

# HTTP Datagram Compression

`draft-rosomakho-masque-connect-ip-optimizations-01`

Yaroslav Rosomakho

Tommy Pauly

MASQUE  
IETF125, March 2026, Shenzhen

# Notable changes in -01

- Expanded scope beyond CONNECT-IP
- Added derived fields
- Aligned capsule naming and structure with other MASQUE drafts
- Added MTU limit for reconstructed packet
- Added option to specify maximum segments per template

# Challenges with CONNECT-IP/CONNECT-ETHERNET implementation

- MTU pressures when using QUIC datagrams
- CPU time lost for computing TCP/UDP checksums

# Three stackable mechanisms proposed

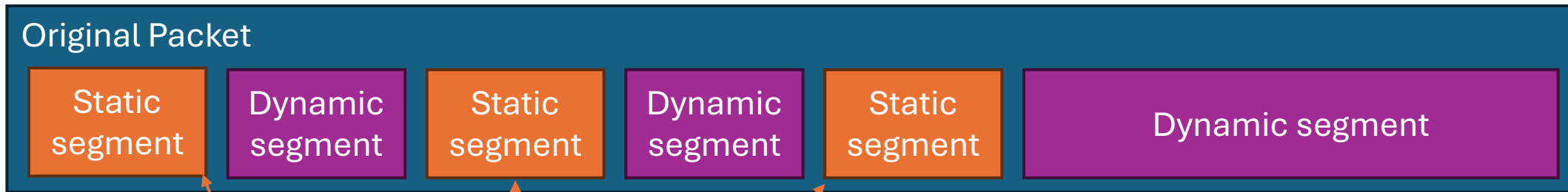
- Templates to remove static segments from the payload
- Derived fields that can be computed by the receiver
- TCP/UDP checksum offload

# Reusable templates

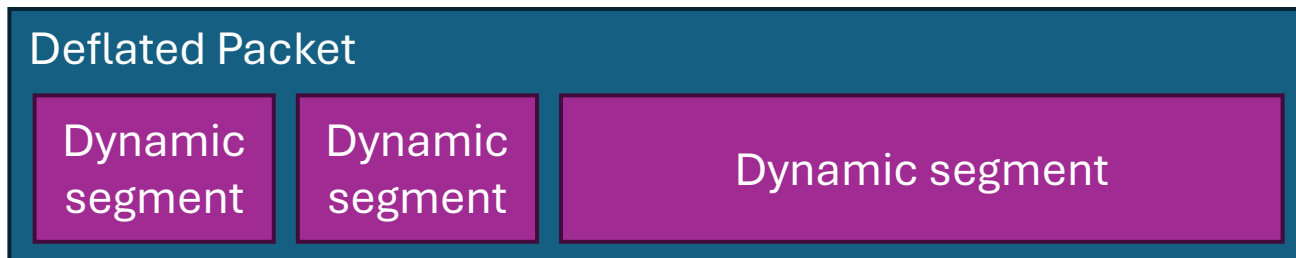
Many sections of packets *of the same flow* can be identical

- IP version
- IPv4 header length
- Traffic Class
- IPv4 id / IPv6 flow label
- IPv4 fragmentation flags / fragment offset
- IPv4 TTL / IPv6 Hop Limit
- IP addresses
- Protocol / Next Header
- Source Port / Destination Port
- TCP header length
- Urgent pointer, some TCP options
- Potentially some headers on application layer

# Splitting packet into segments



Assembled into a context ID specific template  
Each segment is prefixed with offset and length  
Segments must not overlap



# Notes about templating

- It's up to sender to decide how to split the packet. Receiver simply re-assembles the packet. It does not need to be aware of internal packet structure
- Support of templates is announced by each party as well as limit of concurrent templates
- Packets can always be sent with context id 0 (without optimizations)
- Multiple templates can be defined for a single packet flow
- Single template can apply to multiple flows
- Templating can apply to any CONNECT carrying payload in datagrams and using Capsules for signaling

# Derived fields

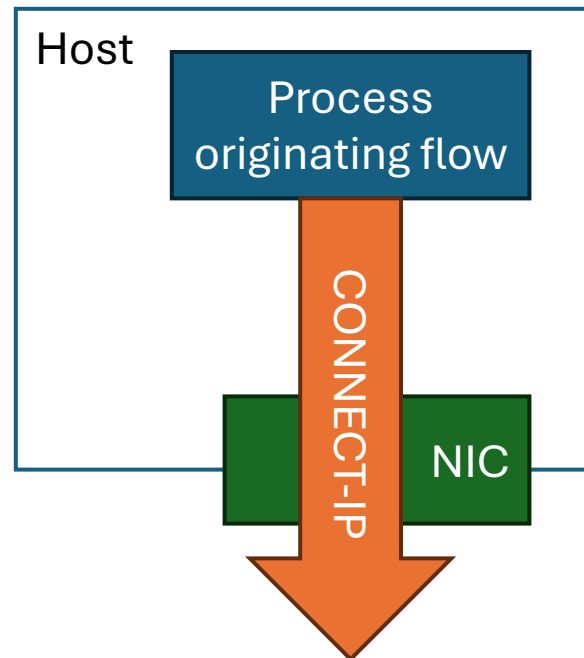
- Some packet fields can be calculated by the receiver
  - IPv4 Total Length / IPv6 Payload Length
  - UDP Length
  - IPv4 header checksum
  - TCP/UDP checksums
- Requires understanding of packet structure
- Proposed IANA registry more fields could be added in the future

# TCP/UDP Checksum offloading

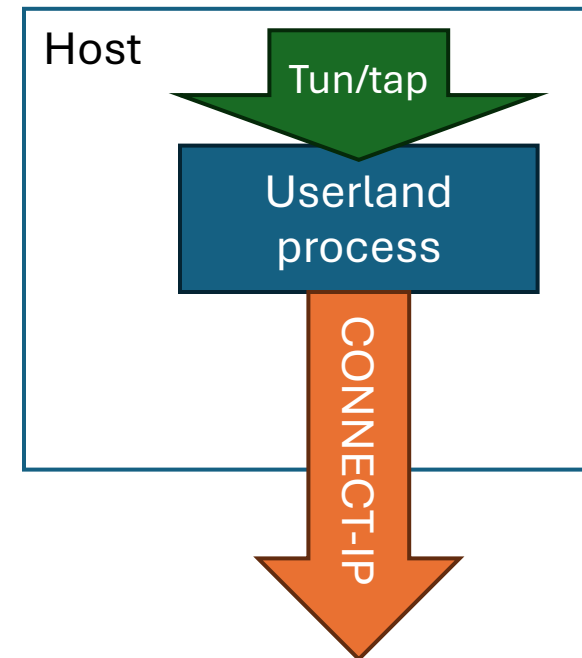
- TCP and (most) UDP packets carry internet checksum
- Checksum calculation can be computationally intensive
  - ~5% on x64 and arm64 CPUs on a fully loaded CONNECT-IP tunnel
  - Much more punishing on RISC-V architecture since it does not have add-with-carry instruction
- Wait... isn't checksum always set by sender? And don't we need to produce valid checksum to the receiver?

# Checksum must be computed by somebody

- Normally checksum is computed by NIC when packet is sent out
- But:

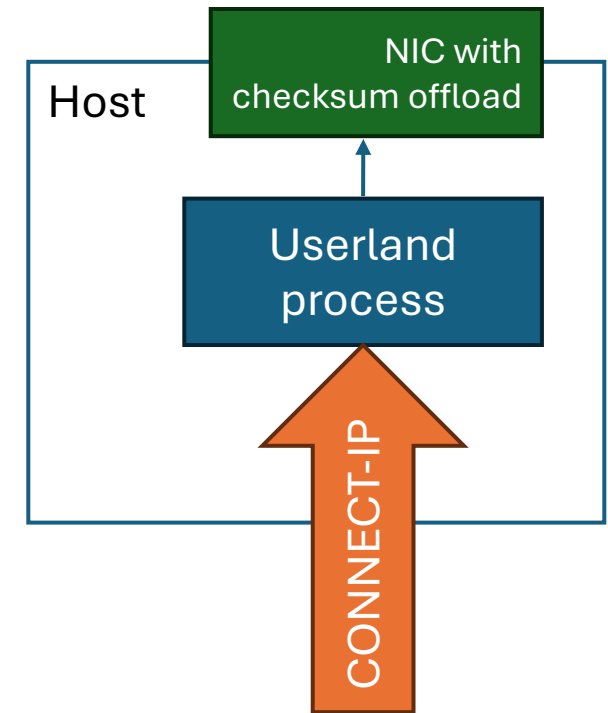
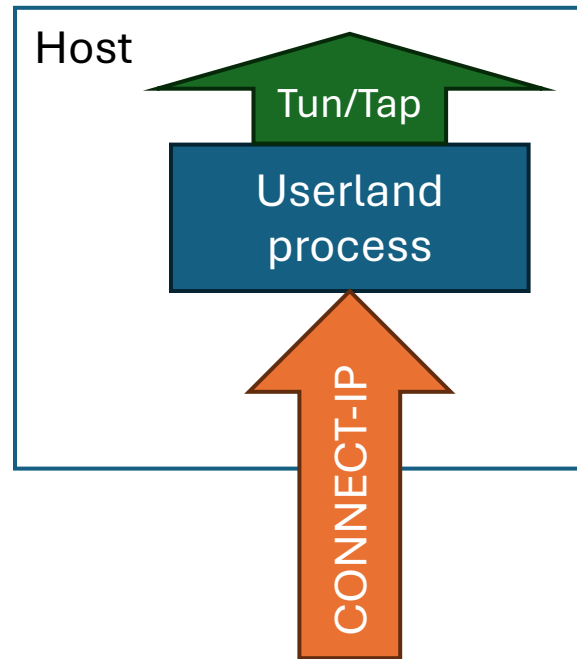
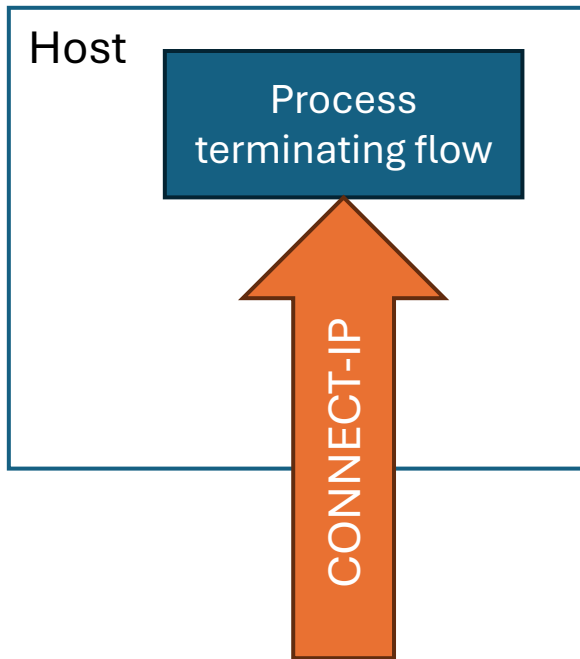


There is no NIC to offload L4 checksum of original packet  
It is computed on CPU by kernel or CONNECT-IP process



Tun/tap interface with GSO does not provide checksums  
Without GSO tunnel interface is limited by ~500kpps

# Checksum may not be required on the receiver side



# Common way to signal checksum offloading

- Process (or driver) and NIC agree on the capability
- TCP and UDP packets do include checksum of the pseudo-header
- Metadata structure associated with each packet includes
  - Offset to the checksum
  - Offset to the beginning of TCP/UDP header
- NIC calculates 1-complement and places the reconstructed checksum into packet
- Relevant APIs are available in XDP, Tun/Tap and other interfaces
- <https://www.kernel.org/doc/html/v6.17/networking/checksum-offloads.html>

# Checksum offloading

- Each party signals if it supports incoming packets with TCP/UDP checksum offload
- Sender indicates offloaded checksum when defining context id
  - Checksum offset
  - Start of checksummed data
- Packets still include checksum of the pseudo-header
- Applies to CONNECT-IP and CONNECT-ETHERNET

# Proposed Template Capsules

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

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

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

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

# Proposed Derived Fields Capsules

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

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

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

# Proposed Checksum Offload Capsules

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

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

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

# Future development based on feedback

- Define recommended “safe” templates for typical use cases
  - Reduce implementation mistakes resulting in MTU oscillation
- Checksum offloading and TCP/UDP checksum derivation cannot occur at the same time

# Next steps

- Thoughts?
- Suggestions?
- Comments?