Workgroup: Network Working Group
Internet-Draft:
draft-kuehlewind-masque-quic-substrate-00
Published: 9 March 2020
Intended Status: Informational
Expires: 10 September 2020
A   M. Kuehlewind   Z. Sarker   T. Fossati   L. Pardue
uEricsson   Ericsson   Arm   Cloudflare
t
h
o
r
s
:

Use Cases and Requirements for QUIC as a Substrate

Abstract

In situations where direct connectivity is not available or desired, proxies in the network are used to forward and potentially translate traffic. TCP is often used as a proxying or tunneling protocol. QUIC is a new, emerging transport protocol and there is a similar expectation that it too will be used as a substrate once it is widely deployed. Using QUIC instead of TCP in existing scenarios will allow proxying and tunneling services to maintain the benefits of QUIC natively, without degrading the performance and security characteristics. QUIC also opens up new opportunities for these services to have lower latency and better multistreaming support. This document summarizes current and future usage scenarios to derive requirements for QUIC as a substrate and to provide additional considerations for proxy signaling and control protocol as proposed by MASQUE.

Table of Contents

1. Introduction

QUIC is a new transport protocol that was developed with a focus on optimizing HTTP traffic by supporting multiplexing without head-of-line-blocking and integrating security directly into the transport. This tight integration of security allows the transport and security handshakes to be combined into a single round-trip exchange, after which both the transport connection and authenticated encryption keys are ready.

Based on the expectation that QUIC will be widely used for HTTP, it follows that there will also be a need to enable the use of QUIC for HTTP proxy services.

Beyond HTTP, however, QUIC provides a general-purpose transport protocol that can be used for many other kinds of traffic, whenever the features provided by QUIC (compared to existing options, like TCP) are beneficial to the high-layer service. Specifically, QUIC's ability to multiplex, encrypt data, and migrate between network paths makes it ideal for solutions that needs to tunnel or proxy traffic.

Existing proxies that are not based on QUIC are often transparent. That is, they do not require the cooperation of the ultimate connection endpoints, and are often not visible to one or both of the endpoints. If QUIC provides the basis for future tunneling and proxying solutions, it is expected that this relationship will change. At least one of the endpoints will be aware of the proxy and explicitly coordinate with it. This allows client hosts to make explicit decisions about the services they request from proxies (for example, simple forwarding or more advance, e.g. performance-optimizing, services), and to do so using a secure communication channel between themselves and the proxy.

MASQUE [I-D.schinazi-masque] is a proposed framework that allows running multiple network or application services inside one QUIC connection to be forwarded to one or more target servers. The end-to-end traffic between the client and the target server will be tunnelled in a (outer) QUIC connection between the client and the MASQUE server. This outer connection can also be used to securely exchange additional signal or control information between the MASQUE server and the client.

This document describes some of the use cases for using QUIC for proxying and tunneling, as proposed by MASQUE, and explains the protocol impacts and tradeoffs of such deployments.

2. Usage Scenarios

2.1. Obfuscation via Tunneling

Tunnels are used in many scenarios within the core of the network from a client endpoint to a proxy middlepoint on the way towards the server. In many cases, when the client explicitly decides to use the support of a proxy in order to connect to a server, it does so because a direct connection may be blocked or impaired. This can either be the case in e.g. enterprise network where traffic is firewalled and web traffic needs to be routed over an explicitly provided HTTP proxy, or other reasons for blocking of certain services e.g. due to censorship, data exfiltration protection, etc.

Tunneling through a proxy server can provide various benefits, particularly when using a proxy that has a secure multiplexed channel like QUIC:

* Obfuscating the traffic patterns of the traffic from the perspective of observers between the client and the proxy. If the content of connections to many end servers can be coalesced as one flow, it becomes increasingly difficult for observers to detect how many inner connections are being used, or what the content of those connections are.

* Obfuscating the client's IP address from the perspective of observers after the proxy, to the end server itself. This allows the client to reduce information leaked about its actual location, improving privacy.

* Obfuscating the end server's IP address from the observers between the client and the proxy, which protects the identity of a private server's address or circumvents local firewall rules.

In any of these tunneling scenarios, including those deployed today, the client explicitly decides to make use of a proxy service while it is usually fully transparent for the server, or even with the intention to hide the client's identity from the server. This is explicitly part of the design as these services are targeting an impaired or otherwise constrained network setup.

Therefore, in this usage scenario the client knows the proxy's address and explicitly selects to connect to the proxy in order to instruct the proxy to forward its traffic to a specific target server. Often the proxy is also preconfigured to "know" the client and therefore the client needs to authenticate itself (e.g. using HTTP Transport Authentication [I-D.schinazi-httpbis-transport-auth]). But even without authentication, at a minimum, the client needs to communicate directly with the proxy to provide the address of the target server it wants to connect to, e.g. using HTTP CONNECT, and potentially other information needed to inform the behaviour of the proxy.

Usually the server is not aware of the proxy in the middle, so the proxy needs to re-write the IP address of any traffic inside the tunnel to ensure that the return traffic is also routed back to the proxy. This is also often used to conceal the address/location of the client to the server, e.g. to access local content that would not be accessible by the client at its current location otherwise.

2.2. Advanced Support of User Agents

Depending on the traffic that is sent "over" the proxy, it is also possible that the proxy can perform additional support services if requested by the client. Today, Performance Enhancing Proxies (PEPs) usually work transparently by either fully or partially terminating the transport connection. For many of these support services the termination is actually not needed and may even be problematic. However, it is often the only, or at least easiest, solution if no direct communication with the client is available. Enabling these services based on an explicit tunnel setup between the client and the proxy provides such a communication channel and makes it possible to exchange information in a private and authenticated way.

It is expected that in-network functions are usually provided close to the client e.g. hosted by the access network provider. Having this direct relation between the endpoint and the network service is also necessary in order to discover the service, as the assumption is that a client knows how to address the proxy service and which service is offered (besides forwarding). Such a setup is especially valuable in access networks with challenging link environments such as satellite or cellular networks. While end-to-end functions need to be designed to handle all kind of network conditions, direct

support from the network can help to optimize for the specific characteristics of the access network such as use of link-specific congestion control or local repair mechanisms.

Further, if not provided by the server directly, a network support function can also assist the client to adapt or prioritize the traffic based on user preferences or device characteristics and capabilities. Again, especially if the access network is constrained, this can benefit both the network provider to save resources and the client to receive the desired service quicker or less impaired. Such a service could even be extended to include caching or pre-fetching if the necessary trust relationship between the client and the proxy exists.

Depending on the function requested, the proxy would need to access or alter the traffic or context which is limiting due to the necessary trust. Therefore alternative models should be pursued in most cases. One such model is explicit exchange of information about the current network state from the proxy to the client. This enables some services to function by having the end-to-end peers act on or inject the learned information from the proxy into the end-to-end connection(s). Thus achieving the benefits without the need to access the content or some of the traffic metadata directly. Especially transport layer optimizations do not need access to the actual user content. Network functions should generally minimize dependencies to higher layer characteristics as those may change frequently.

Similar to previous usage scenario, in this setup the client explicitly selects the proxy and specifies the requested support function. Often the server may not need to be aware of it but depending on the optimization function, server cooperation could be beneficial as well. Usually though, it is expected that even if the server is aware, no direct information exchange is needed between the proxy and the server. Instead, any needed information will be provided "over" the client and thus, the client and the proxy need a direct and secured communication channel in order to request and configure a service and exchange or expose the needed information and metadata.

2.2.1. Security and Access Policy Enforcement

Some deployment models may wish to enforce security or access policies on traffic flowing between domains (physical, logical, administrative, security etc.). To support this, endpoints coordinate through a gateway that can require information about the transport layer, application layer and application content. Policy is generally configured out-of-band, either statically or through some independent control plane.

In one use case, the enforcement function controls egress traffic; a client connects to a proxy, typically inside the same domain, in order to cross the domain boundary. In another use case, the enforcement function controls ingress traffic; a client connects to a proxy that controls access to the ultimate destination. This may be deployed inside the target domain, near it, or further away as a part of a third-party security service. Clients are usually remote

and diverse, and use connections that have crossed several other domains (with or without tunnels).

Enforcement functions typically require some form of client authentication such as username, password, or certificate. Authentication is enforced at the earliest stage of communication.

Enforcement rules might require access to transport characteristics of the ultimate endpoints (such as client source IP address). This might change as traffic moves between domains, whether tunneling is used or not. Therefore, it can be desirable to encapsulate original information in form accessible to the enforcement function.

2.3. Frontend Support for Load Balancing and Migration/Mobility

Application service providers aiming to improve access flexibility might use proxies in front of their services.

In one usage scenario the client communicates with a reverse proxy that assists with access to and selection of the content requested. This proxy that may or may not be under the authority of the service provider. Today such reverse proxies terminate the connection, including the security association, and as such appear as the communication endpoint to the client. Terminating both transport and security may be problematic if the proxy provider is not under the direct authority of the actual service provider (e.g. a contracted third party).

In another usage scenario the client communicates with a frontend proxy that manages traffic steering to assist with load balancing or migration for mobility support of server or client. This proxy is more likely to be located close to the server and under the same administrative domain, or at least has some trust relationship with the application service provider. The server may have its own communication channel with the proxy or tunnel endpoint in order to provide data that is used for decision making. Meanwhile, the client is usually not aware of any specifics of the setup behind the substrate endpoint. However, improving visibility may benefit future explicit tunneling or proxying approaches.

2.4. IoT Gateways

A number of IoT devices are connected via a low-power wireless network (e.g., a Bluetooth LE piconet) and need to talk to their parent cloud service to provide sensor readings or receive firmware updates.

When end-to-end IP connectivity is not possible or desirable for at least some of the devices, one or more IP capable nodes in the piconet can be designated as ad-hoc gateways to forward sensor traffic to the cloud and vice-versa. In other scenarios, a less constrained node - sometimes called a "smart gateway" - can provide the forwarding role permanently. In both cases, the gateway node routes messages based on client's session identifiers, which need to be unique among all the active participants so that the gateway can route unambiguously. The access network attachment is expected to change over time but the end-to-end communication (especially the security association) needs to persist for as long as possible.

A strong requirement for these deployments is privacy: data on the public Internet (i.e., from the gateway to the cloud service) needs to be made as opaque as possible to passive observers, possibly hiding the natural traffic patterns associated with the sensor network. A mechanism to provide discovery of the proxy node to the rest of the piconet is also typically necessary.

Today, the above requirements can be met by composing an end-to-end secure channel (e.g., based on DTLS sessions with client-chosen connection IDs [I-D.ietf-tls-dtls-connection-id] or application layer TLS [I-D.friel-tls-atls] from the sensors to the cloud together with a multiplexed secure tunnel (e.g., using HTTP/2 WebSockets [RFC8441], or a proprietary shim) from the gateway to the cloud. In the future, a more homogeneous solution could be provided by QUIC for both the end-to-end and tunneling services, thus simplifying code dependencies on the gateway nodes.

2.5. Multi-hop Chaining Usage

Providing a generic approach to use QUIC as a substrate also enables the combination of multiple of the above use cases. For example, employing multiple obfuscating proxies in sequence, where the communication with each proxy is individually secured, can enable onion-like layered security. Each proxy will only know the address of the prior hop and after itself, similar as provided by onion routing in Tor project [TOR].

Further, it would also be possible to chain proxies for different reasons. A client may select proxying support from its access network, while a web service provider may utilize a front-end load balancing proxy to provide end-to-end secure communication with the applications components servers. Here the proxy and the load balancer have different tasks. The access network proxy optimizes the aggregated data transport. The load balancer needs to route different set of end-to-end protected data that it aggregates. A third example would be multiple proxies to cooperate and maybe exchange measurement information in order to optimize the QUIC connection over a specific segment.

The above examples indicates that a solution likely have to consider how to establish a security model so that endpoints can selectively choose what connection related information to share with the different proxy entities. The possible efficiency should also be consider and multiple layers of encapsulation should be avoided when the security model allows for it.

2.5.1. Considerations for Multiple Encryption

Using QUIC in a multi-hop fashion will generally cause all user data to be encrypted multiple times, once for each hop. There are two main reasons to encrypt data multiple times in a multi-hop network:

1. To ensure that no hop can see both the connection metadata of the client and the server (thus obfuscating IP addresses and other related data that is visible in cleartext in the transport protocol headers).

2. To prevent an attacker from being able to correlate data between different hops to identify a particular flow of data as it passes through multiple hops.

However, multiple layers of encryption can have a noticeable impact on the end-to-end latency of data. When a Tor-like approach is used, each piece of user data will be encrypted N times, where N is the number of hops. Devices such as IoT devices that may not have support for cryptographic optimizations, or are constrained in terms of processing or power usage, could be significantly slowed down due to the extra overhead or not be able to process such traffic at all.

Since QUIC is an encrypted transport, the content of all packets after the handshake is opaque to any attacker. Short-header packets, particularly those that have zero-length Connection IDs, only send encrypted fields. Thus, for all packets beyond the QUIC handshake, encrypting packets multiple times through a multi-hop proxy primarily achieves benefit 2) described above, since benefit 1) is already achieved by QUIC being forwarded without re-encryption. If a deployment is more concerned with benefit 1) than benefit 2), it might be preferable to use a solution that forwards QUIC packets without re-encrypting once QUIC handshakes are complete.

3. Requirements

To use QUIC as a substrate, it could be beneficial if unreliable transmission is supported as well as having a way to potentially influence or disable congestion control if the inner tunnel traffic is known to be congestion controlled.

Communication between the client and proxy is more likely to be realized as a separate protocol on top of QUIC or HTTP as e.g. proposed by MASQUE. However, a QUIC extensibility mechanism could be used to indicate to the receiver that QUIC is used as a substrate and potentially additional information about which protocol is used for communication between these entities. A similar mechanism could be realized in HTTP instead. In both cases it is important that the QUIC connection cannot be identified as a substrate by an observer on the path.

With QUIC, the use of proxying functions cannot be done transparently. Instead, proxies needs to be explicitly discoverable. The simplest form of such discovery could include pre-configuration or via out-of-band signaling. The proxy could also be discovered through advertisement when a client is connected to a network (for example, the Dynamic Host Configuration Protocol). Alternatively, the client could obtain a white-listed proxy address when making first contact with the server (CNAME/IPaddress). In both cases the proxy needs to have a routable address and name.

4. Review of Existing Approaches

As already mentioned, HTTP proxies are usually realized by use of the HTTP CONNECT method (see Section 4.3.6 of [RFC7231]). This is commonly used to establish a tunnelled TLS session over a TCP connection to an origin server identified by a request-target. In HTTP/1.1, the entire client-to-proxy HTTP connection is converted into a tunnel. In HTTP/2 (see Section 8.3 of [RFC7540]) and HTTP/3

(see Section 4.2 of [I-D.ietf-quic-http]), a single stream gets dedicated to a tunnel. Conventional HTTP CONNECT is only specified to open a TCP connection between proxy and server, even in HTTP/3, so it enables forwarding based on a split TCP-TCP or QUIC-TCP connection but unaltered payload traffic. There is no currently-specified HTTP mechanism to instruct a proxy to create a UDP or IP association to the server. [HINT] contains a deeper analysis of the problem space and potential solutions. Of those explored, a good candidate for MASQUE is the Extended CONNECT method [RFC8441], accepts a ":protocol" pseudo-header that could be used to express an alternative protocol between proxy and server.

An explicit proxy control protocol is the SOCKS protocol [RFC1928]. Version 6 is currently under standardization [I-D.olteanu-intarea-socks-6] which provides fast connection establishment. Use of QUIC could even further improve that. However, SOCKS provides support to establish forwarding sockets using a new connection (with a different port). This behavior is visible to the path and not necessary if the underlying transport is multiplexing capable, as QUIC is. A SOCKS-like protocol could still be used for negotiation and authentication between the client and the proxy. An example proposal for this approach is [I-D.piraux-quic-tunnel].

In that sense the TCP PROXY protocol could also be seen as a light-weight version of SOCKS (see https://www.haproxy.org/download/1.8/doc/proxy-protocol.txt). This protocol was never standardized and only provides a limited set of functionality.

5. Contributors

Magnus Westerlund has contributed two paragraphs on combining proxies.

Tommy Pauly has contributed text on multiple layers of encryption, and other edits to the use cases.

6. References

6.1. Normative References

[I-D.schinazi-masque] Schinazi, D., "The MASQUE Protocol", Work in Progress, Internet-Draft, draft-schinazi-masque-02, 8 January 2020, <http://www.ietf.org/internet-drafts/draft-schinazi-masque-02.txt>.

6.2. Informative References

[HINT] Pardue, L., "HTTP-initiated Network Tunnelling (HiNT)", Work in Progress, Internet-Draft, draft-pardue-httpbis-http-network-tunnelling-01, 18 October 2018, <http://www.ietf.org/internet-drafts/draft-pardue-httpbis-http-network-tunnelling-01.txt>.

[I-D.friel-tls-atls] Friel, O., Barnes, R., Pritikin, M., Tschofenig, H., and M. Baugher, "Application-Layer TLS", Work in Progress, Internet-Draft, draft-friel-tls-atls-04, 4 November 2019,

<http://www.ietf.org/internet-drafts/draft-friel-tls-atls-04.txt>.

[I-D.ietf-quic-http] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-27, 21 February 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-http-27.txt>.

[I-D.ietf-tls-dtls-connection-id] Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", Work in Progress, Internet-Draft, draft-ietf-tls-dtls-connection-id-07, 21 October 2019, <http://www.ietf.org/internet-drafts/draft-ietf-tls-dtls-connection-id-07.txt>.

[I-D.olteanu-intarea-socks-6] Olteanu, V. and D. Niculescu, "SOCKS Protocol Version 6", Work in Progress, Internet-Draft, draft-olteanu-intarea-socks-6-08, 4 November 2019, <http://www.ietf.org/internet-drafts/draft-olteanu-intarea-socks-6-08.txt>.

[I-D.piraux-quic-tunnel] Piraux, M. and O. Bonaventure, "Tunneling Internet protocols inside QUIC", Work in Progress, Internet-Draft, draft-piraux-quic-tunnel-00, 4 November 2019, <http://www.ietf.org/internet-drafts/draft-piraux-quic-tunnel-00.txt>.

[I-D.schinazi-httpbis-transport-auth] Schinazi, D., "HTTP Transport Authentication", Work in Progress, Internet-Draft, draft-schinazi-httpbis-transport-auth-01, 8 January 2020, <http://www.ietf.org/internet-drafts/draft-schinazi-httpbis-transport-auth-01.txt>.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <https://www.rfc-editor.org/info/rfc1928>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <https://www.rfc-editor.org/info/rfc8441>.

[TOR] "TOR Project", 5 June 2019, <https://www.torproject.org/>.

Authors' Addresses

Mirja Kuehlewind
Ericsson

Email: mirja.kuehlewind@ericsson.com

Zaheduzzaman Sarker
Ericsson

Email: zaheduzzaman.sarker@ericsson.com

Thomas Fossati
Arm

Email: thomas.fossati@arm.com

Lucas Pardue
Cloudflare

Email: lucaspardue.24.7@gmail.com