Journal of Universal Computer Science, vol. 32, no. 2 (2026), 181-208
submitted: 2/4/2025, accepted: 3/10/2025, appeared: 28/2/2026 CC BY 4.0

Steganography in the QUIC Communication Protocol

Aleksandar Velinov
(Faculty of Computer Science, Goce Delcev University, Stip, North Macedonia
https://orcid.org/0000-0002-5940-8655, aleksandar.velinov@ugd.edu.mk)

Aleksandra Mileva
(Faculty of Computer Science, Goce Delcev University, Stip, North Macedonia
https://orcid.org/0000-0003-0706-6355, aleksandra.mileva@ugd.edu.mk)

Simon Volpert
(Ulm University, Ulm, Germany
https://orcid.org/0000-0002-4896-7830, simon.volpert@uni-ulm.de)

Sebastian Zillien
(Ulm University, Ulm, Germany
https://orcid.org/0000-0003-3360-1251, sebastian.zillien@uni-ulm.de)

Steffen Wendzel
(Ulm University, Ulm, Germany
https://orcid.org/0000-0002-1913-5912, steffen.wendzel@uni-ulm.de)

Abstract: Network steganography has existed for several decades and it uses network traffic and network protocols as carriers for embedding secret messages in a stealthy manner. Quick UDP Internet Connections (QUIC) is a novel secure and reliable transport layer network protocol that is encapsulated in the User Datagram Protocol (UDP) and utilizes the Transport Layer Security Version 1.3 (TLSv1.3) standard. In addition, Hypertext Transfer Protocol Version 3 (HTTP/3) employs QUIC. In this paper, we present a systematic analysis of the covert channels that can be found in QUIC. Twenty novel covert channels are identified by applying the latest covert channel pattern based taxonomy, and an analysis of their transmission rate, undetectability, and robustness is presented, together with suggested countermeasures. A single covert channel is implemented as a proof of concept tool and is appropriately evaluated.

Keywords: Network steganography, covert channels, QUIC, UDP, HTTP/3
Categories: E.3, H.3.5, H.5.4
DOI: 10.3897/jucs.154672

1 Introduction

Network steganography is a technique that uses network protocols to hide secret data. Data is hidden in legitimate user transmissions without disrupting the legitimate data that are actually transferred. In this way, the so-called covert channels are created through which data can be transmitted secretly. Covert channels were first introduced by Lampson in 1973 [Lampson 1973]. He defines them as channels that are not intended for information transfer at all, such as the effect of service programs on the system load. The usage of

covert channels could be for multiple purposes, for example: military communication in hostile environments [Murdoch 2007], bypassing censorship [Feamster et al. 2002], communication of security agencies [Wendzel & Keller 2014], quality assurance of service for VoIP (Voice over Internet Protocol) traffic [Mazurczyk & Kotulski 2006]. Several covert channels have been discovered in the past [Mileva & Panajotov 2014, Zander et al. 2007, Mazurczyk & Szczypiorski 2008, Mileva et al. 2018, Cabuk et al. 2009, Wendzel et al. 2015, Mazurczyk et al. 2016].

QUIC or *Quick UDP Internet Connections* is an encrypted-by-default UDP-based transport layer network protocol, which is also connection-oriented and reliable. Initially, QUIC was developed by Google engineers in 2012, with the main goal of improving the performance of their web applications [The Chromium Projects 2023]. This version is known as gQUIC or Google QUIC. Chrome developers started conducting experiments with QUIC in 2013, and as a result of this, in 2014 they started a wide-scale implementation of Google QUIC. In 2015, the Internet Engineering Task Force (IETF) submitted a specification of QUIC as an Internet draft for standardization [Iyengar & Swett 2015]. In 2017, the IETF created a version of QUIC that was based on gQUIC. This version is known as IETF QUIC. In May 2021, QUIC was published as an IETF standard [Iyengar & Thomson 2021, Thomson 2021, Thomson & Turner 2021, Iyengar & Swett 2021], while in May 2023, QUIC version 2 was standardized [Duke 2023], with no additional features (the main purpose is to acclimate middleboxes for different versions).

The development of QUIC is related to the need to have a secure and reliable transport-based on UDP for HTTP, and now the last version of HTTP, HTTP/3 [Bishop 2022], operates on the basis of QUIC. In fact, some of the features of HTTP/2 (for example, multiplexing and flow control) [Thomson & Benfield 2022] are now built into QUIC, freeing HTTP/3 of its burden. The older versions, HTTP/1.0, HTTP/1.1 and HTTP/2 use TCP for transmission. The key features of QUIC are: fast handshake and connection establishment with 0-Round Trip Time (0-RTT) and 1-Round Trip Time (1-RTT) handshake mechanism, encrypted and authenticated packets, multiplexing, congestion control, Forward Error Correction (FEC) and connection migration. According to [W3Techs 2024], the QUIC protocol is now used by 8.5% of all websites. For a 4 year period, there is a significant increase in QUIC usage; in 2020 the percentage of QUIC usage was only 2.8%. In addition, 31.2% of all websites currently use HTTP/3 over QUIC [W3Techs 2024]. Google, Amazon, Facebook, YouTube, and Instagram are among the known websites that use QUIC.

Currently, there is only one paper [Huang et al. 2025] that presents a network covert channel using some very basic QUIC properties, without diving too much into the protocol itself. This is the reason why we provide a comprehensive analysis of QUIC susceptibility to network covert channels in this paper. Our main contributions are:

- Identification of 20 novel covert channels that are related to the QUIC features, together with their properties and suggested countermeasures;
- Categorization of novel covert channels by using the pattern taxonomy which is defined in [Wendzel et al. 2025], analysis of their properties and suggestion of appropriate countermeasures;
- Implementation as proof of concept and experimental evaluation of one of the new covert channels.

The remainder of the paper is structured as follows. In Section 2 we provide an overview of the related work on QUIC security and network steganography. In Section

3 we present how QUIC works. Section 4 introduces novel covert channels in QUIC. The properties of covert channels and the suggested countermeasures are presented in Section 5, while the Proof-of-Concept is introduced in Section 6. Section 7 concludes our work.

2 Related work

The steganographic analysis of QUIC is only a part of its security analysis that identifies different technical issues in the protocol. Some surveys of possible security attacks on QUIC are given by Chatzoglou et al. [Chatzoglou et al. 2023] and Joarder and Fung [Joarder & Fung 2022].

QUIC can be used to perform the so-called QUIC flood Distributed Denial of Service (DDoS) attack (or state-overflow or state-exhaustion attack), by overwhelming a targeted server with QUIC data [Nawrocki et al. 2021]. In this case, the targeted server is the victim, because it must process all received QUIC data (decrypt, create, and encrypt the QUIC response). This will consume the server's computational power and will result in much slower service to legitimate users or rejection of their requests. Especially reflective amplification DDoS attacks impose a vulnerability to QUIC. Server "Hello" messages are much larger than client "Hello" messages because they include TLS certificates among other data. So, by spoofing the victim's IP address, the attacker can send a "Hello" message to a server, which will trick the server into sending large portions of unwanted data to the victim.

Iyengar and Thomson identified several possible attacks on QUIC in the standard itself [Iyengar & Thomson 2021], and in the subsequent draft [Iyengar & Thomson 2023], and suggested some measures that can limit or stop these attacks. These attacks are: spoofed ACK attack, optimistic ACK attack, slowloris attacks, stream fragmentation and reassembly attacks, stream commitment attack, explicit congestion notification attacks, request forgery attacks, stateless reset oracle, etc.

Chatzoglou et al. [Chatzoglou et al. 2023] divide existing attacks on QUIC into five categories: handshake attacks, cryptographic attacks, privacy attacks, fuzzing attacks, and transport layer attacks, but also provide several new attacks on QUIC.

Regarding network steganography, there is a steganographic analysis for HTTP/2 [Dimitrova & Mileva 2017], and because some of the functions of HTTP/2 now belong to the QUIC protocol, this research is relevant for QUIC. More recently, the first use of QUIC for covert communication has appeared in [Huang et al. 2025]. The paper presents *QuicCourier*, a framework that creates a network covert channel between a proxy node as a covert sender and a user as a covert receiver, by using QUIC packet manipulation, which involves padding secret data at the end of or before the QUIC packet, or replacing the QUIC packets with `ACK` frames.

3 Functionality of QUIC

QUIC is a general-purpose and secure transport protocol [Iyengar & Thomson 2021]. It creates a stateful interaction between the client and the server, and it is a connection-oriented protocol. In the handshake process, QUIC uses a combination of transport and cryptographic parameters that are negotiated. It has an integrated TLSv1.3 handshake and different measures for packet protection [Thomson & Turner 2021].

QUIC packets, encrypted and carried in UDP datagrams, are used for communication between endpoints. Most packets contain frames that convey application data and control information between endpoints. QUIC assumes a smallest allowed maximum datagram size of 1,200 B.

QUIC connections can be transferred to a new network path. For this purpose, in the connection migration process, QUIC uses connection identifiers. According to [Iyengar & Thomson 2021], only clients are allowed to migrate the connection. This concept enables the continuation of the connection after some address mappings or changes in network topology. An example of this is Network Address Translation (NAT) rebinding.

There are several ways in which the QUIC connection can be terminated. Some applications can cause a graceful shutdown. Errors can immediately trigger connection termination. The timeout period can be used to terminate the connection. There is also a stateless mechanism that can cause connection termination after losing the state of one endpoint. QUIC also has mechanisms for reliable delivery and congestion control to escape network congestion.

3.1 Packets and Frames

There are two types of QUIC packets: packets with long header (`Initial`, `0-RTT`, `Handshake`, `Retry`, and `Version Negotiation` packets), sent prior to the establishment of 1-RTT keys (starting with a 1-bit, cf. Fig. 1), and packets with short header (`1-RTT` packets), sent afterwards (starting with a 0-bit, cf. Fig. 2). The size of the long header is 47 B for the common fields, plus the size of type-specific fields. Short header packets must be at least 21 B in length. A sender can combine multiple QUIC packets in one UDP datagram, with the exception of `Retry` and `Version Negotiation` packets.

The functionality of fields in QUIC packets with a long header is as follows:

- `Header Form` bit (1 bit) - set to 1 for a long header;
- `Fixed Bit` (1 bit) - set to 1, unless the packet is a `Version Negotiation` packet;
- `Long Packet Type` (`LPT`, 2 bits) - set to 00 for `Initial`, 01 for `0-RTT`, 10 for `Handshake`, 11 for `Retry` packets, and ignored in `Version Negotiation` packets;
- `Reserved` bits (`RB`, 2 bits) - set to 00 for `Initial`, `0-RTT` and `Handshake` packets, and unused (random value or ignored) in other packets;
- `Packet Number Length` (`PNL`, 2 bits) - the length of the `Packet Number` field plus one in `Initial`, `0-RTT` and `Handshake` packets, and unused (random value or ignored) in other packets;
- `Version` (32 bits) - current version or `0x00000000` in Version Negotiation packets;
- `Destination Connection ID Length` (8 bits) - the length in bytes of the `Destination Connection ID` field that follows it (with the maximum of 20 bytes);
- `Destination Connection ID` (0-160 bits);
- `Source Connection ID Length` (8 bits) - the length in bytes of the `Source Connection ID` field that follows it (with the maximum of 20 bytes);
- `Source Connection ID` (0-160 bits);

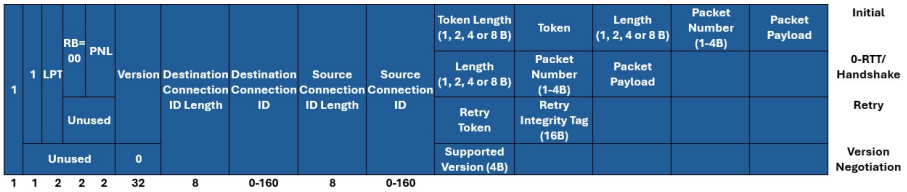


Figure 1: Structure of packets with a long header

Figure 2: Structure of packets with a short header

– `Token Length` (1, 2, 4 or 8 bytes) - a variable-length integer specifying the length of the `Token` field in bytes. This value is set 0 if no token is present;

– `Length` (1, 2, 4 or 8 bytes) - the length of the `Packet Number` and `Packet Payload` fields in bytes, encoded as a variable-length integer;

– `Retry Token` - an opaque token that the server is using for validation of the client's address;

– `Supported Version` (4 bytes)- in `Version Negotiation` packets;

– `Token` (0 or more bytes) - the token value previously provided in a `Retry` packet or `NEW_TOKEN` frame;

– `Packet Number` (1-4 bytes);

– `Retry Integrity Tag` (16 bytes) - the output of AEAD_AES_128_GCM in TLS;

– `Packet Payload` (1 or more bytes) - contains a sequence of frames.

The functionality of fields in QUIC packets with a short header is as follows:

– `Header Form` bit (1 bit) - set to 0 for a short header format;

– `Fixed Bit` - set to 1;

– `Spin Bit` (`SB`, 1 bit) - if set, enables passive latency monitoring from observation points on the network path;

– `Reserved` bits (`RB`, 2 bits) - set to 00;

– `Key Phase` bit (1 bit) - indicates the key phase, which allows a recipient to identify the protection keys that are used to protect the packet;

– `Packet Number Length` (`PNL`, 2 bits) - the length of the `Packet Number` field plus one;

– `Destination Connection ID` (0-160 bits);

– `Packet Number` (1-4 bytes);

– `Packet Payload` (1 or more bytes).

In QUIC, four encryption levels are used, producing keys for `Initial`, `0-RTT`, `Handshake`, and `1-RTT` packets. `Version Negotiation` packets are sent in plain form, `Retry` and `Initial` packets have strong integrity protection, while `Handshake`, `0-RTT` and `1-RTT` packets have strong confidentiality and integrity protection. In fact, the header fields `RB`, `PNL` and `Packet Number` in long header packets, together with `RB`, `Key Phase`, `PNL`, and `Packet Number` fields in short header packets, have header protection.

The payload of QUIC packets is a sequence of multiple (at least one) complete frames that can be of different types. `Version Negotiation` and `Retry` packets do not contain frames.

Each frame starts with an 8-bit `Frame Type` field, indicating its type, followed by additional type-dependent fields. There are 20 different types of QUIC frames, presented in the Tab. 1.

Frame Type Value	Frame Type Name	Packets
0x00	PADDING	Initial, Handshake, 0-RTT, 1-RTT
0x01	PING	Initial, Handshake, 0-RTT, 1-RTT
0x02-0x03	ACK	Initial, Handshake, 1-RTT
0x04	RESET_STREAM	0-RTT, 1-RTT
0x05	STOP_SENDING	0-RTT, 1-RTT
0x06	CRYPTO	Initial, Handshake, 1-RTT
0x07	NEW_TOKEN	1-RTT
0x08-0x0f	STREAM	0-RTT, 1-RTT
0x10	MAX_DATA	0-RTT, 1-RTT
0x11	MAX_STREAM_DATA	0-RTT, 1-RTT
0x12-0x13	MAX_STREAMS	0-RTT, 1-RTT
0x14	DATA_BLOCKED	0-RTT, 1-RTT
0x15	STREAM_DATA_BLOCKED	0-RTT, 1-RTT
0x16-0x17	STREAMS_BLOCKED	0-RTT, 1-RTT
0x18	NEW_CONNECTION_ID	0-RTT, 1-RTT
0x19	RETIRE_CONNECTION_ID	0-RTT, 1-RTT
0x1a	PATH_CHALLENGE	0-RTT, 1-RTT
0x1b	PATH_RESPONSE	1-RTT
0x1c-0x1d	CONNECTION_CLOSE	Initial, Handshake, 0-RTT, 1-RTT
0x1e	HANDSHAKE_DONE	1-RTT

Table 1: Frame Types

The `Initial` packet type is used by a client or a server for any packet that contains an initial cryptographic handshake message in a `CRYPTO` frame (or frames). The first packet sent by the client is always an `Initial` packet with a `CRYPTO` frame, and in response to it, the server sends its first `Initial` packet (Fig. 3). If the server sends a TLS HelloRetryRequest, the client will send another series of `Initial` packets, to continue the cryptographic handshake.

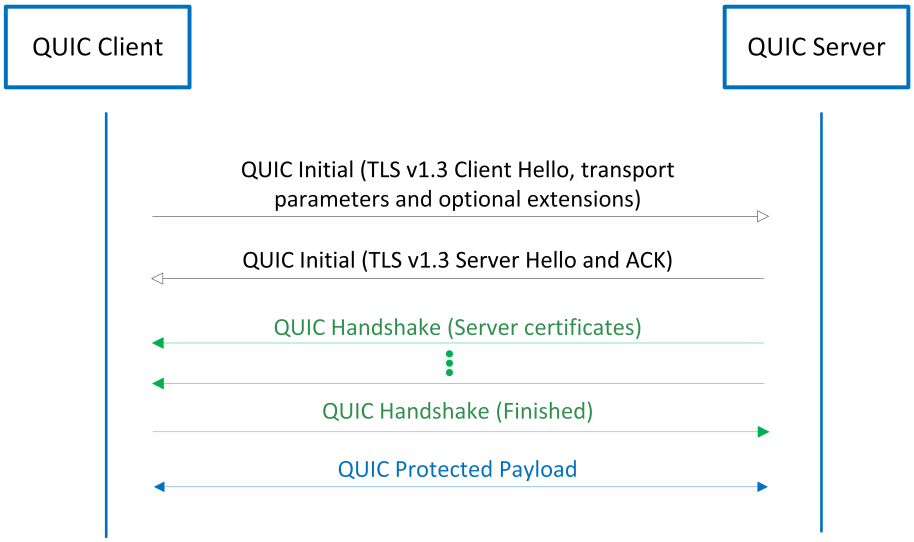


Figure 3: Basic connection establishment in QUIC

It is possible for a QUIC server to validate the client address before starting the cryptographic handshake. For that purpose, upon receiving the client's `Initial` packet, the server can request an address validation by sending a `Retry` packet with a token. The client must provide this token in all future `Initial` packets to the server. The token can be changed by the server by sending a `NEW_TOKEN` frame.

After the client sends its first `Handshake` packet, the client and the server stop sending `Initial` packets. The purpose of `0-RTT` packets is to carry early data from the client to the server, prior to handshake completion. The server can accept or reject these early data.

Finally, a `1-RTT` packet is used after the version and 1-RTT keys are negotiated, to carry the data.

3.2 QUIC connections

The client and the server have a shared state known as a QUIC connection. The first step of a connection is the handshake phase in which the two endpoints using the cryptographic handshake protocol set up a shared secret and agree on the usage of the application protocol with negotiation.

Each connection is identified by a set of `Connection IDs`. When a client sends an `Initial` packet, it chooses the initial values for the `Destination Connection ID` and the `Source Connection ID` fields in the long header. The server can change its own `Source Connection ID` value in the `Retry` packet if it wants to do so. Each endpoint sets the `Destination Connection ID` field in subsequent packets it sends to the value of the received `Source Connection ID` field. The same `Connection ID` can not be used more than once on the same connection. If a `Connection ID` is not needed for routing to the correct endpoint, then a zero-length `Connection ID` can be used, which means that the local address and port are used for connection identification.

The new Connection IDs can be replaced by the endpoint with the NEW_CONNECTION_ID frame. Each newly created Connection ID must be a value that is increased by 1. The Connection ID can be retired using the RETIRE_CONNECTION_ID frame. With this, the endpoint can delete the use of a Connection ID and it will not be used again. The endpoint also must ensure that its peer has a sufficient number of unused and available Connection IDs.

3.3 Streams and Flow Control

Protocols on the application layer use ordered sequences of payload bytes known as streams to exchange information using QUIC. QUIC uses a credit-based scheme to set the maximum amount of data which can be sent and to limit the creation of streams. Streams are created for the purpose of sending data, and after this is done, they are closed (they may also be canceled). Long-lived streams also exist and can last for the entire duration of the connection. The ordering of bytes on different streams is not ensured by QUIC.

There are two types of streams: bidirectional streams and unidirectional streams. Bidirectional streams permit both endpoints to send data, whereas unidirectional streams allow sending data in one direction.

All streams in a connection are identified with a unique 62-bit integer value, known as the stream identifier or Stream ID. The Stream ID is unique for all streams in the connection. That is the reason why an endpoint cannot reuse the same Stream ID for different streams in a single connection. There are four stream spaces, according to the initiator of the stream and the directionality of the stream (unidirectional or bidirectional). The two least significant bits of a stream ID identify a stream type. So, 0x00 is for bidirectional client-initiated streams, 0x01 is for bidirectional server-initiated streams, 0x02 is for unidirectional client-initiated streams, and 0x03 is for unidirectional server-initiated streams.

All stream spaces start with the appropriate minimal values and increase by 4 for each newly created stream of the specific type. For example, if there is an existing unidirectional client-initiated stream with Stream ID 0x02, the next should be with Stream ID 0x06. If the created stream has a Stream ID that is out of order (e.g., 0xA instead of 0x06 for our example), then all streams of that type which have lower Stream ID values are also opened (e.g., stream with Stream ID 0x06 will also be opened).

Data sent by the applications are encapsulated by STREAM frames. The Stream ID and Offset fields from the STREAM frames are used by the endpoints to put data in order.

The receivers can limit the amount of data that they can buffer for a given stream or for all the streams, in order to protect from the consumption of large amount of system resources. QUIC endpoint can also control the maximum number of streams that are initiated by a given peer to restrict the connection concurrency. There are two main types of data flow control in QUIC: stream flow control and connection flow control.

4 Covert Channels in QUIC

4.1 Application Scenario

The application of QUIC covert channels involves a covert sender (CS) and one or more covert receivers (CR). CS and CRs can reside on different networks. We assume that CRs know when CS starts sending covert data.

We are going to use two generic application scenarios that can be applied in different settings, for example, censorship circumvention, and data leakage and/or data exfiltration. In the first application scenario (AS1, Fig. 4a), the CS is located on some node as an overt sender (e.g., a web server) in the form of hidden malware for example. CRs are located on the intermediate devices through which all overt communication used as a covert communication carrier must pass from the network where the CS is located to any overt destination. For example, this can be a router that connects a given LAN to the Internet, a firewall, or some security appliance. This scenario can involve one or more overt communications between a server, where the CS is located, and innocent clients. The main characteristic of the AS1 scenario is that the parts of the QUIC protocol deployed in the CC are not encrypted, and it is enough for CR to eavesdrop the network. But in this way, secret data cannot be exchanged with the remote network.

The second scenario involves communication between the host (as an overt sender) where the CS is located and one innocent overt destination, which can be located on a remote network. CRs are placed on any intermediate devices (e.g. router, firewall) through which the overt communication between the CS and the overt destination must pass (AS2, Fig. 4b) on the outgoing or ingoing network, or at least CRs eavesdrop on the network traffic from these devices. The main characteristics of the AS2 scenario is that the parts of the QUIC protocol deployed in the CC are encrypted, so, to actually access the secret data, decryption is needed. For this purpose, we differentiate two different cases here. In the first case, we can have an insider placed in the overt destination, who can supply the CRs with session keys used for QUIC traffic encryption. They can do this for example by using a malware that has corrupted the QUIC endpoint or QUIC implementation and sets or reads the session keys for all communications. In the second case, the CS itself can have the ability to find and share the session keys with the CRs. Because we envision this scenario for data exfiltration, CR can store saved encrypted traffic for a while, and any time later they can obtain the session keys for decryption. This is easier to exchange than the large amount of data that is sent through the CC.

Because one big application of QUIC is with HTTP/3, we can assume that, as a carrier, an overt communication between a web server and a web client (browser, for example) is used. For many browsers, session keys can be found in their session storage.

4.2 Covert Channel Analysis

To determine covert channels (later referred to as CC) in QUIC, we applied the *hiding pattern*-based approach by Wendzel et al. [Wendzel et al. 2025]. Hiding patterns describe hiding techniques in an abstract and protocol-independent manner [Wendzel et al. 2015]. We analyze QUIC's specification to determine if the known hiding patterns can be realized with the protocol.

E1n1. Network State/Value Modulation and the corresponding sub patterns encode information by modulating the state or value of network elements [Wendzel et al. 2025]. For this class, we can identify several CCs:

- **E1n1_1 CC with stream identifiers** - This bidirectional CC can be applied to QUIC, but with respect to unidirectional and bidirectional streams initiated by the client or the server, which means that consecutive streams of the same type and the same sender differ in 4 (similar CC exists in HTTP/2 [Dimitrova & Mileva 2017]). Thus, a one-bit bidirectional CC can be created in the following way:

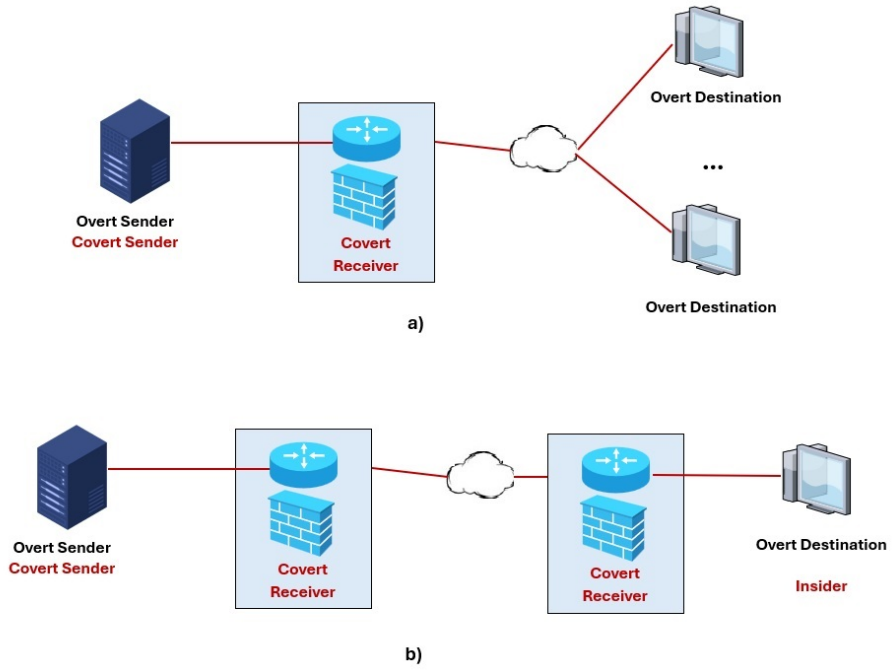


Figure 4: Application scenarios a) AS1 and b) AS2

- The client sends a binary 1 to the server by initiating a new stream with a `Stream ID` $=$ max(CSI) $+ 4$, and a binary 0 with a `Stream ID` $=$ max(CSI) $+ 8$;
- To send a secret binary 1 to the client, the server initiates a new stream with `Stream ID` $=$ max(SSI)$+4$. Likewise, a binary 0 is transferred with `Stream ID` $=$ max(SSI) $+ 8$.

In this scenario max(CSI) and max(SSI) are the largest `Stream ID`s used for streams at a given time, initiated by the client and the server, respectively.

– **E1n1_2 CC with Packet numbers** - Each packet, except `Retry` and `Version Negotiation` packets, has a 62-bit `Packet number`, which belongs to one of three spaces: Initial space (`Initial` packets), Handshake space (`Handshake` packets) and Application data space (`0-RTT` and `1-RTT` packets). Subsequent packets sent in the same packet number space increase the `Packet number` by at least one. Thus, a one-bit bidirectional CC can be created per space using the following rule:
  - The CS transfers a binary 1 by sending the following packet from the same space with the `Packet number` being increased by 1;
  - The CS transfers a binary 0 by sending the following packet from the same space with the `Packet number` being increased by ≥ 2.

– **E1n1_3 CC with variable-length encoded fields** - The QUIC standard uses variable-length encoding for nonnegative integers, or more specifically, integers are encoded with 1, 2, 4, or 8 bytes, which can encode 6-, 14-, 30-, or 62-bit values, respectively. In addition, the QUIC standard does not request values encoded on the minimum number of bytes for them, with the `Frame Type` field as the sole exception. For this reason, one can create a bidirectional one-bit CC between a client and a server per each variable-length encoded field (different than `Frame Type`) that has a value of less than 31 bits (i.e., necessary minimum number of bytes is less than 4), in the following way:
  - CS transfers a binary 1 encoding a variable-length encoded field (e.g., `Length`, `Token Length`) with the minimum number of bytes;
  - CS transfers a binary 0 encoding a variable-length encoded field with more than the necessary minimum number of bytes.

– **E1n1_4 CC with transport parameters** - During connection establishment, both endpoints unilaterally declare their transport parameters, independent of the values chosen by the opposite peer. These declarations are included during the handshake and carried in a TLS extension. Each transport parameter should be sent only once, and if it is not present, the default value is usually 0. Most of the transport parameters have integer values that use variable-length integer encoding. Hiding data in some transport parameters is possible but influences the connection. Possible choices are as follows:
  - $max\_udp\_payload\_size$ - values from 1,200 to the maximum permitted UDP payload of 65,527. If the encoded secret value is in this specific range, all 16 bits can be used;
  - $initial\_max\_data$ - initial maximum amount of data in bytes that can be sent on the connection. Up to 64 bits can be embedded by this parameter. The same approach can be made by sending a `MAX_DATA` frame for the connection immediately after completing the handshake;
  - $initial\_max\_stream\_data\_bidi\_local$ - initial flow control limit for locally initiated bidirectional streams as an integer value. Up to 64 bits can be sent by this parameter. Similarly, one can send a `MAX_STREAM_DATA` frame on every stream after its opening;
  - $initial\_max\_stream\_data\_bidi\_remote$ - initial flow control limit for bidirectional streams initiated by the opposite peer. Similarly, one can hide up to 64 bits in this parameter. Alternatively, one might send a `MAX_STREAM_DATA` frame over every stream after its opening;
  - $initial\_max\_stream\_data\_uni$ - initial flow control limit for unidirectional streams. Can be utilized by sending a `MAX_STREAM_DATA` frame on every stream after its opening (one can hide up to 64 bits in this parameter);
  - $initial\_max\_streams\_bidi$ - initial maximum number of bidirectional streams the receiving endpoint is permitted to initiate. Again, one might alternatively send a `MAX_STREAMS` frame here. Because the `Stream ID` is up to 62 bits while the last two significant bits determine the type of the stream, one can hide up to 60 bits;

- $initial\_max\_streams\_uni$ - initial maximum number of unidirectional streams the receiving endpoint is permitted to initiate. Again, one might send a `MAX_STREAMS` frame (up to 60 bits can be used for hiding);

In summary, using the specified transport parameters, a bidirectional covert channel can be created, with 392 bits $(16 + 4 \times 64 + 2 \times 60 = 392)$ in each direction.

The sub pattern **E1.1n1 Network Reserved/Unused State/Value Modulation** modulates reserved or unused network elements [Wendzel et al. 2025].

– **E1.1n1_1 CC with Unused field in `Version Negotiation` packet** - Because the 7-bit `Unused` field in a `Version Negotiation` packet sent by the server can be randomly filled (except the most significant bit, which must be 1 in some cases), the server can send a 7-bit secret message inside the `Unused` field.

E1.2n1 Network Random State/Value Modulation uses random or pseudo random elements to encode the hidden data in [Wendzel et al. 2025].

– **E1.2n1_1 CC with `Token` value** - A server can send a token to the client to validate a client's address in two ways: (i) In the `NEW_TOKEN` frame, so the client has a new token for a future communication, and (ii) in a `Retry` packet, in which case the client must send the token in each `Initial` packet that follows. There is no specification of the exact length of the token, and there is no well-defined format how they are generated. For example, tokens for `Retry` packets should include a client's IPv4 address (32 bits) and the port (16 bits) in the token. The recommendation in the QUIC standard is that a token value should include at least 128 bits of entropy. Thus, at least an 128-bit unidirectional CC between the server and the client can be realized by using the `Token` field the in `Retry` packet / `NEW_TOKEN` frames.

– **E1.2n1_2 CC with `Stateless Reset Token` value** - Endpoint can send a `NEW_CONNECTION_ID` frame to the other communication peer that contains an alternative connection ID for a stateless reset when connections are migrated. In addition to the alternative connection ID, a 128-bit `Stateless Reset Token` value is used in the process of a stateless reset. There is no fixed method for this value's generation (there is a suggested method for its generation, but it can also be randomized). For this reason, a 128-bit bidirectional CC can be created between the communication parties. Moreover, these tokens can be invalidated when their associated connection ID is retired through a `RETIRE_CONNECTION_ID` frame.

– **E1.2n1_3 CC with Stateless Reset** - Stateless Reset is an option to the communication peers to reset a connection when a peer does not have access to the connection's state. For this purpose, an UDP datagram is created which only contains a Stateless Reset packet. The packet starts with the fixed bits "11" and ends with an 128-bit `Stateless Reset Token` exchanged previously. The receiving host must end the connection immediately if the `Stateless Reset Token` is correctly identified. The standard demands that an UDP datagram with a Stateless Reset is indistinguishable from regular packets with a short header, so a random value is generated before the `Stateless Reset Token` which is at least 38 bits for the packets with smallest length of 21 B, and if the previously received packet is 43 B or less, the size of the Stateless Reset should be one byte shorter, which means at most 206 random bits. So we can create 38-206 bit unidirectional embedding capacity between the communication endpoints before the current connection is ended.

– **E1.2n1_4 CC with `PATH_CHALLENGE` frame** - An endpoint can send a `PATH_CHALLENGE` frame in two cases: to check reachability of the other endpoint and to validate the path during a connection migration. Each `PATH_CHALLENGE` frame contains a 64-bit `Data` field with a random value. The other endpoint must respond with a `PATH_RESPONSE` frame that contains the same value in the `Data` field. Using this procedure, a 64-bit bidirectional CC can be created between the server and the client.

– **E1.2n1_5 CC with Connection ID values** - When an `Initial` packet is first sent, the client can put random initial values in the `Destination Connection ID` and `Source Connection ID` fields, and in this way it can send at most $2 \times 160 = 320$ bits to the server. This means a 320-bit CC can be created from a client to a server. The server can respond with the `Retry` packet, in which it can send at most a 160-bit value through its `Source Connection ID` field. The only requirement is not to issue the same Connection ID more than once on the same connection. The Connection IDs are generated using an implementation-specific method in clients and servers. Thus, one can create a 160-bit CC from a server to a client.

– **E1.2n1_6 CC with a packet of unsupported version** - If the first packet sent by a client indicates an unsupported version (with a `Version` field that follows the pattern `0x?a?a?a?a`, where '?' is any hexadecimal digit) and if it is large enough to initiate a new connection for any supported version, the server will send a `Version Negotiation` packet. The first bit (`Header Form`), `Destination Connection ID Length`, `Source Connection ID Length`, `Destination Connection ID`, `Source Connection ID` and `Version` fields of a long header packet are version-independent. The first packet for an unsupported version can use different semantics and encodings for any version-specific field, and has randomly generated `Destination Connection ID` and `Source Connection ID` fields. Thus, a client can create a $9,326$-bit uni-directional CC to the server by sending a first packet featuring an unsupported version where all version-specific fields (2-bit `LPT`, `RB` and `PNL` fields) together with the 160-bit `Destination Connection ID` and `Source Connection ID` will contain the secret message. Because the first packet needs to be sent in a datagram with the size of at least 1,200 B (with 20 B IP header, 8 B UDP header and 47 B for the common fields in QUIC packet with a long header), the remaining 1,125 B can be randomly filled with the secret message. So the bitrate (per packet) of this CC is $2 + 2 + 2 + 2 \times 160 + 1,125 \times 8 = 9,326$ bits.

The pattern **E2n1. Network Element Occurrence** with its sub patterns, which encode hidden data by manipulating the position of network elements [Wendzel et al. 2025], can be implemented in several ways:

– **E2n1_1 CC with artificial packet loss modulation** - This bidirectional CC can be implemented by artificial drops of QUIC packets, and the dropped packets will be retransmitted because `ACK` frames will not be received for them. This can be realized, for example, by (not) dropping QUIC packets with an even packet number to signal a one.

The sub pattern **E2.1n1 Network Element Enumeration** specifically modulates the number of elements (e.g., header elements, payload bytes etc.) [Wendzel et al. 2025]:

– **E2.1n1_1 CC with repeating frames** - In QUIC, a proper frame does not cause errors or undesirable side effects when received more than once. Thus, one can either repeat a frame to indicate a secret 0-bit or not repeat a frame for a secret 1-bit.

– **E2.1n1_2 CC with different number of frames in QUIC packet** - A QUIC packet contains at least one frame, so, a one-bit bidirectional CC can be created in this way: a packet with one frame (or an odd number of frames) represents a secret 1-bit, while packets with more than one frame (or an even number of frames) represent a 0-bit;

– **E2.1n1_3 CC with different number of `PADDING` frames in QUIC packet** - A one-bit bidirectional CC can be created as follows. A QUIC packet with an even number of `PADDING` frames can be binary 1, while a packet with an odd number of `PADDING` frames can be a binary 0.

– **E2.1n1_4 CC using UDP datagrams with an `Initial` packet** - Clients and servers that send an `Initial` packet must expand the payload of the UDP datagram that carries it to the size of 1,200 B, and one way to do this is by coalescing the `Initial` packet with invalid packets which are discarded by the receiver. Thus, one can construct a bidirectional CC between client and server using invalid packets to carry the secret message. The bitrate is 1,200 B - 20 B (IP header) - 8 B (UDP header) - (47 + 7) B (estimated minimal `Initial` packet size) = 1,118 B.

E2.2n1 Network Element Positioning modulates the spatial or temporal position of network elements, such as header fields or payload elements, [Wendzel et al. 2025].

– **E2.2n1_1 CC with QUIC packet rate modulation** - A QUIC sender can wait different amounts of time so that several frames can be collected before sending them in a single QUIC packet. So, they can use a different QUIC packet rate for representing secret bits.

– **E2.2n1_2 CC with `ACK` frame rate modulation** - While `Initial` and `Handshake` packets generating `ACK` must be acknowledged immediately, all `0-RTT` and `1-RTT` packets generating `ACK` should be acknowledged within the advertised max_ack_delay. In addition, packets containing only `ACK` frames are not congestion controlled, with the only recommendation that an `ACK` frame should be sent after receiving at least two `ACK` generating packets. So, CS can use different `ACK` frame rates to represent different covert symbols. This can be adjusted by using some rate threshold, for example, half the max_ack_delay, and any data rate above (below) the threshold will be binary 1 (binary 0).

– **E2.2n1_3 CC with `PING` frame occurrence** - `PING` frames are usually used to keep a connection alive by preventing the connection from running when frames are not sent during specified $max\_idle\_timeout$. `PING` frames are acknowledged by the receiver. For example, one can create a one-bit bidirectional CC by sending a `PING` frame in an even second for binary 1 and in an odd second for binary 0.

– **E2.2n1_4 CC with `PADDING` frame position** - `PADDING` frames in QUIC are used to increase an `Initial` packet to the minimum required size or to provide protection against traffic analysis for protected packets. They contain only the `Frame Type` field. A one-bit bidirectional CC can be created in this way: placing only one or more `PADDING` frames at the end of the QUIC packet without their presence in the beginning represents a binary 1, and placing at least one `PADDING` frame at the beginning of the QUIC packet represents a binary 0.

5 Properties of the proposed QUIC covert channels

This section covers the three basic performance properties of covert channels (bitrate, undetectability, and robustness), as well as potential countermeasures.

5.1 Bitrate

For an estimation of how many secret bits are transferred per second, the bitrate is used as a metric. A summary is given in Tab. 2.

- For **E1n1_1**, the maximal number of streams that a given peer can open in the specific connection is limited by the value of $max\_streams$, which is defined as a transport parameter, or is changed by the `MAX_STREAMS` frame. According to the standard, the peer can open only streams with a stream ID smaller than $(max\_streams \times 4 + first\_stream\_id\_of\_type)$. So, if for a time interval of t seconds s streams are open in one direction, one can send s/t bps per direction.
- In QUIC all packets, except `Retry` and `Version Negotiation` packets, have a `Packet Number` field, for **E1n1_2**, if the time interval between two consecutive QUIC packets (different from `Retry` and `Version Negotiation`) to the same destination is t seconds, then the estimated bitrate will be up to $1/t$ bps for AS2. In the Initial space, the `Packet Number` is non-encrypted and can also be used for the AS1 scenario. If for the same t, M different `Initial` packets are sent from CS to different overt destinations, the bitrate would be M/t bps.
- The bitrate of **E1n1_3** is difficult to estimate because different QUIC frames (except 6 of them, namely `PADDING`, `PING`, `RETIRE_CONNECTION_ID`, `PATH_CHALLENGE`, `PATH_RESPONSE` and `HANDSHAKE_DONE` frames) feature a different non-zero number of variable-length encoded fields. The frames `NEW_TOKEN`, `MAX_DATA`, `MAX_STREAMS`, `DATA_BLOCKED`, `STREAMS_BLOCKED` have one variable-length encoded field; `STOP_SENDING`, `MAX_STREAM_DATA`, `STREAM_DATA_BLOCKED`, `NEW_CONNECTION_ID`, and `CONNECTION_CLOSE` have two variable-length encoded fields; `CRYPTO` and `STREAM` have at least two; `RESET_STREAM` has three; and `ACK` frames have at least six variable-length encoded fields. But if we assume that within a time interval of t seconds, all sent QUIC frames carry k variable-length encoded fields (different from `Frame Type`), then the estimated bitrate will be up to k/t bps in AS2. Some of the frames, like the `ACK` and `CRYPTO` frames, can be used in non-encrypted packets, like the `Initial` packet. Thus, in specific cases, this CC can also be used in the AS1 scenario. In that case, for the same t, if M variable-length encoded fields are sent in QUIC frames, the bitrate will be up to M/t bps.
- For **E1n1_4**, the maximal number of bits sent before the completion of the handshake is 392 per direction and connection. Thus, if the time interval between two UDP datagrams with transport parameters is t seconds and more connections are established between the two peers, the resulting bitrate will be $392/t$ bps for the AS2 scenario.
- For **E1.1n1_1** CC, the maximal size of hidden bits per `Version Negotiation` packet sent by the server is 7 bits. So, if the time interval between two `Version Negotiation` packets sent from CS to a specific overt destination in the AS2 scenario is t seconds, then the resulting bitrate will be up to $7/t$ bps. If for the same

t, in the AS1 scenario, M different `Version Negotiation` packets are sent to several overt destinations, then the resulting bitrate will be up to $7\ M/t$ bps.

- For **E1.2n1_1** CC, the maximal size of hidden bits per token value is at least 128 bits. If the time interval between two new `TOKEN` values (sent in the `NEW_TOKEN` frame or `Retry` packet) sent to the same destination is t seconds, then the estimated bitrate will be at least $128/t$ bps in the AS2 scenario. In the AS1 scenario, in the same interval t if there are M different packets with new `TOKEN` values sent from CS to several overt destinations, the resulting bitrate will be $128\ M/t$.
- If the time interval between two sent `Stateless Reset Token` values to the same overt destination is t seconds, then the resulting bitrate for **E1.2n1_2** CC for AS2 will be $128/t$ bps, because the maximal size of hidden bits for the `Stateless Reset Token` is 128. In case of scenario AS1, during the same t, M `Stateless Reset Token` values being sent to several overt destinations, the bitrate will be at least $128\ M/t$ bps.
- The maximum number of hidden bits per Stateless Reset packed in one UDP datagram for **E1.2n1_3** ranges from 38 to 206 bits. If the time interval between two stateless reset UDP datagrams sent to the same overt destination is t seconds, the bitrate will range from $38/t$ to $206/t$ bps for AS2. If we have M UDP datagrams carrying a Stateless Reset to several destinations in AS1 during t seconds, the bitrate will range from $38\ M/t$ to $206\ M/t$ bps.
- For **E1.2n1_4**, one can hide the maximum of 64 bits per `PATH_CHALLENGE` frame. If the time interval between two `PATH_CHALLENGE` frames is t seconds, then the estimated bitrate will be $64/t$ bps.
- The maximum number of hidden bits per `Initial` packet for **E1.2n1_5** is 320 bits (sent from a client), and the maximum number of hidden bits per `Retry` packet is 160 bits (sent from a server). Thus, if the time interval between two `Initial` packets sent from the client to a specific server, or two `Retry` packets sent from the server to a specific client is t seconds, then the bitrate will be $320/t$ bps and $160/t$ bps, respectively, in one direction for scenario AS2. For AS1, if during t seconds we have M overt destinations, then the bitrate will be $320\ M/t$ bps and $160\ M/t$ bps in one direction, respectively.
- For **E1.2n1_6**, when the first packet sent by a client indicates an unsupported version, the maximum amount of hidden bits (due to UDP) is $9,326$ bits. If the time interval between two packets that indicate an unsupported version of QUIC is t seconds, then the resulting bitrate will be $9,326/t$ bps per server (AS2), or for the AS1 scenario $9,326\ M/t$ bps, if the client uses M different servers.
- For **E2n1_1**, if the time interval between two QUIC packets in the same number space is t seconds, the resulting bitrate will be $1/t$ bps, in each direction.
- For **E2.1n1_1**, we can assume that copies of the frames are sent in the same QUIC packet as the original frame. Also, we can assume that only for one frame, one or more copies are sent in the QUIC packet. So, if the time interval between two QUIC packets is t seconds, the resulting bitrate will be $1/t$ bps.
- Similarly, for **E2.1n1_2** and **E2.1n1_3**, if the time interval between two QUIC packets is t seconds, the resulting bitrate will be $1/t$ bps.

– For **E2.1n1_4**, the maximal size of hidden bits per UDP datagram within an `Initial` packet is $1,118B = 8,944$ bits. If the time interval between two UDP datagrams with an `Initial` packet inside sent to a specific overt destination is t seconds, then the resulting bitrate will be $8,944/t$ bps for AS2. For AS1, if in t seconds, M UDP datagrams with an `Initial` packet inside are sent to several overt destinations, the bitrate will be $8,944\ M/t$ bps.

– For **E2.2n1_1**, because sending a binary 0 or 1 depends on the QUIC packet rate and some predefined rate threshold f_t that corresponds to the period of t seconds, if n QUIC packets to a specific overt destination are sent in t seconds, the resulting bitrate will be n/t bps for AS2. If during t seconds, m QUIC packets are sent to several destinations, the resulting bitrate will be m/t bps for AS1.

– In **E2.2n1_2**, only acknowledgments of ack-eliciting `0-RTT` and `1-RTT` packets are used, and knowing that they should be acknowledged within the advertised max_ack_delay milliseconds, CS and CR can prearrange some $threshold\_delay$ milliseconds. If in max_ack_delay milliseconds, n acknowledgments are received, the bitrate obtained will be $n * 10^3/max\_ack\_delay$ bps.

– In **E2.2n1_3**, if the time interval between two `PING` packets is t seconds, the resulting bitrate will be $1/t$ bps.

– For **E2.2n1_4**, if the time interval between two `Initial` packets is t seconds, the resulting bitrate will be $1/t$ bps.

CC	Description	Application Scenario	Bitrate bps for AS1	Bitrate bps for AS2
E1n1_1	Bidirectional CC with stream identifiers	AS2	-	up to s/t
E1n1_2	Bidirectional CC with `Packet numbers`	AS1, AS2	up to M/t	up to $1/t$
E1n1_3	Bidirectional CC with variable-length encoded fields	AS1, AS2	up to M/t	up to k/t
E1n1_4	Bidirectional CC with transport parameters	AS2	-	up to $392/t$ in one direction
E1.1n1_1	Unidirectional CC from a server as a CS with Unused field in `Version Negotiation` packet	AS1, AS2	up to $7 * M/t$	up to $7/t$
E1.2n1_1	Unidirectional CC from a server as a CS with `Token` value	AS1, AS2	up to at least $128 * M/t$	up to at least $128/t$
E1.2n1_2	Unidirectional CC with `Stateless Reset Token` value	AS1, AS2	up to $128 * M/t$	up to $128/t$
E1.2n1_3	Unidirectional CC with Stateless Reset	AS1, AS2	from $38 * M/t$ to $206 * M/t$	from $38/t$ to $206/t$
E1.2n1_4	Bidirectional CC with `PATH_CHALLENGE` frame	AS2	-	up to $64/t$
E1.2n1_5	Bidirectional CC with Connection ID values	AS1, AS2	up to $160 * M/t$ from server and $320 * M/t$ from client	up to $160/t$ from server and $320/t$ from client
E1.2n1_6	Unidirectional CC from a client as a CS with a packet of unsupported version	AS1, AS2	up to $9326 * M/t$	up to $9326/t$
E2n1_1	Bidirectional CC with artificial packet loss modulation	AS2	-	up to $1/t$
E2.1n1_1	Bidirectional CC with repeating frames	AS2	-	up to $1/t$
E2.1n1_2	Bidirectional CC with different number of frames in QUIC packet	AS2	-	up to $1/t$
E2.1n1_3	Bidirectional CC with different number of `PADDING` frames in QUIC packet	AS2	-	up to $1/t$
E2.1n1_4	Bidirectional CC with UDP datagrams with `Initial` packet	AS1, AS2	up to $8944 * M/t$	up to $8944/t$
E2.2n1_1	Bidirectional CC with QUIC packet rate modulation	AS1, AS2	up to M/t	up to n/t
E2.2n1_2	Bidirectional CC with QUIC frame rate modulation	AS2	-	up to $n * 10^3/max\_ack\_delay$
E2.2n1_3	Bidirectional CC with `PING` frame occurrence	AS2	-	up to $1/t$
E2.2n1_4	Bidirectional CC with `PADDING` frame position	AS2	-	up to $1/t$

Table 2: Summary of the novel QUIC covert channels

5.2 Robustness

A computer network can interfere with covert communication in different ways. For example, it might introduce packet losses and delays. This interference can result in losing, flipping, or mixing secret bits. The robustness of the CC is its ability to protect the covert message from possible network interference.

In the AS1 application scenario, any CR is an intermediate device through which all communication must pass from the network where the CS is located, such as a router or firewall. Alternatively, it eavesdrops the network traffic from the intermediate network devices. For this reason, CR often resides in the same network as CS or has a direct link to that network, resulting in only small network interference. In the AS2 application scenario, CRs can be placed on intermediate devices that are on the same network as the overt destinations or as the CS. In addition, CRs can intercept network traffic from intermediate network devices. For this type of CRs, network interference can be more significant.

Network delay, especially the one that rapidly changes over time, is particularly an issue for timing CCs, because it can change the packet or frame rate during the time, or the time of occurrence of some specific event. Because of this, the received bits can be incorrectly interpreted. On the other hand, storage CCs are robust against constant network delay.

All QUIC packets, except `Retry`, `Version Negotiation` and Stateless Reset, have their own `Packet number`, which increases by at least 1 (separately in all 3 packet spaces, i.e., Initial, Handshake and Application data). This feature provides robustness of most of the storage CCs against network delay, except those that uses `Retry`, `Version Negotiation` and Stateless Reset packets (E1n1_1, E1.2n1_1 with Retry, E1.2n1_3, and E1.2n1_6). Even in the case of packet losses, QUIC implementations that increase the `Packet number` exactly by 1 (e.g., Quiche4j implementation[1]), provide robustness for these CCs, because loss of any packet with secret bits will be recognized.

Another interference in the ongoing clandestine communication can be the possible modification of the QUIC packets performed by the active warden, located on the path between CS and CR. We will look at different types of wardens, according to knowledge of covert communication (presence unaware, presence aware, partially aware and fully aware) [Mazurczyk et al. 2019]. In the first two cases, the warden does not know any specifics about the CC, and it must perform some anomaly detection techniques – we will cover this scenario in the following section. In the last two cases, the warden can modify existing QUIC packets, destroying the covert communication, or inject legitimately appearing but falsified data to interfere with the CC. This can have effects only in the case of the AS1 scenario, because for the AS2 scenario, this is very difficult, even impossible to do, due to the fact that all QUIC traffic is encrypted, including covert parts.

5.3 Undetectability

Undetectability is another crucial metric for CCs, which is the inability of third parties to differentiate covert from overt communication.

If the timing CC sends a secret message with a low bitrate, it is very difficult to distinguish it from normal traffic. But at higher bitrate one can use variations of epsilon similarity and/or compressibility score techniques, suggested by Cabuk et al. [Cabuk et al. 2009] for detection.

[1] https://kachayev.github.io/quiche4j/

All storage CCs that can be applied in the second scenario AS2 only (E2n1_1, E2.2n1_4, E2.1n1_1, E2.1n1_2, E2.1n1_3, E1n1_1, E1n1_4 and E1.2n1_4) produce traffic which is indistinguishable from the normal network traffic, because both are encrypted in QUIC.

Detection of E1.2n1_6 CC with an unsupported version packet is easier, because these packets are untypical in QUIC communications. So, their presence might be an indicator for covert communication, especially if they appear in some number.

Covert channels that use random values in QUIC packets, specifically E1.2n1_1, E1.2n1_2, E1.2n1_3, and E1.2n1_5, can be detected using different statistical methods that compare the resulting value distribution from the typical distribution of random values, generated by usual cryptographic pseudo-random number generators, used by operating systems and QUIC implementations, and by finding anomalies. However, our experiments with E1.2n1_1 CC (Section 6) demonstrated that if the covert message is encrypted before embedding, covert and overt traffic are indistinguishable. This should also be true for E1.2n1_2, E1.2n1_3, and E1.2n1_5.

In more detail, there are two main categories of detection tests: shape tests and regularity tests. Shape tests analyze traffic characteristics using first-order statistics such as mean, variance, and distribution. Regularity tests, on the other hand, assess traffic patterns based on second- or higher-order statistics, e.g., correlations in the data. Examples of shape tests are the Kolmogorov-Smirnov test and the entropy test, while examples of regularity tests are the use of regularity measure, the relative entropy test and the corrected conditional entropy test [Gianvecchio & Wang 2010]. They can also be applied on timing channels.

E1n1_2 CC can be detected by using compressibility score technique applied to the difference between consecutive `Packet numbers` in the same packet space. The same technique but applied to the mapping of variable-length encoded fields in QUIC packets into the minimal number of bytes (0) or not (1) can be used for detection of E1n1_3 CC. Different regularity tests are also candidates for a possible detection method for these two channels.

5.4 Potential Countermeasures

Different countermeasures that can detect, limit, distort and/or eliminate covert communication can be used against a covert channel [Zander et al. 2007]. It is interesting that some security-conscious organizations today explicitly block QUIC to maintain network visibility and security control. On the Internet one can find recommendations from different vendors for their customers to block UDP ports 80 and 443 on organizational firewalls (or other security appliances), followed by the fallback to TCP and enabling SSL/TLS inspection to maintain full visibility and control[2]. Another possibility is to block QUIC traffic in web browsers. Clearly, if network administrators apply similar security policies, the QUIC CCs will be fully eliminated. The following countermeasures are for the case where QUIC traffic is not blocked or filtered.

Two timing channels E2.2n1_1 and E2.2n1_2 can be eliminated through traffic normalizers by removing the timing behavior of QUIC packets by flattening the QUIC packet data rates and `ACK` frame rates (see [Mazurczyk et al. 2016] for a coverage of the methodology). For example, traffic normalizers can periodically measure the time between consecutive QUIC packets, then calculate the maximal value, and use this as a

[2] https://www.zscaler.com/blogs/product-insights/quic-secure-communication-protocol-shaping-future-of-internet

time to send each subsequent QUIC packet. The calculation of the maximum time can be performed periodically by the normalizer, and the correct maximum can be adjusted to the new value. Another possibility is for the traffic normalizer to use a fixed value, close to max_ack_delay, for each subsequent QUIC packet. On the negative side, both measures can have a negative impact on the legitimate QUIC performance.

In addition, traffic normalizers can distort covert traffic by applying random delays between consecutive QUIC packets or `ACK` frames in the case of scenario A2. Because `PING` frames are embedded into QUIC packets, the same applies also for CC E2.2n1_3. Both countermeasures are difficult to apply for E2.2n1_1 in the case of the A1 scenario, because the CR is located on the same network as the CS, or at the exit of the network.

The E1.1n1_1 CC can be eliminated using a traffic normalization that alters the 7-bit `Unused` field in a `Version Negotiation` packet to normalized values, specified in advance, for example, all 0s, or better all 1s, because in some cases MSB should be 1. The same holds for the E2.1n1_4 CC, where invalid packets that follow an `Initial` packet in a QUIC packet are normalized to predefined values, for example, all 0s.

Because QUIC traffic is end-to-end encrypted, the usual packet inspection methods cannot be applied, so middleboxes on the route can implement this kind of measures instead.

Storage CCs that can be applied in the second scenario AS2 only (E2n1_1, E2.2n1_4, E2.1n1_1, E2.1n1_2, E2.1n1_3, E1n1_1, E1n1_4 and E1.2n1_4), are considered challenging to detect, limit, or eliminate, without having crude effects on legitimate flows. The reason is that the covert traffic is encrypted in the same manner as the overt traffic, so, no network device can examine this traffic.

5.5 QUIC Uniqueness as a Steganographic Carrier

The fact that there is barely a possibility for firewalls and security appliances to decrypt and inspect QUIC traffic is one reason that makes QUIC uniquely susceptible or resilient to covert channels, not only those that are using specific QUIC properties, but also CCs that are using different network protocols above QUIC. QUIC hides much of the data and details about the network protocols that deploy its services. For example, if we compare HTTP/2 over TLS and HTTP/3, in the first case, some metadata can still be inferred and some headers may be logged by middleboxes, which is not a case with HTTP/3, because its entire traffic is encrypted by QUIC.

In addition, because QUIC has built-in mechanisms to handle network changes and disruption of established connections without having to restart the connection process, it provides better resilience and maintaining performance for the serviced protocols, but also for the carried CCs. If covert communication is carried over slow or unreliable Internet connections, QUIC offers faster, more robust, and reliable transmission. In QUIC connections are identified by a set of `Connection IDs`, not IP addresses. This together with other built-in connection migration capabilities offers an ability to seamlessly transition between networks without reconnecting, or continuation of the connection, even in the case where some middlebox influences breaking the connection of the CC from different reasons. This is ideal for mobile and vehicle applications, but also for CCs when some warden tries to disrupt them.

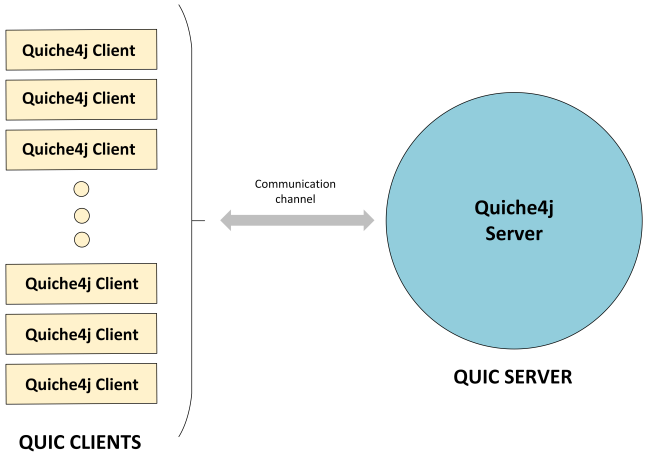


Figure 5: Experimental scenario

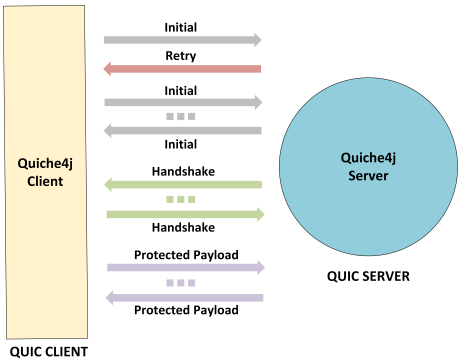


Figure 6: Communication between QUIC Client and QUIC Server

6 Exemplary Covert Channel Evaluation

6.1 Experimental Scenario and Design

According to our research we noticed that there is a lack of public datasets that we could use for experimental purposes. For this reason, we created our own dataset[3]. We implement the AS1 experimental scenario of Fig. 4a. It consists of a QUIC server (as a CS) that sends data to multiple QUIC clients (10 clients in our case as CRs). The scenario can be seen in Fig. 5. Detailed communication between the QUIC server and one QUIC client is shown in Fig. 6.

For our experimental scenario, we created three variants:

– Scenario with legitimate traffic

[3] https://doi.org/10.5281/zenodo.15076824

- Scenario with an implemented covert channel that is embedding ASCII covert messages
- Scenario with implemented an covert channel that is embedding Advanced Encryption Standard (AES)-encrypted covert messages

In the scenario with legitimate traffic, all 10 clients first establish a connection to the server. After that, each of the clients sends a request to the server which responds with sending random data as an HTTP response. Each client, independent of the other clients, sends an HTTP request in a random time interval between 0 and 60 seconds.

For the second variant of our experimental scenario, we implemented the covert channel with the value of the `Retry Token` field inside a Retry packet (E1.2n1_1 CC). In particular, the clients send requests to the server in a random time interval between 0 and 60 seconds independently, and the server responds with a legitimate HTTP response or with an HTTP response with ASCII covert message embedded in the `Retry Token` field of the QUIC packet. So not all HTTP responses from the server have hidden messages inside them. The server randomly chooses for which clients it will send a response with a hidden message. Considering that the hidden message is distributed across different server responses to different clients, we need to introduce a sequence numbering of the carriers of the message. So, the server as an CS uses the value of the SCID field in the QUIC packet as a sequence number. The covert receiver eavesdrops network traffic from the server and learns the secret message.

The third scenario is similar to the second. The only difference is that an AES encrypted messages are sent over the channel instead of ASCII ones.

6.2 Discussion of the Experimental Results

Undetectability. Our analysis builds upon the work of Cabuk et al. [Cabuk et al. 2009] as mentioned above in Section 5.3. They demonstrate the efficacy of compressibility scores in detecting timing network covert channels. They have shown that the inter arrival times of traffic containing covert communication often exhibits a higher degree of regularity than the timings of legitimate traffic, resulting in better compressibility. We adopted this approach for the QUIC covert channel by using the `RETRY_TOKEN` field of QUIC packets as inputs for the compression algorithm. Afterwards, we removed the first 56 (static) bytes from the `RETRY_TOKEN` field and then analyzed the subsequent byte sequences. Next, we combined the `RETRY_TOKEN` field of all packets from a window of $n = 200$ packets and then calculated the compressibility. Similarly to Zillien et al. [Zillien & Wendzel 2023], who analyzed the performance of the compressibility score and other detection methods more thoroughly, we used the Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) curves to assess the detection performance of our approach.

Fig. 7a shows this visualization for the two covert channel scenarios presented above. The ROC shows the ability to distinguish legitimate traffic from traffic containing covert channel communications by plotting the achieved false positive rate against the true positive rate of the detection algorithm. We can see that the detection for “Legitimate vs. ASCII” is perfect with an AUC of 1.0, as shown by the orange line. That means that the compressibility values of legitimate and ASCII covert channel traffic are completely distinct. The “Legitimate vs. AES” scenario on the other hand (blue line) shows an AUC of 0.52 with a near diagonal curve, which denotes an almost random decision between legitimate and AES covert channel traffic. This is a “worst case” result for the detector,

as it can not distinguish between AES-covert channel traffic and legitimate traffic at all. A histogram plot in Fig. 7b shows the relationship between the compressibility scores of the different scenarios quite well. We can see that legitimate and AES-CC traffic is almost identical in distribution, while ASCII-CC traffic has a distinctly different distribution. This again shows the perfect detection of the ASCII-CC and the almost perfect stealthiness of the AES-CC.

To further explain the compressibility ratios, we analyzed the **bigram frequencies** of the `RETRY_TOKEN` field. In this case, "bigrams" refer to the hex-string representation of single bytes in the `RETRY_TOKEN` field (that means `0x00` to `0xFF`). Fig. 8 shows plots for legitimate, AES-CC and ASCII-CC traffic. As expected, legitimate traffic shows an even distribution of all 255 values (Bigram Index). The ASCII-CC traffic shows a clear "structure" and the AES-CC traffic is again evenly distributed. As the visualization is normalized to a total of 1.0, please note the different scales in Y-Axes for the ASCII scenario. With these results, we can again see that the AES variant of the covert channel is a significant improvement in stealthiness over the simple ASCII version.

Bitrate. Regarding bitrate, in our scenario where around 50% of the HTTP responses are used to hide data, we have captured AES-CC traffic for 46,616.664 seconds, with 2,079,872 hidden bits transferred, which corresponds to an average bitrate of 44.6 bps. Similarly, for saved ASCII-CC traffic, we have 2,062,592 hidden bits sent in 46,617.740 seconds, which corresponds to an average bitrate of 44.2 bps.

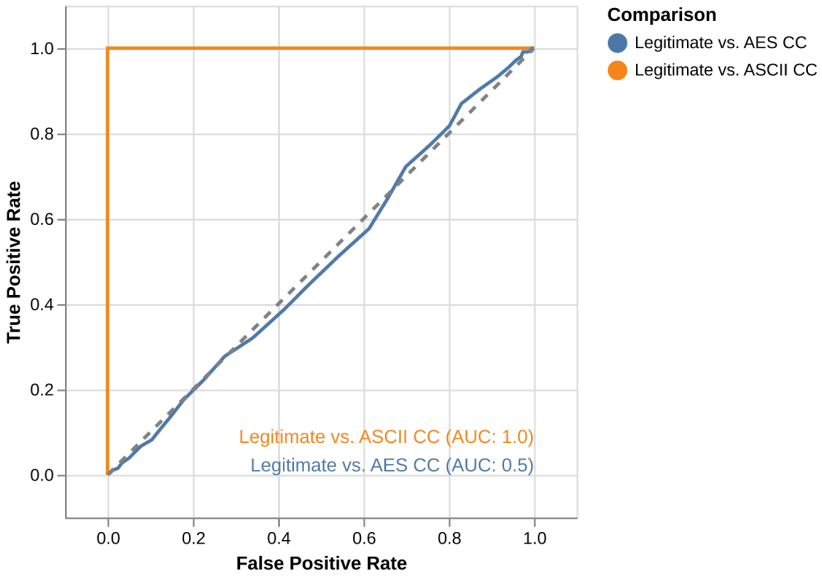
The possible bitrate is much larger. To observe how large this bitrate can be, we analyze NGINX web servers, which are the most used web servers in the world, being used by 33.8% of all websites[4]. The number of requests per second that the web server can handle depends on different factors, e.g., hardware, server configuration, network bitrate, etc. The NGINX worker process processes incoming HTTP requests. The recommendation is to run one worker process per CPU core, for the most efficient use of hardware resources[5], and this can be done by setting the `auto` parameter in the `worker_processes` directive. Another directive –`worker_connections` –specifies how many connections each worker process can handle[6]. The theoretical CC bitrate (bps) for different NGINX configurations (with 1, 2, 4 and 8 CPU cores, and `worker_connections` equal to 10 and 100) is given in Table 3. The assumption is that all HTTP responses carry hidden bits and each second `worker_connections` concurrent connections are established with new clients.

CPU cores	`worker_connections = 10`	`worker_connections = 100`
1	1,280	12,800
2	2,560	25,600
4	5,120	51,200
8	10,240	102,400

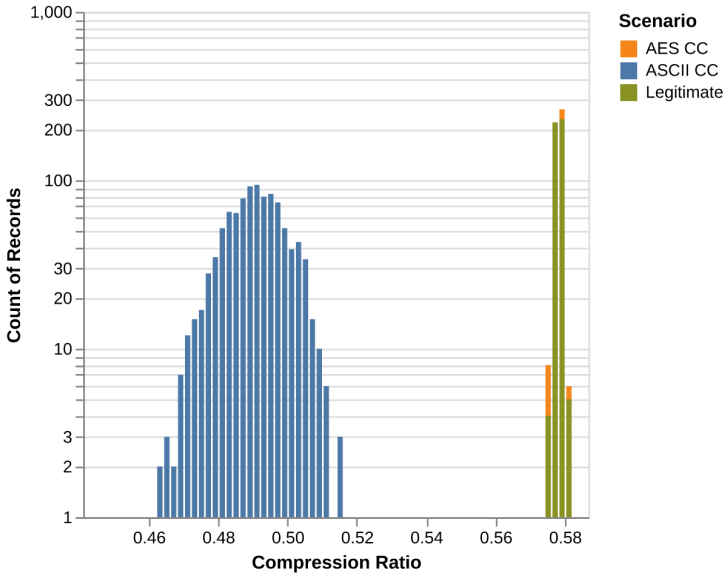
Table 3: Theoretical CC bitrate (bps) for different NGINX configurations

Interested parties (CS and CR) need to agree on the percentage of HTTP responses that will be used for their clandestine communication, as a trade-off between undetectability

[4] https://w3techs.com/technologies/overview/web_server
[5] https://blog.nginx.org/blog/inside-nginx-how-we-designed-for-performance-scale
[6] https://faun.pub/how-nginx-handles-thousands-of-concurrent-requests-199e54974b69



(a)



(b)

Figure 7: Determining Detectability Through the Compression Ratio
(a) ROC Curves for Covert Channel Detection
(b) Histogram Representation of the Compression Ratio

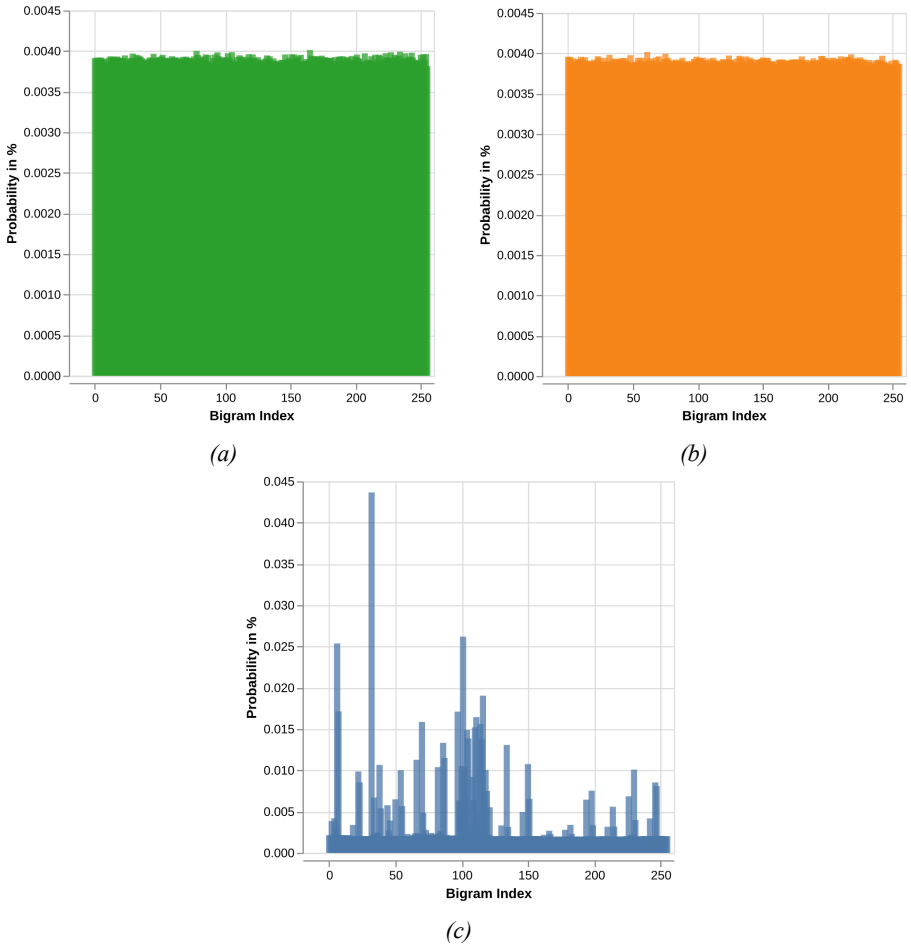


Figure 8: Histogram Representation of the Bigram Probability
(a) Legitimate; (b) AES CC; (c) ASCII CC

and bitrate. A higher percentage leads to larger bitrate, but also more possibilities for detection.

Robustness. The evaluated E1.2n1_1 CC is a storage CC, so the network delay, particularly constant, does not influence the reception of the secret message. The reason is that CR can still follow the sequential order of the message due to the SCID field in the QUIC packet, which is used as a sequence number.

Network interference in the form of packet loss, flipping, or mixing packet bits can lead to loss, flipping, or mixing of secret bits. The effects may be different. If the QUIC packet with bits from the CC is lost, then 128 bits of the message are lost. Even one bit lost or flipped in the `Retry Token` field inside a Retry packet that belongs to the AES-CC can lead to impossible or wrong decryption of all 128 bits carried. In the case of ASCII-CC, one flipped bit can lead to one wrong character in the secret message, while

one lost bit can lead to one or many wrong characters, depending on its position. If the lost bit is at the end of the token, only one character can be wrong, while if its position is at the beginning of the token, all characters can be wrong.

Comparison with QuicCourier [Huang et al. 2025]. QuicCourier allows CRs to access websites through proxies while clandestinely receiving secret bits from the proxy node, which means that CS is located at the intermediate node, while CR is located at the overt destination in the same network. This is a totally different scenario from our scenarios, where CS is located on the overt sender, while CRs are located on the intermediate devices on the same or different network.

QuicCourier adds secret data in 3 different ways: by packet padding – adding the data at the end of the QUIC packet payload; packet insertion – adding the data in front of the QUIC packet payload; and packet replacement – replacing the original QUIC packet payload with the secret data. This means that in all three cases, the size of secret data per QUIC packet is limited by the size of the Maximum Path Transmission Unit [Iyengar & Thomson 2021], which should be at least 1,200 B. This leaves 1,200 B – 20 B (IPv4 header) – 8 B (UDP header) - 22 B (minimal QUIC header for packets with a short header)= 1,150 B = 9,200 bits for hiding data with packet replacement, and much less for the other 2 methods. If we compare only the QuicCourier packet replacement method, our E1.2n1_6 CC has a higher possible bitrate, and our E2.1n1_4 CC has a little bit less bitrate per QUIC packet.

7 Conclusion

We conducted an in-depth pattern-based analysis of the QUIC protocol for network covert channels. We have shown that QUIC-based covert channels can be realized in twenty different ways by using most of the known hiding patterns. For that purpose, we present two different scenarios that can be used for selected scenarios: censorship circumvention, data leakage and/or data exfiltration. We described the performance characteristics of these channels, their undetectibility, bitrate, and robustness, together with possible countermeasures, and compared them with the existing state-of-the-art steganography framework QuicCourier. We implemented one covert channel on a testbed as a proof-of-concept, for which we also evaluated its detectability using the compressibility score and bitrate, but also discussed its robustness.

This work demonstrates the uniqueness of QUIC as a steganographic carrier, not only for CCs built on the specific properties of QUIC but also for CCs built on network protocols above QUIC. The main reasons are the built-in connection migration capabilities, and the impossibility for intermediate devices to inspect QUIC packets etc.

In future work, we plan to implement more of the covered channels presented, so that their detectability can be evaluated.

References

[Bishop 2022] Bishop, M. (Ed.): “HTTP/3. RFC 9114,” June 2022. https://www.rfc-editor.org/info/rfc9114.

[Cabuk et al. 2009] Cabuk, S., Brodley, C. E., Shields, C. (2009). IP covert channel detection. ACM Transactions on Information and System Security (TISSEC), 12(4), 1-29.

[CDNetworks 2023] CDNetworks: “What is QUIC? How Does It Boost HTTP/3?”. https://www.cdnetworks.com/media-delivery-blog/what-is-quic/. (Accessed on 21.04.2023)

[Chatzoglou et al. 2023] Chatzoglou, E., Kouliaridis, V., Karopoulos, G., Kambourakis, G. (2023). Revisiting QUIC attacks: A comprehensive review on QUIC security and a hands-on study. International Journal of Information Security, 22(2), 347-365.

[Dimitrova & Mileva 2017] Dimitrova, B., Mileva, A. (2017). Steganography of hypertext transfer protocol version 2 (http/2). Journal of Computer and Communications, 5, 98-111.

[Duke 2023] Duke, M.: "QUIC Version 2," May 2023. https://www.rfc-editor.org/info/rfc9369.

[Feamster et al. 2002] Feamster, N., Balazinska, M., Harfst, G., Balakrishnan, H., Karger, D. (2002). Infranet: Circumventing web censorship and surveillance. In 11th USENIX Security Symposium (USENIX Security 02).

[Gianvecchio & Wang 2010] Gianvecchio, S., Wang, H. (2010). An entropy-based approach to detecting covert timing channels. IEEE Transactions on Dependable and Secure Computing, 8(6), 785-797.

[Huang et al. 2025] Huang, J., Liu, W., Liu, G., Gao, B., Nie, F. (2025). QuicCourier: Leveraging the Dynamics of QUIC-Based Website Browsing Behaviors Through Proxy for Covert Communication. IEEE Transactions on Dependable and Secure Computing.

[Iyengar & Swett 2015] Iyengar, J., Swett, I.: "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2", June 2015, https://datatracker.ietf.org/doc/html/draft-tsvwg-quic-protocol-00.

[Iyengar & Thomson 2021] Iyengar, J., Thomson, M.: "QUIC: A UDP-Based Multiplexed and Secure Transport" May 2021. https://www.rfc-editor.org/rfc/rfc9000.html.

[Iyengar & Swett 2021] Iyengar, J., Swett, I.: "QUIC Loss Detection and Congestion Control" May 2021. https://www.rfc-editor.org/rfc/rfc9002.html

[Iyengar & Thomson 2023] Iyengar, J., Thomson, M.: "QUIC: A UDP-Based Multiplexed and Secure Transport," May. 2025. https://greenbytes.de/tech/webdav/draft-ietf-quic-transport-16.html (Accessed on 21.06.2025)

[Joarder & Fung 2022] Joarder, Y. A., Fung, C. (2022, October). A Survey on the Security Issues of QUIC. In 2022 6th Cyber Security in Networking Conference (CSNet) (pp. 1-8). IEEE.

[Lampson 1973] Lampson, B. W. (1973). A note on the confinement problem. Communications of the ACM, 16(10), 613-615.

[Mazurczyk & Szczypiorski 2008] Mazurczyk, W., Szczypiorski, K. (2008, June). Covert Channels in SIP for VoIP signalling. In International Conference on Global e-Security (pp. 65-72). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Mazurczyk & Kotulski 2006] Mazurczyk, W., Kotulski, Z. (2006). New security and control protocol for VoIP based on steganography and digital watermarking," Annales UMCS Informatica AI, 5, 417-426, 2006.

[Mazurczyk et al. 2016] Mazurczyk, W., Wendzel, S., Zander, S., Houmansadr, A., Szczypiorski, K. (2016). Information hiding in communication networks: fundamentals, mechanisms, applications, and countermeasures. John Wiley Sons.

[Mazurczyk et al. 2019] Mazurczyk, W., Wendzel, S., Chourib, M., Keller, J. (2019). Countering adaptive network covert communication with dynamic wardens. Future Generation Computer Systems, 94, 712-725.

[Mileva & Panajotov 2014] Mileva, A., Panajotov, B. (2014). Covert channels in TCP/IP protocol stack-extended version. Central European Journal of Computer Science, 4(2), 45-66.

[Mileva et al. 2018] Mileva, A., Velinov, A., Stojanov, D. (2018).New Covert Channels in Internet of Things. In: The 12th International Conference on Emerging Security Information, Systems and Technologies - SECURWARE 2018, September 16-20, 2018, Venice, Italy.

[Murdoch 2007] Murdoch, S. J. (2007). Covert channel vulnerabilities in anonymity systems (No. UCAM-CL-TR-706). University of Cambridge, Computer Laboratory.

[Nawrocki et al. 2021] Nawrocki, M., Hiesgen, R., Schmidt, T. C., Wählisch, M. (2021, November). Quicsand: quantifying quic reconnaissance scans and dos flooding events. In Proceedings of the 21st ACM internet measurement conference (pp. 283-291).

[The Chromium Projects 2023] The Chromium Projects:"QUIC, a multiplexed transport over UDP". https://www.chromium.org/quic/. (Accessed on 23.03.2023).

[Thomson 2021] Thomson, M.: "Version-Independent Properties of QUIC" May 2021. https://www.rfc-editor.org/rfc/rfc8999.html.

[Thomson & Turner 2021] Thomson, M., Turner, S.: "Using TLS to Secure QUIC" May 2021. https://www.rfc-editor.org/rfc/rfc9001.html.

[Thomson & Benfield 2022] Thomson, M., Benfield, C.: "HTTP/2. RFC9113," June 2022. https://www.rfc-editor.org/info/rfc9113.

[Wendzel & Keller 2014] Wendzel, S., Keller, J. (2014). Hidden and under control: A survey and outlook on covert channel-internal control protocols. annals of telecommunications-annales des télécommunications, 69(7), 417-430.

[Wendzel et al. 2025] Wendzel, S., Caviglione, L., Mazurczyk, W., Mileva, A., Dittmann, J., Krätzer, C., Lamshöft, K., Vielhauer, C., Hartmann, L., Keller, J., Neubert, T. Zillien. S.: "A Generic Taxonomy for Steganography Methods," ACM Computing Surveys, 57(9), 2025.

[Wendzel et al. 2015] Wendzel, S., Zander, S., Fechner, B., Herdin, C. (2015). Pattern-based survey and categorization of network covert channel techniques. ACM Computing Surveys (CSUR), 47(3), 1-26.

[W3Techs 2024] W3Techs: "Usage statistics of QUIC for websites". https://w3techs.com/technologies/details/ce-quic. (Accessed on 16.11.2024).

[W3Techs 2024] W3Techs: "Usage statistics of HTTP/3 for websites". https://w3techs.com/technologies/details/ce-http3. (Accessed on 16.11.2024).

[Zander et al. 2007] Zander, S., Armitage, G., Branch, P. (2007). A survey of covert channels and countermeasures in computer network protocols. IEEE Communications Surveys Tutorials, 9(3), 44-57.

[Zillien & Wendzel 2023] Zillien, S., Wendzel, S. (2023). Weaknesses of popular and recent covert channel detection methods and a remedy. IEEE Transactions on Dependable and Secure Computing, 20(6), 5156-5167.