Reusable templates and checksum offload for CONNECT-IP

`draft-rosomakho-masque-connect-ip-optimizations-00`

Yaroslav Rosomakho

MASQUE
IETF124, November 2025, Montreal

Challenges with CONNECT-IP implementation

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

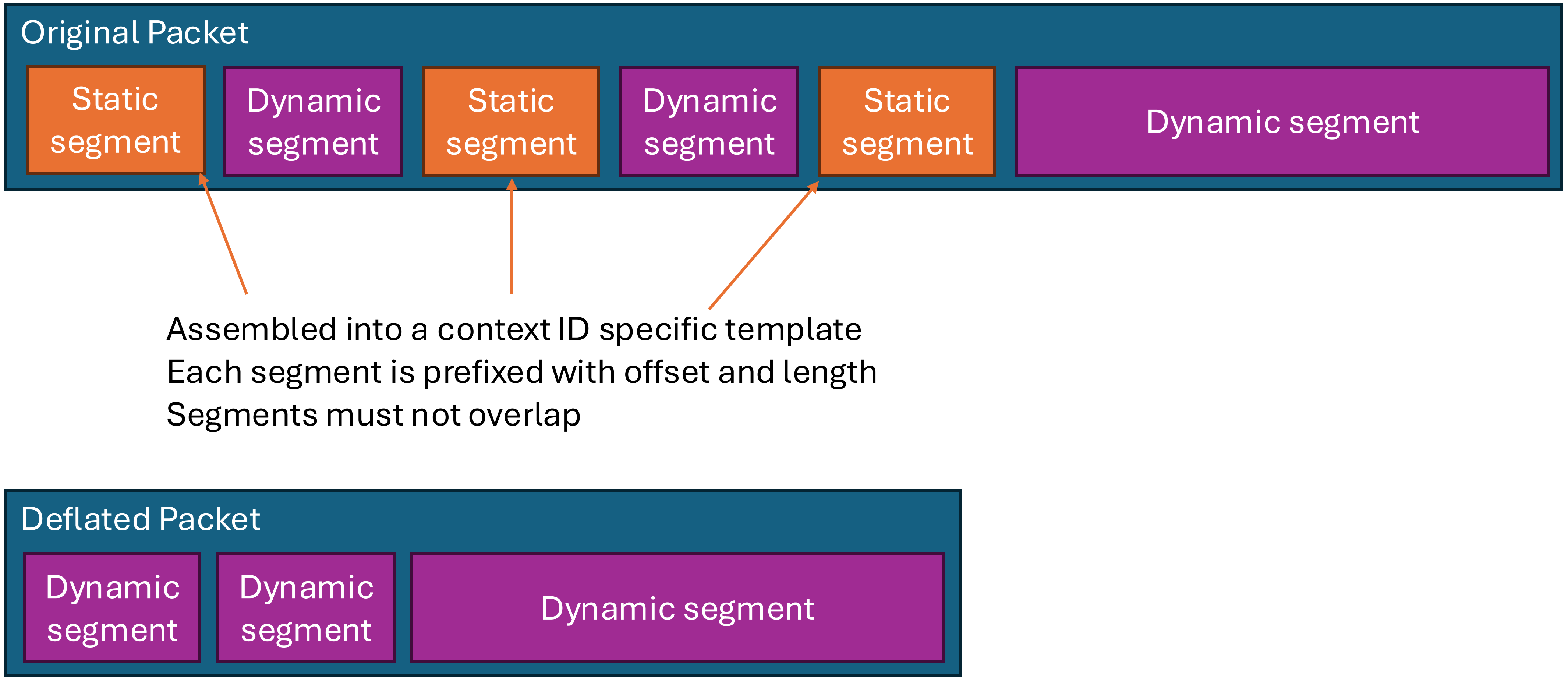
To a certain extent these challenges apply to other IP-in-X encapsulations, but we have a chance to ease them in CONNECT-IP

Reusable templates

Many sections of packets of the same flow are 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

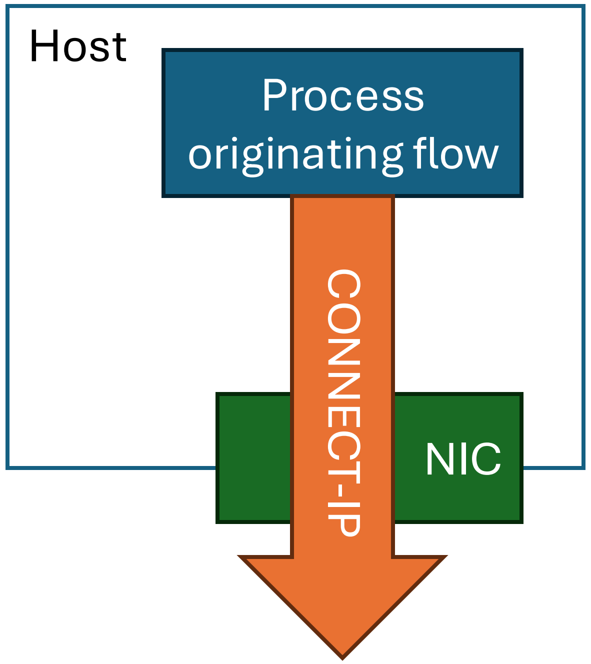
- It’s up to sender to decide how to split the packet. Receiver simply re-assembles the packet. It does not need to be aware of internal packet structure
- Support of templates is announced by each party as well as limit of concurrent templates
- Packets can always be sent with context id 0 (without optimizations)
- Multiple templates can be defined for a single packet flow

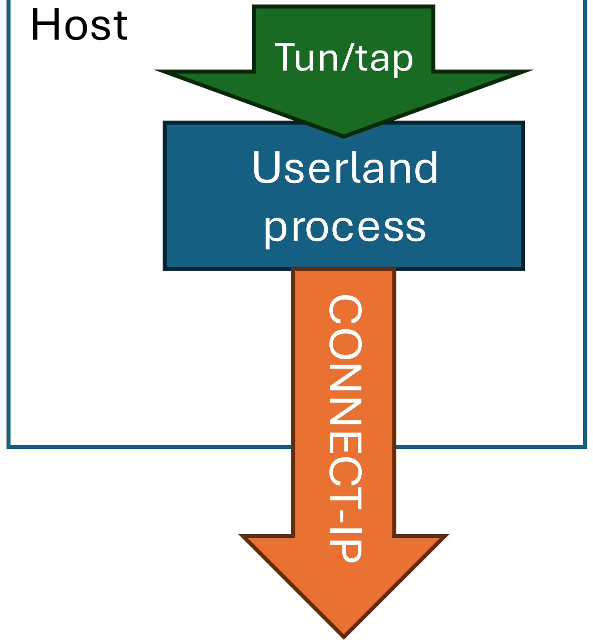
TCP/UDP Checksum offloading

- 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:

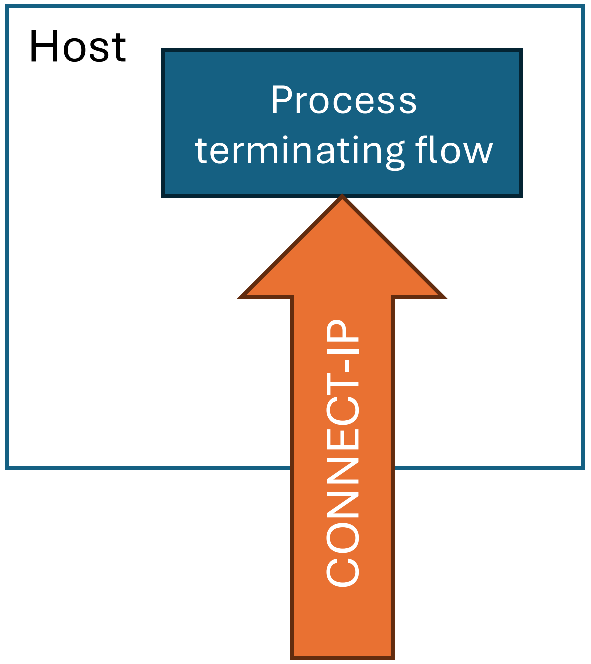


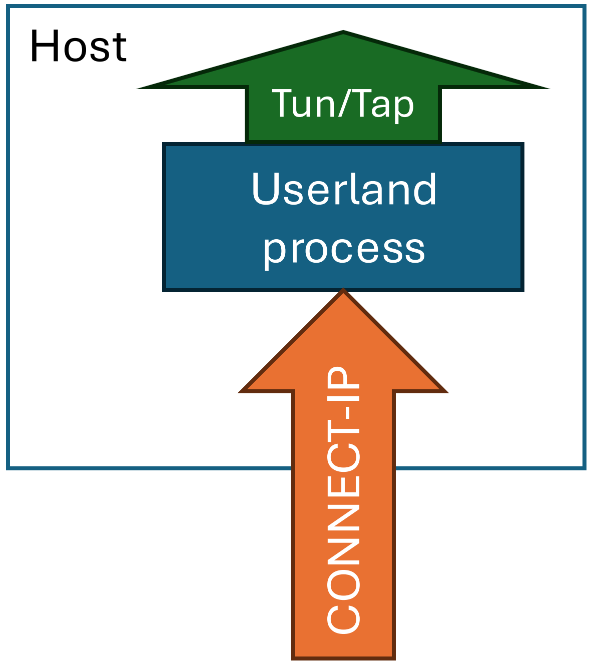


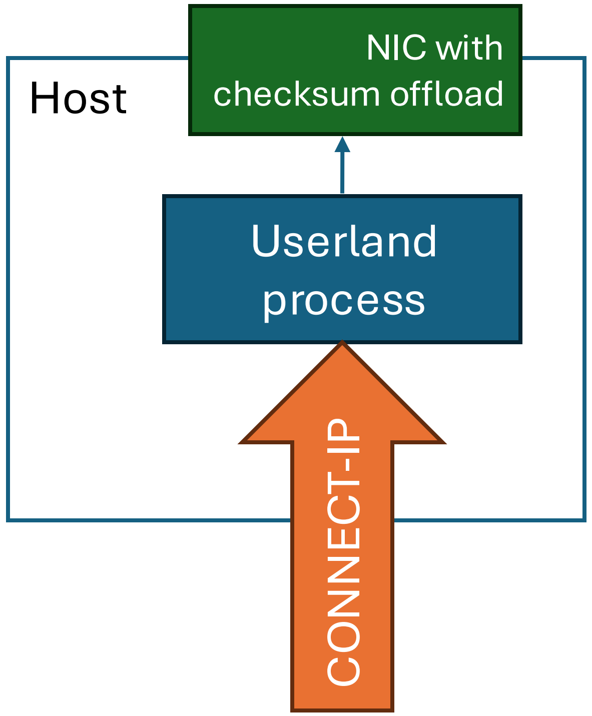
There is no NIC to offload L4 checksum of original packet
It is computed on CPU by kernel or CONNECT-IP process

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 in CONNECT-IP

- 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

Proposed Capsules

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

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

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

Future development based on initial feedback

- Extend scope to all Datagrams (Connect-UDP and Datagram Capsules)
  - How to combine this with ECN/DSCP Context id?
- Generalize optimizations to TLVs with IANA registry
  - Sometimes it is beneficial to omit L4 checksums completely
  - IPv4 header checksum may be omitted
  - Other optimizations could be added in the future

Next steps

- Thoughts?
- Suggestions?
- Comments?