Multiplexed Application Substrate over QUIC Encryption
Internet-Draft
Intended status: Standards Track
Expires: 31 August 2026

Y. Rosomakho
Zscaler
T. Pauly
Apple

27 February 2026

Extensions to Compress and Derive Fields in HTTP Datagrams
draft-rosomakho-masque-connect-ip-optimizations-01

Abstract

This document defines extensions for HTTP Datagram-based protocols that improve transmission efficiency by introducing templates for compressing or deriving datagram fields.

These templates allow endpoints to define parts of datagrams that are static and can be removed, and other parts that can be derived (such as packet lengths and checksum values).

Additionally, this document defines a checksum offload procedure enabling receivers to complete Internet checksums using sender-provided partial values.

These optimisations reduce per-packet overhead, processing cost, and increase the effective maximum transmission unit (MTU) when datagrams are encapsulated in QUIC DATAGRAM frames.

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://yaroslavros.github.io/connect-ip-optimizations/draft-rosomakho-masque-connect-ip-optimizations.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rosomakho-masque-connect-ip-optimizations/.

Discussion of this document takes place on the Multiplexed Application Substrate over QUIC Encryption Working Group mailing list (mailto:masque@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/masque/. Subscribe at https://www.ietf.org/mailman/listinfo/masque/.

Source for this draft and an issue tracker can be found at https://github.com/yaroslavros/connect-ip-optimizations.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The CONNECT-IP method [CONNECT-IP] allows an HTTP client to establish an IP tunnel through an HTTP proxy and exchange IP packets using either HTTP/3 Datagrams (Section 2.1 of [HTTP-DATAGRAMS]) or DATAGRAM capsules (Section 3.5 of [HTTP-DATAGRAMS]). Similarly, CONNECT-ETHERNET [CONNECT-ETHERNET] allows sending Ethernet frames over HTTP Datagrams. These protocols send complete packets or frames by default, including all transport and network headers. This is a

simple approach, but incurs per-packet overhead due to the repeated transmission of largely invariant header fields.

Other HTTP Datagram-based protocols share similar properties: datagrams often contain structured packets where many header fields remain constant across a flow while only a subset of bytes change between packets. Transmitting complete packets therefore wastes bandwidth and processing resources.

This document introduces a set of optional extensions that define Processing Contexts for HTTP Datagram payloads. A Processing Context describes transformations applied to a received datagram payload prior to delivery to the target protocol and may reference a parent context, forming a processing chain.

Reusable templates allow endpoints to associate a Context Identifier with a reusable packet layout consisting of static and variable byte regions. Once a template has been installed using reliable Capsules, datagrams referencing the same Context Identifier carry only the variable portions of the packet. This reduces the size of transmitted datagrams and processing overhead, while remaining compatibile with intermediaries that are unaware of these optimisations.

Derived field processing allows the receiver to reconstruct certain header fields (for example packet length fields and complete checksums) based on the size and structure of the reconstructed packet. This eliminates the need for the sender to transmit such fields for every packet.

In addition, this document defines a checksum offload procedure enabling endpoints to cooperatively compute Internet checksums, where the sender provides a partial checksum and the receiver completes the computation after reconstruction. This mirrors hardware checksum-offload behavior used on network interfaces and tunnel devices, reducing per-packet CPU cost for encapsulating or decapsulating CONNECT-IP and CONNECT-ETHERNET traffic.

When HTTP Datagrams are encapsulated in QUIC DATAGRAM frames, these optimisations also increase the effective maximum transmission unit (MTU) by reducing the number of bytes carried inside each QUIC packet.

All extensions are negotiated during the HTTP request/response handshake and signalled using Capsules on the reliable control stream. Endpoints can always fall back to transmitting complete datagrams using Context Identifier 0, which represents unoptimised datagrams containing the full payload as defined by the underlying HTTP Datagram protocol.

2. Conventions and Definitions

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.

The following terms are used in this document:

Context Identifier (Context ID): A numeric identifier associated with a Processing Context. Context ID encoding and allocation follow the rules defined in Section 4 of [CONNECT-UDP]. Context ID 0 indicates that the datagram payload is delivered without additional processing as defined by the underlying HTTP Datagram protocol.

Processing Context: A set of rules describing how an HTTP Datagram payload is transformed before delivery to the target protocol. A Processing Context may reference a parent context, forming a processing chain. Processing Contexts are immutable once created.

Template: A reusable packet layout consisting of a sequence of static and variable segments. Static segments contain bytes removed from optimized datagrams, while variable segments correspond to bytes still carried in the datagram payload.

Derived Field: A header field whose value is generated by the receiver during reconstruction and written into the reconstructed packet rather than being transmitted in the datagram payload. Derived fields include length fields and complete checksums.

Checksum Offload: A capability allowing the receiver to complete the Internet checksum according to [INCREMENTAL-CHECKSUM] using a sender-provided partial checksum after reconstruction of the packet.

Capsule: A reliable control-stream message, as defined in Section 3 of [HTTP-DATAGRAMS], used in this specification to signal creation, acknowledgement, or deletion of Processing Contexts.

3. Negotiation of Capabilities

Endpoints negotiate support for HTTP Datagram processing contexts during the HTTP request/response handshake by using the http-datagram-contexts HTTP header field, whose value is a Structured Field Dictionary as defined in Section 3.2 of [STRUCTURED-HTTP].

3.1. Header Definition

```
http-datagram-contexts = sf-dictionary
```

Figure 1: http-datagram-contexts header field

This document defines the following optional dictionary keys:

max-templates (Integer): Maximum number of concurrently active template contexts the sender is willing to maintain for templates created by the peer. Absence of this key or value of 0 indicates that the sender does not support reusable templates.

max-templates-segments (Integer): Maximum number of static segments accepted within a single template. Absence of this key or value of 0 indicates that the sender does not impose a limit on number of static segments in a single template.

derived (Inner List): A list of supported Derived Field Types as defined in Section 8.3.

checksum (Boolean): Indicates support for the checksum offload procedure defined in this document. A value of ?1 means the endpoint is willing to complete checksums using sender-provided partial values. If omitted or set to ?0, checksum offload is not supported.

mtu (Integer): Upper limit on maximum reconstructed packet size the receiver is willing to accept.

Endpoints MUST ignore unknown dictionary members. The absence of a member implies that the corresponding capability is not supported for contexts created by the peer.

3.2. Negotiation Behavior

Capabilities are directional. Each endpoint advertises the processing contexts it is willing to receive and maintain for datagrams sent by the peer. An endpoint MAY create a context only if the peer advertised support for the corresponding capability.

3.2.1. Templates

If the peer advertises the max-templates value greater than 0, the endpoint MAY create template contexts up to that limit using capsules defined in Section 4.

An endpoint MUST NOT create templates exceeding the peer's advertised max-template-segments limit when that parameter is present.

If the peer advertises an mtu limit, the sender MUST NOT transmit a datagram that would reconstruct into a packet larger than the advertised limit after all processing contexts have been applied.

3.2.2. Derived Fields

An endpoint MAY create a derived context only if every operation in the capsule appears in the peer's derived list.

3.2.3. Checksum Offload

An endpoint MAY create a checksum offload context only if the peer advertised checksum=?1.

3.3. Example

HTTP/3 sample request (client to proxy):

```
:method = CONNECT
:protocol = connect-ip
:scheme = https
:path = /.well-known/masque/ip/*/*/
:authority = proxy.example.net
capsule-protocol = ?1
http-datagram-contexts = max-templates=20000, max-templates-segments=32, derived=(0 2 4),
 checksum=?1, mtu=1500
```

Figure 2: CONNECT-IP with http-datagram-contexts request example

HTTP/3 sample response (proxy to client):

```
:status = 200
capsule-protocol = ?1
http-datagram-contexts = max-templates=65535, derived=(0 1), checksum=?0, mtu=1500
```

Figure 3: CONNECT-IP with http-datagram-contexts response example

In this example, both peers support reusable templates. The proxy supports a subset of derived fields (ipv4-total-length, ipv4-udp-length and ipv4-header-checksum) and the checksum offload. The

client supports a different subset of derived fields (ipv4-total-length and ipv6-payload-length) without the checksum offload. Both endpoints indicate that the maximum packet size after reconstruction must not exceed 1500 bytes.

4. Processing Context Capsules

This specification defines multiple capsule types to construct, acknowledge, and delete processing contexts.

4.1. Processing Context Overview

4.1.1. Processing Context Construction

Processing contexts are created using capsules that define a new unique non-zero Context ID encoded as a variable-length integer. A Context ID MUST NOT be reused. As specified in Section 4 of [CONNECT-UDP], even-numbered Context IDs are allocated by the client and odd-numbered by the proxy.

Each processing context MAY reference an already-defined parent context using Next Context ID encoded as a variable-length integer. A context MUST reference only a Context ID previously defined by the peer. Forward references are not permitted. Processing context without a parent is identified by Next Context ID set to 0. A processing chain MUST NOT contain more than one context of the same type. A receiver that detects such a condition MUST treat the context as malformed and follow the error-handling procedure defined in Section 3.3 of [HTTP-DATAGRAMS].

A receiver of an *_ASSIGN capsule with an invalid Context ID or unknown Next Context ID MUST treat it as malformed and follow the error-handling procedure defined in Section 3.3 of [HTTP-DATAGRAMS].

4.1.2. Processing Context Acknowledgement

For each *_ASSIGN capsule received, the receiver MUST transmit the corresponding *_ACK capsule after successfully installing the context.

Endpoints MAY transmit datagrams referencing contexts prior to receiving the *_ACK. A receiver MAY buffer datagrams referencing unknown Context IDs but MUST bound buffering by time and memory.

A receiver of an *_ACK capsule with an unknown Context ID or any data after Context ID MUST treat it as malformed and follow the error-handling procedure defined in Section 3.3 of [HTTP-DATAGRAMS].

4.1.3. Processing Context Closure

Processing Contexts are retired by sending corresponding *_CLOSE capsule. Closing a context implicitly closes all contexts that reference it directly or transitively.

A receiver of a *_CLOSE capsule SHOULD retain the closed context and its descendants for a short period to allow in-flight datagrams to arrive, but MUST bound the retention time and memory usage.

*_CLOSE capsules with unknown Context ID or any data after Context ID MUST be treated as malformed. Receiver of such capsules MUST follow the error-handling procedure defined in Section 3.3 of [HTTP-DATAGRAMS].

4.2. Template Capsules

4.2.1. TEMPLATE_ASSIGN Capsule

```
TEMPLATE_ASSIGN Capsule {
  Type (i) = 0x3ee3143f,
  Length (i),
  Context ID (i),
  Next Context ID (i),
  Static Segment (..) ...
}
```

Figure 4: TEMPLATE_ASSIGN Capsule Format

The TEMPLATE_ASSIGN capsule contains a sequence of one or more Static Segments.

```
Static Segment {
  Segment Offset (i),
  Segment Length (i),
  Segment Payload (..),
}
```

Figure 5: Static Segment Format

Each Static Segment contains following fields:

Segment Offset: Byte offset from the start of the reconstructed packet, encoded as a variable-length integer

Segment Length: Length of the Segement Payload field, encoded as a variable-length integer

Segment Payload: Static bytes to insert at the Segment Offset

4.2.1.1. Parsing and validation

The receiver parses a TEMPLATE_ASSIGN capsule by reading, in order: the Context ID, the Next Context ID, and one or more static segments whose encodings consume exactly the remaining length of the capsule. Context ID and Next Context ID processing is described in Section 4.1.1.

Each Static Segment consists of a Segment Offset, a Segment Length, and exactly Segment Length octets of Segment Payload. Static segments MUST appear in strictly increasing Segment Offset order and MUST NOT overlap. There MUST be at least 1 byte between consecutive segments.

A receiver that advertised a max-templates-segments limit MUST ensure that the template does not contain more static segments. A receiver that advertised a mtu limit in http-datagram-contexts MUST ensure that the sum of Segment Offset and Segment Length of the final segment does not exceed the MTU limit. Final reconstructed packet size validation is performed during packet reconstruction (Section 5.2). The capsule MUST end immediately after the last static segment.

If any of the capsule fields are malformed upon reception, the receiver of the capsule MUST follow the error-handling procedure defined in Section 3.3 of [HTTP-DATAGRAMS].

A receiver that has already accepted the maximum number of templates it advertised via the max-templates member in http-datagram-contexts MUST treat any additional TEMPLATE_ASSIGN capsule an error and MUST follow the same error-handling procedure.

Per-packet validation uses the reconstruction procedure described in Section 5.2.

4.2.2. TEMPLATE_ACK Capsule

```
TEMPLATE_ACK Capsule {
  Type (i) = 0x3ee31440,
  Length (i),
  Context ID (i),
}
```

Figure 6: TEMPLATE_ACK Capsule Format

Processing of the TEMPLATE_ACK capsule is described in Section 4.1.2

4.2.3. TEMPLATE_CLOSE Capsule

```
TEMPLATE_CLOSE Capsule {
  Type (i) = 0x3ee31441,
  Length (i),
  Context ID (i),
}
```

Figure 7: TEMPLATE_CLOSE Capsule Format

Processing of the TEMPLATE_CLOSE capsule is described in Section 4.1.3

4.3. Derived Field Capsules

4.3.1. DERIVED_ASSIGN Capsule

```
DERIVED_ASSIGN Capsule {
  Type (i) = 0x3ee31442,
  Length (i),
  Context ID (i),
  Next Context ID (i),
  Derived Field Type (i) ...
}
```

Figure 8: DERIVED_ASSIGN Capsule Format

The DERIVED_ASSIGN capsule defines a processing context that generates and inserts one or more derived fields into the reconstructed packet. The sender does not transmit these fields in the datagram payload.

4.3.1.1. Parsing and validation

The receiver parses a DERIVED_ASSIGN capsule by reading, in order: the Context ID, the Next Context ID, and one or more Derived Field Type values encoded as variable-length integers. Context ID and Next Context ID processing is described in Section 4.1.1.

If a Derived Field Type is not present in the receiver’s advertised derived capability list in http-datagram-contexts or if any Derived Field Type appears more than once in the capsule, the receiver MUST treat the capsule as malformed and follow the error-handling procedure defined in Section 3.3 of [HTTP-DATAGRAMS].

Per-packet validation uses the reconstruction procedure described in Section 5.2.

4.3.2. DERIVED_ACK Capsule

```
DERIVED_ACK Capsule {
  Type (i) = 0x3ee31443,
  Length (i),
  Context ID (i),
}
```

Figure 9: DERIVED_ACK Capsule Format

Processing of the DERIVED_ACK capsule is described in Section 4.1.2

4.3.3. DERIVED_CLOSE Capsule

```
DERIVED_CLOSE Capsule {
  Type (i) = 0x3ee31444,
  Length (i),
  Context ID (i),
}
```

Figure 10: DERIVED_CLOSE Capsule Format

Processing of the DERIVED_CLOSE capsule is described in Section 4.1.3

4.4. Checksum Offload Capsules

4.4.1. CHECKSUM_ASSIGN Capsule

```
CHECKSUM_ASSIGN Capsule {
  Type (i) = 0x3ee31445,
  Length (i),
  Context ID (i),
  Next Context ID (i),
  Checksum Field Offset (i),
  Checksum Start Offset (i),
}
```

Figure 11: CHECKSUM_ASSIGN Capsule Format

The CHECKSUM_ASSIGN capsule defines a processing context that completes an Internet checksum for the reconstructed packet using a sender-provided partial checksum.

In addition to Context ID and Next Context ID CHECKSUM_ASSIGN capsule contains following fields encoded as variable-length integers:

Checksum Field Offset: Byte offset of the 16-bit Internet checksum field within the reconstructed packet

Checksum Start Offset: Byte offset where checksum coverage begins. Coverage runs from this offset to the end of the reconstructed packet.

4.4.1.1. Parsing and validation

The receiver parses a CHECKSUM_ASSIGN capsule by reading, in order: the Context ID, the Next Context ID, Checksum Field Offset and Checksum Start Offset. Context ID and Next Context ID processing is described in Section 4.1.1.

If the peer did not advertise checksum=?1 in http-datagram-contexts, the receiver MUST treat the capsule as malformed and follow the error-handling procedure defined in Section 3.3 of [HTTP-DATAGRAMS].

If Checksum Start Offset is 0, the receiver MUST treat the capsule as malformed and follow the same error-handling procedure.

Per-packet validation uses the reconstruction procedure described in Section 5.2.

4.4.2. CHECKSUM_ACK Capsule

```
CHECKSUM_ACK Capsule {
  Type (i) = 0x3ee31446,
  Length (i),
  Context ID (i),
}
```

Figure 12: CHECKSUM_ACK Capsule Format

Processing of the CHECKSUM_ACK capsule is described in Section 4.1.2

4.4.3. CHECKSUM_CLOSE Capsule

```
CHECKSUM_CLOSE Capsule {
  Type (i) = 0x3ee31447,
  Length (i),
  Context ID (i),
}
```

Figure 13: CHECKSUM_CLOSE Capsule Format

Processing of the CHECKSUM_CLOSE capsule is described in Section 4.1.3

5. Processing Context Operation

This section defines how endpoints construct and consume HTTP Datagram payloads using Processing Contexts.

A datagram carries a Context Identifier that selects the initial Processing Context. A context MAY reference a parent context using Next Context ID. The complete behavior is defined by recursively following parent contexts until reaching Context ID 0.

Context ID 0 indicates that no processing is applied and the payload is delivered unchanged to the underlying HTTP Datagram protocol.

5.1. Sender behavior

When sending a datagram using a Processing Context, the sender constructs the payload so that the receiver can reconstruct the final packet after applying the processing chain.

The sender MUST use a Context ID only after the corresponding *_ASSIGN capsule has been transmitted.

5.1.1. Template Contexts

If the selected context chain contains a Template context, the sender constructs the datagram payload as the concatenation of all variable byte regions not covered by static segments.

Variable regions are emitted in strictly increasing offset order starting at offset 0.

If the context chain contains no Template context, the payload MUST be the complete packet.

5.1.2. Derived Field Contexts

Derived fields are not transmitted by the sender. When a derived context is in use, the sender MUST remove the octets corresponding to derived fields from the datagram payload. These octets are supplied by the receiver during reconstruction.

The sender MUST construct the payload as if the derived field octets were not part of the variable regions. That is, the payload MUST contain only the remaining variable octets in strictly increasing offset order.

5.1.3. Checksum Offload Contexts

If the context chain contains a checksum offload context, the sender MUST place a precomputed partial Internet checksum value into the checksum field at Checksum Field Offset in the reconstructed packet image prior to transmission. This value is combined with the receiver computation as described in Section 5.2.

5.1.4. Context Selection

If a packet does not match any available context, the sender MUST use Context ID 0 and transmit the complete packet.

5.2. Receiver behavior

Upon receiving an HTTP Datagram with a non-zero Context ID, the receiver retrieves the referenced Processing Context and recursively resolves its parent contexts until Context ID 0 is reached.

If any referenced context is unknown, the receiver MAY buffer the datagram as described in Section 4.1.2 or drop it.

If multiple processing contexts are present in a chain, the receiver MUST apply them in the following order:

1. Template reconstruction (if present)

2. Derived field processing (if present)

3. Checksum offload processing (if present)

5.2.1. Template Reconstruction

If a Template context is present, the receiver reconstructs the packet as follows:

1. Allocate a buffer large enough to contain the reconstructed packet.

2. Insert static segment bytes at their specified offsets.

3. Fill all remaining gaps using bytes from the datagram payload in strictly increasing offset order.

If payload bytes are exhausted before all gaps have been filled the datagram MUST be dropped.

Packets larger than the advertised mtu in http-datagram-contexts MUST be dropped.

5.2.2. Derived Field Processing

For each derived field present in the context chain, the receiver computes the field value and inserts it into the reconstructed packet at the location defined by the derived field type.

Derived fields are inserted into the packet image and therefore increase the reconstructed packet size. The receiver MUST compute derived field values based on the final reconstructed packet size and structure.

Initial field definitions are specified in Section 8.3.

If the required header cannot be located, the packet MUST be dropped.

5.2.3. Checksum Offload Processing

If a checksum offload context is present, the receiver completes the Internet checksum after all derived fields have been inserted.

The receiver completes the checksum as follows:

1. Treat the checksum field as zero.

2. Compute the one’s-complement sum from Checksum Start Offset to L.

3. Add (fold) the 16-bit value currently present at the checksum field.

4. Write the final one’s-complement result to the checksum field.

If any offset exceeds the reconstructed packet length, the packet MUST be dropped.

6. Examples

This section illustrates how contexts are created and how senders form compact payloads. All offsets and lengths are in bits in the packet diagrams and field tables. All offsets and lengths are in bytes in segment tables and sample capsules.

6.1. CONNECT-IP: TCP over IPv6 with template, derived fields and checksum offload

Original sample [TCP] over [IPv6] packet layout is illustrated below. In addition to basic IPv6 and TCP headers it contains Timestamp option as defined in Section 3 of [TCP-PERF].

This packet is to be transmitted from the client to the proxy over CONNECT-IP.

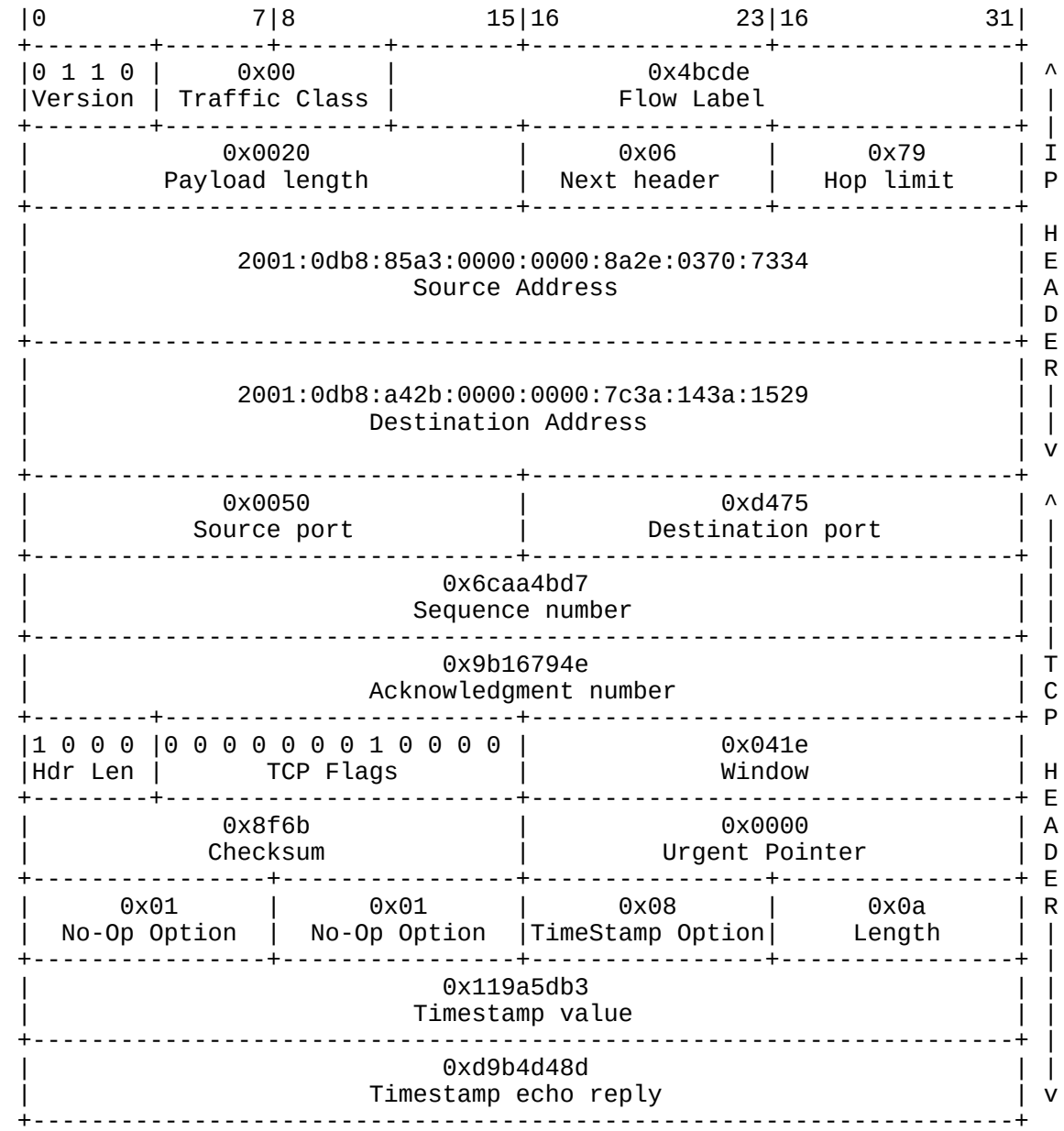

Figure 14: Example TCP over IPv6 packet before optimization

This example assumes that the peer supports templates with at least two segments per template, IPv6 payload length derived field and checksum offloading. These capabilities were communicated using the following http-datagram-contexts HTTP field in proxy response confirming CONNECT-IP extended CONNECT.

```
http-datagram-contexts = max-templates=1, max-templates-segments=2, derived=(1), checksum
=?1, mtu=1500
```

Figure 15: http-datagram-contexts response example

Since the proxy does not support TCP checksum derivation, but it supports checksum offloading, the client calculates checksum of IPv6 pseudo-header and places it in the TCP checksum field. Context for the offloaded checksum is defined using the CHECKSUM_ASSIGN capsule:

```
CHECKSUM_ASSIGN Capsule {
  Type (i) = 0x3ee31445,
  Length (i) = 4,
  Context ID (i) = 2,
  Next Context ID (i) = 0,
  Checksum Field Offset (i) = 56,
  Checksum Start Offset (i) = 40,
}
```

Figure 16: CHECKSUM_ASSIGN Capsule for example IPv6/TCP packet

Payload length field in IPv6 header can derived by the peer, so it is removed before calculating static segments. Resulting context for the derived field is defined using DERIVED_ASSIGN capsule:

```
DERIVED_ASSIGN Capsule {
  Type (i) = 0x3ee31442,
  Length (i) = 3,
  Context ID (i) = 4,
  Next Context ID (i) = 2,
  Derived Field Type (i) = 1
}
```

Figure 17: DERIVED_ASSIGN Capsule for example IPv6/TCP packet

The table below illustrates fields present in IPv6 and TCP headers after derived field was removed, their offsets in bits from the beginning of the packet and whether they are likely to be static for most packets of a given traffic flow

Offset	Field name	Length	Value	Static
0	Version	4	0110b	Yes
4	Traffic Class	8	0x00	Yes
12	Flow label	20	0x4bcde	Yes
32	Next header	8	0x06	Yes
40	Hop limit	8	0x79	Yes
48	Source address	128	2001:0db8:85a3::8a2e:0370:7334	Yes
176	Destination address	128	2001:0db8:a42b::7c3a:143a:1529	Yes
304	Source port	16	0x0050	Yes
320	Destination port	16	0xd475	Yes
336	Sequence number	32	0x6caa4bd7	No
368	Acknowledgement number	32	0x9b16794e	No
400	TCP header length	4	1000b	Yes
404	TCP Flags	12	000000010000b	No
416	Window	16	0x041e	No
432	Checksum	16	0x8f6b	No
448	Urgent pointer	16	0x0000	Yes
464	No-Op option	8	0x01	Yes
472	No-Op option	8	0x01	Yes
480	Timestamp option	8	0x08	Yes
488	Timestamp option length	8	0x0a	Yes

496	Timestamp value	32	0x119a5db3	No
528	Timestamp echo reply	32	0xd9b4d48d	No

Table 1: IPv6 and TCP header fields in example packet

Static segments model the invariant parts except for the isolated 4-bit TCP header length.

Resulting static segments:

Segment Offset	Segment Length	Segment Contents	Segment Payload
0	42	Version, Traffic Class, Flow Label, Next header, Hop limit, Source address, Destination address, Source port, Destination port	0x6004bcde067920010db885a3...
56	6	Urgent pointer, 2 No-Op TCP options, Timestamp option code and length	0x00000101080a

Table 2: Static segments for example IPv6/TCP packet

Resulting TEMPLATE_ASSIGN capsule with client-allocated even context id is illustrated below:

```
TEMPLATE_ASSIGN Capsule {
  Type (i) = 0x3ee3143f,
  Length (i) = 54,
  Context ID (i) = 6,
  Next Context ID (i) = 4,
  Static Segment {
    Segment Offset (i) = 0,
    Segment Length (i) = 42,
    Segment Payload = 0x6004bcde067920010db885a3000000008a2e0370733420010db8a42b000000007
c3a143a15290050d475,
  },
  Static Segment {
    Segment Offset (i) = 56,
    Segment Length (i) = 6,
    Segment Payload = 0x00000101080a,
  }
}
```

Figure 18: TEMPLATE_ASSIGN Capsule for example IPv6/TCP packet

The resulting processing context chain reduces per-packet overhead by removing 50 bytes of repeated header material, increasing the effective MTU when datagrams are encapsulated in QUIC DATAGRAM frames.

The sender concatenates all variable regions in increasing offset order. Packets that do not match this template (for example packets with IPv6 extension headers or without TCP options) are sent using Context ID 0 or associated with a new context.

Upon receiving the datagram with Context ID 6, proxy re-assembles the datagram by concatenating static and variable segments according to the offsets, re-calculates Payload Length and inserts it into IPv6 header and completes the TCP checksum using the sender-provided pseudo-header partial checksum.

6.2. CONNECT-ETHERNET: UDP over IPv4 with template and derived fields

Original sample [UDP] over [IPv4] Ethernet frame layout is illustrated below.

This frame is to be transmitted from the proxy to the client over CONNECT-ETHERNET.

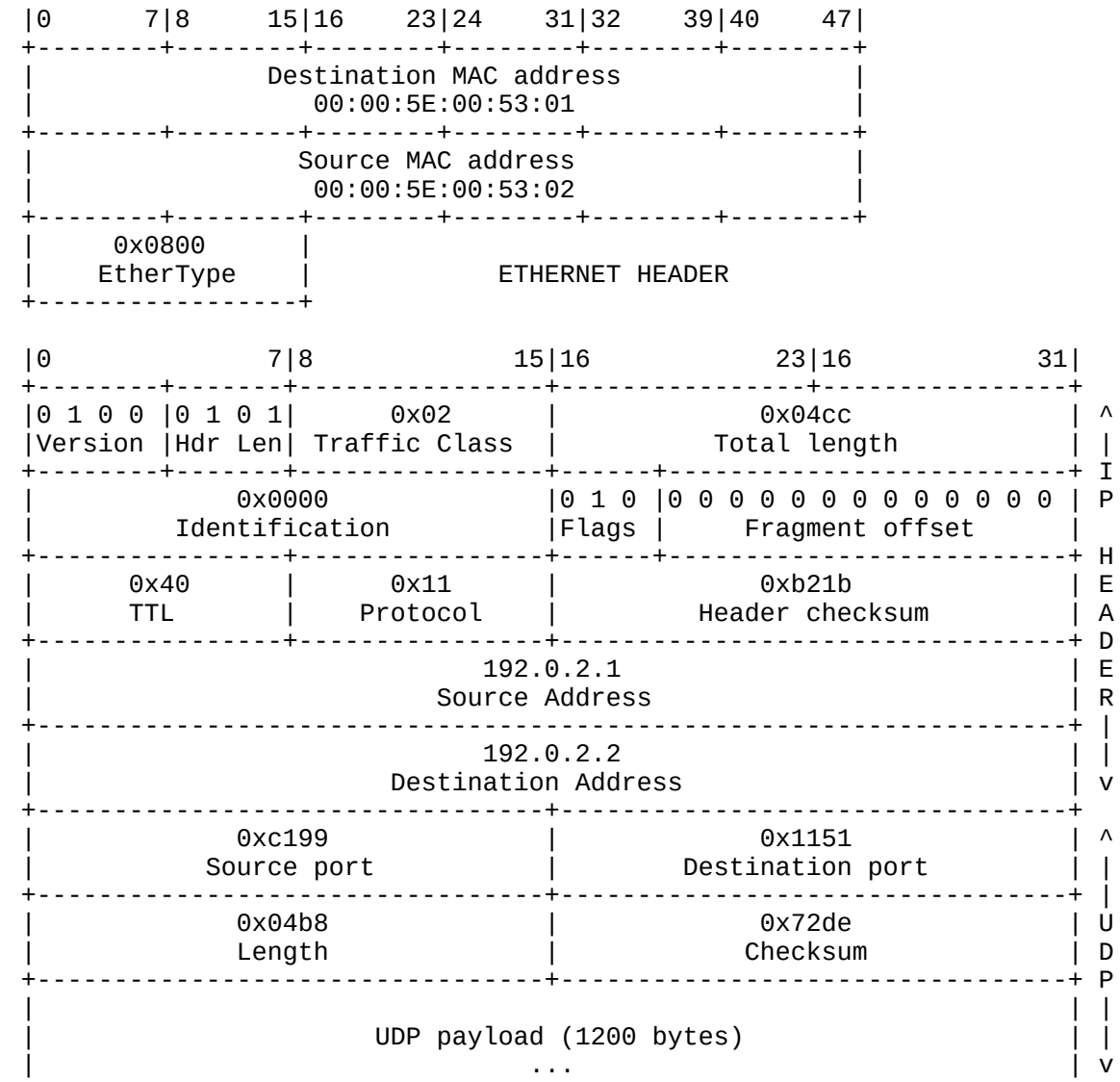

Figure 19: Example UDP over IPv4 Ethernet frame before optimization

This example assumes that the peer supports templates, IPv4 total length, IPv4 header checksum, UDP length in IPv4 packet and UDP checksum in IPv4 packet derived field. These capabilities were communicated using the following http-datagram-contexts HTTP field in client requesting CONNECT-ETHERNET extended CONNECT.

```
http-datagram-contexts = max-templates=1, max-templates-segments=1, derived=(0 2 4 7), mt
u=1500
```

Figure 20: http-datagram-contexts request example

Total length and header checksum in IPv4 header as well as length and checksum in UDP header can be derived by the peer, so these fields are removed before calculating static segments. Resulting context for the derived field is defined using DERIVED_ASSIGN capsule:

```
DERIVED_ASSIGN Capsule {
  Type (i) = 0x3ee31442,
  Length (i) = 6,
  Context ID (i) = 1,
  Next Context ID (i) = 0,
  Derived Field Type (i) = 0
  Derived Field Type (i) = 2
  Derived Field Type (i) = 4
  Derived Field Type (i) = 7
}
```

Figure 21: DERIVED_ASSIGN Capsule for example IPv4/UDP ethernet frame

Table below illustrates fields present in Ethernet, IPv4 and UDP headers after derived fields were removed, their offsets in bits from the beginning of the frame and if they are likely to be static for most packets of a given traffic flow

Offset	Field name	Length	Value	Static
0	Destination MAC address	48	00:00:5E:00:53:01	Yes
48	Source MAC address	48	00:00:5E:00:53:02	Yes
96	EtherType	16	0x0800	Yes
112	Version	4	0100b	Yes
116	Header length	4	0101b	Yes
120	Traffic Class	8	0x02	Yes
128	Identification	16	0x0000	Yes
144	Flags	3	010b	Yes
147	Fragment offset	13	0000000000000b	Yes
160	TTL	8	0x40	Yes
168	Protocol	8	0x11	Yes
176	Source address	32	192.0.2.1	Yes
208	Destination address	32	192.0.2.2	Yes
240	Source port	16	0xc199	Yes
256	Destination port	16	0x1151	Yes
272	UDP payload	9600	...	No

Table 3: Ethernet, IPv4 and UDP header fields in example frame

A single static segment model can be used for the initial part of the HTTP datagram after derived fields were removed:

Segment Offset	Segment Length	Segment Contents	Segment Payload
0	34	Source MAC address, Destination MAC address, EtherType, Version, Header length and Traffic Class, Identification, Flags, Fragment offset, TTL, Protocol, Source address, Destination address, Source port and Destination port	0x00005E00530100005E0053020800450200004000401 1c0000201c0000202c1991151

Table 4: Static segment for example Ethernet/IPv4/UDP frame

Resulting TEMPLATE_ASSIGN capsule with proxy-allocated odd Context ID is illustrated below:

```
TEMPLATE_ASSIGN Capsule {
  Type (i) = 0x3ee3143f,
  Length (i) = 38,
  Context ID (i) = 3,
  Next Context ID (i) = 1,
  Static Segment {
    Segment Offset (i) = 0,
    Segment Length (i) = 34,
    Segment Payload = 0x00005E00530100005E0053020800450200004000401 1c0000201c0000202c1991
151,
```

```
  }
}
```

Figure 22: TEMPLATE_ASSIGN Capsule for example IPv4/UDP ethernet frame

The resulting processing context chain reduces per-frame overhead by removing 34 bytes of repeated header material, increasing the effective MTU when datagrams are encapsulated in QUIC DATAGRAM frames.

The sender concatenates all variable regions in increasing offset order.

Upon receiving the datagram with Context ID 3, client re-assembles the datagram by appending variable segments to the static, re-calculates derived fields and inserts them at appropriate locations in the datagram.

7. Security Considerations

This specification changes how HTTP Datagrams are reconstructed but does not weaken transport-layer integrity or confidentiality protections provided by the underlying HTTP mapping. All Capsules travel on the reliable control stream and inherit those protections.

7.1. Resource Exhaustion

Processing contexts introduce receiver state and reconstruction work. An attacker could attempt to exhaust memory or CPU by creating excessive numbers of templates and static segments, purposely sending datagrams referencing not-yet-installed contexts and causing excessive buffering of unknown Context IDs.

Implementations MUST enforce limits on number of active templates and static segments and restrict memory used for buffering datagrams with unknown contexts.

7.2. Amplification

Derived fields and template reconstruction increase the size of the reconstructed packet relative to the received datagram payload. An attacker could exploit this to amplify processing cost and perform a denial-of-service attack.

Endpoints MUST ensure that reconstructed packet size does not exceed the negotiated MTU and SHOULD apply rate limiting when expansion ratios are abnormally high.

8. IANA Considerations

8.1. HTTP Capsule Types Registration

This specification registers the following values in the "HTTP Capsule Types" registry:

Value	Capsule Type
0x3ee3143f	TEMPLATE_ASSIGN
0x3ee31440	TEMPLATE_ACK
0x3ee31441	TEMPLATE_CLOSE
0x3ee31442	DERIVED_ASSIGN
0x3ee31443	DERIVED_ACK
0x3ee31444	DERIVED_CLOSE
0x3ee31445	CHECKSUM_ASSIGN
0x3ee31446	CHECKSUM_ACK
0x3ee31447	CHECKSUM_CLOSE

Table 5

All of these new entries use the following values for these fields:

Status: provisional (permanent if this document is approved)

Reference: This document

Change Controller: IETF

Contact: MASQUE Working Group masque@ietf.org

Notes: None

8.2. HTTP Field Name Registration

This specification registers the following value in the "HTTP Field Name" registry:

* Field Name: http-datagram-contexts
* Status: provisional (permanent if approved)
* Structured Type: Dictionary
* Reference: This document

* Comments: None

8.3. HTTP Datagram Derived Field Types Registry

IANA is requested to create a new registry titled "HTTP Datagram Derived Field Types". The registration policy is expert review as specified in Section 4.5 of [IANA-POLICY]. This new registry governs the Derived Field types that appear in DERIVED_ASSIGN capsule and derived list of http-datagram-contexts dictionary.

This new registry contains five columns:

Type: A positive integer identifying the field type

Name: A short name of the field

Description: A description of the field

Protocols: A list of HTTP Upgrade Tokens that the derived field type can apply

Reference: An optional reference defining the use of the entry.

The registry’s initial entries are as follows:

Type	Name	Description	Protocols	Reference
0	ipv4-total-length	IPv4 Total Length field derived from reconstructed packet size	connect-ip, connect-ethernet	This document
1	ipv6-payload-length	IPv6 Payload Length field derived from reconstructed packet size	connect-ip, connect-ethernet	This document
2	ipv4-udp-length	UDP Length derived from UDP header to end of IPv4 packet	connect-ip, connect-ethernet	This document
3	ipv6-udp-length	UDP Length derived from	connect-ip, connect-ethernet	This document

		UDP header to end of IPv6 packet			
4	ipv4-header-checksum	IPv4 header checksum computed over IPv4 header	connect-ip, connect-ethernet	This document	
5	ipv4-tcp-checksum	TCP checksum computed over IPv4 pseudo-header and segment	connect-ip, connect-ethernet	This document	
6	ipv6-tcp-checksum	TCP checksum computed over IPv6 pseudo-header and segment	connect-ip, connect-ethernet	This document	
7	ipv4-udp-checksum	UDP checksum computed over IPv4 pseudo-header and segment	connect-ip, connect-ethernet	This document	
8	ipv6-udp-checksum	UDP checksum computed over IPv6 pseudo-header and segment	connect-ip, connect-ethernet	This document	

Table 6

9. References

9.1. Normative References

[CONNECT-ETHERNET]
Sede単o, A., "Proxying Ethernet in HTTP", Work in Progress, Internet-Draft, draft-ietf-masque-connect-ethernet-08, 10 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-masque-connect-ethernet-08>.

[CONNECT-IP]
Pauly, T., Ed., Schinazi, D., Chernyakhovsky, A., K端hlewind, M., and M. Westerlund, "Proxying IP in HTTP", RFC 9484, DOI 10.17487/RFC9484, October 2023, <https://www.rfc-editor.org/rfc/rfc9484>.

[CONNECT-UDP]
Schinazi, D., "Proxying UDP in HTTP", RFC 9298, DOI 10.17487/RFC9298, August 2022, <https://www.rfc-editor.org/rfc/rfc9298>.

[HTTP-DATAGRAMS]
Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August 2022, <https://www.rfc-editor.org/rfc/rfc9297>.

[INCREMENTAL-CHECKSUM]
Rijsinghani, A., Ed., "Computation of the Internet Checksum via Incremental Update", RFC 1624, DOI 10.17487/RFC1624, May 1994, <https://www.rfc-editor.org/rfc/rfc1624>.

[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>.

[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>.

[STRUCTURED-HTTP]
Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

9.2. Informative References

[IANA-POLICY]
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>.

[IPv4] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/rfc/rfc791>.

[IPv6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/rfc/rfc8200>.

[TCP] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/rfc/rfc9293>.

[TCP-PERF] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/rfc/rfc7323>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com

Tommy Pauly
Apple
Email: tpauly@apple.com