Secure Fragmentation for Content Centric Networking

Marc Mosko
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304
e-mail: mmosko@parc.com

Christopher A. Wood
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304
e-mail: cwood@parc.com

***Abstract*—Information Centric Networks (ICN), such as Content Centric Networks (CCNx) or Named Data Networks (NDN) disseminate data using hierarchal names for each chunk of data, where a chunk is roughly the size of an IP datagram. To secure the named exchange, each chunk has a digital signature or a hash-based name. Because these datagrams may be much larger than a link MTU, there is a need for fragmentation, and to date several hop-by-hop, end-to-end, and mid-to-end fragmentation schemes have been proposed. Only one scheme, FIGOA, a cut-through mid-to-end fragmentation scheme (i.e. without reassembly) claims to be secure by using delayed authentication. We propose a secure cut-through fragmentation scheme derived from FIGOA that allows immediate validation. We also consider queuing and timing issues that were not discussed in FIGOA.**

***Index Terms*—CCN, fragmentation, delayed authentication, selective retransmission**

I. Introduction

Information-Centric Networking (ICN) is a family of emerging network architectures for transferring named data between hosts; Content-Centric Networking (CCNx) and Named Data Networking (NDN) are two closely related ICN derivatives driven by the desire to solve many problems with today's Internet architecture. Unlike traditional IP-based networks that use host addresses, CCNx treats content as first class entities with unique names that can be routed through the network. Consumers issue requests (interests) for content with a particular name, and this request is routed to the producer or a network entity (i.e., cache) that is capable of satisfying the request. The corresponding content carrying a matching name is then sent to the consumer along the reverse path. This communication model enables content to be decoupled from its place of origin, i.e., the producer, thereby enabling content to be cached within the network to optimize bandwidth use, reduce latency, and enable effective utilization of multiple network interfaces simultaneously.

Since the designs of both CCNx and NDN separate content from its place of origin, consumers and other network entities, e.g., routers and forwarders, need a way to verify the *authenticity* of content before it is used in a meaningful way. Content objects therefore carry a digital signature that is used to perform this verification check. If a signature is not provided, the authenticity can be verified by checking the cryptographic hash digest of the content object against an expected value. Regardless of the mechanism by which authenticity is ascertained, it may always be the case that the size of a particular content object, will exceed a link MTU. This means that content objects must be fragmented[1].

To date, several hop-by-hop, end-to-end, and cut-through fragmentation schemes have been proposed. However, in this work, we only consider hop-by-hop and cut-through fragmentation. End-to-end fragmentation is unrealistic without proper restrictions on the maximum length of an interest or content object name (see [1] for an informal proof of this claim). Out of all relevant fragmentation protocols, only one – FIGOA [1] – claims to be secure by virtue of its usage of delayed authentication. However, it has several drawbacks. Signature verification is delayed until the last fragment is received. This is because the signature is dependent on the hash computed from the entire message. FIGOA does not enable hash-based content retrieval without explicitly include the name of the message (content object) in every fragment. Due to unbounded name lengths, this is not a viable solution.

Inspired by FIGOA, we present a new cut-through fragmentation protocol for CCNx called Named Network Fragments (NNF). The NNF protocol provides equivalent security to FIGOA but overcomes the aforementioned performance and design issues. Furthermore, NNF provides the ability for consumers or routers to *selectively* request and retransmit fragments of a content object chunk. In addition, the NNF packet format, described in Section IV, could be used to supplement or replace the existing CCNx content object format. In this way, content objects become fragments and fragments are content objects. This uniformity should greatly simplify future protocols and network (router) behavior.

The rest of this paper is organized as follows. Section II presents related work in ICN architectures, including those discussed by the CCNx and NDN communities. Sections III and IV describe the NNF protocol and packet formats. Examples of how NNF would fragment content objects are presented in Section V, and the security of the protocol is analyzed in Section VI. Finally, we conclude with an analytical model and preliminary results on the NNF implementation in Sections VII and VIII.

[1]Interest messages may also need to be fragmented. See Section II for more details.

II. Related Work

Fragmentation is a necessary part of any ICN architecture and protocol. The current NDNLP link layer protocol in NDN supports hop-by-hop fragmentation in NDN. Fragments only carry sequence numbers that allow for, and mandate, intermediate reassembly if signature verification is required [2], [3]. It does not use cut-through forwarding due to its implementation complexity and the barriers incurred by carrying the parent packet name in each fragment [3].

Ghali et al. [1] proposed the cut-through fragmentation protocol FIGOA to alleviate the performance impediments caused by intermediate reassembly. The details of this scheme are outlined in the following section. For now, we simply note that its use of delayed authentication effectively circumvents the need for intermediate reassembly prior to fragmentation. FIGOA was only implemented in the ndnSIM simulation software for a performance assessment, and, to date, has not been adopted by any of the major ICN designs.

Standards for fragmentation in CCNx have recently been proposed in [4] and [5]. PARC has proposals and working implementations for both end-to-end and hop-by-hop fragmentation. The latter was originally implemented in the CCNx 0.8x code [6]. This protocol was updated to include support for hop-by-hop fragmentation. Furthermore, multilink PPP [7] was adopted to ICN so as to support multiple different fragmentation protocols with the same encoding. The PARC end-to-end fragmentation protocol presented in [4] uses MTU discovery in interest messages so that the producer (end-host) can fragment the content object response to the appropriate size. With symmetric forwarding routes, this prevents any intermediate nodes performing re-fragmentation. Security issues beyond link (authenticated) encryption are not discussed in either proposal.

III. Named Network Fragments

We now describe how NNF follows from FIGOA. For reading ease, and where appropriate, we use notation drawn from [1] where CO^n is a raw (unsigned) content of total size n bits and $\overline{CO}^n$ is the signed version of CO^n. The secure fragmentation scheme FIGOA [1] operates by sending fragments in multiples of the hash block size in each packet, along with a byte offset and the intermediate state (IS) of the hash function up to that point. For example, the first fragment carries a byte offset of 0, up to kb blocks of data (where b is the hash block size) such that it fits within the packet's MTU, and the hash's initialization vector (IV). The second fragment carries a byte offset kb, the next kb blocks, and the intermediate state of the hash computation from the first block. The final fragment may carry a short payload under b bytes long and the hash function should operate as natural for the given hash function (e.g., pad with zeros). This creates an implicit hash chain, because a node can securely reassemble the pieces based on the calculated IS and the header of a fragment. FIGOA allows intermediate re-fragmentation because an intermediate node can reduce the payload from kb to some $(k-j)b$ blocks and then create a subsequent fragments of jb blocks with the appropriate starting byte offset and intermediate state; this reduction in fragment size still maintains the hash chain.

While the FIGOA design is an interesting solution when content is requested by name only or by name and KeyId, it does not help when content is requested by a hash name. This is because the hash name is over the entire signed message $\overline{CO}^n$. It is not possible to match a fragment to a hash based interest. The FIGOA *implementation* solves this problem by including the `ContentDigest` parameter as a field in the `ContentFragment` packet. The `ContentDigest` is supposed to match what would be the computed hash of $\overline{CO}^n$ to allow forwarding on hash based names without actually computing the entire hash. This enables a router to match fragments of a larger packet against the hash name provided in an interest. However, this allows the injection of malicious packets because it defers the verification to the end of the hash chain.

Another problem with FIGOA is that signature verification (if performed) is delayed until the final fragment – the hostage fragment – is received. This is because the hash computation based on the final chain, which should equal the `ContentObjectHash` field provided in an interest, is the input to the signature verification procedure. Also, the `ValidationInformation` field of the fragmented content object, which is located at the end of the message, contains the KeyLocator necessary to identify the signature verification key. Since the final fragment is not forwarded until the signature is verified, the overall throughput is necessarily extended by the cost of the signature verification procedure. A better solution would deliver the signed fragment first to amortize the signature verification overhead *while fragments are being delivered*.

To this end, we present the secure Network Named Fragments (NNF) fragmentation protocol. NNF is designed to overcome the FIGOA shortcomings using more efficient signature verification and selective retransmit of individual fragments. We outline the NNF design in the following sections. NNF provides the following benefits over FIGOA:

- **Immediate signature verification** – The first fragment carries a signature, so the overall security context may be established immediately without needing to reassemble all fragments. Observe that if the signature can be verified upfront, the application can begin verification while the reset of the content is received, amortizing signature verification time while providing partial use of fragments to applications, e.g., to pipe data through a decoder or write it to a file.
- **Unbounded content length** – The overall length of the fragmented content is not limited to a specific length, so large payloads could be conveyed. NNF allows for very long content with a known digest or for segments of live stream content, where the digest is not known until the end of the segment.
- **Selective retransmission** – Each fragment of an NNF fragment stream is uniquely identified by the state

```
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>
```

Fig. 1. ABNF Fragment specification.

{`Name`, `OverallDigest`, `PayloadOffset`, `IntermediateState` }, and those items can be encoded as part of the Name of the Fragment, so each fragment could be selectively requested.

- **ContentObject Replacement** – FIGOA encapsulates an existing ContentObject (or "Data"), NNF could be used as a ContentObject replacement. It could encapsulate a large ContentObject or it could encapsulate raw payload without the inner encapsulation.
- **Hash named fragment chains** – The head-of-chain fragment can be explicitly named in an Interest, which enables selective repeat, and it can also be explicitly named with a hash-based name for secure hop-by-hop Interest processing even with fragmentation.

IV. NNF Packet Format

The fragment encapsulation mechanism given in [5] enables any fragmentation mechanism to be implemented and encoded on the wire. NNF uses this mechanism to encode individual fragment packets. A fragment is a CCNx message with a fixed header packet type of PT_FRAG and uses the extended format. The MessageType is T_NNF. The fragment header (which is part of the T_NNF TLV) contains several fields to aid in the reconstruction and verification of the fragment stream.

Fig. 1 specifies the ABNF of a fragment in the CCNx 1.0 format. In the FixedHeader, the PacketType is set to PT_FRAG and the PacketLength is the length of this immediate fragment, which is limited to the MTU size (maximum 64 Kb). the FragmentData is part of "signed information" of the packet, and would be covered by an optional ValidationAlg and ValidationPayload, if desired. For example, the first fragment could have a complete signature that is *immediately verifiable*, which creates the root of a trusted hash chain for the remainder of the fragments.

Fig. 2 shows the details of the `NamedFragment` token. There are five types of NamedFragments.The first two types, `FragmentStart` and `FragmentData`, are used when the `OverallDigest` is known in advance, such as when fragmenting a known file. The last three – `SegmentStart`, `SegmentData`, `SegmentEnd` – are for use with segmented data streams, such as live streams, where an unterminated data stream is transmitted in segments of a known length but with a deferred digest computation.

Fragmented data begins with a `FragmentStart` message that indicates the CCNx `Name`, `OverallLength`, and `OverallDigest`. It is usually in a CCNx 1.0 packet with a

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

Fig. 2. ABNF NamedFragment specification.

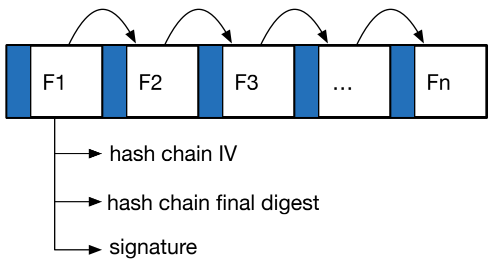

Fig. 3. A trusted NNF hash chain rooted at a signed fragment – $F1$.

signature, so there is an immediate trust environment for the subsequent hash chain. If the FragmentStart is signed, then it cannot be re-fragmented in network without breaking the signature, so often this packet has no Payload or only a small number of payload blocks such that it fits in minimum MTU size. Subsequent fragments are carried in `FragmentData` packets that do not need to be signed so could have a maximum sized `Payload`.

Segmented content fragments are for delay sensitive data or live streams where one wishes to begin fragmented delivery before the `OverallDigest` is known. The original fragment creator generates a `SegmentID`, such as a random number, for the current segment and publishes fragments until the segment ends. In the last segment, the publisher ties together the `SegmentID` and `OverallDigest` in a signed object. This could be a tail object and not carry any payload and be delivered after the processing delay of calculating the `OverallDigest`. Because the tail object is signed, it should be as small as possible because it cannot be re-fragmented.

A high-level depiction of this hash chain is shown in figure 3.

These fields are defined as:

- `Name`: The CCNx name of the payload. The Name is optional in all fragments if supporting CCNx Nameless Objects, otherwise it must appear in at least the first fragment.
- `OverallLength`: The total length of all the fragmented payload.When used with segmented content, it is the

length of the current segment, not the total stream length.

- `OverallDigest`: The digest of all the fragmented payload.
- `SegmentID`: A SegmentID is used when the OverallDigest is not known when the fragmentation begins. It is used to defer the calculation of the OverallDigest until the end of the segment.
- `DigestAlg`: The digest algorithms for OverallDigest, assumed SHA-256 if not present. Taken from a specified enumerated set.
- `PayloadOffset`: The byte offset where this fragment begins.
- `IntermediateState`: The IS value unto this payload.

An intermediate node may only re-fragment if it understands `DigestAlg`. Thus, we specify `DigestAlg` from an enumerated set such that implementations can comply to a specific standard.

A. Unbounded Length Fragments

The `OverallLength` field is not limited in size. In a normal ContentObject, the Payload TLV length is limited to 64 KiB, but in a fragmented content object, it is essentially unbounded. One could transfer a large file, for example, as one stream. In normal use, however, we expect that traditional CCNx chunking will split content in to nominally sized chunks and then fragment each chunk. The chunks, using NNF, may now be any size not limited by the 64 KB CCNx 1.0 packet length.

B. Selective Retransmission

Any standard Link recovery protocol between pairs of routers can be used to retransmit individual fragments if they are dropped or arrive corrupted. However, if, for example, a consumer wishes to identify a subset of fragments to retrieve for a single content object, perhaps because they were lost locally, the consumer should be able to ask for individual fragments instead of the entire content object again. To accomplish this, each fragment must be uniquely addressable.

In NNF, each fragment is uniquely identified by the name {`Name`, `OverallDigest`, `PayloadOffset`, `IntermediateState`}, where Name is allowed to be empty. In one encoding, the *OverallDigest*, *PayloadOffset*, and *IntermediateState* are represented as part of the Name, for example: /parc.com/movie/alto.mkv/OD=123abc/PO=4096/IS=653efa. In effect, this names each fragment and enables selective retransmission of individual fragments. The first fragment still carries only the original name /parc.com/movie/alto.mkv, because a requester would not necessarily know the *OverallDigest*.

It is also possible for chains of fragments to be selective requested. This is done by providing the name of the fragment and an additional payload size. For example, /parc.com/movie/alto.mkv/OD=123abc/PO=4096/IS=653efa/PS=8192. If fragments are individually 1024B, then this would return a chain of four fragments starting at byte offset 4096. If instead the fragment name was issued as /parc.com/movie/alto.mkv/OD=123abc/PO=4096/IS=653efa/PS=7680, the last (fourth) fragment in the returned chain would need to be re-fragmented to 512B. An example of this re-fragmentation is shown in Figure 4. One important caveat for this re-fragmentation is that the first fragment must remain intact so that interest matching can be done correctly (see the next section).

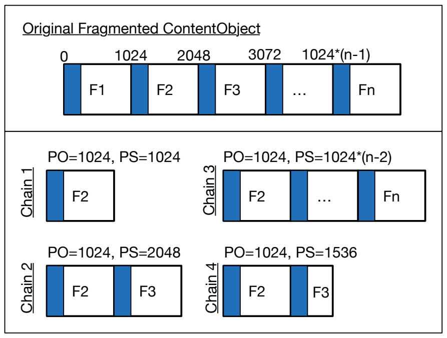

Fig. 4. NNF selective retransmit with refragmentation.

C. Intermediate Node Processing

An intermediate node requires the first fragment to match a PIT entry and begin cut-through forwarding. All intermediate nodes will match the first fragment based on its `Name`, `KeyId`, and `ContentObjectHash` as is normal for a CCNx 1.0 forwarder. The `ContentObjectHash` is computed only over the first fragment. Once the first fragment is verified, the node may begin forwarding other fragments along the same PIT entry. This is different from FIGOA, where the `ContentObjectHash` in an interest matches the `OverallDigest` value, which can only be verified once all fragments have been processed. Note that this requirement for NNF means that the first fragment cannot be re-fragmented by intermediate nodes in the network.

An Intermediate node will maintain a PIT entry reverse path until it has forwarded `OverallLength` bytes via the PIT entry. Because selective retransmissions happen in a different namespace than the first fragment name, they will not count against the PIT entry. Also, an intermediate node may choose to validate the fragment stream via signature verification (as explained above), similar to the method used in FIGOA.

To support selective retransmission, router and forwarder content stores must be able to parse and interpret individual fragment names, and re-fragment if necessary. Algorithms 1 and 2 show the actual fragment and PIT logic necessary to support selective retransmission as described above.

Alg. 1 generally follows the processing steps in FIGOA. It is not necessary to cache any fragments – the reassembly buffer **Buffer** only stores a chain of entries with { `CurrentIS`, `NextIS`, `PayloadOffset`, `PayloadEnd` }. The entries are stored in order of `PayloadOffset`. To begin, we create an initial buffer entry with the SHA-256 initialization value and `PayloadOffset` of 0 and `PayloadEnd` of 0. From there, all received fragments will form a chain. The next four rules are as per FIGOA: the first rule stores a singleton without predecessor or successor, the second rule stores a fragment that continues a previous fragment, the third rule stores a fragment that precedes a fragment, and the fourth rule stores a fragment in between two existing fragments. Any time we store a fragment and it verifies, we update the "in-order" verified length so we know how many bytes have verified in the hash chain. Once we have verified `OverallLength` we have received all fragments and should verify the `OverallDigest`. If the current fragment was the one that verified the `OverallDigest`, we locally mark this packet as the "last fragment" to indicate that the PIT may clear its state.

Algorithm 1 Fragment Logic

```
 1: Input: Content fragment with {Name, OverallDigest, PayloadOffset, IntermediateState (IS)}
 2: if Buffer[OverallDigest] does not exist then
 3:     Create with nextInterState = SHA256 Initialization value.
 4: end if
 5: Compute $IS' = h(IS, payload)$.
 6: if Previous fragment with (PayloadEnd + 1) = current PayloadOffset nor next fragment (with PayloadOffset = PayloadEnd + 1) is in Buffer. then
 7:     Store the packet.
 8: else if Previous fragment is in Buffer, but next fragment is not then
 9:     Verify that previous fragment $IS' = IS$, otherwise drop.
10: else if Next fragment is in Buffer, but the previous is not then
11:     Verify that IS' = next fragment's IS, otherwise drop.
12: else if Both previous and next are in Buffer then
13:     Verify that the hash chain is continuous, otherwise drop.
14: end if
15: Update the in-order verified length
16: if In-order verified length equals the OverallLength then
17:     Verify the OverallDigest. If verified, mark as "last fragment" so the PIT will clear state. Otherwise, drop.
18: end if
```

Alg. 2 shows the PIT table processing logic. If fragments are received in order, then the first fragment will have a name that matches an existing PIT entry. We then create a new PIT entry by `OverallDigest` and remove the PIT entry by name. Subsequent fragments output by Alg. 1 will match on `OverallDigest`. However, fragments may not necessarily arrive in order, so this leads to several more conditions in the algorithm. The block at Line 4 tests if a PIT entry by `OverallDigest` exists, and if so forwards along that PIT entry. The block at Line 8 checks if the fragment has fragment state in the name, in which case it may be a retransmission and should be matched against its own PIT entry by name. The block at Line 13 will try to create the PIT entry by `OverallDigest` if the previous lookup did not succeed. The block at Line 21 is the same as at Line 13, except it handles the case of a named fragment arriving before a chunk named fragment. A named fragment has the fragment state in the name. In this case, we strip the fragment state from the name and proceed as for a chunk named fragment.

Algorithm 2 PIT Logic

```
 1: Input: Content fragment with {Name, OverallDigest, PayloadOffset, IntermediateState (IS)}
 2: CaseA = False
 3: CaseC = False
 4: if OverallDigest entry is in the PIT then
 5:     Forward and remove the PIT entry after the last fragment (it is marked by the Fragment Logic – see Algorithm 1)
 6:     CaseA = True
 7: end if
 8: if The fragment has a Name with OverallDigest, PayloadOffset, IS state in it then
 9:     if A PIT entry with that name exists then
10:         Treat as retransmission request and forward
11:     end if
12: end if
13: if CaseA = False and the fragment has a Name then
14:     if A PIT entry with that name exists then
15:         Create a PIT entry by OverallDigest with same reverse path.
16:     end if
17:     Forward on that PIT entry as above
18:     Remove the PIT entry by name
19:     CaseC = True
20: end if
21: if CaseC = False and the name has {OverallDigest, PayloadOffset, IS} state in it then
22:     if A PIT entry with name but not the state exists then
23:         Proceed as in block line 13.
24:     end if
25: end if
26: Otherwise, drop the fragment.
```

V. Examples

A. FragmentData Example

A file is to be published as FragmentData. It has a `OverallLength` of 10000 and a `OverallDigest` of `0x0011223...` (32-byte SHA256). The IntermediateState (IS) is set to the 32-byte initialization vector for SHA-256.

The first object has the `Name` lci:/parc.com/pubs/spec1.pdf/OD=0x0011223.../PO=0/IS=0x6a09e66... with a `OverallLength` of 10,000 and a `Payload` of the first 1000 bytes of the file. The encoded length of the name is 114 bytes due to the two 32-byte values plus other overhead. Reserving room for an RSA signatures and other fields means the first packet will be approximately 420 bytes of overhead, so it could carry 13 blocks of 64-bytes (832 bytes), for a total packet length of 1252 bytes.

The subsequent blocks may carry similar names, with the IS updated per block for the hash chaining and the `PacketOffset` for the byte location.Because these packets are not signed, we only have 130 bytes of overhead and can send maximum acceptable fragments (say 1500 bytes) because they could be re-fragmented in flight. This would allow 21 blocks of 64 bytes (1344 bytes) per fragment.We would need 7 such packets to finish the 10,000 bytes of payload.

Having a name in each FragmentData packet means that a receiver could selectively request any fragment. One could save 33 bytes per packet only including the necessary `OverallDigest` and `PacketOffset` and `InternState`. As this file is only 8 packets, selective retransmit might not be that important, so one could possibly save one packet by not using a full name.

B. SegmentData Example

A live stream will send a 1Mbps feed that will digest once per second. The `OverallLength` will be 125,000 bytes.As in the previous example, the name will be, for example, 120 bytes.The name will carry a SegmentID (for example a 32-byte random number) instead of an `OverallDigest`.

Assuming the first and last objects are signed, then each segment will take 94 packets (1 start, 92 middle, 1 end). This assumes we have included a full name in each fragment so one could selectively retransmit any one fragment. We could get higher efficiency if, for example, we only included a full name every 10 fragments.

VI. Security Analysis

NNF provides the same security guarantees as FIGOA by virtue of how individual fragments are authenticated. Recall that NNF draws upon the FIGOA delayed authentication mechanism by individually fragments with a single signature verification and corresponding hash chain. In other words, the first fragment in a NNF-fragmented packet contains a signature generated from the overall digest of the sequence of fragments, as well as this overall digest. Procedurally, the verification procedure for such a NNF-fragmented packet is as follows:

1) The signature of the overall digest, provided in the first fragment, is valid.
2) The hash chain of fragments belonging to the same packet is valid. That is, the intermediate hash digest provided in each fragment matches the intermediate state of the hash digest computed up to that fragment.
3) The hash digest of the final fragment is equal to the overall digest provided in the first fragment.

The algorithm for performing step (2) of the verification procedure is provided in [1]. Steps (1) and (3) of the verification procedure are unique to NNF and their implementation is provided in Section IV.

Security, with respect to authenticity of the entire fragmented packet, is thus guaranteed by the following simple facts: (1) the packet signature and overall digest needed for verification are provided in the first fragment, and (2) the overall digest is computed and checked for equality before accepting the packet as valid. Thus, if the signature on the overall digest is valid and the computed overall digest equals the `OverallDigest` in the first fragment, the overall packet is deemed valid. The security of NNF therefore reduces to the security of the underlying hash function and signature scheme, just as with FIGOA. We refer the reader to [1] for a formal proof of security.

Security for segment fragments is guaranteed by a similar argument. The subtle difference is that the overall digest is provided in the final SegmentEnd fragment. The verifier merely ensures that the overall digest computed on the (stream) segment (for a particular segment ID) matches the overall digest in the SegmentEnd fragment, and that the signature provided with this last fragment is valid using this overall digest as its input. Verifying stream segments more closely follows the delayed authentication procedure of FIGOA.

VII. Analytical Model

In this section we present a simple model to capture the expected number of fragment retransmissions sent using NNF, FIGOA, and hop-by-hop fragmentation. In our model we consider the time it takes for a fragmented packet to be transmitted from a producer to a consumer over a path topology of n links. That is, interests from a consumer Cr will traverse a path of n links to producer P, and the content object responses will traverse back to Cr along the backwards path. Assume that each link L_i on this path has a fixed probability of loss p_i. The probability of failure p_f on *any* link for any fragment of a content object is $p_f = 1 - \prod_{i=1}^{n}(1-p_i)$.

Now, consider the transmission of a content object C that is composed into k fragments, and assume that t_f is the time to transmit a single one of these fragments along the entire path of n links, and that t_l is the time to transmit a fragment over a single link. Thus, $t_f = nt_l$. Without any failure, and assuming that each fragment for NNF, FIGOA, and the hop-by-hop fragmentation schemes take the same amount of time[2], the time t to transmit C is $t = t_f + (k-1)t_l = (n+k-1)t_l$.

The time it takes to transfer a single fragment of a content object t_f changes if links are lossy. Specifically, whenever a single failure occurs, the entire content object (and all of its fragments) must be retransmitted. For a content object of k fragments, the time to transmit C, t_{FIGOA}, is

$$t_{FIGOA} = (n+k-1)t_l + kp_f\frac{(n+k-1)t_l}{1-p_f}.$$

The first term captures the time it takes all fragments to be transmitted without any packet loss. In the event that at least one single loss occurs for *any* of the k fragments, the entire packet must be retransmitted again with success probability $(1-pf)$. According to the FIGOA protocol, if packet losses occur at any link, then the router will not be able to authenticate the entire content object. Consequently, the consumer will not receive the entire packet and must retransmit the interest. For simplicity, we do not take this interest retransmission time into account. Note that the time for hop-by-hop re-fragmentation would be at least as long as FIGOA, since the transmission time also incurs the cost of intermediate node processing.

Conversely, with NNF, where selective retransmission is supported, this time t_{NNF} is

$$t_{NNF} \le \frac{(n+k-1)t_l}{1-p_f}.$$

This comes from the fact that NNF can be modeled as the standard selective repeat ARQ protocol [8], [9]. Also, note

[2]This is merely for analytical simplification. In reality, fragment sizes and therefore the number of fragments for a given content object may be different.

that this is an upper bound because it assumes that individual retransmission occur from packet losses in the link adjacent to the consumer. Note that routers may potentially selective retransmit interests for individual fragments to avoid this worst-case delay. However, this is not mandated.

VIII. Experimental Evaluation

In this section we compare the performance of NNF against hop-by-hop fragmentation. We implemented the NNF protocol in the CCNx 1.0 transport stack and Metis forwarder. The evaluation was done on three Dell dual CPU 16-core Intel Xeon E5-2660 systems connected via 10G links. We created a 6-hop network by routing between 7 Metis instances over trunked vlans. We used raw Ethernet encapsulation on the links. We ran the experiments 3 times and averaged the results.

The first set of experiments shown in Figure 5 have a 0% loss rate and the second set of experiments in Figure 6 have an overall 1% loss rate (0.167% per link). Each experiment transfers a 10M file chunked in to 1280, 2560, 3840, 7680, 16640, or 33280 bytes. We use these values because the fragment size (1280) must be a multiple of 64 and the chunk size a multiple of the fragment size in our implementation. The server pre-generates all chunks and fragments, so the measured time does not include any significant computation on the server. Each NNF `OverallDigest` is for a single chunk (not segment), so they are independent chunks. A single client transfers each chunk serially and measures the latency. We compute the average "chunk time" by diving the total latency by the number of chunks. The client uses a fixed 2ms retransmission timeout.

The results of this experiment are shown in Figure 5. The results from a similar experiment with lossy experiments are shown in Figure 6. Our results indicate that NNF outperforms hop-by-hop fragmentation with increasing likelihood as the chunk size of content increases. In other words, NNF provides better latency performance as the number of fragments for a content object increases. Hop-by-hop fragmentation performed slightly better than NNF in cases where small chunk sizes resulted in only 1 or 2 fragments. This is likely due to the fact that the hop-by-hop protocol accumulates less per-hop delay than computing the SHA-256 digest on each fragment. Moreover, we used an unoptimized C implementation of SHA-256, which further contributed to this result.

IX. Conclusion

In this paper we presented Named Network Fragments (NNF), a new cut-through fragmentation protocol that provides security guarantees of FIGOA while offering improved performance. NNF is unique in that it enables (a) fragments to be selective retransmitted, (b) unbounded content object lengths, (c) and immediate signature verification. Furthermore, the NNF packet format could be used to entirely replace the existing content object format. Our analytical model indicates that, in the presence of loss, NNF outperforms hop-by-hop fragmentation due to the ability to selective retransmit lost packets instead of retransmitting the entire message. Our

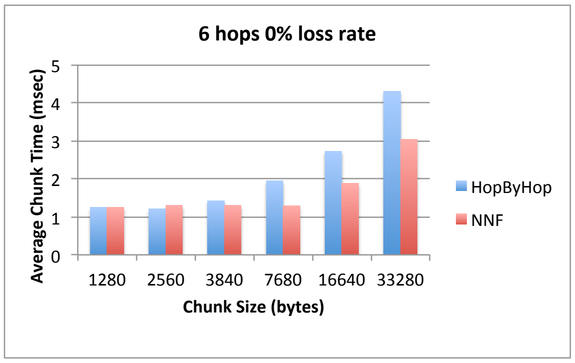

Fig. 5. Chunk retrieval times for NNF and hop-by-hop fragmentation over a 6-hop topology with lossless links.

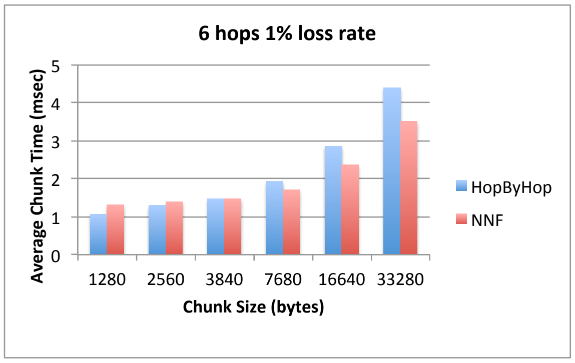

Fig. 6. Chunk retrieval times for NNF and hop-by-hop fragmentation over a 6-hop topology with $p = 0.01$ lossy links.

experimental comparisons support this result. Given the utility and efficiency of NNF, it is likely that this protocol will be adopted as the standard fragmentation protocol in CCNx 1.0, and possibly even a replacement for the existing Interest and Content Object packet format.

References

[1] Cesar Ghali, Ashok Narayanan, David Oran, Gene Tsudik, and Christopher A Wood. Secure fragmentation for content-centric networks. *arXiv preprint arXiv:1405.2861*, 2014.
[2] Junxiao Shi and Beichuan Zhang. NDNLP: A Link Protocol for NDN.
[3] Packet Fragmentation in NDN: Why NDN Uses Hop-By-Hop Fragmentation. http://named-data.net/wp-content/uploads/2015/05/ndn-0032-1-ndn-memo-fragmentation.pdf, 2015.
[4] M. Mosko. CCNx End-To-End Fragmentation, 2015.
[5] M. Mosko. ICN Hop by Hop Fragmentation, 2015.
[6] CCNx Binary Encoding (ccnb), 2012.
[7] B Lloyd, D Carr, G McGregor, and K Sklower. The ppp multilink protocol (mp). 1994.
[8] Alberto Leon-Garcia and Indra Widjaja. *Communication networks*. McGraw-Hill, Inc., 2003.
[9] Miltiades E Anagnostou and Emmanuel N Protonotarios. Performance analysis of the selective repeat arq protocol. *Communications, IEEE Transactions on*, 34(2):127–135, 1986.