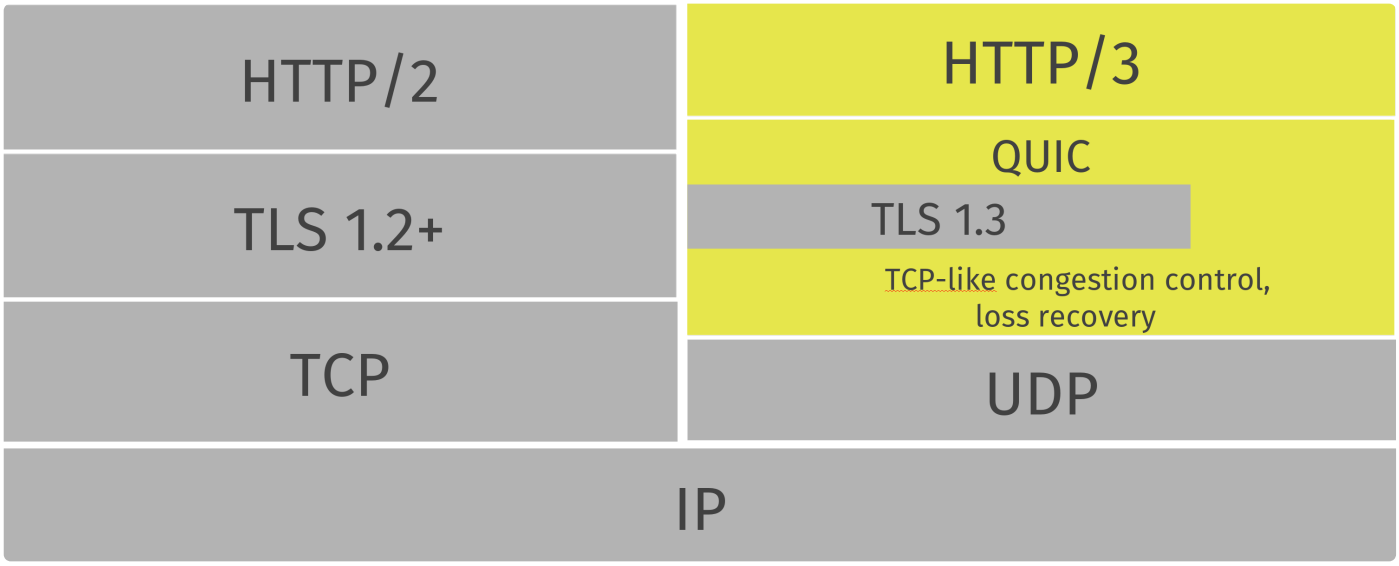
- Les composants UDP, principalement sous Linux, ne sont pas du tout aussi optimisés que la stack TCP, car elle n’a pas été utilisée traditionnellement pour des transferts à grande vitesse comme celui-ci.
- Le déchargement TCP et TLS sur le matériel existe, mais cela est beaucoup plus rare pour UDP et pratiquement inexistant pour QUIC.

Il y a des raisons de croire que les performances et les exigences en matière de processeur s'amélioreront avec le temps

Caractéristiques du protocole

Le protocole QUIC d'un niveau élevé.

Illustré ci-dessous, la stack réseau HTTP/2 à gauche et la stack réseau QUIC à droite, quand utilisées comme transport HTTP.



QUIC logo

UDP

Protocole de transfert sur UDP

QUIC est un protocole de transfert implémenté au-dessus d'UDP. Si vous surveillez votre trafic réseau par hasard, vous verrez QUIC apparaître sous forme de paquets UDP.

Basé sur UDP, il utilise également les numéros de port UDP pour identifier des serveurs spécifiques sur une machine donnée.

Toutes les implémentations QUIC connues se trouvent actuellement dans l'espace utilisateur, ce qui permet une évolution plus rapide que ne permettent généralement pas les implémentations noyau.

Est-ce que ça va fonctionner ?

Certaines entreprises et autres configurations réseau bloquent le trafic UDP sur des ports autres que 53 (utilisé pour DNS). D'autres limitent ces données de manière à rendre QUIC moins performant que les protocoles basés sur TCP. Il n'y a pas de fin à ce que certains opérateurs peuvent faire.

Dans un avenir prévisible, toute utilisation de transports basés sur QUIC devra probablement être en mesure de faire appel à une autre alternative (basée sur TCP). Les ingénieurs de Google ont

précédemment mentionné les taux d'échec mesurés dans de faibles pourcentages à un chiffre.

Cela va-t-il s'améliorer ?

Il est fort probable que si QUIC s'avère être un atout précieux au monde d'Internet, les utilisateurs voudront l'utiliser et le feront fonctionner dans leurs réseaux, ce qui permettra aux entreprises de reconsidérer leurs obstacles. Au fil des années, le développement de QUIC a progressé, le taux de réussite de l'établissement et de l'utilisation de connexions QUIC sur Internet a augmenté.

Fiable

Bien qu'UDP ne soit pas un transport fiable, QUIC ajoute une couche au-dessus d'UDP qui introduit la fiabilité. Il offre la retransmission de paquets, le contrôle de congestion, la stimulation et les autres fonctionnalités présentes par ailleurs dans TCP.

Les données envoyées sur QUIC depuis un point de terminaison apparaîtront dans l'autre tôt ou tard, tant que la connexion est maintenue.

Flux

Semblable à SCTP, SSH et HTTP/2, QUIC propose des flux logiques séparés au sein des connexions physiques. Un certain nombre de flux parallèles pouvant transférer des données simultanément sur une seule connexion sans affecter les autres flux.

Une connexion est une configuration négociée entre deux points de terminaison, similaire au fonctionnement d'une connexion TCP. Une connexion QUIC est établie sur port UDP et une adresse IP, mais une fois établie, la connexion est associée à son "ID de connexion".

Sur une connexion établie, chaque côté peut créer des flux et envoyer des données à l'autre terminaison. Les flux sont livrés dans l'ordre et ils sont fiables, mais différents flux peuvent être livrés dans le désordre.

QUIC offre un contrôle de flux sur la connexion et les flux.

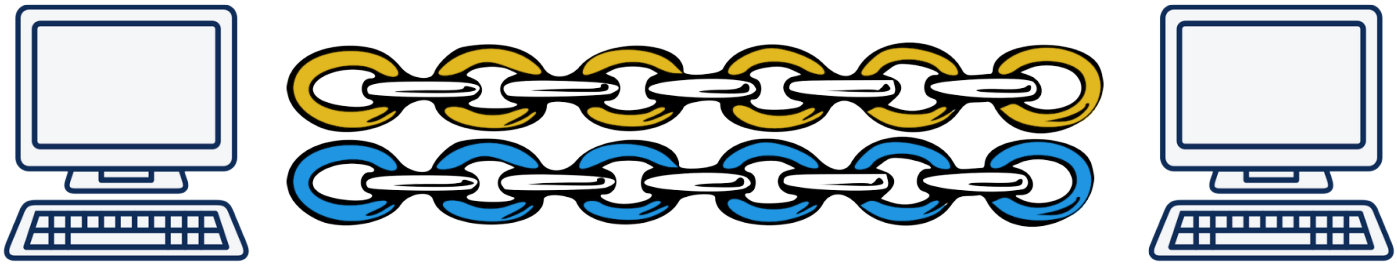
Voir plus de détails dans les sections connexion et flux

Dans l'ordre

QUIC garantit la livraison des flux dans l'ordre, mais pas entre les flux. Cela signifie que chaque flux enverra des données et maintiendra l'ordre des données, mais chaque flux pourra atteindre la destination dans un ordre différent de celui que l'application a envoyé!

Par exemple: les flux A et B sont transférés d'un serveur à un client. Le flux A est démarré en premier, puis ensuite le flux B. Dans QUIC, un paquet perdu affecte uniquement le flux auquel appartient le paquet perdu.

Si le flux A perd un paquet mais pas le flux B, le flux B peut poursuivre ses transferts et se terminer pendant que le paquet perdu du flux A est retransmis. Ce n'était pas possible avec HTTP/2.
Illustré ici avec un flux jaune et un flux bleu envoyés entre deux points de terminaison QUIC sur une seule connexion. Ils sont indépendants et peuvent arriver dans un ordre différent, mais chaque flux est livré de manière fiable, dans l'ordre, à l'application.



deux flux QUIC entre deux ordinateurs

Handshakes rapides

QUIC offre à la fois des configurations de connexion 0-RTT et 1-RTT, ce qui signifie qu'au mieux, QUIC ne nécessitera aucun aller-retour supplémentaire lors de la configuration d'une nouvelle connexion. Le plus rapide des deux, la négociation 0-RTT, ne fonctionne que si une connexion précédente a été établie avec un hôte et qu'un secret de cette connexion a été mis en cache.

Données précoces

QUIC permet à un client d'inclure des données déjà dans le handshake 0-RTT. Cette fonctionnalité permet à un client de transmettre les données à l'homologue aussi rapidement que possible, ce qui permet bien entendu au serveur de répondre et de renvoyer les données encore plus tôt.

TLS 1.3

La sécurité de transport utilisée dans QUIC utilise TLS 1.3 (RFC 8446) et il n'y a jamais de connexions QUIC non chiffrées.

TLS 1.3 présente plusieurs avantages par rapport aux anciennes versions de TLS, mais l'une des principales raisons de son utilisation dans QUIC est que la 1.3 a modifié le handshake pour exiger moins d'allers-retours. Cela réduit la latence du protocole.

L'ancienne version de QUIC de Google utilisait une crypto personnalisée.

Transport et application

Le protocole IETF QUIC est un protocole de transport sur lequel d'autres protocoles d'application peuvent

être utilisés. Le protocole de couche d'application initial est HTTP/3 (h3).

La couche de transport prend en charge les connexions et les flux.

L'ancienne version de QUIC de Google regroupait le transport et le protocole HTTP dans un même fait-tout et constituait un protocole send-http/2-frames-over-udp plus spécifique.

HTTP/3 sur QUIC

La couche HTTP effectue des transports de style HTTP, y compris la compression d'en-tête HTTP à l'aide de QPACK - ce qui est similaire à la compression HTTP/2 appelée HPACK.

L'algorithme HPACK dépend d'une livraison *ordonnée* de flux, il n'a donc pas été possible de le réutiliser pour HTTP/3 sans modifications depuis que QUIC propose des flux pouvant être livrés dans le désordre. QPACK peut être considéré comme la version de HPACK adaptée à QUIC.

Non-HTTP sur QUIC

Les travaux d'envoi de protocoles autres que HTTP sur QUIC ont été reportés après la livraison de la version 1 de QUIC.

Comment QUIC fonctionne

Sans expliquer les bits et les octets exacts sur le réseau, cette section décrit le fonctionnement des blocs de construction fondamentaux du protocole de transport QUIC. Si vous souhaitez implémenter votre propre stack QUIC, cette description doit vous donner une compréhension générale, mais pour tous les détails, référez-vous aux actuel brouillons internets et RFCs de l'IETF.

1. Configurez une connexion
2. ... ce qui inclut la sécurité TLS
3. Puis utilisez les flux

Connexions

Une connexion QUIC est une conversation unique entre deux terminaisons QUIC. L'établissement de la connexion QUIC associe la négociation de la version à la négociation cryptographique et du handshake de transfert afin de réduire le temps d'attente de l'établissement de la connexion.

Pour envoyer des données via une telle connexion, un ou plusieurs flux doivent être créés et utilisés.

ID de connexion

Chaque connexion possède un ensemble d'identifiants de connexion, ou d'IDs de connection, chacun d'entre eux pouvant être utilisé pour identifier la connexion. Les IDs de connexion sont sélectionnés indépendamment par les terminaisons; chaque terminaison sélectionne les identifiants de connexion que son pair utilise.

La fonction principale de ces IDs de connexion est de garantir que les changements d'adressage au niveau des couches inférieures du protocole (UDP, IP et au-dessous) n'entraînent pas la transmission des paquets d'une connexion QUIC à la mauvaise terminaison.

En tirant parti de l'ID de connexion, les connexions peuvent ainsi migrer entre les adresses IP et les interfaces réseau d'une manière que TCP n'aurais jamais pu. Par exemple, la migration permet à un téléchargement en cours de passer d'une connexion réseau cellulaire à une connexion wifi plus rapide lorsque l'utilisateur déplace son appareil dans un emplacement proposant le wifi. De même, le téléchargement peut s'effectuer par la connexion cellulaire si le wifi devient indisponible.

Numéro de port

QUIC est construit sur UDP, donc un champ de numéro de port de 16 bits est utilisé pour différencier les connexions entrantes.

Négociation de version

Une demande de connexion QUIC émanant d'un client indiquera au serveur la version du protocole QUIC qu'il souhaite utiliser, et le serveur répondra avec une liste des versions prises en charge que le client pourra sélectionner.

Les connexions utilisent TLS

Immédiatement après le paquet initial établissant une connexion, l'initiateur envoie une trame de cryptage qui commence à établir le handshake de couche sécurisée. La couche de sécurité utilise la sécurité TLS 1.3.

Il n'y a aucun moyen de vous désabonner ou d'éviter d'utiliser TLS pour une connexion QUIC. Le protocole est conçu pour être difficile à altérer par des boîtes intermédiaires, afin de prévenir l'ossification du protocole.

Flux

Les flux dans QUIC fournissent une abstraction légère, ordonnée et ordonnée.

Il existe deux types de flux de base dans QUIC:

- Les flux unidirectionnels transportent des données dans un sens: de l'initiateur du flux à son homologue.
- Les flux bidirectionnels permettent l'envoi de données dans les deux sens.

L'un ou l'autre type de flux peut être créé par l'un ou l'autre des points de terminaison, peut simultanément envoyer des données entrelacées à d'autres flux et peut être annulé.

Pour envoyer des données via une connexion QUIC, un ou plusieurs flux sont utilisés.

Contrôle de flux

Streams are individually flow controlled, allowing an endpoint to limit memory commitment and to apply back pressure. The creation of streams is also flow controlled, with each peer declaring the maximum stream ID it is willing to accept at a given time.

Les flux sont contrôlés individuellement, ce qui permet à un point de terminaison de limiter l’engagement de la mémoire et d’appliquer une contre-pression. La création de flux est également contrôlée en flux, chaque pair déclarant l’ID de flux maximum qu’il est prêt à accepter à un moment donné.

Identificateurs de flux

Les flux sont identifiés par un entier non signé de 62 bits, appelé ID de flux. Les deux bits les moins significatifs de l'ID de flux sont utilisés pour identifier le type de flux (unidirectionnel ou bidirectionnel) et l'initiateur du flux.

Le bit le moins significatif (0x1) de l'ID de flux identifie l'initiateur du flux. Les clients initient des flux pairs (ceux dont le bit le moins significatif est défini sur 0); les serveurs lancent des flux impairs (avec le bit mis à 1).

Le deuxième bit le moins significatif (0x2) de l'ID de flux différencie les flux unidirectionnels des flux bidirectionnels. Les flux unidirectionnels ont toujours ce bit défini sur 1 et les flux bidirectionnels ont ce bit défini sur 0.

Flux simultané

QUIC permet à un nombre arbitraire de flux de fonctionner simultanément. Une terminaison limite le nombre de flux entrants actifs simultanément en limitant l'ID de flux maximal.

L'ID de flux maximal est spécifique à chaque terminaison et s'applique uniquement à l'homologue qui reçoit

le paramètre.

Envoi et réception de données

Les terminaisons utilisent des flux pour envoyer et recevoir des données. C'est après tout leur but ultime. Les flux sont une abstraction ordonnée de flux d'octets. Les flux séparés ne sont toutefois pas nécessairement livrés dans l'ordre d'origine.

Priorité des flux

Le multiplexage de flux a un effet significatif sur les performances des applications si les ressources allouées aux flux sont correctement hiérarchisées. L'expérience acquise avec d'autres protocoles multiplexés, tels que HTTP/2, montre que des stratégies efficaces de hiérarchisation ont un impact positif significatif sur les performances.

QUIC lui-même ne fournit pas de frames pour l'échange d'informations sur la priorisation. Au lieu de cela, il repose sur la réception d'informations de priorité de l'application qui utilise QUIC. Les protocoles qui utilisent QUIC peuvent définir n'importe quel schéma de priorisation adapté à la sémantique de leurs applications.

Lorsque HTTP/3 est utilisé sur QUIC, la hiérarchisation est effectuée dans la couche HTTP.

0-RTT

Pour réduire le temps nécessaire à l'établissement d'une nouvelle connexion, un client déjà connecté à un serveur peut mettre en cache certains paramètres de cette connexion et établir une connexion **0-RTT** avec le serveur. Cela permet au client d'envoyer des données immédiatement, sans attendre la fin d'un handshake.

Spin Bit

Peut-être l'une des discussions de conception les plus longues au sein du groupe de travail QUIC, qui a fait l'objet de plusieurs centaines de mails et d'heures de discussions, concerne un seul bit: le Spin Bit.

Les partisans de ce bit insistent sur le fait qu'il est nécessaire que les opérateurs et les personnes se trouvant entre deux terminaisons QUIC puissent mesurer le temps de latence.

Les opposants à cette fonctionnalité n'aiment pas la potentielle fuite d'informations.

Faire tourner un bit

Les deux points de terminaison, le client et le serveur, conservent une valeur de rotation, 0 ou 1, pour chaque connexion QUIC, et définissent le bit de rotation sur les paquets qu'il envoie pour cette connexion sur la valeur appropriée.

Les deux côtés envoient ensuite des paquets avec ce bit de rotation défini sur la même valeur pendant toute la durée d'un aller-retour, puis la valeur est modifiée. L'effet est alors une impulsion de uns et de zéros dans ce champ binaire que les observateurs peuvent surveiller.

Cette mesure ne fonctionne que lorsque l'expéditeur n'est ni limité par l'application ni par le contrôle de flux, la réorganisation des paquets sur le réseau peut également rendre les données tumultueuses.

Espace utilisateur

L'implémentation d'un protocole de transport dans l'espace utilisateur permet une itération rapide du protocole, car il est relativement facile de faire évoluer le protocole sans nécessiter que les clients et les serveurs mettent à jour le noyau de leur système d'exploitation pour déployer de nouvelles versions.

Rien d'inhérent dans QUIC ne l'empêche d'être implémenté et proposé ultérieurement par les noyaux de systèmes d'exploitation, si quelqu'un trouve ça une bonne idée.

De nombreuses implémentations

Un des effets évidents de l'implémentation d'un nouveau protocole de transport dans l'espace utilisateur est que nous pouvons nous attendre à de nombreuses implémentations indépendantes.

Différentes applications sont susceptibles d'inclure (ou de superposer) différentes implémentations de HTTP/3 et de QUIC dans un avenir prévisible.

API

L'un des facteurs de succès du TCP classique et des programmes qui l'utilisent est l'API de socket standardisé. Il a des fonctionnalités bien définies et à l'aide de cette API, vous pouvez déplacer des programmes entre de nombreux systèmes d'exploitation différents, car TCP fonctionne de la même manière.

QUIC n'est pas là. Il n'y a pas d'API standard pour QUIC.

Avec QUIC, vous devez choisir l'une des implémentations de bibliothèque existantes et s'en tenir à son API. Cela rend les applications "verrouillées" à une seule bibliothèque dans une certaine mesure. Le passage à une autre bibliothèque signifie une autre API, ce qui peut nécessiter beaucoup de travail.

De plus, étant donné que QUIC est généralement implémenté dans l'espace utilisateur, il ne peut pas simplement enrichir l'API de socket ou sembler similaire à la fonctionnalité existante des protocoles TCP et UDP. L'utilisation de QUIC signifie l'utilisation d'une autre API que l'API socket.

HTTP/3

Comme mentionné précédemment, le premier et principal protocole pour transporter sur QUIC est HTTP.

Tout comme HTTP/2 a été introduit pour transporter HTTP sur le réseau d'une manière totalement nouvelle, HTTP/3 introduit encore une fois une nouvelle façon d'envoyer HTTP sur le réseau.

HTTP maintient toujours les mêmes paradigmes et concepts qu'avant. Il y a des en-têtes et un corps, il y a une demande et une réponse. Il y a les méthodes, les cookies et la mise en cache. Ce qui change principalement avec HTTP/3, c'est la manière dont les bits sont envoyés de l'autre côté de la communication.

Pour pouvoir utiliser HTTP sur QUIC, des modifications étaient nécessaires et le résultat de ceci est ce que nous appelons désormais HTTP/3. Ces modifications étaient nécessaires en raison de la nature différente fournie par QUIC par opposition à TCP. Ces changements incluent:

- Dans QUIC, les flux sont fournis par le transport lui-même, tandis que dans HTTP/2, les flux étaient créés dans la couche HTTP.
- Les flux étant indépendants les uns des autres, le protocole de compression d'en-tête utilisé pour HTTP/2 ne pourrait pas être utilisé sans provoquer une situation de tête de bloc.
- Les flux QUIC sont légèrement différents des flux HTTP/2. La section HTTP/3 le détaillera un peu.

URLs HTTPS://

HTTP/3 sera exécuté en utilisant des URLs `HTTPS://`. Le monde est rempli de ces URLs et il a été jugé peu pratique et tout à fait déraisonnable d'introduire un autre schéma d'URL pour le nouveau protocole. Tout comme HTTP/2 n'a pas besoin d'un nouveau schéma, HTTP/3 non plus.

La complexité supplémentaire de la situation de HTTP/3 réside toutefois dans le fait que, lorsque HTTP/2 était un nouveau moyen de transporter HTTP sur le réseau, il était toujours basé sur TLS et TCP comme l'était HTTP/1. Le fait que HTTP/3 soit effectué sur QUIC change les choses en quelques aspects importants.

Les anciennes, en texte clair, URLs `HTTP://`, seront laissées telles quelles et, au fur et à mesure que nous avançons dans le futur avec des transferts plus sécurisés, elles seront probablement de moins en moins utilisées. Les requêtes adressées à ces URLs ne seront tout simplement pas mises à niveau pour utiliser HTTP/3. En réalité, ils passent rarement à HTTP/2 non plus, mais pour d'autres raisons.

Connexion initiale

La première connexion à un hôte récent, non visité précédemment pour une URL HTTPS:// devra probablement être établie via TCP (éventuellement en plus d'une tentative parallèle de connexion via

QUIC). L'hôte peut être un ancien serveur sans prise en charge de QUIC ou il peut y avoir une boîte intermédiaire entre les obstacles empêchant le succès d'une connexion QUIC.

Un client et un serveur modernes négocieraient probablement HTTP/2 lors du premier handshake. Lorsque la connexion a été configurée et que le serveur répond à une requête HTTP du client, le serveur peut informer le client de sa prise en charge et de sa préférence pour HTTP/3.

Amorçage avec Alt-svc

L'en-tête du service de remplacement (Alt-svc:) et sa trame `ALT-SVC` HTTP/2 correspondante ne sont pas créés spécifiquement pour QUIC ou HTTP/3. Ils font partie d'un mécanisme déjà conçu et créé pour qu'un serveur indique à un client: "Regardez, je lance le même service sur CET HÔTE en utilisant CE PROTOCOLE sur CE PORT" *. Voir les détails dans la RFC 7838.

Un client qui reçoit une telle réponse Alt-svc est ensuite invité, s'il le prend en charge et le souhaite, à se connecter en arrière-plan en parallèle à cet hôte donné - à l'aide du protocole spécifié - et s'il réussit à basculer ses opérations sur cela au lieu de la connexion initiale.

Si la connexion initiale utilise HTTP/2 ou même HTTP/1, le serveur peut répondre et indiquer au client qu'il peut se reconnecter et essayer HTTP/3. Cela pourrait être vers même hôte ou à un autre qui sait comment servir cette origine. Les informations fournies dans une telle réponse Alt-svc ont un temporisateur d'expiration, permettant aux clients de diriger les connexions et demandes ultérieures directement vers l'hôte alternatif à l'aide du protocole alternatif suggéré, pendant une certaine période.

Exemple

Un serveur HTTP incluant une en-tête `Alt-Svc:` dans sa réponse:

```
1 Alt-Svc: h3=":50781"
```

Cela indique que HTTP/3 est disponible sur le port UDP 50781 avec le même nom d'hôte que celui utilisé pour obtenir cette réponse.

Un client peut ensuite essayer de configurer une connexion QUIC avec cette destination et, en cas de succès, continuer à communiquer avec l'origine comme cela au lieu de la version HTTP initiale.

Flux QUIC et HTTP/3

HTTP/3 étant conçu pour QUIC, il tire pleinement parti des flux de QUIC, où HTTP/2 devait concevoir l'ensemble de son concept de flux et de multiplexage au-dessus de TCP.

Les requêtes HTTP effectuées via HTTP/3 utilisent un ensemble spécifique de flux.

Trames HTTP/3

HTTP/3 signifie la configuration de flux QUIC et l'envoi d'un ensemble de trames à l'autre extrémité. Il n'y a qu'un petit nombre fixe (huit!) de trames connues dans HTTP/3. Les plus importants sont probablement:

- HEADERS, qui envoie des en-têtes HTTP compressés
- DATA, envoie le contenu des données binaires
- GOAWAY, veuillez arrêter cette connexion

Requête HTTP

Le client envoie sa requête HTTP sur un flux QUIC *bidirectionnel* initié par le client.

Une requête consiste en une seule trame HEADERS et peut éventuellement être suivie d'une ou deux autres trames: une série de trames DATA et éventuellement d'une trame HEADERS finale pour terminer.

Après avoir envoyé une requête, un client ferme le flux pour l'envoyer.

Réponse HTTP

Le serveur renvoie sa réponse HTTP sur le flux bidirectionnel. Une trame HEADERS, une série de trames DATA et éventuellement une dernière trame HEADERS.

En-têtes QPACK

Les trames HEADERS contiennent des en-têtes HTTP compressés à l'aide de l'algorithme QPACK, QPACK est styliquement similaire à celui de la compression HTTP/2 appelée HPACK (RFC 7541), mais modifiée pour fonctionner avec des flux livrés dans le désordre.

QPACK lui-même utilise deux flux QUIC unidirectionnels supplémentaires entre les deux terminaisons. Ils sont utilisés pour transporter des informations de table dynamique dans les deux sens.

Priorisation

Une des trames de flux HTTP/3 s'appelle `PRIORITY`. Elle est utilisée pour définir la priorité et la dépendance à un flux d'une manière similaire à celle de HTTP/2.

La trame peut définir un flux spécifique pour dépendre d'un autre flux spécifique et définir un "poids" sur un flux donné.

Des ressources dépendantes doivent se voir allouer des ressources que si tous les flux dont il dépend sont fermés ou s'il est impossible de progresser sur ces flux.

Un poids de flux est compris entre 1 et 256 et il est spécifié que les flux avec le même parent **devraient** se voir allouer des ressources proportionnellement en fonction de leur poids.

Push serveur

Un serveur push HTTP/3 est similaire à ce qui est décrit dans HTTP/2 (RFC 7540), mais utilise des mécanismes différents.

Un serveur push est en réalité la réponse à une requête que le client n'a jamais envoyée!

Les push serveur ne sont autorisés que si le côté client les a acceptés. Dans HTTP/3, le client définit même une limite pour le nombre de push qu'il accepte en informant le serveur de l'ID de flux de push maximal. Dépasser cette limite entraînera une erreur de connexion.

Si le serveur estime probable que le client souhaite une ressource supplémentaire qu'il n'a pas demandée mais qu'il devrait avoir de toute façon, il peut envoyer une trame `PUSH_PROMISE` (sur le flux de la requête) indiquant à quoi ressemble la requête dont la réponse est destinée, puis envoyer cette réponse réelle sur un nouveau flux.

Même si les envois ont au préalable été déclarés acceptables par le client, chaque flux envoyé individuellement peut toujours être annulé à tout moment si le client le juge approprié. Il envoie ensuite une trame `CANCEL_PUSH` au serveur.

Problématique

Depuis que cette fonctionnalité a été abordée pour la première fois dans le développement de HTTP/2 et ensuite plus tard, après que le protocole ai été livré et déployé sur Internet, cette fonctionnalité a été discutée, détestée et perfectionnée de nombreuses différentes manières afin de la rendre utile.

Un envoi n’est jamais "gratuit", car même s’il enregistre un demi aller-retour, il utilise toujours de la bande passante. Il est souvent difficile ou impossible pour le côté serveur de savoir avec un niveau élevé de certitude si une ressource doit être envoyée ou non.

Comparaison avec HTTP/2

HTTP/3 est conçu pour QUIC, qui est un protocole de transport qui gère les flux par lui-même.

HTTP/2 est conçu pour TCP et gère donc les flux dans la couche HTTP.

Similitudes

Les deux protocoles offrent aux clients des ensembles de fonctionnalités pratiquement identiques.

- Les deux protocoles offrent des flux
- Les deux protocoles offrent un support push serveur
- Les deux protocoles ont une compression d'en-tête, et QPACK et HPACK ont une conception similaire.
- Les deux protocoles offrent le multiplexage sur une seule connexion utilisant des flux
- Les deux protocoles établissent des priorités sur les flux

Differences

Les différences sont dans les détails et principalement là grâce à l'utilisation de QUIC par HTTP/3:

- HTTP/3 a plus de chances de fonctionner plus tôt grâce aux handshakes 0-RTT de QUIC, alors que TCP Fast Open et TLS envoient généralement moins de données et rencontrent fréquemment des problèmes.
- HTTP/3 a des handshakes beaucoup plus rapides grâce à QUIC vs TCP + TLS.
- HTTP/3 n'existe pas dans une version non sécurisée ou non chiffrée. HTTP/2 peut être implémenté et utilisé sans HTTPS - même si c'est rare sur Internet.
- HTTP/2 peut être négocié directement dans un handshake TLS avec l'extension ALPN, alors que HTTP/3 est sur QUIC donc nécessite une réponse d'en-tête `Alt-Svc:` pour informer le client de ce fait.

Critique générale

UDP ne fonctionnera jamais

Beaucoup d'entreprises, d'opérateurs et d'organisations bloquent ou limitent le débit du traffic UDP en dehors du port 53 (utilisé pour DNS) depuis qu'il a été depuis ces derniers jours principalement abusé pour des attaques. En particulier, certains des protocoles UDP existants et leur implémentations populaires pour serveur ont été vulnérables aux attaques par amplification dans lesquelles un attaquant peut générer une quantité considérable de trafic sortant afin de cibler des victimes innocentes.

QUIC dispose d’une atténuation intégrée contre les attaques d’amplification en exigeant que le paquet initial soit au minimum de 1200 octets et par une restriction dans le protocole qui stipule qu’un serveur NE DOIT PAS envoyer plus de trois fois ça en réponse sans recevoir un paquet du client en réponse.

UDP est lent dans les noyaux

Cela semble être la vérité, du moins aujourd'hui en 2018. Nous ne pouvons bien sûr pas dire comment cela va évoluer et à quel point cela est simplement le résultat des performances de transfert UDP qui ne sont plus au cœur des préoccupations des développeurs depuis de nombreuses années.

Pour la plupart des clients, cette "lenteur" n'est probablement même jamais perceptible.

QUIC prend trop de processeur

Semblable à remarque "UDP est lent" ci-dessus, c'est en partie du fait que l'utilisation du protocole TCP et TLS dans le monde a pris plus de temps pour se développer, s'améliorer et obtenir une assistance matérielle.

Il y a des raisons de s'attendre à ce que cela s'améliore avec le temps. La question est de savoir de combien et combien cette utilisation supplémentaire du processeur va faire mal aux déployeurs.

C'est juste Google

Non ça ne l'est pas. Google a communiqué les spécifications initiales à l'IETF après avoir prouvé, à grande échelle à l'échelle de l'Internet, que le déploiement de ce style de protocole sur UDP fonctionne actuellement et correctement.

Depuis lors, des membres d'un grand nombre d'entreprises et d'organisations ont travaillé au sein de l'organisation indépendante du fournisseur, l'IETF, pour élaborer un protocole de transport standard. Les employés de Google y ont bien sûr participé, tout comme les employés d'un grand nombre d'autres entreprises intéressées par l'état des protocoles de transport sur Internet, notamment Mozilla, Fastly, Cloudflare, Akamai, Microsoft, Facebook et Microsoft. Apple.

C'est une trop petite amélioration

Ce n'est pas vraiment une critique mais une opinion. Peut-être est-ce le cas, et c'est peut-être une amélioration trop minime, trop proche dans le temps depuis la publication de HTTP/2.

HTTP/3 fonctionnera probablement beaucoup mieux dans les réseaux saturés de pertes de paquets, il offre des handshakes plus rapide, ce qui améliore la latence réelle et perçue. Mais est-ce assez d'avantages pour motiver les gens à déployer la prise en charge HTTP/3 sur leurs serveurs et pour leurs services? Le temps et les mesures de performance futures nous le diront sûrement!

Les spécifications

Voici un recueil des dernières versions officielles des différentes parties et composants de QUIC et de

HTTP/3.

Invariants

Version-Independent Properties of QUIC

Transport

QUIC: A UDP-Based Multiplexed and Secure Transport

Récupération

QUIC Loss Detection and Congestion Control

TLS

Using Transport Layer Security (TLS) to Secure QUIC

HTTP

Hypertext Transfer Protocol (HTTP) over QUIC

QPACK

QPACK: Header Compression for HTTP over QUIC

QUIC v2

Afin de se concentrer au mieux sur le coeur du protocole QUIC et de pouvoir le livrer à temps, plusieurs fonctionnalités qui étaient prévues à l'origine pour faire partie du protocole principal ont été reportées et sont maintenant prévues pour être remplacées dans une prochaine version de QUIC. QUIC version 2 ou au-delà.

Cependant, l'auteur de ce document a une boule de cristal plutôt défectueuse, nous ne pouvons donc pas savoir exactement quelles fonctionnalités apparaîtront ou ne figureront pas dans la version 2. Nous pouvons

toutefois mentionner certaines des fonctionnalités et éléments explicitement supprimés du travail de la v1 pour être "travaillé plus tard" et pourraient alors éventuellement apparaître dans une version 2.

Forward Error Correction

La Correction d'Erreur Directe (en englais Forward Error Correction (FEC)) est une méthode d'obtention du contrôle d'erreur dans la transmission de données dans laquelle l'émetteur envoie des données redondantes et le récepteur ne reconnaît que la partie des données qui ne contient aucune erreur apparente.

FEC a été expérimenté par Google dans leur travail original sur QUIC, mais il a été retiré à nouveau car les expériences ne se sont pas bien déroulées. Cette fonctionnalité est un sujet de discussion pour QUIC v2, mais il faut probablement que quelqu'un prouve qu'elle peut en fait être un ajout utile sans pénalité excessive.

Multitrajet

Multitrajet (en anglais multipath) signifie que le transport peut lui-même utiliser plusieurs chemins d'accès réseau pour optimiser l'utilisation des ressources et augmenter la redondance.

Les partisans du SCTP dans le monde aiment mentionner que le SCTP est déjà multitrajet, tout comme le TCP moderne.

Données non fiables

Il a été envisagé de proposer comme option des flux "non fiables", qui permettraient alors à QUIC de remplacer également les applications de type UDP.

Adaptations non-HTTP

DNS sur QUIC est l'un des premiers protocoles non HTTP mentionnés qui pourrait attirer l'attention une fois que QUIC v1 et HTTP/3 seront disponibles. Mais avec un nouveau moyen de transport comme celui-ci ayant été apporté au monde, je ne peux pas imaginer que cela s'arrêtera là.

Italiano

Lo slancio per scrivere questo libro è partito in Marzo 2018. Il piano consiste nel documentare HTTP/3 e il protocollo sottostante, ossia QUIC. Perchè sono stati concepiti, come funzionano, dettagli sul protocollo,

implementazioni, etc.

Questo libro è inteso per essere libero, gratuito; consta dello sforzo condiviso che include chiunque abbia voglia di contribuire.

Prerequisiti

Al lettore di questo libro è richiesto di avere una comprensione basilare di Reti TCP/IP, di HTTP e del mondo web. Per maggiori dettagli e specifiche a proposito di HTTP/2 vi raccomandiamo di consultare preventivamente le informazioni contenute su http2 explained.

L'autore

Il presente libro è stato ideato e scritto da Daniel Stenberg . Daniel è fondatore e sviluppatore principale di curl, il client HTTP più usato al mondo. Daniel ha lavorato con e per HTTP ed altri protocolli internet per più di due decenni ed è l'autore di http2 explained.

Homepage

La homepage del libro si trova su daniel.haxx.se/http3-explained .

Contribuire

Se dovessi trovare errori, omissioni o falsità in questo documento, perfavore inviaci una versione aggiornata del paragrafo e ci assicureremo che la modifica venga pubblicata. Attribuiremo il credito a chiunque si renda utile. Spero di continuare a migliorare questo libro nel corso del tempo.

Preferibilmente, segnalare errori o richieste pull sulla pagina GitHub del progetto.

Licenze

Perchè QUIC

QUIC è un nome, non un acronimo. Si pronuncia esattamente come la parola Inglese "quick" (veloce).
QUIC può essere visto come un nuovo protocollo sicuro in grado di stablire un trasporto adatto a trasportare semantiche simil-HTTP e che possa risolvere alcune delle limitazioni di HTTP/2 su TCP e TLS. Rappresenta un passo avanti nell'evoluzione del trasporto web.

QUIC non si limita al solo trasporto HTTP. Il desiderio di rendere il web e la distribuzione dei contenuti sempre più veloce agli occhi degli utenti è probabilmente la ragione principale ad aver stimolato la creazione di questo nuovo protocollo di trasporto.

Perchè creare un nuovo protocollo di trasporto e perchè mai basarlo su UDP ?

Logo di QUIC

Ricordi HTTP/2 ?

La specifica HTTP/2 RFC 7540 è stata pubblicata nel Maggio 2015 e da allora il protocollo è stato implementato e largamente distribuito attraverso l'intero Internet e il world wide web.

Ad inizio 2018, quasi il 40% dei top-1000 siti web funziona in HTTP/2: circa il 70% di tutte le richieste HTTPS inviate da Firefox ha ricevuto risposte di tipo HTTP/2. Tutti i principali server, proxy e browser supportano h2.

HTTP/2 cerca di risolvere svariati limiti intrinsechi ad HTTP/1; con l'introduzione di tale seconda versione di HTTP gli utenti non sono più obbligati ad utilizzare work-around e gabole complicate. Tali sotterfugi caricano inutilmente gli sviluppatori di responsabilità non previste.

Una delle funzioni principali in HTTP/2 è il multiplexing, tecnica che permette di inviare molteplici flussi logici (indipendenti) all'interno di una stessa connessione TCP. Ciò rendre il tutto più rapido e fluido. Fa si che il controllo di flusso sia applicato in maniera più efficace, permette agli utenti di sfruttare il TCP appieno,

di saturare la banda, di rendere le connessioni TCP più durature -il che permette nella maggior parte dei casi di raggiungere la massima velocità. La compressione degli header permette di risparmiare banda.
Con HTTP/2, i browser usano tipicamente *una* connessione TCP per ogni host rispetto alle precedenti *sei*. In effetti, tecniche come il "desharding" e la condensazione ("coalescing") delle connessioni HTTP/2 potrebbero ridurre ulteriormente il numero di connessioni totali.

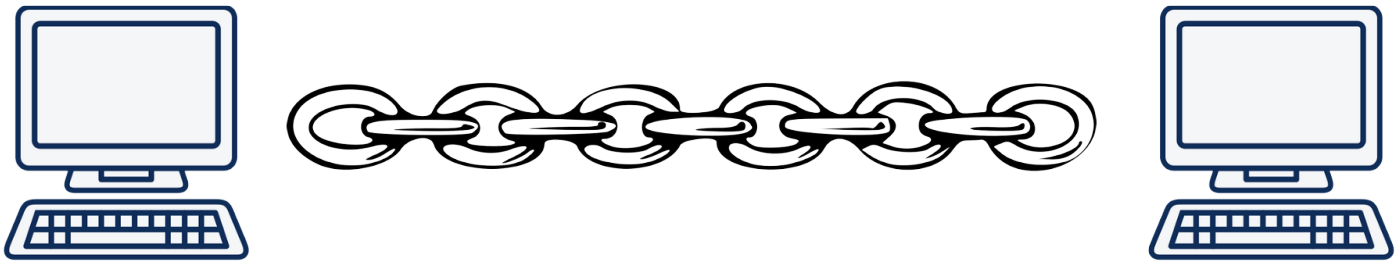
HTTP/2 risolve il problema del "bloccaggio di inizio fila": il client era costretto ad aspettare la fine dell'elaborazione della richiesta in corso prima di poter prendere in carico la risposta alla domanda in attesa. Ora non più.

http2 man

Blocco ad inizio linea, "TCP head of line blocking"

Bloccaggio ad inizio della coda TCP

HTTP/2 è comunque ancora basato su TCP seppur utilizzi un minor numero di connessioni rispetto ai predecessori. Il TCP è un protocollo di trasferimento affidabile; lo si può immaginare come una corda immaginaria tesa fra due host. Ciò che viene appeso ad un estremo arriverà all'altro, nello stesso ordine - a costo dell'interruzione della connessione.

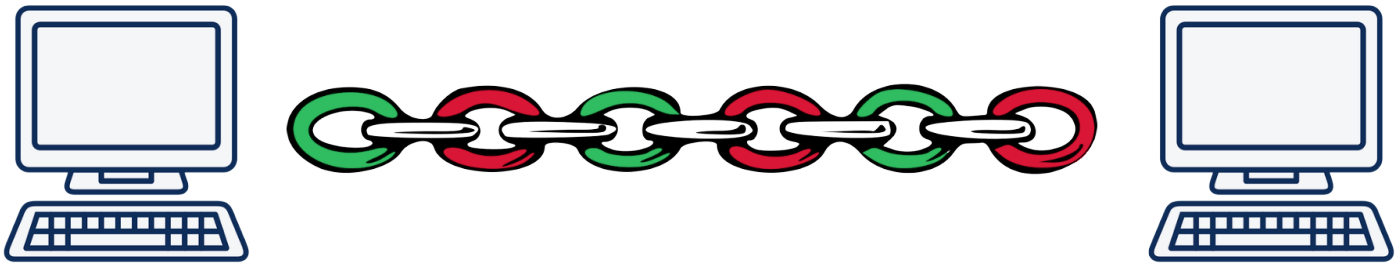


a TCP chain between two computers

Con HTTP/2, il browser avrà la tendenza a soddisfare decine o anche centinaia di trasferimenti attraverso la stessa connessione TCP.

Se un singolo pacchetto venisse perso, abbandonato o scartato da qualche parte sulla rete, fra due endopoints che stessero dialogando in HTTP/2, tale dialogo verrebbe sospeso fino alla ri-trasmissione e all'arrivo di tale pacchetto a destinazione. Osservando la catena con cui è raffigurato il TCP, capiamo come - in caso di interruzione del link- tutti i dati che seguono il pacchetto in sospeso dovranno attendere.

Una illustrazione della metafora della catena nel momento in cui due flussi siano trasmessi sulla medesima connessione. Un flusso in rosso, uno in giallo:



the chain showing links in different colors

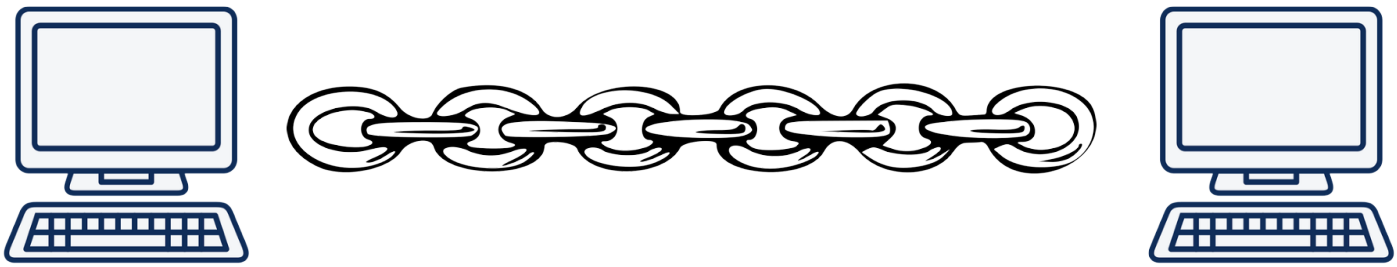
Si parla quindi di bloccaggio ad inizio della coda, basato su TCP!

Se il tasso di perdita di pacchetti dovesse incrementare, HTTP/2 subirebbe un degrado di prestazioni. A 2% di perdita di pacchetti (che comunque rappresenta una qualità infima) svariati test hanno dimostrato come gli utenti di HTTP/1 ottengano risultati di gran lunga migliori, stimando che un tasso di perdita del 2% distribuito su sei connessioni abbia meno influenza che su una sola, permettendo così alle rimanenti connessioni di proseguire senza intralcio.

Porre rimedio a tale empasse non sarà semplice -se non impossibile- finchè si continuerà ad usare il TCP..

L'impiego di flussi indipendenti evita il blocco

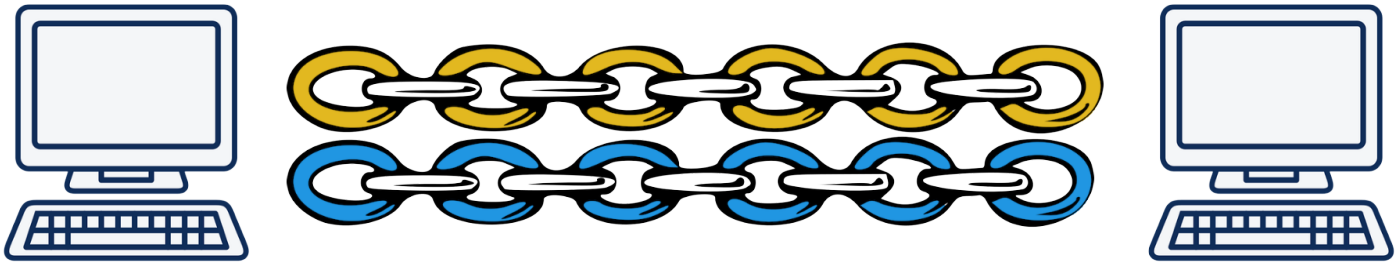
QUIC è composto da una fase di setup della connessione fra i due estremi, fase che rende la connessione sicura e la consegna dei dati affidabile.



una catena QUIC fra due computers

Seppur instradati su una stessa connessione, i due flussi sono trattati in maniera indipendente tanto che se uno dei due collegamenti fosse interrotto, solo il flusso in questione sarà obbligato ad attendere la ri-trasmissione della risorsa perduta, senza alcun impatto sull'altro flusso parallelo.

Qui di seguito illustrati i due flussi (streams) scambiati dai due end-ponts, uno in blu ed uno in giallo.



due flussi QUIC fra due computers

TCP o UDP

TCP o UDP

Dato che non possiamo completamente mettere fine al bloccaggio ad inizio fila all'interno di TCP, dovremmo almeno essere in grado di sviluppare un protocollo di trasporto più vicino a UDP e TCP nello stack di rete. O magari impiegare SCTP che è un protocollo di trasporto standardizzato dalla IETF nella RFC 4960 , contenente la maggior parte delle caratteristiche o funzioni necessarie allo scopo.

Al contrario, nel corso degli ultimi anni, tutti gli sforzi per progettare un nuovo protocollo di trasporto sono stati sospesi a causa delle difficoltà percepite nel distribuire tali protocolli all'interno di infrastrutture reali. La quantità di firewalls, NATs, routers e altri dispositivi di rete che si iterpongono fra l'utente e il server sono semplicemente troppi, e sono a conoscenza dei soli UDP o TCP. Introdurre un altro protocollo di trasporto fa si che un determinato tasso di connessioni N% fallisca, in quanto bloccato dalle suddette middle-boxes che identificherebbero tale proto come malevolo, malformato, nè TCP nè UDP. Un tasso di fallimento N% in ambito internet rappresenta spesso una buona ragione per non intraprendere alcuno sforzo.

In aggiunta, cambiare le cose a livello di "transport protocol layer" nello stack di rete può spesso significare dover mettere mano al kernel del SO. Modificare il kernel è un processo che richiede tempi lunghi, grandi sforzi e non al riparo da pressioni. In anni recenti, abbiamo visto features di TCP diventare standard IETF senza che vi sia stata corrispondenza -ad anni di distanza- con il tasso di distribuzione degli stessi, per via della mancanza di utilizzo o di supporto universale.

Perchè no, SCTP-over-UDP

SCTP è un protocollo di trasporto di flussi affidabile, ed esistono persino alcune implementazioni in ambito WebRTC che utilizzano SCTP via UDP.

Non fu dunque scelto come valida base per QUIC per via di molteplici ragioni, ossia:

- SCTP non risolve il problema del "blocco di inizio riga" con i flussi
- SCTP richiede una decisione sul numero di streams nella fase di inizializzazione

- SCTP non ha un passato particolarmente brillante in quanto a sicurezza TLS
- SCTP utilizza una negoziazione a 4 vie, mentre QUIC ne offre una da 0-RTT
- QUIC è un flusso di bytes ordinato al pari di TCP, SCTP un protocollo a messaggi
- Una connessione QUIC può essere migrata fra indirizzi IP, ma SCTP non lo permette

Per maggiori dettagli relativi alle differenze, vedere Un raffronto tra SCTP e QUIC .

Ossificazione

Internet è una rete fatta di reti. In diversi punti del mondo -sulla rotta fra le varie reti interconnesse- sono installati dispositivi che assicurano il buon instradamento e funzionamento della rete. Questi dispositivi -i varii componenti distribuiti della rete- sono comunemente chiamati "middle-boxes". Scatole che semplicemente si trovano in mezzo al percorso fra i due end-points e che a loro volta sono coinvolte nel trasferimento dati via rete.

Questi apparecchi servono molteplici funzioni ma limitiamoci semplicemente a concludere che se qualcuno li ha installati in tali specifici punti della rete, avrà sicuramente avuto i suoi buoni motivi.

Tali "middle-boxes" instradano i pacchetti IP fra le differenti reti globali, bloccano il traffico maligno, si occupano di tradurre gli indirizzi di rete (NAT - Network Address Translation), aumentano le prestazioni, tentano in certi casi di spiare il traffico passante, ed altro ancora.

Al fine di espletare i propri compiti, questi dispositivi devono avere conoscenza di "networking" e dei protocolli oggetto del traffico stesso. Appositi software sono sviluppati allo scopo, software che tuttavia non ricevono aggiornamenti troppo frequenti.

A parte essere ovviamente i "collanti" che tengono insieme Internet, questi dispositivi al cuore della rete non sono mai altrettanto avanzati in termini di tecnologia quanto quelli agli estremi della rete, ossia clients e servers.

Tutti protocolli di rete che queste macchine potrebbero voler ispezionare e di cui potrebbero voler decidere la sorte in base a origine o contenuto, subiscono questo problema: tali macchine sono state installate in conformità con le opzioni di protocollo disponibili all'epoca e non sono flessibili. Aggiunte o modifiche del comportamento non note in precedenza potrebbero fare si che il dispositivo interpreti in maniera non corretta il pacchetto, scartandolo per malformazione o contenuto ritenuto erroneamente illecito. Tale traffico subirebbe rallentamenti o drop improvvisi al punto da rendere tali nuove opzioni di protocollo indesiderabili da un punto di vista utente.

Siamo di fronte a ciò che possiamo definire "ossificazione del protocollo".

Anche le modifiche al TCP soffrono di ossificazione: alcune delle nuove opzioni TCP sono interpretate e bloccate in quanto sconosciute ai più. Se viene permesso al dispositivo di ispezionare in dettaglio il protocollo, il sistema tederà a definire pattern standard di comportamento per un determinato protocollo e nel tempo diventerà difficile se non appunto impossibile modificare tali assunzioni.

La sola vera ed efficace strategia per il contrasto della "ossificazione" è utilizzare in maniera estensiva la cifratura, cosi da impedire ai box intermedi di desumere (distinguere) quale protocollo stia passando sotto il loro ponte.

Sicuro

QUIC è sempre sicuro. Non esiste alcuna versione clear-text del protocollo, al punto in cui l'aver negoziato QUIC implica l'impiego di crittografia e sicurezza TLS 1.3. Come detto in precedenza, tale pratica impedisce l'ossificazione ed ogni altra sorta di blocco o trattamento speciale, oltre ad assicurare che QUIC rispetti tutte le proprietà di sicurezza HTTPS che gli utilizzatori web oggigiorno si attendono e desiderano.

Solo una parte dei primissimi pacchetti della negoziazione iniziale sono trasmessi in chiaro, prima che il protocollo di cifratura venga negoziato.

Latenza ridotta

QUIC offre entrambe le negoziazioni 0-RTT e 1-RTT al fine di minimizzare il tempo necessario alla negoziazione e setup della connessione. Paragoniamolo all'handshake a 3 vie del TCP.

In aggiunta a ciò QUIC mette a disposizione il supporto per "dati anticipati" sin da subito, per permettere un flusso di dati più intenso, oltre a risultare di semplice impiego rispetto all'opzione "TCP Fast Open"

Con il concetto di stream, un'altra connessione logica verso uno stesso host può essere effettuata direttamente senza aspettare che la prima abbia finito.

Il "TCP Fast Open" è problematico

TCP FO è stato pubblicato come RFC 7413 nel Dicembre 2014; tale specifica descrive come le applicazioni possano inviare dati al server gia a partire dal primo pacchetto TCP SYN.

Arrivare ad un supporto mondiale per tale opzione ha necessitato lungo tempo, e siamo ancora vittime di malfunzionamenti nel 2018. Coloro i quali hanno implementato lo stack TCP hanno evidenziato svariati problemi, cosi come le applicazioni che tentavano di trare vantaggio da tale caratteristica - sia nel comprendere su quali versioni del Sistema Operativo attivarla sia nel tentare di risolvere problemi inerenti la mancanza di supporto lato client (backdown). Molte reti sono state segnalate per l'interferenza al traffico TFO, ed hanno quindi inficiato il buon funzionamento della negoziazione TCP.

Evoluzione

La versione iniziale di QUIC è stata proposta da Jim Roskind di Google, fu implementata nel 2012 e successivamente annunciata al pubblico nel 2013, momento in cui la sperimentazione di Google si espanse fortemente.

A quei tempi l'acronimo QUIC veniva espanso in "Quick UDP Internet Connections" ma fu successivamente abbandonato.

Google ha gestito l'implementazione del protocollo e il successivo deploy su larga scala, tramite il famoso browser Chrome e su tutti i servizi web più famosi, fra cui search, gmail e youtube. Google è stata responsabile per lo sviluppo di numerose versioni del protocollo ed ha dimostrato la fattibilità del progetto utilizzando una vasta porzione di utenti Internet.

Nel Giugno 2015, la prima bozza internet per QUIC fu inviata ad IETF per la standardizzazione ma fu necessario attendere il tardo 2016 prima di osservare la formazione del gruppo di lavoro. Moltissimi vendor e attori del settore IT hanno presto contribuito ad accrescere l'interesse per il nuovo standard.

Nel 2017 i numeri mostrati dagli ingegnieri di QUIC a Google evidenziano come ben il 7% di *tutto* il traffico Internet transiti gia su questo protocollo. La versione "Google" del protocollo.

IETF

Il gruppo di lavoro QUIC, creato per standardizzare il protocollo all'interno di IETF, decise che QUIC avrebbe dovuto essere in grado di veicolare altri protocolli oltre il "semplice" HTTP. Il QUIC-Google si era finora occupato solo di HTTP, più specificamente di frames HTTP/2, servendosi della sintassi gia disponibile per i frames HTTP/2.

Fu inoltre deciso che il QUIC-IETF dovesse basare il proprio framework di sicurezza e cifratura sullo standard TLS 1.3 al posto delle modifiche "fatte in casa" dal team Google.

Per soddisfare la volontà di veicolare "non solamente HTTP", l'architettura QUIC di IETF fu separata in due livelli: il trasporto QUIC e la parte "HTTP over QUIC" (riferendosi al suddetto livello utilizziamo anche il termine "hq").

Questa ultima separazione -innoqua quanto possa sembrare- ha creato sostanziose differenze fra le versioni originali Google e la versione finale IETF.

Il gruppo di lavoro ha comunque deciso abbastanza presto di concentrarsi sulla consegna -entro i termini preposti- della versione 1 di QUIC, ed ha quindi prediletto lo sviluppo di HTTP, rinviando così lo sviluppo del trasporto non-HTTP ad una seconda fase.

Quando abbiamo iniziato a lavorare allo sviluppo di questo libro nel Marzo 2018 l'idea era di consegnare le specifice di QUIC versione 1 verso Novembre 2018, data finalmente riportata a Luglio 2019.

Mentre il lavoro della IETF è avanzato, il team Google ha integrato i dettagli della nuova versione IETF ed ha iniziato ad avanzare progresivamente nella direzione di ciò che potrebbe finalmente diventare lo standard IETF. Google ha quindi continuato ad usare la propria versione di QUIC sia lato browser sia lato server applicativi.

Le più recenti implementazioni in via di sviluppo hanno deciso di focalizzare gli sforzi in direzione della versione IETF ufficiale; non sono perciò compatibili con le versioni proposte da Google.

Esperienza da HTTP/2

La specifica HTTP/2 contenuta nella RFC 7540 è stata pubblicata nel Maggio 2015, un mese prima che QUIC fosse presentato ad IETF per la prima volta.

Con HTTP/2, le fondamenta per modificare HTTP furono gettate; il gruppo di lavoro responsabile della creazione di HTTP/2 era gia dell'idea che tale processo di rinnovamento avrebbe aiutato a rilasciare versioni HTTP più rapidamente che in passato, dove il passaggio da v1 a v2 richiese 16 anni.

Nel contesto di HTTP/2 gli utenti ed i software realizzano come non sia più più d'attualita lavorare con un HTTP testuale, in maniera serializzata.

HTTP-over-QUIC fu rinominato HTTP/3 in Novembre 2018.

Status

Il gruppo di lavoro QUIC ha lavorato alacremente a partire dal tardo 2016 ad una specifica di protocollo e la sua volontà è di completare tale sforzo entro Luglio 2019.

A Novembre 2018, ancora non abbiamo notizia di test di interoperabilità estesi su HTTP/3 - solo fra le due implementazioni esistenti che tuttavia non sono guidate da un browser ne da un software opensource lato server.

Esistono all'incirca quindici Implementazioni di QUIC nella lista wiki del gruppo di lavoro, ma ovviamente non tutte possono operare allo stesso livello dell'ultima bozza delle specifiche.

Implementare QUIC non è affatto semplice, dato che è in continua evoluzione anche quando sembra stabilizzatosi.

Servers

Il supporto per QUIC e HTTP/3 in NGINX è in fase di sviluppo. Il rilascio delle nuove funzionalità è previsto durante il ciclo di sviluppo di NGINX 1.17.

Non vi è alcun comunicato pubblico rispetto al supporto di QUIC in Apache.

Clients

Nessun produttore di web-browser ha rilasciato una versione che sia in grado di utilizzare la versione IETF di QUIC o HTTP/3.

Google Chrome contiene la sua propria implementazione di QUIC in salsa Google da svariati anni; tale versione Google non è compatibile con le bozze IETF e l'implementazione di HTTP differisce dalla versione HTTP/3 ufficiale.

Mozilla sta sviluppando Neqo - un'implementazione di QUIC e HTTP/3 scritta in Rust. Neqo sarà integrato in Necko (una libreria di rete usata in molte applicazioni di Mozilla - incluso Firefox).

curl ha rilasciato un primo supporto sperimentale per HTTP/3 (draft-22) con la release 7.66.0 dell' 11 Settembre 2019. Per l'implementazione di HTTP/3, curl può usare sia la libreria Quiche di Cloudflare che la famiglia di librerie ngtcp2.

Ostacoli all'implementazione

Per non dover re-inventare la ruota e poter contare su standard affermati, QUIC ha deciso di utilizzare TLS 1.3 come fondazione per gli strati di sicurezza e di crittografia. Tuttavia, nel fare ciò, il gruppo di lavoro ha infine riadattato l'utilizzo del protocollo TLS con QUIC, utilizzando solamente "messaggi TLS" e mai "records TLS".

Quello che potrebbe sembrare un'innocente modifica, in realtà ha causato non pochi grattacapi a chi era in procinto di implementare uno stack QUIC. Le librerie TLS esistenti con supporto a TLS 1.3 non dispongono di un numero sufficiente di API per esporre tale funzionalità e permettere quindi a QUIC di accedervi. Se è vero che molti degli sviluppatori/implementatori di QUIC appartengono a grandi organizzazioni che in parallelo sviluppano il proprio stack TLS, ciò non è esattamente vero per tutti.

Ad esempio OpenSSL -leader in campo opensource- non fornisce alcuna API per controllare tale funzione. Il piano per risolvere questo problema è delineato nella PR 8797, che punta ad introdurre delle API simili a quelle di BoringSSL.

Questa situazione porterà ad incontrare ostacoli nella messa in campo della tecnologia QUIC, visto che i differenti stack dovranno basarsi su librerie TLS proprie o di terze parti, utilizzare versioni di OpenSSL modificate o necessitare di aggiornamenti vari.

Kernel e carico sulla CPU

Google e Facebook hanno entrambi affermato che le loro installazioni QUIC su larga scala consumano circa il doppio della CPU per servire la stessa quantità di contenuti rispetto ad una più classica connessione HTTP/2 su TLS e TCP.

Spiegazioni plausibili a questo fenomeno includono:

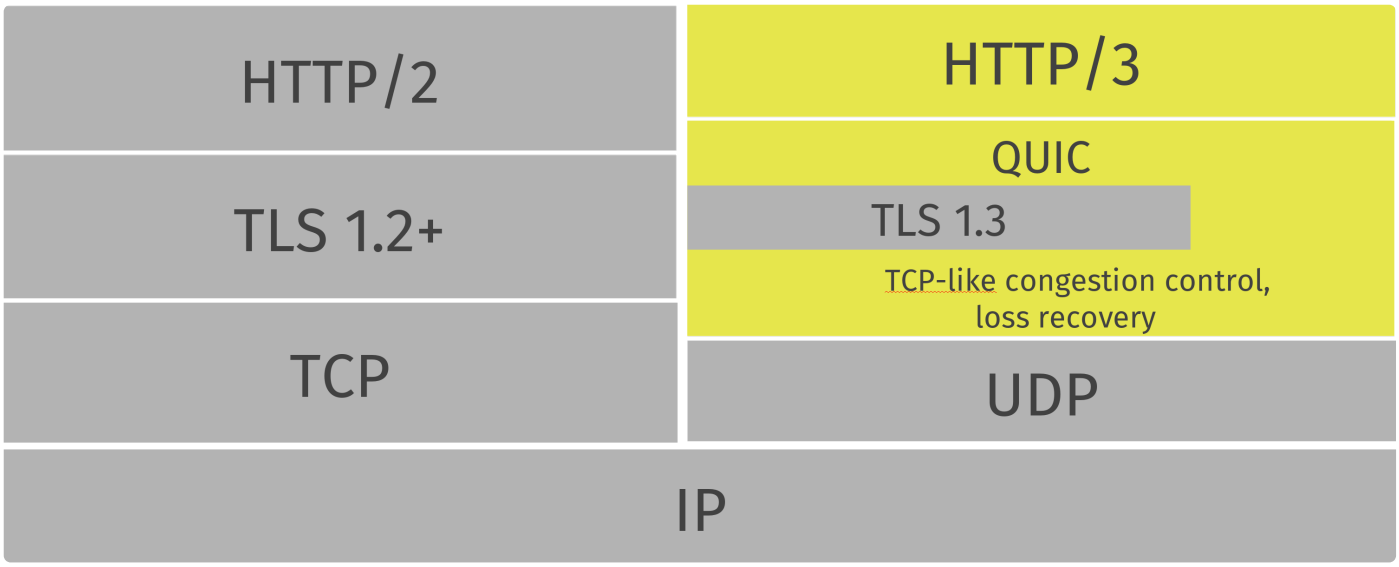
- il fatto che la parte UDP nel kernel Linux non sia altrettanto ottimizzata quanto lo stack TCP, dato che UDP non era finora mai stato utilizzato per trasferimenti ad altissima velocità come questi
- il fatto che per QUIC non esistano dispositivi di accelerazione hardware e offloading come invece è il caso per TCP e TLS. Notiamo rarissime eccezioni per dispositivi dedicati ad UDP.

Non vi sono ragioni per non essere convinti che le performance aumenteranno ed i requisiti CPU si abbasseranno nel tempo.

Caratteristiche del protocollo

Il protocollo QUIC ad alto livello.

Illustrati di seguito, lo stack HTTP/2 sulla sinistra e lo stack rete QUIC sulla destra, quando usati per il trasporto HTTP.



QUIC logo

UDP

Protocollo di trasporto via UDP

QUIC è un protocollo di trasporto implementato su UDP. Se si osserva il traffico di rete con un analizzatore di protocollo, il traffico QUIC appare composto di pacchetti UDP.

Basato su UDP, utilizza anche i numeri di porta per identificare con precisione un determinato servizio di rete su uno specifico indirizzo IP.

Tutte le implementazioni correnti e conosciute di QUIC sono sviluppate in "user-space" il che permette una evoluzione più rapida rispetto ad una tipica implementazione a livello di kernel-space.

Funzionerà?

Ci sono aziende e altri setup di rete che bloccano il traffico UDP sulle porte diverse dalla 53 (DNS). Altri rallentanto i flussi in maniera da influenzare negativamente le prestazioni di QUIC, rendendolo piu lento che il TCP. Non vi è limite alle manipolazioni che gli operatori di rete possono applicare.

Nel futuro prossimo, tutti gli impieghi di trasporto basato su QUIC dovranno prevedere una modalita di fall-back indolore verso una modalità di trasporto alternativa (basata su TCP). Gli ingegneri di Google hanno potuto misurare il tasso di fallimento in percentuali a cifra singola, fra 2 e 5%.

Apporterà miglioramenti?

Ci sono grosse probabilità che QUIC sia di valido aiuto al mondo Internet, chiunque vorrà poterlo usare e ci si aspetterà di vederlo funzionare con grande efficacia, fino al punto in cui le aziende cominceranno a considerare il cambiamento. Negli anni e grazie agli sviluppi apportati a QUIC, il tasso di successo nello stabilire connessioni QUIC è nettamente aumentato.

Affidabile

Mentre sappiamo che UDP non è un modo di trasporto affidabile, notiamo come QUIC aggiunga uno strato di controllo, garantendo infine l'affidabilità. Offre retrasmissione di pacchetti, controllo della congestione, regolazione di flusso e altre caratteristiche gia presenti nel TCP.

I dati spediti attraverso QUIC da un end-point saranno trasportati prima o poi all'altro capo della connessione, tanto che quest'ultima sia mantenuta.

Streams

Similmente a SCTP, SSH e HTTP/2, QUIC sfrutta flussi logicamente separati all'interno di una stessa connessione fisica. Un numero di stream paralleli può trasferire simultaneamente dati sulla stessa connessione senza influenzare gli altri flussi concorrenti.

Una connessione viene negoziata e messa in opera fra due punti remoti, in modo similare a quanto avviene per TCP. Una connessione QUIC è stabilita verso una porta e un indirizzo IP, tuttavia una volta stabilita la connessione viene identificata da un ID detto "ID di connessione".

All'interno di una connessione gia stabilita, ognuno dei due estremi può creare un flusso e spedire dei dati all'altro estremo. Un singolo flusso viene consegnato "in ordine" ed è ritenuto affifabile, mentre altri flussi paralleli potrebbero eventualmente essere ricevuti "fuori ordine".

QUIC offre un controllo di flusso a livello di connessione e di flusso.

Maggiori dettagli nelle sezioni connessioni e streams.

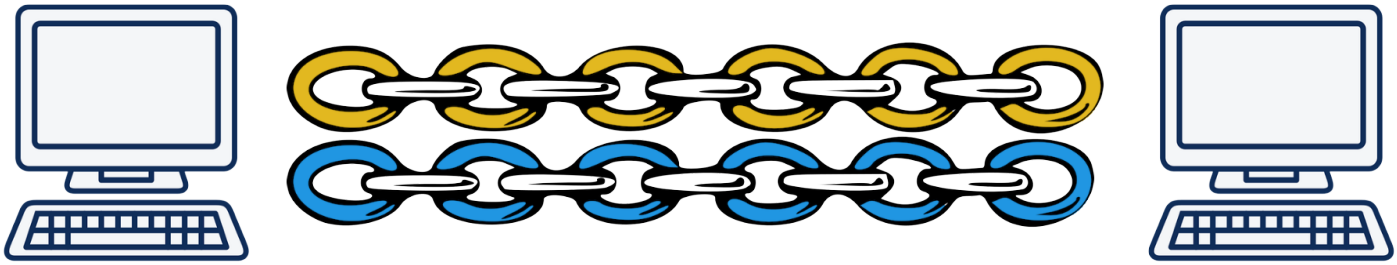
Ordinato

QUIC garantisce una consegna ordinata dei flussi, ma non fra i flussi stessi. Questo significa che ogni flusso invierà dati e manterrà un ordine fra tali dati, ma ogni singolo flusso potrebbe raggiungere la destinazione in un ordine diverso da quello in cui l'applicazione lo avesse inizialmente spedito!

Per esempio: lo stream A e B sono trasferiti dal server al client. Il flusso A inizia per primom il B segue. In QUIC, la perdita di un pacchetto influenza solamente lo stream al quale tale pacchetto perso appartiene. Se

uno stream A perdesse un pacchetto ma lo stream B no, il flusso B continuerebbe il proprio trasferimento mentre il pacchetto perso dallo stream A verrebbe ritrasmesso. Ciò non era possibile in HTTP/2.

Qui di seguito illustrati in giallo e blu due flussi inviati attraverso QUIC all'interno di una singola connessione. Essi sono indipendenti e potrebbero arrivare in un ordine diverso da quello inizialmente previsto, pur venendo entrambi consegnati correttamente a livello di applicazione, in ordine.



due flussi QUIC tra due computer

Negoziazioni veloci

QUIC offre entrambi i metodi di connessione 0-RTT e 1-RTT, nel senso che nel migliore dei casi QUIC non ha bisogno di round-trips aggiuntivi per iniziare una nuova connessione. Il più veloce dei due metodi, la negoziazione a 0-RTT, funziona solo se vi è gia stata una connessione fra i due host in question e solo se il "segreto" per tale connessione è stato aggiunto alla cache.

Dati in anticipo

QUIC permette ai client di aggiungere dati gia a partire dalla fase di negoziazione a 0-RTT. Questa caratteristica permette ad un client di spedire dati al proprio interlocutore il più veloce possibile; ciò permette al server di rispondere aggiungendo altri dati ancora più velocemente.

TLS 1.3

La sicurezza del trasporto utilizzato da QUIC si basa su TLS 1.3 (RFC 8446) quindi non esistono connessioni QUIC non criptate.

TLS 1.3 presenta molteplici vantaggi se paragonato alle versioni precedenti; una delle principali ragioni dell'adozione di TLS da parte di QUIC consiste nel fatto che la 1.3 ha modificato la modalità di negoziazione al fine di richiedere meno va-e-vieni (round-trips). Ciò riduce la latenza del protocollo.

La versione oramai "legacy" di Google-QUIC utilizzava un algoritmo di cifratura personalizzato, non-standard.

Trasporto e applicazione

Il QUIC di IETF è un protocollo di trasporto, sul quale altri protocolli applicativi possono essere innestati. L'applicazione principale (iniziale) di QUIC è al momento HTTP/3 (h3).

Il livello di trasporto supporta sia connessioni sia flussi.

La versione di QUIC sviluppata da Google presentava i layers di trasporto e applicazione (HTTP) come fusi insieme all'interno di un singolo processo, suonava molto come la "versione-speciale-che-manda-frames-http/2-via-udp".

HTTP over QUIC

Il layer HTTP si occupa dei trasporti in stile HTTP, inclusa la compressione degli header utilizzando QPACK - simile alla compressione usata in ambito HTTP/2, detta HPACK.

L'algoritmo HPACK si basa sulla consegna *ordinata* dei flussi quindi non è stato possibile riciclarlo per HTTP/3 senza apportarvi modifiche, visto che QUIC permette la consegna di flussi al di fuori dell'ordine predefinito. QPACK può essere visto come una versione di HPACK adattata su misura per QUIC.

Non-HTTP over QUIC

Il lavoro sul tema dell'incapsulazione di protocolli non-HTTP dentro QUIC è stato posticipato: il tema sarà sviluppato in seguito alla consegna della prima versione ufficiale di QUIC.

Come funziona QUIC

Senza voler dettagliare esattamente bits e bytes °on the wire°, questa sezione si occupa di descrivere i blocchi fondamentali al funzionamento del protocollo di trasporto QUIC. Se vuoi implementare QUIC nel tuo proprio stack, questa descrizione dovrebbe esserti d'aiuto a comprenderne le basi; per maggiori dettagli meglio fare riferimento alle bozze ufficiali e alle RFC pubblicate da IETF.

1. Inizializzare una connessione
2. ... che includa sicurezza TLS
3. Infine usare i flussi

Connessioni

Una connessione QUIC è una conversazione singola fra due end-points QUIC. La modalità di inizializzazione della connessione QUIC combina la negoziazione della versione con le negoziazioni dell'algoritmo crittografico e del trasporto al fine di ridurre la latenza di setup della connessione.

Per essere in grado di inviare dati attraverso tale connessione, uno o più flussi devono essere istanziati ed utilizzati.

ID della connessione

Ogni connessione dispone di un insieme di identificatori di connessione, ognuno dei quali può essere utilizzato per identificare la connessione stessa. Gli ID sono selezionati indipendentemente da ognuna delle estremità; ogni endpoint assegna un ID alla connessione con il suo corrispondente.

La funzione primaria dell'ID di connessione è di assicurare che i cambiamenti nell'indirizzamento dei protocolli di livello inferiore (UDP, IP ed oltre) non provochino malfunzionamenti nella consegna dei pacchetti su connessioni QUIC ad esempio consegnandoli dal lato sbagliato.

Traendo vantaggio dall'ID di connessione, le connessioni possono dunque essere migrate attraverso indirizzi IP ed interfacce, in modi mai previsti dal TCP. Per esempio, si potrà migrare una connessione di download in corso su una rete mobile verso una connessione wifi a banda larga, senza interrompere il download. In modo analogo, il download potrà procedere su rete mobile se la connessione wifi dovesse interrompersi.

Numeri di porta

Essendo QUIC costruito su UDP, un campo a 16bit è utilizzato per distinguere le connessioni in ingresso.

Negoziazione delle versioni

Una connessione QUIC originata da un client renderà noto al server quale versione QUIC esso preferisca parlare, ed il server risponderà con una lista delle versioni supportate, lasciando la scelta finale al client.

Connessioni su TLS

Successivamente al pacchetto iniziale che si occupa di istanziare la connessione, l'estremo che ha iniziato la connessione invierà un "crypto-frame" che scaturisce la negoziazione dello strato di sicurezza. Lo strato di sicurezza si basa su TLS 1.3.

Non vi è modo di evitare o declinare l'uso di TLS per una connessione QUIC. Il protocollo è specificamente concepito per evitare che le "middle-boxes" possano intercettare o modificare il traffico, allo scopo di evitare una ossificazione prematura del protocollo.

Streams

I flussi QUIC permettono l'astrazione leggera e ordinata della sequenza di bytes.

Esistono due tipi di flussi (streams) in QUIC:

- Flussi unidirezionali che trasportano dati in una sola direzione: dal produttore dello stream all'altro estremo, il destinario.
- Flussi bidirezionali che permettono di inviare dati in entrambe le direzioni.

Ciascuno dei due tipi di flusso può essere creato da ognuno dei due estremi, può inviare dati allo stesso tempo su flussi alternati, o può essere soppresso.

Per mandare dati all'interno di una connession QUIC, è necessario utilizzare uno o più flussi.

Controllo di flusso

I flussi sono controllati individualmente, peremettendo ad ogni estremo di limitare l'allocazione della memoria e di applicare una pressione inversa. La creazione dei flussi è anch'essa soggetta a controllo di flusso; si richiede ad ogni peer di dichiarare l'ID di flusso "massimo" al quale si è disposti ad arrivare (dunque l'ID corrisponde al numero totale di flussi gestibili da ogni estremo).

Identificatori di flusso

I flussi sono definiti da un intero di 62-bit privo di segno, al quale si è soliti riferirsi come "Stream ID". All'interno dello Stream-ID, i due bits meno significativi sono utilizzati per identificare il tipo di flusso (uni o bi-direzionale) e chi abbia inizializzato la comunicazione.

Il bit meno significativo dello Stream-ID (0x1) identifica chi dei due abbia inizializzato il flusso. In caso sia il client ad iniziare la connessione il numero di flusso risulterà parii (LSB a 0); tale numero sarà dispari in caso di flusso iniziato dal server (LSB a 1).

Il secondo bit meno significativo dello Stream-ID (0x2) è utilizzato per distinguere fra flussi uni e bi-direzionali. Flussi unidirezionali avranno impostato il bit a 1, flussi bidirezionali utilizzeranno sempre il valore 0.

Concorrenza di flussi

QUIC permette ad un numero arbitrario di flussi di operare in concomitanza. Un estremo limita il numero di flussi concorrenti in entrata comunicando lo Stream-ID massimo che è disposto ad accettare.

Il massimo valore di Stream-ID è specifico ad ogni estremo e si applica solo all'estremità che riceve tale impostazione.

Spedire e ricevere dati

Gli end-points utilizzano i flussi per spedire e ricevere dati. In fondo è questo il loro fine ultimo. Gli streams sono un flusso "astratto" di bytes ordinati. Streams separati non saranno necessariamente consegnati nello stesso ordine di invio.

Prioritizzazione dei flussi

Il multiplexing dei flussi ha un impatto significativo sulle performance dell'applicazione, in caso le risorse assegnatevi siano correttamente ordinate per priorità. L'esperienza con altri protocolli multiplexati -quale HTTP/2- mostra che una buona strategia di prioritizzazione ha certamente un impatto positivo sulle prestazioni.

QUIC in se non mette a disposizione frames adatti a scambiare informazioni sullo stato delle priorità. Al contrario si accontenta di ricevere tali informazioni direttamente dall'applicazione essa stessa basata su QUIC. I protocolli che utilizzano QUIC sono liberi di definire il proprio schema di prioritizzazione secondo le proprie necessità e semantiche applicative.

Quando si utilizza HTTP/3 su QUIC, la prioritizzazione è gestita all'interno dello strato HTTP.

0-RTT

Per ridurre il tempo richiesto per stabilire una nuova connessione, un client che si sia precedentemente collegato ad un server potrebbe mantenere in cache alcuni parametri relativi a tale connessione ed utilizzarli successivamente per iniziare una connessione a **0-RTT** con il server. Questa modalità permette al client di inviare dati fin dal primo pacchetto, senza dover aspettare il termine della negoziazione stessa.

Bit Rotante

Una delle discussioni più estenuanti all'interno del gruppo di lavoro QUIC . oggetto di mille emails e ore di dispute verbali- è legata ad un singolo bit: il bit rotante (spin bit).

I fautori dello spin-bit insistono nell'affermare quanto -per operatori ed amministratori di rete- sia fondamentale poter misurare la latenza fra due end-points QUIC.

Gli opppositori dello spin-bit temono una potenziale fuga di informazioni.

Girare un bit

Entrambi gli estremi mantengono temporaneamente un valore di 0 o 1 per ogni singola connessione QUIC; client e server sono incaricati di impostare un valore all'interno di ciascun pacchetto spedito.

Entrambi gli estremi inviano il pacchetto impostando lo stesso valore per la durata di un solo round-trip, alchè invertono il valore. Il risultato è dunque una serie di pulsazioni di 0 e 1 utile al monitoraggio.

Questo tipo di misurazione è efficace solo se il peer che invia non è limitato a livello di applicazione o di controllo di flusso, in assenza di riordino di pacchetti (causato dalla rete stessa).

User space

Implementare un protocollo di trasporto all'interno dello user-space permette di poter manipolare/programmare il protocollo facilmente, senza necessita di aggiornare il kernel o il sistema operativo ad ogni evoluzione delle librerie client/server.

Non vi sono tuttavia ostacoli fattuali alla implementazione di QUIC all'interno del kernel; qualcuno in futuro se ne occuperà, se veramente ve ne sarà la necessità.

Molte implementazioni

L'implementazione di un nuovo protocollo di trasporto all'interno dello user-space ha per effetto la nascita di molteplici implementazioni.

In un futuro prossimo, è certo che le diverse applicazioni conterranno (o si baseranno) su implementazioni eterogenee di HTTP/3 e QUIC.

API

Uno dei fattori del successo del classico TCP e dei programmi su di esso basati è stata la disponibilita di API socket standardizzazate. Poter offrire funzionalità ben definite usando tali API permette di "portare" programmi fra diversi sistemi operativi dato che TCP funziona in maniera identica indipendentemente dalla piattaforma.

QUIC non è ancora arrivato a quel punto. Non esistono API standard in QUIC.

Con QUIC, ci si deve basare su una delle librerie esistenti e fare affidamento alle API fornite dalla libreria scelta. Ciò fa si che le applicazioni siano "abbinate" ad una singola libreria, almeno in teoria. Cambiare ed adottare un'altra libreria signigica nuove API e potrebbe richiedere molto lavoro di adattamento.

Inoltre, dato che QUIC è spesso implementato in user-space, non è adatto ad estendere le API socket o offrire funzionalità in qualche modo simili agli attuali TCP e UDP. Usare QUIC significa utilizzare un'API esterna al socket.

HTTP/3

Come sottolineato in precedenza, il primo e principale protocollo trasportato via QUIC è HTTP.

In maniera simile a quanto successe per HTTP/2 -introdotto per trasportare HTTP in binario- HTTP/3 introduce una nuova modalità di trasporto per inviare sintassi HTTP sulla rete.

HTTP mantiene ancora gli stessi paradigmi e concetti che gia conteneva in precedenza. Vi sono intestazioni e corpo del messaggio, vi sono richieste e risposte. Esistono verbi, cookies e meccanismi di ritenzione del contenuto (cache). Ciò che in sostanza cambia in HTTP/3 è la modalità di trattamento (trasferimento) riservata alle informazioni da trasferire all'altro capo.

Per fruire di HTTP via QUIC sono stati necessari alcuni cambiamenti, il risultato dei quali prende il nome di HTTP/3. Questi cambiamenti sono legati alla diversa natura di QUIC rispetto a TCP. Le modifiche includono:

- In QUIC i flussi sono forniti dal meccanismo di trasporto stesso, mentre in HTTP/2 i flussi sono gestiti all'interno dello strato HTTP.
- Per via del fatto che i flussi in QUIC sono independenti l'uno dall'altro, non è stato possibile utilizzare la stessa identica modalità di compressione delle intestazioni gia impiegata in HTTP/2 poichè essa avrebbe provocato "bloccaggi di inizio riga".
- Gli streams QUIC sono lievemente diversi da quelli HTTP/2. La sezione dedicata a HTTP/3 tratta più in dettaglio l'argomento.

URLs in HTTPS://

HTTP/3 si servirà di URLs del tipo `HTTPS://`. Il mondo è pieno di questo tipo di URL ed è stato osservato che sarebbe impraticabile ed illogico proporre un ulteriore schema di URL per il nuovo protocollo. Similmente a come HTTP/2 non avesse bisogno di un nuovo schema, cosi fu per HTTP/3.

La complessità aggiunta da HTTP/3 è evidente se guardiamo al fatto che HTTP/2 -seppur rivoluzionario nel trasporto di HTTP a livello di trama- era ancora basato su TLS e TCP come nel caso del predecessore HTTP/1. Il fatto che HTTP/3 lavori su QUIC fa vacillare un certo numero di assunzioni.

Il caro vecchio URL "clear-text" `HTTP://` verrà lasciato intatto e sarà via via abbandonato nel momento in cui si passerà a modalità di trasporto più sicure. Richieste verso questo tipo di URL non riceveranno alcun upgrade a HTTP/3. Nella realtà dei fatti, ben poche di queste connessioni ricevono upgrade a HTTP/2, ma per altri motivi.

Connessione iniziale

La prima connessione ad una risorsa o host non ancora visitato via URL HTTPS:// sarà probabilmente stabilita via TCP (eventualmente in parallelo con un tentativo di connessione QUIC). L'host distante potrebbe essere un vecchio server che non fornisce supporto a QUIC, o potrebero esservi dispositivi sul percorso di rete che impediscono il successo della connessione via QUIC.

Un client moderno negozierebbe probabilment HTTP/2 in prima istanza. Una volta che tale connessione

fosse stata effettuata, e che il server avesse risposto ad una richiesta HTTP, il server potrebbe comunicare al client la disponibilità del supporto e la preferenza per HTTP/3.

Bootstrap via Alt-svc

L'intestazione di servizio alternativo (Alt-svc:) e il corrispettivo frame HTTP/2 `ALT-SVC` non sono specificamente creati per QUIC o HTTP/3. Tali header sono parte di un meccanismo gia ben rodato per cui un server può segnalare ad un client *"guarda, io erogo questo stesso servizio ANCHE su QUESTO HOST, utilizzando QUESTO PROTOCOLLO, su QUESTA PORTA"*. Vedi i dettagli nella RFC 7838.

In caso lo supportasse e lo desiderasse, ad un client che ricevesse tale header Alt-Svc in una risposta verrebbe consigliato di connettersi in parallelo ed in background all'host suggerito -utilizzando il protocollo definito- ed in seguito completare il resto delle operazioni utilizzando questo nuovo canale, al posto della connessione iniziale.

In caso la connessione iniziale stesse utilizzando HTTP/2 o addirittura HTTP/1, il server può decidere di rispondere al client di riprovare tale connessione via HTTP/3. Tale connessione potrebbe essere diretta allo stesso host, o ad un altro host distante, capace di servire tale contenuto sul protocollo prescelto. L'informazione fornita all'interno dell'header Alt-Svc è dotata di una data di scadenza, che permette la redirezione dei clients verso host alternativi entro un certo lasso temporale.

Esempio

Un server HTTP include nella propria risposta un header di tipo `Alt-Svc:`:

```
1 Alt-Svc: h3=":50781"
```

Questo indica che HTTP/3 è disponibile via UDP sulla porta 50781, sullo stesso host utilizzato in prima istanza.

Un client può a quel punto tentare di stabilire una connessione QUIC verso la destinazione indicata, ed in caso di successo continuerebbe a comunicare con l'origine attraverso il nuovo canale, non attraverso l'originale in HTTP.

Gli streams QUIC e HTTP/3

HTTP/2 è stato costretto a concepire il proprio schema di flussi e multiplexing basandosi su TCP, mentre HTTP/3 è studiato per lavorare con QUIC e quindi trae massimo vantaggio dall'utilizzo degli streams.

Le richieste HTTP veicolate su HTTP/3 sfruttano un set di flussi specifico.

Frames HTTP/3

Dialogare in HTTP/3 implica l'attivazione di un flusso QUIC e l'invio di una serie di frames all'altro capo della connessione. Esistono al momento (alle 9 del 18 Dicembre 2018) una serie abbastanza limitata di frames predefiniti in HTTP/3. I più rilevanti sono seza dubbio:

- HEADERS, si occupano di comprimere e spedire le intestazioni
- DATA, tratta l'invio di dati binarii
- GOAWAY, pregasi terminare la connessione

Richiesta HTTP

Il client invia la propria richiesta HTTP su un flusso QUIC *bidirezionale* iniziato dal client.

Una richiesta consiste di un singolo frame HEADERS e può essere seguita da uno o due frames aggiuntivi: una serie di frames DATA ed eventualmente un frame finale contenente gli HEADERS di coda.

Dopo aver spedito la propria richiesta, il client termina il flusso in uscita.

Risposta HTTP

Il server replica spedendo la risposta HTTP sullo stream bidirezionale. Un frame HEADERS, una serie di DATA ed eventualmente un frame HEADERS di coda.

Intestazioni QPACK

I frames HEADERS contengono le intestazioni HTTP compresse con l'algoritmo QPACK. QPACK risulta simile al vecchio HPACK utilizzato in HTTP/2 (RFC 7541), benchè modificato per consegnare i differenti flussi in modalità "disordinata" (out-of-order).

QPACK si avvale di due streams QUIC unidirezionali fra i due estremi della connessione. Tali streams sono utilizzati per trasportare la tabella dinamica di compressione nelle due direzioni opposte.

Prioritizzazione

Uno dei frame facenti parte di ogni flusso HTTP/3 è il campo `PRIORITY`. E' usato per segnalare la priorità e l'ordine di dipendenza fra flussi diversi molto similmente a quanto gia avveniva in HTTP/2.

Il frame può essere impostato in modo che un flusso dipenda da un altro e può definire un "peso specifico" su ogni determinato flusso.

Un flusso dipendente avrà diritto ad allocare risorse solamente se tutti gli altri flussi dai quali esso dipende fossero gia in fase di chiusura, o non fosse possibile avanzare ulteriormente su tali suddetti flussi.
Uno stream assume un valore compreso fra 1 e 256. La regola vuole che un flusso con uno stesso antenato **dovrebbe** allocare risorse in maniera proporzionale basandosi sul peso specifico di ognuna.

Server push

La modalita "server push" di HTTP/3 è tutto sommato simile a quanto descritto per HTTP/2 nella (RFC 7540) pur usando meccanismi diversi.

Una "server push" è a tutti gli effetti la risposta ad una richiesta che il client in realtà non ha mai inviato !

Le "server push" sono difatti autorizzate solamente in caso il client abbia deciso di accettarle. In HTTP/3 il client imposta addirittura un limite al numero massimo di risposte push che è disposto ad accettare, comunicando al server l'ID di fluso più alto che sarà disposto a trattare. Se il server dovesse tentare di inviare flussi al di sopra dell'ID indicato, ciò provocherebbe un errore di connessione.

Se il server dovesse pensare che il client possa aver bisogno di una determinata risorsa -risorsa comunque ritenuta necessaria- potrebbe volergli inviare un frame detto `PUSH_PROMISE` (all'interno dello stesso flusso della richiesta) contenente le coordinate della richiesta per la risorsa in questione ed in seguito inviare la risposta all'interno di un nuovo flusso.

Anche nel momento in cui le "server push" dovessero essere ritenute credibili da parte del client, ognuna delle singole "push" può essere annullata ad ogni momento se il client dovesse in qualche modo ritenerlo necessario. In tal caso invierebbe un frame `CANCEL_PUSH` al server.

Problematiche

Questa caratteristica è stata messa in discussione, criticata e rigirata sin dal primo momento, sia durante lo sviluppo di HTTP/2 sia dopo la sua pubblicazione e impiego su larga scala, tentando di renderla utilizzabile.

Un invio in "push" non è mai gratis, benchè utilizzi solo metà del tempo di round-trip, occupa banda passante. E' spesso difficile -impossibile- dal punto di vista del server stabilire con certezza se una determinata risorsa possa necessitare (beneficiare) del "push" o meno.

Paragone con HTTP/2

HTTP/3 è stato disegnato per QUIC, che è un protocollo di trasporto autonomo nel gestire i proprii flussi.

HTTP/2 è stato concepito su TCP, quindi gestisce i sotto-flussi all'interno dello strato HTTP.

Somiglianze

I due protocolli hanno virtualmente un insieme di caratteristiche identico.

- Entrambi offrono streams (flussi)
- Entrambi supportano la modalità "server push"
- Entrambi utilizzano la compressione degli header, QPACK e HPACK risultano molto simili nel design funzionale
- Entrambi sfruttano il multiplexing via flussi, su una singola connessione
- Entrambi i protocolli contengono la nozione di priorità di flusso

---

Differenze

Le differenze riguardano i dettagli, soprattutto in relazione al fatto che HTTP/3 usi QUIC:

- HTTP/3 funziona meglio con i "dati anticipati" grazie al supporto per la negoziazione a 0-RTT, mentre sappiamo che il TCP Fast Open accoppiato a TLS spesso incontra problemi, inviando comunque minor quantità di dati
- Le negoziazioni (handshakes) sono molto più rapide in HTTP/3 grazie a QUIC
- Non esistono versioni non crittate o non sicure di HTTP/3. HTTP/2 può ancora essere implementato ed utilizzato senza HTTPS - benchè molto raro su Internet
- HTTP/2 può essere negoziato direttamente all'interno di una negoziazione TLS grazie all'estensione ALPN. Diversamente, in HTTP/3 (essendo su QUIC) si dovrà a priori informare il client con una risposta contenente l'intestazione `Alt-Svc:` prima che il client stesso possa avere conoscenza di tale risorsa.

Critiche comuni

UDP non funzionerà mai

Una marea di aziende, operatori ed organizzazioni hanno l'abitudine di bloccare o limitare tutto il traffico UDP non afferente alla porta 53 (DNS) dato che [UDP] è stato spesso sfruttato in attacchi DDoS. Nello specifico alcuni dei protocolli UDP esistenti e le rispettive implementazioni server, sono stati recentemente coinvolti in attacchi detti "di amplificazione", dove a partire da un piccolo volume di traffico si arriva a generare una vera e propria onda di traffico, onde "mirate" verso un obiettivo innocente.

QUIC contiene una tecnica di mitigazione contro gli attacchi "amplificatori" esigendo che il pacchetto iniziale consti almeno di 1200 bytes ed impiegando una restrizione a livello di protocollo che IMPEDISCE al server di inviare una risposta più grande di tre volte il volume della richiesta originale, senza prima aver ricevuto un pacchetto di conferma dal client.

UDP è lento nel kernel

Sembra essere vero, almeno ad oggi nel 2018. Non possiamo ovviamente predire in che direzione andranno gli sviluppi, o quanto questo sia il risultato dei lunghi anni di indifferenza da parte degli sviluppatori verso le prestazioni di UDP stesso.

La maggior parte dei client non percepisce affatto questa lentezza.

QUIC prende troppa CPU

Analogamente a quanto detto sopra -la critica "UDP è lento"- questa situazione deriva dal fatto che TCP e TLS hanno goduto di molto più tempo per maturare, per evolvere, ed hanno conseguentemente beneficiato di un supporto hardware e software più cospicuo.

Abbiamo ragione di credere che questa situazione evolverà in futuro. Una domanda rimane: fino a che punto questi problemi di CPU determineranno un ostacolo per i responsabili della rete ?

C'è solo Google

Non è vero. Google è reponsabile per aver presentato la prima specifica allo IETF, dopo aver dimostrato la potenziale efficacia nell'adozione di un tale protocollo, costruito sul gia noto UDP.

A partire da allora, una molteplicità di persone di compagnie ed entità diverse ha lavorato all'interno dell'organizzazione neturale IETF, al fine di creare un protocollo di trasporto standardizzato. Ovviamente, gli impiegati di Google hanno participato al lavoro di gruppo, ma non dobbiamo trascurare un gran numero di esperti che hanno interesse a contribuire all'evoluzione del trasporto Internet, quali Mozilla, Fastly, Cloudflare, Akamai, Microsoft, Facebook e Apple.

Non rappresenta un gran miglioramento

Questa non è una vera critica ma piuttosto un'opinione. Magari è il caso, magari il miglioramento non è davvero così netto considerando che HTTP/2 è ancora molto recente.

Si suppone che HTTP/3 sarà più performante in reti con alto tasso di perdita di pacchetti, oltre ad essere in grado di attuare una negoziazione più veloce; la latenza percepita e la latenza effettiva ne beneficieranno sicuramente. Saranno questi benefici sufficienti a motivare gli utenti e gli editori a migrare verso un supporto per HTTP/3 a livello di servizi e software? Solo il tempo ed un buon assessment delle prestazioni potranno rispondere!

Le specifiche

Qui troviamo una collezione delle ultimissimee bozze ufficiali per tutti i vari pezzi e componenti di QUIC e HTTP/3.

Invarianti (Costanti)

Proprietà di QUIC indipendenti dalla versione

Trasporto

QUIC: un trasporto "multiplexato" e sicuro basato su UDP

Recupero ("recovery")

Rilevamento delle perdite e controllo della congestione in QUIC

TLS

Usare TLS (sicurezza a livello di trasporto) per securizzare QUIC

HTTP

Protocollo di trasferimento di ipertesto (HTTP) via QUIC

QPACK

QPACK: Compressione degli Header per HTTP via QUIC

QUIC v2

Per potersi concentrare il più possibile sul cuore del protocollo QUIC ed essere in grado di consegnarlo per

tempo, diverse opzioni che inizialmente avrebbero dovuto essere incluse come base del procollo hanno purtroppo dovuto essere posticipate fino al punto da essere state pianificate in una versione successiva di QUIC. Appunto, QUIC versione 2 o maggiore.
L'autore di questo documento dispone di una sfera di cristallo non proprio perfetta, quindi non può predire esattamente quali di queste caratteristiche saranno o meno integrate nella versione 2. Possiamo tuttavia citare alcune delle opzioni esplicitamente rimosse dalla v1 -per essere sviluppate in un secondo momento- opzioni che potrebbero perciò essere presenti nella v2.

Correzione dell'errore anticipata (Forward Error Correction)

La FEC (Forward Error Correction) è un metodo di controllo d'errore relativo alle trasmissioni dati in cui un emissario (sender) spedisca una quantità di dati ridondante e in cui il ricevente (receiver) riconosca solamente la porzione di dati che non contenga alcun errore apparente.

Google ha sperimentato questa tecnica nella sua implementazione originaria di QUIC ma fu successivamente rimossa in quanto gli esperimenti non diedero risultati soddisfacenti. Questa opzione è soggetta a discussione per entrare nella seconda versione di QUIC, ma necessita di argomenti a proprio favore nel mostrare che non penalizzi o influenzi negativamente le prestazioni.

Multipath

Il "multipath" è una tecnica di trasporto che per definizione utilizza percorsi di rete complementari (multipli ed alternativi) al fine di massimizzare l'utilizzo delle risorse ed aumentarne l'affidabilità attraverso la ridondanza.

I fautori e sostenitore di SCTP tengono a sottolineare come SCTP contempli gia il multipath, così come il TCP più moderno.

Dati inaffidabili

E' stato proposto di offrire streams "non-affidabili" come opzione, il che permetterebbe a QUIC di rimpiazzare qualsiasi applicazione che utilizzi UDP.

Adattamenti che non riguardano HTTP

DNS over QUIC è stato uno dei protocolli non-HTTP di cui si è parlato per primo, e al quale probabilmente sarà dedicata molta attenzione una volta che QUIC v1 e HTTP/3 saranno consegnati al pubblico. Ovviamente, con la nascita di un nuovo tipo di trasporto come questo, sono sicuro che il campo di applicazione non si limiterà al solo DoQ.

日本語

この本の試みは2018年3月に始まりました。HTTP/3 と、その根幹のプロトコルである QUIC を文書化することがその目的です (なぜ、どのようにして動作するのか、プロトコルの詳細、その実装など)。

この本は完全に無償で提供され、援助したいと考えるすべての人を巻き込んだ共同作品です。

前提条件

この本の読者は、TCP/IP ネットワーキングの基礎や、HTTP、Web の基本を理解しているものとみなされます。 HTTP/2 に関する詳細や特徴については、http2 explained を最初に読むことを推奨しています。

著者

この本は Daniel Stenberg によって作成されました。 Daniel は、HTTP クライアントソフトウェアとして世界中で最も幅広く使われている curl の作者であり、リードデベロッパーです。 Daniel は20年以上にわたり HTTP やインターネットのプロトコルに関して取り組んでおり、 http2 explained の著者でもあります。

訳者:

- inductor
- ebiiim
- kousukekikuchi1984
- MATTENN
- beagle
- MakTak
- akihirok2k2
- OldBigBuddha
- hidesuke
- ichika
- peacock
- gim_kondo
- hykw
- misato8310
- aoi
- morin_river
- waku-tkd

- smaeda-ks

ホームページ

この本のホームページは daniel.haxx.se/http3-explained にあります。

ヘルプ！

本文書に関する誤字脱字やあからさまな間違いを見つけた場合は、修正した状態の文書を送っていただければ、改訂版を作成します。

助けていただいたすべての方に、適切なクレジットを提供します！ 時間をかけてこの文書を良くしていければと思っています。

よろしければ、誤字の指摘 または プルリクエスト を本の GitHub ページに送ってください。

License

なぜ QUIC なのか

QUIC は略語ではなく名称です。英単語の "quick" と全く同じ発音です。

QUIC は多くの点で、HTTP のようなプロトコルに適した、新しく信頼性の高い安全なトランスポートプロトコルであり、TCP や TLS 上で HTTP/2 を動かす上で知られるいくつかの欠点に対応できる方法と見なすことができます。

Web トランスポートの進化における論理的な次のステップです。

QUIC は HTTP のトランスポートだけに限定されるものではありません。この新しいトランスポートプロトコルを作り始めた、おそらく最大の理由にして最初のきっかけは、エンドユーザーに Web とデータをより早く配信したいという願望です。

それでは、なぜ新しいトランスポートプロトコルなのでしょうか、なぜ UDP の上に作ったのでしょうか？

QUIC logo

HTTP/2、覚えていますか？

HTTP/2 の仕様、RFC 7540 は2015年の5月に公開され、インターネット及び Web に広く展開・実装されてきました。

2018年はじめには、世界トップ1000のウェブサイトのうちほぼ40 %が HTTP/2 で動作しており、Firefox が発行する get レスポンスのうちおよそ70 %の HTTPS リクエストを HTTP/2 が占め、すべての主要なブラウザやサーバー、プロキシが HTTP/2 をサポートしています。

HTTP/2 は HTTP/1 におけるおびただしい数の欠点に対処しており、導入によって HTTP ユーザーが抱える多くの回避策を使わなくてすむようになります。

HTTP/2 の主要な特徴の一つには、多重化があり、論理的な複数のストリームを物理的に単一な TCP 通信で転送することができます。

輻輳制御の動作が飛躍的に向上し、ユーザーが TCP をより効率的に使うことができるため、帯域幅を適切に使い切ることができ、TCP コネクションをより長く持たせるようになります。

結果として、以前よりもより頻繁に最高速の通信に達することができます。

ヘッダー圧縮により帯域幅をより小さくできます。

HTTP/2 では、ブラウザが各ホストに対して確立する TCP コネクションの数は、前バージョンの6個とは異なり、*単一*が一般的です。

実際、HTTP/2 で使われているコネクションの統合と「デシャーディング」技術はそれよりもさらにコネクション数を減らすことがあります。

HTTP/2 は HTTP における head-of-line ブロッキング問題 (クライアントは、次のリクエストを送信するために既存のリクエストのデータを取得し終わるまで待たなければならない問題) を解消しています。

http2 man

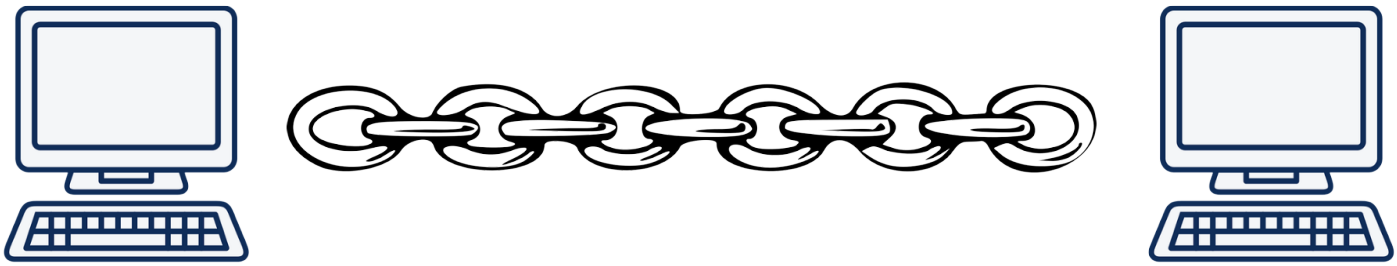
TCP head-of-line ブロッキング

TCP head-of-line ブロッキング

HTTP/2 は TCP を使って動作し、従来のバージョンの HTTP を使用するよりも少ない TCP コネクション数になります。

TCP は信頼性の高い転送のためのプロトコルで、基本的には2つのマシンの間につながった仮想的な鎖のように考えることができます。

一方の端からネットワーク上に出されたデータは、結果的に、もう一方の端に全く同じ順番で到達します（または通信が途切れます）。



a TCP chain between two computers

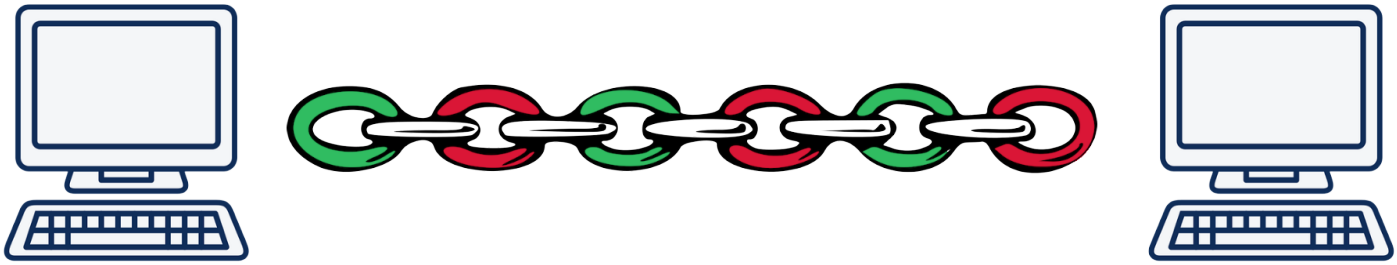
HTTP/2 を使うことで、典型的なブラウザは数十または数百の並列データ転送を単一の TCP コネクション上で行います。

HTTP/2 を話す2つのエンドポイント間にあるネットワーク上で一つのパケットがドロップされると、それはすなわち、その損失したパケットが再送され、宛先に届けられるまでの間、TCP コネクション全体が停止することを意味します。

TCP がこの「鎖」であるため、一つのつなぎ目が突然欠落すると、その欠落したもの以降すべてのつながりが待つ必要があります。

2つの別々のストリームを単一コネクションで送信するときに鎖を使って図解したイラスト。

赤色のストリームと緑色のストリーム:



the chain showing links in different colors

これが、TCP ベースの head-of-line ブロックになります！

パケットロス率が上がるにつれて、HTTP/2 のパフォーマンスはますます低下します。

2 %のパケットロス (これはかなりひどいです、念のため。) があるネットワーク環境では、大抵の場合において HTTP/1 のほうがパフォーマンスが良くなることがテストで証明されています。

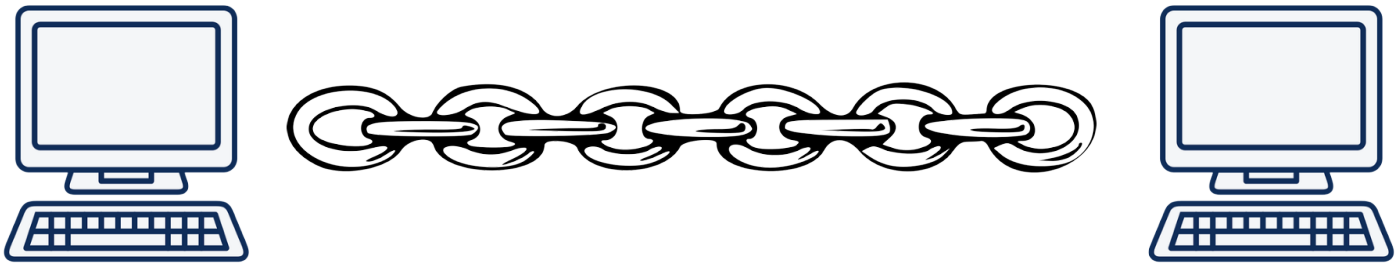
これは、HTTP/1 では通常6つまでの TCP コネクションを使ってパケットを送信するためです。

つまり、パケットが損失した箇所があっても他のコネクションが止まることはないということです。

TCPを使用する限り、この問題を解決することは簡単ではありません。

ブロックを解消するための独立したストリーム

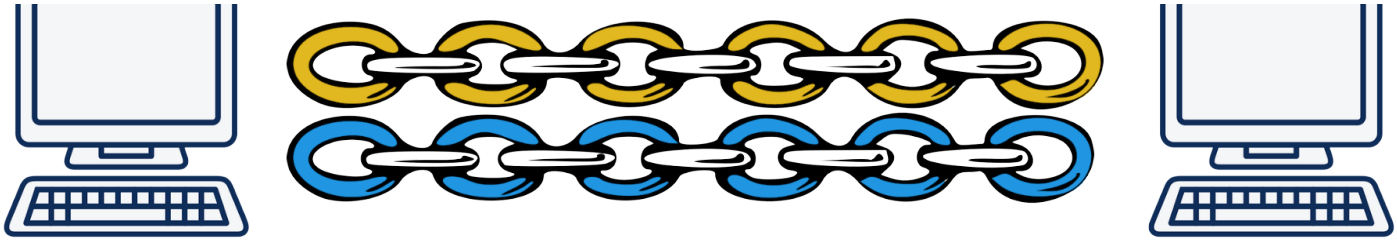
QUIC では、2つのエンドポイントの間に、接続を安全にし、データ配信を信頼できるものにするコネクションのセットアップが依然として存在します。



a QUIC chain between two computers

このコネクション上で2つの異なるストリームをセットアップする際、それらは独立したものとして扱われるため、一つのストリームのあるつなぎ目が損失した場合、そのストリームの、特定のチェインのみが、停止し再送制御を行います。

黄色のストリームと青色のストリームがそれぞれ2つのエンドポイント間で通信を行うイラストがこちらです。



two QUIC streams between two computers

TCP か UDP か

TCP か UDP か

TCP の枠組みで head-of-line ブロッキングを直せないのであれば、理論的には、新しいトランスポートプロトコルをネットワークスタックの中、UDP と TCP に隣接して作れるようになるべきです。もしくは 最悪 SCTP を使うべきかもしれません。SCTP は RFC 4960 の中で IETF によって標準化されたトランスポートプロトコルの一つで、必要とされるいくつかの機能を持っています。

しかし、近年新規のトランスポートプロトコルを作る取り組みは、ほとんどすべて休止しています。何故なら、インターネット上に実際に展開することに困難が伴っていたからです。新規プロトコルの展開は、到達すべきユーザーとサーバーの間に展開されている TCP もしくは UDP のみ許可する、多数のファイアーウォール、NAT、ルーターなど、その他のミドルボックスによって阻まれてきました。他のトランスポートプロトコルを導入することは、N %のコネクションを失敗させることになります。何故なら UDP もしくは TCP ではないということは、何らかの形で不正、もしくは間違っているとボックスにみなされ、ブロックされるからです。多くの場合、 N %の失敗率は努力に対して高すぎるとみなされます。

加えて、通常ネットワークスタックのトランスポートプロトコルレイヤーの中を変更することは、OS のカーネルによって実装されているプロトコルを変更することを意味しています。OS のカーネルを更新して展開することは、多大な努力を必要とする遅いプロセスです。IETF によって標準化された多くの TCP の改善は、広くサポートされていないため、広範囲に渡って展開されたり使用されたりしていません。

なぜ SCTP-over-UDP ではないのか

SCTP はストリームを用いた信頼性のあるプロトコルで、WebRTC には UDP を介してSCTPを使用する実装さえ存在します。

これは QUIC に取って代わるものとして十分ではありませんでした。以下を含む幾つかの理由に原因があります。

- SCTP がストリームの head-of-line ブロッキング問題を解決しないこと
- SCTP では、ストリームの数をコネクションのセットアップ時に決めなければならないこと
- SCTP が確かな TLS/security レイヤーを持たないこと
- SCTP は 4-way ハンドシェイクを使用し、QUIC は 0-RTT を提供すること
- QUIC は TCP 同様バイトストリームで、SCTP はメッセージベースであること

- QUIC コネクションは IP アドレス間を移動することができ、SCTP はできないこと

更なる詳細と違いについては、A Comparison between SCTP and QUIC インターネットドラフトが参考になります。

硬直化

インターネットは「ネットワーク同士のネットワーク」です。

このネットワーク同士のネットワークを想定通り確実に動作させるため、インターネット上の実に様々な場所に機器が設置されています。

ネットワーク上に散りばめられたこれら機器のことを「ミドルボックス」と呼びます。

2つのエンドポイント間のあちこちに存在しているこれらミドルボックスが、旧来からのネットワークデータ転送に関わっています。

ミドルボックスは様々な目的で様々な動作をしていますが、それらはインターネットを正しく動作させるために必要だと誰かが考えた結果設置されたものだ、と広い意味で捉えることができると思います。

ミドルボックスはネットワーク間で IP パケットのルーティングを行います。

また、悪意のあるトラフィックをブロックします。

NAT (Network Address Translation) を行ったり、パフォーマンスを向上させたり、通過するトラフィックを監視したり、それ以外にも様々なことを行います。

これらの役割を担うため、ミドルボックスは「ネットワーク間がどうつながっているのか」や、監視・管理の対象である、プロトコルの詳細について知らなければなりません。

こういった目的で、ミドルボックスはソフトウェアを動かします。

これらのソフトウェアは必ずしも頻繁にアップデートされるとは限りません。

ミドルボックスはインターネットを維持するための接着剤の役割を果たしていますが、最新技術に追従できていないこともよくあります。

世界中のクライアントとサーバーの間にあるミドルボックスは、一般的には末端の機器ほどすぐに変化しません。

ネットワークプロトコルのうち、ミドルボックスにプロトコルを分析させ、通信を許可・ブロックさせたいものは、以下の問題を抱えます。

ミドルボックスは設置した当時の仕様で動作するという問題です。

当時知られていなかった、新しい仕様の追加や振る舞いの変化があると、その機器にとっては不具合や不正といったリスクと見なされます。

このようなトラフィックは、単に捨てられたり、あるいはその機能を使用したくないと思うほど遅延させられたりします。

このような状況は「プロトコルの硬直化」と呼ばれます。

TCP も硬直化によって変化が難しい状況にあります。

クライアントとサーバーの間にある機器が、新しい TCP オプションを不明なものとしてブロックするかもしれません。

プロトコルの詳細が分析できると、それを取り巻くシステムはプロトコルの一般的な振る舞いを学習していき、やがてプロトコルの変更を不可能にしてしまいます。

この硬直化と「戦う」ための真に効果的な唯一の方法は、可能な限り通信を暗号化し、ミドルボックスにそこを通過するプロトコルの詳細を観察させないようにすることです。

セキュア

QUIC は常にセキュアです。

平文版のプロトコルが存在しないので、QUIC コネクションをネゴシエートすることは TLS 1.3 を用いて暗号化とセキュリティを行うことと同義です。

上記のように、(拡張性を損なうという意味での) 硬直化や他の種類のブロックと特別な処理を防ぐと同様に、QUIC は Web ユーザーが望んでいた HTTPS のセキュアなプロパティを全て備えています。

暗号化プロトコルがネゴシエートされる前に、平文で送信されるわずかな初期ハンドシェイクパケットがあるのみです。

レイテンシの軽減

QUIC は新規コネクションのネゴシエート時間を短縮するため、0-RTT ハンドシェイクと 1-RTT ハンドシェイクを提供します。

TCP における 3-way ハンドシェイクと比較してみてください。

加えて、QUIC はより多くのデータを処理するため、最初から "early data" をサポートしており、TCP Fast Open よりも簡単に利用できます。

既存のコネクションが終了するのを待つことなく同じホストへ別のコネクションを接続できることが、ストリームのコンセプトとして挙げられています。

TCP Fast Open の問題について

TCP Fast Open は、2014年12月に RFC 7413 として公開されました。

この仕様では、初回の TCP SYN パケットが既に配信されているサーバーに対して、アプリケーションがどのようにデータを渡すのかについて説明しています。

有志によるこの機能のサポートには時間がかかっており、2018年の現在でも問題が発生しています。

TCP スタックの実装者やアプリケーションがこの機能の利点を利用するためには、OS バージョンに応じた有効化の方法や、問題が発生した際にどのように正常な処理に遷移するのか、の両方を理解する必要があります。
いくつかのネットワークが TFO トラフィックを妨げ TCP ハンドシェイクを台無しにする事が確認されています。

これまでの歩み

最初の QUIC プロトコルは Google の Jim Roskind によって設計され、最初の実装は2012年に行われました。2013年に Google の実験として世界中に公開されました。

当時は QUIC は "Quick UDP Internet Connections" の略語だとされていましたが、そのときからこのように言われなくなりました。

Google はプロトコルを実装し、それに続いて広く使われている Google 製のブラウザ (Chrome) と、広く使われている Google のサーバーサイドのサービス (Google検索、gmail、youtube などなど) に実装しました。彼らは非常に早くプロトコルを改善し、このプロトコルが大量のユーザーでも信頼性が高く動作することを証明しました。

2015年5月に最初の QUIC のドラフトが標準化のために IETF に提出されましたが、QUIC ワーキンググループが承認されスタートするのは2016年後半までかかりました。しかし、多くの人々から注目を集め QUIC ワーキンググループは早急に立ち上がりました。

2017年には Google の QUIC エンジニアが 全ての インターネットトラフィックの約7 %がすでに Google 版の QUIC プロトコルを利用していると報告されました。

IETF

IETF によってプロトコルの標準化が始まり、QUIC ワーキンググループが発足した当初から、QUIC を HTTP "だけ" ではなく他のプロトコルにも適用し、標準化することにしていました。Google-QUIC は HTTP だけを対象にしていましたが、実際は HTTP/2 のフレーム構文を利用し、HTTP/2 でも効果的に動作していました。

また、IETF-QUIC では Google が "カスタム" した手法ではなく、TLS 1.3 に基づいた暗号化とセキュリティを取り入れることにしました。

HTTP 以外のプロトコルにも適用したいという要求を満たすため、IETF QUIC プロトコルのアーキテクチャは2つの異なるレイヤーに分割されました。"transport QUIC" レイヤーと "HTTP over QUIC" レイヤーです (後者は2018年11月に HTTP/3 に名前を変更しました)。

このレイヤー分割は、一見無害のようにも見えますが、IETF-QUIC と Google-QUIC で大きな差分を生み出しました。

ワーキンググループは QUIC の最初のバージョンを予定通りに提供するために、HTTP に範囲を絞り、HTTP で無いものは後に回すことにしました。

私達がこの本を書き始めた2018年3月には QUIC バージョン1の仕様の最終版は2018年11月にリリースさ

れることになっていましたが、数回の遅延を経て、現在(2020年6月)リリースの最終段階に入りました。IETF-QUIC の作業が進むに連れ Google チームは IETF 版から詳細を取り込み、IETF 版が目指しているQUICバージョンにプロトコルを近づけています。Google は Google 版の QUIC を 彼らのブラウザやサービスで引き続き運用しています。

開発中の新しい実装のほとんど は IETF バージョンにフォーカスすると決めており、Google バージョンとの互換性はありません。

HTTP/2 からの経験

HTTP/2 の仕様が RFC 7540 として公開されたのは2015年5月で、QUIC が IETF に最初に提出されるわずか1ヶ月前でした。

HTTP/2 では、既存のHTTPを変えようという機運があり、HTTP/2 を作ったワーキンググループは新しい HTTP は、バージョンが1から2に16年かけて上がったときよりも早いものとなるだろうと確信していました。

HTTP/2 の時点ではソフトウェアスタックもユーザーも HTTP が今後もテキストベースのプロトコルのまま、何かができるとは考えないようになってきました。

HTTP-over-QUIC (HTTP/3) は HTTP/2 に基づいて作成され、多くの概念を引き継いでいますが、いくつかの QUIC が対応した細かい部分を HTTP レイヤから外しています。

現在の状況

QUIC ワーキンググループは2016年後半からプロトコルの策定のために活発に活動し、現在(2020年6月)はリリースのための最終段階に近づいています。

2018年の11月現在では、いまだ HTTP/3 の大規模な相互運用テストは実施されていません。2つの実装が存在し、いずれについても、ブラウザや主要なサーバーソフトウェアにも実装が行われていません。

2019年から2020年にかけて HTTP/3 の相互運用性テストの数は増え、CDNとブラウザは、しばしばオプションのフラグが必要ですが、初期サポートを開始し始めてました。

QUIC ワーキンググループの wiki ページには多くの QUIC 実装リスト が掲載されています。

QUIC の実装は簡単ではなく、プロトコルは毎日のように変更され続けています。

サーバー

NGINX による QUIC および HTTP/3 のサポートは現在開発中で、プレビュー版はすでに告知されています。

Apache が QUIC をサポートしたという公式な発表はありません。

クライアント

IETF バージョンの QUIC や HTTP/3 をサポートしたブラウザをリリースした大規模ベンダーはまだありません。

Google Chrome には Google 版の QUIC が何年も前から組み込まれており、最近はオプションで IETF 版のサポートを開始しました。Firefox も同じようにオプションでサポートを開始しました。

curl は最初の実験的な HTTP/3 サポート (draft-22) を2019年9月11日リリースのバージョン 7.66.0 で対応しました。curl は Cloudflare の Quiche ライブラリ と ngtcp2 ファミリのライブラリを用いてこれを成し遂げました。

実装の障害

QUIC は TLS 1.3 を暗号化とセキュリティのために採用することにしました。これは車輪の再発明を避け、信頼できる既存のプロトコルを利用するためです。しかしながら、これと並行して、QUIC での TLS の利用を本当に効率化することにしました。QUIC では "TLS messages" のみを利用し "TLS records" は利用しないことにしました。

これは無害な変更に聞こえますが、この決定は QUIC を実装している人たちにとってとても高いハードルとなりました。既存の TLS 1.3 をサポートしているライブラリにはこれらの機能にアクセスする API が不足しており、QUIC ではアクセスできないのです。一方で多くの QUIC の実装者は大きな組織に所属しており、それぞれがもっている TLS スタックを別々に使用しているため、全員にとって問題とはなっていません。

2018年11月現在、例えば広く使われているオープンソースの OpenSSL では、これらの必要な API は全くなく、また近々で追加するような要望も上がっていません。

これにより QUIC スタックは OpenSSL 以外の TLS ライブラリを使う、パッチをあてた OpenSSL のビルドを使う、将来の OpenSSL に対しての更新を求める、といったいずれかの選択を取る必要があり、最終的にはデプロイの障害となります。

カーネルと CPU 負荷

Google と Facebook は QUIC を彼らの膨大なトラフィックに適用した場合、HTTP/2 over TLS に比べ約2倍の CPU が必要になると言っています。

しかし、下記のような説明が含まれています。

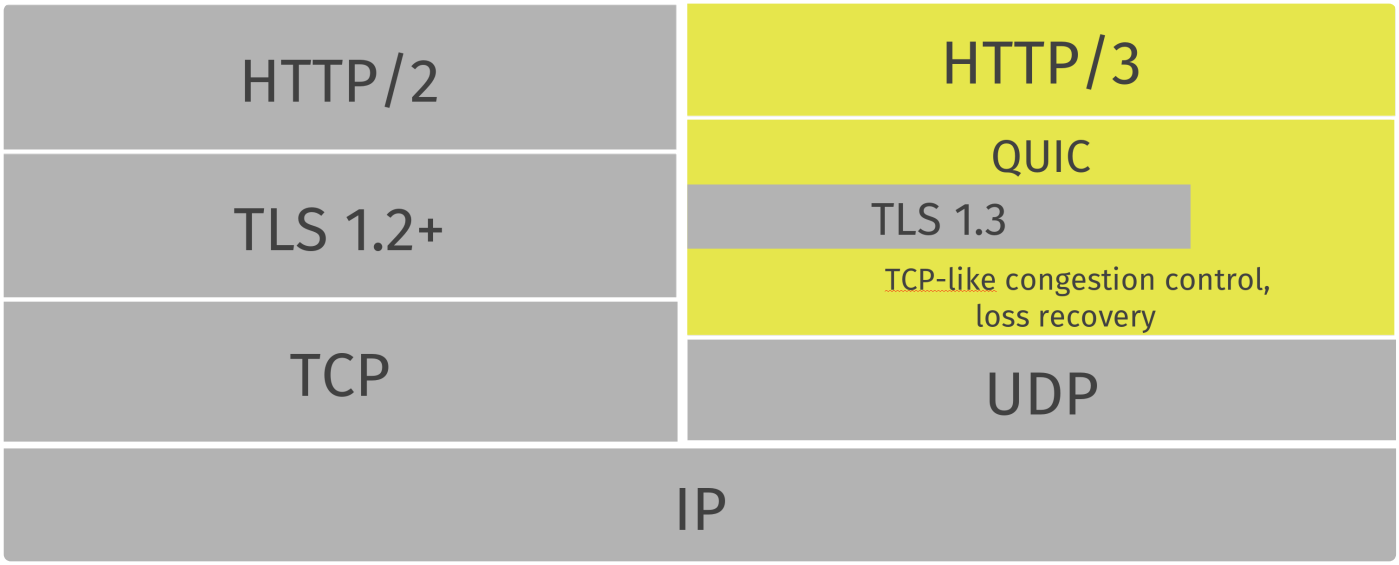
- 歴史的に多くの Linux の UDP スタックはこのような高速通信に利用されることを想定していなかったため、TCP スタックに比べて最適化がされていない
- TCP と TLS の負荷を軽減するハードウェアは存在しているが、UDP のものはほとんどない。基本的に QUIC 向けのものは存在していない

このような問題点がありますが、パフォーマンスと CPU の要求が将来的に改善されると信じています。

プロトコルの機能

高レベルからの QUIC プロトコル

下の図は、HTTP トランスポートとして使われる際の HTTP/2 ネットワークスタックを左側に、QUIC ネットワークスタックを右側に示しています。



QUIC logo

UDP 上の転送プロトコル

UDP 上の転送プロトコル

QUIC は UDP のユーザー空間に実装された転送プロトコルです。

あなたのネットワークトラフィックを軽く確認してみれば、QUIC は UDP パケットとして見えるでしょう。

UDP 上に実装されている以上、QUIC もまた、UDP のポート番号を利用して、与えられた IP アドレス上にある特定のサービスを識別します。

現在、既知の QUIC のすべては、ユーザー空間で動作するように実装されています。

これは通常、カーネル空間で実装するよりも素早い発展が可能です。

うまく動作しますか？

エンタープライズのネットワークなどではポート53 (DNS に使われる) 以外の UDP トラフィックをブロッ

クするように設定していることがあります。
他にも、制御技術が QUIC の性能を TCP を利用したプロトコルよりも悪くすることがあります。

このような、運用者が行うかもしれないことに関する議論には果てがありません。

近いうちに、すべての QUIC をもとにした転送の用途は、もう一つの (TCP をもとにした) 代替手段に正常に切り替えられる必要があるでしょう。

これに関しては、Google のエンジニアによると、計測では1桁台前半の失敗率だったとのことです。

良くなりますか？

QUIC がインターネットの世界にとって価値のある追加物だと証明されれば、人々はそれを使いたいと考え、自分たちのネットワークで機能することを望み、企業はそれに対する障害について考え直すでしょう。

長年にわたり QUIC の開発は進んでおり、インターネットにおいて QUIC を利用した接続の成功率は向上しています。

高信頼性のデータ転送

UDP は信頼性の高い転送プロトコルではありませんが、QUIC は UDP 上に信頼性をもたらすレイヤーを追加します。

これはパケットの再送、輻輳制御、ペーシングおよび現在の TCP が持つこれら以外の機能を提供します。

一方のエンドポイントが QUIC を用いて転送したデータは、コネクションが維持されている限り、遅かれ早かれ他方に届きます。

コネクションは複数のストリームを扱う

SCTP、SSH、HTTP/2 と似たように、QUIC は単一の物理的なコネクションで別々の論理的なストリームを扱えるという特徴があります。

複数の並列ストリームは、それぞれ単一のコネクションで接続しているかのように他のストリームに影響を与えずにデータを転送できます。

QUIC コネクションは、TCP コネクションと似たような仕組みで、2つのエンドポイント間でのネゴシエーションを経て確立します。

QUIC コネクションは UDP ポートと IP アドレスで構成されますが、一旦コネクションが確立すると、それは自身が持つ「コネクション ID」により関連付けられます。

確立済みのコネクションを使って、どちら側もストリームを作成して相手にデータを送信できます。

ストリームのデータは到着順序が保証され、また信頼性があります。
しかし、異なるストリーム間では到着順序は保証されません。

QUIC はコネクションとストリームの両方においてフロー制御を提供します。

詳細は コネクション 節と ストリーム 節で確認できます。

到着順序の保証

QUIC はストリーム内での到着順序を保証しますが、ストリーム間の順序を保証しません。

これは、ストリームは送信したデータの順序を維持しますが、各ストリームが宛先に到達する順序はアプリケーションがそれらを送信したときとは異なるものになり得ることを意味します！

例: ストリーム A と B がサーバーからクライアントに転送されました。

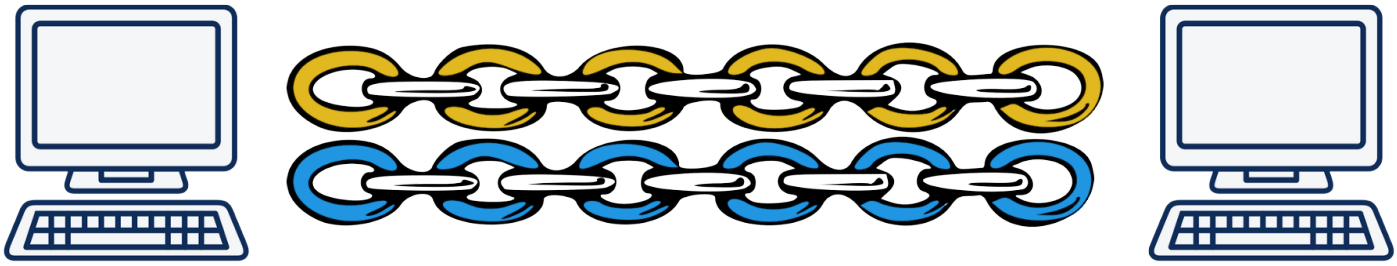
ストリーム A は最初に処理が開始され、ストリーム B はそれに続きました。

QUIC では、パケットの損失はそのパケットが属するストリームのみに影響を及ぼします。

ストリーム A でパケット損失が発生し、ストリーム B では発生しなかった場合、ストリーム A がパケットの再送を行っている間もストリーム B は転送を継続し、先に完了するかもしれません。これは HTTP/2 では不可能でした。

ここに、2つの QUIC エンドポイントが1つのコネクションを介して黄色のストリームと青色のストリームの転送を行う様子を示す図があります。

それぞれのストリームは独立しており、異なる順序で到達する可能性がありますが、いずれのストリームも信頼性と到達順序を損なうことなくアプリケーションに配信されます。



two QUIC streams between two computers

素早いハンドシェイク

QUIC は 0-RTT と 1-RTT 両方のコネクションセットアップを提供し、これは理想的には新しいコネクションを設定する際に追加のやりとりを行う必要がないことを意味します。

0-RTT ハンドシェイクはこの2つのうちで速い方で、事前にコネクションが確立し、かつ、秘密鍵がキャッシュされている状態のときのみ利用することができます。

Early data

QUIC はクライアントが 0-RTT ハンドシェイクの際にデータを含めることを許可しています。

この機能により、クライアントは可能な限り速くピアにデータを配信することができます。

そして、当然のことながら、サーバーは応答して、より早くデータを返すことができます。

TLS 1.3

QUIC におけるトランスポートセキュリティには TLS 1.3 (RFC 8446) を使用しており、暗号化されない QUIC コネクションは一切ありません。

TLS 1.3 には従来の TLS と比較していくつか利点があります。 QUIC が TLS 1.3 を使用する主な理由は、1.3からはラウンドトリップ回数が少なくなるようにハンドシェイクに変更が加えられているためです。これにより、プロトコルの待ち時間を短縮することができます。

かつて Google が提案した QUIC では独自の暗号化技術を使用していました。

トランスポートとアプリケーションレベル

IETF QUIC プロトコルはトランスポートプロトコルであり、その上に他のアプリケーションプロトコルも使用可能です。

最も主要なアプリケーションレイヤープロトコルは HTTP/3 (h3) です。

トランスポートレイヤーではコネクションとストリームをサポートします。

従来の Google 版の QUIC はトランスポートと HTTP が一括で動作するよう一つにまとめられており、HTTP/2 フレームを UDP 上で送ることにより特化したプロトコルでした。

HTTP/3 over QUIC

HTTP/3 と呼ばれる HTTP レイヤーは HTTP に準拠したトランスポートを行います。HTTP ヘッダ圧縮では QPACK を利用しており、これは HTTP/2 における HPACK に似ています。

HPACK アルゴリズムはストリームが順番に届くことを前提としているため、修正せずに HTTP/3 で再利用することは不可能です。

これは QUIC の提供するストリームにおいては転送がばらばらに行われるためです。

QPACK は HPACK の QUIC 対応版といえます。

Non-HTTP over QUIC

HTTP 以外のプロトコルを QUIC を使って通信するための対応は、QUIC のバージョン1が実際にリリースされるまで延期されました。

QUIC の仕組み

ネットワーク上でのビットやバイトに関して詳細な説明はしませんが、このセクションでは QUIC トランスポートプロトコルの基本的な要素がどのように機能するのかを説明します。

独自に QUIC スタックを実装したい場合、この説明はあなたの理解に繋がるはずですが、詳細については IETF のインターネットドラフトと RFC を参照してください。

1. コネクション をセットアップ
2. TLS セキュリティ を組み込み
3. ストリーム を開始

コネクション

QUIC コネクションは2つの QUIC エンドポイント間における一対の対話です。QUIC のコネクションの確立は、レイテンシを低減するために、バージョンネゴシエーションと暗号化およびトランスポートハンドシェイクを組み合わせています。

実際にそのようなコネクションを介してデータ送信するには、1つまたはそれ以上のストリームを作成し、使用する必要があります。

コネクション ID

各コネクションは一組のコネクション ID (コネクション識別子) を持ち、コネクションを識別するために、その ID を使用することができます。コネクション ID はエンドポイントにより独立して選ばれます。各エンドポイントは、ピアが使用するコネクション ID を選択します。

これらのコネクション ID の主な機能は、下位のプロトコル層 (UDP、IP、 及びそれ以下) でのアドレッシングの変更が、QUIC コネクションのパケットを間違ったエンドポイントに配信しないようにすることです。

コネクション ID を利用することにより、TCP が絶対にできない方法で、IP アドレスとネットワークインターフェースの間でコネクションをマイグレーションすることができます。 例えばダウンロードの進行中に、端末がモバイル回線から Wi-Fi に接続を切り替えたとき、セッションを維持しつつ、ダウンロードがより速い Wi-Fi に移行できます。その逆でも同様に、セッションを維持できます。

ポート番号

QUIC は UDP の最上層に組み込まれているので、16bit のポート番号フィールドは、着信相手を区別するために使用されています。

バージョンネゴシエーション

クライアントからの QUIC コネクションリクエストでは、どの QUIC プロトコルバージョンを使用するかサーバーに通知します。サーバーは、クライアントが次に進む際に選択できるように、自身のサポートしているバージョンのリストを返答します。

接続で TLS を使う

接続の設定に利用される最初のパケットが到着するとすぐに、通信の開始者はセキュアレイヤーにおけるハンドシェイクのセットアップを開始する crypto フレームを送信します。

セキュリティレイヤーは TLS 1.3 セキュリティが用いられています。

TLS を使用せずに QUIC コネクションを行う方法はありません。プロトコルの硬直化を防ぐためにQUICはプロトコルレベルでミドルボックスによる通信の改ざんが難しくなるように設計されています。

ストリーム

QUIC におけるストリームは軽量で順序付けられたバイトストリームの概念を提供します。

QUIC には2種類の基本的なストリームが存在します:

- イニシエータからピアへデータを1方向に転送する単方向ストリーム
- お互いにデータを送ることができる双方向ストリーム

どちらのエンドポイントも、両方のタイプのストリームを作成でき、交互に配置した複数ストリームのデータを並行で送信したり中止したりできます。

QUIC のコネクションを経由してデータを送る際には、1つ以上のストリームが利用されます。

フロー制御

ストリームは各自独立にフロー制御が行われ、エンドポイントがメモリの割当量を制限したり、バックプレッシャーをかけたりできるようになっています。

ストリーム作成も同様にフロー制御が行われ、それぞれのピアが一時的に許可される最大 Stream ID を宣言します。

ストリームの識別名

ストリームは 62bit の符号なし整数により識別され、この整数を Stream ID と呼びます。

Stream ID の最下位 2bit はストリームの種類 (単方向もしくは双方向) とストリームのイニシエータの識別に利用されます。

Stream ID の 最下位 bit (0x1) は ストリームのイニシエータを識別します。

クライアントは偶数のストリームを開始し (このときの最下位 bit は 0 に設定されます)、サーバーは奇数のストリームを開始します (このときの最下位 bit は 1 に設定されます)。

Stream ID の最下位 2bit (0x2) は 単方向ストリームと双方向ストリームの両者を区別します。

単方向ストリームでは常にこの bit は 1 に設定され、双方向ストリームではこの bit は 0 に設定されます。

ストリームの並行性

QUIC では任意の数のストリームを並行で操作することができます。エンドポイントは最大の Stream ID を制限することにより、並行して受信できる有効な入力ストリームの個数を制限できます。

Stream ID の上限はエンドポイント特有で、設定を受け取ったピアにのみ適用されます。

データの送受信

エンドポイントはデータの送受信にストリームを利用します。それがつまるところストリームの究極の目的です。

ストリームは順序付けられたバイトストリームの概念です。

別々のストリームは必ずしも元の順序で配信されるとは限りません。

ストリームの優先順位付け

ストリームに割り当てられたリソースに正しい優先順位付けがされているのならば、ストリームの多重化はアプリケーションのパフォーマンスに莫大な効果を与えます。

HTTP/2 のような他の多重化されたプロトコルでの経験から言って、効果的な優先順位付けの計画はパフォーマンスに莫大なプラスの影響を持ちます。

QUIC 自身は優先順位付けの情報を交換するフレームを持ちません。そのかわり、QUIC を利用するアプリケーションからの優先順位情報を信頼します。

QUIC を利用するプロトコルはそのアプリケーションのセマンティクスに合った優先順位付けのスキームを定義することが出来ます。
HTTP/3 に QUIC を利用する場合、HTTP レイヤーにおいては優先順位付けは不要です。

HTTP/2 の優先順位付けモデルは、それが過度に複雑で多くの HTTP/2 サーバにおいて使用および実装されていないことを懸念され、批判されています。現時点では、優先順位付けはメインの HTTP/3 仕様から削除されており、個別の仕様としての取り組みが行われています。

0-RTT

新しいコネクションを確立するのに必要な時間を短縮するため、以前にサーバーと接続していたクライアントは特定のパラメータをキャッシュし、そのパラメータを使ってサーバーとの **0-RTT** コネクションを確立してよいことになっています。これにより、ハンドシェイクの完了を待つことなく、クライアントからすぐにデータを送信することが可能になります。

Spin Bit

QUIC ワーキンググループの中でおそらく最も長い間、多数のメールと議論の時間を費やしてきた主題の一つが、単一ビット、つまり Spin Bit です。

このビットの支持者たちは、2つの QUIC エンドポイント間の経路上に存在するオペレータや人々がレイテンシの観測を出来るようになる必要がある、と主張しています。

この機能の反対者たちは、その潜在的な情報漏洩が気に入らないのです。

Spinning a bit

クライアントとサーバー、双方のエンドポイントは各 QUIC コネクションごとに0か1のスピン値が保持されており、その QUIC コネクションの通信パケットは、spin bit を適切な値にセットして送信しています。

両サイドは spin bit が1往復する間は同じ値がセットされたパケットを送り、次の値にトグルします。その影響は、観測者がモニタ可能な形でビットフィールド内の0か1のパルスとして現れます。

この観測は、送信者がアプリケーションやフロー制御による制限を受けていないときにのみ機能し、ネットワーク上のパケットの並び替えはデータを意図しないものにしてしまうことがあります。

ユーザー空間

ユーザー空間にトランスポートプロトコルを実装することで、プロトコルの開発サイクルを速めることができます。これは、クライアントとサーバー OS のカーネルをアップデートすることなくプロトコルの更新を比較的簡単に行えるためです。

技術的には、QUIC 固有の仕組みをユーザー空間ではなくカーネル空間にて実装することも可能です。一部の開発者にとってはそのほうが都合が良いこともあるでしょう。

多数の実装

ユーザー空間に新しいトランスポートプロトコルを実装する明らかな効果は、独立した実装を多数期待できることです。

将来、種々のアプリケーションは異なる HTTP/3 や QUIC の実装を取り込むことが (最上層に実装することも) 起こりうるでしょう。

API

通常の TCP やそれを使うプログラムの成功要因の一つは、標準化されたソケット API です。

機能は明確に定義されており、その API を利用することで TCP 同様に、同一のプログラムを多くの異なる OS 間で動かすことができます。

QUIC の場合は違います。QUIC には標準 API はありません。

QUIC においては既存のライブラリ実装を一つ選択し、その API を使い続ける必要があります。

これによりアプリケーションを、単一のライブラリにある程度まで「ロックイン」することになります。

別のライブラリへの変更は別の API への変更を意味し、多くの作業が必要になるかもしれません。

また QUIC は通常はユーザー空間で実装されているため、ソケット API を単に拡張することも、既存の TCP や UDP の機能が行っていることと同じようなことを提供することも簡単ではありません。

QUIC の使用は、ソケット API とは別の API を使おうとすることを意味します。

HTTP/3

前述のとおり、QUIC 上でトランスポートする最初の基本的なプロトコルは HTTP です。

かつて HTTP/2 が完全に新しい方法を用いて、ネットワーク経由で HTTP をトランスポートするために導入されたのとよく似ていて、HTTP/3 もまた、ネットワーク上で HTTP 送信を行う新しい方法を導入します。

HTTP は今でも以前と同じパラダイムとコンセプトを維持しています。

ヘッダとボディがあり、リクエストとレスポンスがあります。

HTTP メソッドがあり、クッキーがあり、キャッシュ機能があります。

HTTP/3 で主に変わったことは、どのようにビットを伝達相手に送るかです。

HTTP over QUIC を実現するために、様々な変更が必要とされ、その成果がいま私達が HTTP/3 と呼んでいるものです。
これらの変更は QUIC が TCP とは異なった性質を持つために必要とされました。

これらの変更は、以下のものを含んでいます。

- QUIC において、ストリームはトランスポート自体により提供されますが、その一方で HTTP/2 において、ストリームは HTTP レイヤーで実行されます。
- ストリームが互いに独自のものであるため、HTTP/2 で使われているヘッダ圧縮プロトコルを使用すると head-of-line ブロック状態を引き起こすようになりました。
- QUIC ストリームは HTTP/2 ストリームとは僅かに異なります。HTTP/3 セクションでは、これについてやや詳細に説明します。

HTTPS:// の URL

HTTP/3 は `HTTPS://` URL を利用して実行されます。

世界はすでにこれらの URL で満ち溢れていて、新しいプロトコルのために別の URL スキームを導入することは現実的ではなく、とても無理があると考えられています。

HTTP/2 が新しいスキームを必要としなかったように HTTP/3 もまた同様です。

HTTP/3 で状況がさらに複雑になったのは、HTTP/2 が完全に新しい方法で HTTP を転送するプロトコルであったものの、それがまだ TLS や TCP など HTTP/1 の仕様に基づいていたからです。

HTTP/3 が QUIC 上で行われるということは、いくつかの重要な側面において物事を変えることになります。

従来の平文 `HTTP://` URL は現在のまま残りますが、私達が安全な転送へとさらに進んでいくにあたり、おそらく徐々に使われなくなっていくでしょう。こうした平文 URL へのリクエストは簡単には HTTP/3 へアップグレードされません。現実にはその他の理由により HTTP/2 にも稀にしかアップグレードされません。

最初のコネクション

以前に訪れたことがない新しい HTTPS:// URL のホストへの最初のコネクションは、おそらく TCP 経由で行われる必要があります (加えて QUIC 経由での接続を並行して試みることもあるでしょう) 。

接続先のホストは QUIC をサポートしないレガシーなサーバーかもしれませんし、その中間に QUIC コネクションの障壁となるミドルボックスが存在するかもしれません。

モダンなクライアントやサーバーであれば、おそらく最初のハンドシェイクで HTTP/2 をネゴシエートします。

コネクションが確立され、サーバーがクライアントの HTTP 要求に応答する時に、サーバーはクライアントに対して自身の HTTP/3 のサポートとその優先度を伝えることができます。

Alt-svc を使ったブートストラップ

Alternate service (Alt-svc:) ヘッダとそれに対応する `ALT-SVC` HTTP/2 フレームは、QUIC や HTTP/3 用に設計されたわけではありません。

これらはサーバーがクライアントに対して「*ちなみに私は同じサービスをこのホスト、プロトコル、ポートでも実行していますよ*」と伝えるための、既に設計・作成されているメカニズムの一部です。

詳細は RFC 7838 を参照してください。

このような Alt-svc 応答を受け取ったクライアントは、クライアントが代替プロトコルをサポートしておりかつそれを希望する場合に、指定されたプロトコルで当該ホストへバックグラウンドで並行接続をし、その接続が成功した場合には最初のコネクションの代わりにそのプロトコルへと切り替えを行います。

最初のコネクションが HTTP/2、または HTTP/1 を使用している場合、サーバーはクライアントに対して HTTP/3 で再接続可能であると伝えることができます。それは同じホストに対してかもしれませんし、そのオリジンを提供することが可能な別のホストかもしれません。

こうした Alt-svc 応答で与えられる情報には、ある特定の期間だけクライアントが提案された代替プロトコルを使用して後続のコネクションおよび要求を代替ホストに直接指示できるよう、有効期限が設けられます。

Example

HTTP サーバーはレスポンスヘッダに `Alt-Svc:` を含みます:

```
1 Alt-Svc: h3=":50781"
```

これは HTTP/3 がこのレスポンスを得るために使われた同じホスト名の UDP ポート 50781 番で利用可能であることを示します。

クライアントはその宛先ホストに対して QUIC コネクションの確立を試みることが可能で、成功した場合は最初の HTTP バージョンではなく、確立されたその代替接続でオリジンと通信を続けることができます。

QUIC ストリームと HTTP/3

HTTP/2 では完全なストリームと多重化のコンセプトを TCP 上で独自に設計する必要がありました。

一方 HTTP/3 は QUIC 用に作られたので、QUIC のストリームを最大限に活用することができます。

HTTP/3 を介して行われる HTTP リクエストはストリームの特定のセットを利用します。

HTTP/3 フレーム

HTTP/3 はつまり QUIC ストリームを作成し、フレームのセットを通信先の相手へ送信することを意味します。

HTTP/3 にはいくつかの (2018年12月18日の時点では9個！) フレームが存在します。中でも最も重要なものは次のとおりです:

- HEADERS 圧縮された HTTP ヘッダを送信
- DATA バイナリデータを送信
- GOAWAY このコネクションをシャットダウンしてください

HTTP リクエスト

クライアントは、自身が開始した *双方向* QUIC ストリーム上で HTTP リクエストを送信します。

1つのリクエストは1つの HEADERS フレームで構成され、場合によっては他に1、2つのフレームが続きます: 一連の DATA フレーム、またはトレーラー用の 最後の HEADERSリクエストを送信した後、クライアントはストリームを閉じます。

HTTP レスポンス

サーバーは、HTTP レスポンスを双方向ストリーム上で返します。

一連の DATA フレーム、またはトレーラー HEADERS フレームからなる1つの HEADERS フレーム。

QPACK ヘッダ

HEADERS フレームには QPACK アルゴリズムにより圧縮された HTTP ヘッダが含まれています。

QPACK は HTTP/2 で使われている HPACK 圧縮 (RFC7541) と似ていますが、送信されるストリームの順序に関わらず動作するように変更されています。

QPACK は エンドポイント間で2つの単方向 QUIC ストリームを加えて利用します。

これらのストリームは動的なテーブル情報をそれぞれ伝えるために使われます。

プライオリティ制御

前述のとおり、ストリーム間の優先順位付けはメインの HTTP/3 仕様から削除され、個別の取り組みになりました。これは、HTTP/2 の優先順位付けモデルとその実世界における実装 (またはその欠如) から学んだことです。
HTTP ヘッダフィールドと限られた数の優先度設定による HTTP/2 よりも簡易的な優先順位付けモデルが提案されています。これは HTTP/2 ヘッダフレームの "Dependency" と "Weight" フラグからの重要な変更で、アプリケーションレイヤがより理解できるようにしました。

優先順位変更の仕組みとこれをサポートするかどうかはまだ議論中です。HTTP/2 はこれに対応するための優先順位フレームがありますが、QUIC と HTTP/3 の真に独立したストリームはこれをより複雑にするので、複雑さに対する利益についてまだ議論しています。

より良い優先順位付けモデルが HTTP/3 で (もし!) 合意された場合、それを HTTP/2 にもバックポートし、そこにある複雑さと実装に関する懸念に対処することが望まれます。

Server push

HTTP/3 サーバープッシュは HTTP/2 で説明されているもの (RFC7540) と似ていますが、異なるメカニズムを利用します。

サーバープッシュは、クライアントからのリクエストがなくても返すことができるレスポンスです。

サーバープッシュはクライアント側が合意した場合にのみ受け取ることが可能です。

加えて HTTP/3 では、クライアントがサーバーに対してプッシュストリームの最大 ID を通知することでプッシュをいくつまで受け入れ可能か制限を設定しています。この制限を超えると、コネクションエラーとなります。

サーバーが、クライアントからの要求はなかったが、いずれ必要になる追加のリソースが他にもあると判断する場合、レスポンスがプッシュであることを示す `PUSH_PROMISE` フレームをリクエストフレームを通して送ることができ、その後実際のレスポンスを新しいストリーム上で送信します。

予めクライアントによってプッシュの受け入れが可能であると通知されている場合であっても、それぞれのプッシュストリームはクライアント側の判断によって拒否することができます。

その際には `CANCEL_PUSH` フレームがサーバーへ送信されます。

問題点

サーバープッシュは HTTP/2 の開発時に当初議論され、プロトコルの策定の後インターネットへと展開されたにもかかわらず、その有用性について数えきれないほどの議論が行われ、叩かれ嫌われ続けてきました。

プッシュは決して "無料" ではありません。ラウンドトリップの半分を削減することができますが、それでも帯域を使うことに変わりないからです。サーバー側にとって、リソースが実際にプッシュされるべきかどうかをハイレベルで確実に判断することは非常に難しく、おおよそ不可能です。

HTTP/2 と比較した HTTP/3

HTTP/3 は、自身でストリームを処理するトランスポートプロトコルである QUIC に合わせた設計になっています。

HTTP/2 は TCP を前提とした設計のため、HTTP レイヤーでストリームを処理します。

類似点

この2つのプロトコルは、クライアントに対して事実上同じ機能を提供します。

- どちらのプロトコルも、サーバープッシュのサポートを提供します。
- どちらのプロトコルもヘッダー圧縮が提供され、QPACK と HPACK は設計上似ています。
- どちらのプロトコルも、1つの接続でストリームによる多重化が提供されます。

相違点

差異は主に詳細にあり、HTTP/3 が QUIC を利用することによるものです。

- TCP Fast Open と TLS では、一般的にあまりデータ量を送信せず、しばしば問題を引き起こす一方で、HTTP/3 は QUIC の 0-RTT ハンドシェイクのおかげで、early data のサポートがあります。
- HTTP/3 は QUIC のおかげで TCP と TLS の組み合わせより高速なハンドシェイクを実現します。
- HTTP/3 では全て暗号化された安全な通信です。HTTP/2 は、インターネット上においては稀なことであるにせよ、 HTTPS なしで実装することも可能です。
- HTTP/2 では TLS ハンドシェイクを ALPN 拡張 を用いて、直接ネゴシエーションが可能です。一方で、HTTP/3 は 事実上、QUIC 上で動くため、クライアントにこの内容を知らせるため `Alt-Svc:` ヘッダーレスポンス が最初に必要です。
- HTTP/3 は優先度制御を提供しません。HTTP/3 において採用予定だった HTTP/2 の優先度制御に対するアプローチは、複雑すぎるかあるいは紛れもない失敗とみなされており、より単純な仕組みを作る取り組みが行われています。現在予定されているより単純な仕組みは、HTTP/2にバックポートし、HTTP/2の拡張仕様の優先度制御を利用することです。

よくある疑問点

UDP は多くの企業や組織で動くものではない

多くの企業、運用者、組織は、昨今においてほとんどが攻撃に悪用されるという理由のため、ポート53 (DNS に使われる) 以外の UDP トラフィックをブロックするか、あるいは利用制限をかけています。

特に、いくつかの既存 UDP プロトコルや UDP プロトコルを用いた一般的なサーバー上の実装はアンプ攻撃に対して脆弱であり続けていて、攻撃者は大量のトラフィックを無関係な第三者に送りつけることができます。

QUIC ではアンプ攻撃を軽減する方法が備わっています。サーバーから受け取る最初のパケットは最低でも1,200バイトを要求するとともに、クライアントからのレスポンスのパケットを受け取っていない場合では、応答としてサーバーは3つよりも多くのパケットを送ってはならない (MUST NOT)、とプロトコル上で明記されています。

UDP はカーネル内で遅い

少なくとも初期においては、 UDP がカーネル内で遅い点は正しいように思われます。

純粋に長年の間 UDP 転送のパフォーマンスは開発者の関心を集めてこなかったため、果たしてどの程度遅いのか、今後どのように発展していくのかは分かりません。

ほとんどのクライアントにとって、この「遅さ」が気になることはないはずです。

QUIC は CPU 使用量が高すぎる

先述の「UDP は遅い」と同様ですが、これも TCP や TLS の世界的な普及がより成熟したり、ハードウェア支援ができるまでに必要な時間をかけているため、CPU 使用量が高すぎる点に関してもあまり当てはまりません。

もちろん、時間を経るごとに、こういった改善が見込めます。そして、既にこの問題に対するいくつかの改善が見えています。問題は利用者がどれだけ余剰な CPU 使用の増加を気にしなければならないかです。

これは Google の規格でしょ？

いいえ、違います。 Google はインターネット規模で UDP を用いた QUIC 同様の規格が正しく動き、良いパフォーマンスであることを証明し、最初の仕様を IETF へ提案しました。

それ以来、多数の企業や組織から参加している個人が、ベンダーニュートラル な IETF でプロトコルの標準化に取り組んでいます。

もちろん Google の従業員は参加していますが、他にも Mozilla、Fastly、Cloudflare、Akamai、Microsoft、Facebook、Appleなど、インターネット上のトランスポートプロトコルの発展に興味を持つ多くの企業の従業員が参加しています。

改善にしては小さすぎる

それは批判ではなく、1つの意見です。おそらくその通りで、HTTP/2 が展開されてからの改善としては非常に小さいものかもしれません。

HTTP/3 はパケットロスの多いネットワーク上でより優れた性能を発揮し、より高速なハンドシェイクによって数字上でも体感でもレイテンシの改善が見込まれます。

しかしそれがサーバーやサービスへ HTTP/3 サポートを導入するほどのメリットであると言えるでしょうか？

時が経てば、そして将来のパフォーマンス測定結果が間違いなくそれを教えてくれるでしょう！

The specifications (ja/仕様)

ここでは QUIC や HTTP/3 の構成に関する最新のドラフトをまとめています。

Invariants (不変事項)

Version-Independent Properties of QUIC (バージョンに依存しない QUIC の要素)

Transport (トランスポート プロトコル)

QUIC: A UDP-Based Multiplexed and Secure Transport (QUIC: UDP をベースにした多重でセキュアなトランスポートプロトコル)

Recovery (パケットの修復)

QUIC Loss Detection and Congestion Control (QUIC のパケットロスの検知機能と輻輳制御)

TLS

Using TLS to Secure QUIC (TLS を用いたセキュアな QUIC)

HTTP

Hypertext Transfer Protocol Version 3 (HTTP/3)

QPACK

QPACK: Header Compression for HTTP/3 (QPACK: HTTP/3 のためのヘッダー圧縮方法)

QUIC v2

コアの QUIC プロトコルに重点を置き、予定通りリリースできるようにするため、もともとコアプロトコルの一部として計画されていたいくつかの機能の実装が延期されました。延期された機能は QUIC バージョン2もしくはそれ以降のバージョンで実装される予定です。

しかし、この文書の著者はかなり不完全な水晶玉を持っているため、バージョン2ではどのような機能が実装されるのか正確に予測できません。

ただし、v1 から明示的に削除された機能や事柄については「今後実装する」とも言えます。それらはバージョン2にて再び実装される可能性があります。

前方誤り訂正

前方誤り訂正 (FEC) は、送信機が冗長データを送信し受信機がエラーを含まないデータのみを認識する、データ送信においてエラーを制御する方法です。

Google はオリジナルの QUIC の動作でこれを実験しましたが、その後実験はうまくいかなかったため再度削除されました。この機能は QUIC v2 の議論の対象となりますが、おそらく誰かが多くのペナルティなしで有用な追加になることを証明する必要があります。

マルチパス

マルチパスとは、トランスポート自体が複数のネットワーク経路を使用してリソースの使用率を最大化することで、冗長性を高めることを意味します。

世界中の SCTP 支持者達は、SCTP はマルチパスを備えていると主張していますが、現代の TCP もマルチパスを備えています。

信頼できないデータ

「信頼できない」ストリームをオプションとして提供することが検討されているため、UDP スタイルのアプリケーションを QUIC へ置き換えることができます。

HTTP 以外の対応

DNS over QUIC は、QUIC v1 と HTTP/3 がリリースされる際に注目されるかもしれない非 HTTP プロトコルの1つでした。

しかし、新しいトランスポートが世界中にもたらされれば、他にもこのようなトランスポートが登場するかもしれません。

한국어

이 책은 2018년 3월에 작성하기 시작했다. HTTP/3와 그 기반 프로토콜인 QUIC이 왜, 어떻게 동작하는지와 프로토콜의 상세내용, 그 구현체 등을 설명할 계획이다.

이 책은 완전히 무료이므로 돕고자 하는 사람은 누구나 참여해서 같이 만들 수 있다.

사전 요구사항

이 책의 독자는 TCP/IP 네트워크, HTTP의 기본, 웹을 어느 정도 이해하고 있다고 가정한다. HTTP/2에 대해서 더 알고 싶다면 http2 explained를 읽어보기를 권장한다.

저자

이 책은 Daniel Stenberg가 시작해서 만들어졌다. Daniel은 세계에서 가장 널리 사용되는 HTTP 클라이언트 소프트웨어인 curl을 만든 사람이자 curl의 리드 개발자다. Daniel은 20년 넘게 HTTP와 인터넷 프로토콜로 일했고 http2 explained의 저자이기도 하다.

홈페이지

이 책의 홈페이지는 daniel.haxx.se/http3-explained다.

도움 요청

이 문서에서 실수, 누락, 오류, 속 보이는 거짓말 등을 발견한다면 해당 문단의 수정 버전과 함께 보내주면 수정하고 도움 준 사람에게 적절한 크레딧을 줄 예정이다. 이 문서가 점점 나아지기를 기대한다.

가능하면 이 책의 GitHub 페이지에 이슈를 생성하거나 풀 리퀘스트를 제출해주기 바란다.

라이센스

왜 QUIC인가

QUIC은 약어가 아니라 이름이다. 영어단어 "quick"과 똑같이 발음한다.

QUIC은 많은 부분에서 HTTP 같은 프로토콜에 적합하다. TCP와 TLS에서 동작하는 HTTP/2의 단점으로 알려진 문제를 해결하면서 안정적이고 안전한 새 전송 프로토콜로 볼 수 있다.

QUIC은 HTTP 전송에만 국한되지 않는다. 최종 사용자에게 웹과 데이터를 더 빨리 전달하려는 바람이 초기 이 새로운 전송 프로토콜을 만들게 된 가장 큰 이유이자 원동력이다.

그러면 새로운 전송 프로토콜은 왜 만들고 왜 UDP 상에서 동작하게 했는가?

QUIC 로고

HTTP/2를 기억하는가?

HTTP/2 명세인 RFC 7540는 2015년 5월에 발행되었고 이후 인터넷과 월드 와이드 웹에 널리 구현되고 배포되었다.

2018년 초 상위 1,000개의 웹사이트 중 거의 40%가 HTTP/2로 동작하고 있으며 Firefox가 보낸 모든 HTTPS 요청의 70% 정도가 HTTP/2 응답을 받았고 주요 모든 브라우저와 서버, 프락시가 이를 지원하고 있다.

HTTP/2는 HTTP/1의 수많은 결점을 수정했고 HTTP의 두 번째 버전을 도입함으로써 사용자들이 수많은 우

회법을 사용하지 않게 되었다. 그 중 일부는 웹 개발자에게 상당한 부담이었다.

HTTP/2의 주요 기능 중 하나인 멀티플렉싱을 사용하여 같은 물리 TCP 연결을 통해 다수의 논리 스트림을 보낼 수 있게 된 것이다. 이는 많은 것을 더 좋고 빠르게 만들었다. 또한, 혼잡 제어 작업을 훨씬 낫게 해주어 사용자가 TCP를 훨씬 잘 사용하게 되면서 대역폭을 적절하게 가득 채워서 사용해 TCP 연결을 더 오래 유지되도록 만들었다. 이는 이전보다 더 자주 최대 속도를 낼 수 있기 때문에 좋아진 것이다. 헤더 압축은 대역폭을 적게 사용하게끔 해준다.

이전에는 브라우저가 호스트당 *6개의* TCP 연결을 사용했지만, HTTP/2를 사용하면 보통 *하나의* TCP 연결을 사용한다. 사실 HTTP/2에서 연결 병합과 "desharding" 기술을 사용하면 연결 수를 훨씬 더 줄일 수도 있다.

HTTP/2는 클라이언트가 다음 요청을 보내기 전에 첫 요청이 끝나기를 기다려야 하는 HTTP HOL(head of line) 블로킹 문제를 고쳤다.

http2 man

TCP HOL(ko/head of line) 블로킹

TCP HOL(head of line) 블로킹

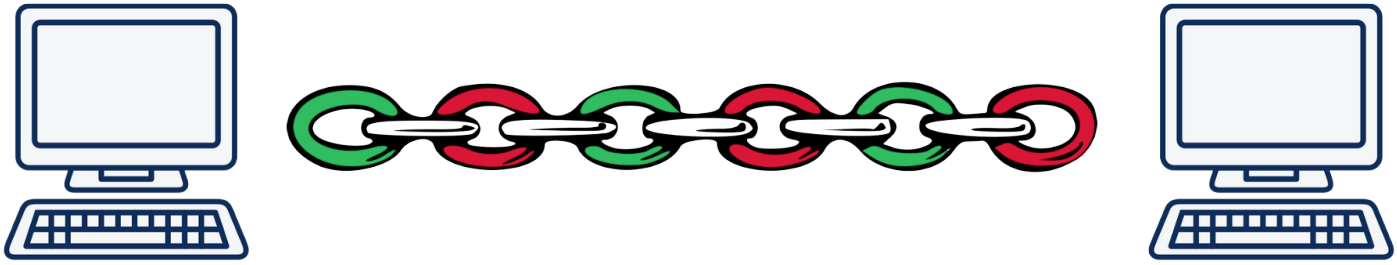
HTTP/2는 TCP를 사용하며 이전 HTTP 버전을 사용할 때 보다 더 적은 TCP 연결을 사용한다. TCP는 신뢰할 수 있는 전송 프로토콜이고 기본적으로 두 머신 간의 가상 체인으로 생각해도 된다. 네트워크의 한쪽 끝에 넣은 것이 최종적으로 다른 쪽 끝에 같은 순서로 나올 것이다.(아니면 연결이 끊어진다.)

두 컴퓨터 사이의 TCP 체인

HTTP/2를 사용하는 일반적인 브라우저는 TCP 연결 한개로 수십, 수백 개의 병렬 전송을 한다.

HTTP/2로 통신하는 두 엔드포인트 사이 네트워크 어딘가에서 하나의 패킷이 빠지거나 없어진다면 없어진 패킷을 다시 전송하고 목적지를 찾는 동안 전체 TCP 연결이 중단되게 된다. 즉, TCP는 "체인"이기 때문에 한 링크가 갑자기 사라지면 그 링크 이후에 와야 하는 모든 것들이 기다려야 한다는 뜻이다.

체인 메타포를 사용해서 이 연결을 통해 두 스트림을 보내는 경우를 그린 그림을 보자. 빨간색 스트림과 녹색 스트림이 있다.



체인에서 링크는 다른 색으로 보여준다

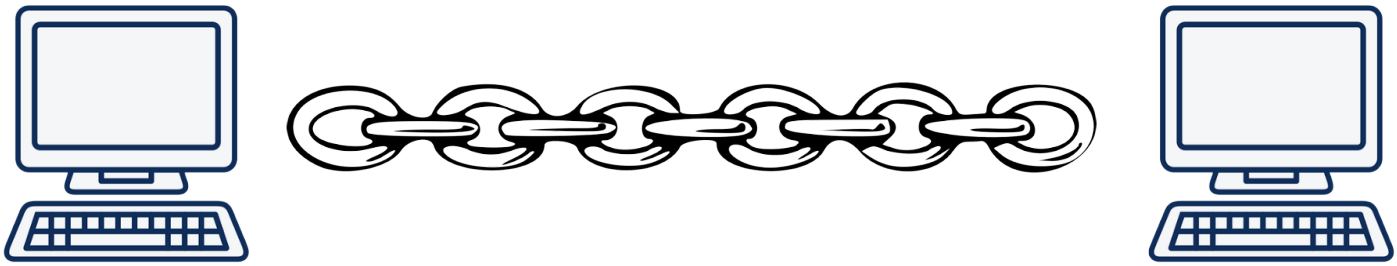
이는 TCP에 기반을 둔 head of line 블로킹이 된다!

패킷 손실률이 증가하면 HTTP/2의 성능도 저하된다. 테스트를 통해 패킷 손실률이 2%(상기하자면, 이는 끔찍한 네트워크 품질이다) 일때 HTTP/1 사용자가 더 나은 것으로 입증했다. 그 이유는 HTTP/1은 보통 손실된 패킷을 분배하는데 6개의 TCP 연결을 갖고 있어서 손실된 패킷이 없는 다른 연결은 계속 사용할 수 있기 때문이다.

TCP를 사용하는 한 이 이슈를 고치는 것은 (가능할 수도 있지만) 쉽지 않다.

차단을 피하는 독립 스트림

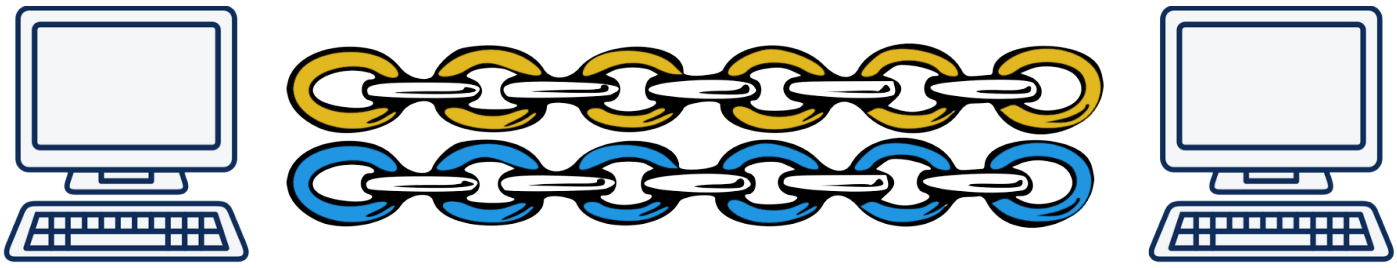
QUIC에는 두 엔드포인트간에 연결을 안전하게 하고 데이터 전달을 신뢰할 수 있게 하는 연결 설정이 있다.



두 컴퓨터 사이의 QUIC 체인

이 연결을 통해 두 가지 다른 스트림을 설정했을 때 이들을 독립적으로 다루므로 스트림 중 하나에서 어떤 링크가 사라지더라도 해당 스트림만(특정 체인) 멈추고 재전송될 없어진 링크를 기다란다.

다음은 두 엔드포인트간에 노란색과 파란색 스트림을 보내는 그림이다.



두 컴퓨터 사이의 두 가지 QUIC 스트림

TCP or UDP

TCP 혹은 UDP

TCP에서 head-of-line 블로킹을 해결할 수 없다면 이론적으로 네트워크 스택에서 UDP와 TCP 옆에 새로운 전송 프로토콜을 만들 수 있다. 아니면 RFC 4960에서 IETF가 표준화하고 여러 가지 원하는 특성이 있는 전송 프로토콜인 SCTP를 사용할 수도 있다.

하지만, 최근 수년간 새로운 전송 프로토콜을 만들려는 노력은 인터넷에서 그것을 배포하는 어려움 때문에 완전히 멈춰 있다. 새 프로토콜 배포는 그 프로토콜이 도달해야 하는 사용자와 서버 사이에 있는 TCP와 UDP만 허용하는 방화벽, NAT, 라우터 등의 미들박스에 의해 방해받고 있다. 다른 전송 프로토콜의 도입은 UDP 또는 TCP 가 아닌 경우 이를 악의적이거나 뭔가 잘못되었다고 판단하고 차단해 버리는 박스들에 의해 차단되므로 연결의 N%가 실패한다. 노력을 기울이기에 N% 실패율은 너무 높다고 여겨진다.

게다가 보통 네트워크 스택의 전송 프로토콜 계층에서 뭔가를 바꾼다는 것은 보통 운영체제 커널에서 구현한 프로토콜을 말한다. 새로운 운영체제 커널을 갱신하고 배포하는 것은 상당한 노력이 필요한 느린 과정이다. IETF가 표준화한 수많은 TCP 개선사항도 광범위하게 지원되지 않아서 널리 배포되거나 사용되지 않고 있다.

왜 SCTP-over-UDP가 아닌가

SCTP는 스트림을 가진 신뢰성 있는 전송 프로토콜이고 WebRTC처럼 UDP 위에서 이 프로토콜을 사용하는 기존의 구현체도 있다.

다음의 몇 가지 이유로 QUIC의 대체재로 충분하지 않다고 생각했다.

- SCTP는 스트림에 대해 head-of-line 블로킹 문제를 고치지 못한다
- SCTP는 연결을 설정할 때 스트림의 수를 결정해야 한다
- SCTP에는 견고한 TLS/보안에 대한 언급이 없다
- SCTP는 4단계 핸드쉐이크(4-way handshake)를 사용하고 QUIC은 0-RTT를 제공한다
- QUIC는 TCP 같은 바이트스트림(bytestream)이고 SCTP는 메시지 기반이다
- QUIC 연결은 IP 주소 사이에서 마이그레이션을 할 수 있지만 SCTP는 할 수 없다

자세한 차이점이 알고 싶다면 A Comparison between SCTP and QUIC을 참고하라.

고착화(ko/Ossification)

인터넷은 네트워크의 네트워크다. 이 네트워크의 네트워크가 의도대로 동작하게 하는 장비가 인터넷 여러 곳에 설치되어 있다. 우리는 이러한 기기(네트워크에 분포된 박스)를 미들박스라고 부른다. 이 박스는 전통적인 네트워크 데이터 전송의 주요 요소인 두 엔드포인트 사이에 있다.

박스는 여러 가지 다른 목적이 있지만 무언가 동작하게 하려면 이 박스가 해당 위치에 있어야 한다고 누군가 생각했기 때문에 곳곳에 깔린 것이라 생각한다.

미들박스는 네트워크 사이에서 IP 패킷을 라우팅하고, 악성 트래픽을 차단하고, NAT(Network Address Translation) 역할을 하고, 성능을 향상시키고, 통과하는 트래픽을 감시하는 등의 역할을 한다.

이러한 박스는 의무를 다하려면 자신들이 모니터링하고 수정하는 네트워크 및 프로토콜에 대해 반드시 알아야 한다. 박스는 이런 목적으로 소프트웨어를 실행한다. 그 소프트웨어를 항상 자주 업그레이드하는 것은 아니다.

박스는 인터넷이 유지되도록 연결하는 구성 요소이지만 종종 최신 기술을 따라잡지 못하는 경우가 있다. 보통 네트워크 중간 영역은 전 세계의 클라이언트나 서버 같은 엣지 만큼 빠르게 바뀌지 않는다.

어떤 프로토콜을 검사하기를 원하는지, 그리고 어떤것이 괜찮고 안 괜찮은지 알고있는 박스들은 과거에 배포되었고 그 당시의 프로토콜 기능 셋을 가지고 있다. 이전에는 몰랐던 새로운 기능이나 동작의 변경 사항이 추가되면 박스가 이를 잘못된 것이나 허용하지 않는 것으로 판단할 위험이 있다. 그래서 이러한 트래픽은 사용자가 해당 기능을 사용하고 싶지 않을 정도로 지연되거나 중단될 수 있다.

이를 "프로토콜 고착화(ossification)"라고 부른다.

TCP를 변경하는 것도 고착화 문제를 겪는다. 클라이언트와 원격 서버 사이에 있는 박스 중 일부는 알지 못하는 새로운 TCP 옵션을 발견하고 이 옵션이 무엇인지 몰라서 해당 연결을 차단해버릴 것이다. 프로토콜의 상세 내용을 탐지하도록 허용한 경우 시스템은 프로토콜이 보통 어떻게 동작하는지 배우게 되고 시간이 지나면 프로토콜을 변경할 수 없게 된다.

유일하게 고착화를 "방지하는데" 효과적인 방법은 미들박스가 지나가는 프로토콜에서 많은 것을 볼 수 없도록 통신을 최대한 암호화하는 것이다.

안전

QUIC은 항상 안전하다. 이 프로토콜에는 일반 텍스트 버전이 없으므로 QUIC 연결과 협상하려면 TLS 1.3을 사용해서 암호화 및 보안을 수행해야 한다. 위에서 언급했다시피, 이는 다른 차단이나 특별한 처리 같은 고착화 문제를 방지할 뿐만 아니라 QUIC이 웹 사용자가 기대하고 원하는 HTTPS의 모든 보안 속성을 가지게 만든다.

암호화 프로토콜 협상이 이뤄지기 전 초기 핸드쉐이크 패킷에 일반 텍스트 메시지가 아주 조금 있다.

감소된 지연시간

QUIC는 0-RTT, 1-RTT 핸드쉐이크 둘 다 제공하는데 이는 새로운 연결을 협상하고 설정하는데 걸리는 시간을

줄여준다. TCP의 3단계 핸드쉐이크(3-way handshake)와 비교해 보면 된다.
여기에 추가로 QUIC은 더 많은 데이터를 허용하는 "이른 데이터(early data)"를 처음부터 지원하고 이는 TCP Fast Open보다 더 쉽게 사용할 수 있다.

스트림 개념을 사용해서 기존에 존재하는 연결이 끝나기를 먼저 기다릴 필요 없이 같은 호스트로의 또 다른 논리 연결도 동시에 처리될 수 있다.

문제가 있는 TCP Fast Open

TCP Fast Open는 2014년 12월 RFC 7413로 발행되었고 이 명세는 이미 전달된 첫 번째 TCP SYN 패킷에 서버로 보낼 데이터를 전달하는 방법을 설명한다.

현실에서 이 기능의 실제 지원은 시간이 걸렸고 2018년인 오늘날까지도 문제로 가득하다. TCP 스택을 구현하는 사람이 문제를 이미 겪었으므로 애플리케이션이 이 기능의 이점을 취하려고 시도했다. 이를 활성화하기 위해 OS 버전이 무엇인지 알아야 할 뿐만 아니라 문제가 발생하면 그레이스풀하게 철회하고 문제를 다루는 방법을 찾아내야 한다. 여러 네트워크가 TFO 트래픽을 방해하는 것으로 밝혀졌고 이러한 TCP 핸드쉐이크를 적극적으로 망쳤다.

과정

Google의 Jim Roskind가 초기 QUIC 프로토콜을 설계하고 2012년 처음 구현했으며 Google의 실험을 확대한 2013년 전 세계에 공개적으로 발표했다.

그 당시에는 QUIC이 "Quick UDP Internet Connections"의 약자라고 주장했었지만, 이후에는 없어졌다.

Google이 프로토콜을 구현하고 이어서 널리 사용되는 자신들의 브라우저(Chrome)와 서버사이드 서비스(Google 검색, gmail, youtube 등)에 배포했다. Google은 프로토콜 버전을 꽤 빠르게 올리면서 이 개념이 시간이 지남에 따라 엄청난 수의 사용자에게 신뢰할 수 있게 동작한다는 것을 증명했다.

2015년 6월 표준화를 위해 QUIC의 첫 번째 인터넷 드래프트 버전을 IETF에 제출했지만 2016년 후반이 되어서야 QUIC 워킹 그룹이 승인되어 시작되었다. 하지만 이후 바로 많은 단체의 엄청난 관심을 받으면서 시작되었다.

2017년 Google의 QUIC 엔지니어가 인용한 수치에 따르면 *전체* 인터넷 통신량의 약 7%가 이미 QUIC을 사용한다고 한다. 이는 QUIC 프로토콜의 Google 버전이다.

IETF

IETF에서 프로토콜을 표준화하려고 만든 QUIC 워킹 그룹은 QUIC 프로토콜이 "단순히" HTTP가 아닌 다른 프로토콜을 전송할 수 있어야 한다고 빠르게 결정했다. Google-QUIC은 오로지 HTTP만 전송했고 실제로 HTTP/2 프레임 문법을 사용해서 HTTP/2 프레임을 효과적으로 전송했다.

IETF-QUIC은 Google-QUIC이 사용한 "커스텀" 접근 방법이 아니라 TLS 1.3의 암호화와 보안을 기반으로 두어야 한다고도 발표했다.

HTTP보다 더 많이 보내야 한다는 요구를 만족시키기 위해 IETF QUIC 프로토콜 아키텍처를 두 가지 별도의 계층인 전송 QUIC과 "HTTP over QUIC" 계층(후자는 종종 "hq"라고도 한다)으로 분할했다.

무해할 것처럼 들렸던 이 계층 분할로 인해 IETF-QUIC은 원래의 Google-QUIC과 많이 달라졌다.

하지만 워킹 그룹은 곧 적절하게 집중해서 제때 QUIC 버전 1을 제공할 수 있도록 HTTP를 제공하는 데 집중하고 HTTP가 아닌 전송은 나중 작업으로 남겨두기로 했다.

2018년 3월 이 책의 작업을 시작할 때 QUIC 버전 1의 최종 명세를 2018년 11월에 발표 할 계획이었다. 이는 나중에 2019년 7월로 연기되었다.

IETF-QUIC 작업이 진행되는 동안 Google 팀은 IETF 버전의 세부 내용을 받아들였고 IETF 버전이 만들어질 방향으로 Google 버전의 프로토콜을 천천히 발전시켰다. Google은 자사의 브라우저와 서비스에서 QUIC의 Google 버전을 계속해서 사용하고 있다.

개발 중인 대부분의 새로운 구현은 IETF 버전에 집중하고 Google 버전과는 호환되지 않는다.

HTTP/2에서의 경험

HTTP/2 명세 RFC 7540는 2015년 5월 발행되었는데 이는 QUIC이 처음으로 IETF에 들어오기 바로 한 달 전이다.

HTTP/2에서 유선으로 HTTP를 통한 HTTP를 변경할 기반이 마련되었고 HTTP/2를 만든 워킹그룹이 버전 1에서 버전 2로 가는 것보다 새로운 HTTP 버전으로 가는 걸 돕는 것이 훨씬 빠르다고 생각하게 되었다.(약 16년)

HTTP/2를 겪으면서 사용자와 소프트웨어 스택은 HTTP는 더는 텍스트 기반 프로토콜이라고만 가정할 수 없다고 생각하게 되었다.

HTTP-over-QUIC은 2018년 11월에 HTTP/3로 개명되었다.

상태

QUIC 워킹 그룹은 2016년 후반부터 프로토콜을 제정하는 일에 열심히 노력했고 현재 계획은 2019년 7월에 완료하는 것이다.

2018년 11월까지도 HTTP/3의 아직 대규모 연동 테스트를 진행하지 않았다. 기존에 두 개의 구현체가 있는데 둘 다 브라우저나 인기있는 공개 서버 소프트웨어와의 테스트를 하지 않았다.

QUIC 워킹 그룹의 위키 페이지에 나열된 15개 정도의 서로 다른 QUIC 구현체가 있지만 이들 모두 최신 명세 드래프트 개정판과 연동되지 않는다.

QUIC을 구현하는 것은 쉽지 않고 프로토콜은 오늘날 까지도 계속 진화하며 변하고 있다.

서버

지금까지 Apache나 nginx에서 공개적으로 QUIC을 지원한다는 발표는 없다.

클라이언트

아직 대형 브라우저 벤더중 아무도 QUIC이나 HTTP/3의 IETF 버전을 실행할 수 있는 버전을 제공하거나 발표하지 않았다.

수년간 Google Chrome은 Google 자체의 QUIC 버전의 구현체를 포함해서 배포했지만 이는 IETF QUIC 프로토콜과 연동되지 않고 그 HTTP 구현체는 HTTP/3와도 다르다.

구현 방해물

QUIC은 뭔가 새로운 것을 발명하지 않고 신뢰할 수 있는 기존 프로토콜을 사용하고자 TLS 1.3을 암호화와 보안 계층의 기반으로 사용하기로 했다. 하지만 이 작업을 하면서 워킹 그룹은 QUIC에서 TLS의 사용을 실제로 간소화하기로 발표했다. 즉, 프로토콜은 "TLS 레코드"는 사용하지 않고 "TLS 메시지"만 사용해야 한다.

이것이 무해한 변화처럼 들릴 수 있지만 수많은 QUIC 스택 구현자들에게는 커다란 걸림돌이 되었다. TLS 1.3을 지원하는 기존 TLS 라이브러리는 이 기능을 공개하거나 QUIC이 접근할 수 있는 API가 충분치 않다. 더 큰 조직에서 온 몇몇 QUIC 구현자들은 자신들의 TLS 스택에도 동시에 작업을 하고 있지만 모두가 그런 것은 아니다.

예시로 중량급 지배적 오픈 소스인 OpenSSL은 이에 대한 API가 전혀 없고 근래에 제공할 의사를 밝힌 적이 없다.(2018년 11월 기준)

이는 QUIC 스택이 다른 TLS 라이브러리(별도로 수정한 OpenSSL 빌드)를 사용하거나 향후 OpenSSL 버전을 수정할 필요가 있기 때문에 결국 배포 상의 걸림돌이다.

커널과 CPU 부하

Google과 Facebook은 QUIC의 대규모 배포가 TLS에서 HTTP/2로 서비스 할때보다 같은 트래픽 기준으로 거의 2배의 CPU가 필요하다고 얘기했다.

이는 다음과 같은 이유 때문이다.

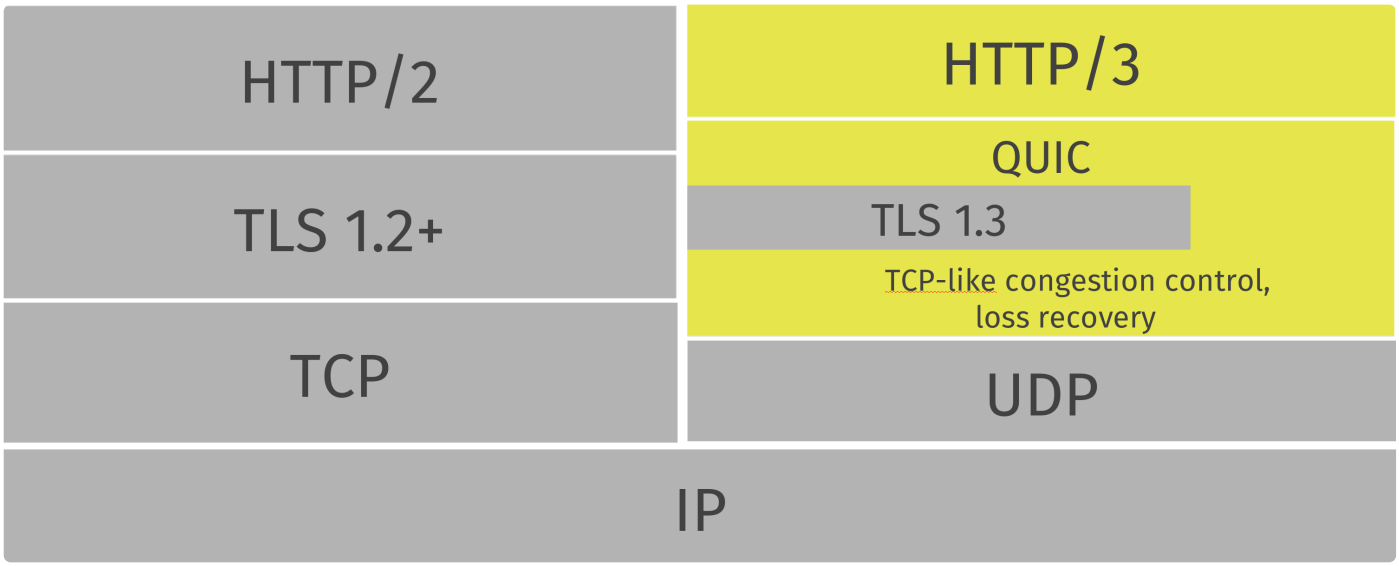
- 본질적으로 Linux의 UDP 부분은 TCP 스택만큼 최적화되어 있지 않다. 이는 전통적으로 지금처럼 고속 전송에 사용해오지 않았기 때문이다.
- 하드웨어가 TCP와 TLS의 부담을 덜어주고 있지만 UDP에 대해서 그렇게 하는 경우는 드물고 기본적으로 QUIC에 대해서는 아예 존재하지 않는다.

시간이 지나면 성능 및 필요한 CPU가 개선될 것이라고 생각한다.

프로토콜 기능

고수준에서 QUIC 프로토콜을 보자.

아래 그림은 HTTP를 전송할 때 HTTP/2 네트워크 스택을 왼쪽에 QUIC 네트워크 스택을 오른쪽에 보여준다.



QUIC 스택

UDP

UDP 상의 전송 프로토콜

QUIC은 UDP 위에 구현한 전송 프로토콜이다. 우리가 임의로 네트워크 트래픽을 보면 QUIC이 UDP 패킷으로 나타나는 것을 볼 것이다.

UDP에 기반을 둔 QUIC은 UDP 포트 번호를 사용해서 주어진 IP 주소의 특정 네트워크 서비스를 식별한다.

현재 알려진 모든 QUIC 구현체는 사용자 영역에 있으므로 커널 영역의 구현체보다 훨씬 빠른 발전이 가능하다.

동작할 것인가?

(DNS에 사용되는) 53 포트가 아닌 포트의 UDP 트래픽을 차단하는 기업용 및 기타 네트워크 설정이 있다. 또 다른 설정은 여러 방법으로 이러한 데이터를 제한하기 때문에 QUIC은 TCP에 기반을 둔 프로토콜보다 성능이 더 좋지 않다. 이 때문에, 몇몇 운영자가 해야 할 일에는 끝이 없다.

가까운 미래에 QUIC 기반의 모든 전송은 아마도 그레이스풀하게 (TCP 기반의) 다른 대안 프로토콜로 전환될 수 있어야 한다. Google 엔지니어는 측정한 실패 비율이 낮은 한 자릿수라고 얘기했다.

나아질 것인가?

QUIC이 인터넷 세상에 가치를 더할 수 있다고 증명한다면 사람들은 사용하기를 원할 것이고 그들의 네트워크에서 동작하길 원할 것이다. 그러면 회사들은 자신들의 장애물을 재고하기 시작할 것이다. 수년간 QUIC 개발은 진척이 있었으며, 인터넷에서 QUIC 연결을 설정하고 사용하는 성공률이 증가하고 있다.

신뢰성

UDP가 데이터 전송의 신뢰성을 보장하지 않지만, QUIC은 UDP 위에 새로운 계층을 추가함으로써 신뢰성을 제공한다. 추가된 계층은 TCP에 존재하는 패킷 재전송, 혼잡 제어, 속도 조정 및 다른 기능들을 제공한다.

한 엔드포인트로부터 QUIC을 통해 전송된 데이터는 연결이 유지되는 한 다른 엔드포인트에서 수신할 수 있다.

스트림

QUIC은 SCTP, SSH, HTTP/2와 마찬가지로 물리 연결 내에서 논리적 스트림을 나눌 수 있다. 하나의 연결로 다수의 병렬 스트림으로 다른 스트림에 영향을 주지 않고 데이터를 동시에 전송할 수 있다.

두 엔드포인트 사이에 연결 협상이 이뤄지는 방식은 TCP 연결이 동작하는 방식과 비슷하다. QUIC 연결은 UDP 포트와 IP 주소로 이루어져 있지만 일단 연결을 만들고 나면 "connection ID"로 연결된다.

만들어진 연결을 통해 양쪽에서 스트림을 만들어 다른 쪽으로 데이터를 보낼 수 있다. 스트림은 순서대로 전달되고 신뢰할 수 있지만 서로 다른 스트림은 순서 없이 전달될 수 있다.

QUIC은 연결과 스트림 모두에서 흐름 제어를 제공한다.

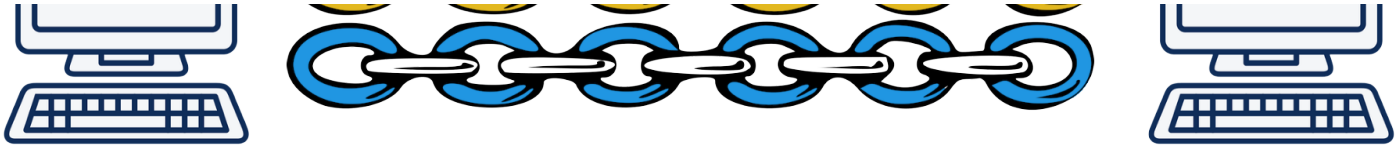
더 자세한 내용은 연결과 스트림 부분을 참고하라.

순서에 맞는 전송

QUIC은 스트림의 순서 있는 전송을 보장하지만 스트림 사이에서는 (순서를) 보장하지 않는다. 스트림은 순서대로 데이터를 전송하고 유지하지만, 각 스트림은 애플리케이션이 보낸 것과는 다른 순서대로 목적지에 도착할 수 있다!

스트림 A와 B가 서버에서 클라이언트로 이동하는 예를 생각해 보자. 스트림 A를 먼저 시작하고 이어서 스트림 B를 시작했다. QUIC에서 잃어버린 패킷은 해당 패킷이 속한 스트림에만 영향을 준다. 스트림 A는 패킷을 잃어버렸지만 스트림 B는 잃어버리지 않았다면 스트림 A의 잃어버린 패킷을 다시 전송하는 동안 스트림 B는 계속해서 전송하면서 완료될 수 있다. 이는 HTTP/2에서는 불가능했다.

다음은 하나의 연결을 통해 두 QUIC 앤드포인트 사이에 보내진 노란색 스트림과 파란색 스트림의 그림이다. 이 두 스트림은 독립적이고 다른 순서로 도착할 것이지만 각 스트림은 신뢰할 수 있게 애플리케이션에 순서대로 전달된다.



두 컴퓨터 사이에 두 QUIC 스트림

빠른 핸드쉐이크

QUIC은 0-RTT와 1-RTT 연결 설정을 둘 다 지원한다. 즉, 최선의 상황에서는 새로운 연결을 설정할 때 추가적인 라운드 트립이 전혀 필요치 않는다. 둘 중 가장 빠른 0-RTT 핸드쉐이크는 호스트에 이미 이전 연결이 구성되어 있고 해당 연결의 시크릿이 캐시 되어 있을 때만 동작한다.

초기 데이터

QUIC은 클라이언트가 이미 0-RTT 핸드쉐이크에 데이터를 넣을 수 있다. 이 기능으로 데이터를 피어에 최대한 빨리 전달할 수 있으므로 서버가 더 빨리 응답하고 데이터를 돌려줄 수 있다.

TLS 1.3

QUIC에서 사용하는 전송 보안은 TLS 1.3(RFC 8446)이며 암호화하지 않은 QUIC 연결은 절대로 존재하지 않는다.

이전 버전의 TLS와 비교해서 TLS 1.3에 몇몇 장점이 있지만 QUIC이 TLS 1.3을 사용한 주된 이유는 핸드쉐이크에 더 적은 라운드 트립이 필요하도록 바뀌었기 때문이다. 이는 프로토콜 지연을 줄여준다.

QUIC의 Google 레거시 버전은 커스텀 암호화를 사용했다.

전송 계층과 애플리케이션 계층

IETF QUIC 프로토콜은 전송 프로토콜로써 다른 애플리케이션 프로토콜이 그 위에서 사용할 수 있다. 첫 애플리케이션 계층 프로토콜은 HTTP/3(h3)다.

전송 계층은 연결과 스트림을 지원한다.

레거시 Google 버전 QUIC은 전송과 HTTP가 하나로 합쳐져 있어서 (IETF QUIC) 보다 더 특수한 목적의 send-http/2-frames-over-udp 프로토콜이었다.

QUIC을 통한 HTTP/3

HTTP/3라고 부르는 HTTP 계층은 HTTP 형태의 전송을 한다. 그 중에는 HPACK이라 부르는 HTTP/2 헤더 압

축과 비슷한 QPACK을 사용한 HTTP 헤더 압축도 있다.
HPACK 알고리즘은 *순차적인* 스트림 전달에 의존하는데 QUIC은 스트림을 순서 없이 전달할 수 있으므로 HPACK을 수정하지 않고는 HTTP/3에서 재사용할 수 없다. HPACK을 QUIC에 맞게 수정한 것을 QPACK이라 볼 수 있다.

QUIC을 통한 HTTP가 아닌 프로토콜

QUIC을 통해 HTTP가 아닌 프로토콜을 보내는 작업은 QUIC 버전 1 출시 이후로 연기되었다.

QUIC의 동작 방식

이번 장에서는 QUIC 전송 프로토콜의 기본적인 구성 요소가 어떻게 동작하는지를 설명한다. QUIC 스택을 직접 구현하고자 한다면 이 책의 설명을 통해 전반적인 내용을 이해할 수는 있지만, 세부 내용은 IETF의 인터넷 드래프트와 RFC를 참고하기 바란다.

1. 연결을 설정한다.
2. 여기에는 TLS 보안이 포함되어 있다.
3. 스트림을 사용한다.

연결

QUIC 연결은 두 QUIC 엔드포인트 사이의 대화이다. QUIC의 연결 설정은 버전 협상, 암호화, 전송 핸드쉐이크로 구성되어 있으므로 연결 설정의 지연시간을 줄여준다.

이러한 연결을 통해 실제 데이터를 보내려면 하나 이상의 스트림을 만들어서 사용해야 한다.

연결 ID

각 연결은 연결 식별자나 연결 ID를 가지므로 이를 통해 연결을 식별한다. 엔드포인트가 자유롭게 연결 ID를 선택한다. 각 엔드포인트는 엔드포인트의 피어가 사용할 연결 ID를 선택한다.

이 연결 ID의 주요 기능은 하위 프로토콜 계층(UDP, IP 혹은 그 아래 계층)에서 주소가 변경되더라도 QUIC 연결의 패킷이 잘못된 엔드포인트로 전달되지 않도록 보장하는 것이다.

연결 ID를 이용하면 TCP에서는 불가능했던 방법으로 IP 주소와 네트워크 인터페이스 사이에서 연결이 마이그레이션 할 수 있다. 예를 들면 사용자가 자신의 기기를 들고 wifi가 지원되는 곳으로 이동했을 때 다운로드를 진행하면서 셀룰러 네트워크 연결에서 더 빠른 wifi 연결로 변경되는 것이 이 마이그레이션을 통해 가능하다. 마찬가지로 wifi를 이용할 수 없게 되었을 때 셀룰러 연결을 통해 다운로드를 진행할 수 있다.

포트 번호

QUIC이 UDP 위에 만들어졌으므로 들어오는 연결을 구분하기 위해 16비트 포트 번호 필드를 사용한다.

버전 협상

클라이언트는 QUIC 연결 요청에서 어떤 QUIC 프로토콜 버전으로 통신하고 싶은지 서버에게 알려주고 서버는 클라이언트가 선택할 수 있도록 지원하는 버전 목록을 응답한다.

TLS을 사용하는 연결

초기 패킷이 연결을 설정하자마자 초기화하는 쪽에서 보안 계층 핸드쉐이크의 설정을 시작하는 암호화 프레임을 보낸다. 보안 계층은 TLS 1.3 보안을 사용한다.

QUIC 연결에서에는 반드시 TLS를 사용해야 한다. QUIC 프로토콜은 프로토콜 고착화를 방지하기 위해 미들박스가 조작하기 어렵게 설계되었다.

스트림

QUIC 스트림은 경량이면서 순서가 있는 바이트 스트림 추상화를 제공한다.

QUIC에는 두가지 기본 스트림 타입이 있다.

- 단방향 스트림은 한쪽으로 데이터를 전달한다. 즉, 스트림을 시작한 쪽에서 피어로 전달된다.
- 양방향 스트림은 양쪽으로 데이터를 보낼 수 있다.

두 엔드포인트가 두 타입의 스트림을 모두 만들 수 있고 스트림은 다른 스트림과 상호배치된 데이터를 동시에 보낼 수 있고 취소할 수도 있다.

QUIC 연결을 통해 데이터를 보낼 때 하나 이상의 스트림을 사용한다.

흐름 제어

스트림은 개별적으로 흐름 제어가 되므로 엔드포인트가 메모리 사용을 제한할 수 있고 백프레셔(back pressure)를 적용할 수도 있다. 스트림의 생성도 흐름 제어가 되고 각 피어는 일정 시간 동안 받고자 하는 최대 스트림 ID를 정의할 수 있다.

스트림 식별자

스트림 ID라고 부르는 부호없는 62비트 정수로 스트림을 식별한다. 스트림 ID의 최하위 2비트를 스트림(단방향이나 양방향이나)의 타입과 스트림을 시작한 쪽을 식별하는 데 사용한다.

스트림 ID의 최하위 비트(0x1)는 누가 스트림을 시작했는지를 식별한다. 클라이언트는 짝수의 스트림(최하위 비트가 0으로 설정된 스트림)을 초기화 하고 서버는 홀수의 스트림(최하위 비트가 1으로 설정된 스트림)을 초기화한다.

스트림 ID의 두 번째 하위 비트(0x2)는 단방향 스트림과 양방향 스트림을 구분한다. 단방향 스트림을 항상 이 비트를 1로 설정하고 양방향 스트림은 이 비트를 0으로 설정한다.

스트림 동시성

QUIC에서는 임의의 스트림이 다수 동시에 동작할 수 있다. 엔드포인드는 최대 스트림 ID를 제한해서 동시에 받아들이는 활성 스트림의 수를 제한한다.

엔드포인트마다 최대 스트림 ID를 설정하고 설정을 받는 피어에만 적용된다.

데이터 보내기와 받기

엔드포인트는 스트림을 사용해서 데이터를 보내고 받고 이는 스트림의 원래 목적이다. 스트림은 순서가 맞는 바이트 스트림 추상화다. 하지만 별도의 스트림은 원래 순서대로 전달되지 않는다.

스트림의 우선순위

스트림에 할당된 리소스의 우선순위가 제대로 설정되면 스트림 멀티플렉싱은 애플리케이션 성능에 상당한 영향을 준다. HTTP/2 같은 멀티플렉싱을 지원하는 다른 프로토콜의 경험에서 상당히 긍적적인 성능 효과를 보여주는 효율적인 우선순위 전략을 볼 수 있다.

QUIC 자체에서 우선순위를 결정하는 정보를 교환하는 프레임을 제공하지는 않는다. 대신 QUIC을 사용하는 애플리케이션에서 우선순위 정보를 받는데 의존한다. QUIC을 사용하는 프로토콜은 애플리케이션의 의미에 적합한 우선순위 스키마를 정의할 수 있다.

QUIC을 통해 HTTP/3을 사용할 때 우선순위는 HTTP 계층에서 동작한다.

0-RTT

새로운 연결을 설정하는데 필요한 시간을 줄이려고 이전에 서버에 연결했던 클라이언트는 해당 연결의 특정 파라미터를 캐시한 뒤 이어서 서버와 **0-RTT** 연결을 설정할 수 있다. 이로 인해서 클라이언트는 핸드쉐이크가 완료되기를 기다리지 않고 바로 데이터를 보낼 수 있다.

스핀 비트

QUIC 워킹그룹에서 수백 개의 이메일을 주고받고 수십 시간의 토론을 걸친 가장 긴 설계 토론 중 하나가 단일 비트 즉, 스핀 비트(Spin Bit)에 관한 것이다.

이 스핀 비트를 지지하는 사람들은 두 QUIC 엔드포인트 사이에 있는 운영자나 사람들이 지연 시간을 측정할 필요가 있다고 주장한다.

이 기능을 반대하는 사람들은 잠재적인 정보 유출을 좋아하지 않았다.

비트 회전시키기

클라이언트와 서버 두 엔드포인트는 각 QUIC 연결에 대해 0 또는 1의 스핀 값을 유지하면서 해당 연결에 적절한 값으로 스핀 비트를 설정해서 패킷을 보낸다.

양 측은 한 라운드 트립동안 같은 값으로 설정된 스핀 비트를 가진 패킷을 보내고 이후 이 값을 토글한다. 이로 인해 비트 필드에 0과 1의 파동이 나타나고 관찰자가 이를 모니터링할 수 있다.

발송자가 애플리케이션도 아니고 제약된 흐름 제어도 아닌 경우에만 이 측정을 할 수 있고 네트워크에서 패킷의 순서가 재정리될 때도 데이터에 잡음이 낄 수 있다.

사용자 영역

사용자 영역에서 전송 프로토콜을 구현하면 클라이언트와 서버가 새로운 버전을 배포하기 위해 운영체제 커널을 업데이트할 필요가 없어서 비교적 쉽게 프로토콜을 발전시킬 수 있으므로 프로토콜을 빠르게 반복할 수 있다.

미래에 운영체제 커널에서 구현되거나 제공하지 않도록 막는 것은 QUIC에 아무것도 없으므로 누군가 좋은 방법을 찾아야 한다.

많은 구현체

사용자 영역에서 새로운 전송 프로토콜을 구현하는 한 가지 명백한 효과는 다수의 독립적인 구현체를 볼 수 있다는 것이다.

예견할 수 있는 미래에 다른 애플리케이션은 다른 HTTP/3와 QUIC 구현체를 포함할(또는 계층 위에서) 것이다.

API

TCP와 TCP를 사용하는 프로그램의 성공 요소 중 하나는 표준화된 소켓 API이다. 소켓 API는 기능이 잘 정의되어 있고 이 API를 사용하면 TCP가 똑같이 동작하므로 다수의 다른 운영체제 사이에서 프로그램을 이동시킬 수 있다.
QUIC은 그렇지 않다. QUIC에는 표준 API가 없다.

QUIC에서는 기존 라이브러리 구현체 중 하나를 선택하고 그 API를 따라야 한다. 이는 애플리케이션을 어떤 부분에서 하나의 라이브러리에 "락인(locked in)"시킨다. 다른 라이브러리로 바꾼다는 것은 다른 API를 뜻하므로 많은 작업이 필요할 것이다.

또한 QUIC이 보통 사용자 영역에서 구현되므로 소켓 API를 쉽게 확장할 수 없고 기존의 TCP나 UDP 기능과 비슷하게 보일 수 없다. QUIC을 사용한다는 것은 소켓 API와는 다른 API를 사용한다는 것을 의미한다.

HTTP/3

이전에 얘기했듯이 QUIC을 통해 전송하는 첫 주요 프로토콜은 HTTP다.

완전히 새로운 방법으로 유선을 통해 HTTP를 전송하려고 HTTP/2를 도입한 것처럼 HTTP/3는 다시 한번 네트워크를 통해 HTTP를 전송하는 새로운 방법을 도입한다.

HTTP는 여전히 이전과 같은 개념과 패러다임을 유지한다. 헤더와 보디가 있고 요청과 응답이 있고 동사, 쿠키, 캐시가 있다. 통신 상대측으로 비트를 보내는 방법이 HTTP/3에서 주된 변경점이다.

QUIC을 통해 HTTP를 보내기 위해 변경이 필요했고 그 결과물을 HTTP/3라고 부른다. QUIC이 제공하는 특성이 TCP의 특성과는 다르므로 변경이 필요했다. 변경사항은 다음과 같다.

- QUIC에서 전송 자체에서 스트림을 제공하지만 HTTP/2에서는 HTTP 계층에서 스트림이 제공된다.
- 스트림이 서로 독립적이므로 HTTP/2에서 사용된 헤더 압축 프로토콜을 head of line 블로킹 문제를 발생시키지 않으면서 사용할 수 없다.
- QUIC 스트림은 HTTP/2 스트림과는 다소 다르다. HTTP/3 부분에서 자세히 설명할 것이다.

HTTPS:// URL

HTTP/3는 `HTTPS://` URL을 사용해서 실행될 것이다. 세상은 이러한 URL로 가득 차 있고 새로운 프로토콜에 또 다른 URL 스킴을 도입하는 것은 실용적이지도 않고 완전히 불합리하다고 여겨졌다. HTTP/2에 새로운 스킴이 필요 없었듯이 HTTP/3에도 필요 없다.

하지만 HTTP/2가 유선으로 HTTP를 전송하는 완전히 새로운 방법이지만 HTTP/1이 그랬듯이 여전히 TLS와 TCP에 기반을 두고 있었기에 HTTP/3 상황에서 복잡성을 추가하게 되었다. HTTP/3가 QUIC을 통해 수행된다는 점은 몇 가지 중요한 관점에서 변경이 발생했다.

레거시, 평문, `HTTP://` URL은 지금 그대로 유지될 것이지만 더 안전한 전송을 하는 미래로 나아갈수록 점점 덜 사용될 것이다. 이러한 URL에 대한 요청은 HTTP/3를 사용하도록 업그레이드되지 않을 것이다. 현실에서 이러한 것들이 HTTP/2로 업그레이드하는 경우는 거의 없지만 다른 이유 때문이다.

초기 연결

이전에 방문한 적이 없는 HTTPS:// URL에 대한 새로운 첫 연결은 아마도 TCP를 통해 수행될 것이다. (추가로 QUIC을 통한 병렬연결 시도가 있을 수 있다.) 호스트가 QUIC을 지원하지 않는 레거시 서버일 수도 있고 중간에 있는 미들박스가 QUIC 연결이 성공적으로 이뤄지지 않도록 하는 장애물일 수도 있다.

최신 클라이언트와 서버는 아마도 첫 핸드쉐이크에서 HTTP/2를 협상할 것이다. 연결이 설정되고 클라이언트의 HTTP 요청에 서버가 응답할 때 서버는 HTTP/3의 지원과 선호도에 관해 클라이언트에게 알려줄 수 있다.

Alt-svc로 부트스트랩하기

대체 서비스 헤더(Alt-svc:)와 이에 대응되는 `ALT-SVC` HTTP/2 프레임이 QUIC이나 HTTP/3를 위해 특별히 만들어진 것은 아니다. 서버가 클라이언트에게 *"봐라. 이 포트에서 이 프로토콜로 이 호스트에서 같은 서비스를 운영하고 있다."*라고 말할 수 있도록 이미 설계되고 만들어진 메커니즘의 일부이다. 자세한 내용은 RFC 7838를 참고해라.

이러한 Alt-svc 응답을 받은 클라이언트는 (이를 지원하고 원하는 경우) 주어진 다른 호스트에 지정된 프로토콜로 백그라운드에서 병렬 연결을 하도록 권고받는다. 성공적으로 전환된다면 초기 연결 대신 새로운 연결을 통해서 해당 작업을 수행한다.

초기 연결이 HTTP/2나 HTTP/1을 사용하더라도 서버는 다시 연결해서 HTTP/3를 시도할 수 있다고 클라이언트에게 알려줄 수 있다. 이는 같은 호스트일 수도 있고 요청한 내용을 제공하는 방법을 아는 다른 호스트일 수도 있다. 이러한 Alt-svc 응답에서 제공된 정보에는 만료 타이머가 있어서, 클라이언트가 일정 시간 동안 제안받은 대체 프로토콜로 직접 대체 호스트에 이어진 연결과 요청을 할 수 있다.

예시

HTTP 서버는 응답에 `Alt-Svc:` 헤더를 다음과 같이 포함한다.

```
1 Alt-Svc: h3=":50781"
```

이는 해당 응답을 얻는데 사용한 것과 같은 호스트 이름의 50781 UDP 포트에서 HTTP/3를 사용할 수 있음을 나타낸다.

그 다음 클라이언트는 해당 목적지에 QUIC 연결을 설정하려고 시도하고 연결이 성공하면 초기 HTTP 버전 대신 이러한 출처와 계속해서 통신한다.

QUIC 스트림과 HTTP/3

HTTP/2가 TCP를 기반으로 전체적인 스트림과 멀티플렉싱 개념을 설계해야했던 반면 HTTP/3는 QUIC을 위해 만들어졌으므로 QUIC 스트림이 가진 이점을 최대한 활용한다.

HTTP/3를 통해 수행되는 HTTP 요청은 특정 스트림 세트를 사용한다.

HTTP/3 프레임

HTTP/3는 QUIC 스트림을 설정하고 반대쪽 끝으로 프레임 세트를 보내는 것을 의미한다. HTTP/3에는 몇 가지 알려진 프레임이 고정되어 있다. 이 중 가장 중요한 것은 다음과 같다.

- HEADERS, 압축된 HTTP 헤더를 보내라.
- DATA, 바이너리 데이터 콘텐츠를 보내라.
- GOAWAY, 이 연결을 종료해라.

HTTP 요청

클라이언트는 클라이언트가 초기화한 *양방향* QUIC 스트림으로 HTTP 요청을 보낸다.

단일 HEADERS 프레임으로 구성된 요청은 하나 또는 두 개의 다른 프레임, 즉 일련의 DATA 프레임과 뒤이어 오는 것들을 위한 최종 HEADERS 프레임이 따라올 수 있다.

요청을 보낸 후 클라이언트는 전송용 스트림을 닫는다.

HTTP 응답

서버는 양방향 스트림으로 해당 HTTP 응답을 돌려보낸다. 여기에는 HEADERS 프레임과 일련의 DATA 프레임이 있고 뒤따라오는 HEADERS 프레임이 있을 수 있다.

QPACK 헤더

HEADERS 프레임은 QPACK 알고리즘으로 압축된 HTTP 헤더를 담고 있다. QPACK은 HPACK이라고 부르는 HTTP/2의 압축과 형식 면에서 비슷하지만, 순서가 맞지 않게 전송된 스트림에서 동작하도록 수정되었다.

QPACK 자체는 두 엔드포인트 사이에서 추가적인 두 개의 단방향 QUIC 스트림을 사용한다. 이 스트림은 양방향으로 동적 테이블 정보를 전달하는 데 사용된다.

우선순위 정하기

HTTP/3 스트림 프레임 중에는 `PRIORITY` 라는 프레임이 있다. 이 프레임은 HTTP/2에서 동작한 방식과 비슷하게 스트림의 우선순위와 의존성을 설정하는 데 사용한다.

이 프레임은 특정 스트림이 다른 스트림에 의존하도록 설정할 수 있고 해당 스트림에 "가중치"를 설정할 수 있

다.
종속된 스트림은 의존하는 스트림이 모두 닫혔을 때만 리소스가 할당받아야 하고 아니면 진행될 수 없다.

스트림 가중치는 1부터 256 사이의 값을 가지며 같은 부모를 가진 스트림은 **반드시** 가중치에 따라 리소스를 할당받아야 한다.

서버 푸시

HTTP/3 서버 푸시는 HTTP/2에서 설명된 내용(RFC 7540)과 비슷하지만 다른 메커니즘을 사용한다.

서버 푸시는 클라이언트가 보낸 적이 없는 요청에 대한 효율적인 응답이다!

서버 푸시는 클라이언트 쪽에서 서버 푸시에 동의했을 때만 발생할 수 있다. HTTP/3에서는 클라이언트가 최대 푸시 스트림의 ID가 무엇인지 서버에게 알려주어 클라이언트가 얼마나 많은 푸시를 받을지를 제한한다. 이 제한을 초과하면 연결 오류가 발생할 것이다.

클라이언트가 요청하지는 않았지만 어쨌든 서버가 받아야 하는 추가 리소스가 있다고 판단하면, (요청 스트림을 통해) 서버가 요청에 푸시로 응답한 것처럼 보이는 `PUSH_PROMISE` 프레임을 보낼 수 있다. 그다음 실제 응답은 새로운 스트림을 통해 보낸다.

클라이언트가 미리 푸시를 받을 수 있다고 말했더라도, 클라이언트가 적합하다고 판단하면 푸시된 개별 스트림을 언제든 취소할 수 있다. 그리고 서버에 `CANCEL_PUSH` 프레임을 보낸다.

문제점

이 기능은 HTTP/2 개발에서 처음 논의된 이후 HTTP/2 프로토콜이 나오고 인터넷에 배포된 뒤, 이 기능을 유용하게 만들기 위해 셀 수 없이 다양한 방법으로 논의되고, 싫어하게 했으며, 두들겨 맞았다.

라운드 트립의 절반을 줄일 수 있지만, 여전히 대역폭을 사용하기 때문에 푸시는 결코 "공짜"가 아니다. 실제로 리소스가 푸시되어야 하는지 아닌지를 서버 측에서 확실하게 알기가 어렵거나 불가능하다.

HTTP/3과 HTTP/2의 비교

HTTP/3는 자체적으로 스트림을 다루는 전송 프로토콜인 QUIC을 위해 설계되었다.

HTTP/2는 TCP를 위해 설계되었으므로 HTTP 계층에서 스트림을 다룬다.

유사점

이 두 프로토콜을 사실상 같은 기능을 제공한다.

- 두 프로토콜은 스트림을 제공한다.

- 두 프로토콜은 서버 푸시를 지원한다
- 두 프로토콜은 헤더 압축을 제공한다. QPACK과 HPACK은 설계상 비슷하다.
- 두 프로토콜은 스트림을 이용해서 하나의 연결을 통해 멀티플랙싱을 제공한다.
- 두 프로토콜은 스트림에 우선순위를 정한다.

차이점

세부 내용에 차이점이 있는데 주로 HTTP/3의 QUIC 사용 때문에 생긴다.

- QUIC의 0-RTT 핸드쉐이크 덕에 HTTP/3에서는 이른 데이터 지원이 더 낫게 잘 동작한다. TCP Fast Open과 TLS는 더 적은 데이터를 보내지만, 종종 문제점에 직면한다.
- HTTP/3는 QUIC 덕에 TCP + TLS보다 훨씬 더 빠른 핸드쉐이크를 제공한다.
- HTTP/3에는 안전하지 않거나 암호화되지 않은 버전이 없다. 인터넷에서 드물기는 하지만 HTTP/2는 HTTPS 없이 구현하고 사용할 수 있다.
- HTTP/2가 ALPN 확장을 이용하여 즉시 TLS 핸드쉐이크 협상을 완료할 수 있는 반면 HTTP/3는 QUIC을 사용하므로 클라이언트에 이 사실을 알리기 위해 `Alt-Svc:` 헤더 응답이 먼저 있어야 한다.

일반적인 비판

UDP는 절대 동작하지 않을 것이다

(DNS에서 사용하는) 53 포트가 아닌 UDP 트래픽이 최근에는 주로 공격에 사용되기 때문에 많은 기업, 운영자, 조직에서 차단하거나 속도를 제한하고 있다. 특히 기존의 UDP 프로토콜과 잘 알려진 서버 구현체 중 일부는 증폭 공격에 대한 취약점이 있으므로 공격자가 무고한 대상 피해자에게 대량의 트래픽을 보낼 수 있다.

QUIC에서는 초기 패킷이 최소 1200바이트여야 한다는 조건과 서버가 클라이언트로부터 응답 패킷을 받지 않으면 요청 크기의 3배 이상은 절대 보내면 안 된다는 프로토콜의 제약사항으로 증폭 공격을 완화하는 기능이 들어있다.

커널에서 UDP는 느리다

이는 적어도 2018년 현재까지 사실로 보인다. 물론 앞으로 어떻게 발전할 것인지, 수년간 UDP의 전송 성능이 개발자의 관심사가 아니었다는 결과가 간단히 어느 정도인지 말할 수 없다.

대부분 클라이언트는 이 "느림"을 결코 알아채지 못했다.

QUIC은 CPU를 너무 많이 사용한다

위에서 얘기한 "UDP는 느리다"와 비슷하게 이 또한 부분적으로는 TCP와 TLS가 세계적으로 더 오랫동안 성숙하고 개선되고 하드웨어의 지원을 받았기 때문이다.

시간이 지나면 개선되리라 기대할 근거는 있다. 문제는 추가적인 CPU 사용이 배포자에게 얼마나 영향을 끼치는가이다.

그냥 구글이다

전혀 그렇지 않다. Google이 대규모 인터넷 환경에서 증명을 마친 후 IETF에 초기 명세를 가져왔다. UDP를 통한 이 방식의 프로토콜 배포는 실제로 잘 동작한다.

그 이후 많은 회사와 조직의 사람들이 벤더 중립적 조직인 IETF에서 독립적으로 표준 전송 프로토콜을 작업했다. 물론 이 작업에 Google 직원도 참여했지만, 인터넷 전송 프로토콜의 상태를 향상하려는 Mozilla, Fastly, Cloudflare, Akamai, Microsoft, Facebook, Apple 등 많은 회사의 직원들도 참여했다.

개선된 부분이 너무 적다

이것은 정말 비판이 아니라 의견일 뿐이다. 어쩌면 HTTP/2가 나온 지 얼마 안 되었기 때문에 개선된 부분이 너무 적을 수도 있다.

HTTP/3는 패킷 손실이 많은 네트워크에서 훨씬 더 잘 동작할 것이고, 더 빠른 핸드쉐이크를 제공하므로 체감 지연시간과 실제 지연시간을 모두 개선할 것이다. 하지만, 사람들이 자신의 서버와 서비스에 HTTP/3 지원을 추가하고 싶을 정도로 장점이 충분할까? 시간과 미래의 성능 측정으로 분명히 알 수 있을 것이다!

명세

다음은 QUIC과 HTTP/3의 여러 부분 및 구성요소의 공식 드래프트 최신 버전이다.

불변

QUIC의 버전 독립적인 프로퍼티

전송

QUIC: UDP에 기반을 둔 멀티플랙싱 보안 전송

복구

QUIC 손실 탐지와 혼잡 제어

TLS

Transport Layer Security(TLS)를 사용해서 QUIC 안전하게 하기

HTTP

QUIC을 통한 Hypertext Transfer Protocol(HTTP)

QPACK

QPACK: QUIC을 통한 HTTP의 헤더 압축

QUIC v2

핵심 QUIC 프로토콜에 가장 집중해서 제때 출시할 수 있도록 핵심 프로토콜의 일부로 원래 계획되어 있던 몇 가지 기능을 연기했고 QUIC 2나 그 뒤 후속 QUIC 버전에 포함할 예정이다.

하지만 이 문서의 작성자가 잘못된 예측을 할 수도 있으므로 버전 2에 어떤 기능이 들어가고 어떤 기능이 안 들어갈지 정확하게 얘기할 수는 없다. 그래도 버전 1에서 명시적으로 제거되어 "나중에 작업"하기로 해서 버전 2에 포함될 가능성이 있는 기능은 얘기할 수 있다.

순방향 오류 정정(Forward Error Correction)

순방향 오류 정정(FEC)은 데이터 전송에서 오류 제어를 하는 방법으로 송신기는 중복 데이터를 전송하고 받는 쪽에서는 식별할 수 있는 오류가 없는 데이터의 부분만을 인식한다.

Google이 원래의 QUIC 작업에서 이를 실험했지만, 실험 결과가 좋지 않아서 후에 다시 제거되었다. 이 기능은 QUIC v2의 토론 주제이지만 너무 큰 불이익이 없고 유용한 기능이라는 것을 실제로 누군가 증명해야 할 것이다.

다중 경로

다중 경로는 리소스 사용을 최대화하고 중복을 늘리기 위해 전송 자체가 다중 네트워크 경로를 사용하는 것을 의미한다.

SCTP 지지자는 SCTP에는 이미 다중 경로 기능이 있고 오늘날의 TCP도 그렇다고 말할 것이다.

신뢰할 수 없는 데이터

"신뢰할 수 없는" 스트림을 선택사항으로 제공해서 UDP 방식의 애플리케이션도 QUIC이 대체할 수 있게 하는 것이 논의되었다.

HTTP가 아닌 프로토콜 도입

QUIC을 통한 DNS는 QUIC v1과 HTTP/3가 출시되면 관심을 끌 수 있는 HTTP가 아닌 프로토콜로 초기에 언급된 것 중 하나이다. 하지만 이 새로운 전송을 세상에 내놓으면 거기서 끝이라고 상상할 수 없다.

Română

Lucrul la această carte a început în martie 2018. Planul este documentarea HTTP/3 și protocolul pe care se bazează: QUIC. De ce, cum funcționează, detaliile protocolului, implementările și multe altele.

Cartea, complet gratuită, este menită a fi un efort cooperativ care să implice pe oricine și pe toată lumea care dorește să ajute.

Precondiții

Se presupune că o persoană care citește această carte are o înțelegere de bază a conexiunilor TCP/IP, conceptelor din spatele HTTP și internetului. Pentru mai detalii și caracteristici ale HTTP/2, recomandăm mai întâi citirea http2 explained.

Autorul

Cartea este începută de Daniel Stenberg. Daniel este creatorul și dezvoltatorul principal al curl, cel mai folosit client HTTP din lume. Daniel lucrează cu (și la) HTTP și protocoale de internet de mai bine două decenii și este autorul http2 explained.

Pagina principală

Pagina principală a acestei cărți se găsește la daniel.haxx.se/http3-explained.

Cum puteți ajuta

Dacă găsiți greșeli, scăpări, erori sau minciuni evidente în acest document, vă rugăm să ne trimiteți o versiune mai bună a paragrafului cu pricina și noi vom face modificările necesare. Vom menționa cum se cuvine pe toată lumea care ajută. Sper să pot îmbunătăți cu timpul acest document.

Este de preferat să trimiteți problemele sau pull requests pe pagina de GitHub a cărții.

Licență

De ce QUIC

QUIC este un nume, nu un acronim. Este pronunțat la fel ca "quick" în limba engleză.

QUIC este din mai multe puncte de vedere ceea poate fi considerat o modalitate de a construi un nou protocol de transport, de încredere și sigur, pentru protocoale precum HTTP. QUIC rezolvă câteva dintre neajunsurile conexiunilor HTTP/2 peste TCP și TLS. Este următorul pas logic în evoluția comunicării web.

QUIC nu este limitat doar la transportul HTTP. Cel mai mare motiv și motivație care a dus la crearea acestui protocol este dorința de a face ca informațiile și datele, în general, să poată fi livrate mai repede utilizatorilor finali.

Deci, de ce să creezi un protocol de transport nou și de ce este construit peste UDP?

QUIC logo

Vă mai aduceți aminte de HTTP/2?

Specificația HTTP/2, RFC 7540, a fost publicată în mai 2015, iar, de atunci, protocolul a fost implementat și publicat la o scară largă pe internet și pe World Wide Web.

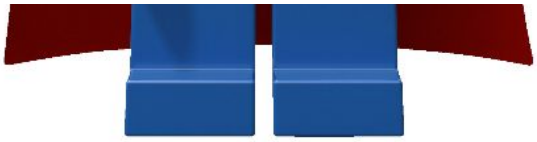
La începutul lui 2018, aproape 40% din primele 1000 de pagini de internet din lume rulau pe HTTP/2, aproximativ 70% din toate cererile HTTPS făcute de Firefox primeau răspunsuri HTTP/2, iar toate browsere-le, servere și proxy-uri ofereau suport pentru el.

HTTP/2 rezolvă o multitudine de neajunsuri ale HTTP/1 și, o dată cu introducerea celei de-a doua versiuni a HTTP, utilizatorii se pot opri din a folosi unele metode neoficiale menite să rezolve aceste neajunsuri. Câteva dintre aceste metode sunt destul de dificile pentru programatorii web.

Una dintre funcționalitățile principale ale HTTP/2 este că se folosește de multiplexing, astfel încât mai multe fluxuri logice sunt trimise peste aceeași conexiune TCP. Acest lucru face ca totul să se întâmple mai bine și mai repede. Efortul pentru controlul congestionărilor este mai bun, iar utilizatorii au posibilitatea să folosească conexiunile TCP mult mai bine și, deci, să satureze corespunzător lărgimea de bandă. Astfel, conexiunile TCP sunt active mai multe, ceea ce este bine pentru că ajung la viteza maximă mult mai frecvent ca înainte. Compresia headere-lor ajută conexiunile să folosească mai puțină lărgime de bandă.

Prin HTTP/2, browsere-le folosesc, de obicei, *o conexiune* TCP pentru fiecare host, în loc de cele *șase conexiuni* folosite în trecut. De fapt, tehnicile de contopire și "desharding" ale conexiunilor, folosite cu HTTP/2, pot reduce numărul de conexiuni și mai mult de atât.

HTTP/2 a reparat și problema pe care protocolul o avea cu blocarea vârfului stivei (head of line blocking), în urmă căreia clienții trebuia să aștepte ca prima cerere din stivă să fie finalizată înainte ca următoarea cerere să poată fi trimisă.

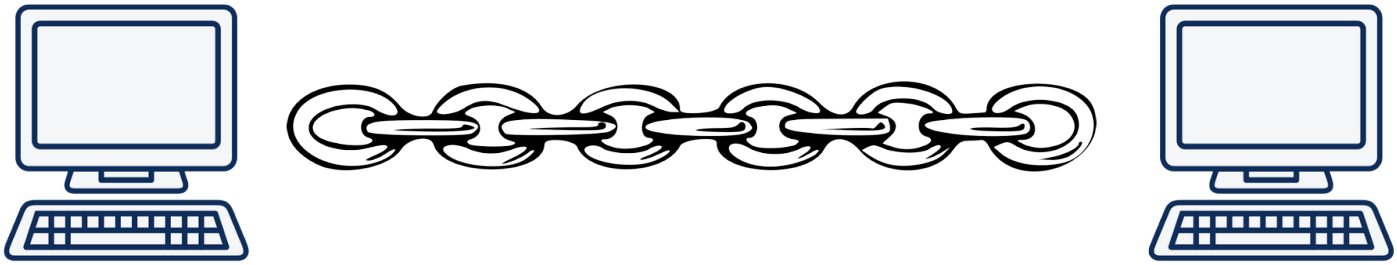


http2 man

Blocarea vârfului stivei în TCP

Blocarea vârfului stivei în TCP

HTTP/2 este implementat peste TCP și folosește mult mai puține conexiuni decât versiunile sale anterioare. TCP este un protocol pentru transferuri fiabile - vă puteți gândi la el ca la un lanț imaginar între două sisteme. Ce este trimis prin rețea de la un capăt va ajunge, eventual, la celălalt capăt. Altfel, conexiunea va fi întreruptă.

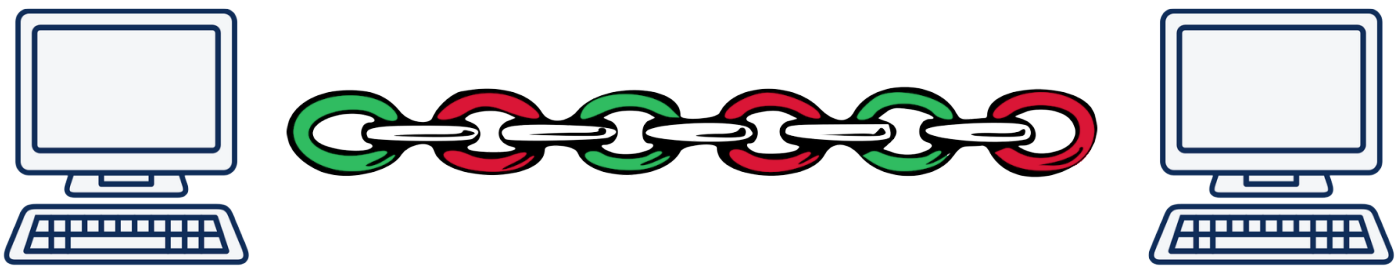


un lanț TCP între două computere

Cu HTTP/2, browsere-le normale fac zeci sau sute de transferuri în paralel peste o singură conexiune TCP.

Dacă un singur pachet este ignorat, sau pierdut în rețea undeva între două componente care "vorbesc" HTTP/2, înseamnă că toată conexiunea TCP este oprită până când pachetul pierdut este retransmis și își regăsește calea către destinatar. De vreme ce TCP este un "lanț", dacă o za a acestui lanț dispare subit, tot ce vine după ea trebuie să aștepte.

O ilustrație care se folosește de metafora "lanțului" pentru a transmite două fluxuri, unul roșu și unul verde, printr-o singură conexiune:



lanțul conținând zale în mai multe culori

Blocarea vârfului de stivă TCP:

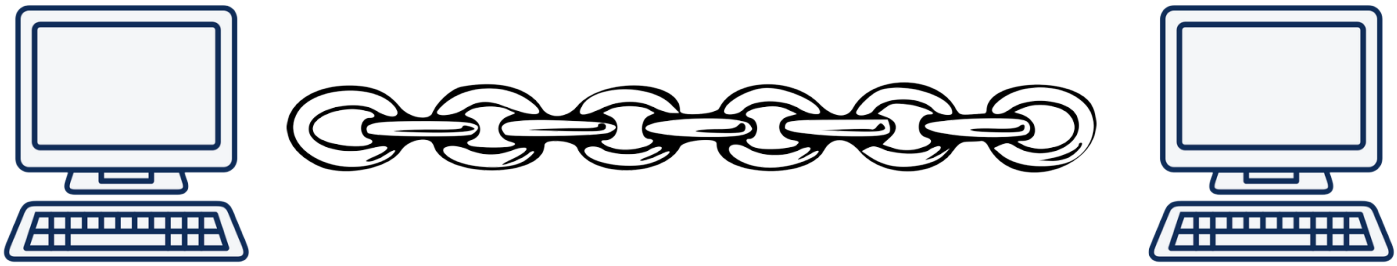
Pe măsură ce rata de pierdere a pachetelor crește, HTTP/2 se mișcă din ce în ce mai rău. La o pieredere de 2% (care reprezintă o calitate groaznică a rețelei), testele au arătat că utilizatorii HTTP/1 sunt mai avantajați,

deoarece au până la 6 conexiune TCP deschise prin care se distribuie rata de pieredere a pachetelor. Asta înseamnă că, dacă o conexiune pierde un pachet, celelalte pot continua.

O rezolvare a acestei probleme nu este ușoară sau, poate chiar posibilă, atunci când se folosește TCP.

Fluxurile independente evită blocarea

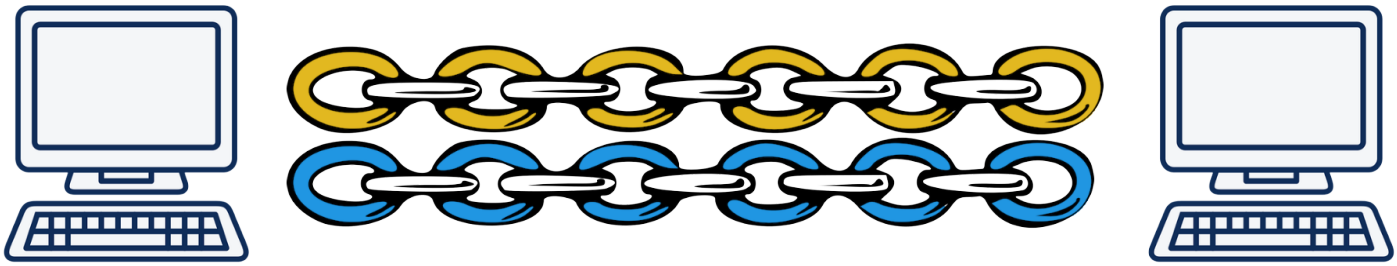
Atunci când este folosit QUIC, se face, de asemenea, o organizare a unei conexiuni între două sisteme, ceea ce face conexiunea să fie sigură și datele să fie livrate fiabil.



un lanț QUIC între două computere

Două fluxuri separate trimise prin această conexiune sunt prelucrate independent, astfel că, dacă o parte lipsește dintr-unul dintre fluxuri, numai acel flux și acel lanț trebuie să se oprească și să aștepte ca acea bucată să fie retrimisă.

Conceptul este ilustrat aici cu un flux galben și unul albastru, trimise între două sisteme:



două fluxuri QUIC între două sistee

TCP sau UDP

TCP sau UDP

Dacă nu putem să rezolvăm problema blocării vârfului de stivă din TCP, atunci, teoretic, ar trebui să putem să creăm un nou protocol de transport, diferit de UDP și TCP. Sau, poate chiar să folosim SCTP, care este un protocol de transport standardizat de IETF în [RFC4960] (https://tools.ietf.org/html/rfc4960) și care include câteva dintre caracteristicile dorite.

Totuși, în ultimii ani, eforturile de a crea noi protocoale de transport au fost oprite aproape în totalitate de dificultățile în a le publica pe internet. Publicarea de noi protocoale este limitată de multe firewall-uri, NAT-uri, routere și alte middlebox-uri care permit doar conexiuni TCP și UDP între utilizatori și servere-le cu care au nevoie să comunicae. Introducerea unui nou protocol de transport face ca N% dintre conexiuni să

eșueze pentru că sunt blocate de sisteme care văd că nu sunt UDP sau TCP, deci le consideră dăunătoare sau, pur și simplu, greșite. Rata de eșec de N% este considerată, deseori, prea mare pentru a merita efortul.

În plus, schimbările în layer-ul protocolului de transport al stivei de rețea, înseamnă, de obicei, protocoale implementate de kernel-urile sistemelor. Actualizarea și publicarea kernel-urilor sistemelor de operare este un proces încet, care presupune efort semnificativ. Multe îmbunătățiri ale TCP, standardizate de IETF, nu sunt publicate la scară largă sau folosite pentru că nu sunt implementate, în general.

De ce nu SCTP-peste-UDP?

SCTP este un protocol de transport fiabil cu fluxuri șim pentru WebRTC, există deja implementări care îl folosesc peste UDP.

SCTP nu a fost considerat îndeajuns de bun față de QUIC din cauza câtorva motive, printre care:

- SCTP nu rezolvă problema blocării vârfului de stivă pentru fluxuri
- SCTP are nevoie ca numărul de fluxuri să fie predefinit la momentul stabilirii conexiunii
- SCTP nu are o istorie puternică cu TLS/securitate
- SCTP funcționează cu un handshake în 4 pași, în timp ce QUIC oferă 0-RTT
- QUIC este un flux de octeți ca TCP, în timp ce SCTP se folosește de mesaje
- Conexiunile QUIC pot migra între adrese IP, în timp ce ale SCTP nu

Pentru mai multe detalii legate de diferențe, vedeți A Comparison between SCTP and QUIC.

Osificarea

Internetul este o rețea de rețele. Există dispozitive instalate pe internet în multe locuri diferite care se asigură că această rețea de rețele funcționează cum ar trebui. Aceste dispozitive, sisteme care sunt distribuite pe rețea, sunt ceea ce numim middlebox-uri - echipamente care sunt poziționate undeva între două sisteme și care sunt părțile principale implicate într-un transfer tradițional pe rețea.

Aceste echipamente servesc mai multe scopuri diferite, dar cred că putem spune cu siguranță că sunt puse acolo pentru că cineva crede că trebuie să fie acolo astfel încât lucrurile să funcționeze cum trebuie.

Middlebox-uri redirecționează pachete IP între rețele, blochează traficul răuvoitor, fac NAT (Network Address Translation - traducerea adreselor de rețea), îmbunătățesc performanța, unele încearcă să spioneze traficul care trece prin ele și așa mai departe.

Ca să își poată îndeplini sarcinile, aceste sisteme trebuie să aibă cunoștințe de rețelistică și despre protocoalele pe care trebuie să le monitorizeze sau să le modifice. Pentru acest scop, rulează aplicații software, iar aceste aplicații nu sunt mereu actualizate frecvent.

Chiar dacă sunt componente care fac internetul să funcționeze, de multe ori nu țin pasul cu ultimele tehnologii. Mijlocul rețelei, de obicei, nu se mișcă la fel de repede ca marginile - aplicațiile-client și server-le

din lume.

Protocoalele de rețea pe care aceste echipamente ar vrea să le inspecteze și despre care ar vrea să stie dacă sunt OK sau nu au această problemă: aceste echipamente au fost publicate cu ceva timp în urmă, când protocoalele avea un set de caracteristici, valabil la acel moment. Introducerea unor noi funcționalități sau schimbări în comportament care nu erau cunoscute atunci sunt considerate rele sau interzise de aceste echipamente. Astfel de trafic poate fi oprit sau întârziat până la punctul în care utilizatorii nu își mai doresc să folosească funcționalitățile.

Asta se numește "osificarea protocoalelor".

Schimbările TCP "suferă", de asemenea, de osificare: unele echipamente între o aplicație-client și server o să identifice opțiuni noi TCP pe care nu o să le recunoscă și o să blocheze întreaga conexiune. Dacă le este permis să detecteze detalii legate de protocol, sistemele învață cum ar trebui să se comporte protocoalele de obicei și, cu timpul, devin imposibil de schimbat.

Singura metodă eficientă de a preveni osificarea este criptarea a unei bucăți cât mai mari din comunicare pe cât posibil, astfel încât middlebox-urile să nu poată vedea detalii ale protocoalelor care trec prin ele.

Securitate

QUIC oferă întotdeauna securitate. Nu există o versiune în clar a protocolului, așa că, pentru a negocia o conexiune QUIC este nevoie de criptografie și securitate bazată pe TLS 1.3. Cum am menționat mai sus, asta previne osificarea, la fel ca alte tipuri de blocaje sau tratamente speciale, asigurându-se, în același timp că QUIC folosește toate proprietățile HTTPS pe care utilizatorii au ajuns să și le dorească și să le aștepte.

Există doar câteva pachete inițiale trimise în clar, la handshake, înainte ca protocoalele criptografice să fie negociate.

Timp de așteptare redus

QUIC oferă handshake-uri 0-RTT și 1-RTT care reduc timpul necesar negocierii și stabilirii unei noi conexiuni. În comparație, TCP folosește un handshake în trei pași.

În plus, QUIC oferă suport pentru "informații din timp", care îi dau posibilitatea să transmită mai multe informații și este folosit mult mai ușor decât TCP Fast Open.

Folosindu-se conceptul de fluxuri, o nouă conexiune logică poate fi deschisă către același host fără a fi nevoie să se aștepte terminarea uneia existente.

TCP Fast Open este problematic

TCP Fast Open a fost publicat ca RFC 7413 în decembrie 2014, iar specificația descrie cum pot programele transmite informații către un server, livrându-le deja în primul pachet TCP SYN.

Implementarea acestei componente în realitate a durat mult timp și este plină de probleme, chiar și acum în 2018. Cei care implementează specificația TCP au avut probleme și, astfel, și programele care încearcă să folosească această funcționalitate au avut la fel - în primul rând în a ști ce sisteme de operare să activeze și apoi cum să anuleze schimbările atunci când au apărut probleme. Au fost identificate câteva rețele care au interferat cu traficul TCP Fast Open, stricând, deci, permanent, astfel de TCP handshakes.

Procesul

Prima versiune a protocolului QUIC a fost proiectată de Jim Rosking de la Google, și a fost implementată prima oară în 2012. A fost anunțată public în 2013 când Google au extins experimentele.

Atunci, QUIC reprezenta un acronim pentru "Quick UDP Internet Connections" (Conexiuni UDP Rapide pe Internet)

Google au implementat și publicat, ulterior, protocolul în browser-ul popular pe care îl dezvoltă (Chrome) și în serviciile server-side proprii, la fel de populare (motorul de căutare Google, gmail, youtube și altele). Au dezvoltat noi versiuni repede și, cu timpul, au demonstrat fiabilitatea conceptului pentru o plajă largă de utilizatori.

În iunie 2015, prima schiță pentru QUIC a fost trimisă către IETF pentru standardizare, dar a durat până la sfârșitul lui 2016 pentru ca un grup de lucru dedicat să fie aprobat și să înceapă. După aceea, lucrul a început aproape imediat, mai mulți participant declarându-și interesul.

În 2017, numele citate de inginerii Google care lucrează la QUIC menționau că aproximativ 7% din *tot* traficul de pe internet folosește deja acest protocol. Versiunea Google a protocolului.

IETF

Grupul de lucru QUIC, care a fost organizat pentru a standardiza protocolul în interiorul IETF, a decis repede că acesta ar trebui să poată să transfere și alte protocoale în afară de "doar" HTTP. Google-QUIC nu a transferat niciodată altceva în afară de HTTP. În practică, a transferat frame-uri HTTP/2, folosindu-se de sintaxa pentru frame-uri a HTTP/2.

S-a spus, de asemenea, că IETF-QUIC ar trebui să își bazeze criptarea și securitatea pe TLS 1.3, în loc de implementarea "proprie" folosită de Google-QUIC.

Ca să îndeplinească cerință de trimite-mai-mult-decât-HTTP, arhitectura protocolului QUIC al IETF a fost separată în două părți: transferul QUIC și "HTTP peste QUIC". A doua parte este numită uneori și "hq".

Această despărțire, chiar dacă sună inofensivă, a făcut ca specificația IETF-QUIC să difere destul de mult de cea originală, Google-QUIC.

Grupul de lucru a decis repede, totuși, că, pentru a putea avea puterea de concentrare și abilitatea de a livra versiunea 1 a QUIC, se va ocupa de livrarea HTTP, lăsând transferurile non-HTTP pe mai târziu.

În martie 2018, când am început să lucrez la această carte, planul era ca specificația finală a versiunii 1 să

fie publicată în noiembrie 2018. Această dată a fost amânată până în iulie 2019.
În timp ce munca la IETF-QUIC a progresat, echipa de la Google a implementat deja detalii din versiunea IETF și a început să migreze versiunea proprie a protocolului către ce ar putea să devină versiunea IETF. Google au continuat să își folosească propria versiune a QUIC în browser și servicii.

[Majoritatea implementărilor în curs de desfășurare] (https://github.com/quicwg/base-drafts/wiki/Implementations) au decis să se concentreze pe versiunea IETF și nu sunt compatibile cu versiunea Google.

Experiența acumulată cu HTTP/2

Specificația HTTP/2, RFC 7540, a fost publicată în mai 2015, cu doar o lună înainte ca QUIC să fie prezentat IETF pentru prima oară.

Cu HTTP/2, fundația pentru schimbarea HTTP a fost deja pusă și grupul de lucru a adoptat o mentalitate potrivită pentru trecerea la versiuni noi de HTTP mult mai repede ca înainte (trecerea de la HTTP/1 la 2 a durat aproximativ 16 ani).

După lansarea HTTP/2, utilizatorii și creatorii de software s-au obșinuit cu ideea că HTTP nu mai poate fi considerat a fi un protocol finalizat.

HTTP-peste-QUIC a fost redenumit HTTP/2 în noiembrie 2018.

Starea curentă

Grupul de lucru QUIC a muncit asiduu începând cu sfârșitul lui 2016 pentru a specifica particularitățile protocolului, iar planul este ca totul să fie gata până în iulie 2019.

La timpul scrierii acestui text, în noiembrie 2018, încă nu există teste de interoperabilitate la scară largă pentru HTTP/3 - există doar două implementări majore, și niciuna dintre ele nu este făcută de un browser sau de un software open-source popular.

Există 15 [implementări QUIC] (https://github.com/curl/curl/wiki/QUIC-implementation) înregistrate în paginile wiki ale grupului de lucru, dar foarte puține dintre ele pot funcționa pe ultimele modificări ale schiței specificației.

Implementarea QUIC nu este ușoară și protocolul a fost actualizat încontinuu, schimbându-se chiar și la această dată.

Servere

Nu a fost făcut niciun anunț public de implementare a QUIC de către Apache sau ngnix.

Programele-client

Niciunul dintre browsere-le mari nu a publicat până acum vreo versiune, în orice stare, care să poată să ruleze versiunea IETF a QUIC sau HTTP/3.

Google Chrome este publicat deja de mulți ani cu o implementare funcțională a propriei implementări QUIC, dar funcționalitățile ei nu sunt compatibile cu protocolul dezvoltat de IETF, iar implementarea lor pentru HTTP/3 este diferită.

Obstacolele din calea implementării

Pentru QUIC, s-a decis folosirea TLS 1.3 ca fundație pentru criptare și layer-ul de securitate (pentru a evita inventarea a ceva nou, comunicarea bazându-se, în schimb, pe un protocul de încredere deja existent. Totuși, făcând asta, grupul de lucru a decis, de asemenea, ca pentru a optimiza folosirea TLS în QUIC, acesta ar trebui să folosească numai "mesaje TLS" și nu "înregistrări TLS".

Chiar dacă sună ca o schimbare inofensivă, această decizie a pus piedici semnificative multora dintre cei care implementează QUIC. Librăriile software care implementează TLS 1.3 pur și simplu nu au destule API-uri pe care să le expună pentru ca QUIC să le acceseze. În timp ce câțiva dintre cei care implementează vin din organizații mai mare care lucrează la propria stivă TLS în paralel, acest lucru nu se aplică la toată lumea.

Bine-cunoscutul OpenSSL, de exemplu, nu implementează niciun API în acest moment și nu a făcut niciun anunț care să exprime planuri în această direcție în viitorul apropiat (afirmație valabilă în noiembrie 2018).

Asta va duce, eventual, și la obstacole în calea implementării din cauză că QUIC va avea nevoie fie să se bazeze pe alte librării TLS, fie să folosească o versiune separată și modificată a OpenSSL, fie să ceară o actualizare a viitoarelor versiune OpenSSL.

Kernel-uri și capacitatea de procesare

Și Google și Facebook au menționat că implementările proprii la scară largă ale QUIC vor necesita aproximativ dublul capacității de procesare (CPU) față de același trafic servit prin HTTP/2 peste TLS.

Câteva explicatii pentru asta sunt:

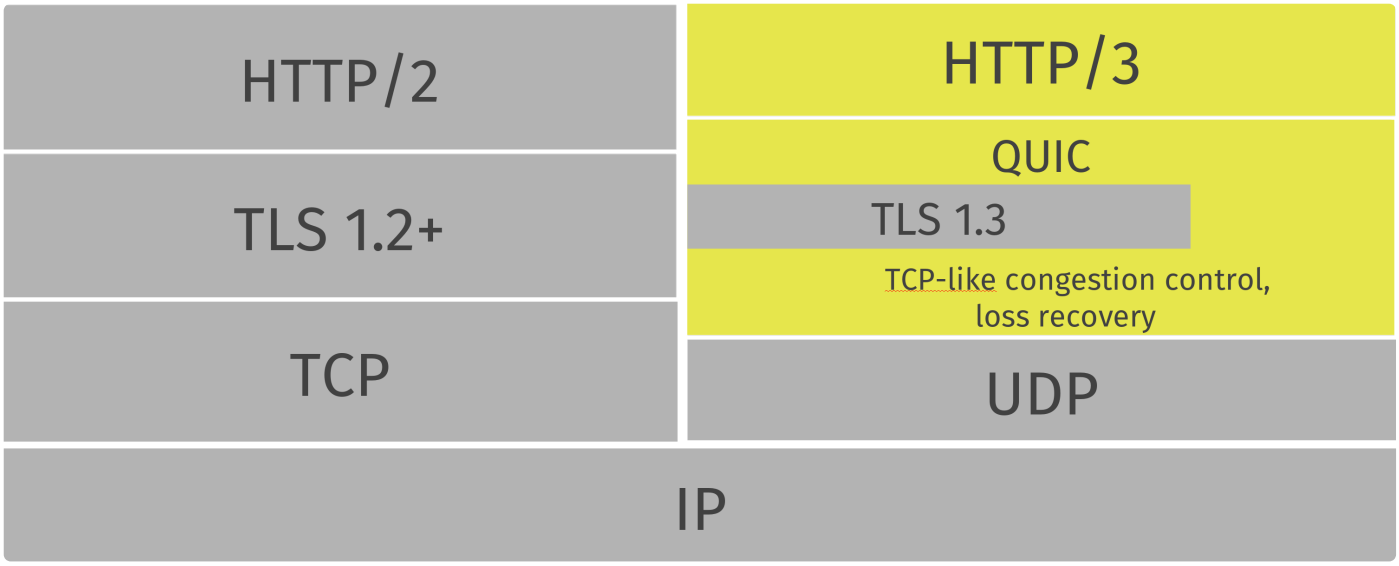
- componenta UDP, mai ales în Linux, nu este deloc optimizată la fel cum este stiva TCP, deoarece nu a fost folosită în mod normal pentru transferuri de mare viteză.
- Pentru TCP și TLS există descărcare către hardware (offloading to hardware), dar pentru UDP exemplele sunt mult mai rare, iar pentru QUIC este practic inexistentă.

Există motive să credem că performanța și cerințele CPU se vor îmbunătăți cu timpul.

Funcționalitățile protocolului

O privire de ansamblu asupra protocolului QUIC.

Mai jos, în stânga, sunt ilustrate componentele unei rețele HTTP/2, iar în dreapta componentele rețelei QUIC când este folosită pentru a face un transfer HTTP.



QUIC logo

UDP

Protocolul de transfer peste UDP

QUIC este un protocol de transfer implementat peste UDP. Dacă vă urmăriți traficul rețelei, o să vedeți că QUIC apare ca pachete UDP.

Tot bazându-se pe UDP, folosește apoi port-uri UDP pentru a identifica servicii de rețea specifice care rulează la o anumită adresă IP.

Toate implementările QUIC cunoscute sunt, acum, în user-space, ceea ce permite o evoluție mult mai rapidă decât permit implementările kernel-space.

Va funcționa?

Există organizații și alte structuri de rețele care blochează traficul UDP pe alte porturi decât 53 (folosit pentru DNS). Altele limitează astfel de date în feluri care fac ca QUIC să funcționeze mai greu ca protocoalele bazate pe TCP. Nu există limite la ce pot face unii operatori.

În viitorul predictibil, toate transferurile bazate pe QUIC va trebui să poată să regreseze (fall-back) la alte alternative (bazate pe TCP). Inginerii Google au menționat în trecut rate de eșec de ordinul numerelor cu o singură cifră.

Se va îmbunătăți?

Șansele sunt ca, dacă QUIC se dovedește să fie o adiție valoroasă pentru internet, oamenii să vrea să îl folosească și să funcționeze în rețelele lor, iar atunci companiile își vor reconsidera obstacolele. În timpul anilor de dezvoltare, rata de succes în a stabili și folosi conexiuni QUIC pe internet a crescut.

Fiabilitate

În timp ce UDP nu este un transfer fiabil, QUIC adaugă un layer deasupra UDP care introduce această fiabilitate. Oferă retransmiterea pachetelor, controlul congestiilor, pacing și toate celelalte funcționalități prezente în TCP.

Datele trimise prin QUIC de la un capăt vor apărea în celălalt capăt mai devreme sau mai târziu, atâta timp cât conexiunea este menținută.

Fluxuri

Similar cu SCTP, SSH și HTTP/2, QUIC oferă fluxuri logice separate în interiorul conexiunilor fizice - in număr de fluxuri paralele care pot transfera informații simultan peste o singură conexiune fără să afecteze celelalte fluxuri.

O conexiune este negociere între două sisteme, similar cu modul de funcționare al TCP. O conexiune QUIC este făcută pe un port UDP și o adresă de IP, dar, odată stabilită, conexiunea este asociată cu propriul "ID de conexiune".

Peste o conexiune stabilită, oricare dintre părți poate crea fluxuri și să trimită informații la celălalt capăt. Fluxurile sunt livrate în ordinea în care sunt trimise și sunt fiabile, dar este posibil ca fluxuri diferite să fie livrate într-o altă ordine.

QUIC oferă control și asupra conexiunilor și a fluxurilor.

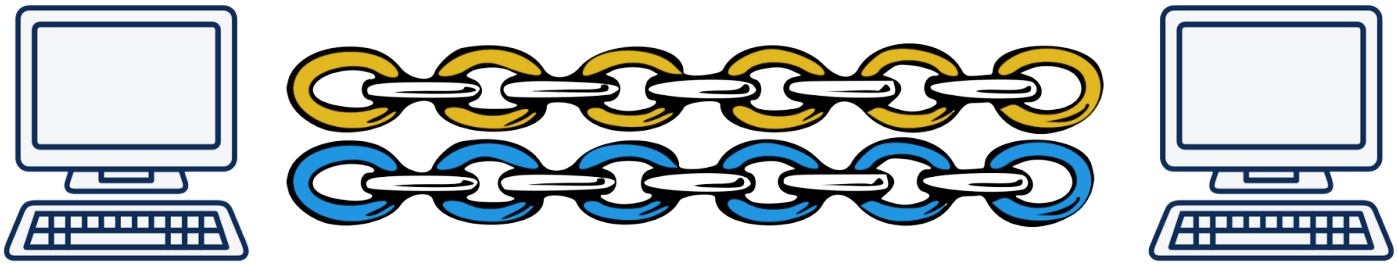
Puteți citi mai multe detalii în secțiunile conexiuni și fluxuri.

În aceeași ordine

QUIC garantează livrarea în aceeași ordine a fluxurilor în sine, dar nu și între fluxuri. Asta înseamnă că fiecare flux va trimite informații și va menține ordinea informațiilor, dar fluxurile pot ajunge la destinație în altă ordine decât cea în care au fost trimise de la sursă.

De exemplu: fluxul A și B suntr transferate de la server către un program. Fluxul A este început primul și apoi fluxul B. În QUIC, un pachet pierdut afectează doar fluxul căruia îi aparține. Dacă fluxul A pierde un pachet, dar fluxul B nu, fluxul B își poate continua și finaliza transferul, în timp ce pachetul pierdut de fluxul A este retransmis. Acest lucru nu era posibil cu HTTP/2.

Ilustrat aici cu două fluxuri, unul galben și unul albastru, trimise între două sisteme QUIC, peste o singură conexiune. Fluxurile sunt independente și pot ajunge la destinație într-o ordine diferită, dar fiecare dintre ele este livrat cu siguranță.



două fluxuri QUIC între două computere

Handshakes rapide

QUIC oferă stabilirea conexiunilor cu 0-RTT și 1-RTT, însemnând că, într-un scenariu ideal, QUIC nu face round-trip-uri în plus atunci când stabilește o conexiune. Cea mai rapidă metodă dintre cele două, handshake-ul 0-RTT, funcționează doar dacă a fost stabilită în trecut o conexiune la același host și o cheie secretă de la acea conexiune a fost salvată în cache.

Informații din timp

QUIC permite unui sistem-client să includă informații deja în handshake-ul 0-RTT. Această funcționalitate îi permite sistemului-client să livreze informații celuilalt sistem pe cât de repede este posibil, iar asta înseamnă, bineînțeles, că server-ul va răspunde și va trimite informații mai repede.

TLS 1.3

Securitatea transferurilor în QUIC este implementată folosing TLS 1.3 (RFC 8446) și nu există conexiuni QUIC necriptate.

TLS 1.3 are câteva avantaje asupra versiunilor mai vechi de TLS, dar un motiv principal de a-l folosi în QUIC a fost că versiunea 1.3 a micșorat numărul de handshakes necesare. Asta reduce timpii de încărcare ai protocolului.

Vechea versiune QUIC de la Google folosea un algoritm propriu de criptare.

Nivelurile de aplicație și de transfer

Protocolul IETF QUIC este un protocol de transfer, peste care pot fi folosite alte protocoale. Layer-ul inițial de aplicație al protocolului este HTTP/3 (h3).

Layer-ul de transfer susține și conexiuni și fluxuri.

Versiunea inițială de QUIC a Google lipea transferul și HTTP împreună într-un singur protocol "bun la toate" și reprezenta un protocol specializat de trimitere-frame-uri-http/3-peste-udp.

HTTP/3 peste QUIC

Layer-ul HTTP face transferuri tip-HTTP, inclusiv compresia headere-lor HTTP folosind QPACK - similar cu compresia HTTP/2 numită HPACK.

Algoritmul HPACK se bazează pe livrarea *ordonată* a fluxurilor, astfel că nu a fost posibil să fie refolosit pentru HTTP/3 fără modificări, de vreme ce QUIC oferă fluxuri care pot fi livrate în orice ordine. QPACK poate fi considerat a fi versiunea adaptă pentru QUIC a HPACK.

Non-HTTP peste QUIC

Munca legată de trimiterea altor protocoale, în afară de HTTP, peste QUIC a fost amânată până după publicarea versiunii 1.

Cum funcționează QUIC

Fără a explica exact biți și baiții de pe rețea, această secțiunea descrie cum funcționează conceptele de bază ale protocolului de transfer QUIC. Dacă doriți să aveți propria implementare a QUIC, această descriere vă va oferi o înțelegere generală, dar, pentru toate detaliile, citiți ciornele și RFC-urile IETF.

1. Stabilirea unei conexiuni
2. ... care include securitatea TLS
3. Și apoi folosiți fluxuri

Conexiuni

O conexiune QUIC este o singură conversație între două sisteme QUIC. Stabilirea conexiunii QUIC combină negocierea versiunii cu handshake-urile criptografice și de transport pentru a reduce timpii de încărcare.

Pentru a trimite efectiv informații prin acest tip de conexiune, unul sau mai multe fluxuri trebuie să fie create

și folosite.

ID-ul conexiunii

Fiecare conexiune deține un set de identificatori, sau ID-uri de conexiune, iar fiecare dintre ei poate fi folosit pentru a identifica o conexiune. ID-urile de conexiune sunt alese independent de capetele conexiunii; fiecare capăt alege ID-urile pe care le folosește celălalt capăt.

Funcționalitatea principală a acestor ID-uri de conexiune este să se asigure că schimbările de adresare ale protocoalelor din layere-le de dedesubt (UDP, IP și mai jos) nu cauzează ca pachetele unei conexiuni QUIC să fie livrate către un alt sistem.

Profitând de ID-uri, conexiunile pot, deci, migra între adrese de IP și interfețe de rețea în moduri în care TCP nu putea. De exemplu, migrația permite ca download-uri în curs de desfășurare să se mute de la o conexiune la o rețea celulară la una mai rapidă peste wifi atunci când utilizatorul își aduce device-ul într-un loc care oferă wifi. Similar, download poate continua peste rețeaua celulară dacă cea wifi devine indisponibilă.

Port-urile

QUIC este construit peste UDP, așa că un număr de port pe 16 biți este folosit pentru a diferenția conexiunile primite.

Negocierea versiunii

O cerere de conexiune QUIC venită de la o aplicație-client o să îi spună server-ului ce versiune de protocol QUIC vrea să folosească, iar server-ul va răspunde cu o listă de versiuni din care aplicația poate alege.

Conexiunile folosesc TLS

Imediat după primul pachet care stabilește o conexiune, inițiatorul trimite un frame criptografic care începe stabilirea handshake-ului pentru layer-ul de securitate.

Nu există nicio cale de a evita acest lucru sau de a evita folosirea TLS pentru conexiune QUIC. Protocolul este proiectat pentru a îngreuna manipulările middlebox-urilor, cu scopul de a evita osificarea protocolului.

Fluxuri

Fluxurile în QUIC oferă o abstractizare simplă a unui flux de octeți ordonat.

Există două tipuri de fluxuri de bază în QUIC:

- Fluxuri unidirecționale care transferă informațiile într-o singură direcție: de la inițiatorul fluxului la receptor.
- Fluxuri bidirecționale care permit ca informațiile să fie transmise în ambele direcții.

Oricare tip de flux poate fi creat de oricare dintre capetele conexiunii, poate trimite informații amestecat cu alte fluxuri și poate fi anulat.

Pentru a trimite date printr-o conexiune QUIC, unul sau mai multe fluxuri sunt folosite.

Controlul debitului

Debitul fluxurilor este controlat individual, permițqnd ca unul dintre capete să limiteze memoria alocată și să aplice presiune înapoi. Crearea de fluxuri are, de asemenea, debitul controlat, cu fiecare dintre sisteme declarând ID-ul maxim pe care este dispus să îl accepte la un moment dat.

Identificatorii fluxurilor

Fluxurile sunt identificate de un integer unsigned pe 62 de octeți, numit și ID-ul fluxului (Stream ID). Cei mai puțini semnificativi doi octeți ai ID-ului sunt folosiți pentru identificarea tipului fluxului (unidirecțional sau bidirecțional) și inițiatorul fluxului.

Cel mai puțin semnificativ bit (0x1) al id-ului identifică inițiatorul fluxului. Aplicațiile-client inițiază fluxuri folosind numere pare (acelea cu cel mai puțin semnificativ bit setat pe 0); servere-le inițiază fluxuri folosind numere impare (cu bitul setat pe 1);

Al doilea cel mai puțin semnificativ bit (0x2) al ID-ul fluxului diferențiază între fluxuri unidirecționale și bidirecționale. Fluxurile unidirecționale au acest bit setat pe 1, iar fluxurile bidirecționale au acest bit setat pe 0.

Paralelismul fluxurilor

QUIC permite ca un număr arbitrar de fluxuri să funcționeze în paralel. Un capăt al conexiunii limitează numărul de fluxuri paralele active prin limitarea ID-ului maxim pe care îl poate avea un flux.

ID-ul maxim al fluxului este specific fiecărui capăt și se aplică numai sistemului care primește această setare.

Trimiterea și primirea de informații

Sistemele folosesc fluxuri ca să trimită și să primească informații. Acesta este, în fond, scopul lor principal. Fluxurile sunt abstracții de octeți ordonați. Fluxurile separate nu sunt, totuși, livrate în ordinea originală.

Prioritizarea fluxurilor

Multiplexing-ul fluxurilor are un efect semnificativ asupra perfomanțelor aplicației dacă resursele alocate fluxului nu sunt corect prioritizate. Experiența cu alte protocoale care fac multiplexing, cum ar fi HTTP/2, arată că strategiile de prioritizare eficiente au un efect pozitiv asupra performanței.

QUIC, în sine, nu oferă cadre pentru prioritizarea informației. În schimb, se bazează pe primirea prioritizării de la aplicația care folosește QUIC. Protocoalele care folosesc QUIC au posibilitatea de a defini orice schemă de prioritizare care se potrivește semanticii proprii.

Când HTTP/3 este folosit peste QUIC, prioritizarea este făcută în layer-ul HTTP.

0-RTT

Pentru a reduce timpul necesar stabilirii unei noi conexiuni, o aplicație-client, care s-a conectat în trecut la un server, are posibilitatea să salveze în cache anumiți parametri și, ulterior, să stabilească o conexiune **0-RTT** cu server-ul. Asta permite aplicației-client să trimită informații imediat, fără să aștepte ca handshake-ul să fie finalizat.

Spin Bit

Una dintre cele mai lungi discuții de design din cadrul grupului de lucru QUIC, care a fost subiectul a câtorva sute de ore de e-mail-uri și ore de discuție, se referă la un singur bit: Bit-ul rotativ.

Susținătorii acestui bit insită e nevoie ca operatorii și oamenii, care sunt pe drumul dintre două capete ale unei conexiuni QUIC, să poată să măsoare timpii de încărcare.

Opozanților acestei funcționalități nu le plac potențialele pierderi de informații.

Rotirea unui bit

Ambele capete, aplicația-client și server-ul, mențin o valoare de rotație, 0 sau 1, pentru fiecare conexiune QUIC, și setează valoarea potrivită pe bit-ul rotativ în pachetele pe care le trimite.

Ambele părți, apoi, trimit pachete cu acel bit rotatic setat pe aceeași valoare, pe toată durata unui drum dus-întors și apoi comută valoarea. Efectur este, apoi, un puls de unu și zero în câmpul acelui bit, pe care observatorii îl pot monitoriza.

Această măsurătoare funcționează doar când expeditorului nu îi sunt limitate nici aplicația nici debitul, iar

reordonarea pachetelor în rețea, poate face ca informațiile neclare.

User space

Implementarea unui protocol de transfer în user-space ajută la iterarea rapidă a protocolului, deoarece evoluția protocolului este mult mai ușoară fără necesitatea ca aplicațiile-client și servere-le să își actualizeze kernel-ul sistemului de operare pentru a publica noi versiuni.

Nu există nimic inerent în QUIC care să îl împiedice să fie implementat și oferit prin kernel-urile sistemelor de operare în viitor, dacă cineva va considera că asta este o idee bună.

Mai multe implementări

Un efect evident al implementării protocolului de transport în user-space este că ne putem aștepta să vedem multe implementări independente.

Diferitele aplicații vor include probabil (sau vor fi construite ca un strat deasupra a) diferite implementări HTTP/3 și QUIC în viitorul previzibil.

API

Unul dintre motivele succesului pentru TCP-ul obșinuit și programele care îl folosesc este că folosesc un API de socket-uri standardizat. Are funcționalități bine definite și poți muta programe între sisteme de operare diferite, iar TCP funcționează la fel.

QUIC încă nu a ajuns la acest nivel. Nu există un standard de API pentru QUIC.

Cu QUIC, ai nevoie să alegi una dintre librăriile deja implementate și să rămâi la API-ul ei. Asta face, la un anumit nivel, ca aplicațiile să fie "legate" de o singură librărie. Schimbarea la o altă librărie înseamnă un alt API, iar asta poate implica foarte multă muncă.

De asemenea, de vreme ce QUIC este implementat, de obicei, în user-space, nu poate să extindă pur și simplu API-ul socket-urilor sau să arate la fel ca funcționalitățile existente TCP și UDP. Folosirea QUIC va însemna folosirea unui alt API decât cel al socket-urilor.

HTTP/3

După cum am menționat mai devreme, primul și principalul protocol de transfer peste QUIC este HTTP.

La fel cum HTTP/2 a fost introdus pentru a transfera HTTP într-un mod complet nou, HTTP/3 introduce, din nou, o modalitate nouă de a transfera HTTP prin rețea.

HTTP încă păstrează aceleași paradigme și concepte ca înainte. Încă există headere și un body, există o

cerere și un răspuns. Există verbe, cookie-uri și caching. Ce se schimbă, în principal, în HTTP/3 este cu sunt transmișii biții către partea cealaltă a comunicației.
Ca să se poată face HTTP peste QUIC, a fost nevoie de anumite schimbări, iar asta numim acum HTTP/3. Aceste schimbări au fost necesare din cauza naturii diferite a QUIC față de TCP. Aceste schimbări incluc:

- în QUIC, fluxurile sunt susținute de transferul în sine, în timp ce în HTTP/2 fluxurile sunt făcute în cadrul layer-ului HTTP.
- Din cauză că fluxurile sunt independente unul față de celălalt, protocolul pentru compresia headere-lor folosit pentru HTTP/2 nu a putut fi folosit fără a crea o situație de blocare a capului de stivă.
- Fluxurile QUIC sunt puțin diferite de fluxurile HTTP/2. Secțiunea HTTP/3 va detalia, oarecum, asta.

Adresele HTTPS://

HTTP/3 va rula folosind adrese `HTTPS://`. Lumea este plină de aceste URL-uri și a fost stabilit nepractic și cu totul nerezonabil să se introducă o nouă schemă de URL pentru noul protocol. Cum HTTP/2 nu a avut nevoie de o nouă schemă, nici HTTP/3 nu va avea nevoie.

Complexitatea adăugată în situația HTTP/3 este, totuși că, în timp ce HTTP/2 a fost o modalitate complet nouă de a transfera HTTP, era în continuare bazat pe TLS și TCP, la fel cum era și HTTP/1. Faptul că HTTP/3 este făcut peste QUIC schimbă lucurile în câteva aspecte importante.

Adresele tradiționale, în text clar, `HTTP://` for fi lăsate așa cum sunt și, pe măsură ce avansăm într-un viitor cu transferuri mai sigure, vor fi folosite din ce în ce mai puțin frecvent. Cererile către astfel de URL-uri pur și simplu nu vor fi actualizate să folosească HTTP/3. În realitate, sunt rareori actualizate să folosească chiar și HTTP/2, dar din cu totul alte motive.

Conexiunea inițială

Prima conexiune către o gazdă nouă, nefolosită în trecut pentru un URL HTTPS://, va fi făcută, probabil, peste TCP (posibil, în plus față de o încercare paralelă de conectare prin QUIC). Gazda poate fi un server tradițional, fără suport pentru QUIC sau poate exista un middlebox care ridică obstacole care previn successul conexiunii QUIC.

O aplicație-client și un server moderne vor negocia, presupus, HTTP/2 în primul handsjake. Când conexiunea a fost stabilită și server-ul răspunde la o cerere HTTP de la client, el îi poate spune clientului despre ce capacilități are și despre o preferință pentru HTTP/3.

Bootstrap cu Alt-svc

Header-ul pentru serviciul alternativ (Alt-svc) și cadrul corespondent din HTTP/2, `ALT-SVC` nu au fost create special pentru QUIC sau HTTP/3. Ele sunt parte a unui sistem deja proiectat și creat pentru ca un

server să îi poată spune unei aplicații-client: *"uite, rulez același serviciu pe ACEASTĂ GAZDĂ, folosind ACEST PROTOCOL, pe ACEST PORT"*. Vedeți mai multe detalii în RFC 7838.
Un client care primește un astfel de răspuns Alt-svc este, apoi, sfătuit, dacă implementează și dorește, să se conecteze la cealaltă gazdă folosită în paralel în fundal - folosind protocolul specificat - și, dacă schimbul reușește, să își schimbe toate operațiile pe acelea în loc de conexiunea inițială.

Dacă conexiunea inițială folosește HTTP/2 sau chiar HTTP/1, server-ul poate răspunde și să îi spună clientului că se poate conecta înapoi și să încerce HTTP/3. Poate fi către aceeași gazdă sau către o alta care știe cum să servească cererea originală. Informația oferită într-un astfel de răspuns Alt-svc are un timp de expirare, făcând ca clienții să poată să redirecționeze conexiuni și cereri ulterioare către gazdă alternativă folosind protocolul alternativ sugerat, pentru o anumită perioadă de timp.

Exemplu

Un server HTTP include un header `Alt-Svc` în răspunsul său:

```
1 Alt-Svc: h3=":50781"
```

Asta indică că HTTP/3 este disponibil pe port-ul UDP 50781, pe același nume de gazdă care a fost folosit pentru a primi acest răspuns.

Un client poate apoi să încerce să stabilească o conexiune QUIC către acea destiație și, dacă reușește, să continuie să comunice cu acea origin așa în loc de versiunea HTTP originală.

Fluxurile QUIC și HTTP/3

HTTP/3 este făcut pentru QUIC, așa că profită la maxim de fluxurile QUIC, în timp ce HTTP/2 a trebuit să își proiecteze întregul conept de fluxuri și multiplexing peste TCP.

Cererile HTTP făcute peste HTTP/3 folosesc un set specific de fluxuri.

Frame-uri HTTP/3

HTTP/3 înseamnă stabilirea unor fluxuri QUIC și trimiterea unui set de frame-uri către celălalt capăt. Există un număr mic de frame-uri (de fapt doar nouă pe data de 18 decembrie 2018!) cunoscute în HTTP/3. Cele mai importante sunt, probabil:

- HEADERS, care trimite headere HTTP compresate
- DATA, trimite conținut binar
- GOAWAY, te rog închide această conexiune

Cererea HTTP

Clientul își trimite cererea HTTP printr-un flux QUIC bidirecțional inițiat de client.

O cerere este formată dintr-un singur frame HEADERS și poate, opțional, fi urmat de unul sau două alte frame-uri: o serie de frame-uri DATA și, posibil, un frame HEADERS final pentru trailers [date decalate].

După ce a trimis o cerere, un client închide fluxuri pentru trimiteri.

Răspunsul HTTP

Server-ul trimite înapoi răspunsul său HTTP pe un flux bidirecțional. Un frame HEADERS, o serie de frame-uri DATA și, posibil, un frame HEADERS decalat.

Headere-le QPACK

Frame-urile HEADERS conțin headere HTTP compresate folosind algoritmul QPACK. QPACK este similar în stil cu algoritmul de compresie al HTTP/2 numit HPACK (RFC 7541), dar modificat să lucreze cu fluxuri trimise neordonat.

QPACK în sine folosește două fluxuri QUIC unidirecționale între două sisteme. Sunt folosite pentru a transfera infomații tabulare dinamice în oricare direcție.

Prioritizare

Unul dintre frame-urile fluxurilor HTTP/3 se numește `PRIORITY`. Este folosit ca să se stabilească prioritatea și dependințele pe un flux într-un mod similar cu modul de funcționare din HTTP/2.

Frame-ul poate stabili că un flux specific depinde un alt flux specific și poate da unui anumit flux "greutate".

Unui flux dependent ar trebui să îi fie alocate resurse ori dacă toate celelalte fluxuri de care depinde sunt închise sau dacă nu este posibil să progreseze.

O greutate a fluxului este o valoarea între 1 și 256 și este specificat ca fluxuri cu același părinte **ar trebui** să aibă alocate resurse proporțional pe baza greutății lor.

Server push

Trimiterea de către server în HTTP/3 este similară cu cea descrisă în HTTP/2 (RFC 7540), dar folosește mecanisme diferite.

O trimitere de către server este, efectiv, răspuns la o cerere pe care un client nu a trimis-o niciodată!

Trimiterile de către server sunt permise numai dacă clientul a fost de acord cu ele. În HTTP/3, clientul stabilește o limită pentru cât de multe trimiteri acceptă, informqnd server-ul care este ID-ul maxim de flux. Trecerea peste acea limită va cauza o eroare de conexiune.

Dacă server-ul consideră probabil că clientul dorește o resursă în plus, pe care nu a cerut-o, dar pe care ar trebui să o aibă oricum, poate trimite un frame `PUSH_PROMISE` (prin fluxul de cerere) arătând cum arată cererea pentru care trimiterea este un răspuns, și apoi să trimită răspunsul în sine printr-un nou flux.

Chiar și atunci când s-a spus în prealabil că trimiterile sunt acceptate de către un client, fiecare flux individual poate fi anulat la orice moment dacă clientul consideră necesar. Atunci trimite un frame `CANCEL_PUSH` către server.

Problematic

Încă de când această funcționalitate a fost discutată prima oară în dezvoltarea HTTP/2 și apoi, după ce protocolul a fost publicat pe internet, această funcționalitate a fost discutată, displăcută și lovită în nenumărate moduri pentru a o face utilizabilă.

Trimiterea nu este niciodată "gratuită", deoarece, chiar dacă salvează jumătate de round-trip, tot folosește lățime de bandă. Este deseori greu sau imposibil ca server-ul să știe cu un nivel ridicat de certitudine dacă o resursă ar trebui să fie trimisă sau nu.

Comparația cu HTTP/2

HTTP/3 este proiectat pentru QUIC, care este un protocol de transport care manipulează fluxurile singur.

HTTP/2 este proiectat pentru TCP și, deci, manipulează fluxurile în layer-ul HTTP.

Similarități

Cele două protocoale oferă clienților seturi de funcționalități practic identice.

- Ambele protocoale oferă fluxuri
- Ambele protocoale oferă suport pentru trimiteri de către server
- Ambele protocoale au compresia headere-lor, iar QPACK și HPACK sunt similare în proiectare.
- Ambele protocoale oferă multiplexing peste o singură conexiune folosind fluxuri
- Ambele protocoale oferă prioritizare pe fluxuri

Diferențe

Diferențele sunt în detalii și sunt acolo mulțumită folosirii de către HTTP/3 a QUIC:

- HTTP/3 are un suport pentru date timpurii mai bun și mai probabil să meargă, mulțumită handshake-urilor 0-RTT ale QUIC, în timp ce TCP Fast Open și TLS trimit mai puține date și întâlnesc, deseori, probleme.
- HTTP/3 are handshakes mult mai rapide datorită QUIC față de TCP + TLS.
- HTTP/3 nu există într-o versiune nesigură și necriptată. HTTP/2 poate fi implementat și folosit fără HTTPS - chiar dacă asta este acum rar peste internet.
- HTTP/2 poate fi negociat direct într-un handshake TLS cu extensia ALPN, în timp ce HTTP/3 există peste QUIC, deci are nevoie înainte de un header de răspuns `Alt-Svc` pentru a informa clientul de acest lucru.

Critici comune

UDP nu va merge niciodată

Multe companii, operatori și organizații blochează sau limitează traficul UDP în afara port-ului 53 (folosit pentru DNS) din moment ce, în ultimul timp, a fost abuzat pentru atacuri. În mod special, câteva dintre protocoalele UDP existente și implementări de server populare pentru ele au fost vulnerabile la atacuri de amplificare, în care un atacator poate genera o cantitate mare de trafic de ieșire pentru a afecta victime nevinovate.

QUIC are o construită o temperare pentru atacurile de amplificare prin cererea ca pachetul inițial să fie de minim 1200 de octeți și printr-o regulă în protocol care spune că un server NU TREBUIE să trimită în răspuns mai mult de trei mărimea cererii, fără să fi primit un pachet de la client în răspuns.

UDP este încet în kernel-uri

Asta pare a fi adevărat, cel puțin acum, în 2018. Nu putem, bineînțeles, să spunem cum se va dezvolta acest lucru și cât din asta este pur și simplu că rezultatul performanței transferurilor UDP nu a fost în prim plan pentru dezvoltatori de mulți ani.

Pentru majoritatea clienților, această "încetinire" nu va fi probabil observată vreodată.

QUIC folosește prea multă capacitate de procesare

Similar cu remarca "UDP este încet" de deasupra, asta este parțial din cauză că folosirea TCP și TLS în

lume a avut mult timp să se maturizeze, îmbunătățească și să primească asistență hardware.

Există motive să ne așteptăm că asta se îmbunătăți cu timpul. Întrebarea este cât de mult această capacitate de procesare în plus va afecta dezvoltatorii.

Asta e numai Google

Nu, nu este. Google a adus specificația inițială la IETF după ce au demonstrat, la o scară largă pe internet, că publicarea acestui tip de protocol peste UDP chiar funcționează și are performanțe bune.

De atunci, persoane de la un număr mare de companii și organizații au lucrat în organizația neutră IETF să pună la punct un protocol de transport din ea. La acea muncă au participat, bineînțeles, și angajați Google, dar la fel au participat și angajați de la un număr mare de alte companii interesate în avansarea stării protocoalelor de transfer pe internet, inclusiz Mozilla, Fastly, Cloudfare, Akamai, Microsoft, Facebook și Apple.

Asta e o îmbunătățire mult prea mică.

Asta nu este cu adevărat o critică, cât o opinie. Poate că este, și poate că este o îmbunătățire mult prea mică așa de aproape în timp de când a fost publicat HTTP/2.

HTTP/3 este probabil să funcționeze mult mai bine în rețele pline de pachete pierdute, oferă handshakes mai rapide, așa că va îmbunătăți timpii de încărcare atât percepuți cât și actuali. Dar este asta de ajuns ca benificii pentru a motiva oamenii să publice suport pentru HTTP/3 pe servere-le lor și pentru serviciile lor? Timpul și măsurătorile de perfomanță din viitor vor spune cu siguranță.

Specificația

Aici este o colecție a ultimelor ciorne oficiale pentru diverse părți și componente ale QUIC și HTTP/3.

Invariante

Proprietățile independente de versiune ale QUIC

Transfer

QUIC: Un transfer multiplexed și sigur bazat pe UDP

Recuperare

Detectarea pierderilor și control congestiilor în QUIC

TLS

Folosirea TLS pentru a securiza QUIC

HTTP

Hypertext Transfer Protocol (HTTP) peste QUIC

QPACK

QPACK: Compresia headere-lor HTTP peste QUIC

QUIC v2

Pentru a obține cel mai bun nivel de concentrare posibil pe miezul protocolului QUIC și pentru a putea livra la timp, câteva dintre funcționalitățile planificate inițial să fie parte din protocolul de bază au fost amânate și, acum, sunt planificate să fie gata într-o versiune QUIC ulterioară. Versiunea QUIC 2 sau mai mare.

Autorul acestui document deține, totuși, un glob de cristal relativ stricat, așa că nu putem spune sigur exact ce funcționalități vor apărea sau nu în versiunea 2. Putem, totuși să menționăm câteva dintre funcționalitățile și lucrurile care au fost scoase explicit din versiunea 1 a muncii pentru a fi "lucrate mai târziu" și care e posibil să apară într-o versiune 2.

Forward Error Correction

Forward error correction (FEC) este o metodă de a obține control asupra erorilor în transmisiuni în care emițătorul trimite informații redundante, iar receptorul recunoaște numai o bucată din informația care nu conține, aparent, nicio eroare.

Google a experimentat cu asta în munca lor originală asupra QUIC, dar, ulterior, a înlăturat-o din nou, de vreme ce experimentele nu au ieșit atât de bine. Această funcționalitate este supusă discuției pentru QUIC v2, dar, probabil, va dura mult până când cineva va dovedi că este o adiție folositoare, fără prea multe penalizări.

Multipath

Multipath înseamnă că transferul poate, în sine, folosi mai multe rute de rețea pentru a maximiza folosirea resurselor și creșterea redundanței.

Apărătorii SCTP din lume doresc să menționeze că SCTP deja include multipath, la fel ca și TCP modern.

Informații inconsistente

A fost discutată oferirea ca opțiune de fluxuri "inconsistente", ceea ce ar permite QUIC să înlocuiască și programele stil-UDP.

Adaptări non-HTTP

DNS peste QUIC a fost una dintre mențiunile de protocoale non-HTTP timpurii, care ar putea primi niște atenție odată ce versiunea 1 a QUIC și HTTP/3 sunt lansate. Dar cu un nou transfer ca acesta oferit lumii, nu îmi pot imagina că se va opri acolo

简体中文

本书的编撰自2018年3月开始，计划是提供HTTP/3以及其底层协议QUIC的文档，介绍它们的目的、原理、协议细节以及实现等。

本书完全免费，并且是一个协作项目，欢迎所有愿意帮忙的人前来协助！

先导知识

本书假定读者对TCP/IP网络、HTTP以及Web技术有着基本的理解。关于HTTP/2的更多知识和细节，我们推荐首先阅读HTTP/2详解。

作者

本书作者为Daniel Stenberg，他是世界上最流行的HTTP客户端软件curl的创造者与牵头开发者。Daniel在HTTP与互联网协议中已有20余年的开发经验，他也是HTTP/2详解的作者。

译者：Yi Bai

主页

本书的主页位于daniel.haxx.se/http3-explained。

帮助我们

如果您发现了本书中的任何纰漏、遗漏、错误或公然的谎言，欢迎您将受影响章节的修正版本传给我们，我们将酌情修改。我们将为提供帮助的每个人给予适当的表彰。我们希望随着时间推移，这份文档越来越好。

提交问题或者拉取请求到本书的GitHub页面是最好的反馈方式。

许可协议

为什么需要QUIC

QUIC就是一个名字，不是什么的缩略词。它的发音与英语单词“quick”相同。

QUIC在许多方面可以被视为一种新型的可靠且安全的传输层协议，它适合为形似HTTP的协议提供服务，并且可以解决一些在基于TCP和TLS传输的HTTP/2协议中存在的缺点。它是合乎情理的次世代Web传输层协议。

QUIC的用途不局限于HTTP的传输。将Web与数据更快地交付给最终用户，是创造QUIC这一新型传输层协议的最大原因和源动力。

我们接下来会解释，我们为什么要创造一个新的传输层协议，以及为什么要基于UDP来实现。

QUIC

QUIC logo

回顾HTTP/2

HTTP/2协议规范（RFC 7540）于2015年5月发表，在那之后，该协议已在互联网和万维网上得到广泛的实现和部署。

2018年初，最热的前一千个网站中约40%运行着HTTP/2，而在Firefox发出的HTTPS请求中，约70%的请求得到了HTTP/2响应。主流的浏览器、服务器以及代理都支持了HTTP/2。

HTTP/2解决了HTTP/1中存在的一大堆缺点，其中相当一部分对于开发者来说非常麻烦。在HTTP/2出现前，开发者要用许多种变通方法来解决，而HTTP/2解决了它们。

HTTP/2的一个主要特性是使用多路复用（multiplexing），因而它可以通过同一个TCP连接发送多个逻辑数据流。复用使得很多事情变得更快更好，它带来更好的拥塞控制、更充分的带宽利用、更长久的TCP连接————这些都比以前更好了，链路能更容易实现全速传输。标头压缩技术也减少了带宽的用量。

采用HTTP/2后，浏览器对每个主机一般只需要 一个 TCP连接，而不是以前常见的 六个 连接。事实上，HTTP/2使用的连接聚合（connection coalescing）和“去分片”（desharding）技术还可以进一步缩减连接数。

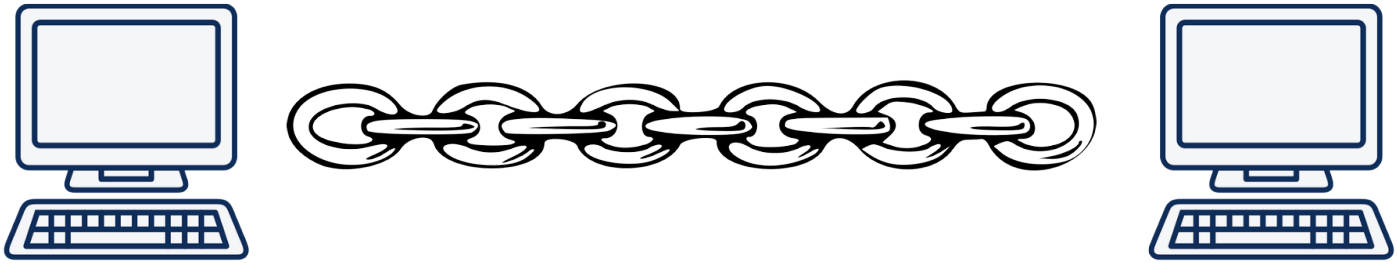
HTTP/2解决了HTTP的队头拥塞（head of line blocking）问题，客户端必须等待一个请求完成才能发送下一个请求的日子过去了。

http2 man

TCP队头阻塞

TCP队头阻塞（head of line blocking）

HTTP/2是基于TCP实现的。相比之前的版本，HTTP/2使用的TCP连接数少了很多。TCP是一个可靠的传输协议，基本上，你可以将它视为在两台计算机间建立的一个虚拟链路，由一端放到网络上的内容，最终总会以相同的顺序出现在另一端。（或者遭遇连接中断）

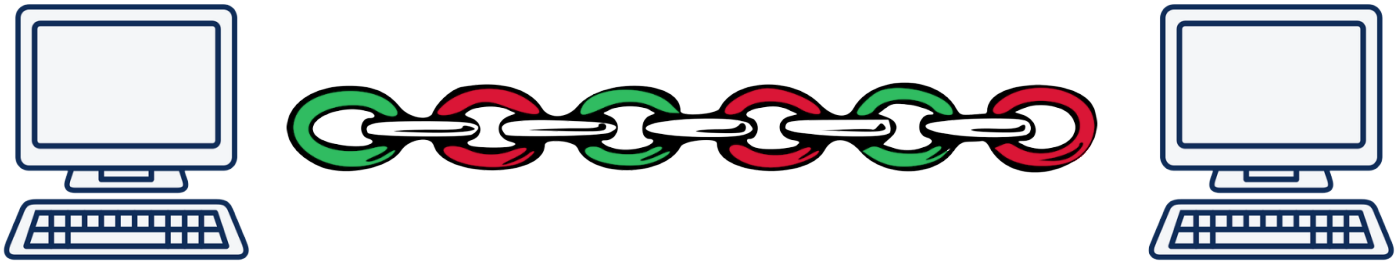


两台计算机间的TCP链路

采用HTTP/2时，浏览器一般会在单个TCP连接中创建并行的几十个乃至上百个传输。

如果HTTP/2连接双方的网络中有一个数据包丢失，或者任何一方的网络出现中断，整个TCP连接就会暂停，丢失的数据包需要被重新传输。因为TCP是一个按序传输的链条，因此如果其中一个点丢失了，链路上之后的内容就都需要等待。

如下图所示，我们一个用链条来表现一个连接上发送的两个流（传输），红色的与绿色的数据流：



不同颜色的链条代表着不同的链路

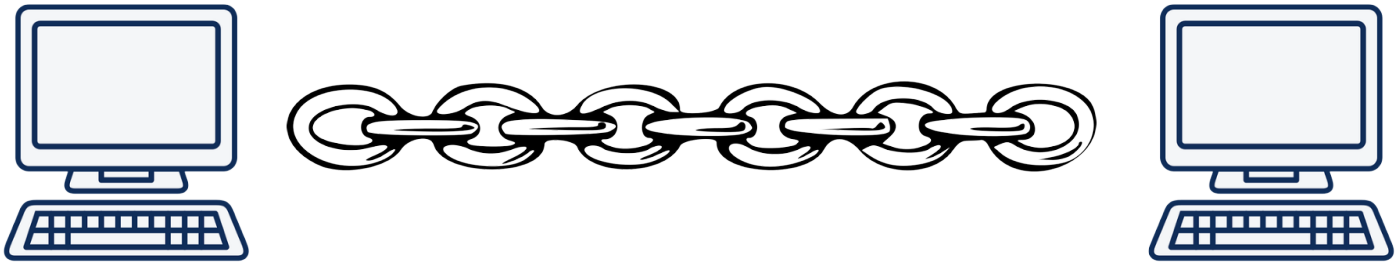
这种单个数据包造成的阻塞，就是TCP上的队头阻塞（head of line blocking）。

随着丢包率的增加，HTTP/2的表现越来越差。在2%的丢包率（一个很差的网络质量）中，测试结果表明HTTP/1用户的性能更好，因为HTTP/1一般有六个TCP连接，哪怕其中一个连接阻塞了，其他没有丢包的连接仍然可以继续传输。

在限定的条件下，在TCP下解决这个问题相当困难。

独立的数据流避免阻塞问题

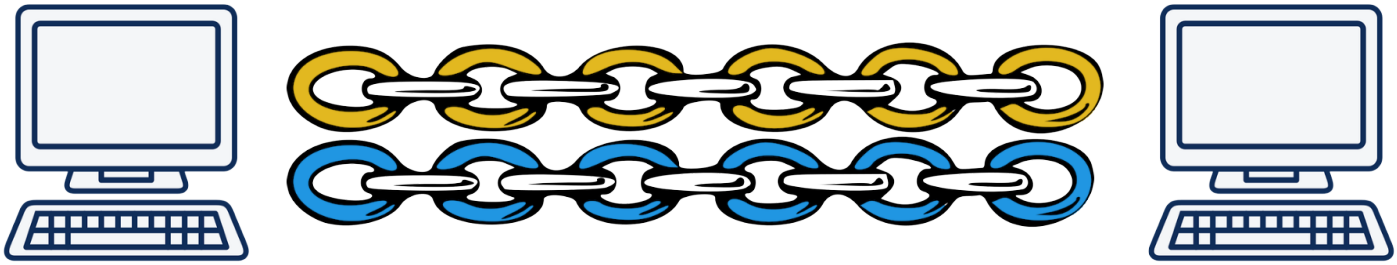
使用QUIC时，两端间仍然建立一个连接，该连接也经过协商使得数据得到安全且可靠的传输。



两台电脑间的一条QUIC链路

但是，当我们在这个连接上建立两个不同的数据流时，它们互相独立。也就是说，如果一个数据流丢包了，只有那个数据流必须停下来，等待重传。

下面是两个端点间的示意图，黄色与蓝色是两个独立的数据流。



两台电脑间的两个QUIC数据流

用TCP还是UDP

用TCP还是UDP

如果我们无法解决TCP内的队头阻塞问题，那么按道理，我们应该在网络栈中发明一个UDP和TCP之外的新型传输层协议。或者我们应该用IETF在RFC 4960中标准化的SCTP传输层协议，它也有多个我们所需的特征。

但在近些年来，因为在互联网上部署遭遇很大的困难，创造新型传输层协议的努力基本上都失败了。用户与服务器之间要经过许多防火墙、NAT（地址转换）、路由器和其他中间设备（middle-box），这些设备有很多只认TCP和UDP。如果使用另一种传输层协议，那么就会有N%的连接无法建立，这些中间设备会认为除TCP和UDP协议以外的协议都是不安全或者有问题的。如此高的的失败率一般被认为不值得再做出努力。

另外，网络栈中的传输层协议改动一般意味着操作系统内核也要做出修改。更新和部署新款操作系统内核的过程十分缓慢，需要付出很大的努力。由IETF标准化的许多TCP新特性都因缺乏广泛支持而没有得到广泛的部署或使用。

为什么不基于UDP使用SCTP

SCTP是一个支持数据流的可靠的传输层协议，而且在WebRTC上已有基于UDP的对它的实现。

这看上去很好，但与QUIC相比还不够好，它：

- 没有解决数据流的队头阻塞问题
- 连接建立时需要决定数据流的数量
- 没有稳固的TLS/安全性支持
- 建立连接时候需要4次握手，而QUIC一次都不用（0-RTT）
- QUIC是类TCP的字节流，而SCTP是信息流（message-based）
- QUIC连接支持IP地址迁移，SCTP不行

若要了解更多SCTP与QUIC的差异，请参阅A Comparison between SCTP and QUIC

协议僵化

互联网（Internet）——多个网络互联之网。为了保证互联网工作正常，我们需要在互联网各处搭建各种设备。这些在互联网上分布式架设的设备被称为中间设备（middle-box）。传统网络传输中两个端点之间的中间设备服务于网络数据传输过程。

这些中间设备有着许多、各式各样的用途和目的，我们简单来说，这些设备是为了实现放置人在该位置所要完成的特定目的而安置。

中间设备的目的包括：在网络之间转发（路由）数据包、阻挡恶意流量、执行地址转换（NAT）、提升性能、监视流量等等。

为了完成这些目的，这些设备必须了解网络以及它们所要监视或修改的数据包协议。它们为此依赖于一些软件——一些很少升级的软件。

虽然这些设备是将互联网“粘”在一起的关键元素，但是它们经常跟不上最新的技术。网络的核心部分与边缘部分（客户端、服务器）相比，更新很慢。

所以当这些设备决定了经过的流量是否合格时，就有些问题了——在这些设备部署之后的一段时间里，协议有了新的特征。而在这些设备引入（了解）这些新特性之前，它们会认为这种特征的数据包是非法的、恶意的，于是会将这种流量直接扔掉，或是拖延到用户不再想使用这些新特征的程度。

这种问题就被称之为“协议僵化”。

协议僵化也影响了TCP协议的改变：当客户端与远程服务器之间的某些中间设备检测到对于它们来说未知的新的TCP选项时，中间设备将拦截这些流量，因为它们不知道这些选项的作用。如果中间设备监测了协议的实现细节，它们会学习到协议的典型行为，在一段时间后，这些行为变得没法更改。

尽可能将通信加密是对抗僵化的唯一有效手段，加密可以防止中间设备看到协议传输的绝大部分内容。

安全性

QUIC始终保证安全性。QUIC协议没有明文的版本，所以想要建立一个QUIC连接，就必须通过TLS 1.3来安全地建立一个加密连接。如上文所说，加密可以避免协议僵化等拦截和特殊处理。这也使QUIC具有了Web用户所期望的所有的HTTPS安全特性。

QUIC只在加密协议协商时会发送几个明文传送的初始握手报文。

减少延迟

与TCP的3次握手相比，QUIC提供了0-RTT和1-RTT的握手，这减少了协商和建立新连接时所需的时间。

除此之外，QUIC提供了提早传输更多数据的“早期数据”（early data）特性，并且它的使用比TCP快速打开（TCP Fast Open）更加简便。

因为数据流概念的引入，客户端不用等待前一个连接结束，便可以与同一个主机建立另一个逻辑连接。

TCP快速打开存在的问题

TCP快速打开选项由2014年12月发布的RFC 7413定义，该规范中介绍了客户端如何通过第一个TCP SYN报文向服务器推送数据。

但在互联网上，对这个选项的实际支持仍需要时间，在2018年的今天，这个选项还是充满很多问题。想要在TCP栈上实现这个选项以获得改进的工程师，不仅要注意操作系统的版本，还要小心地处理发生问题之时，如何优雅地降级到没有该选项的状态。已知有多个网络在积极破坏这种TCP握手，从而干扰了TCP快速打开的流量。

协议进展

最初的QUIC协议由Jim Roskind在Google设计并于2012年实现，经过Google的扩大试验后，于2013年向全世界公开发布。

回顾那时，QUIC还是“快速UDP互联网连接”（Quick UDP Internet Connections）的缩写，但现在已经不是了。

Google实现了QUIC协议，并随后将它部署在其广泛流行的浏览器（Chrome）和最流行的服务（搜索、Gmail、YouTube等等）中。他们相当迅速地迭代该协议的版本，经过一段时间，协议的理念被许多用户的使用证明可以面向大量用户可靠运作。

2015年6月，首个QUIC的互联网草案被提交到IETF以进行标准化，但直到2016年下半年，一个QUIC工作组才被批准成立并投入工作。随后它在各方的高度关注下迅速发展。

在2017年，Google的QUIC工程师称，整个互联网中大约有7%的流量在使用该协议（Google版本）。

IETF

IETF为QUIC标准化成立的QUIC工作组很快就决定，IETF标准化的QUIC协议应该支持HTTP以外的其他应用层协议。Google版的QUIC只传输HTTP——在实践中，它则被用来传输符合HTTP/2帧语义的片段。

另外，工作组最初也决定IETF-QUIC应该基于TLS 1.3进行加密与安全传输，而不使用Google版QUIC定制的方法。
为满足不局限于HTTP的传输需求，IETF QUIC协议的架构被分为两个独立的层：传输层QUIC和“基于QUIC的HTTP”（HTTP over QUIC）层。后者在2018年11月被重命名为HTTP/3。

尽管这种分层结构看似人畜无害，但这实质上造成IETF-QUIC与Google版QUIC有着诸多不同。

工作组很快发现了这一点，为保持适当关注和能按时交付第一版QUIC，工作组的重心转移到了先交付“基于QUIC的HTTP”传输部分，非HTTP传输部分将留待今后研究。

2018年3月，当我们开始写这本书的时候，第一版QUIC最终的规范计划于2018年11月发布。不过发布时间在这之后被推迟到了2019年7月。

在IETF-QUIC取得进展的同时，Google团队已经整合了IETF版本的细节并逐渐推进他们的协议版本，以期最终可能符合IETF定义的规范。尽管如此，Google在他们的浏览器和服务中继续使用自己的QUIC版本。

正在开发的大多数新实现 已经决定着眼于IETF版本，与Google版本并不兼容。

HTTP/2的经验

HTTP/2规范RFC 7540发布于2015年5月，只比QUIC首次进入IETF早一个月。

HTTP/2的发布为HTTP进化奠定了基础，同时这让HTTP/2工作组认识到迭代更新的HTTP版本会比从第一版到第二版（花费约16年）快得多。

随着HTTP/2的发布，用户和软件堆栈不再假设HTTP是一个序列化的、基于文本的协议。

HTTP-over-QUIC(HTTP/3)建立在HTTP/2的基础上，并从中借鉴了许多概念。由于某些具体的概念已被QUIC所涵盖，所以从HTTP层中移除。

标准化进展情况

QUIC工作组自2016年底以来在积极标准化该协议，现计划于2019年7月之前完成。

到2018年11月为止，还没有过大型的HTTP/3互通性测试。目前来说，只有2个实现进行过测试，且这两个实现不基于浏览器和主流的开源服务器软件。

QUIC工作组的wiki页面目前列出了大约15种QUIC实现（QUIC实现列表），但距离它们都能与最新版的规范草案互操作仍有很长的路。

实现QUIC并不容易，且到目前为止，协议本身还在不断演变。

服务器

Apache和Nginx还没有对QUIC支持的公开声明。

客户端

还没有任何主流浏览器的任何状态的任何版本支持IETF版本的QUIC或者HTTP/3协议。

Google Chrome在数年前已经支持Google版的QUIC，但是该版本不能与官方的QUIC版本互操作，且它的HTTP实现与HTTP/3不同。

实现的障碍

为了避免重复发明轮子，以及依靠可信赖的现有协议，QUIC决定使用TLS 1.3作为它的加密和安全协议层。不过工作组决定大幅精简QUIC中TLS的使用，只使用“TLS信息”（TLS Messages）而不是协议中的“TLS记录”（TLS Records）。

这听上去可能人畜无害，但也事实上成为了很多QUIC堆栈实现者的重大障碍。现存的支持TLS 1.3的TLS库都没有提供此功能的API并允许QUIC访问它。有一些QUIC的实现由大型机构完成，这些机构可能有自己的TLS协议栈，但并不是所有实现都能如此。

例如，主流的重量级开源软件OpenSSL就没有这些API，且到目前（2018年11月）为止，没有表达过在任何时间点提供这些API的意愿。

这最终将成为QUIC协议栈部署的障碍。因为QUIC要么基于其他TLS库，要么使用补丁版OpenSSL或者等待OpenSSL版本更新。

操作系统内核、CPU负载

据Google和Facebook称，与基于TLS的HTTP/2相比，它们大规模部署的QUIC需要近2倍的CPU使用量。

对此的进一步解释包括：

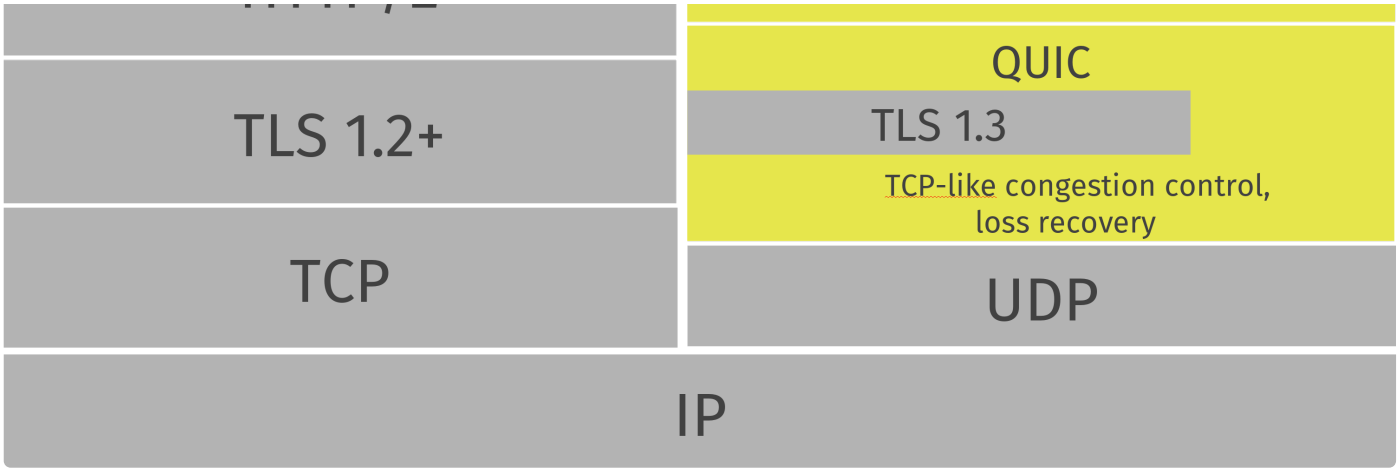
- Linux内核的UDP部分没有得到像TCP堆栈那样的优化，因为传统上没有使用UDP进行如此高速的信息传输。
- TCP和TLS有硬件加速（负载卸载到硬件，offload），而这对于UDP很罕见，对于QUIC则基本不存在。

就上述理由，我们可以相信QUIC的CPU使用量能随着时间的推移得到改善。

协议特点

本章从高层次出发概述QUIC协议。

下图是使用HTTP传输时，HTTP/2（左）和HTTP/3（右）的协议栈对比。



QUIC logo

基于UDP

基于UDP的传输层协议

QUIC是基于UDP在用户空间实现的传输协议。如果不观察细节，你会觉得QUIC跟UDP报文差不多。

基于UDP意味着它使用UDP端口号来识别指定机器上的特定服务器。

目前已知的所有QUIC实现都位于用户空间，这使它能得到更快速的迭代（相较于内核空间中的实现）。

跑得起来吗？

有一些网络上的中间设备会拦截端口53（用于DNS）以外的UDP流量。还有一些网络会节流（throttle）UDP流量，使得QUIC的表现慢于基于TCP的协议。更多网络的表现未知。

在可预见的未来，所有基于QUIC的传输可能会配有一个优雅地回退到另一个基于TCP的后备方案的替代机制。据Google多位工程师早前的报告，协议的失败率不足10%。

会好起来吗？

如果QUIC被证明确实是互联网世界的一个有益补充，用户会希望能正常使用QUIC，网络公司就可能重新解决上述的拦截。多年以来，随着QUIC取得进展，在互联网上建立和使用QUIC的成功率有所提高。

可靠性

虽然UDP不提供可靠的传输，但QUIC在基于UDP之时增加了一层带来可靠性的层。它提供了数据包重传、

拥塞控制、调整传输节奏（pacing）以及其他一些TCP中存在的特性。
只要连接没有中断，从QUIC一端传输的数据迟早会出现在另一端。

数据流

类似SCTP、SSH和HTTP/2，QUIC在同一物理连接上可以有多个独立的逻辑数据流。这些数据流并行在同一个连接上传输，不影响其他流。

连接在两个端点之间经过类似TCP连接的方式协商建立。QUIC连接基于UDP端口和IP地址建立，而一旦建立，连接通过其“连接ID”（connection ID）关联。

在已建立的连接上，双方均可以建立传输给对方的数据流。单一数据流的传输是可靠、有序的，但不同的数据流间可能无序传送。

QUIC可对连接和数据流分别进行流量控制（flow control）。

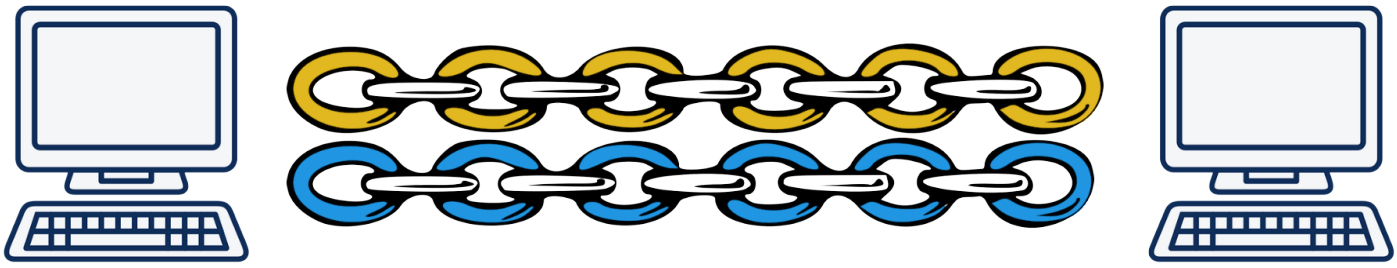
进一步细节参见连接和数据流。

有序交付

QUIC的单个数据流可以保证有序交付，但多个数据流之间可能乱序。这意味着单个数据流的传输是按序的，但是多个数据流中接收方收到的顺序可能与发送方的发送顺序不同！

举个例子：服务器传送流A和B到客户端。流A先启动，然后是流B。在QUIC中，丢包只会影响该包所处的流。如果流A发生了一次丢包，而流B没有，流B将继续传输直到结束，而流A将会进行丢包重传过程。而在HTTP/2中这不可能发生。

下图展示了连通两个QUIC端点的单一连接中的黄色与蓝色的数据流。它们互相独立，所以可能乱序到达，但是每个流内的信息将按序可靠到达。



两台电脑间的两个QUIC数据流

快速握手

QUIC提供0-RTT和1-RTT的连接建立，这意味着QUIC在最佳情况下不需要任何的额外往返时间便可建立新连接。其中更快的0-RTT仅在两个主机之间建立过连接且缓存了该连接的“秘密”（secret）时可以使用。

早期数据（Early data）

QUIC允许客户端在0-RTT的情况下直接捎带数据。这使得客户端能尽早向对方传送数据，当然也使得服务器能更快地发回数据响应。

TLS 1.3

QUIC使用TLS 1.3传输层安全协议（RFC 8446）。QUIC没有非加密的版本。

与更早的TLS版本相比，TLS 1.3有着很多优点，但使用它的最主要原因是其握手所花费的往返次数更低，从而能降低协议的延迟。

Google的传统QUIC使用一个自行定制的加密法。

传输层与应用层协议

IETF版QUIC是一个传输层协议，在该协议之上可以运行其他应用层协议。初始的应用层协议是HTTP/3（h3）。

传输层协议负责连接和数据流处理。

在Google的传统QUIC中，传输层与HTTP融在一起，为包揽一切的全功能设计，它是一个更有指向性的“基于UDP传输HTTP/2帧”（send-http/2-frames-over-udp）的协议。

QUIC之上的HTTP协议

HTTP层以HTTP风格传输内容，包括使用QPACK进行HTTP头部压缩——这和HTTP/2中使用HPACK压缩头部类似。

HPACK算法依赖于数据流的有序交付，由于HTTP/3的数据流之间可能乱序，所以该算法需要修改才能使用。QPACK可被视作适用于QUIC版本的HPACK。

QUIC之上的非HTTP协议

基于QUIC传输非HTTP协议的相关工作已被推迟到第一版QUIC发布之后实现。

QUIC工作原理

本章节将解释QUIC传输层协议各基本模块的功能，而不会逐比特逐字节解释协议的报文。如果你想自己实现QUIC，这些介绍会加强你对协议的理解，但有关具体细节，请参考IETF的互联网草案（Internet Draft）和

RFC。

1. 建立一个连接
2. 协商安全的TLS连接
3. 使用数据流

连接

QUIC连接是两个QUIC端点之间的单次会话（conversation）过程。QUIC建立连接时，加密算法的版本协商与传输层握手合并完成，以减小延迟。

在连接上实际传输数据时需要建立并使用一个或多个数据流。

连接ID（Connection ID）

每个连接过程都有一组连接标识符，或称连接ID，该ID用以识别该连接。每个端点各自选择连接ID。每个端点选择对方使用的连接ID。

连接ID的基本功能是确保底层协议（UDP、IP及其底层协议）的寻址变更不会使QUIC连接传输数据到错误的端点。

利用连接ID的优势，连接可以在IP地址和网络接口迁移的情况下得到保持——而这TCP永远做不到。举例来说，当用户的设备连接到一个Wi-Fi网络时，将进行中的下载从蜂窝网络连接转移到更快速的Wi-Fi连接。与此类似，当Wi-Fi连接不再可用时，将连接转移到蜂窝网络连接。

端口号

QUIC基于UDP建立，因此使用16比特的UDP端口号字段来区分传入的不同连接。

版本协商

客户端的QUIC连接请求会告知服务器所希望使用的QUIC协议版本，服务器端会回复一系列支持的版本供客户端选择。

使用TLS的连接

在初始的数据包建立连接之后，连接发起者会马上发一个加密的帧以开始安全层握手。安全层使用TLS 1.3协议。

在QUIC中，没有方法或机制避免使用TLS连接。该设计旨在使中间设备难以篡改数据包，防止协议僵化。

数据流

数据流（Streams）在QUIC中提供了一个轻量级、有序的字节流的抽象化。

QUIC中有两种基本的数据流类型：

- 从发起者到对等端（Peer）的单向数据流。
- 双向均可发出数据的双向数据流。

连接端点的任意一方都可以建立这两种数据流，数据流之间可并行、交错地传输，并且可以被取消。

通过QUIC发送数据需要建立一个或多个数据流。

流量控制（Flow control）

每个数据流都有独立的流量控制，端点可以通过此实现内存控制和反压（back pressure）。数据流的创建本身也有流量控制，连接双方可以声明最多愿意创建几个流ID。

流标识符

数据流通过一个无符号的62比特整数标识，也称流ID。流ID的最低2位比特用于识别流的类型（单向或双向）和流的发起者。

流ID的最低1位比特（0x1）用于识别流的发起者。客户端发起双数（最低位置0）流，服务器发起单数（最低位置1）流。

第2个比特（0x2）识别单/双向流。单向流始终置1，双向流则置0。

流并发

QUIC允许任意数量的并发流。端点通过闲置最大流ID来控制并发活动的传入流数量。

每个端点指定自己的最大流ID数，并只对对等端端点有效。

收发数据

端点使用流来收发数据，这是流的最终用途。QUIC数据流是有序的字节流抽象。但是，不同流之间是无序

的。

流优先度

如果正确设置了各流的优先度，流复用机制可以显著提升应用的效率。使用其他多路复用协议（如HTTP/2）的经验表明，有效的优先度划分策略对效率具有显著的正面影响。

QUIC本身没有提供交换优先度信息的报文。接收优先度信息依赖于使用QUIC的应用层。应用层可以定义所有符合其语义的优先度方案。

基于QUIC使用HTTP/3时，优先度信息在HTTP层完成。

0-RTT

先前已连接过一个服务器的客户端可能缓存来自该连接的某些参数，并在之后与该服务器建立一个无需等待握手完成就可以立即传输信息的**0-RTT**连接，从而减少建立新连接所必需的时间。

旋转比特位

在QUIC工作组的设计讨论中，最长的主题之一就是旋转比特位，人们花费了数百封邮件和数百个小时来讨论它。

旋转比特位的支持者认为，两个QUIC端点之间路径上的运营商和人员需要有办法来测量延迟。

反对者则反感此功能潜在的信息泄露。

旋转一个比特

QUIC连接的客户端、服务器这两个端点各为每一个QUIC连接维护一个旋转的值——0或1，在传送时候它们在报文中设置该值。

然后，在每一次往返时，连接双方都翻转这一比特的值。效果是观察者可以检测该比特字段的0与1脉冲。

这一观测只在发送方未被应用层或流量控制限制的情况下有效，并且网络上经过重新排序的数据包也会给数据带来噪声。

用户空间实现

在用户空间中实现一个传输层协议有助于协议的快速迭代，协议的演进更为容易，不需要客户端和服务器更新其操作系统内核才能部署新的版本。

QUIC本身没有固有的东西阻碍未来在操作系统内核中实现和提供QUIC协议。

众多的实现

在用户空间中实现一个新的传输层协议时，一个显而易见的效果是我们会看到很多独立的实现。

在可预见的未来，不同的应用程序可能包含（或基于）不同的HTTP/3和QUIC实现。

API

常规TCP与程序最成功的因素之一便是标准化的套接字（socket）API。其API有着定义良好的功能，使用它能让你轻松地在各操作系统之间移植程序，因为TCP采用同样的方式运作。

但QUIC不是如此。QUIC目前没有标准化的API。

使用QUIC时，你需要选择一个现有的库实现，并坚持使用它的API。这在某种程度上把应用“绑定”到了单一的库上。换库意味着使用另外一套API，这可能带来相当的工作量。

另外，由于QUIC一般在用户空间中实现，所以它不像现有的TCP和UDP套接字API那样能轻松扩展。使用QUIC意味着选择了套接字API之外的另一套API。

HTTP/3

上文提到过，基于QUIC传输的第一个也是最基础的协议是HTTP。

就像HTTP/2是通过网络传输HTTP流量的一种新方式，HTTP/3是另一种通过网络传输HTTP的新方法。

HTTP的范例和概念没有改变。它含有头部（header）和正文（body），请求和回复，还有动词（verb）、Cookie和缓存。HTTP/3的主要改变是将这些报文比特传送到另一端的方式。

为了使HTTP可以通过QUIC传输，协议的某些方面要进行修改，修改的结果便是HTTP/3。这些必要修改是因QUIC与TCP在某些性质上的不同所致，修改包括：

- 在QUIC中，数据流由传输层本身提供，而在HTTP/2中，流由HTTP层完成。
- 由于数据流互相独立，HTTP/2中使用的头部压缩算法如果不做改动，会造成队头阻塞。
- QUIC流与HTTP/2略有不同。本书的HTTP/3章节会做详细介绍。

HTTPS:// URL

HTTP/3将使用`HTTPS://` URL履行。我们的世界里充斥着HTTPS URL，并且为新协议引入另一种URL方案被认为不切实际且完全不合理。如同HTTP/2一样，HTTP/3不会引入新的URL方案。

HTTP/2是传输HTTP的一种新方式，但是它还是基于TLS和TCP，这和HTTP/1一样。而在基于QUIC的

HTTP/3中，情况更加复杂，它在一些重要的地方做了一些改变。

历史遗留下来的明文`HTTP://` URL的处理方式将保持原样，随着我们迈入安全传输更加普及的未来，它的使用可能会越来越少。对HTTP URL的请求不会升级为使用HTTP/3。在实践中，因为其他一些理由，它们也很少升级到HTTP/2。

初始连接

当尝试连接到一个全新的、未访问过的网站时，到HTTPS:// URL的连接可能必须通过TCP（也许会有并行的一个QUIC连接）。因为主机可能是一个不支持QUIC的传统服务器，或者链路之间可能有阻碍QUIC成功连接的中间设备。

现代的浏览器和服务器可能在首次握手时协商HTTP/2协议。在连接建立并且服务器响应客户端的HTTP请求时，服务器可以通告客户端它对HTTP/3的支持与偏好。

使用Alt-svc自举

替代服务（alternative service, Alt-svc:）头部和它相对应的`ALT-SVC` HTTP/2帧并不是特别为QUIC和HTTP/3设计的。它是为了让服务器可以告诉客户端*“看，我在这个主机的这个端口用这个协议提供相同的服务”*而设计的。详见RFC 7838。

如果初始连接使用的是HTTP/2（甚至HTTP/1），服务器可以响应并告诉客户端它可以再试试HTTP/3。连接可以指向相同主机或者不同但提供相同服务的主机。Alt-svc回复中有一个到期计时器，让客户端可以在指定的时间内使用建议的替代协议将后续的连接和请求直接发送给替代主机。

例子

一个HTTP服务器的响应中包含了如下的一个`Alt-Svc:`头部：

```
1 Alt-Svc: h3=":50781"
```

这指示了同一名称的主机在UDP端口50781提供HTTP/3服务。

然后，客户端可以尝试与该端口建立QUIC连接。如果成功，后续将通过该连接继续通信，代替初始的HTTP版本。

QUIC流与HTTP/3

HTTP/3针对QUIC设计，所以它可以利用QUIC流的所有好处。而HTTP/2不得不在TCP之上构建它的数据流和复用概念。

通过HTTP/3传输的HTTP请求使用一系列的数据流完成。

HTTP/3帧（frame）

HTTP/3意味着建立QUIC数据流，并将一系列帧发送给对方。HTTP/3中的数据帧种类不多且固定（截至2018年12月18日有九种）。最关键的帧可能是：

- HEADERS：发送压缩的HTTP头部
- DATA：发送二进制数据内容
- GOAWAY：请关闭此连接

HTTP请求

客户端通过其发起的 双向 QUIC流来发送HTTP请求。

一个请求包括一个HEADERS帧，之后可能有一两种其他的帧：一系列的DATA帧，以及可能有一个作为末尾的HEADERS帧。

发送一个请求后，客户端会关闭该数据流以进行发出。

HTTP响应

服务器在双向流上发回其HTTP响应。其中含有一个HEADERS帧，一系列DATA帧，末尾可能有一个HEADERS帧。

QPACK头部

HEADERS含有用QPACK算法压缩的HTTP头部。QPACK与HTTP/2中的HPACK（RFC 7541）类似，并针对乱序流做了相应修改。

QPACK本身在两个端点间使用两个额外的单向QUIC流，用于在两个方向上传递动态表信息。

优先度

HTTP/3中有一种帧是优先度（`PRIORITY`）。与HTTP/2中的类似，它用于设定一个流的优先度和依赖关系。

该帧可以设定一个流依赖于另一个流，也可以设定特定流的“权重”。

服务器应该只在一个流所依赖的所有流都被关闭，或者都无法取得进展时为该流分配资源。

一个流的权重是介于1到256之间的值，有着相同父系流的流**应该**按照权重的比例分配资源。

服务器推送

HTTP/3的服务器推送与HTTP/2（RFC 7540）类似，但机制上有所不同。

服务器推送实际上就是对一个未曾发出的客户端请求做出响应！

服务器推送仅在客户端同意的前提下才允许发出。在HTTP/3中，客户端甚至能通过通告给服务器的最大推送流ID来设置所接受推送的次数限制。超出限制将导致连接错误。

如果服务器端认为客户端可能需要某个并未要求但应该有的额外资源，服务器可以通过请求流发送一个 `PUSH_PROMISE` 帧，使该推送请求看上去像是一个响应，然后通过新的流发送实际响应。

虽然客户端之前已经表示过推送可接受，但如果客户端认为适合，每个推送流仍可以随时取消，然后发送一个 `CANCEL_PUSH` 帧到服务器。

问题重重

自从推送这一特性在HTTP/2中讨论、开发、部署以来，它就备受争议、讨论和抨击。为了让它有用，人们付出了许多努力。

推送从来不是没有代价的，尽管它省了半个往返的延迟，它还是会消耗带宽。服务器也很难或不可能从高层面确定一个资源是否应该被推送过去。

与HTTP/2的比较

HTTP/3面向QUIC设计，QUIC是一个自己处理数据流的传输层协议。

HTTP/2面向TCP设计，因此数据流在HTTP层处理。

相似之处

这两个协议为客户端提供了几乎相同的功能集。

- 两者都提供数据流
- 两者都提供服务器推送
- 两者都有头部压缩，QPACK与HPACK的设计非常类似
- 两者都通过单一连接上的数据流提供复用
-

- 两者都提供数据流的优先度设置

不同之处

两个协议的主要不同点在于细节，不同之处主要由HTTP/3使用的QUIC带来。

- 得益于QUIC的0-RTT握手，HTTP/3可以提供更好的早期数据支持，而TCP快速打开和TLS通常只能传输更少的数据，且经常存在问题。
- 得益于QUIC，HTTP/3的握手速度比TCP+TLS快得多。
- HTTP/3不存在明文的不安全版本。尽管在互联网上很少见，HTTP/2还是可以不配合HTTPS来实现和使用。
- 通过ALPN拓展，HTTP/2可以直接在TLS握手时进行协商。HTTP/3基于QUIC，所以需要凭借响应中的 `Alt-Svc:` 头部来向客户端宣告。

常见批评

UDP永远不会通

很多企业、运营商和组织对53端口（DNS）以外的UDP流量进行拦截或者限流，因为这些流量近来常被滥用于攻击。特别是一些现有的UDP协议和实现易受放大攻击（amplification attack）威胁，攻击者可以控制无辜的主机向受害者投放发送大量的流量。

QUIC内置了对放大攻击的缓解处理。它要求初始数据包不小于1200字节，并且协议中限制，服务器在未收到客户端回复的情况下，不能发送超过请求大小三倍的响应内容。

内核处理UDP很慢

我们必须承认这在2018年的今天是一个事实。当然，UDP技术会发展，这些年开发者对UDP的重视程度也不够，这些东西都自不必说了。

对于大多数客户端来说，这个程度的“缓慢”从未被觉察到。

QUIC太吃CPU

类似上文的“UDP很慢”，一部分原因是TCP和TLS长期以来的成熟发展、改进，以及得到硬件协助，造成UDP看上去比较慢。

我们有理由期望这会随着时间得到改善。问题在于，这额外的CPU占用会对部署者带来多大的影响。

只有Google在弄

并非如此。Google通过大规模的部署证明，通过UDP部署这种协议可以正常运行且表现良好，这为IETF带来了初始的规范。

在那之后，很多公司和组织的人员都在这个利益方中立的IETF组织下推进标准化。在这个阶段，虽然Google的雇员也有参与，但Mozilla、Fastly、Cloudflare、Akamai、微软、Facebook、苹果等等很多公司的员工也参与进来，共同推进互联网的传输层协议。

进步太小

这个是一个观点，而不是批评。也许进步是很小，这可能与相距HTTP/2的发布很近有着关系，时间太短了。

HTTP/3在高丢包的网络中可能表现更好，它提供了更快的握手，所以能改善可感知和实际的延迟。这些进步足够推动人们在服务器和服务上部署HTTP/3的支持吗？时间以及未来的性能测试会给我们答案！

技术标准

以下是QUIC和HTTP/3各个部分的最新官方IETF草案列表。

不变性

Version-Independent Properties of QUIC

传输层

QUIC: A UDP-Based Multiplexed and Secure Transport

自动恢复

QUIC Loss Detection and Congestion Control

TLS

Using Transport Layer Security (TLS) to Secure QUIC

HTTP

Hypertext Transfer Protocol (HTTP) over QUIC

QPACK

QPACK: Header Compression for HTTP over QUIC

QUIC v2

为了尽力专注于QUIC的核心特性，以及为了赶上发布进度，最初计划为核心协议一部分的几个特性已被推迟，计划到QUIC第二版或以后的版本中完成。

本文档的作者买的水晶球是山寨的，所以我们不能知道哪些特性会或者不会出现在第二版中。但我们可以谈谈那些在第一版中被标记为“之后做”的可能出现在第二版中的特性。

前向纠错（Forward Error Correction）

前向纠错（FEC）是一种通过向接收方发送冗余数据，使得接收方能快速识别出含有不明显错误的一种错误控制方式。

Google在它们的原版QUIC成果中实验过该特性，但由于实验结果不理想而已经去除。此特性可供QUIC v2讨论，但可能需要有人证明它在代价不大的情况下确实有用。

多路径（Multipath）

多路径意味着通信可以通过多个网络路径传输，以期最大化利用资源并增加冗余。

SCTP支持者可能有话说了，SCTP和现代TCP早就已经支持这一点。

不可靠数据

通过QUIC传输不可靠的数据流也得到过广泛讨论，这样QUIC就能在传统UDP风格的应用程序中作为代用品。

非HTTP适应

基于QUIC的DNS是早期被提出的非HTTP协议之一，这可能在QUIC v1和HTTP/3发布后得到更多的关注。但我想，如果我们拥有了这样的新型传输层协议，DNS远不是终点。

繁體中文

撰寫此書的計畫始於2018年3月。 目的是要記錄 HTTP/3 及其基礎協定: QUIC。包含為什麼，它們是如何運作，協定的內容及如何實作...等等。

此書完全免費並且是協作的成果，代表著需要任何和所有想提供幫助的人共同努力。

前提

在閱讀此書前，我們期望您對 TCP/IP 網路架構，HTTP 和 Web 的基礎知識有基本的了解。 若想要進一步了解 HTTP/2，我們建議您先閱讀以下的內容 http2 explained。

作者

本書作者為 Daniel Stenberg，他是世界上最流行的 HTTP 客戶端程式 curl 的創造者與領頭開發者。 Daniel 在 HTTP 與網路通訊協定中已有20多年的開發經驗，他也是 詳解 HTTP/2 的作者。

繁體中文譯者：Wei Jhih Lian

主頁

本書的主頁位於 daniel.haxx.se/http3-explained。

協助我們

如果您發現了本書中的任何錯字、遺漏或是錯誤的訊息內容，歡迎您將受影響章節的修正版本回報給我們，我們將根據情況修正。 我們將為提供幫助的每個人給予適當的表彰。同時我們希望隨著時間推移，此書的內容能越來越好。

您可以提交 issue 或是 pull request 於此書的 GitHub 頁面。

許可協議

為什麼是QUIC

QUIC 是一個名字，它不是某個單詞的縮寫。它的發音與英文中的 "quick" 相同。

QUIC 在許多方面可以被視為一種新型可靠且安全的傳輸層協定，它適用於像是 HTTP 協定，並且可以解決一些在基於 TCP 和 TLS 傳輸的 HTTP/2 協定中存在的缺點。理所當然的，它成為網路傳輸發展的下一步。

QUIC 的用途不侷限於 HTTP 的傳輸。將 Web 和資料整體上能更快地交付給終端客戶的渴望，是創造 QUIC 這一新型傳輸層協定的最大原因。

我們接下來會解釋，我們為什麼要創造一個新的傳輸層協定，以及為什麼要基於 UDP 來實現。

QUIC logo

回顧HTTP/2

自 HTTP/2 規範 RFC 7540 於2015年5月發布以來，此協定已在 Internet 和 World Wide Web 上廣泛的實作和部署。

在2018年初，排名前 1000 位的網站中有 40％ 運行於 HTTP/2，在 Firefox 發出的請求中，大約 70％ 的 HTTPS 請求得到了 HTTP/2 的回應，且所有主要瀏覽器，服務器和代理器也都支援 HTTP/2。

HTTP/2 解決了 HTTP/1 中的許多缺點，隨著第二版 HTTP 的導入，用戶可以停止使用許多不必要的

workarounds。 這些 workarounds 有一部分對於 Web 開發人員來說負擔很重。

HTTP/2 的主要特徵之一是它利用了多工（ multiplexing ），因此許多邏輯流是通過同一個 TCP 連接發送的，這使很多事情變得更好更快。 它使擁塞（ congestion) 控制運作的更好，它使用戶可以更好地使用 TCP，更充分的使用頻寬，更長久的 TCP 連接 - 這樣做的好處是，它們比以前更頻繁地達到全速運行。標頭壓縮使其能使用較少的頻寬。

使用 HTTP/2 時，瀏覽器通常對每個主機使用 *一個* TCP 連接，而不是先前的 *六個*。 實際上，與 HTTP/2 一起使用的連接合併（ connection coalescing ）和 "去分片" 技術實際上可以減去更多的連接。

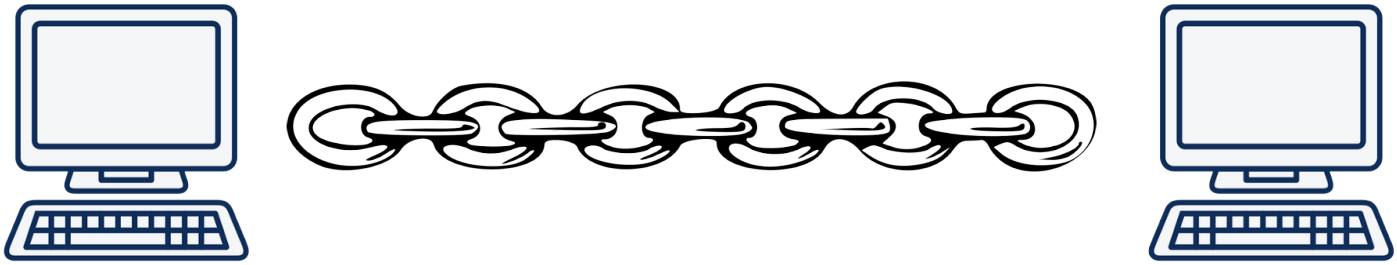
HTTP/2 解決了 HTTP 隊頭阻塞的問題，即客戶端必須等待隊中的第一個請求完成，才能發出下一個請求。

http2 man

TCP隊頭阻塞

TCP 隊頭阻塞（ head of line blocking ）

HTTP/2 是基於 TCP 實作的。對比早期的 HTTP 版本，HTTP/2 使用的 TCP 連接數少了很多。 TCP 是一個可靠的通訊協定，基本上，你可以將它視為在兩台機器間建立的一個虛擬鏈，由其中一端放置於網路上的內容，最終總會以相同的順序出現在另一端。（ 或者遭遇連接中斷 ）

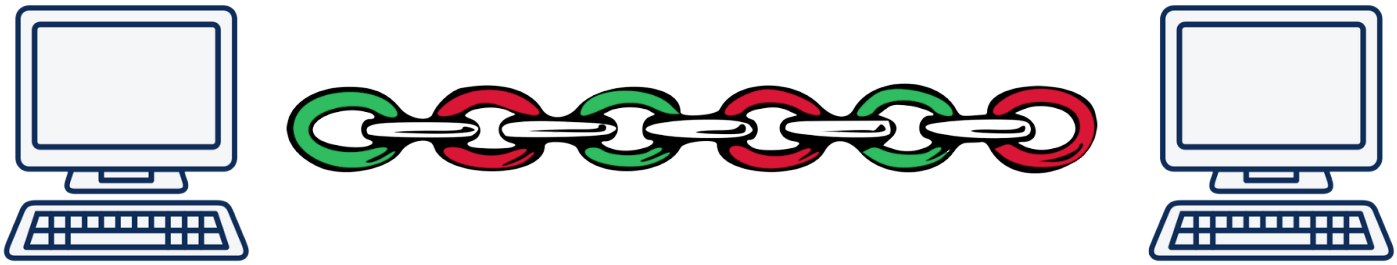


a TCP chain between two computers

採用 HTTP/2 時，典型的瀏覽器通過單個 TCP 連接進行數十或數百個並行傳輸。

如果 HTTP/2 連接雙方的網路中有一個封包丟失，或者任何一方的網路出現中斷，整個TCP連接就會暫停，丟失的封包需要被重新傳輸。 因為 TCP 是一個按序傳輸的鏈，因此，如果其中一個點丟失了，在該點之後的所有內容將陷入等待。

如下圖所示，我們用鏈條來表現一個連接上發送的兩個傳輸流，紅色與綠色的傳輸流：



the chain showing links in different colors

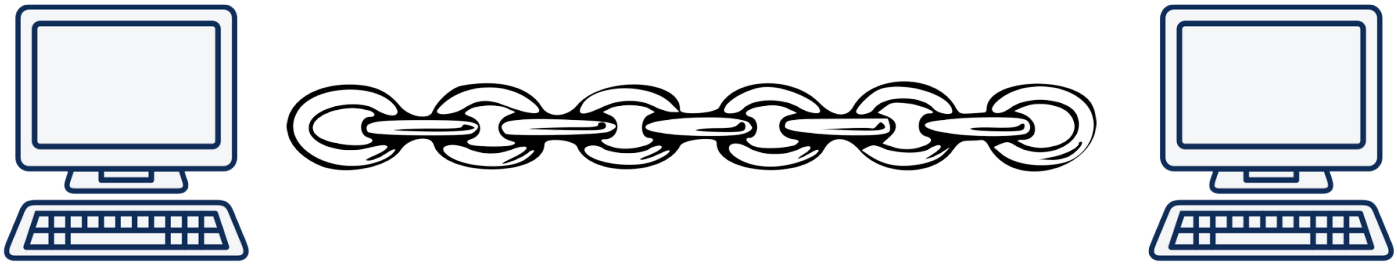
這種單一封包造成的阻塞，就是所謂 TCP 上的對頭阻塞（ head of line blocking ）！

隨著封包遺失機率的增加，HTTP/2 的表現將越來越差。在2%的封包丟失率下（ 一個很差的網路環境 ），測試結果顯示 HTTP/1 用戶的性能更好，因為 HTTP/1 一般有六個TCP連接能夠分配處理丟失的封包，就算其中一個連接阻塞了，其他連接仍然可以繼續進行傳輸。

即使有可能，使用TCP修復此問題也並非想像中容易。

獨立的資料串流避免阻塞

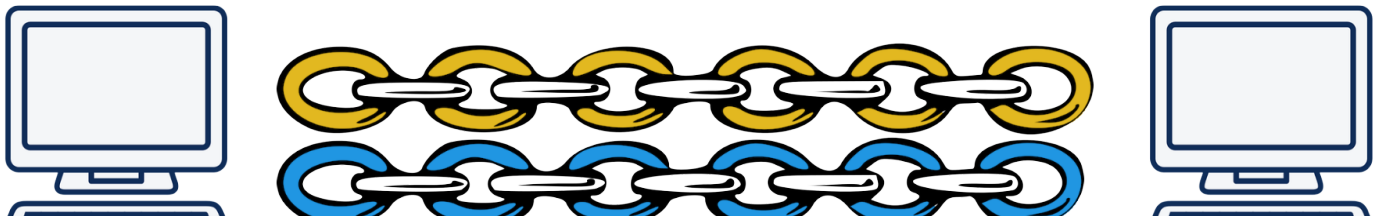
使用 QUIC 時，兩個終端間仍建立一個連接，該連接也經過協商使得資料得到安全且可靠的傳輸。



a QUIC chain between two computers

但是，當我們在這個連接上建立兩個不同的資料串流時，它們將被視為互相獨立。也就是說，如果其中一個資料串流遺失封包了，那只有該資料串流必須停下來，然後等待重傳。

下面為兩終端間的示意圖，黃色與藍色是兩個獨立的資料串流。



two QUIC streams between two computers

TCP還是UDP

TCP 還是 UDP

如果我們無法解決 TCP 中的隊頭阻塞問題，那麼從理論上來說，我們應該能夠在網路堆疊（network stack）中建立一個 UDP 和 TCP 之外的新傳輸協定。甚至使用 SCTP 這是 IETF 在 RFC 中標準化的傳輸協定 RFC4960 ，它也擁有我們需要的幾項特質。

但是近年來，由於在 Internet 上部署它們十分困難，幾乎完全停止了建立新傳輸協定的工作。 許多防火牆、NAT、路由器和其它中間設備（middle-boxes）阻礙了新協定的部署，這些防火牆和設備僅認得 TCP 或 UDP。 所以導入新的傳輸協定會使 N％ 的連接失效，因為它將會被這些設備所阻擋，因為它被視為不是 UDP 或 TCP（被視為不安全的協定）。 N％ 的失敗率被認為太高，不值得花費時間努力。

此外，更改網路堆疊的傳輸協定層中的內容通常表示該協定是由作業系統的核心所實作。更新和部署新的作業系統核心是一個緩慢的過程，需要大量的精力。由 IETF 標準化的許多 TCP 改進並未得到廣泛部署或使用，因為它們沒有得到廣泛支持。

為何不實作基於 UDP 的 SCTP

SCTP 是一個支持資料串流的可靠傳輸層協定，且在 WebRTC 上已有基於 UDP 的對它的實作。

但跟 QUIC 比較起來它還不夠好，原因如下：

- SCTP 沒有解決資料串流的隊頭阻塞問題
- SCTP 建立連接時就需要決定資料串流的數量
- SCTP 沒有穩固的 TLS / 安全性支持
- SCTP 建立連接時需要進行4-way handshake，QUIC 一次都不需要（0-RTT）
- QUIC 是類 TCP 的 bytestream，而 SCTP 是訊息流（message-based）
- QUIC 連接可以在 IP 位址間遷移，SCTP 不行

更多 SCTP 與 QUIC 的差異，請參見 A Comparison between SCTP and QUIC.

協定僵化

網際網路（Internet）是由多個網路互相相連而成。為了保證網際網路正常運作，我們需要在網際網路各處搭建各種設備。這些在網際網路上分佈式架設的設備被稱為中間設備（middle-box）。傳統網際網路傳輸中，兩個端點之間的中間設備肩負著網路資料的傳輸。

這些中間設備有很多，各式各樣的用途和目的，但我可以說，這些設備被放置在該位置是因為有人認為它一定要在那個位置才能讓網路的工作完成。

中間設備的綜合用途：在網路間轉送封包（路由）、阻擋惡意流量、網路位址轉換（NAT）、提升性能、監控流量...等等。

為了完成這些目的，這些設備必須了解網路以及它們所要監視或修改的協定。它們為此依賴於一些程式，一些很少更新或升級的程式。

儘管它們是使網路連在一起的關鍵元素，但它們通常也跟不上最新的技術。網路核心的更新速度通常不如世界上的客戶端或伺服器快。

所以當這些設備能夠決定經過的流量是否安全或合格時，就有問題了。在這些設備部署之後的一段時間裡，協定有了新的功能或更新。 而在這些設備引入（理解）這些新功能之前，它們會認為這種特性的封包是非法、惡意的，於是這些流量將直接被丟棄，或是拖延直到用戶不再想使用這些新的功能。

這種現象我們稱之為 "協定僵化"

TCP 的更改也容易僵化：客戶端和遠程伺服器之間的某些中間設備會發現未知的 TCP 選項並阻止此類連接，因為它們不知道這些選項是什麼。如果中間設備監測了協定的實作細節，它們會學習到協定的典型行為，一段時間後，這些行為將無法更改。

盡可能的將通信加密是對抗僵化的唯一有效手段，加密可以防止中間設備看到協定傳輸的絕大部分內容。

安全性

QUIC 保證安全性。該協定沒有明文版本，要建立一個 QUIC 連接，就必須通過 TLS 1.3 來安全地建立一個加密連接。 如上文所說，加密可以避免協定僵化，流量被攔截或被做其它處理。這讓 QUIC 具有了 Web 用戶所期望及需要的 HTTPS 安全特性。

QUIC 只在加密協定協商之前，有少數幾個以明文發送的初始握手封包。

減少延遲

對比 TCP 的 3 次交握，QUIC 提供了 0-RTT 和 1-RTT 的交握方式，這減少了協商和建立新連接時所需的時間。

除此之外，QUIC 能處理更多資料，它有提早傳輸更多資料的「早期資料」(early data ）特性，使用上比 TCP Fast Open 更加簡單。

導入串流的概念，客戶端不用等待前一個連接結束便能與同一主機建立另一個連接。

TCP Fast Open 是有問題的

TCP Fast Open 在2014年12月被提交為 RFC 7413 本規範描述了應用程式如何將資料傳遞到已傳遞第一個

TCP SYN 數據包的伺服器。

但在網際網路上，對這個選項的實際支援仍然需要時間，在 2018 年的今天，這個選項還是有許多問題。 想要在 TCP 堆疊上實現這個選項以獲得改進的工程師，不僅要注意作業系統的版本，還要注意發生問題時如何優雅地降級到原本狀態。 已知有多個網路正在積極破壞這種 TCP 握手，從而干擾了 TCP Fast Open 的流量。

協定的進展

最初的 QUIC 協定是 Google 的 Jim Roskind 所設計，首次實作是在 2012 年，在 Google 的擴大實驗後於 2013 年向世界公開。

最初，QUIC 為 "Quick UDP Internet Connections" 的縮寫，但現在已經不是。

Google 實施了該協定，隨後將其部署在廣泛使用的瀏覽器（ Chrome ）和廣泛使用的伺服器端服務（ Google搜索、gmail、youtube...等）中。 他們相當快地迭代了協定的版本，並且隨著時間的流逝，他們證明了該概念可以為絕大多數用戶可靠地工作。

2015 年 6 月，QUIC 的第一個網際網路草案被發送到 IETF 進行標準化，但直到2016年下半年，QUIC 工作組才獲得批准並開始。 隨後，它很快就引起了各方的高度關注並迅速發展。

在 2017 年，由 Google QUIC 工程師所引用的數字提到，大約 7％ 的網際網路流量已經在使用該協定（ Google版本)。

IETF

為標準化 IETF 中協定而成立的 QUIC 工作組很快就決定了，除了 HTTP 應用層協定以外，QUIC 還應該能夠支援傳輸其他協定。 Google 版的 QUIC 僅用於 HTTP - 但實際上使用了 HTTP/2 框架語法，並與 HTTP/2 兼容。

另外，還決定採用基於 TLS 1.3 的加密和安全性，而不是 Google 的 "自定義" 方法。

為了滿足不局限於 HTTP 的傳輸需求，IETF QUIC 協定的架構被分為兩個獨立的層： 傳輸層 QUIC 和 "基於 QUIC 的 HTTP"（ HTTP over QUIC ）層（ 有時縮寫為 "hq" ）。

乍看之下這種分層似乎無害，但在 IETF-QUIC 和 Google-QUIC 之間卻有很大的不同。

工作組很快就發現了這點，為保持適當關注和能按時交付第一版 QUIC，工作組將重心轉移到了 HTTP 傳輸，非 HTTP 的傳輸將留待今後研究。

在 2018 年 3 月，當我們開始編寫本書時，工作組計劃是在 2018 年 11 月發布 QUIC 第一版的最終規範，不過目前已經延遲了多次，在撰寫本文時（ 2020 年 6 月 ）工作組正在進入最終定稿階段。

隨著有關 IETF-QUIC 的工作的進行，Google 團隊已經整合了 IETF 版本的細節並逐漸推進他們的協定版本，以便能符合 IETF 定義的規範。 Google 繼續在其瀏覽器和服務上運行 Google 版本的 QUIC。

正在開發的大多數新實作 已決定專注於 IETF 版本，並且與Google 版本不兼容。

從HTTP/2得到的經驗

HTTP/2 的規範 RFC 7540 發布於 2015 年 5 月，在 QUIC 首次引入 IETF 的前一個月。

HTTP/2 的發布為 HTTP 進化提供了好的經驗。 這一經驗讓大家相信，迭代更新的 HTTP 版本會比從第一版到第二版（ 花費約16年 ）快得多。

使用 HTTP/2，用戶和軟體堆疊習慣了這樣的想法，不再假設 HTTP 是一個序列化的、基於文本的協定。

QUIC 之上 的 HTTP （ HTTP/3 ）建立在 HTTP/2 基礎之上，並遵循許多相同的概念，不過它們從 HTTP 中移走了一些已經被 QUIC 所包含的規範。

HTTP-over-QUIC 於 2018 年 11 月更名為 HTTP/3。

目前的狀況

QUIC 工作組自 2016 年末以來一直在努力製定協定，在撰寫本文時（ 2020 年 6 月 ）已接近最後階段。

在 2019 年和 2020 年間，使用 HTTP/3 進行互操作性測試的數量不斷增加 [https://docs.google.com/spreadsheets/d/1D0tW89vOoaScs3IY9RGC0UesWGAwE6xyLk0l4JtvTVg/edit#gid=1268516408] CDN 和瀏覽器已經開始提供初始的支持 - 儘管它們通常會進度落後。

QUIC 工作組的 Wiki 頁面上有更多的資訊。 QUIC implementations listed

實現 QUIC 並不容易，到目前為止，協定本身還在不斷演變中。

伺服器端

NGINX 對 QUIC 和 HTTP/3 的支援正在開發中，並且 已發布預覽版.

至於 Apache 對 QUIC 的支援則是還沒有公開的聲明。

客戶端

尚未有任何主流瀏覽器供應商在任何狀態下提供可以運行 IETF 版本的 QUIC 或 HTTP/3 協定。

多年來，Google 瀏覽器一直在提供 Google 自己的 QUIC 版本，並且最近已經開始支持 IETF 版本。 Firefox 同樣之後會開始支援。

curl 在 2019 年 9 月 11 日發布了 7.66.0 版本中的第一個實驗性的 HTTP/3 支援（draft-22）。 curl 使用 Cloudflare 的 Quiche 或 ngtcp2 來完成工作。

實作的障礙

QUIC 決定使用 TLS 1.3 作為加密和安全層的基礎來避免發明新的東西，且依靠可信賴的現有協定。 不過工作組決定大幅精簡 QUIC 中 TLS 的使用，只使用 "TLS訊息"（ TLS Messages ）而不是協定中的 "TLS記錄"（ TLS Records ）。

這聽起來像是一個無害的更改，但實際上已對許多 QUIC 堆疊的實作者造成了重大障礙。 現存的支持 TLS 1.3 的 TLS libraries 都沒有提供此功能的 API 並允許 QUIC 訪問它。 有一些 QUIC 的實作由大型機構完成，這些機構可能有自己的 TLS 堆疊，但並不是所有人都能做到這種地步。

例如，占主導地位的開源重量級 OpenSSL 對此沒有任何 API。 解決此問題的計劃可能以在他們的 PR8797 中，它們計畫一種與 BoringSSL 非常相似的 API。

由於 QUIC 堆疊將需要基於其他 TLS libraries，使用單獨的修補 OpenSSL 構建或需要更新到將來的 OpenSSL 版本，這些最終也將導致部署障礙。

作業系统核心、CPU 負載

Google 和 Facebook 都提到，他們的 QUIC 大規模部署所需的 CPU 數量大約是通過 TLS 提供 HTTP/2 時相同流量負載的兩倍。

進一步的解釋

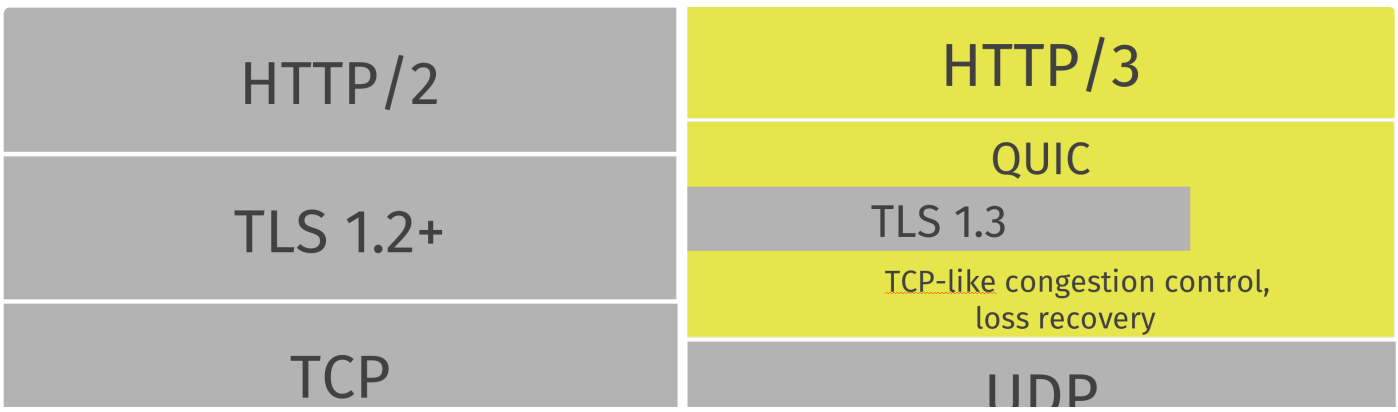
- 主要是 Linux 中的 UDP 部分根本沒有像 TCP 堆疊那樣優化，因為傳統上它沒有像這樣被用於高速傳輸。
- TCP 和 TLS 有硬體加速（負載卸載到硬體，offload），而這對於 UDP 很罕見，對於 QUIC 則基本上不存在。

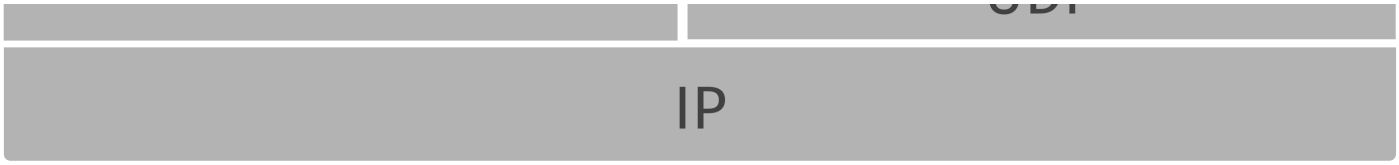
就上述理由，我們可以相信 QUIC 的 CPU 使用量能隨著時間得到改善。

協定的功能

本章節將從高層次出發概述 QUIC 協定。

下圖顯示了用作 HTTP 傳輸時，左側的 HTTP/2 網路堆疊和右側的 QUIC 網路堆疊的對比。





HTTP/3 over QUIC stack overview

基於UDP的傳輸層協定

基於 UDP 的傳輸層協定

QUIC 是在 UDP 用戶空間中實現的傳輸協定。如果快速瀏覽一下網絡流量，QUIC 將看起來像一個 UDP 封包。

正如在 UDP 上實現的那樣，QUIC 還利用 UDP 端口號來標識給定 IP 地址上的特定服務。

目前已知的 QUIC 實作都位於用戶空間，這使它能得到更快速的迭代（對比內核空間中的實作）。

能跑得動嗎？

有一些網路上的中間設備會攔截 port 53（用於DNS）以外的 UDP 流量。還有一些網路會節流（throttle）UDP 流量，使得 QUIC 的表現慢於基於 TCP 的協定。更多的情況是我們不知道運營商會怎麼做。

在可預見的將來，所有基於 QUIC 的傳輸的使用都可能必須能夠優雅地回退到另一個（基於TCP）的替代方法。 Google 工程師以前曾提到過，故障率可能會是百分比的個位數。

會被優化嗎？

如果 QUIC 被證明確實是網際網路的一個有益補充，客戶會希望能正常使用 QUIC，那麼網路公司就可能重新解決上述的問題。 多年以來，隨著 QUIC 取得進展，在網際網路上建立和使用 QUIC 的成功率已有所提高。

高度可靠的傳輸

雖然 UDP 不提供可靠的傳輸，但 QUIC 在基於 UDP 時增加了一層帶來可靠性的層。 它提供了封包重傳，擁塞控制，調整傳輸節奏（pacing）及其他一些 TCP 中存在的特性。

只要連接沒有中斷，從 QUIC 一端傳輸的資料都會出現在另一端，只是時間早或晚的問題。

串流

與 SCTP、SSH 和 HTTP/2 相似，QUIC 在物理連接內具有獨立的邏輯流。 許多並行流可以透過單個連接同時傳輸資料，而不會影響其他流。

連接兩個端點之間的協商設置是類似於 TCP 連接的工作方式。 QUIC 連接建立到 UDP 端口和 IP 地址，但是一旦建立，該連接就由其 "連接ID"（ connection ID ）關聯。

通過已建立的連接，任何一方都可以創建流並將資料發送到另一端。 單一串流的傳輸是可靠、有序的，但不同的串流間可能會有無序傳送。

QUIC 可對連接和串流分別進行流量控制（ flow control ）。

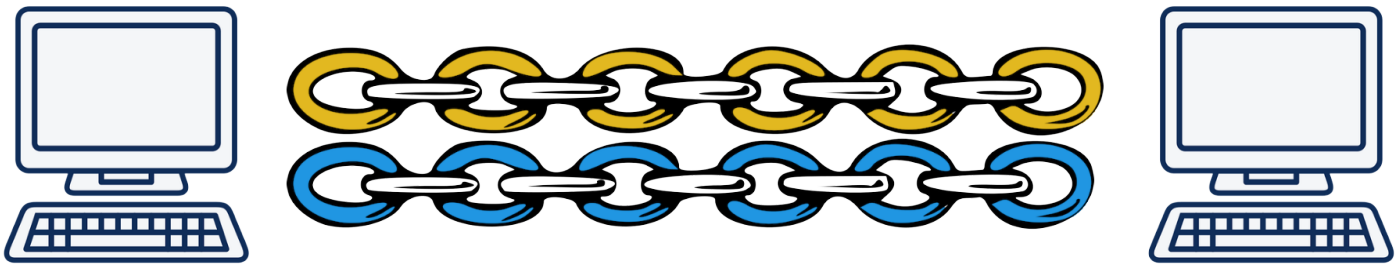
進一步細節請參考 連接 和 QUIC串流。

有序交付

QUIC 保證串流的有序交付，但不能保證各個串流之間的有序交付。 這代表每個串流將發送數據並保持數據順序（ 串流中的數據將有序抵達 ），但是每個串流之間可能以不同的順序到達目的地！

舉個例子： 串流 A 和串流 B 從伺服器端傳輸到客戶端。串流 A 先出發，然後是串流 B。在 QUIC 中，遺失的封包僅影響該遺失的封包所屬的串流。所以當串流 A 遺失了封包但串流 B 沒有的情況下，串流 B 會持續傳輸並且完成，而串流 A 的遺失封包將重新傳輸。這種狀況不會發生在 HTTP/2。

此處通過一個連接在兩個 QUIC 端點之間發送了一個黃色和一個藍色的串流進行說明。 它們是獨立的，並且可能以不同的順序到達，但是每個串流中的封包都按順序可靠地傳遞給應用程序。



兩個端點之間的兩個QUIC串流

快速交握

QUIC 提供 0-RTT 和 1-RTT 的連接建立，這意味著 QUIC 在最佳情況下不需要任何額外往返時間便可以建立新連接。 兩者中較快的 0-RTT 僅在兩主機之間建立過連接且緩存了該連接的 "秘密"（ secret ）時才能使用。

早期資料（Early data）

QUIC 允許客戶端在 0-RTT 的情況下直接攜帶資料。這使得客戶端能儘早向對方傳送資料，當然也使得伺服

器端能更快地發回響應。

TLS 1.3

QUIC 中使用的傳輸層安全協定是 TLS 1.3 (RFC8446) QUIC 沒有任何非加密的連線。

與更早的 TLS 版本相比，TLS 1.3 有許多優點，但使用它的最主要原因是其握手所花費的往返次數更低，從而能降低協定的延遲。

Google 舊版的 QUIC 使用一個自定義的加密法。

傳輸層與應用層協定

IETF 的 QUIC 是一個傳輸層協定，在該協定之上可以運行其他應用層協定。初始的應用層協定是 HTTP/3（h3）。

傳輸層協定負責連接和串流的處理。

在Google 舊版 QUIC 中，已將傳輸和 HTTP 粘合在一起，成為一個單一的 "do-it-all" 工具。 它的用途更加特殊，是一個更有指向性的 send-http/2-frames-over-udp 的協定。

HTTP/3 over QUIC

HTTP 層稱為 HTTP/3，執行 HTTP 風格的傳輸，包括使用 QPACK 進行 HTTP 標頭壓縮 - 這和 HTTP/2 中使用 HPACK 壓縮標頭類似。

HPACK 算法依賴於串流的有序交付，由於 HTTP/3 的串流之間可能亂序，所以該算法需要修改才能使用。QPACK 可被視為適用於 QUIC 版本的 HPACK.

Non-HTTP over QUIC

基於 QUIC 傳輸非 HTTP 協定的相關工作已被延後到第一版 QUIC 發布後實作。

QUIC運作原理

本章節將解釋 QUIC 傳輸層協定各基本模組的功能，但不會逐字解釋協定。 如果您想自己實現 QUIC，這些介紹會加強你對協定的理解，但有關具體細節，請參考 IETF 的網際網路草案和 RFC。

1. 建立一個 連接
2. TLS 安全性
3.

3. 使用 串流(streams)

連接

QUIC 連接是兩個 QUIC 端點之間的單次對話（ conversation ）過程。

QUIC 建立連接時，加密算法的版本協商與傳輸層交握被合併完成以減小建立連接時的延遲。

為了通過這樣的連接實際發送資料，它必須創建和使用一個或多個串流。

連接ID (Connection ID)

每個連接過程都有一組連接標識符，或稱連接ID，該ID用以識別該連接。連接ID由端點獨立選擇，每個端點都選擇其對等方使用的連接ID。

這些連接ID的主要功能是確保較低協定層（ UDP，IP及其底層協定 ）的地址更改不會導致 QUIC 連接的封包傳遞到錯誤的端點。

通過利用連接ID，連接可以以 TCP 永遠無法實現的方式在 IP 地址和網路接口之間遷移。 例如，當客戶端裝置移動到一個有 Wifi 網路的環境時，將正在下載的項目從蜂巢式網路轉已到更快的 Wifi 連接。 同樣，如果 Wifi 不可用時，下載可以繼續通過蜂窩連接進行。

端口編號 (Port numbers)

QUIC 建立在 UDP 之上，因此使用16位元的端口號字段來區分傳入的連接。

版本協商 (Version negotiation)

源自客戶端的 QUIC 連接請求將告訴伺服器端它要使用哪個版本的 QUIC 協定，服務器將響應回覆一個列表包含所支援的版本來供客戶端選擇。

使用TLS的連接

在初始封包建立連接後，發起方立即發送一個加密frame，開始建立安全層交握。安全層使用 TLS 1.3 協定。

在 QUIC 中，沒有方法或機制避免使用 TLS 連接。該設計是為了使中間設備難以篡改封包，防止協定僵化。

QUIC串流

在 QUIC 中的資料串流（ Streams ）提供了一個輕量級，有序的字節流的抽象化。

QUIC 中有兩種基本的資料串流類型：

- 單向資料串流: 從發起者到對等端（ Peer ）的單向資料串流。
- 雙向均可發出資料的雙向資料串流。

連接端點的任意一方都可以建立這兩種資料串流，資料串流之間可並行、交錯地傳輸，並且可以被取消。

通過 QUIC 發送資料將建立一個或多個資料串流。

流量控制 (Flow control)

每個資料串流都有獨立的流量控制，端點可以通過此實現內存控制和反壓（ back pressure ）。資料串流的創建本身也受到流量控制，連接雙方可以聲明最多願意接受多少數量的資料串流ID。

資料串流識別符 (Stream Identifiers)

資料串流由一組無符號62位元的整數標示，稱為資料串流ID。資料串流ID的最低2位元用於標示資料串流類型（ 單向或雙向 ）和資料串流啟動器。

資料串流ID的最低位元（ 0x1 ）標示資料串流的發起者。客戶端發起為偶數（ 設置為0 ）資料串流，伺服器端發起為奇數（ 設置為1 ）資料串流。

第2個位元（ 0x2 ）用於識別單/雙向資料串流。單向資料串流始終為1，雙向資料串流則為0。

資料串流的並行性 (Stream concurrency)

QUIC 允許任意數量的資料串流同時運行。端點通過閒置最大資料串流ID來控制並行的傳入資料串流數量。

每個端點指定自己的最大資料串流ID數，並只對對等端端點有效。

傳送和接收資料 (Sending and Receiving Data)

端點使用資料串流來發送和接收資料。這是資料串流的最終目的。 QUIC 資料串流是有序的字節流抽象。但是不同資料串流之間是無序的。

資料串流的優先順位 (Stream Prioritization)

如果正確設置了各資料串流的優先度，流復用機制可以顯著提升應用的效率。使用其它多路復用協定（如 HTTP/2）的經驗表明，有效的優先度劃分策略對效率具有顯著的正面影響。

QUIC 本身沒有交換優先級訊息的框架。 相反，它信任來自使用 QUIC 的應用程序的優先級訊息。利用 QUIC 的協定可以定義與它們的應用程序語義相匹配的優先級分配方案。

有許多對 HTTP/2 優先級排序模型的批評，並且擔心它過於復雜，並且沒有被許多 HTTP/2 伺服器端使用和實現。 目前，HTTP/3 中的優先級已從主要的 HTTP/3 規範中刪除，且正被做成一個 separate specification.

0-RTT

為了減少建立新連接所需的時間，曾連接到該伺服器端的客戶端會緩存某些參數，並使用這些參數與伺服器端建立 **0-RTT** 連接。 這樣，客戶端可以立即發送資料，而不必等待交握的動作完成。

Spin Bit

在 QUIC 工作組的設計討論中，最長的主題之一就是 Spin Bit，人們花費了數百封郵件和數百個小時來討論它。

Spin Bit 的支持者認為，兩個 QUIC 端點之間路徑上的運營商和人員需要有辦法來測量延遲。

反對者則反感此功能潛在的訊息洩露。

Spinning a bit

客戶端和伺服器端這兩個端點都為每個 QUIC 連接維護旋轉值0或1，並且將為該連接發送的封包上的旋轉位設置為適當的值。

然後，在每一次往返時，連接雙方都翻轉這一 bit 的值。效果就是觀察者可以監視的那個位域中的1和0的脈衝。

僅當發送人不受應用程序或流控制的限制時，此觀察才有效，並且在網路上對封包進行排序可能會使封包意外出現。

用户空間

在用戶空間中實作傳輸協定有助於協定的快速迭代，因為相對容易地發展協定而無需客戶端和伺服器端更新其作業系統核心以部署新版本。

QUIC 本身沒有固有的東西阻礙未來在作業系統核新中實現和提供 QUIC 協定。

更多的實作 (Many implementations)

在用戶空間中實作一個新的傳輸層協定時，一個顯而易見的效果是我們會看到很多獨立的實作。

在可預見的未來，不同的應用程序可能包含（或基於）不同的 HTTP/3 和 QUIC 實作。

API

常規 TCP 和使用該協定的程式成功的因素之一是標準化的 socket API。 它具有定義明確的功能，使用此 API，您可以在許多不同的作業系統之間移動程式，因為 TCP 的工作原理相同。

但 QUIC 並不是。 QUIC 目前沒有標準化的 API。

使用 QUIC 時，你需要選擇一個現有的程式庫來實作，並堅持使用它的 API。這在某種程度上把應用 "綁定" 到了單一的庫上。換庫代表著使用另外一套 API，這可能帶來相當的工作量。

另外，由於 QUIC 一般在用戶空間中實作，所以它不像現有的 TCP 和 UDP socket API 那樣能輕鬆擴展。 使用 QUIC 代表著選擇了 socket API 之外的另一套 API。

HTTP/3

如前所述，通過 QUIC 傳輸的第一個也是主要的協定是 HTTP。

就像曾經引入 HTTP/2 以全新的方式通過有線傳輸 HTTP 一樣，HTTP/3 是另一種通過網路傳輸 HTTP 的新方法。

HTTP 仍然保持與以前相同的範例和概念。 有標頭（header）和主體 (body)，有請求 (request) 和響應 (response)。 有動詞，Cookie 和緩存。 HTTP/3 的主要變化是如何將這些以位元傳送到通信的另一端。

為了在 QUIC 上進行 HTTP，協定需要進行更改，其結果就是我們現在所說的 HTTP/3。 由於 QUIC 提供的特性與 TCP 不同，因此需要進行這些更改。 這些更改包括：

- 在 QUIC 中，串流由傳輸器本身提供，而在 HTTP/2 中，串流在 HTTP 層內完成。
- 由於串流彼此獨立， HTTP/2 中使用的頭部壓縮算法如果不做改動，會造成隊頭阻塞。
- QUIC 串流與 HTTP/2 串流略有不同，本書的 HTTP/3 章節會做詳細介紹。

HTTPS:// URL

HTTP/ 3 將使用 `HTTPS://` URL 執行。世界上到處都是這些 URL，為新協定導入另一種 URL 方案被認為是不切實際和完全不合理的。 就像 HTTP/2 不需要新的方案一樣，HTTP/3 也不需要。

HTTP/2 是傳輸 HTTP 的一種新方式，但是它和 HTTP/1 一樣還是基於 TLS 和 TCP。 而在基於 QUIC 的 HTTP/3 中，情況更加複雜，它在一些重要的地方做了一些改變。

過去遺留下來的明文 `HTTP://` URL 的處理方式將保持原樣，隨著我們邁入安全傳輸更加普及的未來，它的使用可能會越來越少。 對這類 HTTP URL 的請求將不會升級為 HTTP/3。實際上，由於其它因素，過去它們也很少被升級到 HTTP/2。

初始連接 (Initial connection)

當嘗試連接到一個全新的、未訪問過的網站時，到 HTTPS:// URL 的連接可能必須通過 TCP （也許會有平行的一個 QUIC 連接）。 因為主機可能是一個不支持 QUIC 的傳統伺服器，或者鏈路之間可能有阻礙 QUIC 成功連接的中間設備。

現代的瀏覽器和伺服器可能在首次握手時協商 HTTP/2 協定。 在連接建立並且伺服器回應客戶端的 HTTP 請求時，伺服器可以通告客戶端它對 HTTP/3 的支援與偏好。

使用Alt-svc進行引導

替代服務（alternative service, Alt-svc:）標頭和它相對應的 `ALT-SVC` HTTP/2 frame 並不是特別為 QUIC 和 HTTP/3 所設計的。 它是為了讓伺服器端能告訴客戶端 *"看，我在這個主機的這個端口用此協定提供相同的服務"* 。 詳見RFC 7838。

收到這樣的 Alt-svc 回應後，如果客戶端支援並希望使用其他協定，則它將在後台使用指定的協定與主機進行並行連接，並且連接成功。 如果是這樣，它將切換到該協定，而不是使用第一個連接。

如果第一個連接使用 HTTP/2 或 HTTP/1，則伺服器端可以告訴客戶端可以通過 HTTP/3 重新連接。 它可以用於同一主機或是不同但提供相同服務的主機。

Alt-svc 回應中有一個到期計時器，讓客戶端可以在指定的時間內使用建議的替代協定後將後續連接和請求直接發送給替代主機。

範例

一個帶有 `Alt-Svc:` 標頭的 HTTP 伺服器回應：

```
1 Alt-Svc: h3=":50781"
```

這指出了同一名稱的主機在 UDP 端口50781能提供 HTTP/3 服務。

客戶端可以嘗試與該端口建立 QUIC 連接。如果成功，後續將通過該連接繼續通訊，代替一開始初始的 HTTP 版本。

QUIC串流與HTTP/3

HTTP/3 是為 QUIC 設計的，因此它能利用 QUIC 的好處，而 HTTP/2 不得不在 TCP 上建構它的串流和復用

概念。
通過 HTTP/3 傳輸的 HTTP 請求是使用一系列特定的串流完成。

HTTP/3 frames

HTTP/3 代表著建立 QUIC 串流，並將一系列 frame 發送給對方。 HTTP/3 中的數據 frame 種類不多且固定（截至 2018 年 12 月 18 日有九種）。其中最重要的 frame 是：

- HEADERS, 發送壓縮的 HTTP 標頭
- DATA, 發送二進制資料內容
- GOAWAY, 請關閉此連接

HTTP Request

客戶端通過其發起的 *雙向* QUIC 串流來發送 HTTP 請求。

一個請求由單個 HEADERS frame 組成，並可以選擇後面跟隨一個或兩個 frame： 一系列的 DATA frame，以及可能有一個作為末尾的 HEADERS frame。

發送一個請求後，客戶端會關閉該串流以進行發出。

HTTP Response

伺服器端在雙向串流上發回其 HTTP 回應。其中含有一個 HEADERS frame，一系列 DATA frame，末尾可能有一個 HEADERS frame。

QPACK headers

HEADERS 含有用 QPACK 算法壓縮的 HTTP 標頭。 QPACK 與 HTTP/2 中的 HPACK (RFC7541) 類似，並針對亂序串流做了對應的修改。

QPACK 本身在兩個端點之間使用了兩個附加的單向 QUIC 流。 它們用於在任一方向上傳送動態表訊息。

優先次序

如前所述，串流之間的優先級已從主要的 HTTP/3 規範中刪除，之後會分開處理。

這是從 HTTP/2 優先級排序模型中學到的東西，以及它在現實世界中的實現（或缺少）。

一個比HTTP/2更簡單的優先級模型 使用 HTTP 標頭字段和有限數量的優先級設置。這是對 HTTP/2 標頭 frame 中的 "相依關係"和"權重"標誌的關鍵更改，並使其在應用程序層能更好地被理解。

是否支援重定優先級 (Reprioritisation) 仍在討論中。 HTTP/2 具有優先級框架來處理此問題，但是 QUIC 和 HTTP/3 中串流真正的獨立使這一問題變得更加複雜，因此，此項目仍在討論中。

當 （或如果 ）為 HTTP/3 商定了更好的優先級模型時，希望也能夠將該模型反向移植到 HTTP/2，以解決那裡的複雜性和實現問題。

伺服器推送

HTTP/3 的伺服器端推送與 HTTP/2 (RFC7540) 類似，但機制上有所不同。

伺服器端推送實際上是對客戶端從未發送過的請求的回應！

伺服器端推送僅在客戶端同意的前提下才允許發出。在 HTTP/3 中，客戶端甚至能通過告知伺服器端的最大推送流ID來設置所接受推送的次數限制。超出限制將導致連接錯誤。

如果伺服器端認為客戶端可能需要某個並未要求但應該有的額外資源，伺服器端可以通過請求流發送一個 `PUSH_PROMISE` frame，使該推送請求看上去像是一個回應，然後通過新的串流發送實際回應。

即便客戶端之前已經表示過可接受推送，但如果客戶端仍然可以隨時取消每個推送串流，然後發送一個 `CANCEL_PUSH` frame 到伺服器端。

Problematic

自從推送這一特性在 HTTP/2 中討論、開發、部署以來，它就備受爭議、討論和抨擊。為了讓它有用，人們付出了許多努力。

推送從來不是 "免費" 的，儘管它省了半個往返的延遲，它還是會消耗帶寬。伺服器也很難或不可能從高層面確定一個資源是否應該被推送過去。

與HTTP/2的比較

HTTP/3 是為 QUIC 設計的，QUIC 是一種能自行處理資料串流的傳輸層協定。

HTTP/2 是為 TCP 設計的，因此資料串流在 HTTP 層處理。

相似處 (Similarities)

這兩種協定為客戶端提供了幾個幾乎相同的功能集。

-

兩者都提供伺服器端推送
- 兩種協定都具有標頭壓縮，並且 QPACK 和 HPACK 在設計上相似
- 兩種協定都在單個連接上提供串流多路復用。

不同處 (Differences)

區別主要在細節上，並且歸因於 HTTP/3 對 QUIC 的使用:

- 得益於 QUIC 的 0-RTT 握手，HTTP/3 可以提供更好的 early data 支援，而 TCP Fast Open 和 TLS 通常只能傳輸更少的數據，且經常存在問題。
- 得益於 QUIC，HTTP/3 的握手速度比 TCP+TLS 快得多。
- HTTP/3 不存在明文的不安全版本。儘管在網際網路上很少見，HTTP/2 還是可以不配合 HTTPS 來實現和使用。
- 通過 ALPN 拓展，HTTP/2 可以直接在 TLS 握手時進行協商。 HTTP/3 基於 QUIC，所以需要憑藉回應中的 `Alt-Svc:` 標頭來向客戶端宣告。
- HTTP/3 不提供優先級控制。 計劃用於 HTTP/3 的 HTTP/2 優先級控制方法被認為過於復雜或根本無法適用，目前正在努力創建一種更簡單的機制。 當前計劃的更簡單的機制是向後移植到 HTTP/2，並利用 HTTP/2 的擴展規範優先級控制。

常見的錯誤批評

UDP 在許多公司和組織中無法作用

許多企業，運營商和組織都在端口 53 （ 用於DNS ） 之外阻止或限制UDP流量，因為此種流量常常被濫用於攻擊。 尤其是一些現有的 UDP 協定和實作容易受放大攻擊 （ amplification attack ） 的威脅，攻擊者可以控制無辜的主機向受害者投放發送大量的流量。

QUIC 內置了對放大攻擊的緩解處理。它要求初始封包不能小於 1200 字節，並且協定中限制，伺服器端在未收到客戶端回覆的情況下，不能發送超過請求大小三倍的響應內容。

UDP 在內核中很慢

這似乎是個事實，至少在最初的時候是。 我們當然不能說這將如何發展，以及其中有多少僅僅是由於 UDP 傳輸性能多年來未引起開發人員關注的結果。

對於大多數客戶端來說，這個程度的 "緩慢" 從未被覺察到。

QUIC 使用過多的 CPU

與上述 "UDP慢" 類似，部分原因是世界上 TCP 和 TLS 的使用已經有較長的時間來成熟，改進和獲得硬體的幫助，造成 UDP 看上去比較慢。

隨著時間的推移這種情況會有所改善，實際上 我們正在這個領域看到一些改進. 問題在於，這額外的 CPU 佔用部署會對部署人員帶來多大的影響。

只有 Google 在做

事實並非如此。Google 通過大規模的部署證明，通過 UDP 部署這種協定可以正常運行且表現良好，這是 IETF 帶來了初始的規範。

在那之後，很多公司和組織的人員都在這個利益方中立的 IETF 組織下推進標準化。在這個階段，雖然 Google 的員工也有參與，但 Mozilla、Fastly、Cloudflare、Akamai、Microsoft、Facebook、Apple 等等很多公司的員工也參與進來，共同推進網際網路的傳輸層協定。

進步太小

這是一個觀點而非批評。也許進步很小，這可能與相距 HTTP/2 的發布時間點很近有密切關係，時間距離太短了。

HTTP/3 在容易遺失封包的網路中可能表現更好，它提供了更快的交握，所以能改善可感知和實際的延遲。這些進步足夠推動人們在伺服器端和服務上部署 HTTP/3 的支援嗎? 時間以及未來的性能測試將會給我們答案！

技術標準

下列為 QUIC 和 HTTP/3 各部分和組件的最新官方草案列表。

不變性 (Invariants)

Version-Independent Properties of QUIC

傳輸層 (Transport)

QUIC: A UDP-Based Multiplexed and Secure Transport

自我修復 (Recovery)

QUIC Loss Detection and Congestion Control

TLS

Using TLS to Secure QUIC

HTTP

Hypertext Transfer Protocol Version 3 (HTTP/3)

QPACK

QPACK: Header Compression for HTTP/3

QUIC v2

為了盡力專注於 QUIC 的核心特性，以及為了趕上發布進度，最初計劃為核心協定一部分的幾個特性已被延後，延到 QUIC 第二版或以後的版本中完成。

本文檔作者沒有正版的水晶球在手上，所以我們不能知道哪些特性會或者不會出現在第二版中。 但我們可以談談那些在第一版中被標記為 "之後做" 的可能出現在第二版中的特性。

前向糾錯 (Forward Error Correction)

前向糾錯 （ FEC ） 是一種通過向接收方發送冗餘數據，使得接收方能快速識別出含有不明顯錯誤的一種錯誤控制方式。

Google 在它們的原版 QUIC 成果中實驗過該特性，但由於實驗結果不理想而已經去除。此特性可供 QUIC v2 討論，但可能需要有人證明它能在成本不大的情況下有效用。

Multipath

Multipath 代表著通訊可以通過多個網路路徑傳輸，以其最大化利用資源並增加冗餘。

SCTP 支持者可能有話說了，SCTP 和現代 TCP 早就已經支援這一點。

資料不可靠 (Unreliable data)

已經討論了提供 "不可靠"串流作為選項之一，然後允許 QUIC 替換 UDP-style 的應用程式。

Non-HTTP adaptions

基於 QUIC 的 DNS 是早期提到的非 HTTP 協定之一，一旦 QUIC v1 和 HTTP/3發布，它可能會引起注意。

但是，就像曾經新的交通工具被帶到世界上，因此我無法想像它會是終點。