CCN 1.0 Protocol Architecture

Marc Mosko, Ignacio Solis, Ersin Uzun
Palo Alto Research Center
{mmosko, isolis, euzun}@parc.com

ABSTRACT

In contrast to today's IP-based host-oriented Internet architecture, Information-Centric Networking (ICN) emphasizes content by making it directly addressable and routable. Content-Centric Networking (CCN) is an instance of ICN designed by PARC as a reference implementation that works in many environments from high-speed data centers to resource constrained sensors. By flexibly caching content within the network (in routers) and providing security at the network layer, CCN is well-suited for large-scale content distribution, the Internet of Things and for meeting the needs of increasingly mobile and bandwidth-hungry applications that dominate today's Internet. This paper describes the CCN 1.0 architecture and protocols, from message semantics to wire format to transmission rules.

1. INTRODUCTION

The functional goal of the Internet Protocol as conceived and created in the 1970s was to enable two machines, one comprising resources and the other desiring access to those resources, to have a conversation with each other [1]. The operating principle was to assign addresses to end-points, thereby enabling these end-points to locate and connect with one another. Since those early days, there have been fundamental changes in the way the Internet is used — from the proliferation of social networking services to viewing and sharing digital content such as videos, photographs, documents, etc. Instead of providing basic connectivity, the Internet has become largely a distribution network with massive amounts of video and web page content flowing from content providers to viewers. Internet users of today are demanding faster, more efficient, and more secure access to content without being concerned with where that content might be located.

To address the Internet's modern-day requirements with a better fitting model, PARC has created a new networking architecture called Content-Centric Networking (CCN), which operates by addressing and delivering Content Objects directly by name instead of merely addressing network end-points. In addition, the CCN security model explicitly secures individual Content Objects rather than securing the connection or "pipe". Named and secured content resides in distributed caches automatically populated on demand or selectively pre-populated. When requested by name, CCN delivers named content to the user from the nearest cache, thereby traversing fewer network hops, eliminating redundant requests, and consuming less resources overall.

CCN is based on a few simple key tenets. **Access content by name, not machine address**: Networking and communications are better served by using **names** to access information as opposed to using machine addresses. Packets have no source or destination addresses, the name identifies a request and response. **Secure the content, not the connection**: Security must be the foundation of any network architecture: Securing the data is more important (and useful) than securing the connections. Security does not rely on secure communication end-points or secure communication channels. **Add computing and memory into the network**: Computation and memory continue to decline in cost, making it feasible to add computation and memory to routers and, as a result, maintain state and object stores. CCN may cache content throughout the network essentially creating a fully-managed peer-to-peer network without losing control of content.

CCNx 1.0 is PARC's forthcoming reference implementation of the CCN 1.0 architecture. In the remainder of the paper, we describe the protocols used in CCNx 1.0 to deliver an efficient, flexible, ICN implementation.

2. ARCHITECTURE OVERVIEW

The CCN 1.0 architecture comprises a set of core protocols including basic messages and semantics, name formats, and packet formats. Higher level value-add protocols are layered upon the core protocols in order to achieve additional functionality such as large object chunking, cache control, versioning etc. The result of this approach is that the minimum set of core protocols and specifications are a subset of the original CCN vision 5 years ago [2]. The protocols from older implementations of CCN such as 0.8 [2] may be built on top of the new core, and in fact many of the higher-level protocols specified in the new layered approach find their roots in those original ideas. The new higher-level protocols, however, have a concrete unambiguous specification, with their concerns separated from other concerns. It

is expected that additional value-add protocols will be added by others in the research community.

A CCN 1.0 network is populated by Content Producers, Content Publishers, and Content Consumers with traffic supported by Forwarders and in-network caches called Content Stores. Each of these actors fulfills specific roles. A Content Producer creates user data. This could be an individual taking photos, a sensor producing a reading, or a media corporation creating movies. A Content Publisher converts user data into network objects with associated cryptographic identity, and publishes to the network using the CCN 1.0 protocols. A Content Consumer uses the CCN 1.0 protocols to retrieve publisher authenticated user data. Content is published and located by Publisher-given Names, not physical addresses. The core protocol uses two message types: Interest and Content Object. An Interest message is used to request data from the network. The data is returned in the form of a Content Object. An Interest message is forwarded along routing paths, leaving reverse-path state with each Forwarder. A Content Object is a chunk of authenticated user data sent back along the reverse path of the Interest, consuming the state left by the Interest. Section 3 describes these two messages in detail and specifies how they are processed by nodes in the network. CCN 1.0 uses a nested Type-Length-Value (TLV) format to encode all messages on the wire. This is a new general packet format based on a fixed header, a set of optional headers, and a CCN protocol message.

The two key network elements in CCN 1.0 are Forwarders and in-network caches called Content Stores. A Forwarder uses the Forwarding Information Base (FIB) table for routing and the Pending Interest Table (PIT) to record Interest state such that Content Objects may follow the reverse Interest path. These are fully described in section 4. Optional Forwarder Content Object caches called "Content Stores" are used to temporarily store Content Objects. Using a temporary Content Object store allows the Forwarder to repair lost packets if a downstream node re-requests the same Content Object or to reduce upstream traffic by caching popular content. Content Stores are described in section 4.

2.1 Names

A core feature of CCN 1.0 is that units of data, called Content Objects, have publisher-given Names. The Name serves two purposes: it is used to match against entries in router tables to forward the Interest message in its search for Content Objects, and it is used to identify the correct Content Object when it is found. In many respects, a CCN Name looks like a routable URI as shown in Name (1). A CCN name is an absolute URI without an authority (hostname). Following the nomenclature of URIs [3], we use the *scheme* "lci" and the *hier-part* is a *path-absolute* [3, Sec 3.3]. The absolute path is composed of a "/" followed by zero or more *segment*. CCN 1.0 follows the convention that each URI name segment has a label and a value, which leads to the scheme name Labeled Content Identifier (lci). The label identifies the purpose of the segment - it might be a general name segment used for routing or a specialized segment used to sequence numbers, timestamps, or content chunk numbers. Application-specific labels may be used. If no label is present, it is assumed to be a general name segment.

(1) `lci:/Name=foo/Name=bar/SerialNumber=7/ChunkNumber=30`

For example, in Name (1), the prefix `lci:/Name=foo/Name=bar` corresponds to a publisher-given Name. The segment `SerialNumber=7` indicates the revision of the document using a serial number. The segment `ChunkNumber=30` indicates that the user data was split into multiple chunks with this chunk vein the 30th in the series. These labels, shown in human-readable URI form, are encoded to TLV types.

It is not sufficient to leave the determination of label to application-specific conventions, as was done in the prior CCN protocols using optional command-markers (a non-UTF8 character as the first byte). In a networked system with multiple applications accessing resources generated by other applications, there needs to be a common understanding of conventions. For example, if one application uses a base 64 encoding of a frame number (e.g. base64(0xbdea)) and another uses "ver=" to represent a document version, there is an ambiguity because base64(0xbdea) is the string "ver=".

2.2 Large Objects

Publisher data such as a large text document, audio file, or even a 2GB or larger movie file, may be too large to fit in a single Content Object or MTU. A Content Object is limited to a maximum of 64 KB by the TLV wire format (see Sec. 3.4) and a typical MTU for Ethernet is only 1500 bytes. When this occurs, the system will automatically chunk the data into pieces in order to fit into the network Maximum Transmission Unit (MTU) size using a fragmentation process (see Sec 3.5). However, a more efficient way of handling this is through the use of Manifests.

A Manifest describes the collection of Content Objects that constitute one logical entity. A Manifest is a Content Object with a well-known payload format and a ContentType of "Manifest", rather than a normal ContentType of "Data". The Manifest provides meta information about the collection and enumerates the ordered, hash-based names of every constituent piece of the collection. Manifests are hierarchical to enable a single root Manifest to represent an unbounded length Content Object through chaining. The use of hierarchical Manifests means that only the root manifest needs a cryptographic signature as all other Content Objects and subsequent hierarchical Manifests are requested via a name and a ContentObjectHash, forming a trusted hash chain from the original signature. Manifests also allow the amortization of a single public key operation over a very large object thereby keeping the individual constituent Content Object sizes under the MTU and preventing fragmentation.

There are several alternatives to using manifest fragmentation to the minimum MTU. Hop-by-hop fragmentation, which has its own performance and buffering problems [4], may have a large buffer burden on each forwarder and exposes the forwarder to buffer starvation denial of service attacks. The secure fragmentation proposal of [4] requires two new behaviors at a forwarder: an Interest must be able to name the first fragment of a hash chain by a hash name and the forwarder must understand the hash chaining. Even with these two new behaviors, it is still possible that out-of-order fragments means that a forwarder will forward an incorrect fragment because it does not have all the links up to that point. The forwarder must also maintain the hash chain in memory if it wants to enforce the ContentObjectHash restriction. Cut-through fragmentation invalidates one of the key design principles of CCN 1.0: a forwarder cannot ensure that a forwarded Content Object matches an Interest's ContentObjectHash restriction; at best a solution like secure hash chains only begin dropping fragments once it detects a broken hash chain.

Using a manifest means that each network packet maintains its verifiability at the packet granularity. For non-realtime traffic, Manifest-based fragmentation is likely the best approach. It is not, however, always the best solution for some traffic types such as live broadcast or interactive multimedia. In those cases, using an end-to-end fragmentation scheme, such as in Sec 3.5 or [4] or simply requesting data by Name and KeyIdRestriction may be all that can be done.

As an example, assume we have 2GB of user data with a 64-byte Name in 4 Name segments. With TLV encoding, each Content Object has 100 bytes of overhead, leaving 1400 bytes for the payload in a typical Ethernet MTU. This data would therefore need to be broken into 1,428,572 Content Objects to fit into MTUs. The hierarchical Manifest enumerates the Content Objects by their 32 byte ContentObjectHashes. Each Manifest should also fit in an MTU, so a given Manifest Content Object has room for 40 ContentObjectHashes and 120 bytes of Manifest metadata. Our 2GB Content Object will need 35,715 Manifests. With a fanout of 40, we need a three-deep manifest hierarchy, which can handle up to 2,560,000 leaf Content Objects. In this example, no fragmentation is needed for the 1,464,287 Content Objects of 1500 bytes each. A total of of 2,196,530,500 bytes is transferred with 9.8% overhead.

Let us assume we used the fragmentation scheme in Sec 3.5 for the previous example. We could encode the 2GB into 65,536 byte Content Objects and then fragment them down to MTUs. This would require 32,518 of those large Content Objects. Each fragment needs a 24 byte fragment header, so the available MTU is 1376 in the first fragment and 1461 in subsequent fragments, resulting in a total of 44 fragments. The total transmission size is 2,146,188,000 bytes (7.3% overhead). Relative to the Manifest method, fragmentation uses 50,342,500 fewer bytes, so is 2.3% more byte efficient.

If there is any loss, the unit of retransmission is a 65,536 byte Content Object for fragmentation, whereas in the Manifest method the unit of retransmission is a 1500-byte Content Object. The break-even point is at $50,342,500/(65536 - 1500) = 786$ losses, or a 0.05% loss rate. If the loss rate is any higher, then the Manifest method is more byte efficient, assuming uniformly random drops and not counting the Interest overhead. This is a very low loss rate for Internet traffic, which is often over 1% (or much more) [5].

3. NETWORK MESSAGES

This section describes the two CCN 1.0 messages used to transfer user data. The CCN 1.0 protocols use a request message called an Interest to retrieve payload encapsulated in a Content Object response message.

3.1 Interest Message

An Interest carries a Name, an optional publisher identifier called the KeyId, and an optional inescapable identity called a ContentObjectHash, which is a secure cryptographic hash of a Content Object message.

Interest messages use a HopLimit field to prevent loops. We use a 1-byte hop limit, so at most 255 hops are allowed. When a Forwarder receives an Interest with a HopLimit, it decrements the value. It may not forward an Interest with a HopLimit of 0 out an external interface; it may only satisfy it from the built-in Content Store or forward it to a local application.

An Interest also carries a few directives that affect how it should be handled by the network. The Lifetime field specifies an upper limit on how long the Interest should persist in the network. The requester should not re-express the Interest more frequently than the Lifetime. The network, however, may store unanswered Interests for less than the specific lifetime.

If a protocol, API, or application desires an Interest to be answered only by an authoritative source, it should include a nonce in the Name; not to be confused with an Interest Nonce, a Name nonce is a name component with a random number used for end-to-end uniqueness. The authoritative source must also understand the use of nonces in Names. As long as the nonce is unique among requesters, there will be no cache hits for it and only an authority will reply.

The prior version of CCN included a flag called "Answer Origin Kind" used to choose between "fresh" or "stale" Content Objects. This flag is insufficient because there is no way to determine if a Content Object was supplied from a Content Store or from an authoritative source due to Interest aggregation. For example, assume an Interest follows the routing path and asks for data which is cached. If a second Interest with the same Name comes along and asks only for non-cached data, a Forwarder would also send it upstream. When a Content Object response comes, there would be no way to tell if the response was for the first, cached allowed, or the second, cache not allowed, Interest.

An Interest message sometimes carries state. For exam-

ple, a Name segment could encode data such as a session identifier. In this case an application-specific type known to the application as "SessionId" would be added as a Name segment to identify the session.

(2) `lci:/Name=foo/Name=bar/App:SessionId=0x5512334`

A Name segment could also identify a computation task to be done by the authoritative source, such as retrieving the current account balance. In this case, the Interest message is used to trigger behavior at a source and the computation result is returned in the Content Object.

(3) `lci:/Name=foo/Name=bar/App:SessionId=0x5512334/App:Task=AccountBalance`

An Interest may carry even more complex state. One could serialize a complex data structure into a Name segment. For example, a JSON data record could be encoded as a string in a Name segment, or a binary record could be included as-is in a binary path segment as in:

(4) `lci:/Name=foo/Name=bar/App:State={SessionId:"0x5512334",Task:"AccountBalance",...}`

We are experimenting with Interests that use a new payload field inside the Interest to separate out state from the name. In these experiments, the Interest Name carries a hash or nonce to indicate the embedded state. The payload is not stored in the Pending Interest Table and does not play a role in hop-by-hop matching of a Content Object to an Interest; it is only additional payload that a content producer would use to generate a dynamic Content Object. This is required so returning Content Objects correctly match the appropriate Interest with payload if there are multiple outstanding at a Forwarder. The embedded state – as it is not part of the Name – does not need to be kept at each hop to facilitate reverse path routing, so the state is smaller and the memory bandwidth is lower.

An Interest message typically has a Message Integrity Check (MIC) Validator appended to it. The Validator specifies the ValidationAlgorithm (e.g. CRC32C) and ValidationPayload (i.e. the checksum). This allows end-to-end validation of unauthenticated messages. An Interest message could have a stronger Validator, such as a Message Authentication Code (MAC) or Signature.

3.2 Content Object Message

A Content Object carries a Name and a Payload. Commonly, a Validator comes along with the Content Object. The Validator includes the Validation Algorithm used and the Validation Payload (e.g. a signature or HMAC) produced. The Validation Algorithm also carries a KeyId, which identifies the publisher authenticating the Content Object. If an Interest carries a KeyId in addition to a Name, it limits the universe of Content Objects that match the Name to those validated with that KeyId. The ContentObjectHash is a cryptographic hash of the entire Content Object. If used in an Interest in addition to a Name, it selects one specific Content Object of that Name.

The ContentObjectHash is the SHA-256 hash of the Content Object's wire format, from the opening Content Object tag to the end of the Content Object. It does not include the fixed or optional packet headers. The ContentObjectHash is not an explicit field in the packet, but must be calculated. This calculation provides an assurance that, for correctly behaving nodes within the network, if a user requests a Content Object by hash, the network delivers the correct packet.

A Content Object ay not include a Validator or it may use a Validator that is only an integrity check (e.g. CRC32C). Such a Content Object should only be retrieved by its self-certified name, the ContentObjectHash. This type of "bare" Content Object is appropriate when used in conjunction with a Manifest, where the Manifest is full Content Object with a KeyId and cryptographic signature and enumerates a set of bare Content Objects. The bare Content Objects, without the overhead of the KeyId or Signature, may consist of only a small name and payload and thus be pre-generated to a network MTU and not require fragmentation.

3.3 Matching

The protocols also include two operators: Equals and ComputeContentObjectHash. These two operators work on the three fields of an Interest message to match against a Content Object. One Interest message receives, at most, one Content Object message, so there is flow balance in messages. An Interest message may be considered a type of flow control, as a client may issue a set of Interest messages for different Content Objects. The client opens up a window for responses. Unlike most data networking protocols, the window is measured in messages, not bytes.

For a Content Object to match an Interest, the Name of the Content Object must be equal to the Name of the Interest. If the Interest has a KeyId restriction, then it must exactly equal the KeyId that validates a Content Object. This is not a cryptographic operation. If the Interest carries a ContentObjectHash restriction, then the Forwarder must compute the SHA-256 digest of the Content Object and match it for equality to the Interest's restriction.

3.4 Wire Format

CCN 1.0 uses a TypeLengthValue format to encode messages into a "TLV Packet". This is a new general packet format based on a fixed header, a set of optional headers in TLV format, and a CCN message containing nested TLV encoded content. The Type and Length fields are both exactly two bytes. Using a fixed type and length size avoids issues with aliases and is simple for high speed equipment to parse. In this section, we describe the TLV format, the fixed packet header, and the skeletons of Interest and Content Object messages. Aliases occur when there are multiple potential encodings for the same value. For example, if a TLV

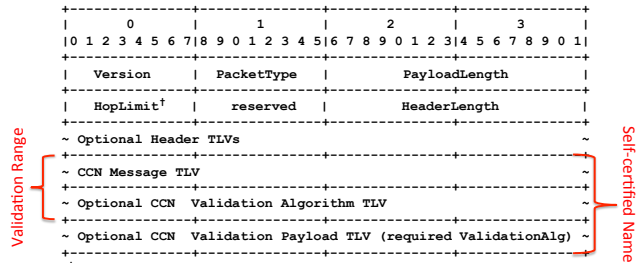

Figure 1: Overall packet format

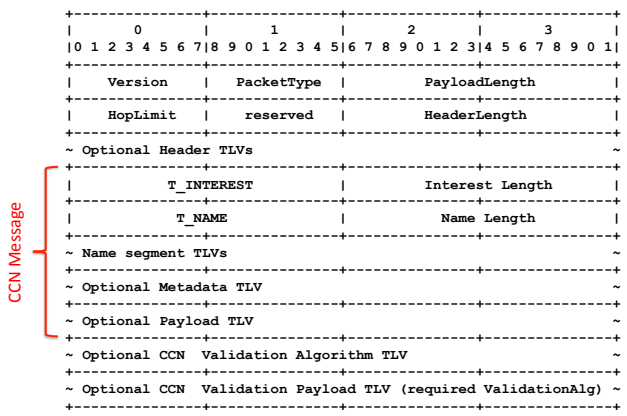

Figure 2: TLV Interest

type can be 1, 2, or 3 bytes, how does one encode a 0 value? If one may use 0x00, 0x0000, or 0x000000, should those be interpreted as the same value? If they are the same value, then this encoding will confound using hashes to check for equality. If they are not allowed, then a conforming parser must check for minimum length encoding.

Each CCN 1.0 packet, as shown in Fig. 1, begins with a fixed header followed by a set of optional headers in TLV format. The optional headers are not considered part of the CCN message, but rather communicate in-network state such as a DSCP class. The headers are followed by the CCN message - an Interest message, a Content Object message, or a future message type. The CCN message is followed by optional ValidationAlgorithm and ValidationPayload TLVs. The ValidationAlgorithm field specifies how to verify the CCN message. Examples include verification with a Message Integrity Check (MIC), a Message Authentication Code (MAC), or a cryptographic signature. The ValidationPayload field contains the validation output, such as the CRC32C code or the RSA signature.

```
Packet = FixedHeader
         *OptionalHeader (Interest / Content Object)
         [ValidationAlgorithm ValidationPayload]
FixedHeader = <Version, PacketType, PayloadLength,
               HopLimit, HeaderLength>
OptionalHeader = <a TLV container>
CCNMessage = Interest or ContentObject
```

Following the CCN Message with optional ValidationAlgorithm and ValidationPayload fields has several desirable features. In this scheme, the validation of a CCN message is modular and the same for all packet types. It is important that all messages, including Interests, can carry at least a MIC. In studies on TCP, UDP, and DNS messages [7, 8], researchers observe a packet error rate (PER) on the order of 10^{-5}, which is far greater than what one would expect from an Ethernet Mean Time To False Packet Acceptance (MTTFPA), which is usually at least as long as the age of the universe for properly functioning hardware and cabling. Therefore, we allow Interests and other messages to carry any one of a MIC, MAC, or Signature for transport-level validation.

The Fixed Header begins with the Version field which indicates the overall protocol version for the TLV encoding. The PacketType field indicates the type of packet that follows the optional headers, namely Interest or Content Object. HopLimit is used in Interest messages as a hop limit to prevent loops. PayloadLength is the number of octets of the Payload after the optional headers. HeaderLength is the number of bytes of header, including the fixed and all optional headers.

Fig. 2 shows an Interest message in TLV format. The value T_INTEREST is a protocol constant used to identify an Interest message. The value T_NAME is a protocol constant used to identify a CCN 1.0 Name, which is a nested TLV structure. The Metadata of an Interest carries the optional additional restrictions that control matching a Content Object. The KeyIdRestriction field limits the match to a specific publisher. The ContentObjectHashRestriction field will only return a Content Object whose computed SelfCertifiedName equals the ContentObjectHashRestriction. The SelfCertifiedName is the SHA-256 hash of the CCN Message, ValidationAlgorithm, and ValidationPayload.

An Interest may also have a payload. The payload carries state about the the Interest, but is not directly used to match a Content Object. The Interest Payload carries data that a publisher might use to generate a Content Object or information to help select among multiple Content Objects that would otherwise match. It is important that an Interest with a payload have a Name that is specific to that payload – otherwise it may be aggregated with other Interests with the same Name. We suggest that a Name segment be set to the hash of the Interest payload to differentiate payloads.

```
Interest = Name [Metadata] [Payload]
Metadata = [KeyIdRestriction] [InterestLifetime]
           [ContentObjectHashRestriction]
           *OtherMetadata
Payload = *OCTET
OtherMetadata = <by external specification>
```

Fig. 3 shows a TLV encoded Content Object. The CCN 1.0 Content Object no longer has a signature; that is moved to the ValidationAlgorithm and ValidationPayload. The field ContentObjectLength is the length of the Content Object

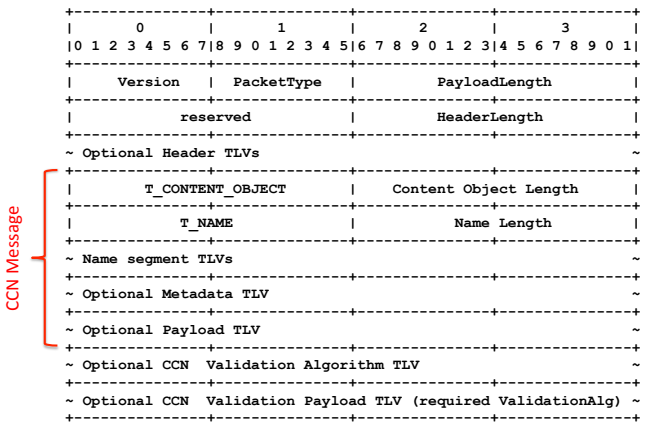

Figure 3: TLV Content Object

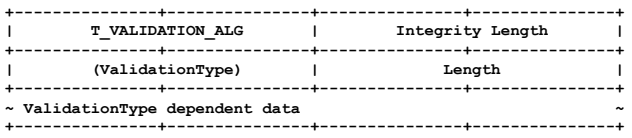

Figure 4: ValidationAlgorithm

through the payload; it does not include the signature block. The packet begins in the same way as an Interest. It starts with a fixed header, a set of optional headers in TLV format, the message container (T_CONTENT_OBJECT) and the length of the unsigned content object. The first field in the unsigned content object is the Name TLV. The remaining TLV containers are specific to a Content Object. The Metadata container encloses nested TLV structures for related protocols, such as a chunking or versioning scheme. For example, a chunking scheme could include a field here to indicate the last chunk number. The Contents TLV contains an opaque value that is the payload of the Content Object.

```
ContentObject = Name [Metadata] [Payload]
Metadata = [ContentType] [PublishTime]
           [CreationTime] [ExpirationTime]
           *OtherMetadata
Payload = *OCTET
OtherMetadata = <by external specification>
```

Fig. 4 shows the ValidationAlgorithm section. The type (ValidationType) defines the type of validator. Each MIC, MAC, or signature has its own TLV type, which in turn has its own data. The list here is an example of suggested types. We plan to add more types, such as Elliptical Curve. The use of CRC32, as in Ethernet, is not suggested because it duplicates the link layer Frame Check Sequence; we would recommend using a different MIC though the analysis of what would make the best compliment is beyond the scope of the present work. Fig. 5 shows examples for a CRC32 and an HMAC with SHA-256 hash. The CRC32 MIC requires no additional specification, so it has a 0 length. The HMAC-SHA256 MAC requires a KeyId parameter so the receiver

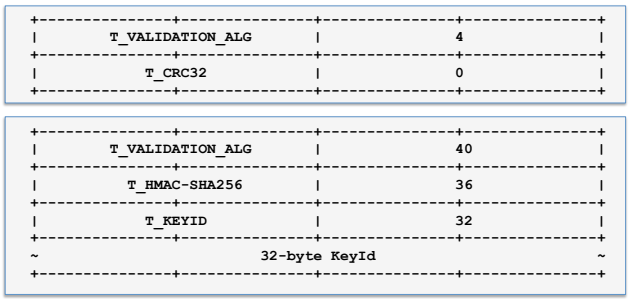

Figure 5: ValidationAlgorithm Examples

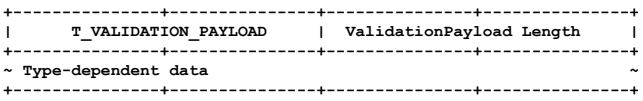

Figure 6: Validation Payload

knows which shared key to use.

```
VerificationType = CRC32 / CRC32C / RFC793 /
                   HMAC-SHA256 / RSA-SHA256
CRC32  = <empty>
CRC32C = <empty>
RFC793 = <empty>
HMAC-SHA256 = KeyId
RSA-SHA256 = KeyId [KeyLocator]
KeyLocator = PublicKey / Certificate / KeyName
PublicKey = <der encoded public key>
Certificate = <der encoded X.509 certificate>
KeyName = LinkMessage
```

3.5 Fragmentation

A CCN 1.0 message may be larger than a specific networking technology allows. The limitation, known as the Maximum Transmission Unit (MTU), typically varies from 1280 octets (for some IPv6 tunnels) to 1500 octets (Ethernet) to 9000 octets (Ethernet Jumbo frames), with 1500 octets being the most common. Both Interest and Content Object messages, however, need to accommodate potentially large Names and have key and signing overhead. Additionally, a Content Object may be as large as 64 KB. Therefore, CCN messages often must be fragmented over specific network media.

In order to avoid the complexity and buffer space requirements of the hop-by-hop option, we elected to use end-to-end fragmentation instead. End-to-end fragmentation is based on the principle that intermediate systems should not have to fragment packets. To achieve this, an Interest is always fragmented to the minimum MTU required by the path and the forward path's MTU is recorded in the Interest, so that a system sending back a Content Object will know what the required fragment size needs to be. An intermediate system's Content Store may store only pre-fragmented objects, responding only if an Interest's MTU is no smaller than that

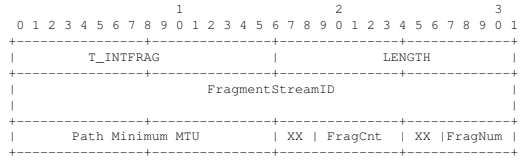

Figure 7: InterestFragmentHeader

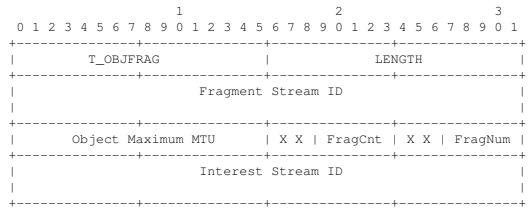

Figure 8: Content Object Fragment Header

stored, or it may re-assembly complete objects and fragment on demand.

Because the Interest's path serves as the path discovery mechanism for Content Objects, all Interests must carry an InterestFragmentationHeader so the reverse path MTU is known by the node responding with a Content Object. The minimum MTU required is 1280 octets. Any system implementing a physical layer with a smaller MTU must implement link fragmentation, for example using a PPP layer over the small MTU network.

Systems must create fragment streams in the most compressed packing. That is, all fragments except the last must be fragmented to the same size. This means that all Interests are fragmented to 1280 byte fragments, except for the last fragment. A Content Object may be fragmented to any size that is no more than the path MTU discovered, but all fragments except the last must be the same size. This ensures that any two fragment streams of the same Content Object with the same MTU have the same representation.

When an end system creates a fragment stream, it generates a random 64-bit number for the FragmentStreamID. This number identifies a contiguous stream of fragments. A system at the other end uses the FragmentStreamID for reassembly. An intermediate system uses the FragmentStreamID of a Content Object to ensure that only one stream of Content Object fragments follows a reverse PIT entry. If the Maximum Path MTU of a Content Object fragment is larger than the supported MTU on an egress interface, the fragment stream should be dropped on that interface, even if some fragments fit within the MTU.

At an end system re-assembling fragments, reassembly should timeout if all fragments are not received within a system-dependent timeout. If the re-assembly of an Interest times out before the PIT entry, the PIT entry on the local system should be removed to allow a new fragment stream to arrive. If the re-assembly of a Content Object times out, the received fragments' PIT state should be cleared.

As an Interest goes through the FIBs, it records the minimum path MTU based on the egress interface's MTU. A Content Object sent in response must be fragmented to less than or equal to the minimum path MTU. A Forwarder may choose to put 1280 in the Minimum Path MTU field even if it supports larger MTUs.

Interests follow the FIB and all fragments of an Interest (i.e. with the same fragment id) should follow the same FIB choices. If at a later time a similar Interest arrives with a smaller minimum path MTU, it should be forwarded even though it is similar, to ensure that a returned Content Object is fragmented to a size that satisfies the Interest's path.

This system of fragmentation is designed to not penalize large Content Objects with large names. If the Name has to be repeated in each fragment, then large Content Objects would have to bear extremely high overheads for the Name to fit in a media MTU. In this system the fragments can arrive in any order and be cut-through forwarded because Content Object fragments are routed based on the 64-bit FragmentStreamID. Interests, however, are penalized if they are larger than one MTU, because they cannot be routed against the FIB until the Name arrives, so fragment 0 must arrive before any forwarding can happen.

A disadvantage of end-to-end fragmentation is that one cannot verify a ContentObjectHash at each hop. To calculate the ContentObjectHash, one needs to run the hash over all the bytes of the Content Object in order, requiring reassembly or in-order delivery. Therefore, a Content Object fragment stream should be represented as a single MTU Manifest that enumerates the ContentObjectHash of each constituent fragment. This preserves the network guarantees around delivering the correct content when asked for by ContentObjectHash without using an insecure fragmentation protocol.

4. FORWARDING

A Content Consumer issues an Interest request for data over the network. The Interest is transmitted through a set of Forwarders until the Content Object is found or the Interest's Lifetime elapses.

A Forwarder consists of three notional tables: Forwarding Information Base (FIB), Pending Interest Table (PIT), and Content Store (CS). An actual implementation may use a different information organization as long as the external behavior is the same. The FIB is the routing table. It is a longest-matching-prefix name table populated by a routing protocol or with static routes. The Forwarder records Interest state in the PIT such that Content Objects that satisfy the Interest may follow the reverse Interest path back to the requester. It also implements Interest aggregation, so that multiple similar Interests do not get forwarded upstream. The Content Store is an optional in-network Content Object cache that may be used to satisfy Interest messages with fewer hops. Depending on its role in the network, a Forwarder may also run additional protocols such as service

and content discovery protocols.

Definition 1. (Satisfy Interest) A Content Object satisfies an Interest if and only if (a) the Content Object name exactly matches the Interest name, and (b) the Validation Algorithm KeyId of the Content Object exactly equals the Interest KeyId restriction, if given, and (c) the ContentObjectHash equals the Interest Content Object hash restriction, if given.

As a Content Object moves through the "fast path" of the network, it matches pending Interests at each node according to the rules in Definition 1. Only end systems verify cryptographic signatures, which decreases the computational load on each node and therefore increases throughput. However, each hop may need to compute the ContentObjectHash, if the pending Interest includes an ContentObjectHash restriction.

The Content Store has more restrictive matching rules in order to prevent persistent failures resulting from incorrect content getting into the persistence store.

Definition 2. (Content Store) If an Interest has a KeyId restriction, then the Content Store must verify a cached object's signature before returning it to the requester. This is a significantly more restrictive step than in the fast path. This means that either the Interest must carry the desired public key or the Content Object itself must carry its own public key. If an Interest has a ContentObjectHash restriction, then the Content Store must verify a Content Object's hash prior to returning it to the requester.

The CCN 1.0 node behavior is as follows. The CCN packet (Fig. 1) arrives either in direct layer 2 encapsulation – such as an Ethernet frame with a specific CCN ethertype – or via a layer 3 tunnel. The forwarder determines which processing rules to use based on the Fixed Header PacketType field (Fig. 1).

CCN distinguishes between "local" and "remote" next hops. A local next hop is a directly attached application running locally on the system. A remote next hop is not local to the current system. These conventions along with how we use HopLimit result in expected behavior. If a local application sends an Interest with a HopLimit 0, that Interest will only go to other applications on the system. If it sends an Interest with a HopLimit 1, it will go to local applications plus the 1-hop neighbors of the system. If a system receives a remote Interest with a HopLimit 1, it will be decremented to 0 and then only forwarded to local applications.

A Forwarder uses the following rules to forward Interests and Content Objects appropriately:

1. Receive Interest

(a) If the Interest came in a "remote" previous hop:
   i. if the HopLimit value is "0", drop the Interest. That is an invalid value for an Interest that arrives over the network.
   ii. decrement the HopLimit.

(b) If the Interest has a HopLimit "0" it may only be forwarded to "local" applications.

(c) If the Interest has a HopLimit "1" or more it may be forwarded to "local" applications or "remote" next hops.

(d) Match the Interest to the Content Store, if matched return that object to the previous hop, then discard the Interest.

(e) Match the Interest to the PIT using the most restrictive matching (i.e. by (Name, KeyId) or ContentObjectHash).
   i. If a match is found, aggregate the current Interest with the previous one. This means to add the previous hop of the current Interest to the previous hop set of the existing PIT entry. Each previous hop carries its own expiration time based on the Interest lifetime. If the new Interest extends the Lifetime of the PIT entry, then the Forwarder should re-forward the Interest after the current Lifetime expires, with the Lifetime reduced to the remainder. The second Interest must have a MinimumPathMTU no more than the existing PIT entry.
   ii. If there is no match, add the Interest to the PIT. The Interest is now said to be "pending" at this Forwarder.

(f) If the Interest was not aggregated, look up the Interest name in the FIB
   i. Retrieve the set of all potential next hops, as populated by routing protocols or statically.
   ii. Remove the Ingress previous hop.
   iii. Apply the forwarding Strategy, such as "best route" or "weighted round robin" or "all".
   iv. If the result is an empty set, remove the PIT entry (or aggregation). We do not keep a PIT entry if we cannot forward or satisfy the Interest.
   v. Forward the Interest to the resulting set of next hops.

(g) The PIT lifetime is based on the maximum previous hop lifetime. If a PIT entry expires, it is removed. Based on implementation, this may be a lazy process.

2. Receive Content Object

(a) A node looks in the PIT to find all PIT entries satisfied by the Content Object (see Definition 1). If one or more of the PIT entries has a ContentObjectHash restriction, the Forwarder must verify that the ContentObjectHash of the received Content Object matches the ContentObjectHash of the PIT entry, otherwise it does not satisfy those Interests. If the Forwarder finds one or more PIT entries satisfied by the Content Object, it forwards the Content Object to the previous hops denoted in the PIT entry's ingress set removes the satisfied PIT entries. The forwarder also removes expired previous hops in the Ingress Set.

(b) A Content Object that does not match any PIT entries should be dropped.

(c) A Forwarder should verify that a Content Object arrived from an expected previous hop. If the previous hop is not in the FIB forwarding path for the Interest, it is likely the Content Object is an off-path injection attack, and should be dropped.

(d) If the node has a Content Store, it may save the Content Object in the Content Store.

To forward messages fragmented as per Sec 3.5, the following rules need to be used in addition to the above. A system may aggregate two Interests if they have the same constraints *and the second Interest's MTU is greater than or equal to the first*. That is, if a second Interest arrives and has a smaller Path Minimum MTU, then it must not be aggregated because a responding Content Object may be larger than its allowable MTU. The rules given below include the CCN Fragmentation protocol described in Sec. 3.5.

1. Originate Interest

(a) All Interets carry a fragmentation header, even if the Interest itself is not fragmented. The Path Minimum MTU must be set to the maximum supported value (i.e. 64KB).

(b) Interests must be fragmented by the originating system to the minimum MTU (1280). The name must fit in the first fragment.

2. Transmit Interest

(a) When an Interest goes out a specific interface, the PathMinimumMTU field must be set to the minimum of the current value or the interface's MTU.

3. Receive Interest

(a) If the Interest is part of a fragmented Interest A system must wait to receive fragment 0 in order to get the routable information in it. If other fragments arrive out of order, a system may buffer them or discard them. All forwarding requires fragment 0 so out of order arrival is limited to the link from the previous CCN hop.

(b) To match against the Content Store, a fragmented Interest must have all matching information in fragment 0; this includes the complete Name and complete Metadata field. If these are not all completely in fragment 0, it will not be matched to the Content Store.

(c) Match the Interest to the PIT using the most restrictive matching (i.e. by KeyId or ContentObjectHash). If the Interest is a later fragment, beyond fragment 0, then the lookup is done based on the Fragment Stream ID.

i. If the existing Interest's MinimumPathMTU is greater than the current Interest's MinimumPathMTU, then the two are not aggregated, and the current Interest is treated as a separate PIT entry and forwarded.

ii. If the same Fragment Stream ID exists, match the current fragment to the BitMap and if not set, set the BitMap and forward the Interest fragment. Otherwise, record the new Fragment Stream ID.

(d) If the fragmented Interest was not aggregated, then it must be forwarded along the same path(s) that the earlier fragments took.

4. Originate Content Object

(a) A process that receives an Interest and satisfies it with a Content Object must ensure the Content Object MTU is no larger than the Interest's PathMinimumMTU. It must fragment or re-fragment the Content Object for that MTU, otherwise it must not respond.

5. Receive Content Object

(a) A node receiving a fragmented Content Object matches it to PIT entries based on the Interest Fragment Stream ID carried in the Content Object Fragment Header.

i. If no Content Object Fragment Stream ID is recorded in the PIT, then note the current fragment stream's ID in the PIT and forward it.

ii. If the PIT entry already has a Content Object Fragment Stream ID, then drop the current packet if the Fragment Stream ID does not match.

iii. The PIT must track which fragments of the Content Object have been forwarded along the reverse path, in the same way it tracked Interests along the forward path.

(b) If there is a Path MTU recorded in the PIT entry equal to or greater than the Content Object's Maximum Path MTU, then the Content Object fragment stream may be used. Forward the Content Object along those reverse paths and remove them from the PIT entry

5. CONCLUSION

The CCN 1.0 protocols deliver an efficient, flexible Information Centric Networking implementation. This paper described the new architecture from wire format through message semantics. In the new architecture, a Content Object's Name must equal an Interest's Name. If the Interest carries a KeyIdRestriction or a ContentObjectHashRestriction, then the Content Object's ValidationAlgorithm's KeyId must match the Interest's KeyIdRestriction and the computed ContentObjectHash must match the ContentObjectHashRestriction. CCN 1.0 does not use nonces for loop prevention or multi path detection. It uses a HopLimit field in the packet header to prevent loops.

Fragmentation, or conversely aggregation, is still an important area to consider. We propose using a Manifest to enumerate minimum MTU Content Objects as the main method of transport. This begs the question of how to aggregate on larger MTU links. We believe that aggregation will result in

safer operations because it preservers all integrity and validation checks for each Content Object atom. We have also proposed an end-to-end fragmentation scheme that does not preserve the self-certified name check property at each hop, but does allow fast operation when this is not possible, such as live streams where Interests carry only a KeyIdRestriction.

6. REFERENCES

[1] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864.

[2] Van Jacobson, D. K. Smetters, James D. Thornton, Michael Plass, Nick Briggs, and Rebecca Braynard. Networking Named Content. In *CoNext*, 2009.

[3] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD), January 2005. Updated by RFC 6874.

[4] Ashok Narayana, Nathan Naveen, Dave Oran, and Gene Tsudik. Secure fragmentation for content-centric networks. In *CCNxCon 2013 at PARC*, Sep 2013.

[5] Hung X. Nguyen and Matthew Roughan. Rigorous statistical analysis of internet loss measurements. *IEEE/ACM Trans. Netw.*, 21(3):734–745, June 2013.

[6] Urs Hengartner, Sue Moon, Richard Mortier, and Christophe Diot. Detection and analysis of routing loops in packet traces. In *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, pages 107–112, 2002.

[7] Duane Wessels. Observations on checksum errors in DNS UDP messages. In *Domain Name System Operations Analysis and Research DNS-OARC*.

[8] Jonathan Stone and Craig Partridge. When the CRC and TCP checksum disagree. In *SIGCOMM '00*, pages 309–319, 2000.