Secure Fragmentation for Content-Centric Networks

arXiv:1405.2861v1 [cs.NI] 12 May 2014

***Abstract*—Content-Centric Networking (CCN) is a communications paradigm that emphasizes content distribution. Named-Data Networking (NDN) is an instantiation of CCN, a candidate Future Internet Architecture. NDN supports human-readable content naming and router-based content caching which lends itself to efficient, secure and scalable content distribution. Because of NDN's fundamental requirement that each content object must be signed by its producer, fragmentation has been considered incompatible with NDN since it precludes authentication of individual content fragments by routers. The alternative of hop-by-hop reassembly is problematic due to the substantial incurred delay. In this paper, we show that secure and efficient content fragmentation is both possible and even advantageous in NDN and similar information-centric architectures that involve signed content. We design a concrete technique that facilitates efficient and secure content fragmentation in NDN, discuss its security guarantees and assess performance. We also describe a prototype implementation and compare performance of cut-through with hop-by-hop fragmentation and reassembly.**

I. Introduction

The Internet is a *de facto* public utility used by a significant fraction of the humankind who rely on it for numerous daily activities. However, despite unparalleled success and unexpected longevity, the current TCP/IP-based protocol architecture may be obsolete. To this end, several research efforts to design a next-generation Internet architecture have sprung up in recent years [23].

One key motivator for a new Internet architecture is the fundamental shift in the nature of traffic: from the mainly low-bandwidth interactive (e.g., remote log-in) and store-and-forward (e.g., email) nature of the early Internet to the web-dominated Internet of today. At the same time, massive and rapidly-increasing amounts of content are produced and consumed (distributed) over the Internet. This transpires over social networks such as Facebook, media-sharing sites such as YouTube and productivity services such as GoogleDocs. Consequently, the emphasis of Internet communication has shifted from telephony-like conversations between two IP interfaces to a consumer who wants content delivered rapidly and securely, regardless of where it comes from. This motivates reconsidering the Internet architecture.

Content-Centric Networking (CCN) [14], [16], [18] is an approach to (inter-)networking designed for efficient, secure and scalable content distribution [16]. In CCN, named content – rather than named interfaces or hosts – is treated as a first-class entity. Named-Data Networking (NDN) [22] is an example of CCN that stipulates that each piece of named content must be signed by its producer (also known as publisher). This allows decoupling of trust in content from trust in an entity (host or router) that might store and/or disseminate that content. These NDN features facilitate in-network caching of content to optimize bandwidth use, reduce latency, and enable effective simultaneous utilization of multiple network interfaces.

NDN is an on-going research effort and one of several architectures being considered as a candidate future Internet architecture. Other such efforts include: ChoiceNet [33], XIA [15], Mobility-First [28] and Nebula [3]. Regardless of which, if any, approach eventually succeeds, all of them need to address some of the same issues, such as naming/addressing, routing/forwarding and security/privacy.

A. Fragmentation:

One issue that straddles both networking (specifically, packet forwarding) and security is fragmentation of large packets. Originally present in IPv4 [26], intermediate fragmentation was deprecated in IPv6 [10], for a number of reasons, many of which were identified in by Mogul [17], e.g., router overhead and code complexity. Also, there have been attacks that took advantage of IPv4 reassembly [34]. Thus, eschewing fragmentation made sense for IPv6. However, the same might not hold for all network architectures. We believe that, for some very different types of networking, such as CCN/NDN, fragmentation is sometimes unavoidable and might even be beneficial.

B. Focus:

This work represents the first exploration of efficient and secure fragmentation in the context of CCN/NDN. Its primary value is the construction of a secure fragmentation scheme which addresses several important security and efficiency issues. (Section VIII-B describes current handling of fragmentation in CCN/NDN.)

The intended contribution of this paper is two-fold: **First,** we discuss, in detail, numerous issues reated to fragmentation of both interest and content packets in NDN and arrive at the following key conclusions:

- Interest fragmentation is unavoidable: if encountered, hop-by-hop reassembly is a must.
- Content fragmentation is similarly unavoidable.
- Minimal MTU discovery helps, but does not obviate the need for fragmentation.
- Intermediate reassembly is viable but buffering can be costly and latency is problematic.
- Intermediate re-fragmentation is also unavoidable for content fetched from router caches.
- Reconciling cut-through forwarding of fragments (no intermediate reassembly) with content authentication is possible in an efficient manner.

Second, we construct a secure fragmentation and reassembly method for content-centric networks architectures, exemplified

by NDN. We call this method *Fragmentation with Integrity Guarantees and Optional Authentication* or FIGOA. It supports fragmentation of content packets at the NDN network layer and allows cut-through switching in routers by avoiding hop-by-hop fragmentation and reassembly, thus lowering end-to-end delay.

FIGOA is fundamentally compatible with the NDN's tenet of not securing the channel but rather the content flowing through it. As its basis, FIGOA employs a delayed authentication method similar to [32], which allows routers to verify signed content based on out-of-order arriving fragments. Furthermore, FIGOA supports nested (successive or recursive) fragmentation. In the event that a given content ultimately fails either integrity or authenticity check, reassembly and eventual delivery of corrupt content to consumers is prevented.

a) Organization: Section II provides some background on NDN. Thne, Section III summarizes fragmentation in IP. Fragmentation issues in NDN are discussed Section IV, The proposed FIGOA scheme is presented in Section V. Section VI describes its prototype implementation which is then evaluated in Section VII. Next, Section VIII overviews related work. The paper ends with the summary and future work in Section IX.

II. NDN Overview

This section overviews NDN. In case of familiarity with NDN, it can be skipped with no lack of continuity.

NDN supports two types of packets: *interest* and *content* [7]. The latter contains a human-readable name, actual data (content), and a digital signature over the packet computed by the content producer. Names are hierarchically structured, e.g. `/ndn/usa/cnn/frontpage/news` where "/" is the boundary between name components. An interest packet contains the name of the content being requested, or a name prefix, e.g. `/ndn/usa/cnn/` is a prefix of `/ndn/usa/cnn/frontpage/news`. In case of multiple contents under a given name prefix, optional control information can be carried within the interest to restrict the content returned. Content signatures provide data origin authenticity, however, trust management between a key and a name prefix is the responsibility of the application.

All NDN communication is initiated by a (content) consumer that sends an interest for a specific content. NDN routers forward this interest towards the content producer responsible for the requested name, using name prefixes (instead of today's IP prefixes) for routing. A *Forwarding Information Base* (FIB) is a lookup table used to determine interfaces (called *faces* in NDN) for forwarding incoming interests, and contains [*name_prefix*, *face*] entries. Multiple entries with the same *name_prefix* are allowed, supporting multiple paths over which a given *name_prefix* namespace is reachable. Akin to an IP forwarding table, FIB can be populated either by a routing protocol or manually.

NDN communication adheres to the "pull" model, whereby content is delivered to consumers only following an explicit request (interest). NDN content includes several fields. In this paper, we are only interested in the following:

- `Signature` – a public key signature, generated by the content producer, covering the entire content, including all explicit components of the name and a reference to the public key needed to verify it.
- `Name` – a sequence of explicit name components followed by an implicit digest (hash) component of the content that is recomputed at every hop. This effectively provides each content with a unique name and guarantees a match with a name provided in an interest. However, in most cases, the hash component is not present in interest packets, since NDN does not provide any secure mechanism to learn a content hash a priori.
- `KeyLocator` – a reference to the public key required to verify the signature. This field has three options: (1) verification key, (2) certificate containing the verification key, or (3) NDN name referencing the content that contains the verification key.

Each NDN router maintains a Pending Interest Table (PIT) – a lookup table containing outstanding [*interest*, *arrival-interface(s)*] entries. The first component of each entry reflects the name of requested content, and the second – a set of interfaces via which interests for this content have arrived. When an NDN router receives an interest, it looks up its PIT to determine whether an interest for the eponymous content is pending. There are three possible outcomes:

1) If the same name is already in the router's PIT and the arrival interface of the present interest is already in the set of *arrival-interfaces* of the corresponding PIT entry, the interest is discarded.
2) If a PIT entry for the same name exists, yet the arrival interface is new, the router updates the PIT entry by adding a new interface to the set; the interest is not forwarded further.
3) Otherwise, the router looks up its cache (called Content Store) for a matching content. If it succeeds, the cached content is returned and no new PIT entry is needed. Conversely, if no matching content is found, the router creates a new PIT entry and forwards the interest using its FIB.

An interest might reach the actual content producer if no corresponding content has been cached by any intervening router on the path between consumer and producer. Upon receipt of the interest, the producer injects requested content into the network, thus *satisfying* the interest. The content is then forwarded towards the consumer, traversing – in reverse – the path of the preceding interest. Each router on the path flushes state (deletes the PIT entry) containing the satisfied interest and forwards the content out on all arrival interfaces of the associated interest. In addition, each router may cache a copy of forwarded content. Unlike their IP counterparts, NDN routers can forward interests out on multiple interfaces in order to increase likelihood of fastest content retrieval.

Not all interests result in content being returned. If an interest encounters either: (1) a router that can not forward it further or (2) a content producer that has no such content, no error packets are generated. PIT entries in intervening routers

simply expire if content can not be retrieved. The consumer (who also maintains its local PIT) can choose to re-issue the same interest after a timeout.

III. Fragmentation Synopsis

We define *fragmentation* as a means of splitting large packets into smaller packets, at the network layer, independent of content publisher and without changing any actual *content*. This is in contrast with *segmentation* where a content publisher splits a large content object into smaller ones, signing and namning each separately. TCP/IP has an analogous dichotomy: TCP *segments* a byte stream into IP packets, whereas, IPv4 can *fragment* IP packets into smaller packets in order to fit them into a link MTU.

Since late 1980s, network-layer fragmentation has been widely considered to be a headache and something to be avoided, based primarily on the IPv4 experience [17]. To understand this further, we briefly discuss IPv4 fragmentation concepts.

Packet fragmentation is not a singular concept; it can be divided into two types: source-based and network-based. Source-based fragmentation is performed exclusively by the sender and is relatively simple: assuming knowledge of the Maximum Transmission Unit (MTU) for a given path to the destination, the source can fragment a packet with almost[1] no fear that further fragmentation might be encountered along the path. Of course, knowledge of the MTU does not come for free; an MTU discovery protocol is needed, e.g., [20]. Also, the whole premise of source-based fragmentation is questionable: **Why should the source fragment a large IP packet instead of simply "segmenting" it into a sequence of separate *bona fide* IP packets** [19]. Source-based segmentation often allows for more efficient use of smaller datagrams; for example, segmented TCP datagrams can be individually acked, whereas a larger TCP segment split using IP fragmentation can only be processed as a whole by TCP.

Network-based fragmentation requires routers to support extra functionality (i.e., additional code) which entails appreciable processing overhead [17]. Having to fragment a packet takes a router off its critical path and can thus cause congestion; this can also be exploited as denial-of-service attack. Nonetheless, at a conceptual level, it can be claimed that intermediate fragmentation offers better bandwidth utilization than its source based counterpart, or no fragmentation at all.

Further issues are prompted by reassembly of fragments. In IPv4, reassembly takes place only at the destination. Each IP packet is allocated a buffer that stores fragments that have arrived thus far; possibly, out-of-order. Once all fragments are received, the packet is physically reassembled and passed on to the upper layer. This seemingly simple procedure has been a source of many attacks and exploits [34]. Reassembly by intermediate hops/routers is not viable in IP, since fragments of the same packet are not guaranteed to travel over the same path.

[1]Dynamic routing in IP may cause successive packets to take different paths, affecting the source's perceived MTU.

IV. Fragmentation in NDN

NDN architecture does not ptovide explicit support for intermediate fragmentation [1]. However, the current NDN implementation, that runs as part of the NDN testbest [2], is implemented as an overlay on top of TCP or UDP. In this a setup, fragmentation is handled by either: (1) transport layer protocols, i.e.,TCP segmentation, or by (2) network layer protocols, i.e., IP fragmentation. Moreover, if NDN is running directly over the link layer, protocols such as NDNLP [30] can be used to handle fragmentation. (See Section VIII-B for details.) The main drawback of thess methods is that they all need reassembly at every hop.

The rest of this section discusses certain factors motivating fragmentation in NDN. Using terminology in Figure 1, fragmentation is considered in the context of interest and content packets, respectively.

Maximum Transmission Unit (MTU): largest unit (packet) size for network-layer transmission over a given link between two adjacent nodes.
Publisher or Producer: an entity that produces and signs content; we use these two terms interchangeably.
Consumer: an entity that requests (consumes) content.
Router: a network-layer entity that routes content but neither produces nor consumes it (except perhaps for routing information).
Content object (CO): a unit of NDN data, named and signed by its producer/publisher.
Content fragment (CF): a unit of NDN network layer transmission; content fragment is the same as content object if the latter fits within the MTU of a link between two adjacent routers.
Segmentation: a process of partitioning large content into separate content objects by explicitly naming and signing each one. Can be performed only by a producer of content.
Fragmentation: a process of splitting an (already signed and named) content object into multiple content fragments. Can be performed by a producer, a router or any other NDN entity that produces, stores or caches content.
Re-fragmentation: a process of splitting a fragment of a content object into multiple fragments. Sometimes referred to as inter-network fragmentation [17]. Can be performed by a router.
Reassembly: a process of re-composing a content object from its fragments. Can be performed by a consumer or a router (in case of intermediate reassembly).
Fragment Buffering: a process of maintaining a stash of fragments until complete packet reassembly becomes possible.
Cut-Through Switching (of fragments): forwarding of individual content fragments without reassembly.

Fig. 1. Terminology

A. Fragmentation of Interests

As discussed in Section II, an interest packet carries the name of content requested by the consumer. NDN does not mandate any confidentiality, integrity or authenticity requirements for interest packets. Due to no limitation on the length of content names, it is quite plausible that an interest packet carrying a very long name might not fit into a network-layer MTU, thus prompting the need for source-based and/or intermediate fragmentation. Fortunately, this does not pose any real challenges, since the "design space" of interest fragmentation is very confined. Specifically, we claim that:

If interest packets are fragmented and, possibly re-fragmented, hop-by-hop (intermediate) reassembly of fragmented interest packets is a necessity.

This claim is easy to support because, as described in Section II, each router that receives an interest must look

up its PIT and/or cache using the name carried in that interest. If the name itself spans multiple fragments which are processed independently (without reassembly), such lookups are infeasible.

Furthermore, since the consumer issuing an interest has no *a priori* knowledge of the smallest MTU on the path to the closest copy of requested content, it can not pre-fragment an interest in order to avoid further fragmentation by intermediate routers, unless there is a well-defined global minimum MTU for NDN interests.

Based on the above, for the remainder of this paper, we assume source-based (and possibly intermediate) fragmentation coupled with intermediate reassembly for interests. The remaining discussion of fragmentation is limited to NDN content packets.

B. Fragmentation of Content

Recall that NDN mandates each content to be signed by its producer. This means that, in principle, any NDN entity, whether router or consumer[2] is able to check content integrity and authenticity, based on the producer's public key, itself embedded in a separate signed content (a de facto credential or certificate). The public key can be either referred to by name in the content header, or enclosed in its entirety as part of the content.

Consequently, in order to abide by NDN tenets, fragmentation must not preclude routers from verifying signatures, i.e., checking authenticity and integrity of content. This speaks in favor of either: (1) no intermediate fragmentation at all, or (2) intermediate (hop-by-hop) reassembly.

1) **Producer-based Fragmentation or Segmentation:** At the first glance, there seems to be no reason for a content producer to fragment large content. Instead, it can simply *segment* it into individually named and separately signed content objects. This segmentation approach is sensible for content meant to be pushed (e.g., email) or generated dynamically, e.g., in response to a database query. Segment size can be determined from an MTU discovery protocol (see Section IV-B2 below). This would assure no intermediate fragmentation.

However, for content that is meant to be pulled (distributed), a producer may benefit from signing and naming it once and not worrying about repeating (possibly expensive) segmentation procedure each time it receives an interest for the same content. In this case, when an interest arrives, the producer may choose to fragment a previously produced content object. This entails no real-time cryptographic overhead. Alternatively, a producer could choose to segment content using the smallest MTU of all of its interfaces, thus incurring even less processing at interest arrival time.

An important issue is the content header overhead incurred when generating small-size segments. Segmenting a large object into many MTU-sized segments requires each of them to have its own header, dominated by the Signature component which contains a number of fields.

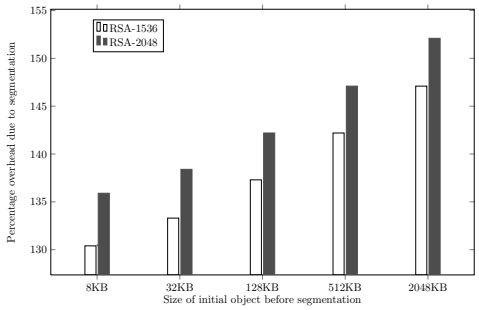


Fig. 2. Byte count overhead for small signed segments

Without getting into details of NDN signature format, Figure 2 shows the overhead of segmenting larger objects down to MTU size. We use a standard $1,500$-byte link MTU and SHA-256 as the hash algorithm. We considered both RSA-1536 and RSA-2048 signatures. The Signature field therefore contains: 12 bytes of fixed overhead (headers), actual signature bits (192 bytes for RSA-1536, 256 bytes for RSA-2048). However, estimating the exact size of the signature field is more complex. This is because the `KeyLocator` field (which is part of the signature field) can be of arbitrary size (and if it carries a certificate, it can be *very* large). For now, we assume a small 20-byte `KeyLocator`, along with an SHA-256 hash. Figure 2 shows that there is a definite penalty for segmenting at the publisher. Even in the most favorable case (8KB data objects, RSA-1536), over 30% of the bits are wasted on redundant information. As we move to larger objects, this overhead can grow to 50%.

2) **Whither Intermediate Fragmentation:** Regardless of whether a producer segments or fragments content, intermediate fragmentation can not avoided or ruled out, since NDN does not mandate a globally minimal MTU. Even if it existed, segmenting content to adhere to this MTU might be very wasteful due to poor bandwidth utilization (on links that have higher MTUs) and cryptographic overhead due to increased costs of signature generation by producers and signature verification by consumers as well as (optionally) by routers.

Another possibility is to introduce an MTU discovery method, whereby, for example, an interest traveling towards requested content could have a new field reflecting the smallest MTU (μMTU) discovered thus far on its path.[3] This is a viable and light-weight approach, particularly because, in NDN, a content must traverse, in reverse, the very same path taken by an interest for that content. Hence, when the first entity that stores, caches or produces requested content receives an interest, it can use μMTU to fragment (or segment, if this entity is the producer). Note that "entity" could encompass: (1) an application-level repository that stores content it

[2] Content signature verification is mandatory for consumers and optional for routers.

[3] This is actually the MTU of the *opposite* link direction from the direction the interest traveled - links may have asymmetric MTUs

does not produce, (2) a router that caches content or a (3) producer/publisher that generates its own content. This way, fragmentation would occur only once per interest.

However, fragmentation via interest-based μMTU discovery does not eliminate the need for re-fragmentation. Consider the following scenario:

1) Consumer A issues interest int_A for CO to router R.
2) R receives and marks int_A with $\mu MTU = MTU_{(R \to A)}$ (MTU corresponding $R - A$ link). It then creates a PIT entry for int_A.
3) R forwards int_A to adjacent producer P.
4) Since $MTU_{(P \to R)} > \mu MTU_{(R \to A)}$, P does not change μMTU in int_A.
5) P immediately satisfies int_A, fragmenting CO according to μMTU.
6) Meanwhile, between Step 3 and now, consumer B issues interest int_B and forwards it to R.
7) R receives int_B and marks it with $MTU_{(R \to B)}$ where $MTU_{(R \to B)} < \mu MTU$. R collapses int_B into existing PIT entry for int_A. At this time R is buffering fragments which have arrived from P, however, not all fragments of CO have arrived yet.
8) R partially satisfies int_B using fragments available in the buffer, previously forwarded to A. These fragments are re-fragmented with $MTU_{(R \to B)}$. Any further fragments which arrive from P are also re-fragmented by R to B using $MTU_{(R \to B)}$.

Despite the fact that μMTU discovery does not eliminate re-fragmentation, it is practically free in terms of extra processing and bandwidth overhead. More importantly, it results in less re-fragmentation, since it assures that re-fragmentation occurs **at most once** for each collapsed interest at each intermediate router. This can be particularly advantageous in the case of monotonically shrinking MTUs where re-fragmentation must occur at each hop. With μMTU, this is curtailed at the source of content (which is either some intermediate router or the producer) due to pre-fragmention.

C. Considering Intermediate Reassembly

We now discuss intermediate reassembly. There are at least two factors that motivate it.

First, we consider the case of increasing MTUs on links that compose the reverse path taken by content fragments on the way to the consumer. If MTUs increase monotonically, it might make sense to reassemble fragments (at least partially) to obtain better bandwidth utilization. However, this benefit is arguably outweighed by costs incurred by reassembly, i.e., processing, memory and code in routers. (Recall that, as in today's IPv4, reassembly would require a router to maintain a separate buffer pool for all fragments of a given content received so far.)

The second factor is security: if a fragment does not carry the content producer's signature, how can a router check its authenticity? As mentioned earlier, NDN stipulates that routers, though not required to do so, must be able to verify content signatures. As we argued above, it seems infeasible

	Number of flows					
	5	**10**	**20**	**30**	**50**	**100**
Inter-fragment gap (ms)	0.52	1.04	2.08	3.12	5.20	10.4
First-to-last fragment gap (ms)	3.22	6.34	12.58	18.82	31.30	62.50
E2E latency: reassembly (ms)	105.79	130.75	180.67	230.59	330.43	580.03
E2E latency: cut-through (ms)	83.22	86.34	92.58	98.82	111.30	142.50
Reassembly slowdown %-age	127.12	151.43	195.14	233.34	296.87	407.03

TABLE I
LATENCY DUE TO PER-HOP CONTENT REASSEMBLY

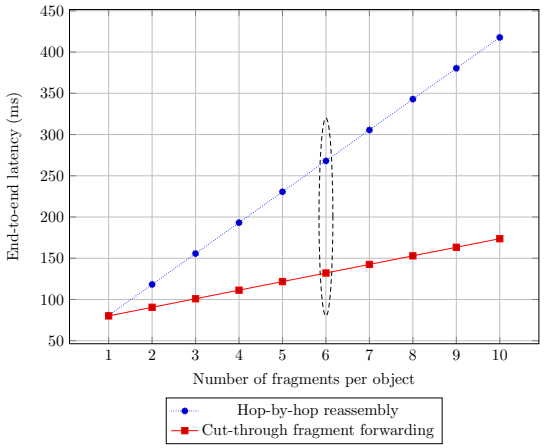


Fig. 3. Latency with different fragment counts per object

for the producer to pre-fragment or pre-segment content such that each possible future fragment of that content would carry the producer's signature.

Hop-by-hop reassembly of content fragments would clearly solve the problem and address both factors mentioned above. With it, a router would receive fragments in arbitrary order and neither cache nor forward them until all fragments arrive. It would then reassembles them and verify the content signature with the producer's public key. (See Section V-E for more details.)

The main problem with hop-by-hop reassembly is increased end-to-end latency, resulting into lower throughput for adaptive algorithms, such as TCP. If multiple flows are passing through the router, the fairest distribution of latency overhead is to interleave fragments, as in MLPPP LFI [31]. This interleaving causes significant latency between consecutive fragments of an object, which grows with the number of simultaneous flows. Latency accumulates at each hop, since all fragments need to be reassembled and then re-fragmented for transmission. The alternative is cut-through fragment forwarding, where each fragment is forwarded immediately, as it arrives.

We attempt to evaluate the benefits of cut-through fragment forwarding by considerng a simple topology with a linear 8-hop path with 100 Mb/s links. Each link accumulates 10ms of latency, ignoring intra-hop and queueing delays for now. We assume $8,400$-byte content objects split into 7 fragments of $1,300$ bytes each.

Table I shows the slowdown caused by intermediate reassembly as each node waits for all fragments of an object, for

varying numbers of parallel flows (which controls the amount of interleaving). *Inter-fragment gap* is the time elapsed between consecutive fragments of an object, caused by fragments of other objects being interleaved. *First-to-last fragment gap* is the time elapsed between the arrival of the beginning of the first fragment and the end of the last fragment. *E2E latency – reassembly* is the total latency for each content object, with intermediate reassembly. *E2E latency – cut-through* is the total latency in case of an object fragmented at the first hop and all fragments cut-through forwarded with no re-fragmentation or reassembly in route. Finally, *Reassembly Slowdown* shows the extra cost of reassembling and refragmenting at every hop, as compared with cut-through forwarding.

Figure 3 shows the evolution of increased latency for various object sizes and fragment counts. We re-use the 8-hop topology described above and vary the number of flows on each link. Two links (close to the ends) have 10 flows across them, two links have 20 flows, two links have 50 flows and the two core links have 100 flows. The graph shows that even a small number of fragments can significantly increase latency over commonly seen path lengths and flow counts. It takes only 6 fragments per object to **double** end-to-end latency of hop-by-hop reassembly when compared with cut-through forwarding of fragments. This clearly shows that any fragmentation scheme that requires hop-by-hop reassembly of every content object (as is the case today with CCNx [6] and NDNLP [30]) incurs severe penalties in a wide-scale deployment. We believe that content object fragments must be forwarded in a cut-through fashion; consequently, our scheme implements this feature.

At this point, it is worth asking: should routers perform reassembly and verify signatures? And if yes, which ones? We believe that it does not make much sense for backbone routers to do so since potential attacks are not likely to originate in the backbone, but rather at the edges of the Internet. Whereas, signature verification at stub AS (ingress) routers is more appropriate, e.g., because of policy dictating that no fraudulent content must reach consumers.[4] Also, stub AS egress routers might reassemble fragments and verify signatures if there is a policy disallowing any fraudulent content to exit an AS, e.g., for reasons of liability.

The above discussion yields a trivial observation that reassembly implies ability to verify signatures. However, it is unclear whether signature verification implies the need for reassembly. This triggers the following challenge which we attempt to address in the remainder of this paper:

> *If content objects are fragmented (and, possibly re-fragmented) and intermediate reassembly is not viable, can routers still check content authenticity?*

In other words, if verifying integrity/authenticity is the main reason for intermediate reassembly, is there a way to obtain the former while avoiding the latter?

[4]Even though NDN stipulates consumer-based signature verification.

D. Fragment Delivery Order

One important issue relevant to intermediate reassembly and to the proposed technique (Section V) is whether fragments are always delivered in transmission order between any two adjacent NDN routers.

Clearly, reassembly is easier if ordered delivery can be guaranteed. In a hypothetical network setting where NDN is universally deployed directly on top of the physical network links, ordered delivery of content fragments might be a reasonable assumption. [5] However, certain connectivity and communication choices make ordered delivery less likely. For instance, if adjacent routers support multiple/parallel physical links with variable speeds, it is possible that an earlier-transmitted fragment is received later than a later-transmitted one. Also, an error on one of the links might cause the same situation even if link speeds are comparable. Even without multiple links, if pipelined data-link layer transmission is used, especially over the wireless channel, one fragment could be corrupted and discarded and the next one could be received intact, resulting in the latter being received first.

E. Incremental or Deferred Fragment Caching?

Recall that one of the key features of NDN is router-based content caching. This is not, strictly speaking, a hard requirement, however, it is expected that each NDN router will maintain a Content Store (CS), i.e., a cache, of a certain size.

A router that employs intermediate reassembly can defer the decision to cache content until it receives all fragments and, optionally, verifies overall content integrity and/or authenticity. Whereas, a router that employs *cut-through switching* of individual fragments has a choice to either: (1) cache fragments incrementally as they arrive, or (2) defer caching (i.e., buffer fragments) until all fragments arrive and, optionally, their overall content integrity and/or authenticity is verified. Assuming that most content is authentic, the former optimizes the common case of quickly caching the last fragment once optional security checks are performed. On the other hand, incremental caching may complicate matters, since it might, depending on the specific cache architecture, involve non-contiguous caching of related fragments.

If deferred caching is used, another fragmentation-and caching-related issue is how to store fragments? One possibility is to store them in the same form they arrive. This might work if no re-fragmentation is performed locally. Otherwise, it might make sense to store fragments in the form they are forwarded. This gets more complicated in case of collapsed interests, i.e., when content needs to be forwarded out on multiple interfaces with different MTUs. Another approch would be to proactively re-fragment cached fragments for all possible link MTUs on the router. This pre-fragmentation would reduced delay at the cost of additional buffer space. We believe this issue deserves further consideration; which is beyond this paper's scope.

[5]All content fragments traverse, in reverse, the very same path taken by an interest.

V. Secure Fragmentation

This section describes a scheme called FIGOA: Fragmentation with Integrity Guarantees and Optional Authentication. It supports arbitrary intermediate fragmentation [17] of content while preserving security, without requiring intermediate reassembly.[6] FIGOA allows free mixing of routers that do not perform intermediate reassembly with those that do. It is primarily geared for routers that support cut-through switching and maintain dedicated storage for buffering content fragments, distinct from Content Store. While cut-through fragment switching is generally beneficial, it complicates signature verification, as discussed in Section IV-C, FIGOA addresses this problem by using delayed authentication.

A. Delayed Authentication

Delayed authentication is an old method described in [32]. Its goal was to: *"reconcile fragmentation and dynamic routing with network-level authentication in IP gateways."* The essence of delayed authentication is that a given packet's authenticity can be obtained from authenticity of its fragments. Packet authentication is computed incrementally as individual fragments are received (possibly out of order), processed and forwarded by a router. This requires a queue for each partially received packet that maintains the current state of partial verification. For every fragment, incremental verification is performed, queue state is updated and the fragment is forwarded. Upon receipt of the final fragment, the router completes verification. If it succeeds, the final fragment (called a "hostage") is forwarded. Otherwise, it is discarded along with the entire queue. The end-result is that the destination receives the packet in its entirety only if it is verified by the router.[7]

Main differences between delayed authentication in its original IPv4 context [32] and our proposed use in NDN are as follows:

- Symmetric routing: unlike IP, where fragments of the same IP packet might travel via different paths, fragments of the same NDN content are guaranteed to follow the same sequence of NDN routers, retracing PIT state set up by a preceding interest. This results in higher probability of ordered fragment delivery and faster time-outs in cases of lost or corrupted fragments. Note that it is the responsibility of consumers to re-request the whole content in case of lost or corrupted fragments.
- Not just ingress routers: delayed authentication was initially suggested only for ingress routers (i.e., border routers of the destination's AS). In NDN, any intermediate router can unilaterally choose to perform delayed authentication.
- Possible intermediate reassembly: in IP, only the destination reassembles fragments, whereas, any intervening router can decide to reassemble whether or not it decides to do cut-through forwarding.
- Signatures instead of CBC-based MAC: delayed authentication was initially proposed for authenticating IP packet traffic flowing between two hosts (in two stub AS-s) that share a symmetric key. The same key is shared with the ingress router. Actual packet authentication is attained via a message authentication code (MAC) based on the chained-block cipher (CBC) mode of symmetric encryption. The main idea is to insert intermediate MAC values thus allowing incremental authentication of fragments. Whereas, in the NDN context, MAC-s are not viable, since doing so would require sharing a symmetric key among all intervening routers.

B. Hash Functions

The last item above – the use of signatures – is what most distinguishes delayed authentication in NDN from its IP counterpart. NDN routers do not use symmetric cryptography for packet authentication. Even if they did, assuming a key shared among (possibly all) routers that forward a given content is unrealistic. The only means of authenticating content in routers is by verifying signatures. This prompts the question: how to reconcile delayed authentication (of fragments) with signatures?

We approach this issue by observing that a signature is computed over a fixed-size hash digest (or simply hash) of content, i.e., using the so-called "hash-and-sign" paradigm. A hash provides integrity while a signature of a hash provides authenticity or origin authentication. The underlying cryptographic hash function $H(\cdot)$ must satisfy a set of standard properties [21]. Unlike a MAC or a keyed hash [4], a hash function requires no secret key and can be computed by anyone.

Most modern hash functions operate on input of practically any[8] size. They typically use an iterative model (also known as the Merkle-Damgard construction), whereby input is broken into a number of fixed-size blocks and is processed one block at a time by an internal compression function $HC(\cdot)$. The latter forms the core of the hash function; after processing each block, it produces an intermediate value (internal state that we call IS) that is usually of the same size as the final hash. In case of the first block, the intermediate state is fixed and referred as the Initialization Vector (IV). The last block is typically padded with zeros followed by the total input size in bits. For example, the well-known SHA-256 [29] operates on 512-bit blocks, maintains 256-bit internal state and yields a 256-bit hash.

As described below, in constructing FIGOA, we take advantage of internal state produced by the underlying compression function $HC(\cdot)$. The main idea is to include, in each fragment, the internal state of the hash function **up to**, but not including, that fragment. This allows incremental hashing of each fragment without having received either preceding or subsequent fragment(s).

[6] A variant of FIGOA can be used in conjunction with intermediate reassembly, with the key advantage of faster cryptographic processing.

[7] Recall that, in IPv4, the destination must flush all fragments of a packet that it can not reassemble, either due to a time-out or another error.

[8] We consider 2^{64} or 2^{128} bits as "practically any".

β	block size of $HC(\cdot)$
CO^n	Raw (unsigned) content of total size n bits.
$SIG(CO^n)$	Producer's signature on CO^n.
$\overline{CO}^N$	Signed version of CO^n of size $N = n + \|SIG(CO^n)\|$ bits.
$b_{v,s}$	Contiguous component of $\overline{CO}^N$ where $0 \leq v < N$, i.e., $b_{v,s}$ represents s bits, starting with offset v and ending with offset $v + s - 1$, inclusive. s and v are multiples of β.
$CF^N_{v,s}$	Fragment of $\overline{CO}^N$ that carries $b_{v,s}$.
IS_v	Internal state of $HC(\cdot)$ after processing v bits of input. v is a multiple of β
hs	Fragment header size, includes: content name, v, s and IS_v. See Section VI for details.
$o\mathcal{MTU}$	MTU of router's outgoing interface.
$ao\mathcal{MTU}$	$o\mathcal{MTU}$ adjusted for fragment header size hs, i.e., $ao\mathcal{MTU} = o\mathcal{MTU} - hs$
$\mathbb{F}$	Set of content fragments.
$\mathbb{B}$	Temporary buffer storing all fragments received so far.

TABLE II
NOTATION

We assume that the absolute minimum MTU of any link or interface that takes advantage of FIGOA is at least one block of data, one block of internal state and whatever size is needed to accommodate a content fragment header (i.e., content name, flags, etc.). More precisely, we assume that any fragment must carry at least a header, internal state and some data represented as (at least one) some blocks of data. All data must be aligned with block boundaries.

C. FIGOA Description

From here on, we use additional notation reflected in Table II. The proposed scheme includes three main tasks, described separately below.

1) Content Fragmentation: This task, shown in Algorithm 1, is triggered whenever an NDN node (router or producer) needs to forward a content object larger than $o\mathcal{MTU}$. Each resulting fragment $CF^N_{v,s}$ includes: (1) s bits of original content – $b_{v,s}$, (2) starting offset v, and (3) IS_v – intermediate state, i.e., output of $HC(\cdot)$ on inputs of: IV and $b_{0,v-1}$.[9] ($IS_v = HC(IV, b_{0,v-1})$.) To simplify presentation, Algorithm 1 makes two assumptions: **First**, $ao\mathcal{MTU}$ is a multiple of β, i.e., $ao\mathcal{MTU} = s * \beta$. **Second,** N (signed content size) is a multiple of $ao\mathcal{MTU}$, i.e., $N = k * ao\mathcal{MTU}$, which makes all fragments of equal size.

Algorithm 1: Fragment-Content

1: **Input:** signed content $\overline{CO}^N = b_{0,N-1}$, $ao\mathcal{MTU}$, IV, $HC(\cdot)$
2: **Output:** $\mathbb{F}$
3: $\mathbb{F} := \emptyset$, $v = 0$, $IS_v = IV$
4: $s = ao\mathcal{MTU}/\beta$, $k = N/s$
5: **for** $i = 0$, $i < k$, $i++$ **do**
6: $\quad CF^N_{v,s} := \langle v, b_{v,s}, IS_v \rangle$
7: $\quad \mathbb{F} := \mathbb{F} \cup CF^N_{v,s}$
8: $\quad IS_v := HC(IS_v, b_{v,s})$
9: $\quad v = v + s$
10: **end for**
11: Output $\mathbb{F}$

[9]In the very first fragment, $v = 0$ and $IS_v = IV$.

2) Fragment Re-fragmentation: This task, in Algorithm 2 is very similar to the initial fragmentation task, except that it is performed only by NDN routers, and on content fragments, instead of content objects.

Algorithm 2: Refragment-Fragment

1: **Input:** $CF^N_{v,s} = \langle v, b_{v,s}, IS_v \rangle$, $o\mathcal{MTU}$, $HC(\cdot)$
2: **Output:** $\mathbb{F}$
3: $\mathbb{F}$:= Fragment-Content($b_{v,s}$, $o\mathcal{MTU}$, IS_v, $HC(\cdot)$)
4: Output $\mathbb{F}$

3) Content Verification: As mentioned earlier, FIGOA provides integrity/authenticity for fragments received in any order. Recall that a router or a consumer can unilaterally decide whether to either: (1) incrementally verify integrity of each fragment as it is received, or (2) defer overall verification until all fragments are received. Regardless of the choice, a router should forward each fragment in a cut-through fashion, i.e., without waiting for others to arrive. Moreover, a node receiving fragments should store them in a buffer until the last fragment arrives and (final or overall) verification is performed. (See Section V-E.)

When a router performing incremental fragment verification receives $CF^N_{v,s}$, one of the following cases occurs:

1) $CF^N_{v,s}$ is the very first received fragment. A new buffer $\mathbb{B}$ is created where $CF^N_{v,s}$ is placed. $IS^*_w = HC(IS_v, b_{v,s})$ is computed and stored.
2) Neither previous $CF^N_{u,s}$ (for $v = u + s$) nor next $CF^N_{w,s}$ (for $w = v + s$) fragment is in the buffer. $CF^N_{v,s}$ is placed in $\mathbb{B}$. $IS^*_w = HC(IS_v, b_{v,s})$ is computed and stored.
3) $CF^N_{u,s}$ is in the buffer (along with IS^*_v) but $CF^N_{w,s}$ is not. IS^*_v must match IS_v in $CF^N_{v,s}$. $IS^*_w = HC(IS_v, b_{v,s})$ is computed and stored.
4) $CF^N_{w,s}$ is in the buffer but $CF^N_{u,s}$ is not. $IS^*_w = HC(IS_v, b_{v,s})$ is computed and must match IS_w from $CF^N_{w,s}$.
5) Both $CF^N_{u,s}$ and $CF^N_{w,s}$ have already been received. IS^*_v must match IS_v in $CF^N_{v,s}$ $IS^*_w = HC(IS_v, b_{v,s})$ is computed and must match IS_w from $CF^N_{w,s}$

Once the last fragment is received, authenticity of the entire content can be finally verified. If verification fails, the last fragment is dropped, the PIT entry is flushed, and nothing is cached. The same applies for any failed check in the 5 cases above. This process is illustrated in more detail in Algorithm 3. We assume that routers perform incremental verification of fragments and verify reassembled content signature. If signature verification is not possible, routers must verify that the reassembled content hash matches the original content hash included in every fragment (see Section VI for details.)

D. Examples

We now describe FIGOA via two operational examples:

In the first example, consider a situation where $\overline{CO^N}$ has two fragments: $CF^N_{0,s}$ and $CF^N_{s,s}$, i.e. $N = 2 \times s$. A router R first receives $CF^N_{0,s}$. First, R invokes $HC(\cdot)$ iteratively and computes $IS^*_s = HC(IV, b_{0,s})$. Then, it forwards $CF^n_{0,s}$ out

Algorithm 3: Verify-Fragment

```
1:  Input: received CF^N_{v,s}, associated PIT entry e, HC(·)
2:  Output: no output
3:  if is_first(CF^N_{v,s}) then
4:      B := get_new_buffer();
5:  end if
6:  INSERT CF^N_{v,s} in B
7:  STORE IS*_w = HC(IS_v, b_{v,s})
8:  if CF^N_{u,s} ∈ B and IS*_v ≠ IS_v in CF^N_{v,s} then
9:      goto CleanUp
10: end if
11: if CF^N_{w,s} ∈ B and IS*_w ≠ IS_w of CF^N_{w,s} then
12:     goto CleanUp
13: end if
14: if is_not_last(CF^N_{v,s}) then
15:     FORWARD CF^N_{v,s} according to e
16: end if
17: if content_complete() then
18:     CO^N := assemble(B)
19:     if verify_sig(CO^N) then
20:         FORWARD CF^N_{v,s} according to e
21:         CACHE CO^N
22:         return
23:     end if
24: end if
25: CleanUp: FLUSH B and e
```

on the interface(s) reflected in the corresponding PIT entry. R creates a buffer for $\overline{CO}^N$ where it records the fact that it received the first s bits of content, along with the computed IS^*_s. Now, R receives $CF^N_{s,s}$. It compares stored IS^*_s with IS_s carried in $CF^N_{s,s}$; if they do not match, R discards the buffer and flushes the corresponding PIT entry. Otherwise, it invokes $HC(\cdot)$ iteratively and computes $IS^*_N = HC(IS_s, b_{s,s})$. At the end, R extracts $SIG(CO^n)$ (from the received content), and computes a putative hash H' of entire reassembled CO^n. Finally, R verifies whether $SIG(CO^n)$ is the producer's signature on H'. If so, $CF^N_{s,s}$ is forwarded; otherwise, it is discarded along with the buffer and the PIT entry.

A similar process takes place if $CF^N_{0,s}$ and $CF^N_{s,s}$ arrive out of order. R first receives $CF^N_{s,s}$. Using IS_s carried in this fragment, R invokes $HC(\cdot)$ iteratively on each block of data and terminates with IS^*_N. Next, R forwards $CF^N_{s,s}$. Then, R creates a buffer for $\overline{CO}^N$ where it records the fact that it received the last $N-s$ bits (which is in fact the last s bits) of content, along with IS_s and IS^*_N. Now, R receives $CF^N_{0,s}$. It invokes $HC(\cdot)$ iteratively and computes $IS^*_s = HC(IV, b_{0,s})$ which should match IS_s received earlier as part of $CF^N_{s,s}$: if they do not match, R discards the buffer and the PIT entry. Otherwise, R computes a putative hash H' of entire reassembled CO^n, extracts $SIG(CO^n)$ and verifies whether it is the producer's signature on H'. If so, $CF^N_{0,s}$ is forwarded; otherwise, it is discarded along with the buffer and the PIT entry.

The second operational example involves R receiving a fragment $CF^N_{x,s}$ of content $\overline{CO}^N$. The total size of this fragment is $(s + hs)$ bits. Suppose that, after processing this fragment as in the first example, R needs to forward it out on an interface with $o\mathcal{MTU}$ smaller than the total size of $CF^N_{x,s}$, e.g., R needs to re-fragment it into two sub-fragments. R creates $CF^N_{x,s'}$ with IS_x and $CF^N_{y,s'}$ with IS_y; where (1) $s' < s$, (2) $y = x + s'$, (3) IS_x is simply copied from $CF^N_{x,s}$, and (4) $IS_y = HC(IS_x, b_{x,s'})$. This example aims to show that R can easily re-fragment already-fragmented content while preserving overall content integrity.

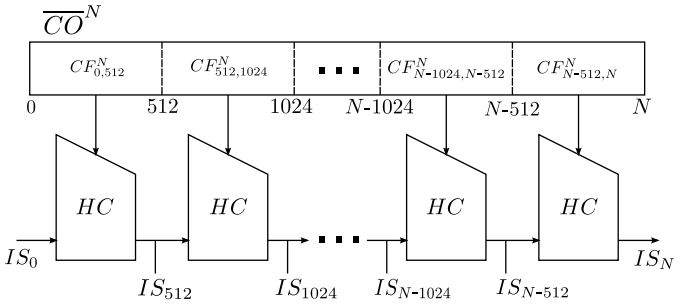


Fig. 4. Implementing Merkle-Damgard Construction to Generate Content Fragments

Figure 4 demonstrates how the Merkle-Damgard construction can be used to generate content fragments. The hash function used in the figure is SHA-256, and the length of the input is discarded at the end of the construction for simplicity of demonstration.

E. Content Authentication

Although trust and key management are out of the scope of this paper, we can not ignore the fact that authenticating a content object requires not only the presence of a signature, but also availability of a public key which must somehow be trusted [13]. Recall that NDN stipulates that public keys are encapsulated in named and signed content objects, i.e., a form of a certificate. Also, NDN allows the public key to be either (1) referred to by name within a content object's header, or (2) enclosed with the content object itself (using the `KeyLocator` field).[10] In the former case, unless the referred public key is already cached, the router presumably must fetch it by name, i.e., issue an interest for it. This is a burdensome task that routers should not perform, for obvious reasons.

F. Security Analysis

Security of FIGOA is based on that of delayed authentication (DA).

We say that $H(\cdot)$ is constructed using the Merkle-Damgard construction using its inner compression function $HC(\cdot)$ as a building block. If $HC(\cdot)$ is collision-resistant, then so is $H(\cdot)$.

A function F is collision-resistant if it is "computationally infeasible" to find inputs $x \neq y$ such that $F(x) = F(y)$. See [21] for information regarding Merkle-Damgard construction and hash-and-sign paradigm; also known as "digital signature with appendix."

A signature computed via hash-and-sign over an unfragmented content object is considered secure. Whereas, with DA, a content object is fragmented and we arrive at the final hash of the content packet by incrementally hashing its fragments.

[10]However, no trust management architecture is defined in NDN.

To subvert DA we consider an adversary who is given a valid $\overline{CO}^N$ with signature $SIG(CO^n)$. The goal of the adversary is to send to some router R a sequence of fragments, $CF'^{N'}_{x_0=0,s}, CF'^{N'}_{x_1,s}, \ldots, CF'^{N'}_{x_k,s}$ $(x_{i+1} = x_i + s, 0 \leq i \leq k-1)$ corresponding to $\overline{CO'}^{N'} \neq \overline{CO}^N$ with $H(CO'^{x_k}) = H(CO^n)$. Recall that $CF^N_{v,s}$ embodies intermediate state IS_v of the hash function computed up to, but not including, v bits of the entire content.

First, consider fragments arriving in order. R receives $CF'^{N'}_{0,s}$, initializes HC with IV, and computes and retains $IS'^*_{x_1} = HC(IV, b_{0,s})$. Now, when R receives subsequent fragments $CF'^{N'}_{x_i,s}, i = 1, \ldots, k$, it compares the current (computed) $IS'^*_{x_1}$ with IS'_{x_1} contained in $CF'^{N'}_{x_1,s}$. If they match, R invokes $HC(\cdot)$ iteratively over each block of $CF'^{N'}_{x_i,s}$ using IS'_{x_i} as the starting intermediate state, and compares obtained $IS'^*_{x_{i+1}}$ with $IS'_{x_{i+1}}$ in $CF'^{N'}_{x_{i+1},s}$. This process is exactly the same as computing $H(\cdot)$ over entire $\overline{CO'}^{N'}$. If $\overline{CO'}^{N'} \neq \overline{CO}^N$, the adversary must have found a collision for $H(\cdot)$, which violates our collision-resistance assumption.

Now, assume that fragments arrive out-of-order. R receives $CF'^{N'}_{x_i,s}$. It can readily compute $IS'^*_{x_{i+1}}$ by invoking $HC(\cdot)$ over the blocks starting with IS'_{x_i}. R retains IS'_{x_i} as part of its state until $IS'^*_{x_i}$ is computed (using all previous fragments) and matched. If $IS'^*_{x_i}$ has been already computed, then R must have invoked $HC(\cdot)$ over the data in $CF'^{N'}_{x_{i-1},s}$ using $IS'_{x_{i-1}}$. If R arrives at the final hash output and its state contains *only* $IS'^*_{N'}$, then R has compared $IS'^*_{x_i}$ with IS'_{x_i} for $i = 1, \ldots, k$ such that each match was successful. In other words, all fragments have arrived and $i = 1, \ldots, k, IS'^*_{x_i} = IS'_{x_i}$. We observe that the set of equations that must be *satisfied* here is exactly the same as that in the in-order-arrival case. Therefore, the same argument applies.

VI. Implementation

In this section, we describe the implementation of FIGOA in CCNx version 0.8.2 [6] (latest version while writing this paper.) Our implementation performs fragmentation with cut-through switching, and intermediate reassembly. We strive to remain as consistent and compatible with the existing CCNx codebase, without changing the architecture or design except to support fragmentation. Due to lack of support for signature verification and key management in the implementation of the CCNx codebase, our implementation does not support signature verification of content objects processed by routers. However, it can naturally be extend to authentication should this feature becomes present in a future CCNx version.

CCNx is an open source content-centric networking stack developed by Palo Alto Research Center (PARC). The software suite comprises of a forwarder (*ccnd*) and client (*libccn*) implemented in the C programming language. A client for Java is also available. We refer to [7] for more detailed specifications regarding the CCNx protocol.

Our implementation only requires modification of the forwarder code. Our design limits fragmentation, reassembly, and cut-through switching for outgoing interfaces. Therefore, a forwarder must reassemble fragments prior to forwarding over the content to the application.

To implement fragmentation, we introduce a new type of NDN packet, `ContentFragment`. This packet is used for both initial fragmentation of content objects and re-fragmentation of content fragments. The structure of `ContentFragment` contains the following fields:

- `Name`: identical to content name without an additional implicit component digest.
- `ContentObjectSize`: size of the original content object before fragmentation takes place.
- `InternalState`: stores internal state of a SHA-256 computation up to `PayloadOffset` of the content.
- `PayloadOffset`: specifies where the fragmented data begins with respect to the unsigned content.
- `PayloadSize`: size of fragment payload, which is a multiple of 512-bits (the input block size for SHA-256) except for the last fragment.
- `ContentDigest`: contains the digest of the original content object. Appending this digest to the end of `Name`, forms the content's unique name. This field allows router to match fragments with interests (in PIT) containing the content digest as part of their names. Moreover, for routers not verifying content signature, this field must match the hash computed after reassembling the content.
- `Payload`: fragmented data of the content.

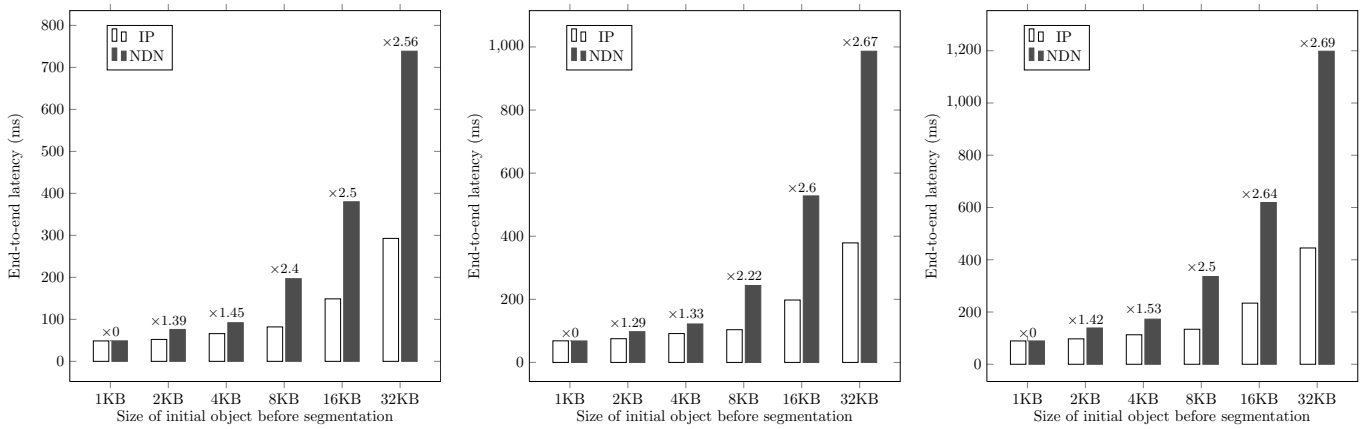
Once all fragments are received and the content is reassembled, the router caches it, if its integrity is verified.

The above format lends itself to natural re-fragmentation. If a `ContentFragment` requires further fragmentation only `InternalState`, `PayloadOffset`, `PayloadSize`, and `Payload` fields needs to be adjusted, reflecting the new fragments. This prevents nested fragments and simplifies reassembly. Thus, increasing routers performance and reducing consumers end-to-end latency.

To evaluate our implementation, we compare its performance to an unmodified version of CCNx 0.8.2. This version (similarly to the current NDN testbed) is running as an overlay network on top of TCP or UDP. When TCP is used to connect CCNx nodes, content larger than the negotiated MTU (at the connection setup) will be segmented by TCP. This reduces the chance of IP fragmentation to take place unless the MTU dropped to a smaller value at an intermediate router. On the other hand, when UDP is used, IP will be responsible of fragmenting and reassembling content. In this case, every CCNx node receives the whole content object from the UDP socket after reassembly is performed by IP. For the purpose of our experiments, we use UDP as a transport layer protocol to compare the performance of our FIGOA implementation to that of IP fragmentation.

VII. Evaluation

We employ a server equipped with 8-core Intel i7-3770 CPU at 3.40GHz and 16GB of memory. The server runs Ubuntu 12.10 and KVM hypervisor to run virtual machines.



(a) 3 nodes (b) 4 nodes (c) 5 nodes

Fig. 5. End-to-end latency of various sized content retrieval. IP represents the unmodified version of CCNx and NDN represents FIGOA. The values above the bars represent the difference between NDN and IP fragmentation (NDN / IP).

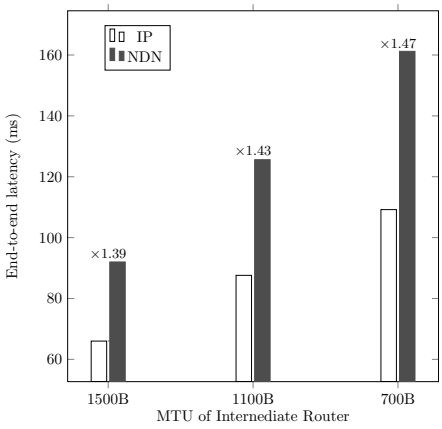


Fig. 6. End-to-end latency of various MTU-s at intermediate routers for content size 4KB. IP represents the unmodified version of CCNx and NDN represents FIGOA. Values above the bars represent the difference between NDN and IP fragmentation (NDN/IP).

We construct a testbed by provisioning virtual machines to act as CCNx nodes interconnected in the same LAN and NATed by the host server. Each node is connected to virtual ethernet interface at 100Mbps and MTU set to 1500 bytes.

Experiments are run on a 3, 4, and 5 nodes topology. The first hop acts as consumer sending interests with a specific content published by the last hop (the producer). For each topology, we run the experiments in which consumers request content with data size of 1, 2, 4, 8, 16, and 32 KB.

Results are shown in Figure 5 demonstrating the average consumer end-to-end latency measured from many repeated experiments.[11] For all settings, IP performs consistently better than our cut-through approach. The bottleneck of FIGOA is that routers need to perform additional processing to compute the hash of every fragment. Since all computations are currently performed in software, these results make sense. However, we believe that once NDN/CCN is deployed as a replacement of IP, all nodes (especially routers) will be capable of performing hash computation at the hardware level at a rate much faster than what is shown in Figure 5.

We run another 3 nodes experiment that involves refragmenting fragments. We measure the end-to-end latency at the consumer for different values of intermediate router's MTU (1500, 1100, and 700 bytes). Consumer pulls content of size 4KB. In the case of MTU value equals to 1500 bytes, the content is fragmented (at the producer) into 4 fragments, each, except the last one, is of size 1152 bytes[12] of effective fragment payload plus fragment header length. However, when MTU drops to 1100 bytes (at the intermediate router), the payload length of each outgoing becomes 768 bytes, leading to re-fragmentation of each fragment into 2 smaller ones. Similarly, each fragment is re-fragmented into 3 smaller fragments when MTU value drops to 700 bytes.

Results are shown in Figure 6. We can notice that when MTU value decreases, the end-to-end latency increases for a fixed content size. This is a logical conclusion due to the fact that smaller MTU leads to more processing imposed by re-fragmentation. Although re-fragmentation using FIGOA requires additional hash computations at each hop after where re-fragmentation occurs[13], the ratio of NDN end-to-end latency to IP end-to-end latency is not increasing dramatically. The reason is due to the fact that when fragmentation happens at the IP layer, reassembling the content is required at every hop before it is delivered by the UDP socket to NDN. Since this is not the case when FIGOA is implemented, IP reassembly adds more end-to-end latency that compensates the additional hash computation overhead imposed by FIGOA.[14]

VIII. Related Work

Related work falls into several categories discussed here.

[11] All nodes start with an empty cache at the beginning of every experiments.

[12] Multiple of SHA-256 block size which is 64 bytes.

[13] Recall that IP re-fragmentation does not required hash computation.

[14] Refer to Section IV-C for more details about the delay imposed by IP reassembly at each hope.

A. Secure Fragmentation

The first attempt to address security in IP fragmentation is [32] which tackled a specific problem of how to authenticate, in egress/ingress routers, fragmented IP packets. A source host is assumed to share a key with approrpriate router(s). Two techniques are proposed: The first one is delayed authentication (DA) where an authenticating router verifies a packet MAC incrementally from its fragments. Since fragments of the same packet might flow through different routers, to prevent reassembly of a corrupted packet at the destination, an authenticating router holds one small fragment "hostage" until authenticity of the entire packet authenticity is confirmed. The second scheme is an MTU probe mechanism that a source host can use to pre-segment a large packet into smaller authentic packets sized to the smallest MTU on the (current) path. Some extensions to [32] were later proposed in [25]: extended delayed authentication (EDA) requires fragments to always traverse the same path. [25] also provides a detailed comparison of several secure fragmentation techniques.

[24] presents a secure fragmentation scheme for Delay-Tolerant Networks (DTN) [8]. This scheme is referred to as "toilet-paper" approach to securing fragments. The basic idea is that, prior to bundling, data is checkpointed into fragments using cryptographic hash at specified intervals. Hashes are included in a bundle and authenticated with a signature. Given a fragment, the hash, and the signature a gateway can authorize if the fragmented data should be delivered over the link. A variation of the scheme allows for variable increments of authentic fragments, allowing routers greater flexibility to choose fragment size potentially saving valuable link resources.

Today, most networks employ IPSec [12] to provide network-level authentication in IP networks. IPSec is compatible with both IPv4 and IPv6. IPSec operates in transport and tunnel mode. Transport mode is used by two hosts which establish a security association (exchanging keys) to authenticate and encrypt IP payloads. Tunnel mode allows the creation of secure Virtual Private Networks (VPN) which comprise of IPSec-enabled gateways that share bilateral security associations. Gateways secure and authenticate whole IP datagrams, encapsulating them as payloads for IP datagrams destined for processing between IPsec-enabled gateways.

Regardless of mode chosen, fragmentation of packets between IPSec-enabled hosts (gateways) occurs at the IP layer. Since IPSec authenticated/encrypted packets have a destination address of another IPSec-capable host (or gateway), it must undergo packet-level scrutiny which requires reassembly of the packet. In essence hop-by-hop reassembly at IPSec adjacent hosts ensures that security is not subverted.

B. Fragmentation in ICN

CCNx currently serves as a reference implementation for NDN. It currently supports TCP/UDP tunnels to interconnect forwarders. Fragmentation is relegated to IP, limiting the maximum packet size to that of IP. Hop-by-hop reassembly allows routers to authenticate content (although not presently supported in the forwarder implementation), but at the increased cost of reassembly.

NDN Link Protocol (NDNLP) [30] attempts to amend this issue while allowing operation over both link-layer and virtual transports, such as Ethernet and TCP/UDP. Fragmentation occurs for both interest and content packets. It specifically features intermediate reassembly and therefore remains compatible with NDN security requirement, however uses an incompatible packet format to support cut-through fragmentation which can result in incurred delay. NDNLP also supports reliability layer.

The CCN-lite project [5] aims to provide a "level-0" forwarder for CCN. It is compatible with the CCNx protocol and provides a rudimentary implementation of the forwarder with simple data structures for PIT, FIB, and CS. Native fragmentation and reassembly is supported over Ethernet and TCP/UDP. Fragments are identified by sequence number without any addressing scheme on per-fragment basis implying cut-through fragmentation is not supported. The fragmentation scheme also provides optional support for reliable fragments transmission.

CONET [11] is a derivative ICN of CCNx. In [27] a transport scheme called ICN Transport Protocol (ICTP) is specified, which implements TCP native to ICN. Similar to TCP, ICTP segments data to avoid further fragmentation. In essence, this provides cut-through delivery of fragments. Akin to TCP, this doesn't prevent fragmentation from occurring at a lower-layer. Unlike our scheme, ICTP does not address content authentication at intermediate routers.

The NetInf project [9] is an emerging ICN architecture which supports location-independent named data objects (NDO) (similar to content objects in NDN/CCN). NDOs are signed and cacheable units. The project does not envisage a scheme for segmentation and relies on a "convergence layer" (CL) to synthesize necessary services for heterogeneous transports used to connect NetInf gateways. The CL is delegated the responsibility of fragmentation and reassembly of NDOs. With no native fragmentation and reassembly scheme available, NetInf appears to rely on intra-hop reassembly for verification of NDO authenticity.

IX. Conclusion

Secure fragmentation is an important issue in NDN. It is complicated by the rule that each content object must be signed by its producer. Thus far, fragmentation of content objects has been considered incompatible with NDN since it precludes authentication of individual fragments by routers. In this paper, we showed that secure and efficient content fragmentation is both possible and advantageous in NDN and similar architectures that involve signed content. We demonstrated a concrete technique (FIGOA) that facilitates efficient and secure content fragmentation in NDN, discussed its security features and assessed its performance. Finally, we described a prototype implementation and presented preliminary results.

References

[1] Named data networking (NDN) project 2012-2013 annual report. http://named-data.org. Retrieved Apr. 2014.

[2] NDN testbed. http://named-data.net/ndn-testbed/. Retrieved Apr. 2014.

[3] T. Anderson et al. The NEBULA future internet architecture. In *The Future Internet*, LNCS, pages 16–26. Springer, 2013.

[4] M. Bellare, R. Canetti, and H. H. Krawczyk. Keying hash functions for message authentication. In *CRYPTO*, 1996.

[5] CCN-Lite. http://ccn-lite.net/.

[6] CCNx. http://www.ccnx.org. Retrieved Feb. 2013.

[7] CCNx protocol reference. http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html. Retrieved Feb. 2013.

[8] V. Cerf et al. Delay-Tolerant Networking Architecture. RFC 4838 (Informational), 2007.

[9] C. Dannewitz et al. Network of information (netinf) - an information-centric networking architecture. *Computer Communications*, 2013.

[10] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), 1998.

[11] A. Detti, N. B. Melazzi, S. Salsano, and M. Pomposini. CONET: a content centric inter-networking architecture. In *SIGCOMM ICN*, 2011.

[12] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, 2011.

[13] C. Ghali, G. Tsudik, and E. Uzun. Elements of trust in named-data networking. *arXiv preprint arXiv:1402.3332*, 2014.

[14] M. Gritter and D. Cheriton. An architecture for content routing support in the internet. In *USENIX USITS*, 2001.

[15] D. Han et al. XIA: Efficient support for evolvable internetworking. In *USENIX NSDI*, 2012.

[16] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking named content. In *ACM CoNEXT*, 2009.

[17] C. Kent and J. Mogul. Fragmentation considered harmful. In *ACM SIGCOMM*, 1987.

[18] T. Koponen et al. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM*, 2007.

[19] K. Lahey. TCP Problems with Path MTU Discovery. RFC 2923 (Informational), 2000.

[20] M. Mathis and J. Heffner. Packetization Layer Path MTU Discovery. RFC 4821 (Proposed Standard), 2007.

[21] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone. Handbook of applied cryptography, August 2001.

[22] Named data networking project (NDN). http://named-data.org. Retrieved Feb. 2013.

[23] National science foundation (NSF) future of internet architecture (FIA) program. http://www.nets-fia.net/.

[24] C. Partridge. Authentication for fragments. In *ACM SIGCOMM HotNets-IV*, 2005.

[25] R. Popp. Implications of internet fragmentation and transit network authentication. In *Local Area Network Interconnection*. 1993.

[26] J. Postel. Internet Protocol. RFC 791, 1981.

[27] S. Salsano et al. Transport-layer issues in information centric networks. In *ACM SIGCOMM ICN*, 2012.

[28] I. Seskar, K. Nagaraja, S. Nelson, and D. Raychaudhuri. MobilityFirst future internet architecture project. In *ACM AINTEC*, 2011.

[29] Secure hash standard. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf, 2002. Federal Information Processing Standard 180-2.

[30] J. Shi and B. Zhang. NDNLP: a link protocol for NDN. Technical Report NDN-0006, University of Arizona, July 2012.

[31] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti. The PPP Multilink Protocol (MP). RFC 1990 (Draft Standard), 1996.

[32] G. Tsudik. Datagram authentication in internet gateways: Implications of fragmentation and dynamic routing. In *IEEE JSAC*, 1989.

[33] T. Wolf et al. Choice as a principle in network architecture. *SIGCOMM CCR*, 2012.

[34] G. Ziemba, P. Traina, and D. Reed. Security considerations for ip fragment filtering. RFC 1858, 1995.