Workgroup: Network Working Group
Internet-Draft: draft-hurst-sip-quic-00
Published: 25 October 2022
Intended Status: Experimental
Expires: 28 April 2023
Authors: S. Hurst
BBC Research & Development

SIP-over-QUIC: Session Initiation Protocol over QUIC Transport

Abstract

This document describes a mapping of Session Initiation Protocol (SIP) semantics over QUIC Transport. It allows the creation, modification and termination of media sessions with one or more participants, possibly carried over the same QUIC transport connection, using RTP/AVP directly, or some mixture of both.

SIP-over-QUIC enables a more efficient use of network resources by introducing field compression to the header fields carried in SIP transactions.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://probable-train-1d24d093.pages.github.io/draft-hurst-sip-quic.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-hurst-sip-quic/.

Source for this draft and an issue tracker can be found at https://github.com/bbc/draft-hurst-sip-quic.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 28 April 2023.

Copyright Notice

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [RFC3261] is widely used for managing media sessions over the Internet. Examples of these media sessions include Internet telephony services, video conferencing and live streaming of media.

[SIP2.0] uses whitespace-delimited text fields to convey SIP messages in a similar format to HTTP/1.1 [HTTP1.1], and may optionally be transported over TLS (so-called "SIPS"). SIP-over-QUIC, as defined by this document, uses a binary framing layer carried over QUIC streams and is protected by the mandatory TLS encryption afforded by the QUIC transport connection.

> **Author's Note:** A future optional extension may introduce the ability to carry SIP messages in QUIC Datagrams [QUIC-DATAGRAMS].

1.1. Conventions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the variable-length integer encoding from Section 16 of [QUIC-TRANSPORT].

Packet and frame diagrams in this document use the format described in Section 1.3 of [QUIC-TRANSPORT].

1.2. Definitions

The following terms are used:

abort: An abrupt termination of a connection or stream, possibly due to an error condition.

endpoint: Either the client or server of the connection.

connection: A transport-layer connection between two endpoints using QUIC as the transport protocol.

connection error: An error that affects the entire SIP-over-QUIC connection.

frame: The smallest unit of communication on a stream in SIP-over-QUIC, consisting of a header and a variable-length sequence of bytes structured according to the frame type.

Protocol elements called "frames" exist in both this document and [QUIC-TRANSPORT]. Where frames from [QUIC-TRANSPORT] are referenced, the frame name will be prefaced with "QUIC". For example, "QUIC CONNECTION_CLOSE frames". References without this preface refer to frames defined in Section 7.2.

peer: An endpoint. When discussing a particular endpoint, "peer" refers to the endpoint that is remote to the primary subject of discussion.

receiver: An endpoint that is receiving frames.

sender: An endpoint that is transmitting frames.

SIP/2.0: The SIP/2.0 specification as described in RFC3261 [SIP2.0].

SIP-over-QUIC connection: A QUIC connection where the negotiated application protocol is SIP-over-QUIC.

stream: A bidirectional or unidirectional bytestream provided by the QUIC transport. All streams within an SIP-over-QUIC connection can be considered "SIP-over-QUIC streams", but multiple stream types are defined within SIP-over-QUIC.

stream error:
An application-level error on the individual stream.

transport client: The endpoint that initiates a SIP-over-QUIC connection.

transport server: The endpoint that accepts a SIP-over-QUIC connection.

The terms "call", "dialog", "header", "header field", "header field value", "initiator", "invitee", "message", "method", "proxy server", "request", "(SIP) transaction", "user agent client" and "user agent server" are defined in Section 6 of [SIP2.0].

2. SIP-over-QUIC Protocol Overview

SIP-over-QUIC provides a transport for SIP semantics using the QUIC transport protocol and an internal framing layer inspired by [HTTP3].

Once a user agent client knows that a user agent server supports SIP-over-QUIC, it opens a QUIC connection. QUIC provides protocol negotiation, stream-based multiplexing, and flow control services to the SIP-over-QUIC transaction layer above. SIP transactions are multiplexed across QUIC streams as described in Section 2 of [QUIC-TRANSPORT]. Each request and response message pair in a SIP-over-QUIC transaction consumes a single QUIC stream.

Within each QUIC stream, the basic unit of SIP-over-QUIC communication is a frame as described in Section 7. Each frame type serves a different purpose. The HEADERS and DATA frames form the basis of the offer/answer transaction model described in [RFC3264] and are described in Section 3.2.

In [SIP2.0], some header fields may be compressed by using abbreviated versions. In SIP-over-QUIC, all request and response header fields are compressed for transmission using [QPACK], in which header fields may be mapped to indexed values, or literal values may be encoded using a codepoint in a Huffman table. [QPACK] uses two tables for its indexed values: the static table is predefined with common SIP header fields and values, and the dynamic table can be used to encode frequently used header fields in a SIP-over-QUIC connection to reduce repetition. Because [QPACK]'s static table is designed to work with [HTTP3], this specification replaces the default static table defined in Appendix A of [QPACK] with the one in Appendix B.

2.1. QUIC Transport

SIP-over-QUIC relies on QUIC version 1 as the underlying transport. The use of other QUIC transport versions with SIP-over-QUIC **MAY** be defined by future specifications.

QUIC connections are established as described in [RFC9000]. During connection establishment, SIP-over-QUIC support is indicated by selecting the ALPN token "sips/quic" in the TLS handshake. Support for other application-layer protocols **MAY** be offered in the same handshake.

> **Author's Note:** The original intention of this document was to define the next major version of SIP, i.e. SIPv3. Therefore, the future ALPN token could be sips/3 or s3 (similarly to h2/h3).

QUIC version 1 uses TLS version 1.3 or greater as its handshake protocol. SIP-over-QUIC user agents **MUST** support a mechanism to indicate the target host to the server during the TLS handshake. If the target destination user agent server is identified by a domain name [RFC8499], clients **MUST** send the Server Name Indication ([SNI]) TLS extension unless an alternative mechanism to indicate the target host is used.

SIP-over-QUIC messages are carried in reliable QUIC streams; therefore, the SIP-over-QUIC protocol defined by this document is a reliable SIP transport. Thus, client and server transactions using SIP-over-QUIC for transport **MUST** follow the procedures and timer values for reliable transports as defined in [SIP2.0].

2.1.1. Draft Version Identification

> **RFC Editor's Note:** Please remove this section prior to publication of a final version of this document.

Only implementations of the final, published RFC can identify themselves as "sips/quic". Until such an RFC exists, implementations **MUST NOT** identify themselves using this string. Implementations of draft versions of the protocol **MUST** add the string "-h" and the corresponding draft number to the identifier. For example, draft-hurst-sip-quic-00 is identified using the string "sips/quic-h00".

Non-compatible experiments that are based on these draft versions **MUST** append the string "-" and an experiment name to the identifier. For example, an experimental implementation based on draft-hurst-sip-quic-00 which uses QUIC datagrams instead of QUIC streams to carry SIP messages might identify itself as "sips/quic-h00-datagrams". Note that any label **MUST** conform to the "token" syntax defined in Section 5.6.2 of [HTTP-SEMANTICS]. Experimenters are encouraged to coordinate their experiments.

2.2. Connection Reuse

SIP-over-QUIC connections are persistent across multiple request-response transactions. A SIP-over-QUIC connection **MAY** also be shared by multiple concurrent dialogs, with each dialog individually identified by the Call-ID header and tag= parameters on the To and From headers.

3. Expressing SIP Semantics Over QUIC Transport

3.1. QUIC Clients and Servers

This specification introduces two new terms: "transport client" to denote the host that initiates the QUIC transport connection for exchanging SIP-over-QUIC messages, and "transport server" as its peer. These terms are orthogonal to SIP's concepts of user agent client (UAC) and user agent server (UAS): transport clients and servers may take on either role in the SIP-over-QUIC connection.

3.2. SIP Transaction Framing

SIP-over-QUIC transactions begin with a request message sent by a UAC on a request stream, which is a bidirectional QUIC stream; see Section 5. If the user agent client making the request is accessing the transport connection from the transport client endpoint, then the request stream is carried on a client-initiated bidirectional QUIC stream. If the user agent client making the request is accessing the transport connection from the transport server endpoint, then the request stream is carried on a server-initiated bidirectional QUIC stream.

Each SIP-over-QUIC transaction has exclusive use of a request stream. Only one request is made per request stream. The UAS sends zero or more provisional responses on the same stream as the request, followed by one or more final responses. See Section 7.2 of [SIP2.0] for a description of provisional and final responses.

On a given request stream, receipt of multiple requests **MUST** be treated as malformed.

A SIP message (request or response) consists of:

1. the header section, including message control data, sent in SIP-over-QUIC as a single HEADERS frame,

2. optionally, the message body, if present, sent in SIP-over-QUIC as a series of DATA frames.

Headers are described in Section 7.3 of [SIP2.0]. Message bodies are described in Section 7.4 of [SIP2.0].

Receipt of an invalid sequence of frames **MUST** be treated as a connection error of type SIP_FRAME_UNEXPECTED. In particular, a DATA frame received before any HEADERS frame is considered invalid. Other frame types, especially unknown frame types, **MAY** be permitted, subject to their own rules, see Section 9.

The HEADERS frame might reference updates to the QPACK dynamic table. While these updates are not directly part of the message exchange, they **MUST** be received and processed before the message can be consumed.

After sending a request, the UAC **MUST** close the stream for sending (see Section 3.4 of [QUIC-TRANSPORT]). After sending the last final response, the UAS **MUST** close the stream for sending. At this point, the QUIC stream is fully closed.

When a stream is closed, this indicates the completion of the SIP-over-QUIC transaction. If a stream terminates without enough of the request to provide a complete response, the UAS **SHOULD** abort the stream with the error code SIP_REQUEST_INCOMPLETE.

3.2.1. Request Cancellation and Rejection

Once a request stream has been opened, the request **MAY** be cancelled by either endpoint for the reasons given in Section 9 of [SIP2.0]. Cancellations are categorised as either graceful or abrupt. An endpoint **MAY** abruptly cancel any request by resetting the stream using a RESET_STREAM frame and aborting the reception of further data on that stream using a STOP_SENDING frame as described in Section 2.4 of [QUIC-TRANSPORT].

The UAC **SHOULD** gracefully cancel requests if the response is no longer of interest by using the CANCEL frame. For example, where a UAC is attempting to reach a user at multiple endpoints, and has already received a final response from one endpoint that it is satisfied with.

UACs **MAY** use the error code SIP_REQUEST_CANCELLED to abruptly cancel requests. Upon receipt of this error code, a UAS **MAY** abruptly terminate the response using the error code SIP_REQUEST_REJECTED if no processing was performed. A UAC **MUST NOT** use the SIP_REQUEST_REJECTED error code, except when the corresponding UAS has requested closure of the request stream with this error code.

A UAS receiving a CANCEL request (not frame) **MUST** respond to the request immediately with a 405 Method Not Allowed error as described in Section 21.4.6 of [SIP2.0]. A user agent server receiving a

CANCEL frame for a stream that has not been opened **MUST** be treated as a connection error of type SIP_CANCEL_FRAME_CLOSED.

> **Author's Note:** This is because the CSeq header has been removed, so the CANCEL request method cannot be used.

The UAS cancels requests if they are unable or choose not to respond. UAS cancellations are always abrupt cancellations. When a UAS abruptly cancels a request without performing any application processing, the request is considered "rejected". In this case, the UAS **SHOULD** abort its response stream with the error code SIP_REQUEST_REJECTED. (In this context, "processed" means that some data from the request stream was passed to some higher layer of software that might have taken some action as a result.) The UAC **MAY** treat a request rejected by the UAS as though it had never been sent at all, and may retry the request later.

A UAS **MUST NOT** use the SIP_REQUEST_REJECTED error code for requests that were partially or fully processed. When a UAS abandons a response after partial processing, it **SHOULD** abort its response stream with the error code SIP_REQUEST_CANCELLED.

3.2.2. Malformed Requests And Responses

A malformed request or response is one that is a sequence of syntactically valid SIP-over-QUIC frames but that is invalid due to:

- an invalid sequence of SIP-over-QUIC frames, such as a DATA frame preceding a HEADERS frame,
- the presence of prohibited header fields or pseudo-header fields in a HEADERS frame,
- the absence of mandatory pseudo-header fields in a HEADERS frame,
- invalid values for pseudo-header fields in a HEADERS frame,
- pseudo-header fields after header fields in a HEADERS frame,
- the inclusion of uppercase header field names in a HEADERS frame,
- the inclusion of invalid characters in field names or values in a HEADERS frame.

A request or response that is defined as having content when it contains a Content-Length header field (see Section 18.3 of [SIP2.0]) is malformed if the value of the Content-Length header field does not equal the sum of the received DATA frame lengths.

Intermediaries that process SIP-over-QUIC request or response messages (such as a proxy server) **MUST NOT** forward a malformed request or response. Malformed requests or responses that are detected **MUST** be treated as a stream error of type SIP_MESSAGE_ERROR.

A UAS **MAY** respond to a malformed request, indicating the error prior to closing or resetting the stream.

A UAC **MUST NOT** accept a malformed response.

3.3. SIP Header Fields

SIP messages carry metadata as a series of key-value pairs called "SIP header fields"; see Section 7.3 of [SIP2.0]. For a listing of registered SIP header fields, see the "Session Initiation Protocol (SIP) Parameters - Header Fields Registry" maintained at https://www.iana.org/assignments/sip-parameters/sip-parameters.xhtml#sip-parameters-2.

3.3.1. SIP-over-QUIC Header Compression

The abbreviated forms of SIP header fields described in Section 7.3.3 of [SIP2.0] **MUST NOT** be used with SIP-over-QUIC. Instead, header fields (including the control data present in the header section) are compressed an decompressed using the [QPACK] codec.

A SIP-over-QUIC implementation **MAY** impose a limit on the maximum size of the encoded field section it will accept for an individual SIP message using the SETTINGS_MAX_FIELD_SECTION_SIZE parameter. Unlike HTTP, there is no response code in SIP for the size of a header block being too large. If a user agent receives an encoded field section larger than it has promised to accept, it **MUST** treat this as stream error of type SIP_HEADER_TOO_LARGE, and discard the response.

Section 4.2 of [QPACK] describes the definition of two unidirectional stream types for the encoder and decoder streams. The values of these stream types are identical when used with SIP-over-QUIC, see Section 5.2.

The static table defined in Appendix A of [QPACK] is designed for use with HTTP and, as such, contains header fields that are of little interest to SIP endpoints. Appendix B in this document defines a replacement static table that **MUST** be used by SIP-over-QUIC transport clients and servers.

To bound the memory requirements of the decoder for the QPACK dynamic table, the decoder limits the maximum value the encoder is

permitted to set for the dynamic table capacity, as specified in Section 3.2.3 of [QPACK]. The dynamic table capacity is determined by the value of the SETTINGS_QPACK_MAX_TABLE_CAPACITY parameter sent by the decoder. Use of the dynamic table can be disabled by setting this value to zero. If both endpoints disable use of the dynamic table, then the endpoints **SHOULD NOT** open the encoder and decoder streams.

When the dynamic table is in use, a QPACK decoder may encounter an encoded field section that references a dynamic table entry that it has not yet received, because QUIC does not guarantee order between data on different streams. In this case, the stream is considered "blocked" as described in Section 2.1.2 of [QPACK]. The HTTP/3 setting SETTINGS_QPACK_BLOCKED_STREAMS is replicated as a SIP-over-QUIC parameter, set by the recipient, determines the maximum number of streams that are allowed to be "blocked" by pending dynamic table updates. If a decoder encounters more blocked streams than it promised to support, it **MUST** treat this as a connection error of type SIP_HEADER_DECOMPRESSION_FAILED.

Stream blocking can be avoided by sending Huffman-encoded literals instead of updating the QPACK dynamic table.

3.3.2. SIP Control Data

SIP-over-QUIC employs a series of pseudo-header fields where the field name begins with the : character (ASCII 0x3a). These pseudo-header fields convey message control data, which replaces the Request-Line described in Section 7.1 of [SIP2.0].

Pseudo-header fields are not SIP header fields. Endpoints **MUST NOT** generate pseudo-header fields other than those defined in this document. However, an extension could negotiate a modification of this restriction; see Section 9.

Pseudo-header fields are only valid in the context in which they are defined. Pseudo-header fields defined for requests **MUST NOT** appear in responses; pseudo-header fields defined for responses **MUST NOT** appear in requests. Pseudo-header fields **MUST NOT** appear in trailer sections. Endpoints **MUST** treat a request or response that contains undefined or invalid pseudo-header fields as malformed.

All pseudo-header fields **MUST** appear in the header section before regular header fields. Any request or response that contains a pseudo-header field that appears in a header section after a regular header field **MUST** be treated as malformed.

3.3.2.1. Request Pseudo-header fields

The following pseudo-header fields are defined for requests:

*":method": Contains the SIP method. See Section 6 to understand SIP-over-QUIC-specific usages of SIP methods.

*":request-uri": Contains the SIPS URI as described in Section 19.1 of [SIP2.0].

All SIP-over-QUIC requests **MUST** include exactly one value for the :method and :request-uri pseudo-header fields. The SIP-Version element of the Request-Line structure in Section 7.1 of [SIP2.0] is omitted, and all SIP-over-QUIC requests implicitly have a protocol version of "2.0".

A SIP request that omits any mandatory pseudo-header fields or contains invalid values for those pseudo-header fields is malformed.

3.3.2.2. Response Pseudo-header fields

For responses, a single ":status" pseudo-header field is defined that carries the SIP status code, see Section 7.2 of [SIP2.0].

All SIP-over-QUIC responses **MUST** include exactly one value for the ":status" pseudo-header field. The SIP-Version and Reason-Phrase elements of the Status-Line structure in Section 7.2 of [SIP2.0] are omitted, and all SIP-over-QUIC responses implicitly have a protocol version of "2.0". If it is required, for example to provide a human readable string of a received status code, the Reason-Phrase can be inferred from the list of reason phrases accompanying the status codes listed in Section 21 of [SIP2.0].

3.3.3. Via Transport Parameter

The Via header field in SIP messages carry a transport protocol identifier. This document defines the value "QUIC" to be used for SIP-over-QUIC requests over QUIC transport.

The updated ABNF (Augmented Backus-Naur Form) [RFC5234] for this parameter is the following:

```
transport  =/  "QUIC"
```

A full example Via: header is as follows:

```
Via: SIP/2.0/QUIC wxp6O3dffes2.example.com;branch=z9hG4bKXG1gNkhgOiNR
```

3.3.4. SIP URI Transport Parameter

This document defines the value "quic" as the transport parameter value for a SIP-over-QUIC URI [RFC3986] where the transport mechanism used for sending SIP messages will be QUIC transport, extending the parameter names defined in Section 19.1.1 of [SIP2.0].

The updated ABNF for this parameter is the following:

```
transport-param  =/  "transport=" "quic"
```

3.3.5. Transaction Sequence Number

SIP-over-QUIC endpoints **MUST NOT** use the CSeq header field (see Section 20.16 of [SIP2.0]). The correct order of SIP-over-QUIC transactions is instead inferred from the QUIC stream identifier as described in Section 5. Intermediaries that forward SIP-over-QUIC messages to SIP sessions running over other transports are responsible for defining the value carried in the CSeq header field for those messages, and for mapping those values back to the correct SIP-over-QUIC request stream in the opposite direction.

3.4. Connection Keep-Alive

SIP-over-QUIC endpoints may keep their QUIC connection active and open by sending periodic PING frames to defer the QUIC idle timeout as described in Section 10.1.2 of [QUIC-TRANSPORT].

4. Compatibility With Earlier SIP Versions

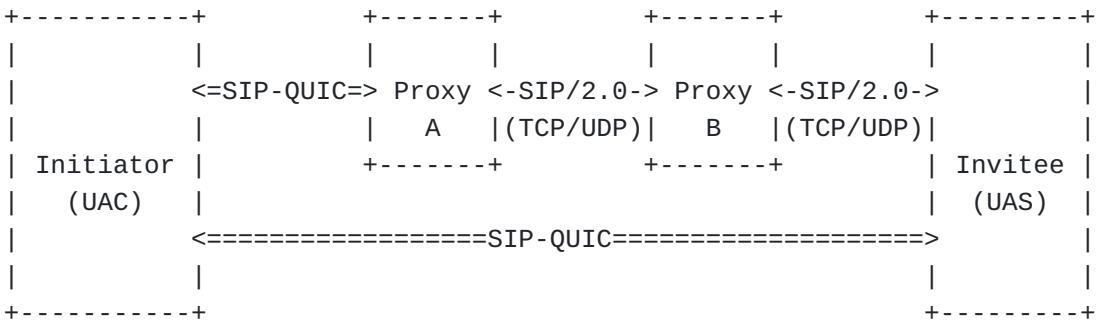

Figure 1: Example showing mixed SIP transports

In the above example, the Initiator, Invitee and the proxy server identified as "Proxy A" all support SIP-over-QUIC, but the proxy server identified as "Proxy B" only supports SIP/2.0 over TCP/TLS and UDP. When Proxy A attempts to connect to Proxy B, it may have previous knowledge of the lack of support for SIP-over-QUIC on Proxy B, or either of the DNS SRV record [RFC2782] or DNS service binding

record [DNS-SVCB] may have indicated that the server only supports SIPS services over TCP, thereby implying SIP/2.0.

If Proxy B only supports unencrypted SIP over UDP, then Proxy A **MUST NOT** forward messages from the secure SIP-over-QUIC over an unencrypted protocol because this could constitute a downgrade attack. Instead, if the designated Invitee cannot be contacted by means other than via Proxy B, then Proxy A **MUST** return a response of 502 Bad Gateway to the initiator for that transaction.

When initiating direct communication with an invitee after the conclusion of the initial INVITE transaction, SIP-over-QUIC **SHOULD** be used if:

- The DNS SRV record for the SIPS URI indicates that the invitee supports SIPS over UDP, or
- The DNS service binding record for the SIPS URI indicates the sips/quic ALPN token as described in Section 2.1, or
- The Contact: header field indicates a SIPS URI carrying the "quic" transport parameter as described in Section 3.3.4.

4.1. Transactions

In [SIP2.0], messages pertaining to a given SIP transaction are identified as such using the branch parameter on the Via: header fields carried in requests and responses. In SIP-over-QUIC, individual transactions are tracked using the QUIC streams that are used to carry them. SIP-over-QUIC endpoints **MAY** omit this parameter. For intermediaries converting between SIP-over-QUIC and SIP sessions running over other transport protocols, these endpoints **SHOULD** insert missing branch parameters. To avoid leaking details of the QUIC transport connection, these converted branch parameters **MUST NOT** be textual representations of the stream IDs used to carry a given transactions, or any other representation that could be used to infer stream IDs that have been used in a given QUIC transport connection.

4.2. Dialogs

In [SIP2.0], dialogs are tracked by use of the Call-ID: header field and the tag= parameter on the To: and From: header fields. The current document does not introduce any additional means for tracking dialogs, and as such the Call-ID: and tag= values **MUST** continue to be used in SIP-over-QUIC.

5. Stream Mapping and Usage

A QUIC stream provides reliable and in-order delivery of bytes on that stream, but makes no guarantees about order of delivery with regard to bytes on other streams. The semantics of the QUIC stream layer is opaque to the SIP framing layer. The transport layer buffers and orders received stream data, exposing a reliable byte stream to the application. Although QUIC permits out-of-order delivery within a stream, SIP-over-QUIC does not make use of this feature.

QUIC streams can be either unidirectional, carrying data only from initiator to receiver, or bidirectional, carrying data in both directions. In the context of this specification, bidirectional streams are used to convey SIP-over-QUIC request and response messages; unidirectional streams are used only for controlling the SIP-over-QUIC session itself. A bidirectional stream ensures that the response can be readily correlated with the request. These streams are referred to as request streams.

[SIP2.0] is designed to run over unreliable transports such as UDP. Since QUIC guarantees reliability, some of the features of SIP/2.0 are not required. User agents **MUST NOT** send the CSeq header field in requests or responses because the messages are already associated with a QUIC stream. Intermediaries that convert SIP-over-QUIC to other transport protocols when forwarding messages are responsible for handing the mapping of the CSeq header field to individual transactions.

> **Author's note:** The author invites feedback as to whether the **MUST NOT** in relation to the CSeq header could be relaxed to a **SHOULD NOT**, or whether there is a valid use case that I have not identified that means this restriction should be relaxed even further.

If the [QPACK] dynamic table is used, then the unidirectional encoder and decoder streams described in Section 4.2 of [QPACK] will be in operation in a SIP-over-QUIC connection.

5.1. Bidirectional Streams

Bidirectional QUIC streams are used for SIP requests and responses. These streams are referred to as request streams.

5.2. Unidirectional Streams

SIP-over-QUIC makes use of unidirectional QUIC streams. The purpose of a given unidirectional stream is indicated by a stream type, which is sent as a variable-length integer at the start of the

stream. The format and structure of data that follows this integer is determined by the stream type.

```
Unidirectional Stream Header {
  Stream Type (i),
}
```

Figure 2: Unidirectional Stream Header

One stream type is defined in this document: the control stream. In addition, the HTTP/3 stream types defined by Section 4.2 of [QPACK] are mapped to the same values in SIP-over-QUIC (0x2 for the encoder stream and 0x3 for the decoder stream).

In addition, the stream type value of 0x4 is reserved by this document for future use as the media stream.

Each SIP-over-QUIC user agent **MUST** create at least one unidirectional stream for the SIP-over-QUIC control stream. If the QPACK dynamic table is used, then each endpoint will open two additional unidirectional streams each. Other extensions might require further unidirectional streams. Therefore, the transport parameters sent by both endpoints **MUST** allow the peer to create at least three unidirectional streams. These transport parameters **SHOULD** also provide at least 1,024 bytes of flow-control credit to each unidirectional stream.

If the stream header indicates a stream type that is not supported by the recipient, the receiver **MUST** abort reading the stream, discard incoming data without further processing, and reset the stream with the SIP_STREAM_CREATION_ERROR error code. The recipient **MUST NOT** consider unknown stream types to be a connection error of any kind.

Since certain stream types can affect connection state, a recipient user agent **SHOULD NOT** discard data from incoming unidirectional streams prior to reading the stream type.

Implementations **SHOULD** wait for the reception of a SETTINGS frame describing what stream types their peer user agent supports before sending streams of that type. Implementations **MAY** send stream types that do not modify the state or semantics of existing protocol components before it is known whether the peer user agent supports them, but **MUST NOT** send stream types that do (such as QPACK).

A sender can close or reset a unidirectional stream unless otherwise specified. A receiver **MUST** tolerate unidirectional streams being closed or reset prior to the reception of the unidirectional stream header.

5.2.1. Control Streams

A control stream is indicated by a stream type of 0x00. Data on this stream consists of SIP-over-QUIC frames, as defined in Section 7.

Each SIP-over-QUIC user agent **MUST** initiate a single control stream at the beginning of the connection and send its SETTINGS frame as the first frame on this stream. If the first frame of the control stream is any other frame type, this **MUST** be treated as a connection error of type SIP_MISSING_SETTINGS. Only one control stream is permitted per user agent; receipt of a second stream claiming to be a control stream **MUST** be treated as a connection error of type SIP_STREAM_CREATION_ERROR.

The control stream **MUST NOT** be closed by the sender, and the receiver **MUST NOT** request that the sender close the control stream. If either control stream is closed at any point, this **MUST** be treated as a connection error of type SIP_CLOSED_CRITICAL_STREAM.

Connection errors are described in Section 8.

Because the contents of the control stream are used to manage the behaviour of other streams, user agents **SHOULD** provide enough flow-control credit to keep the peer's control stream from becoming blocked.

6. SIP Methods

The REGISTER, INVITE, ACK and BYE methods as described in [SIP2.0] continue to operate in SIP-over-QUIC as they do in SIP running over other transport protocols.

The CANCEL method **MUST NOT** be used in SIP-over-QUIC. If a SIP-over-QUIC request needs to be cancelled, the CANCEL frame **SHOULD** be used instead, or the stream for that request reset using the QUIC RESET_STREAM frame (Section 19.4 of [QUIC-TRANSPORT]). Note that SIP-over-QUIC messages in flight at the time may still arrive on a stream before the cancellation is received and processed by the peer.

> **Author's note:** I have not done a comprehensive review of all SIP/2.0 extensions and their applicability to this document, so I invite feedback on any other methods that may be problematic.

7. SIP Framing Layer

SIP-over-QUIC frames are carried on QUIC streams, as described in Section 5. SIP-over-QUIC defines a single stream type: the Request Stream. This section describes SIP-over-QUIC frame formats; see Table 1 for an overview.

Frame	Request Stream	Control Stream	Section
DATA	Yes	No	Section 7.2.1
HEADERS	Yes	No	Section 7.2.2
CANCEL	No	Yes	Section 7.2.3
SETTINGS	No	Yes	Section 7.2.4

Table 1

Note that, unlike QUIC frames, SIP-over-QUIC frames can span multiple QUIC or UDP packets.

7.1. Frame Layout

All frames have the following format:

```
SIP-over-QUIC Frame Format {
  Type (i),
  Length (i),
  Frame Payload (...)
}
```

Figure 3: SIP-over-QUIC Frame Format

A frame includes the following fields:

- *Type: A variable-length integer that identifies the frame type.
- *Length: A variable-length integer that describes the length in bytes of the Frame Payload.
- *Frame Payload: A payload, the semantics of which are determined by the Type field.

Each frame's payload **MUST** contain exactly the fields identified in its description. A frame payload that contains additional bytes after the identified fields of a frame payload that terminates before the end of the identified fields **MUST** be treated as a connection error of type SIP_FRAME_ERROR.

When a stream terminates cleanly, if the last frame on the stream was truncated, this **MUST** be treated as a connection error of type SIP_FRAME_ERROR. Streams that terminate abruptly may be reset at any point in a frame.

7.2. Frame Definitions

7.2.1. DATA

DATA frames (type=0x00) are only sent on Request Streams. The frame payload carried in the Data field conveys an arbitrary, variable-

length sequences of bytes associated with a SIP-over-QUIC request or response message.

DATA frames **MUST** be associated with a SIP request or response.

```
DATA Frame {
  Type (i) = 0x00,
  Length (i),
  Data (..)
}
```

Figure 4: DATA Frame

7.2.2. HEADERS

HEADERS frames (type=0x01) are only sent on Request Streams. They are used to carry the collection of SIP header fields associated with a SIP request or response message, as described in Section 3.3. The payload, carried in Encoded Field Section, is encoded using [QPACK].

```
HEADERS Frame {
  Type (i) = 0x01,
  Length (i),
  Encoded Field Section (..)
}
```

Figure 5: HEADERS Frame

7.2.3. CANCEL

The CANCEL frame (type=0x02) is only sent on a Control Stream and informs the receiver that its peer user agent does not intend to do any further processing on the message carried by the associated bidirectional stream ID. If the receiver has already completed the processing for the message, sent the response and closed the sending end of the stream, it **MUST** disregard this frame.

> **Author's Note:** Remove the length from this frame type as the stream ID field is self-describing.

```
CANCEL Frame {
  Type (i) = 0x02,
  Length (i),
  Stream ID (i)
}
```

Figure 6: CANCEL Frame

Senders **MUST NOT** send this frame with a stream ID that has not been acknowledged by its peer. A user agent that receives a CANCEL frame with a stream ID that has not yet been opened **MUST** respond with a connection error of type SIP_CANCEL_STREAM_CLOSED error.

7.2.4. SETTINGS

The SETTINGS frame (type=0x04) is only sent on a Control Stream. It conveys configuration parameters that affect how SIP-over-QUIC user agents communicate, such as preferences and constraints on peer behaviour. The parameters always apply to an entire SIP-over-QUIC connection, never to a single transaction. A SETTINGS frame **MUST** be sent as the first frame of each Control Stream by each peer user agent, and it **MUST NOT** be sent subsequently. If a SIP-over-QUIC user agent receives a second SETTINGS frame on the control stream, or any other stream, the user agent **MUST** respond with a connection error of type SIP_FRAME_UNEXPECTED.

SETTINGS parameters are not negotiated; they describe characteristics of the sending user agent that can be used by the receiving user agent. However, a negotiation can be implied by the use of SETTINGS: each user agent uses SETTINGS to advertise a set of supported values. Each user agent combines the two sets to conclude which choice will be used. SETTINGS does not provide a mechanism to identify when the choice takes effect.

Different values for the same parameter can be advertised by the two user agents.

The same parameter **MUST NOT** occur more than once in the SETTINGS frame. A receiver **MAY** treat the presence of duplicate setting identifiers as a connection error of type SIP_SETTINGS_ERROR.

The payload of a SETTINGS frame consists of zero or more parameters. Each parameter consists of a parameter identifier and a value, both encoded as QUIC variable-length integers.

```
Parameter {
  Identifier (i),
  Value (i)
}

SETTINGS Frame {
  Type (i) = 0x04,
  Length (i),
  Parameter (..) ...
}
```

Figure 7: SETTINGS Frame

An implementation **MUST** ignore any parameter with an identifier it does not understand.

7.2.4.1. Defined SETTINGS Parameters

The following parameters are defined in SIP-over-QUIC:

SETTINGS_QPACK_MAX_TABLE_CAPACITY (0x01):	The default value is zero. See Section 3.3 for usage.
SETTINGS_MAX_FIELD_SECTION_SIZE (0x06):	The default value is unlimited. See Section 3.3 for usage.
SETTINGS_QPACK_BLOCKED_STREAMS (0x07):	The default value is zero. See Section 3.3 for usage.

8. Error Handling

When a request cannot be completed successfully, or if there is an issue with the underlying QUIC stream, QUIC allows the application protocol to abruptly reset that stream and communicate a reason (see Section 2.4 of [QUIC-TRANSPORT]. This is referred to as a "stream error". A SIP-over-QUIC implementation can decide to close a QUIC stream and communicate the type of error. The wire encoding of error codes is defined in Section 8.1. Stream errors are distinct from SIP status codes that indicate error conditions. Stream errors indicate that the sender did not transfer or consume the full request or response message, while SIP status codes indicate the result of a request that was successfully received and processed by the recipient.

If an entire connection needs to be terminated, QUIC similarly provides mechanisms to communicate a reason (see Section 5.3 of [QUIC-TRANSPORT]). This is referred to as a "connection error". Similar to stream errors, a SIP-over-QUIC implementation can terminate a QUIC connection and communicate the reason using an error code from Section 8.1.

Although called a "stream error", this does not necessarily indicate a problem with either the implementation or the connection as a whole. Streams **MAY** also be reset if the result of a SIP response is no longer of interest to the user agent client, see Section 3.2.1.

Section 9 specifies that extensions may define new error codes without negotiation. Use of an unknown error code or a known error code in an unexpected context **MUST** be treated as equivalent to SIP_NO_ERROR.

8.1. SIP-over-QUIC Error Codes

The following error codes are defined for use when abruptly terminating streams, aborting reading of streams, or immediately closing SIP-over-QUIC connections.

SIP_NO_ERROR (0x0300): No error. This is used when the connection or stream needs to be closed, but there is no error to signal.

SIP_GENERAL_PROTOCOL_ERROR (0x0301): Peer violated protocol requirements in a way that does not match a more specific error code or endpoint declines to use a more specific error code.

SIP_INTERNAL_ERROR (0x0302): An internal error has occurred in the SIP-over-QUIC stack.

SIP_STREAM_CREATION_ERROR (0x0303): The endpoint detected that its peer created a stream that it will not accept.

SIP_CLOSED_CRITICAL_STREAM (0x0304): A stream required by the SIP-over-QUIC connection was closed or reset.

SIP_FRAME_ERROR (0x0305): A frame that fails to satisfy layout requirements or with an invalid size was received.

SIP_FRAME_UNEXPECTED (0x0306): A frame was received that was not permitted in the current state or on the current stream.

SIP_CANCEL_FRAME_CLOSED (0x0307): A CANCEL frame was received that referenced an unknown stream ID.

SIP_SETTINGS_ERROR (0x0309): An endpoint detected an error in the payload of a SETTINGS frame.

SIP_MISSING_SETTINGS (0x030a): No SETTINGS frame was received at the beginning of the control stream.

SIP_REQUEST_INCOMPLETE (0x030d): An endpoint's stream terminated without containing a fully formed request.

SIP_REQUEST_REJECTED (0x030b): A server rejected a request without performing any application processing.

(0x030c):
SIP_REQUEST_CANCELLED
(0x030c): The request or its response is cancelled.

SIP_MESSAGE_ERROR **(0x030e):** A SIP message was malformed and cannot be processed.

SIP_HEADER_COMPRESSION_FAILED **(0x0310):** The QPACK decoder failed to interpret an encoded field section and is not able to continue decoding that field. section

SIP_HEADER_TOO_LARGE **(0x0311):** The received encoded field section was larger than the receiver has previously promised to accept. See Section 3.3.

9. Extensions to SIP-over-QUIC

SIP-over-QUIC permits extension of the protocol. Within the limitations described in this section, protocol extensions can be used to provide additional services or alter any aspect of the protocol. Extensions are effective only within the scope of a single SIP-over-QUIC connection.

This applies only to the protocol elements defined in this document. This does not affect the existing options for extending SIP, such as defining new methods, status codes or header fields.

Extensions are permitted to use new frame types (Section 7.2), new settings (Section 7.2.4.1), new error codes (Section 8.1), or new stream types (Section 5).

> **RFC Editor's Note:** Establish registries for frame types, settings, error codes and stream types.

Implementations **MUST** ignore unknown or unsupported values in all extensible protocol elements. This means that any of these extension points can be safely used by extensions without prior arrangement or negotiation. However, where a known frame type is required to be in a specific location, such as the SETTINGS frame (see Section 5.2.1), an unknown frame type does not satisfy that requirement and **SHOULD** be treated as an error.

Extensions that could change the semantics of existing protocol components **MUST** be negotiated before being used. For example, an extension that allows the multiplexing of other protocols such as media transport protocols over bidirectional QUIC streams **MUST NOT** be used until the peer user agent has given a positive signal that this is acceptable.

This document does not mandate a specific method for negotiating the use of any extension, but it notes that a parameter (Section 7.2.4.1) could be used for that purpose. If both peer user agents set a value that indicates willingness to use the extension, then the extension can be used. If a parameter is used in this way, the default value **MUST** be defined in such a way that the extension is disabled if the setting is omitted.

10. Future Carriage of Media Sessions

> **Author's Note:** This section is intended to foster discussion around how the QUIC transport connection established and used by SIP-over-QUIC could be also used for carriage of media.
>
> **RFC Editor's Note:** Please remove this section prior to publication of a final version of this document.

Future versions of this specification may include support for carrying media sessions within the same QUIC transport connection as SIP-over-QUIC, with the intention being that they will be negotiated using the SDP offer/answer mechanism.

There already exists several attempts to define carriage of media over QUIC transport, such as [QRT], [RTP-over-QUIC], [QuicR-Arch], [RUSH] and [Warp].

In the case of media carried in QUIC datagrams, a user agent cannot propose sending media using this mechanism unless its peer has indicated its support for receiving datagrams by means of the max_datagram_frame_size parameter as described in Section 3 of [QUIC-DATAGRAMS].

In the case of media carried in QUIC streams, if the media streams are transmitted using unidirectional streams, then new stream types will need to be defined. This document reserves the stream type value 0x04 for this, see Section 5.2. In the unlikely case where media streams are to be transmitted using bidirectional streams, the stream type mechanism will need to be extended to cover bidirectional streams, because this specification currently assumes that SIP-over-QUIC messages have exclusive use of the bidirectional streams.

10.1. Carriage Of RTP In A QUIC Transport Session

Both [QRT] and [RTP-over-QUIC] define ways to carry RTP and RTCP messages over QUIC DATAGRAM frames. With SIP and SDP already closely aligned with RTP media sessions, adapting SIP-over-QUIC to coexist

within the same QUIC transport connection as RTP/RTCP would save at least one network round trip.

* QRT only defines a way to carry RTP and RTCP in QUIC DATAGRAM frames.

* RTP-over-QUIC defines a way to carry RTP and RTCP over unidirectional QUIC STREAM frames as well as QUIC DATAGRAM frames.

In addition, QRT attempts to define SDP attributes to allow the negotiation of QRT sessions in SIP. [SDP-QUIC] also describes a different set of SDP attributes to perform a similar task.

Future versions of this document or the above documents may specify a mechanism for signalling that a given media session will be carried in the same QUIC connection as the SIP-over-QUIC session.

10.2. Carriage Of Non-RTP Media Streaming Protocols In A QUIC Transport Session

[RUSH] does not specify a means to discover the presence of a RUSH streaming session, nor a mechanism for negotiating the encoding parameters of media that is being exchanged. RUSH has two modes of operation: Normal and Multi Stream modes. Normal mode, as described in Section 4.3.1 of [RUSH], uses a single bidirectional QUIC stream to send and receive media streams. Multi Stream mode, as described in Section 4.3.2 of [RUSH], uses a bidirectional QUIC stream for each individual media frame. Bidirectional streams appear to be used in order to give error feedback, as opposed to having a separate control stream for handling errors or using the QUIC transport error mechanism. If the stream type mechanism described in Section 5.2 is expanded to cover bidirectional streams as well, then SIP-over-QUIC could be used with RUSH.

[Warp] specifies that sessions are established using HTTP/3 WebTransport ([WebTransH3]). However, to the author's best knowledge WebTransport does not yet contain any signalling or media negotiation similar to how WebRTC would use SDP offer/answer exchanges, so some form of session establishment mechanism like SIP-over-QUIC could be useful in filling this gap. Warp uses QUIC unidirectional streams for sending media. Similar to MPEG-DASH, media is sent in ISO-BMFF "segments", with each stream carrying a single segment. This can easily be accommodated by the media stream type reserved in Section 5.2.

[QuicR-Arch] describes SDP as overly complicated, and [QuicR-Proto] defines the QuicR Manifest for advertising media sessions and endpoint capabilities and, as such, SIP-over-QUIC probably isn't required. However, it is possible that trying to design this

manifest mechanism from scratch is likely to require extra time and effort to develop, while SDP is a perfectly usable solution.

11. Security Considerations

The security considerations of SIP-over-QUIC should be comparable to those of [SIP2.0] and [HTTP3].

SIP-over-QUIC relies on QUIC, which itself relies on TLS 1.3 and thus supports by default the protections against downgrade attacks described in [BCP195]. QUIC-specific issues and their mitigations are described in Section 21 of [QUIC-TRANSPORT].

Section 4.1 gives specific guidance on the conversion of transactions between SIP-over-QUIC and carriage of SIP over other transport protocols which make use of the branch parameter, in order to avoid leaking details of the underlying QUIC transport connection.

12. IANA Considerations

> **Author's Note**: This section of the document reflects future IANA registrations, and not current ones. The intention is for these registrations to occur once this Internet-Draft becomes an RFC.

This document registers a new ALPN protocol IDs (Section 12.1) and creates new registries that manage the assignment of code points in SIP-over-QUIC (Section 12.2).

12.1. Registration Of SIP Identification Strings

This document creates a new registration of SIP-over-QUIC in the "TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry established in [RFC7301].

The "sips/quic" string identifies SIP-over-QUIC:

Protocol: SIP-over-QUIC

Identification Sequence: 0x73 0x69 0x70 0x73 0x2F 0x71 0x75 0x69 0x63 ("sips/quic")

Specification: This document

This document creates a new registration of SIP/2.0 over TLS in the "TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry established in [RFC7301].

The "sips/2.0" string identifies SIP/2.0 over TLS:

Protocol:
SIP/2.0 over TLS

Identification Sequence: 0x73 0x69 0x70 0x73 0x2F 0x32 0x2E 0x30 ("sips/2.0")

Specification: [SIP2.0]

This document creates a new registration of SIP/2.0 over UDP in the "TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry established in [RFC7301].

The "sip/2.0" string identifies SIP/2.0 over UDP:

Protocol: SIP/2.0 over UDP

Identification Sequence: 0x73 0x69 0x70 0x2F 0x32 0x2E 0x30 ("sip/2.0")

Specification: [SIP2.0]

12.2. New Registries

New registries created in this document operate under the QUIC registration policy documented in Section 22.1 of [QUIC-TRANSPORT]. These registries all include the common set of fields listed in Section 22.1.1 of [QUIC-TRANSPORT]. These registries are collected under the "Session Initiation Protocol over QUIC Transport (SIP-over-QUIC)" heading.

The initial allocations in these registries are all assigned permanent status and list a change controller of the IETF and a contact of the *[TBC]* working group.

12.2.1. Frame Types

This document establishes a registry for SIP-over-QUIC frame type codes. The "SIP-over-QUIC Frame Types" registry governs a 62-bit space. This registry follows the QUIC registry policy; see Section 12.2. Permanent registrations in this registry are assigned using the Specification Required policy [RFC8126]), except for values between 0x00 and 0x3f (in hexadecimal; inclusive), which are assigned using Standards Action or IESG Approval as defined in Sections 4.9 and 4.10 of [RFC8126].

In addition to common fields as described in Section 12.2, permanent registrations in this registry **MUST** include the following fields:

Frame type: A name or label for the frame type.

Specifications of frame types **MUST** include a description of the frame layout and its semantics, including any parts of the frame that are conditionally present.

The entries in Table 2 are registered by this document.

Frame Type	Value	Specification
DATA	0x00	Section 7.2.1
HEADERS	0x01	Section 7.2.2
CANCEL	0x02	Section 7.2.3
SETTINGS	0x04	Section 7.2.4

Table 2: Initial SIP-over-QUIC Frame Types

12.2.2. Settings Parameters

This document establishes a registry for SIP-over-QUIC parameters. The "SIP-over-QUIC Parameters" registry governs a 62-bit space. This registry follows the QUIC registry policy; see Section 12.2. Permanent registrations in this registry are assigned using the Specification Required policy [RFC8126]), except for values between 0x00 and 0x3f (in hexadecimal; inclusive), which are assigned using Standards Action or IESG Approval as defined in Sections 4.9 and 4.10 of [RFC8126].

In addition to common fields as described in Section 12.2, permanent registrations in this registry **MUST** include the following fields:

Parameter Name: A symbolic name for the parameter. Specifying a parameter name is optional.

Default: The value of the parameter unless otherwise indicated. A default **SHOULD** be the most restrictive possible value.

The entries in Table 3 are registered by this document.

Parameter Name	Value	Specification	Default
SETTINGS_QPACK_MAX_TABLE_CAPACITY	0x01	Section 7.2.4.1	0
SETTINGS_MAX_FIELD_SECTION_SIZE	0x06	Section 7.2.4.1	Unlimited
SETTINGS_QPACK_BLOCKED_STREAMS	0x07	Section 7.2.4.1	0

Table 3: Initial SIP-over-QUIC Parameters

12.2.3. Error Codes

This document establishes a registry for SIP-over-QUIC error codes. The "SIP-over-QUIC Error Codes" registry governs a 62-bit space. This registry follows the QUIC registry policy; see Section 12.2.

Permanent registrations in this registry are assigned using the Specification Required policy [RFC8126]), except for values between 0x0300 and 0x033f (in hexadecimal; inclusive), which are assigned using Standards Action or IESG Approval as defined in Sections 4.9 and 4.10 of [RFC8126].

In addition to common fields as described in Section 12.2, permanent registrations in this registry **MUST** include the following fields:

Name: A name for the error code.

Description: A brief description of the error code semantics.

The entries in Table 4 are registered by this document. These error codes were selected from the range that operates on a Specification Required policy to avoid collisions with HTTP/2 and HTTP/3 error codes.

Name	Value	Description	Specification
SIP_NO_ERROR	0x0300	No error.	Section 8.1
SIP_GENERAL_PROTOCOL_ERROR	0x0301	Non-specific error code.	Section 8.1
SIP_INTERNAL_ERROR	0x0302	An internal error occurred.	Section 8.1
SIP_STREAM_CREATION_ERROR	0x0303	Peer created an unacceptable stream.	Section 8.1
SIP_CLOSED_CRITICAL_STREAM	0x0304	A required stream was closed.	Section 8.1
SIP_FRAME_ERROR	0x0305	An invalid frame was received.	Section 8.1
SIP_FRAME_UNEXPECTED	0x0306	A not permitted frame was received.	Section 8.1
SIP_CANCEL_FRAME_CLOSED	0x0307	A CANCEL frame referenced an unopened stream ID.	Section 8.1
SIP_SETTINGS_ERROR	0x0309	An error was detected in a SETTINGS frame.	Section 8.1
SIP_MISSING_SETTINGS	0x030a	No SETTINGS frame was received.	Section 8.1
SIP_REQUEST_REJECTED	0x030b		Section 8.1

Name	Value	Description	Specification
		User agent server rejected a request.	
SIP_REQUEST_CANCELLED	0x030c	The request or its response is cancelled.	Section 8.1
SIP_REQUEST_INCOMPLETE	0x030d	Stream terminated without a full request.	Section 8.1
SIP_MESSAGE_ERROR	0x030e	A SIP message was malformed.	Section 8.1
SIP_HEADER_COMPRESSION_FAILED	0x0310	Failed to interpret an encoded field section.	Section 8.1
SIP_HEADER_TOO_LARGE	0x0311	Received encoded field section is too large.	Section 8.1

Table 4: Initial SIP-over-QUIC Error Codes

12.2.4. Stream Types

This document establishes a registry for SIP-over-QUIC stream types. The "SIP-over-QUIC Stream Types" registry governs a 62-bit space. This registry follows the QUIC registry policy; see Section 12.2. Permanent registrations in this registry are assigned using the Specification Required policy [RFC8126]), except for values between 0x00 and 0x3f (in hexadecimal; inclusive), which are assigned using Standards Action or IESG Approval as defined in Sections 4.9 and 4.10 of [RFC8126].

In addition to common fields as described in Section 12.2, permanent registrations in this registry **MUST** include the following fields:

Stream Type: A name or label for the stream type.

Sender: Which endpoint on a SIP-over-QUIC connection may initiate a stream of this type. Values are "Client", "Server", or "Both".

The entries is Table 5 are registered by this document.

Stream Type	Value	Specification	Sender
Control Stream	0x00	Section 5.2.1	Both
QPACK Encoder Stream	0x02	Section 5.2	Both
QPACK Decoder Stream	0x03	Section 5.2	Both

Stream Type	Value	Specification	Sender
Reserved	0x04	Section 5.2	Both

Table 5: Initial SIP-over-QUIC Stream Types

13. References

13.1. Normative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015.
Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, March 2021.
<https://www.rfc-editor.org/info/bcp195>

[HTTP-SEMANTICS] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[HTTP3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[QPACK] Krasic, C., Bishop, M., and A. Frindell, Ed., "QPACK: Field Compression for HTTP/3", RFC 9204, DOI 10.17487/RFC9204, June 2022, <https://www.rfc-editor.org/rfc/rfc9204>.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3261]
Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/rfc/rfc3261>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI

10.17487/RFC3264, June 2002, <https://www.rfc-editor.org/rfc/rfc3264>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[SIP2.0] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/rfc/rfc3261>.

13.2. Informative References

[DNS-SVCB] Schwartz, B., Bishop, M., and E. Nygren, "Service binding and parameter specification via the DNS (DNS SVCB and HTTPS RRs)", Work in Progress, Internet-Draft, draft-ietf-dnsop-svcb-https-11, <https://datatracker.ietf.org/doc/html/draft-ietf-dnsop-svcb-https-11>.

[HTTP1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[QRT] Hurst, S., "QRT: QUIC RTP Tunnelling", Work in Progress, Internet-Draft, draft-hurst-quic-rtp-tunnelling-01, <https://datatracker.ietf.org/doc/html/draft-hurst-quic-rtp-tunnelling-01>.

[QUIC-DATAGRAMS] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", RFC 9221, DOI 10.17487/RFC9221, March 2022, <https://www.rfc-editor.org/rfc/rfc9221>.

[QuicR-Arch] Jennings, C. and S. Nandakumar, "QuicR - Media Delivery Protocol over QUIC", Work in Progress, Internet-Draft, draft-jennings-moq-quicr-arch-01, <https://datatracker.ietf.org/doc/html/draft-jennings-moq-quicr-arch-01>.

[QuicR-Proto] Jennings, C., Nandakumar, S., and C. Huitema, "QuicR - Media Delivery Protocol over QUIC", Work in Progress, Internet-Draft, draft-jennings-moq-quicr-proto-01, <https://datatracker.ietf.org/doc/html/draft-jennings-moq-quicr-proto-01>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/rfc/rfc2782>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/rfc/rfc7301>.

[RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <https://www.rfc-editor.org/rfc/rfc8499>.

[RTP-over-QUIC] Ott, J. and M. Engelbart, "RTP over QUIC", Work in Progress, Internet-Draft, draft-ietf-avtcore-rtp-over-quic-01, <https://datatracker.ietf.org/doc/html/draft-ietf-avtcore-rtp-over-quic-01>.

[RUSH] Pugin, K., Frindell, A., Cenzano, J., and J. Weissman, "RUSH - Reliable (unreliable) streaming protocol", Work in Progress, Internet-Draft, draft-kpugin-rush-01, <https://datatracker.ietf.org/doc/html/draft-kpugin-rush-01>.

[SDP-QUIC] Dawkins, S., "SDP Offer/Answer for RTP using QUIC as Transport", Work in Progress, Internet-Draft, draft-dawkins-avtcore-sdp-rtp-quic-00, <https://datatracker.ietf.org/doc/html/draft-dawkins-avtcore-sdp-rtp-quic-00>.

[SNI] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/rfc/rfc6066>.

[Warp] Curley, L., "Warp - Segmented Live Media Transport", Work in Progress, Internet-Draft, draft-lcurley-warp-02, <https://datatracker.ietf.org/doc/html/draft-lcurley-warp-02>.

[WebTransH3] Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-03, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-03>.

Appendix A. Acknowledgments

The author would like to acknowledge Richard Bradbury as the inspiration for the idea behind this document, and to Piers O'Hanlon for his review comments.

Appendix B. QPACK Static Table

Author's Note: This is only a preliminary table. The original HPACK static table was created after analysing the frequency of common HTTP header fields and their values. QPACK repeated that effort at a later date, which resulted in a different static table. The author welcomes any data that would permit a similar level of analysis for the frequency of common SIP header fields and their values.

Index	Name	Value
0	:request-uri	
1	from	
2	to	
3	call-id	
4	via	
5	:method	REGISTER
6	:method	INVITE
7	:method	ACK
8	:method	BYE
9	:method	CANCEL
10	:method	UPDATE
11	:method	REFER
12	:method	OPTIONS
13	:method	MESSAGE

Index	Name	Value
14	:status	100
15	:status	180
16	:status	200
17	:status	301
18	:status	302
19	:status	400
20	:status	401
21	:status	404
22	:status	407
23	:status	408
24	contact	
25	content-type	application/sdp
26	content-type	text/html
27	content-disposition	session
28	content-disposition	render
29	content-length	
30	accept	application/sdp
31	accept-encoding	gzip
32	accept-language	
33	alert-info	
34	allow	REGISTER
35	allow	INVITE
36	allow	ACK
37	allow	BYE
38	allow	CANCEL
39	allow	UPDATE
40	allow	REFER
41	allow	OPTIONS
42	allow	MESSAGE
43	authentication-info	
44	authorization	
45	call-info	
46	content-encoding	
47	content-language	
48	date	
49	error-info	
50	expires	
51	in-reply-to	
52	max-forwards	
53	min-expires	
54	mime-version	
55	organization	
56	priority	Non-urgent
57	priority	Normal
58	priority	Urgent

Index	Name	Value
59	priority	Emergency
60	proxy-authenticate	
61	proxy-authorization	
62	proxy-require	
63	record-route	
64	reply-to	
65	require	
66	retry-after	
67	route	
68	server	
69	subject	
70	supported	
71	timestamp	
72	unsupported	
73	user-agent	
74	warning	300
75	warning	301
76	warning	302
77	warning	303
78	warning	304
79	warning	305
80	warning	306
81	warning	307
82	warning	330
83	warning	331
84	warning	370
85	warning	399
86	www-authenticate	

Table 6: Static Table

Index

Author's Address

Sam Hurst
BBC Research & Development

Email: sam.hurst@bbc.co.uk