Secure Fragmentation for Content-Centric Networks

Cesar Ghali
University of California, Irvine
cghali@uci.edu

Ashok Narayanan
Google
ashokn@ashokn.org

David Oran
Cisco Systems
oran@cisco.com

Gene Tsudik
Christopher A. Wood
University of California, Irvine
{gene.tsudik,woodc1}@uci.edu

***Abstract*—Content-Centric Networking (CCN) is a communication 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 is to perform hop-by-hop reassembly, which incurs prohibitive delays. In this paper, we show that secure and efficient content fragmentation is both possible and even advantageous in NDN and similar content-centric network 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 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 [25].

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) [15], [17], [19] is an approach to (inter-)networking designed for efficient, secure and scalable content distribution [17]. In CCN, named content – rather than named interfaces or hosts – are treated as first-class entities. Named-Data Networking (NDN) [24] is an example of CCN which stipulates that each piece of named content must be signed by its producer (also known as a publisher). This allows trust in content to be decoupled from trust in an entity (host or router) that might store and/or disseminate that content. These features facilitate in-network caching of content to optimize bandwidth use, reduce latency, and enable effective utilization of multiple network interfaces simultaneously.

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 [34], XIA [16], Mobility-First [30] and Nebula [2]. Regardless of which of these (if any) efforts eventually succeed, all of them need to address some of the same issues, such as naming/addressing, routing/forwarding and security/privacy.

One issue that straddles both networking (specifically, packet forwarding) and security is fragmentation of large packets. Originally present in IPv4 [28], intermediate fragmentation was deprecated in IPv6 [10] for a number of reasons, many of which were identified by Mogul in [18], e.g., router overhead and code complexity. Also, there have been attacks that took advantage of IPv4 reassembly [35]. Thus, eschewing fragmentation made sense for IPv6. However, the same might not hold for all network architectures. We show in Section IV that, for some very different types of networking, such as CCN/NDN, fragmentation is sometimes unavoidable and might even be beneficial.

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.

The intended contribution of this paper is two-fold: **First,** we discuss, in detail, numerous issues related 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 required.
- 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 network layer and allows cut-through switching in routers by avoiding hop-by-hop fragmentation and reassembly, thus lowering end-to-end delay. FIGOA employs a delayed authentication method similar to [33], which allows routers to efficiently verify signed content based on out-of-order arriving fragments. In the event that a given content ultimately fails either integrity or authenticity checks, reassembly and eventual delivery of corrupt content to consumers is prevented.

a) Organization: Section II provides some background on NDN. Then, 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 name string, 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`. Content signatures provide data origin authenticity [1].

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 for routing. A *Forwarding Information Base* (FIB) is a lookup table used to determine interfaces for forwarding incoming interests, and contains [*name_prefix*, *interface*] 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, the FIB can be populated manually or by a routing protocol.

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 (or certificate) 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 reflects a set of interfaces via which interests for this content have arrived. When an NDN router receives an interest, it searches 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 the reverse 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.

III. Fragmentation Synopsis

We define *fragmentation* as a means of splitting large packets into smaller packets, at the network layer, independent of the 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 naming 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 the late 1980s, network-layer fragmentation has been widely considered to be a headache and something to be avoided, based primarily on the IPv4 experience [18]. We briefly discuss pertinent IPv4 fragmentation concepts below.

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[2] 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., [21]. Also, the entire 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 IP packets [20]? 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 [18]. Having to fragment a packet

[1]Trust between a key and name prefix is an application concern [14].

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

takes a router off its critical path and can thus cause congestion; this can also be exploited as a denial-of-service attack. Nevertheless, 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 [35]. Reassembly by intermediate hops/routers is not viable in IP, since fragments of the same packet are not guaranteed to travel the same path.

IV. Fragmentation in NDN

NDN does not provide explicit support for cut-through fragmentation – only hop-by-hop fragmentation with reassembly [22]. However, the current NDN implementation, which runs as part of the NDN testbed [1], is implemented as an overlay on top of TCP or UDP. In this setup, fragmentation is handled by either (1) transport layer protocols, e.g., TCP segmentation, or by (2) network layer protocols, e.g., IP fragmentation. Moreover, if NDN is running directly over the link layer, protocols such as NDNLP [32] can be used to handle fragmentation. The main drawback of these methods is that they all require reassembly at every hop.

The rest of this section discusses certain factors motivating fragmentation in NDN. Using the terminology presented 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 [18]. 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 limitations 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 the following:

Claim. *If interest packets are fragmented and, possibly re-fragmented, hop-by-hop (intermediate) reassembly of fragmented interest packets is unavoidable.*

The intuition behind this claim is obvious: as described in Section II, each router that receives an interest must search its PIT and/or cache using the name carried in said 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 cannot pre-fragment an interest in order to avoid further fragmentation by intermediate routers, unless there is a well-defined and globally-accepted 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[3] 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 embedded within the content.

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

1) **Producer-based Fragmentation or Segmentation:** At 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. The segment size can be determined from an MTU discovery protocol (see Section IV-B2 below). This would ensure 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 a (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 itself contains a number of fields.

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

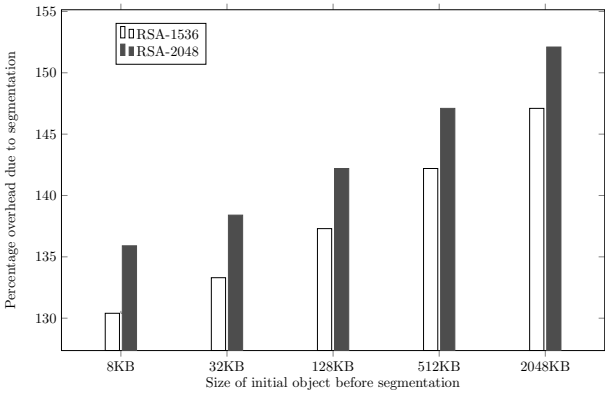


Fig. 2. Byte count overhead for small signed segments

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) and the 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 a 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 cannot avoided or ruled out, since NDN does not mandate a globally minimum 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 increased overhead due to the costs of signature generation by producers and verification by consumers and, optionally, by routers.

Another possibility is to introduce an MTU discovery method, whereby, an interest traveling towards requested content would carry a new field reflecting the smallest MTU (μMTU) discovered thus far on its path.[4] This is a viable and light-weight approach, particularly because 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 only stores content, (2) a router that caches content, or (3) a 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 involving content CO:

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

1) Consumer A sends interest int_A for CO to router R.
2) R receives and marks int_A with $\mu MTU = MTU_{(R \to A)}$ (MTU corresponding to the $R - A$ link). It then creates a PIT entry for int_A.
3) R forwards int_A to the 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 decreasing 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-fragmentation.

C. Considering Intermediate Reassembly

We now discuss intermediate reassembly, motivated by at least two factors. 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. 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.

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 reassemble them and verify the 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. 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 upon arrival.

We attempt to evaluate the benefits of cut-through fragment forwarding by considering a simple topology with a linear 8-hop path with 100 Mb/s links. Each link accumulates 10ms

TABLE I. LATENCY DUE TO PER-HOP CONTENT REASSEMBLY

	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

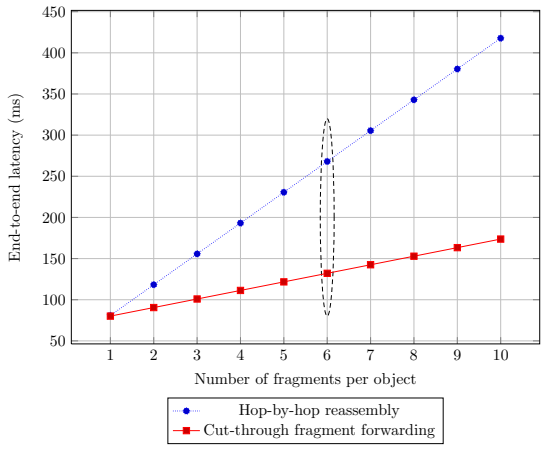


Fig. 3. Latency with varying fragment counts per object

of latency, ignoring intra-hop and queuing delays. 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). The *inter-fragment gap* is the time elapsed between consecutive fragments of an object, caused by fragments of other objects being interleaved. The *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. Using the aforementioned 8-hop topology, we 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 to 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 [32]) 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. Signature verification at stub AS (ingress) routers is more appropriate, e.g., because of policy dictating that no fraudulent content must reach consumers.[5] Also, stub AS egress routers might reassemble fragments and verify signatures if there is a policy disallowing any fraudulent content to exit the 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:

Challenge. *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?

V. SECURE FRAGMENTATION

This section describes a scheme called FIGOA: Fragmentation with Integrity Guarantees and Optional Authentication. It supports arbitrary intermediate fragmentation [18] of content while preserving security, and without requiring intermediate reassembly before forwarding all fragments.[6] FIGOA does not rely on in-order arrival of fragments, nor does it mandate any type of fragment caching strategy. Also, it 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 (DA).

A. Delayed Authentication

Delayed authentication was first introduced in [33]. 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]

The main differences between delayed authentication in its original IPv4 context [33] and our proposed use in NDN are as follows:

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

[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 cannot reassemble, either due to a time-out or another error.

- 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 traverse the same set of NDN routers. This results in much higher probability of ordered fragment delivery and faster time-outs in cases of lost or corrupted fragments.
- Not just ingress routers: delayed authentication was initially designed 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 MACs: delayed authentication was initially proposed for authenticating IP packet traffic flowing between two hosts (in two stub AS-s) that share a symmetric key. Message authentication codes were used for incremental fragment authentication. In NDN, signatures are used to ascertain content authenticity.

B. Hash Functions

The only realistic means of authenticating content in routers is by verifying signatures. This prompts the question: how does one reconcile delayed authentication (of fragments) with signatures?

We approach this issue by observing that a signature is computed over a fixed-size hash (digest) 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 [23]. 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-Damgård 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 $h(\cdot)$. 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 hash algorithm [31] operates on 512-bit blocks, maintains 256-bit internal state and yields a 256-bit hash.

In constructing FIGOA, we take advantage of internal state produced by the underlying compression function $h(\cdot)$. 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)*.

C. FIGOA Description

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

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

TABLE II. NOTATION

β	block size of $h(\cdot)$
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 $h(\cdot)$ 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.

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 $h(\cdot)$ on inputs of IV and $b_{0,v-1}$.[9] ($IS_v = h(IV, b_{0,v-1})$.) 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 ⟨CO⟩^N = b_{0,N-1}, aoMTU, IV, h(·)
2: Output: F
3: F := ∅, v = 0, IS_v = IV
4: s = aoMTU/β, k = N/s
5: for i = 0, i < k, i++ do
6:     CF^N_{v,s} := ⟨ v, b_{v,s}, IS_v ⟩
7:     F := F ∪ CF^N_{v,s}
8:     IS_v := h(IS_v, b_{v,s})
9:     v = v + s
10: end for
11: Output F
```

2) Fragment Re-fragmentation: This task 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.

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 to store $CF^N_{v,s}$. $IS^*_w = h(IS_v, b_{v,s})$ 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

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

in $\mathbb{B}$. $IS^*_w = h(IS_v, b_{v,s})$ 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 $CF^N_{v,s}$. $IS^*_w = h(IS_v, b_{v,s})$ is computed and stored.
4) $CF^N_{w,s}$ is in the buffer but $CF^N_{u,s}$ is not. $IS^*_w = h(IS_v, b_{v,s})$ 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 $CF^N_{v,s}$ $IS^*_w = h(IS_v, b_{v,s})$ 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 2. 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.)

Algorithm 2 Verify-Fragment

```
1: Input: received CF^N_{v,s}, associated PIT entry e, h(·)
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 = h(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 return
22:     end if
23: end if
24: CleanUp: FLUSH B and e
```

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 $h(\cdot)$ iteratively and computes $IS^*_s = h(IV, b_{0,s})$. Then, it forwards $CF^n_{0,s}$ out 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 the stored IS^*_s value 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 $h(\cdot)$ iteratively and computes $IS^*_N = h(IS_s, b_{s,s})$. At the end, before the content is cached and the last fragment is forwarded, R extracts $SIG(CO^n)$ from the received content, and computes a putative hash H' of the 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 $h(\cdot)$ 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 $h(\cdot)$ iteratively and computes $IS^*_s = h(IV, b_{0,s})$ 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) $IS_y = h(IS_x, b_{x,s'})$. This example aims to show that R can easily re-fragment already-fragmented content while preserving overall content integrity.

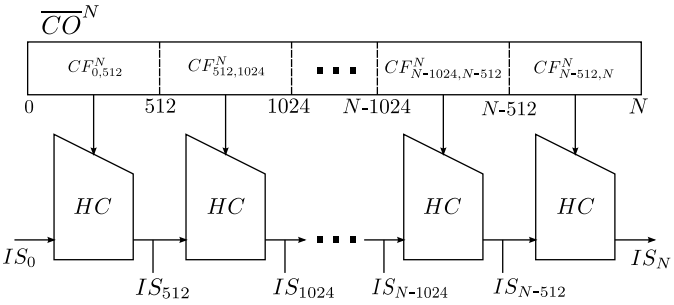


Fig. 4. Implementing Merkle-Damgård Construction to Generate Content Fragments

Figure 4 demonstrates how to use any hash function based on the Merkle-Damgård construction to generate content fragments. The hash function used in Figure 4 is SHA-256, and the length of input is discarded at the end of construction to simplify demonstration.

E. Content Authentication

Although trust and key management are out of the scope of this paper, we cannot 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 [14]. 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 header, or (2) enclosed with the content object itself, using the `KeyLocator` field. 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-Damgård construction with $h(\cdot)$ as the building block. If $h(\cdot)$ 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 [23] for information regarding Merkle-Damgård construction and hash-and-sign paradigm.

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. 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 $H(CO'^{x_k}) = H(CO^n)$. Assuming a secure DA scheme, the probability of constructing such a fragment sequence is negligible. This is formally proven in [13].

VI. Implementation

Our implementation performs fragmentation with cut-through switching and intermediate reassembly. It is consistent and compatible with the CCNx 0.8.2 code base, with no changes to the architecture except to support fragmentation. Modifications are confined to the CCNx forwarder code.

To implement fragmentation, we introduce a new type of NDN packet, `ContentFragment`(Figure 5 shows the format of this new type). It is used for both initial fragmentation of content objects and re-fragmentation of content fragments.

- `Name`: identical to content name
- `ContentObjectSize`: size of original content object before fragmentation
- `InternalState`: internal state of SHA-256 computation up to `PayloadOffset` of the content
- `PayloadOffset`: where fragmented data begins with respect to the unsigned content
- `PayloadSize`: size of fragment payload – a multiple of 512-bits, except for the last fragment
- `ContentDigest`: digest of the original content object. Appending this digest to `Name`, forms the content's unique name. This field allows routers to match fragments with interests (in PIT). For routers not verifying content signatures, this field must match the hash computed after reassembling the content.
- `Payload`: actual content fragment

Fig. 5. `ContentFragment` format specification.

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` need to be adjusted to reflect the new fragments. This prevents nested fragments and simplifies reassembly, which increases router performance and reduces end-to-end latency.

To evaluate the implementation, we compared its performance to an unmodified version of CCNx 0.8.2. This version runs as an overlay atop TCP or UDP (similar to the current NDN testbed). With TCP, content larger than the negotiated MTU (at connection setup) is segmented by TCP. This reduces the chance of IP fragmentation, unless the MTU is lower at an intermediate router. With UDP, IP is responsible for content fragmentation and reassembly. In this case, every CCNx node receives the whole content object from the UDP socket after reassembly is performed by IP. We use UDP as a transport layer protocol in the experiments.

VII. Evaluation

We employ a server equipped with 8-core Intel i7-3770 CPU at 3.40GHz and 16GB of RAM. It runs Ubuntu 12.10 and KVM hypervisor. We construct a testbed by provisioning virtual machines to act as CCNx nodes interconnected on the same LAN and NAT-ed by the host server. Each node is connected to a virtual Ethernet interface at 100Mbps, with MTU set to 1500 bytes.

Experiments are run on a 3-, 4-, and 5-node linear topology. The first hop acts as consumer sending interests for specific content published by the last hop – the producer. For each topology, the consumer requests content of size: 1, 2, 4, 8, 16, and 32 KB. We chose linear topology since content objects and fragments thereof always traverse the same path, in reverse, of preceding interests.

Results in Figure 6 demonstrate the average consumer end-to-end latency measured from many repeated experiments.[10] 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 computation is currently performed in software, these results make sense. However, we believe that if NDN/CCNx is deployed as a network layer, hash computation would be performed in hardware, with much better performance.

We run another 3-node experiment that involves re-fragmenting fragments. We measure end-to-end latency at the consumer for different values of intermediate router's MTU: 1,500, 1,100, and 700 bytes. The consumer requests content of size 4KB. In case of MTU=1,500, content is fragmented at the producer into 4 fragments, with each, except last, of 1,152 bytes[11] of payload plus fragment header length. However, when MTU drops to 1,100, payload length of each outgoing drops to 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 drops to 700.

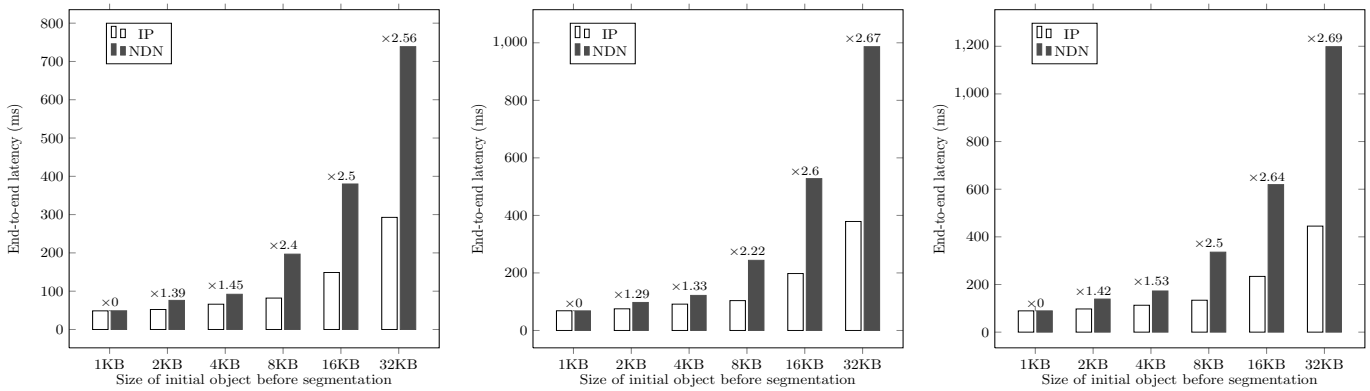
Results in Figure 7 indicate that, as MTU decreases, end-to-end latency increases for fixed content size. This makes sense, since smaller MTU leads to more processing due to re-fragmentation. Although re-fragmentation using FIGOA requires additional hash computations at each hop after the point where re-fragmentation occurs[12], the ratio of NDN end-to-end latency to IP end-to-end latency does not increase dramatically. This is because, when fragmentation occurs in IP, reassembling content is required at every hop before it is delivered by the UDP socket to NDN. Since the same does not hold with FIGOA, IP reassembly adds more end-to-end latency that compensates for additional hash computation overhead imposed by FIGOA.[13]

[10] All nodes start with an empty cache at the beginning of each experiment.

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

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

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



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

Fig. 6. End-to-end latency for retrieval of various sizes of content. IP represents the unmodified version of CCNx and NDN represents FIGOA. Values above bars represent the difference between NDN and IP fragmentation (NDN / IP).

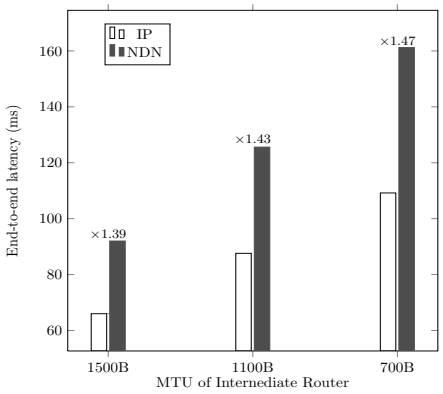


Fig. 7. End-to-end latency of various MTU-s at intermediate routers for content size 4KB. IP represents the unmodified version of CCNx and NDN represents FIGOA. Values above bars represent the difference between NDN and IP fragmentation (NDN/IP).

VIII. Related Work

The first attempt to address security in IP fragmentation is [33] which considered the specific problem of how to authenticate fragmented IP packets in egress and ingress routers of stub AS-s. A source host is assumed to share a key with the appropriate 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 [33] were later proposed in [27]: extended delayed authentication (EDA) requires fragments to traverse the same path. [27] also provides a detailed comparison of few secure fragmentation techniques. [26] presents a secure fragmentation scheme for Delay-Tolerant Networks (DTN) [8]. This scheme is referred to as "toilet-paper" approach to securing fragments, and was further enhanced in [3]. 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 whether fragmented data should be delivered over the link.

After bundle fragmentation occurs, the sender builds a hash tree and only signs the root node. Fragment verification requires knowledge of this signature and $\log(n)$ hashes (where n is the number of fragments). To authenticate all fragments, the verifier computes $n \log(n)$ hashes and verifies one signature; instead of n hashes and n signature verifications. However, these approaches are not applicable in NDN since they do not support in-network fragmentation. Moreover, as described earlier, authentication of all fragments in FIGOA requires the computation of only n hashes and one signature verification.

Currently, IPSec [12] is the most common approach for network-level authentication in IP networks. IPSec is compatible with both IPv4 and IPv6. It operates in transport and tunnel mode. In IPv4, regardless of the mode, packet fragmentation between IPSec-enabled hosts (gateways) can still occur. Since IPSec authenticated/encrypted packets have a destination address of another IPSec-capable host (or gateway), packet-level security requires reassembly.

Fragmentation in existing ICN architecture is far less pervasive. The current NDN implementation supports TCP/UDP tunnels to interconnect forwarders. Fragmentation is relegated to IP, and the packet size is limited to that of IP. Routers can authenticate content by reassembling all fragments. This, however, dramatically decreases the performance core network routers. NDNLP [32] is a link layer protocol for NDN that support fragmentation and reassembly of NDN packets (interests and contents) which sizes is greater than link MTU. Once an NDN packet is fragmented, all fragments carry a sequence number and their index in the corresponding original packet. However, since received content by routers needs to be verified and cached, NDNLP requires reassembly at each hop. Moreover, NDNLP uses an incompatible packet format to support cut-through fragmentation resulting in extra delay.

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. Therefore, cut-through frag-

mentation is not supported. The fragmentation scheme also provides optional support for reliable fragments transmission.

CONET [11] is a derivative of CCNx. Also, in [29], 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, but doesn't prevent lower-layer intermediate fragmentation.

The NetInf project [9] is an emerging ICN architecture that supports location-independent named data objects (NDO) – similar to content objects in NDN/CCNx. NDOs are signed and cacheable units. NetInf does not specify 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 responsible for the fragmentation and reassembly of NDOs. With no native fragmentation and reassembly scheme, NetInf appears to rely on per-hop reassembly scheme for 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] NDN testbed. http://named-data.net/ndn-testbed/. Retrieved Apr. 2014.

[2] T. Anderson et al. *The NEBULA future internet architecture*. Springer, 2013.

[3] N. Asokan, K. Kostiainen, P. Ginzboorg, J. Ott, and C. Luo. Towards securing disruption-tolerant networking. *Nokia Research Center, Tech. Rep. NRC-TR-2007-007*, 2007.

[4] M. Bellare, R. Canetti, and H. H. Krawczyk. Bellare, m and canetti, r and krawczyk, h. 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. RFC 4838: Delay-tolerant networking architecture, 2007.

[9] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of information (NetInf) - an information-centric networking architecture. *Computer Communications*, 2013.

[10] S. Deering and R. Hinden. RFC 2460: Internet protocol, version 6 (IPv6) specification, 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. RFC 6071: IP security (IPsec) and internet key exchange (IKE) document roadmap, 2011.

[13] C. Ghali, A. Narayanan, D. Oran, G. Tsudik, and C. A. Wood. Secure fragmentation for content-centric networks. *arXiv preprint arXiv:1405.2861*, 2014.

[14] C. Ghali, G. Tsudik, and E. Uzun. Network-layer trust in named-data networking. *ACM CCR*, 44(5):12–19, 2014.

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

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

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

[18] C. Kent and J. Mogul. *Fragmentation considered harmful*. Digital Equipment Corporation Western Research Laboratory [WRL], 1987.

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

[20] K. Lahey. RFC 2923: TCP problems with path MTU discovery, 2000.

[21] M. Mathis and J. Heffner. RFC 4821: Packetization layer path MTU discovery, 2007.

[22] N. Memo. Packet fragmentation in ndn: Why ndn uses hop-by-hop fragmentation.

[23] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

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

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

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

[27] R. L. Popp. Implications of internet fragmentation and transit network authentication. In *Local area network interconnection*. 1993.

[28] J. Postel. RFC 791: Internet protocol, 1981.

[29] S. Salsano, A. Detti, M. Cancellieri, M. Pomposini, and N. Blefari-Melazzi. Transport-layer issues in information centric networks. In *ACM SIGCOMM ICN*, 2012.

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

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

[32] J. Shi and B. Zhang. NDNLP: A link protocol for NDN. *The University of Arizona, Tucson, AZ, NDN Technical Report NDN-0006*, 2012.

[33] G. Tsudik. Datagram authentication in internet gateways: implications of fragmentation and dynamic routing. *IEEE J-SAC*, 7(4):499–504, 1989.

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

[35] G. Ziemba, P. Traina, and D. Reed. RFC 1858: Security considerations for IP fragment filtering, 1995.