SECURITY CLASSIFICATION BASED QUIC TRAFFIC STEERING IN A SECURE INTERNET GATEWAY (SIG)

Prashanth Patil
Rajesh I. V
Ram Mohan R

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
SECURITY CLASSIFICATION BASED QUIC TRAFFIC STEERING IN A SECURE INTERNET GATEWAY (SIG)

AUTHORS:
Prashanth Patil
Rajesh I V
Ram Mohan R

ABSTRACT
The Quick User Datagram Protocol (UDP) Internet Connection (QUIC) protocol is slated to become the next (third) major version of the Hypertext Transfer Protocol (HTTP) – i.e., HTTP/3. As applications transition to QUIC for web traffic, a Secure Internet Gateway (SIG) needs to effectively load balance, proxy, and classify QUIC traffic. Techniques are presented herein that make use of a custom Connection ID (CID) artifact to allow a load balancer to determine, with minimal processing, a target server, and potentially the application that the QUIC flow is serving, in support of steering traffic to the appropriate upstream services. Additionally, techniques are presented herein that leverage an exchange of data over an out-of-band channel in support of the enforcement of Quality of Service (QoS) requirements on an enterprise gateway.

DETAILED DESCRIPTION
The QUIC protocol is slated to become the next (third) major version of the HTTP – i.e., HTTP/3. As applications transition to QUIC for web traffic, a Secure Internet Gateway (SIG) needs to effectively load balance, proxy, and classify QUIC traffic. For example, a device is to intelligently steers different traffic types (such as, for example, Domain Name System (DNS), HTTP, non-HTTP, etc.) to the appropriate upstream service, or even to a Service Function Chain (SFC), to process the traffic as needed. Appropriate policies may be enforced on these traffic types as defined by a customer.

Techniques are presented herein that support the efficient processing and routing of QUIC traffic and the enforcement of QoS requirements. Elements of particular interest and note within the techniques that are presented herein are discussed below.
A first element of the techniques presented herein comprises identifying and steering traffic to the appropriate upstream services using a CID at a load balancer. Since QUIC employs UDP as a transport protocol it inherently supports multipath flows and connection migration. Consequently it is nontrivial for an intelligent proxy to inspect QUIC flows. However, QUIC does support the concept of a CID as part of the public portion of a packet.

Load balancers that support QUIC need to derive routable target server information from the CID. QUIC's consistent CID allows connections to survive changes to the client's Internet Protocol (IP) address and/or port, such as those caused by a client or a server migrating to a new network or a session transition in a server layer. However, CIDs can potentially change over the course of a session. This may be done to enable alternative CIDs to may be used to break link-ability when migrating connections. The load balancer must therefore maintain a relatively expensive table of server generated CIDs and will not route packets correctly if they use a CID that was originally communicated in a protected NEW_CONNECTION_ID frame.

To achieve stateless load balancing, servers can encode server identity in the CID as suggested by the draft https://tools.ietf.org/html/draft-ietf-quic-applicability-02. However, there are significant challenges with such a solution.

For example, traffic inspection is not possible in the absence of flows across multiple paths and connection migrations. Without the intelligent proxy ‘peeking’ at the full flow, it will not be able to apply appropriate policies to different flows. Further, CIDs are meant to prevent link-ability of connections across deliberate address migration through the use of protected communications between a client and a server. Avoiding link-ability in a CID of migrated connections creates issues for load balancing intermediaries.

The Internet Engineering Task Force (IETF) draft https://tools.ietf.org/html/draft-ietf-quic-load-balancers-02 proposes a mechanism to address load balancing issues. However, that approach demands a significant number of message exchanges between the cluster of load balancers and the pool of servers. It relies on load balancers creating CIDs and distributing them to all of the servers and the servers needing to inform the load balancers regarding stock availability, etc. This is extremely inefficient and complex, especially in a large scale data center that may have multiple load balancers and a
significant server pool. There are no efficient solutions to address the server-side state migration from one server to other, which results in a change of CID.

Additionally, in an elastic cloud environment that is orchestrated around microservices and auto-scaling it becomes difficult to keep track of servers and the type of applications and associate them with appropriate policies.

Aspects of a first element of the techniques presented herein address the problems described above by efficiently managing CID schemes across a pool of upstream servers. Relevant information is inserted in the CID such that a load balancer can efficiently route packets forward to a desired cluster and use the CID to determine server/application type for the traffic in question. This ensures that traffic is steered to the appropriate upstream servers across multipath flows and connection migrations.

A second element of the techniques presented herein comprises enforcing QoS for QUIC traffic on enterprise gateways. Traffic is routed or tunneled to a SIG from an enterprise gateway. QUIC, since it is based on Transport Layer Security (TLS) 1.3, offers very little connection information in the clear and it is therefore impossible for enterprise middleware to enforce QoS. TLS 1.3-related issues are clearly documented at, for example, https://tools.ietf.org/html/draft-camwinget-tls-ns-impact-00.

Aspects of a second element of the techniques presented herein employ the CID scheme that is described below so that an enterprise gateway may efficiently enforce QoS on the traffic without having to 'peek' into connections. In brief, the CID scheme supports visibility into the CID. The enterprise gateway that is aware of the CID scheme being employed is thus able to, possibly among other things, determine application types from the CID and then easily enforce QoS on the desired traffic (without having to burden a SIG in the cloud).

The CID scheme that is described below allows for relevant information to be carried without compromising any security and privacy requirements. A load balancer may determine, with minimal processing, the target server and potentially the application that the QUIC flow is serving. Further, support is provided for CID migrations. CID migration may result from, for example:

- A client-side 4-tuple changing due, possibly among other things, to Network Address Translation (NAT), IP renumbering, roaming, etc.
• A server-side 4-tuple changing due to, possibly among other things, NAT the transfer of a session from one server to another (e.g., due to internal issues or load requirements), etc.

The IETF document https://tools.ietf.org/html/draft-ietf-quic-transport-27#section-5.1 discusses CIDs in detail. Aspects of the techniques presented herein support each server generating its Source CID according to the following scheme:

Raw CID: <ServerGroupId><ApplicationId><Random>

where:

• **ServerGroupId** is a 16-bit alphanumeric identifier that is assigned to each group of servers.

• **ApplicationId** is a 16-bit alphanumeric identifier that is assigned to business critical applications that are supported by the servers.

• **Random** is a 32-bit alphanumeric value that is randomly generated for each CID. 16 bits of that value may potentially be used to carry a **TenantId** to determine and identify tenant-specific traffic.

The raw CID is then encrypted using a 64-bit key:

CID : <encrypted raw CID><Key Id><Random>

where:

• **Key ID** is a 4-bit identifier that indicates the key that was used to encrypt the raw CID. This is optional and is only used when encryption keys need to be frequently rotated. This identifier is useful when CIDs are encrypted with more than one key due to overlapping session times. For example, an update in keys may take place while an older session is still in flight, resulting in the older session carrying a CID encrypted with an older key but newer sessions carrying CIDs encrypted with a new key.

The maximum allowed size for a CID is 144 bits, and the scheme that was presented above utilizes that entire length by having the random value bits complete the maximum size. The number of bits available for a random value may vary based on the selected encryption algorithm. It is important to note that this is only a suggested means to create a CID. Other variations may be possible depending upon, for example, deployments and requirements, and bits may be moved around as desired. It is also important to note that
encryption is optional - it is only introduced if the deployment requires that CID schemes and CID contents be private.

Figure 1, below, presents a high-level block diagram that encompasses aspects of the techniques presented herein.

![Figure 1: High-Level Block Diagram](image)

Setup activities that may occur under aspects of the techniques presented herein may include, for example:

1. If the solution uses an orchestration framework, the orchestrator programs each load balancer and each server in the pool with the following details:

   2. {
      "cid_scheme": "<ServerGroupID><ApplicationID><Random>",
      "algorithm": "<Algorithm>",
      "key": {
        "key_id": "<id>",
        "secret": "<secret>"
      }
   }

2. If the deployment requires that keys be rotated frequently, a Key Management System (KMS) may push new keys accordingly.

3. In the absence of a KMS, one of the load balancers assumes the role of a CID controller and performs Steps 1, 2, and 3 to push new keys.
5. Each server is configured to insert the appropriate application Id in the CID. The server inserts its Source CID when it sends out its first packet, on the assumption that the server knows the type of application that is being served by that time. If not, the server may insert something more generic, such as a type of server/service. For example, if an enterprise deploys Cisco networking gear and configures Security Group Tags (SGT), the server may insert an appropriate Security Tag in this space. The result is the insertion of an indicator that may be useful for the load balancer and other network devices.

Under aspects of the techniques presented herein various of the interactions or exchanges that are possible may include, for example:

1. A client submits a new request.

2. A load balancer, based on a configured load balancer algorithm, selects a target server and forwards the request.

3. The server, when it responds, inserts a CID based on the scheme that was described above.

4. The load balancer may enforce QoS and/or other network policy on any return traffic by gleaning information from the CID. The load balancer, because it possesses all of the required keys, may decrypt the CID and select any relevant information.

5. The client continues the session. The load balancer is able to decrypt the CID (the <encrypted raw CID> portion of the CID string), determine the target server, and route forward as needed. The load balancer may optionally cache aspects of this information to avoid subsequent decryption.

Figure 2, below, presents a flow diagram that illustrates portions of the interactions or exchanges that were discussed above.
Figure 2: Illustrative Flow Diagram
When a connection migration takes place on the client side, upon receiving a NEW_CONNECTION_ID frame the server may create a new CID based on the CID scheme that was described above. (The random value ensures that a new unique CID is generated).

If a server intends to transfer a session across to another server, the retiring server identifies the new server that will be taking over, creates a new CID using the Server_Id of the new server, and then advertises the new CID to a client. The session may then be migrated to the new server. Any further requests that arrive on the old connection are routed by the load balancer to the new designated server.

A load balancer and an enterprise gateway may employ an out-of-band channel to exchange various details (such as, for example, the keys that are required to decrypt a CID) so that the enterprise gateway is able to understand a CID. The enterprise gateway may then be programmed to enforce QoS based on the desired bits included in the CID. For example, the gateway may be programmed to determine the application type from a CID and select an appropriate QoS policy. When an enterprise gateway encounters QUIC traffic, the required QoS is enforced based on the CID in the clear QUIC headers without having to ‘peek’ inside (e.g., inspect) the packet itself.

Regarding the CID scheme that was described above, it is important to note that it is only necessary for a QUIC proxy (e.g., a forward proxy) to follow that scheme. The QUIC proxy generates the CID; however, the proxy does not influence the client CID. Rather, the proxy only creates the service CID based on the CID scheme described herein. The enterprise gateway can ‘peek’ into the CID (which is in the clear QUIC headers) to enforce any local policy based on the scheme and constituents of the CID. The client and the ultimate server don't care or need to comprehend the CID. It is simply a connection identifier for them. In fact, the ultimate server will probably not even see such a CID because the QUIC connection between the proxy and the ultimate server may carry different CIDs altogether. The QUIC connection created between the proxy and the server does not have to follow the CID scheme; it has no real bearing on the connection between the client and the forward proxy.

In summary, techniques have been presented that make use of a custom CID artifact to allow a load balancer to determine, with minimal processing, the target server and
potentially the application that the QUIC flow is serving in support of steering traffic to appropriate upstream services. Additionally, techniques have been presented that leverage an exchange of data over an out-of-band channel in support of the enforcement of QoS requirements on an enterprise gateway.