Multiplexed Application Substrate over QUIC Encryption
Internet-Draft
Intended status: Standards Track
Expires: 19 April 2026

Y. Rosomakho
Zscaler

16 October 2025

Reusable templates and checksum offload for CONNECT-IP
draft-rosomakho-masque-connect-ip-optimizations-00

Abstract

This document defines extensions to the CONNECT-IP protocol (RFC 9484) that improve the efficiency of datagram transmission by introducing reusable templates and checksum offload capabilities.

Reusable templates allow endpoints to associate Context Identifiers with static portions of packet headers, enabling datagrams to omit repeated byte sequences while remaining self-contained and stateless on the wire. Checksum offload enables endpoints to delegate computation of transport-layer checksums to the receiver by signaling the relevant offsets within the reconstructed packet.

These optimisations reduce per-packet overhead, processing cost, and effectively increase the usable maximum transmission unit (MTU) when CONNECT-IP 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 specified in Section 2.1 of [HTTP-DATAGRAMS] or DATAGRAM capsules specified in Section 3.5 of [HTTP-DATAGRAMS]. Each packet is carried in full, including all transport and network headers, which provides simplicity and interoperability but incurs per-packet overhead due to the repeated transmission of largely invariant header fields.

This document introduces two optional extensions, Reusable Templates and Checksum Offload, that optimise CONNECT-IP datagram transmission without changing the existing protocol semantics.

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 maintaining on-the-wire statelessness and compatibility with intermediaries that are unaware of these optimisations.

Checksum offload enables endpoints to delegate computation of transport-layer checksums to the receiver by identifying the checksum field offset and the start of the checksum coverage within the reconstructed packet. This mirrors hardware checksum-offload behavior used on network interfaces and tunnel devices, reducing per-packet CPU cost for encapsulating or decapsulating CONNECT-IP traffic.

When CONNECT-IP 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.

Both extensions are negotiated at CONNECT-IP establishment and signalled using Capsules on the reliable control stream. When necessary, endpoints can fall back to transmitting complete IP packets using Context ID 0, which represents unoptimised datagrams containing the full IP packet as defined in Section 5 of [CONNECT-IP].

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 (CID): A numeric identifier associated with a specific packet reconstruction and processing behavior. A CONNECT-IP tunnel can maintain multiple CIDs in each direction. Context ID 0 always represents transmission of complete, unoptimised IP packets as defined in Section 5 of [CONNECT-IP].

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

Checksum Offload: A capability allowing the receiver to compute and finalize the Internet checksum according to [INCREMENTAL-CHECKSUM] for the reconstructed packet, based on two offsets: Checksum Field Offset and Checksum Start Offset.

Capsule: A reliable control-stream message, as defined in Section 3 of [HTTP-DATAGRAMS], used in this specification to signal creation or deletion of Context Identifiers and their associated templates.

3. Negotiation of Capabilities

Endpoints negotiate support for these optimizations when establishing a CONNECT-IP tunnel by using a connect-ip-optimizations HTTP header field, whose value is a Structured Field Dictionary as defined in Section 3.2 of [STRUCTURED-HTTP].

3.1. Header Definition

```
connect-ip-optimizations = sf-dictionary
```

Figure 1: Connect-ip-optimizations header field

This document defines the following optional dictionary keys:

templates (sf-interger): Indicates support for reusable templates and specifies the maximum number of templates that the sender is willing to maintain for templates received from the peer. A positive integer value indicates full support for reusable templates, and the value represents an upper bound on the number of concurrently active templates that can be created by the peer. A value of 0 indicates that the sender supports reusable templates only for its own transmissions but does not accept templates created by the peer. If this member is omitted, reusable templates are not supported in either direction.

checksum (sf-boolean): Indicates support for checksum offload semantics. A value of ?1 means that checksum offload is supported in both directions. A value of ?0 means that checksum offload may be used for packets sent by this endpoint but not for packets received from the peer. If this member is omitted, checksum offload is not supported in either direction.

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

3.2. Negotiation Behavior

If both peers indicate support for templates (that is, templates member is present and non-zero in both directions), either endpoint MAY create and delete Context Identifiers and their templates using capsules as described in Section 4.

If both peers indicate support for checksum offload (that is, checksum=?1 in both directions), either endpoint MAY install checksum offload parameters for specific Context Identifiers using capsules as described in Section 4.2.

Peers of endpoints that advertise support in only one direction (for example, checksum=?0 or templates=0) MUST NOT send capsules that would require those endpoints to maintain state for the corresponding capability.

Capabilities that were not successfully negotiated MUST NOT be used within the tunnel.

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
connect-ip-optimizations = templates=20000, checksum=?1
```

Figure 2: CONNECT-IP with connect-ip-optimizations request example

HTTP/3 sample response (proxy to client):

```
:status = 200
capsule-protocol = ?1
connect-ip-optimizations = templates=65535, checksum=?0
```

Figure 3: CONNECT-IP with connect-ip-optimizations response example

In this example, both peers support reusable templates, and checksum offload is supported only for packets sent from the proxy to the client.

After this exchange, both endpoints may define Context Identifiers and associated templates and/or checksum parameters using capsules on the reliable control stream.

4. Optimization capsules

This specification defines two capsule types:

CONNECT_IP_OPTIMIZATION_CREATE (capsule type 0x1a768469): Creates an immutable optimization context bound to a Context ID.

CONNECT_IP_OPTIMIZATION_DELETE (capsule type 0x1a76846a): Retires a CID.

All capsules are sent on the reliable control stream of the CONNECT-IP tunnel.

4.1. Context ID allocation

The Context ID carried in these capsules is encoded as a QUIC variable-length integer defined in Section 16 of [QUIC]. Even-numbered CIDs are allocated by the client, and odd-numbered CIDs are allocated by the proxy, consistent with Section 4 of [CONNECT-UDP]. CID 0 is reserved for unoptimized raw packets and MUST NOT appear in these capsules.

4.2. CONNECT_IP_OPTIMIZATION_CREATE capsule

CONNECT_IP_OPTIMIZATION_CREATE capsule is used to create a new, immutable optimization context for the indicated CID.

```
CONNECT_IP_OPTIMIZATION_CREATE Capsule {
  Type (i) = 0x1a768469,
  Length (i),
  Context ID (i),
  Static Segments Length (i),
  Static Segment (..) ...,
  Checksum Field Offset (i)?,
  Checksum Start Offset (i)?,
}
```

Figure 4: CONNECT_IP_OPTIMIZATION_CREATE Capsule Format

CONNECT_IP_OPTIMIZATION_CREATE capsule contains the following fields:

Context ID: Context Identifier, encoded as a variable-length integer, defined by this capsule.

Static Segments Length: Aggregate length in bytes of all subsequent Static Segments. A value of 0 indicates that no template is used (that is, the payload of datagrams for the given Context ID contains a full reconstructed packet, modulo checksum finishing if configured).

Checksum Field Offset: An optional field, encoded as a variable-length integer, containing byte offset of the 16-bit Internet checksum field within the reconstructed packet. This field is omitted if checksum offload is not used.

Checksum Start Offset: An optional field, encoded as a variable-length integer, containing byte offset where checksum coverage begins. Coverage runs from this offset to the end of the reconstructed packet. Not included in the capsule if checksum offloading is not used.

The CONNECT_IP_OPTIMIZATION_CREATE capsule contains a sequence of zero 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: Number of bytes in this static segment, encoded as a variable-length integer

Segment Payload: the static bytes to insert at the Segment Offset

4.2.1. Parsing and validation

The receiver parses an CONNECT_IP_OPTIMIZATION_CREATE capsule by reading, in order: the Context ID, the Static Segments Length, zero or more static segments whose encodings consume exactly Static Segments Length bytes, and then two checksum offsets if they are present.

The Context ID MUST be non-zero, even-numbered when created by the client, and odd-numbered when created by the proxy. The Context ID MUST NOT have been used previously. Static Segments Length is the total size, in bytes, of the static segments section including each Segmenet Offset, Segment Length, and Segment Payload.

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.

After Static Segments Length bytes have been consumed, the capsule either ends immediately or contains exactly two additional fields: Checksum Field Offset followed by Checksum Start Offset. If only one variable-length integer is present, or if any bytes remain after the two length integers, the capsule is malformed.

A receiver that did not negotiate acceptance of checksum offload in its direction as defined in Section 3 MUST treat an CONNECT_IP_OPTIMIZATION_CREATE capsule that includes checksum offsets as an error and MUST follow the error-handling procedure in Section 3.3 of [HTTP-DATAGRAMS]. A receiver that has already accepted the maximum number of templates it advertised via the templates member of connect-ip-optimizations MUST treat any additional CONNECT_IP_OPTIMIZATION_CREATE capsule containing a template (that is, with Static Segments Length > 0) as an error and MUST follow the same error-handling procedure.

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].

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

4.3. CONNECT_IP_OPTIMIZATION_DELETE capsule

CONNECT_IP_OPTIMIZATION_DELETE capsule is used to indicate that a Context ID previously defined by CONNECT_IP_OPTIMIZATION_CREATE capsule will no longer be used.

```
CONNECT_IP_OPTIMIZATION_DELETE Capsule {
  Type (i) = 0x1a76846a,
  Length (i),
  Context ID (i),
}
```

Figure 6: CONNECT_IP_OPTIMIZATION_DELETE Capsule Format

After sending an CONNECT_IP_OPTIMIZATION_DELETE Capsule, the sender MUST NOT use the Context ID. Upon receipt, the peer retires the indicated context and releases any negotiated template budget consumed by that context if, and only if, the retired context included a template (that is, its CONNECT_IP_OPTIMIZATION_CREATE had a non-zero Static Segments Length). Retiring a context that had no template (for example, checksum-only) does not affect the template budget.

If an CONNECT_IP_OPTIMIZATION_DELETE capsule is received for a Context ID that was not previously and correctly defined by an CONNECT_IP_OPTIMIZATION_CREATE capsule from the peer, the receiver MUST follow the error-handling procedure defined in Section 3.3 of [HTTP-DATAGRAMS].

5. Optimized Datagram Operation

This section defines how endpoints construct and consume datagrams once a Context ID has been created with CONNECT_IP_OPTIMIZATION_CREATE capsule.

5.1. Sender behavior

For each packet sent under a Context ID, the sender constructs the datagram payload according to the context's template. If the context has no template (that is, Static Segments Length is 0), the payload MUST be a complete reconstructed packet. If the context has a template, the payload MUST be the concatenation of all variable byte ranges, taken in strictly increasing offset order, corresponding to the gaps not covered by static segments starting at offset 0.

When checksum offload is configured (that is, the context includes Checksum Field Offset and Checksum Start Offset), the sender MUST set the checksum field in the reconstructed packet image to the 16-bit one's-complement sum of the appropriate pseudo-header before transmission. The two bytes at Checksum Field Offset in the reconstructed image therefore MUST contain this pseudo-header sum; these bytes originates from static bytes in the template as well as from variable bytes in the payload. The final reconstructed image MUST contain the correct preseeding value.

A sender uses Context ID 0 for any one-off packet that does not fit an existing context (for example, a transient change in header layout).

5.2. Receiver behavior

A datagram that arrives with a non-zero Context ID that has not been previously and correctly defined by an CONNECT_IP_OPTIMIZATION_CREATE capsule from the peer MUST be dropped.

If the CID has no template (that is, Static Segments Length is 0), the datagram payload is the complete reconstructed packet. If the CID has a template, the receiver reconstructs the packet from offset 0 upward by writing static bytes at their declared offsets and filling all remaining byte positions with bytes consumed from the datagram payload in strictly increasing offset order. Reconstruction continues until all payload bytes have been consumed. If the payload is exhausted before the offset of the last static segment have been filled, the packet MUST be dropped.

When Checksum Field Offset and Checksum Start Offset are present for the CID, the receiver finishes the Internet checksum as follows: it computes the checksum over the byte range starting at Checksum Start Offset and ending at the reconstructed packet length while treating the checksum field as zero during the sum; it then adds (folds) the 16-bit value currently at Checksum Field Offset, performs end-around carry, and writes the final one’s-complement result back at Checksum Field Offset. If either checksum offset is greater than or equal to the reconstructed packet length for this packet, 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. TCP over IPv6 with template 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].

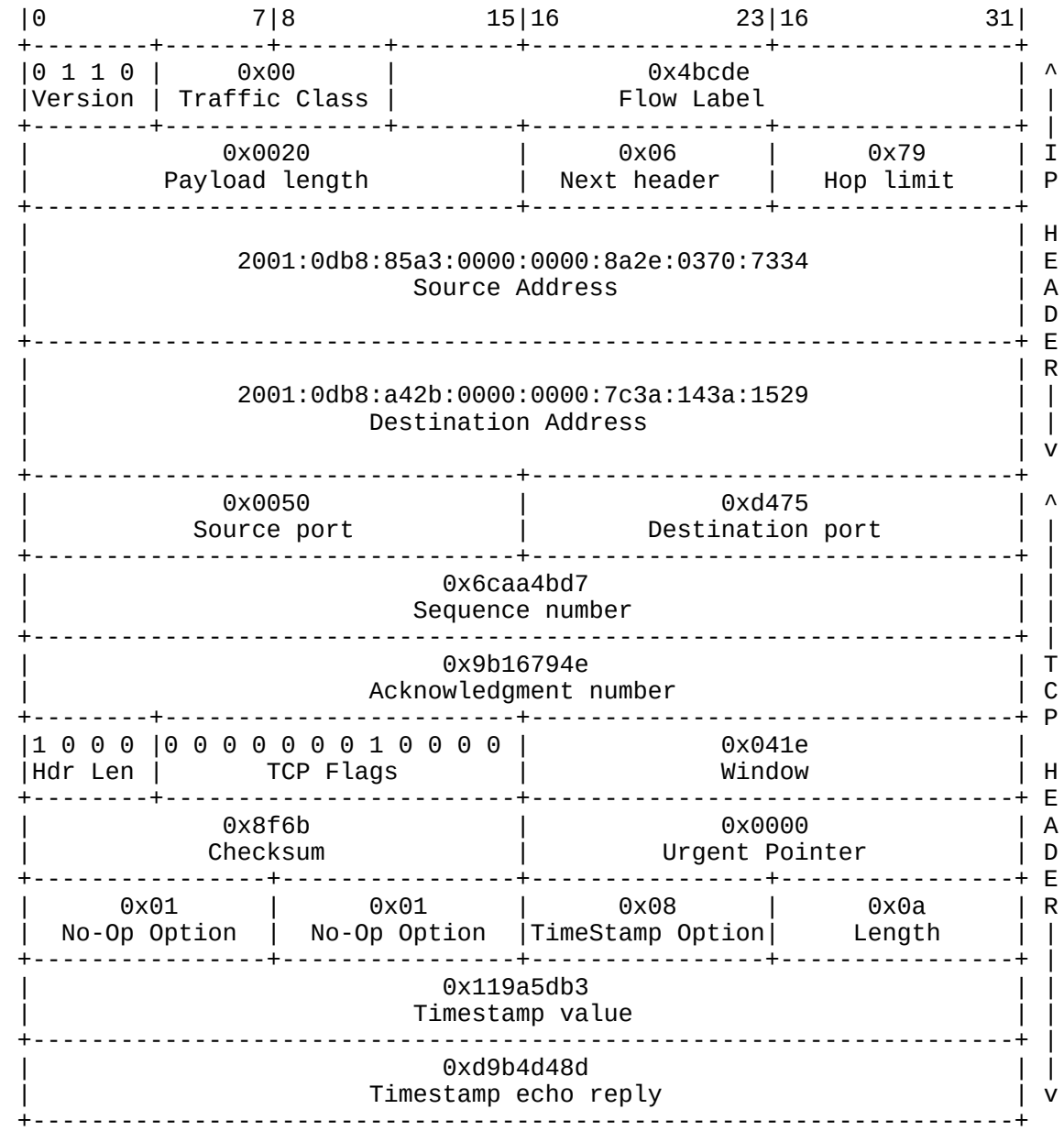

Figure 7: Example TCP over IPv6 packet before optimization

Table below illustrates fields present in IPv6 and TCP headers, 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	Payload length	16	0x0020	No
48	Next header	8	0x06	Yes
56	Hop limit	8	0x79	Yes
64	Source address	128	2001:0db8:85a3::8a2e:0370:7334	Yes
192	Destination address	128	2001:0db8:a42b::7c3a:143a:1529	Yes
320	Source port	16	0x0050	Yes
336	Destination port	16	0xd475	Yes
352	Sequence number	32	0x6caa4bd7	No
384	Acknowledgement number	32	0x9b16794e	No
416	TCP header length	4	1000b	Yes
420	TCP Flags	12	000000010000b	No
432	Window	16	0x041e	No
448	Checksum	16	0x8f6b	No
464	Urgent pointer	16	0x0000	Yes
480	No-Op option	8	0x01	Yes
488	No-Op option	8	0x01	Yes
496	Timestamp option	8	0x08	Yes

504	Timestamp option length	8	0x0a	Yes
512	Timestamp value	32	0x119a5db3	No
544	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	4	Version, Traffic Class and Flow Label	0x6004bcde
6	38	Next header, Hop limit, Source address, Destination address, Source port, Destination port	0x067920010db885a3...
58	6	Urgent pointer, 2 No-Op TCP options, Timestamp option code and length	0x00000101080a

Table 2: Static segments for example IPv6/TCP packet

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

```
CONNECT_IP_OPTIMIZATION_CREATE Capsule {
  Type (i) = 0x1a768469,
  Length (i) = 58,
  Context ID (i) = 2,
  Static Segments Length (i) = 54,
  Static Segment {
    Segment Offset (i) = 0,
    Segment Length (i) = 4,
    Segment Payload = 0x6004bcde,
  },
  Static Segment {
    Segment Offset (i) = 6,
    Segment Length (i) = 38,
    Segment Payload = 0x067920010db885a3000000008a2e0370733420010db8a42b000000007c3a143a1
5290050d475,
  },
  Static Segment {
    Segment Offset (i) = 58,
    Segment Length (i) = 6,
    Segment Payload = 0x00000101080a,
  },
  Checksum Field Offset (i) = 56,
  Checksum Start Offset (i) = 40,
}
```

Figure 8: CONNECT_IP_OPTIMIZATION_CREATE Capsule for example IPv6/TCP packet

This template reduces per-packet overhead by omitting 48 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 and replaces checksum field with the IPv6/TCP pseudo-header checksum. 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.

6.2. UDP over IPv4 with template and without checksum offload

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

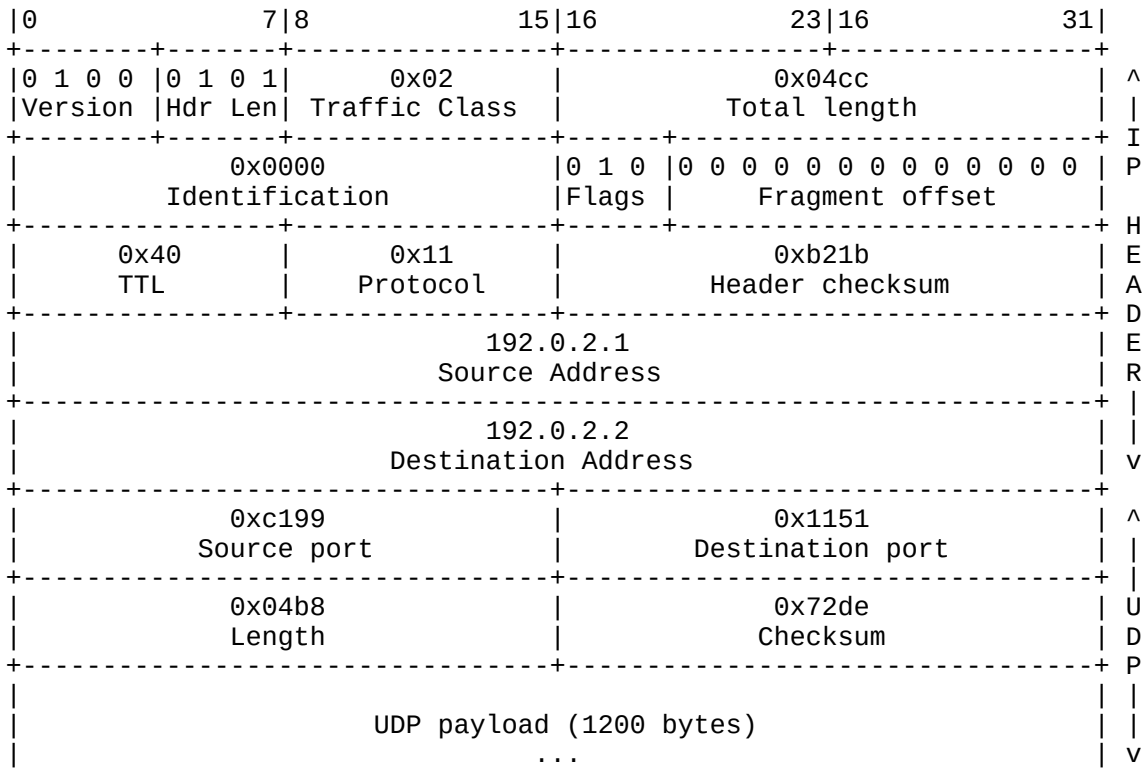

Figure 9: Example UDP over IPv4 packet before optimization

Table below illustrates fields present in IPv4 header and UDP, their offsets in bits from the beginning of the packet and if they are likely to be static for most packets of a given traffic flow

Offset	Field name	Length	Value	Static
0	Version	4	0100b	Yes
4	Header length	4	0101b	Yes
8	Traffic Class	8	0x02	Yes
16	Total length	16	0x04cc	No
32	Identification	16	0x0000	Yes
48	Flags	3	010b	Yes
51	Fragment offset	13	0000000000000b	Yes
64	TTL	8	0x40	Yes
72	Protocol	8	0x11	Yes
80	Header checksum	16	0xb21b	No
96	Source address	32	192.0.2.1	Yes
128	Destination address	32	192.0.2.2	Yes
160	Source port	16	0xc199	Yes
176	Destination port	16	0x1151	Yes
192	Length	16	0x04b8	No
208	Checksum	16	0x72de	No
224	UDP payload	9600	...	No

Table 3: IPv4 header and UDP fields in example packet

Static segments model the invariant parts:

Segment Offset	Segment Length	Segment Contents	Segment Payload
0	2	Version, Header length and Traffic Class	0x4502
4	6	Identification, Flags, Fragment offset, TTL and Protocol	0x000040004011
12	12	Source address, Destination address, Source port and Destination port	0xc0000201c0000202c1991151

Table 4: Static segments for example IPv4/UDP packet

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

```
CONNECT_IP_OPTIMIZATION_CREATE Capsule {
  Type (i) = 0x1a768469,
  Length (i) = 28,
  Context ID (i) = 3,
  Static Segments Length (i) = 26,
  Static Segment {
    Segment Offset (i) = 0,
    Segment Length (i) = 2,
    Segment Payload = 0x4502,
  },
  Static Segment {
    Segment Offset (i) = 4,
    Segment Length (i) = 6,
    Segment Payload = 0x000040004011,
  },
  Static Segment {
    Segment Offset (i) = 12,
    Segment Length (i) = 12,
    Segment Payload = 0xc0000201c0000202c1991151,
  }
}
```

Figure 10: CONNECT_IP_OPTIMIZATION_CREATE Capsule for example IPv4/UDP packet

This template omits 20 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. Because this template does not define checksum offload, the UDP checksum is computed by the sender as usual; the receiver does not perform checksum finishing for this context.

7. Security Considerations

This specification changes how CONNECT-IP datagrams are constructed 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.

Context state can be abused for resource exhaustion. Endpoints enforce negotiated limits from connect-ip-optimizations; they MUST reject creations that exceed the declared template budget and must release budget when a context is retired with CONNECT_IP_OPTIMIZATION_DELETE. Implementations SHOULD bound the number of static segments, validate lengths before allocation and cap per-context memory.

Negotiation and Capsule handling are directional and immutable to reduce desynchronization risk. Even/odd Context ID allocation prevents collisions between endpoints; CID 0 is reserved and must not appear in Capsules. Unknown CIDs must be dropped. Reusing a CID within the same connection after deletion is not permitted; endpoints MUST allocate a fresh CID to change behavior.

8. IANA Considerations

8.1. HTTP Capsule Types Registration

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

| Value | Capsule Type + --- + --- + | 0x1a768469 | CONNECT_IP_OPTIMIZATION_CREATE | | 0x1a76846a | CONNECT_IP_OPTIMIZATION_DELETE |

8.2. HTTP Field Name Registration

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

* Field Name: connect-ip-optimizations
* Status: provisional (permanent if approved)
* Structured Type: Dictionary
* Reference: This document

9. References

9.1. Normative References

[CONNECT-IP]
Pauly, T., Ed., Schinazi, D., Chernyakhovsky, A., Khlewind, 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

[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>.

[QUIC] 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>.

[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.

Author’s Address

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com