# ICN Hop-By-Hop Fragmentation

Marc Mosko
Palo Alto Research Center
marc.mosko@parc.com

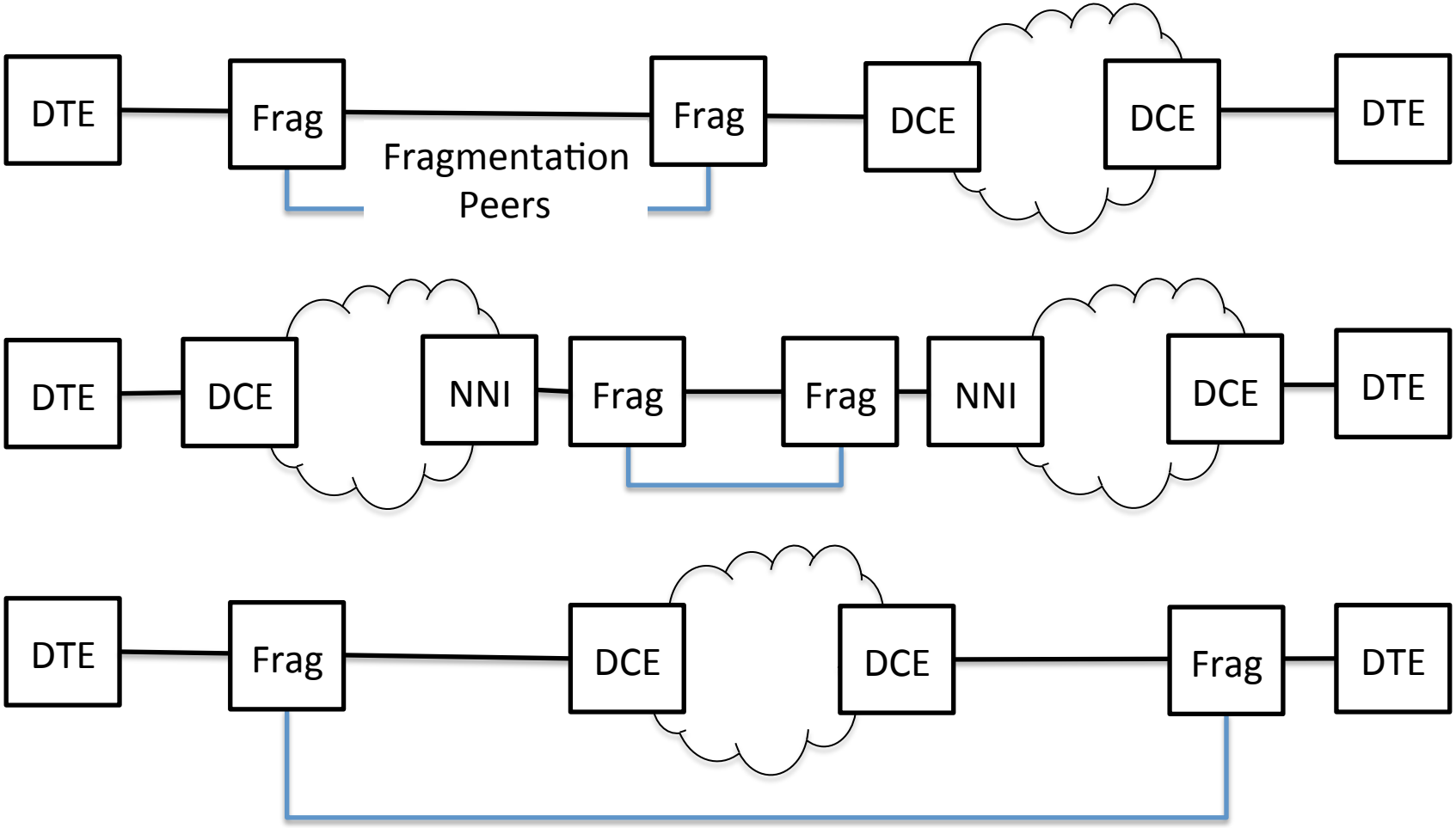
Christian Tschudin
University of Basel
christian.tschudin@unibas.ch

CCNxCON 2015, Palo Alto, CA

# Introduction

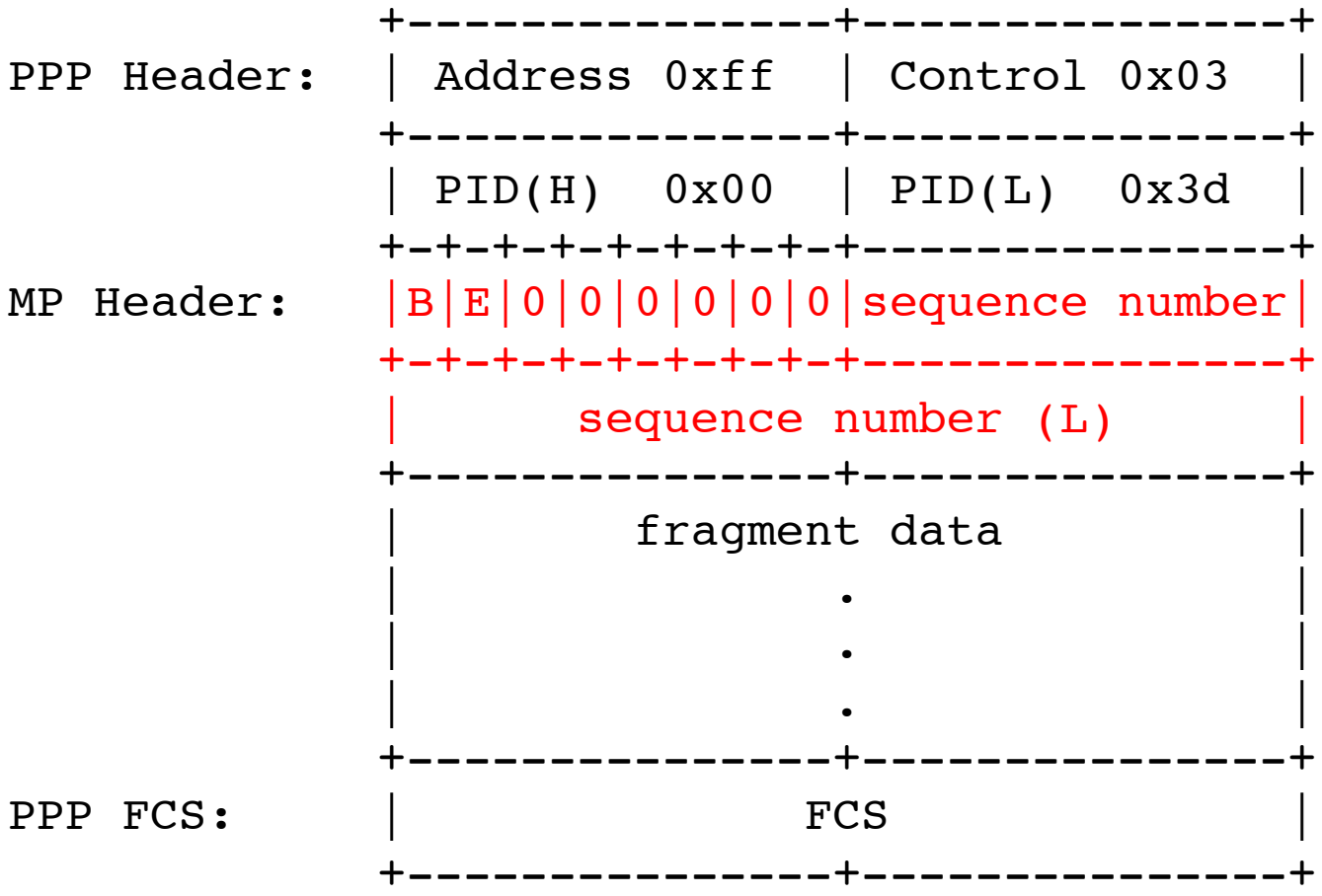
- In 2012, PARC (Mosko and Plass) worked with Tschudin to develop a *ccnb* hop-by-hop fragmentation protocol, which has been in CCN-Lite ever since.
- After the Cambridge interim meeting, Mosko and Tschudin updated the protocol for CCNx 1.0 and added ability to support other fragmentation protocols.
- The present work is not CCNx-centric, though we do only show CCNx 1.0 encodings of the protocol.

# Possibilities (a la FRF.12)




https://www.broadband-forum.org/technical/download/FRF.12/frf12.pdf

# Multilink PPP (RFC 1990, Aug 1996)



# Multilink PPP Algorithm

- B = Begin Fragment
- E = End Fragment (may have BE in one packet)
- Algorithm
  1. Discard until you receive a B
  2. If missed sequence number, goto 1
  3. Assemble until E, pass up stack
- Expects that packets arrive in order between peers (i.e. “point-to-point”)

# Adaption of PPP to ICN

- Used *inside* an ICN network
  (we would expect an IP tunnel to use IP fragmentation)
- Carry a fragmentation protocol ID to allow different fragmentation protocols (experimentation)
- We specify basic protocol like multilink PPP
- Allow for variable-size sequence number

# Terminology

A fragment is not a packet, or confusion ensues ...

- (Network Protocol) Packet:
  *the thing to be fragmented*
- Fragment:
  *byte array containing all serialized data fields*
  - fragment header: serialized control data
  - fragment data: payload
- Frame:
  *A layer-2 datagram for transferring one or more fragments*

# Essential data

## (expressed in a encoding agnostic way)

```
Fragment        = FragProtocolID FragProtocolData
FragProtocolID  = <SequencedFragmentsProtocolID> /
                  (other protocols' ids)
FragProtocolData = SequencedFragmentsData /
                   (other protocols' data)
SequencedFragmentsData = Flags FragSeqNo FragLength FragData
Flags           = B / E / BE / I
B               = <begin flag>
E               = <end flag>
BE              = <begin and end flag>
I               = <idle flag>
FragSeqNo       = 1*OCTETS
FragLength      = <Octets of fragment data>
FragData        = <Continuous octets (portion of Packet)>
```

# Peers

- A sender keeps a sequence number per peer.
- A receiver keeps a reassembly buffer and sequence number per peer.
- A peer is (but not limited to):
  - An IP socket (tcp, udp, multicast, gre, etc.) [use IP frag].
  - An ethernet tuple (dmac, smac, ethertype), including group dmac addresses.
  - The MAC address 01:00:5E:00:17:AA is the CCNx assigned group address for its 224.0.23.170 IP multicast address.
  - Could use security associations instead of addresses (e.g. 802.1x MKAs, ipsec SAs, etc.).

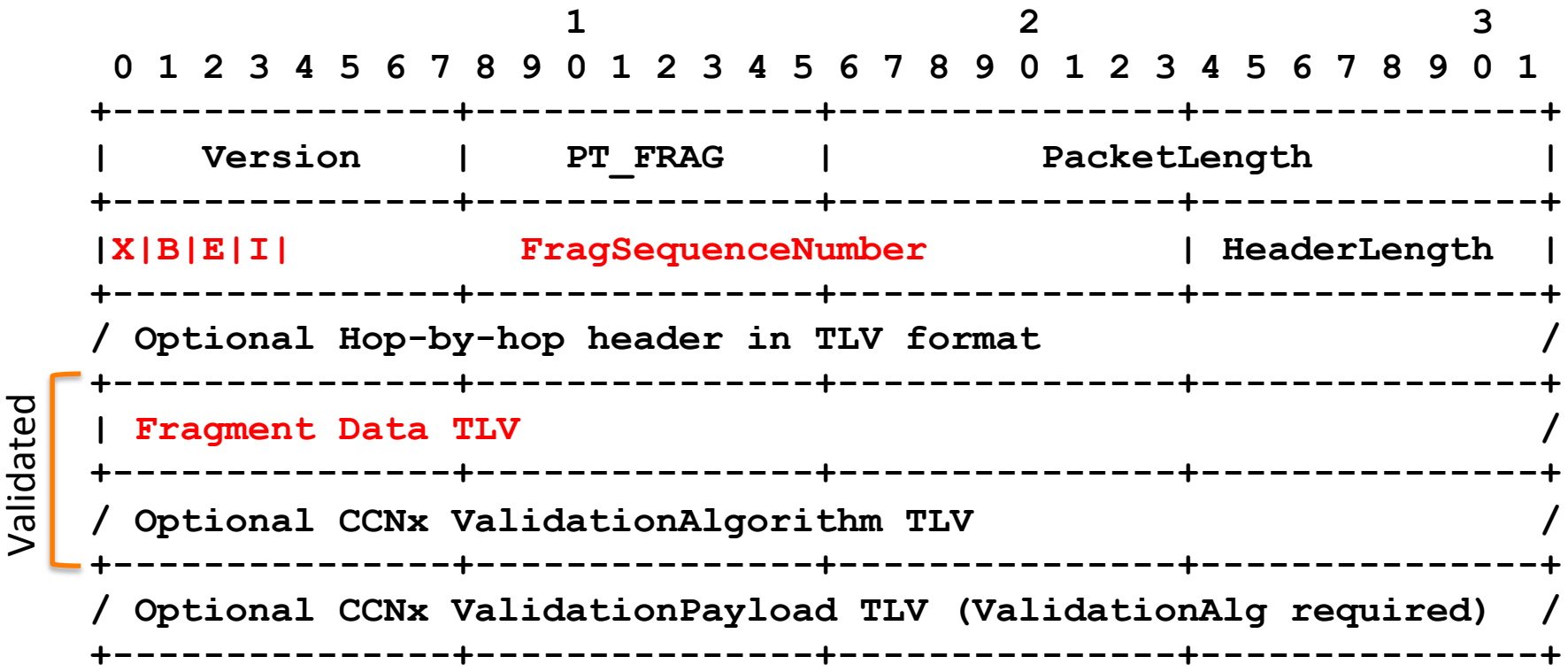
# (full) Sender Protocol

1. A sender maintains a separate state machine for each peer.
2. When a peering is established, the FragSequenceNumber begins at 0.
3. After sending a Fragment, FragSequenceNumber is incremented by one.
4. In the first fragment for a packet, set the B bit to '1'.
5. In the last fragment for a packet, set the E bit to '1'.
6. Both the B and E bits must be set to '1' for a single fragment.
7. If both the B and E and I bits are not set, the fragment is in the middle of a series.
8. When not sending a fragment (with fragment data), the sender may send an Idle fragment with only the 'I' bit set. This indicates that the sender has no packet to send. Idle frames may only be sent in between E and B frames.

# (full) Receiver Protocol

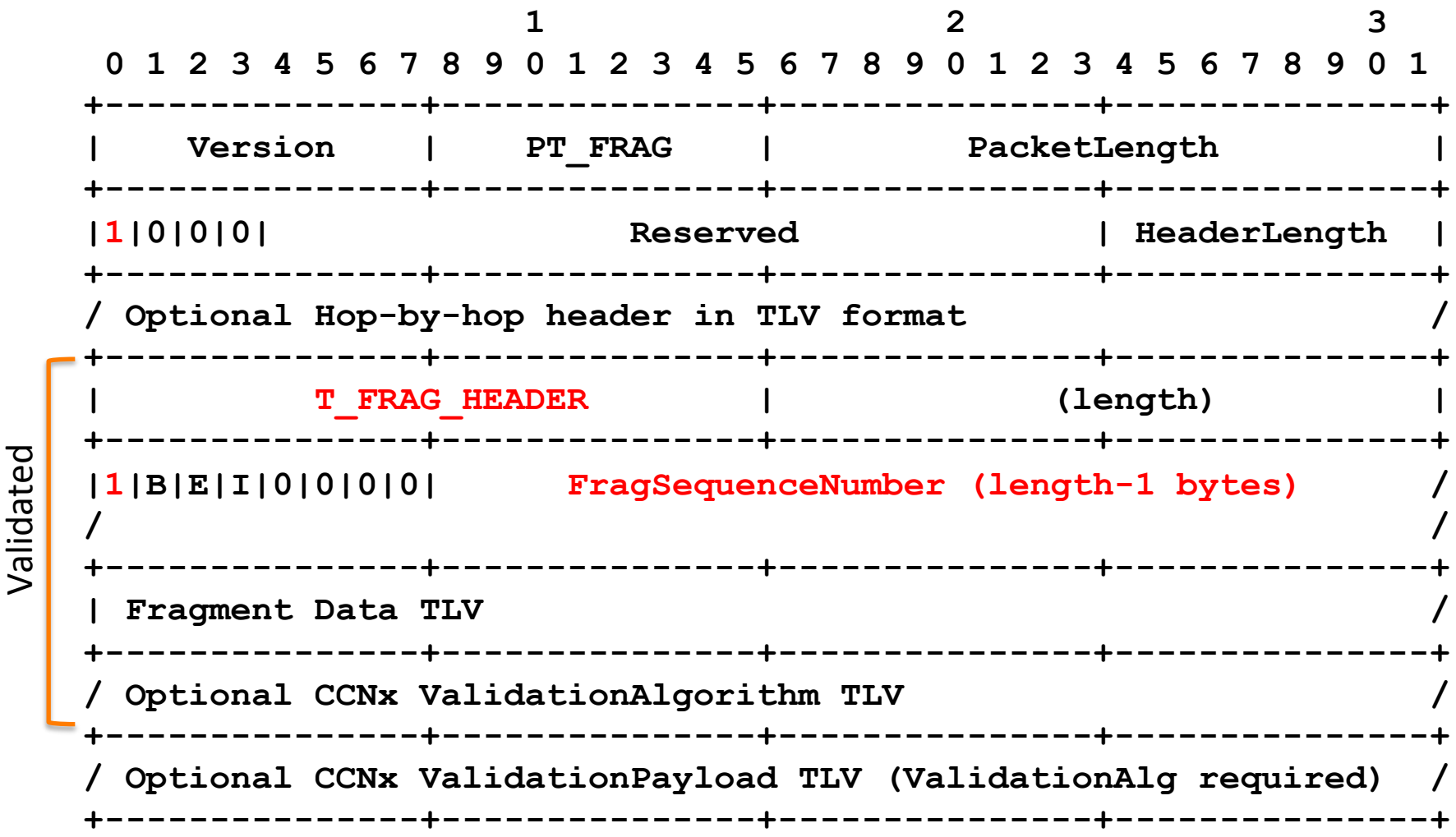
1. A receiver maintains one reassembly queue per peer.
2. Discard Idle fragments.
3. Discard fragments until a 'B' fragment is received. Store the received sequence number for this sender.
4. If an out-of-order fragment is received next, discard the reassembly buffer and go to step (2).
5. Continue receiving in-order fragments until the first 'E' fragment. At this time, the fragmented packet is fully re-assembled and may be passed on to the next layer.
6. The receiver cannot assume it will receive the 'E' fragment or a subsequence 'I' frame, so it should use a timeout mechanism appropriate to the link to release allocated memory resources.

# CCNx Basic Encoding

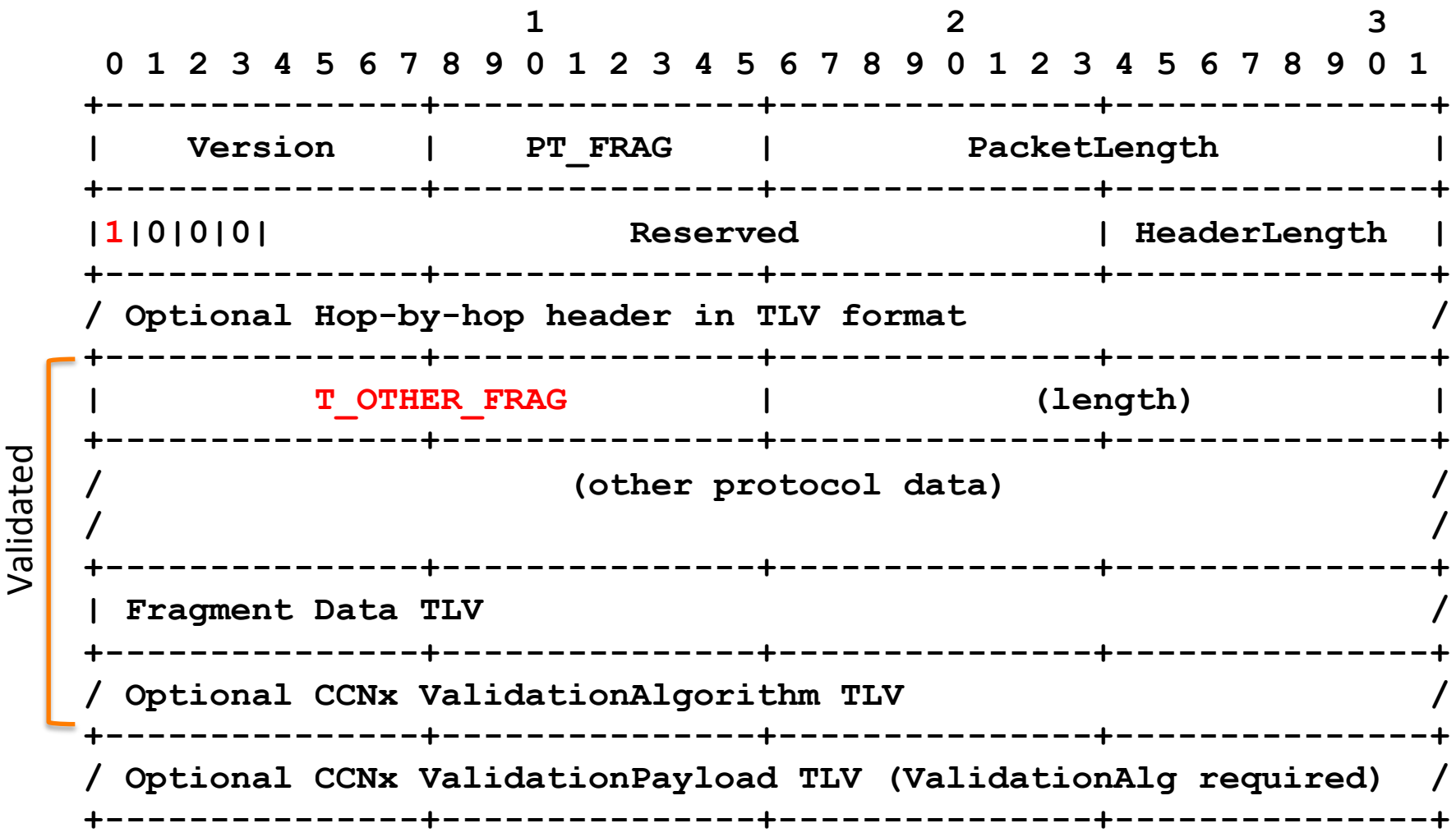



- X: Extended Format (X=0 shown above)
- FragSequenceNumber is 20 bits
- Validation could be HMAC or CRC or Signature

# CCNx Extended Encoding



# CCNx Multiple Protocol Support



# Total overhead per fragment

- For CCNx1.0 / 2+2 TLV
  - 12 bytes (basic format, no validation)
  - 20 bytes (extended format, no validation)
  - 28 bytes (basic format, CRC32c)
  - 64 bytes (basic with HMAC SHA256)
  - 72 bytes (extended with HMAC SHA 256)

# Frame Packing with Fragments

- A single frame may carry multiple fragments.
  - Example output queue to a peer
    - interest1(200B)
    - content1(3500B)
    - interest2(200B)
    - content2(500B)

    → *6 Ethernet frames ?!*
  - Possible packing (basic, no validation)
    - [frag(interest1, 200B), frag(content1, 1276B)]
    - [frag(content1, 1488B)]
    - [frag(content1, 236B), frag(interest2, 200B), frag(content2, 500B)]

    → *3 Ethernet frames*

# Other Extension

- Using Idle frames and Hop-By-Hop headers for link quality measurements.
- Allowing the receiver to support out-of-order packets.
- Link negotiation protocol could agree on starting fragment number and fragment sequence number length.

# Security Considerations

- The basic protocol has no protection for the fragmentation header
- The extended protocol could be strongly authenticated, but MACSEC might be a better option as it will be lower overhead and provide encryption.

# Conclusion

- Provide a fragmentation container.
- Specify one protocol, like PPP.
- Specify a basic and extended encoding for CCNx 1.0.
- Future: compare end-to-end with hop-by-hop, add encodings for NDNLP.