Secure Fragmentation for Content Centric Networking

Christopher A. Wood
Palo Alto Research Center
cwood@parc.com

Marc Mosko
Palo Alto Research Center
mmosko@parc.com

IEEE CCN 2015
Dallas, TX, USA
10/19/2015
Agenda

1. CCNx overview
2. Fragmentation and segmentation
3. CCNx fragmentation options
4. Named Network Fragments
5. Performance Results
6. Q&A
1. CCNx overview
CCNx 101

• Content is named and transferred through the network from producers to consumers upon request

• Consumers issue interest packets for content objects

• Forwarders (routers) move interests from consumers to producers

• Forwarder FIBs store forwarding information, Pending Interest Tables (PITs) store interest state, and Content Stores (caches) store previously requested content

• Producers satisfy interests and return the resulting content object
\[ C_{rA} \rightarrow R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow P \]

\[ R_4 \rightarrow C_{rB} \]

/youtube/videos/presidentspeech
$C_r_A \rightarrow R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow P$

$C_r_B \rightarrow R_4$

/youtube/videos/presidentspeech
2. Fragmentation and segmentation
Network Links

The Internet connects heterogeneous devices over heterogeneous links with different:

• Physical layers (copper, fiber, radio)

• Link layers (Ethernet, WiFi)

• Maximum Transmission Unit (MTU) sizes

  (determined by link layer)
Fragmentation: splitting a packet into fragments that fit into an outgoing link MTU

- Fragment header encodes information (e.g., ordering) of related fragments
- Re-fragmentation can occur if smaller MTU is encountered
Segmentation: cutting up large pieces of data at the transport (or higher) layer

- Segmentation is not fragmentation

- Both may occur for a given consumer-to-producer path
How do fragmentation and segmentation apply to CCNx?
CCNx Messages

Interest
- Header(s)
- Body
- Validation

ContentObject
- Header(s)
- Body
- Validation
CCNx Messages

Interest

<table>
<thead>
<tr>
<th>Header(s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Body (name + payload)</td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td></td>
</tr>
</tbody>
</table>

Names are unbounded

Payload contents are unbounded

ContentObject

<table>
<thead>
<tr>
<th>Header(s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Body (name + payload)</td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td></td>
</tr>
</tbody>
</table>

Names are unbounded

Payload contents are unbounded
CCNx Messages

Interest

<table>
<thead>
<tr>
<th>Header(s)</th>
<th>Names are unbounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body (name + payload)</td>
<td>Payload contents are unbounded</td>
</tr>
<tr>
<td>Validation</td>
<td></td>
</tr>
</tbody>
</table>

ContentObject

<table>
<thead>
<tr>
<th>Header(s)</th>
<th>Names are unbounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body (name + payload)</td>
<td>Payload contents are unbounded</td>
</tr>
<tr>
<td>Validation</td>
<td></td>
</tr>
</tbody>
</table>
CCNx Segmentation

• CCNx packet fixed header imposes constraints on the message size
  • Names and payloads sizes are bounded

• If the payload for a Content Object (or Interest) exceeds this bound, it must be segmented
  • akin to TCP segmentation
CCNx Segmentation Problems

- Is (Content Object) segmentation a substitute for fragmentation?
  - Maybe, if the minimum path MTU is known
  - No, otherwise (i.e., for Interests)
CCNx Segmentation Problems

• Is (Content Object) segmentation a substitute for fragmentation?
  • Maybe, if the minimum path MTU is known
  • No, otherwise (i.e., for Interests)

• Does MTU discovery work?
  • Not all the time!
CCNx Segmentation Problems

• Is (Content Object) segmentation a substitute for fragmentation?
  • Maybe, if the minimum path MTU is known
  • No, otherwise (i.e., for Interests)
• Does MTU discovery work?
  • Not all the time!
CCNx Segmentation Problems

• Is (Content Object) segmentation a substitute for fragmentation?
  • Maybe, if the minimum path MTU is known
  • No, otherwise (i.e., for Interests)

• Does MTU discovery work?
  • Not all the time!
CCNx Segmentation Problems

- Is (Content Object) segmentation a substitute for fragmentation?
  - Maybe, if the minimum path MTU is known
  - No, otherwise (i.e., for Interests)

- Does MTU discovery work?
  - Not all the time!

![Diagram showing MTU discovery example](image)
CCNx Segmentation Problems

• Is (Content Object) segmentation a substitute for fragmentation?
  • Maybe, if the minimum path MTU is known
  • No, otherwise (i.e., for Interests)

• Does MTU discovery work?
  • Not all the time!

\[ \text{Link MTU} = 1000 \quad R_1 \quad \text{Link MTU} = 2000 \quad P \quad \text{Len} = 2000 \]
CCNx Segmentation Problems

• Is (Content Object) segmentation a substitute for fragmentation?
  • Maybe, if the minimum path MTU is known
  • No, otherwise (i.e., for Interests)

• Does MTU discovery work?
  • Not all the time!
CCNx Segmentation Problems

• Is (Content Object) segmentation a substitute for fragmentation?
  • Maybe, if the minimum path MTU is known
  • No, otherwise (i.e., for Interests)

• Does MTU discovery work?
  • Not all the time!
CCNx Segmentation Problems

• Problem #1: routers do not have access to signing keys
CCNx Segmentation Problems

- Problem #1: routers do not have access to signing keys
- Problem #2: producer cannot segment for all MTUs
CCNx Segmentation Problems

• Problem #1: routers do not have access to signing keys
• Problem #2: producer cannot segment for all MTUs
• Problem #3: Interests cannot be segmented since the MTU is not known (among other reasons)

fragmentation is unavoidable
3. CCNx fragmentation options
CCNx Fragmentation Options

There are two flavors of CCNx fragmentation proposals:

- Hop-by-hop (Begin-End Fragmentation) [1]
- Cut-through (FIGOA) [2]

Begin-End Fragmentation

- Run between a sender and “peer”
- B, E, and BE flags are used to signal the start, end, and entry of a fragment series
- Fragments are tagged with monotonically increasing sequence numbers
- Idle fragments can be used to advance the fragment number
Begin-End Fragmentation Protocol

• Senders:
  • Break up a message into fragments with increasing numbers
  • Mark fragments with B and E bits as needed

• Receivers (peers):
  • Maintain one reassembly queue per sender
  • Gather while in-order fragments are received
  • Reassemble and pass up when end fragment is received
  • Discard the queue when an out-of-order fragment is received
FIGOA Fragmentation

- Based on the concept of *delayed authentication*

- Fragment packets based on MTU size and tag with byte offsets (not indexes)
  - Fragment data size is a multiple of the hash function digest size
  - Append the IV or intermediate state of the Merkle-Damgard hash function computation to each fragment (next slide)

- Allows fragments to be re-fragmented if needed
Merkle-Damgard Hash Functions

\[ m_1, m_2, m_3, m_4, \ldots, m_k, \text{pad} \]

\[ H^0 \rightarrow f \rightarrow H^1 \rightarrow f \rightarrow H^2 \rightarrow f \rightarrow H^3 \rightarrow f \rightarrow H^4 \rightarrow f \rightarrow H \]

Hash Value
Merkle-Damgard Hash Functions

$$\begin{align*}
m_1 & \quad m_2 & \quad m_3 & \quad m_4 & \quad \cdots & \quad m_k & \quad \text{pad} \\
H^0 & \quad \downarrow f & \quad \downarrow f & \quad \downarrow f & \quad \downarrow f & \quad \downarrow f & \quad \downarrow f \\
H^1 & \quad H^2 & \quad H^3 & \quad H^4 & \quad H \\
\text{Hash Value} &
\end{align*}$$
FIGOA Fragments

Signature (on H)

Name

Data

\( F_1 \) Fragment-Info

\[
\begin{align*}
\text{IntState} &= H^0 \\
\text{... Data ...}
\end{align*}
\]

\( F_2 \) Fragment-Info

\[
\begin{align*}
\text{IntState} &= H^1 \\
\text{... Data ...}
\end{align*}
\]

\( F_3 \) Fragment-Info

\[
\begin{align*}
\text{IntState} &= H^2 \\
\text{... Data ...}
\end{align*}
\]

ContentObjectSize

FragmentOffset

FragmentSize

SignatureInfo

Signature
FIGOA Fragmentation Protocol (Sender)

- Fragment message into blocks that are multiples of the hash function digest
- Tag with the byte offset and include the hash function IV or intermediate state (IS)
- Send the fragments...
FIGOA Fragmentation Protocol (Receiver)

- Maintain one reassembly queue per message

- Upon receipt of a fragment without the previous fragment, compute hash and store in queue. If the successor is present, compare the output hash against the successor’s IS

- Upon receipt of a fragment with the previous fragment, check against the computed IS, and do the step above.

- Forward all fragments except the last right away.

- Once the full message is received, verify the output digest (if given) and signature, and forward the fragment.
4. Named Network Fragments
FIGOA Shortcomings

• Not possible to match a fragment to a hash-based interest (format does not specify digest in the fragment response)

• Signature verification is deferred to the end “hostage” fragment
Named Network Fragments

NNF improvements over FIGOA:

• Unbounded content length
Named Network Fragments

NNF improvements over FIGOA:

• Unbounded content length

• Immediate signature verification
Named Network Fragments

NNF improvements over FIGOA:

• Immediate signature verification
• Unbounded content length
• Selective retransmission for dropped fragments
Named Network Fragments

NNF improvements over FIGOA:

- Immediate signature verification
- Unbounded content length
- Selective retransmission for dropped fragments
- Hash-based named fragment “chains”
Named Network Fragments

NNF improvements over FIGOA:

• Immediate signature verification
• Unbounded content length
• Selective retransmission for dropped fragments
• Hash-based named fragment “chains”
• Complete ContentObject replacement
NNF Packet Format

Fragment := FixedHeader *OptionalHeader NamedFragment
Payload [ValidationAlg ValidationPayload]

FixedHeader := <as per CCNx 1.0 spec>
OptionalHeader := <as per CCNx 1.0 spec>
NamedFragment := <see right>
Payload := <blocks of original content>
ValidationAlg := <as per CCNx 1.0 spec>
ValidationPayload := <as per CCNx 1.0 spec>

NamedFragment := (FragmentStart | FragmentData | SegmentStart | SegmentData | SegmentEnd) ChainData
FragmentStart := Name [DigestAlg] OverallLen OverallDigest
FragmentData := [Name] OverallDigest
SegmentStart := Name [DigestAlg] OverallLen SegmentID
SegmentData := [Name] SegmentID
SegmentEnd := [Name] SegmentID OverallDigest
ChainData := PayloadOffset InterState
Name := <as per CCNx 1.0 spec>
OverallLen := Integer
SegmentID := 1*OCTET
OverallDigest := 1*OCTET
DigestAlg := SHA256 / <others>
PayloadOffset := Integer
InterState := 1*OCTET
NNF Packet Format

Fragment := FixedHeader *OptionalHeader NamedFragment
  Payload [ValidationAlg ValidationPayload]

FixedHeader := <as per CCNx 1.0 spec>
OptionalHeader := <as per CCNx 1.0 spec>
NamedFragment := <see below>
Payload := <blocks of original content>
ValidationAlg := <as per CCNx 1.0 spec>
ValidationPayload := <as per CCNx 1.0 spec>

NamedFragment := (FragmentStart | FragmentData |
  SegmentStart | SegmentData | SegmentEnd)
  ChainData
FragmentStart := Name [DigestAlg] OverallLen
  OverallDigest
FragmentData := [Name] OverallDigest
SegmentStart := Name [DigestAlg] OverallLen SegmentID
SegmentData := [Name] SegmentID
SegmentEnd := [Name] SegmentID OverallDigest
ChainData := PayloadOffset InterState
Name := <as per CCNx 1.0 spec>
OverallLen := Integer
SegmentID := 1*OCTET
OverallDigest := 1*OCTET
DigestAlg := SHA256 / <others>
PayloadOffset := Integer
InterState := 1*OCTET
NNF Selective Retransmission

- Link recovery protocols can be used to retransmit dropped or corrupted packets (fragments)
- Selective retransmit is used if (groups of) fragments need to be requested over more than a single hop (link)
- Fragments are uniquely defined by
  \{Name, OverallDigest, PayloadOffset, IntermediateState\}
- Nodes (routers or consumers) can retransmit
NNF Selective Retransmission

Original Fragmented ContentObject

<table>
<thead>
<tr>
<th>0</th>
<th>1024</th>
<th>2048</th>
<th>3072</th>
<th>1024*(n-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F2</td>
<td>F3</td>
<td>...</td>
<td>Fn</td>
</tr>
</tbody>
</table>

Chain 1

PO=1024, PS=1024

| F2 |

Chain 2

PO=1024, PS=2048

| F2 | F3 |

Chain 3

PO=1024, PS=1024*(n-2)

| F2 | ... | Fn |

Chain 4

PO=1024, PS=1536

| F2 | F3 | F3 |
NNF Fragmentation Logic

Similar to FIGOA fragmentation logic, except:

• Only the OverallDigest must be verified upon the arrival of the last fragment

• If valid, the PIT entry is cleared
NNF PIT Logic

• PIT entries are partially satisfied as fragments arrive
  • PIT lookup needs to take this into account
• See the paper *or upcoming spec* for specific details
5. Performance Analysis
Experimental Analysis

• Two experiments with a 6-hop topology:
  • 0% loss and 1% loss
  • Transfers of a 10MB file chunked (segmented) into 1280, 2560, 3840, 7680, 16640, and 33280 bytes
    • Fragment size must be a multiple of 64 and the implementation chunk size
  • Clients transfer chunks serially and measure the latency
Experimental Analysis

6 hops 0% loss rate

6 hops 1% loss rate
• Discussed existing CCNx fragmentation proposals
• Described the new NNF fragmentation protocol
• Displayed some preliminary experimentation results
What’s Next?

Write the specification for increased clarity
Questions?...