Internet Research Task Force (IRTF)
Request for Comments: 8569
Category: Experimental
ISSN: 2070-1721

M. Mosko
PARC, Inc.
I. Solis
LinkedIn
C. Wood
University of California Irvine

July 2019

Content-Centric Networking (CCNx) Semantics

Abstract

This document describes the core concepts of the Content-Centric Networking (CCNx) architecture and presents a network protocol based on two messages: Interests and Content Objects. It specifies the set of mandatory and optional fields within those messages and describes their behavior and interpretation. This architecture and protocol specification is independent of a specific wire encoding.

The protocol also uses a control message called an Interest Return, whereby one system can return an Interest message to the previous hop due to an error condition. This indicates to the previous hop that the current system will not respond to the Interest.

This document is a product of the Information-Centric Networking Research Group (ICNRG). The document received wide review among ICNRG participants. Two full implementations are in active use and have informed the technical maturity of the protocol specification.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Research Task Force (IRTF). The IRTF publishes the results of Internet-related research and development activities. These results might not be suitable for deployment. This RFC represents the consensus of the Information-Centric Networking Research Group of the Internet Research Task Force (IRTF). Documents approved for publication by the IRSG are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8569.

Copyright Notice

Table of Contents

1. Introduction

This document describes the principles of the CCNx architecture. It describes a network protocol that uses a hierarchical name to forward requests and to match responses to requests. It does not use endpoint addresses, such as Internet Protocol. Restrictions in a request can limit the response by the public key of the response's signer or the cryptographic hash of the response. Every CCNx forwarder along the path does the name matching and restriction checking. The CCNx protocol fits within the broader framework of Information-Centric Networking (ICN) protocols [RFC7927]. This document concerns the semantics of the protocol and is not dependent on a specific wire encoding. The CCNx Messages [RFC8609] document describes a type-length-value (TLV) wire-protocol encoding. This section introduces the main concepts of CCNx, which are further elaborated in the remainder of the document.

The CCNx protocol derives from the early ICN work by Jacobson, et al. [nnc]. Jacobson's version of CCNx is known as the 0.x version ("CCNx 0.x"), and the present work is known as the 1.0 version ("CCNx 1.0"). There are two active implementations of CCNx 1.0. The most complete implementation is Community ICN (CICN) [cicn], a Linux Foundation project hosted at fd.io. Another active implementation is CCN-lite [ccn-lite], with support for Internet of Things (IoT) systems and the RIOT operating system. CCNx 0.x formed the basis of the Named Data Networking (NDN) [ndn] university project.

The current CCNx 1.0 specification diverges from CCNx 0.x in a few significant areas. The most pronounced behavioral difference between CCNx 0.x and CCNx 1.0 is that CCNx 1.0 has a simpler response processing behavior. In both versions, a forwarder uses a hierarchical longest prefix match of a request name against the forwarding information base (FIB) to send the request through the network to a system that can issue a response. A forwarder must then match a response's name to a request's name to determine the reverse path and deliver the response to the requester. In CCNx 0.x, the Interest name may be a hierarchical prefix of the response name, which allows a form of Layer 3 (L3) content discovery. In CCNx 1.0, a response's name must exactly equal a request's name. Content discovery is performed by a higher-layer protocol.

The selector protocol "CCNx Selectors" [selectors] is an example of using a higher-layer protocol on top of the CCNx 1.0 L3 to perform content discovery. The selector protocol uses a method similar to the original CCNx 0.x techniques without requiring partial name matching of a response to a request in the forwarder.

This document represents the consensus of the Information-Centric Networking Research Group (ICNRG). It is the first ICN protocol from the RG, created from the early CCNx protocol [nnc] with significant revision and input from the ICN community and RG members. This document has received critical reading by several members of the ICN community and the RG. The authors and RG chairs approve of the contents. This document is sponsored under the IRTF, is not issued by the IETF, and is not an IETF standard. This is an experimental protocol and may not be suitable for any specific application. The specification may change in the future.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Architecture

We describe the architecture of the network in which CCNx operates and introduce certain terminology from [terminology]. The detailed behavior of each component and message grammar is in Section 2.

A producer (also called a "publisher") is an endpoint that encapsulates content in Content Objects for transport in the CCNx network. A producer has a public/private keypair and signs (directly or indirectly) the Content Objects. Usually, the producer’s KeyId (hash of the public key) is well known or may be derived from the producer’s namespace via standard means.

A producer operates within one or more namespaces. A namespace is a name prefix that is represented in the forwarding information base (FIB). This allows a request to reach the producer and fetch a response (if one exists).

The FIB is a table that tells a forwarder where to send a request. It may point to a local application, a local cache or Content Store, or to a remote system. If there is no matching entry in the FIB, a forwarder cannot process a request. The detailed rules on name matching to the FIB are given in Section 2.4.4. An endpoint has a FIB, though it may be a simple default route. An intermediate system (i.e., a router) typically has a much larger FIB. A core CCNx forwarder, for example, would know all the global routes.

A consumer is an endpoint that requests a name. It is beyond the scope of this document to describe how a consumer learns of a name or publisher KeyId; higher-layer protocols built on top of CCNx handle those tasks, such as search engines or lookup services or well-known names. The consumer constructs a request, called an Interest, and forwards it via the endpoint's FIB. The consumer should get back either a response (called a Content Object) that matches the Interest or a control message (called an Interest Return) that indicates the network cannot handle the request.

There are three ways to detect errors in Interest handling. An Interest Return is a network control message that indicates a low-level error like "no route" or "out of resources". If an Interest arrives at a producer, but the producer does not have the requested content, the producer should send an application-specific error message (e.g., a "not found" message). Finally, a consumer may not receive anything; in which case, it should timeout and, depending on the application, retry the request or return an error to the application.

1.3. Protocol Overview

The goal of CCNx is to name content and retrieve the content from the network without binding it to a specific network endpoint. A routing system (specified separately) populates the FIB tables at each CCNx router with hierarchical name prefixes that point towards the content producers under that prefix. A request finds matching content along those paths, in which case a response carries the data, or, if no match is found, a control message indicates the failure. A request may further refine acceptable responses with a restriction on the response's signer and the cryptographic hash of the response. The details of these restrictions are described below.

The CCNx name is a hierarchical series of name segments. Each name segment has a type and zero or more bytes. Matching two names is done as a binary comparison of the type and value, and is done segment by segment. The human-readable form is defined under a URI scheme "ccnx:" [ccnx-uri], though the canonical encoding of a name is a series of pairs (type, octet string). There is no requirement that any name segment be human readable or UTF-8. The first few segments in a name will be matched against the FIB, and a routing protocol may put its own restrictions on the routable name components (e.g., a maximum length or character-encoding rules). In principle, name segments and names have unbounded length, though in practice they are limited by the wire encoding and practical considerations imposed by a routing protocol. Note that in CCNx, name segments use binary comparison, whereas in a URI, the authority uses a case-insensitive hostname (due to DNS).

The CCNx name, as used by the forwarder, is purposefully left as a general octet-encoded type and value without any requirements on human readability and character encoding. The reason for this is that we are concerned with how a forwarder processes names. We expect that applications, routing protocols, or other higher layers will apply their own conventions and restrictions on the allowed name segment types and name segment values.

CCNx is a request and response protocol that fetches chunks of data using a name. The integrity of each chunk may be directly asserted through a digital signature or Message Authentication Code (MAC), or, alternatively, indirectly via hash chains. Chunks may also carry weaker Message Integrity Codes (MICs) or no integrity protection mechanism at all. Because provenance information is carried with each chunk (or larger indirectly protected block), we no longer need to rely on host identities, such as those derived from TLS certificates, to ascertain the chunk legitimacy. Therefore, data integrity is a core feature of CCNx; it does not rely on the data transmission channel. There are several options for data confidentiality, discussed later.

This document only defines the general properties of CCNx names. In some isolated environments, CCNx users may be able to use any name they choose and either inject that name (or prefix) into a routing protocol or use other information foraging techniques. In the Internet environment, there will be policies around the formats of names and assignments of names to publishers, though those are not specified here.

The key concept of CCNx is that a subjective name is cryptographically bound to a fixed payload. These publisher-generated bindings can therefore be cryptographically verified. A named payload is thus the tuple {{Name, ExtraFields, Payload, ValidationAlgorithm}, ValidationPayload}, where all fields in the inner tuple are covered by the validation payload (e.g., signature). Consumers of this data can check the binding integrity by recomputing the same cryptographic hash and verifying the digital signature in ValidationPayload.

In addition to digital signatures (e.g., RSA), CCNx also supports message authentication codes (e.g., Hashed Message Authentication Code (HMAC)) and message integrity codes (e.g., Cyclic Redundancy Checks (CRC)). To maintain the cryptographic binding, there should be at least one object with a signature or authentication code, but not all objects require it. For example, a first object with a signature could refer to other objects via a hash chain, a Merkle tree, or a signed manifest. The later objects may not have any validation and rely purely on the references. The use of an integrity code (e.g., CRC) is intended for detecting accidental corruption in an Interest.

CCNx specifies a network protocol around Interests (request messages) and Content Objects (response messages) to move named payloads. An Interest includes the Name field, which identifies the desired response, and optional matching restrictions. Restrictions limit the possible matching Content Objects. Two restrictions exist: the Key ID restriction (KeyIdRestr) and Content Object Hash restriction (ContentObjectHashRestr). The first restriction on the KeyId limits responses to those signed with a ValidationAlgorithm KeyId field equal to the restriction. The second is the Content Object Hash restriction, which limits the response to one where the cryptographic hash of the entire named payload is equal to the restriction. Section 9 fully explains how these restrictions limit matching of a Content Object to an Interest.

The hierarchy of a CCNx name is used for routing via the longest matching prefix in a forwarder. The longest matching prefix is computed name segment by name segment in the hierarchical name, where each name segment must be exactly equal to match. There is no requirement that the prefix be globally routable. Within a deployment, any local routing may be used, even one that only uses a single flat (nonhierarchical) name segment.

Another concept of CCNx is that there should be flow balance between Interest messages and Content Object messages. At the network level, an Interest traveling along a single path should elicit no more than one Content Object response. If some node sends the Interest along more than one path, that node should consolidate the responses such that only one Content Object flows back towards the requester. If an Interest is sent broadcast or multicast on a multiple-access media, the sender should be prepared for multiple responses unless some other media-dependent mechanism like gossip suppression or leader election is used.

As an Interest travels the forward path following the FIB, it establishes state at each forwarder such that a Content Object response can trace its way back to the original requester(s) without the requester needing to include a routable return address. We use the notional Pending Interest Table (PIT) as a method to store state that facilitates the return of a Content Object.

The notional PIT stores the last hop of an Interest plus its Name field and optional restrictions. This is the data required to match a Content Object to an Interest (see Section 9). When a Content

Object arrives, it must be matched against the PIT to determine which entries it satisfies. For each such entry, at most one copy of the Content Object is sent to each listed last hop in the PIT entries.

An actual PIT is not mandated by this specification. An implementation may use any technique that gives the same external behavior. There are, for example, research papers that use techniques like label switching in some parts of the network to reduce the per-node state incurred by the PIT [dart]. Some implementations store the PIT state in the FIB, so there is not a second table.

If multiple Interests with the same {Name, [KeyIdRestr], [ContentObjectHashRestr]} tuple arrive at a node before a Content Object matching the first Interest comes back, they are grouped in the same PIT entry and their last hops are aggregated (see Section 2.4.2). Thus, one Content Object might satisfy multiple pending Interests in a PIT.

In CCNx, higher-layer protocols are often called "name-based protocols" because they operate on the CCNx name. For example, a versioning protocol might append additional name segments to convey state about the version of payload. A content discovery protocol might append certain protocol-specific name segments to a prefix to discover content under that prefix. Many such protocols may exist and apply their own rules to names. They may be layered with each protocol encapsulating (to the left) a higher layer’s name prefix.

This document also describes a control message called an Interest Return. A network element may return an Interest message to a previous hop if there is an error processing the Interest. The returned Interest may be further processed at the previous hop or returned towards the Interest origin. When a node returns an Interest, it indicates that the previous hop should not expect a response from that node for the Interest, i.e., there is no PIT entry left at the returning node for a Content Object to follow.

There are multiple ways to describe larger objects in CCNx. Aggregating L3 Content Objects into larger objects is beyond the scope of this document. One proposed method, File-Like ICN Collection (FLIC) [flic], uses a manifest to enumerate the pieces of a larger object. Manifests are, themselves, Content Objects. Another option is to use a convention in the Content Object name, as in the CCNx Chunking [chunking] protocol where a large object is broken into small chunks and each chunk receives a special name component indicating its serial order.

At the semantic level, described in this document, we do not address fragmentation. One experimental fragmentation protocol, BeginEnd Fragments [befrags], uses a multipoint PPP-style technique for use over L2 interfaces with the specification for CCNx Messages [RFC8609] in TLV wire encoding.

With these concepts, the remainder of the document specifies the behavior of a forwarder in processing Interest, Content Object, and Interest Return messages.

2. Protocol

This section defines the grammar of a CCNx Message (Interest, Content Object, or Interest Return). It then presents typical behaviors for a consumer, a publisher, and a forwarder. In the forwarder section, there are detailed descriptions about how to handle the forwarder-specific topics, such as HopLimit and Content Store, along with detailed processing pipelines for Interest and Content Object messages.

2.1. Message Grammar

The CCNx Message ABNF [RFC5234] grammar is shown in Figure 1. The grammar does not include any encoding delimiters, such as TLVs. Specific wire encodings are given in a separate document. If a Validation section exists, the Validation Algorithm covers from the Body (BodyName or BodyOptName) through the end of the ValidationAlg section. The InterestLifetime, CacheTime, and Return Code fields exist outside of the validation envelope and may be modified.

HashType, PayloadType, and Private Enterprise Number (PEN) need to correspond to IANA values registered in the "CCNx Hash Function Types" and "CCNx Payload Types" registries [ccnx-registry], as well as the "Private Enterprise Numbers" registry [eprise-numbers], respectively.

The various fields, in alphabetical order, are defined as:

AbsTime: Absolute times are conveyed as the 64-bit UTC time in milliseconds since the epoch (standard POSIX time).

CacheTime: The absolute time after which the publisher believes there is low value in caching the Content Object. This is a recommendation to caches (see Section 4).

Cert: Some applications may wish to embed an X.509 certificate to both validate the signature and provide a trust anchor. The Cert is a DER-encoded X.509 certificate.

ConObjField: These are optional fields that may appear in a Content Object.

ConObjHash: The value of the Content Object Hash, which is the SHA256-32 over the message from the beginning of the body to the end of the message. Note that this coverage area is different from the ValidationAlg. This value SHOULD NOT be trusted across domains (see Section 5).

ContentObjectHashRestr: The Content Object Hash restriction. A Content Object must hash to the same value as the restriction using the same HashType. The ContentObjectHashRestr MUST use SHA256-32.

ExpiryTime: An absolute time after which the Content Object should be considered expired (see Section 4).

Hash: Hash values carried in a Message carry a HashType to identify the algorithm used to generate the hash followed by the hash value. This form is to allow hash agility. Some fields may mandate a specific HashType.

HashType: The algorithm used to calculate a hash, which must correspond to one of the IANA "CCNx Hash Function Types" [ccnx-registry].

HopLimit: Interest messages may loop if there are loops in the forwarding plane. To eventually terminate loops, each Interest carries a HopLimit that is decremented after each hop and no longer forwarded when it reaches zero. See Section 2.4.

InterestField: These are optional fields that may appear in an Interest message.

KeyId: An identifier for the key used in the ValidationAlg. See Validation (Section 8) for a description of how it is used for MACs and signatures.

KeyIdRestr: The KeyId Restriction. A Content Object must have a KeyId with the same value as the restriction.

KeyLink: A Link (see Section 6) that names how to retrieve the key used to verify the ValidationPayload (see Section 8).

Lifetime: The approximate time during which a requester is willing to wait for a response, usually measured in seconds. It is not strongly related to the network round-trip time, though it must necessarily be larger.

Name: A name is made up of a nonempty first segment followed by zero or more additional segments, which may be of 0 length. Name segments are opaque octet strings and are thus case sensitive if encoding UTF-8. An Interest MUST have a Name. A Content Object MAY have a Name (see Section 9). The segments of a name are said to be complete if its segments uniquely identify a single Content Object. A name is exact if its segments are complete. An Interest carrying a full name is one that specifies an exact name and the Content Object Hash restriction of the corresponding Content Object.

Payload: The message’s data, as defined by PayloadType.

PayloadType: The format of the Payload field. If missing, assume Data type (T_PAYLOADTYPE_DATA) [ccnx-registry]. Data type means the payload is opaque application bytes. Key type (T_PAYLOADTYPE_KEY [ccnx-registry]) means the payload is a DER-encoded public key or X.509 certificate. Link type (T_PAYLOADTYPE_LINK [ccnx-registry]) means it is one or more Links (see Section 6).

PublicKey: Some applications may wish to embed the public key used to verify the signature within the message itself. The PublickKey is DER encoded.

RelTime: A relative time, measured in milliseconds.

ReturnCode: States the reason an Interest message is being returned to the previous hop (see Section 10.2).

SigTime: The absolute time (UTC milliseconds) when the signature was generated. The signature time only applies to the validation algorithm; it does not necessarily represent when the validated message was created.

Vendor: Vendor-specific opaque data. The Vendor data includes the IANA Private Enterprise Numbers [eprise-numbers], followed by vendor-specific information. CCNx allows vendor-specific data in most locations of the grammar.

```
Message       = Interest / ContentObject / InterestReturn
Interest      = IntHdr BodyName [Validation]
IntHdr        = HopLimit [Lifetime] *Vendor
ContentObject = ConObjHdr BodyOptName [Validation]
ConObjHdr     = [CacheTime / ConObjHash] *Vendor
InterestReturn= ReturnCode Interest
BodyName      = Name Common
BodyOptName   = [Name] Common
```

```
   Common         = *Field [Payload]
   Validation     = ValidationAlg ValidationPayload

   Name           = FirstSegment *Segment
   FirstSegment   = 1*OCTET / Vendor
   Segment        = *OCTET / Vendor

   ValidationAlg = (RSA-SHA256 / EC-SECP-256K1 / EC-SECP-384R1 /
                    HMAC-SHA256 / CRC32C) *Vendor
   ValidationPayload = 1*OCTET
   PublicAlg      = KeyId [SigTime] [KeyLink] [PublicKey] [Cert]
   RSA-SHA256     = PublicAlg
   EC-SECP-256K1 = PublicAlg
   EC-SECP-384R1 = PublicAlg
   HMAC-SHA256    = KeyId [SigTime] [KeyLink]
   CRC32C         = [SigTime]

   AbsTime        = 8OCTET ; 64-bit UTC msec since epoch
   CacheTime      = AbsTime
   ConObjField    = ExpiryTime / PayloadType
   ConObjHash     = Hash
   ExpiryTime     = AbsTime
   Field          = InterestField / ConObjField / Vendor
   Hash           = HashType 1*OCTET
   HashType       = 2OCTET ; IANA "CCNx Hash Function Types"
   HopLimit       = OCTET
   InterestField = KeyIdRestr / ContentObjectHashRestr
   KeyId          = Hash
   KeyIdRestr     = Hash
   KeyLink        = Link
   Lifetime       = RelTime
   Link           = Name [KeyIdRestr] [ContentObjectHashRestr]
   ContentObjectHashRestr  = Hash
   Payload        = *OCTET
   PayloadType    = OCTET ; IANA "CCNx Payload Types"
   PublicKey      = *OCTET ; DER-encoded public key
   Cert           = *OCTET ; DER-encoded X.509 Certificate
   RelTime        = 1*OCTET ; msec
   ReturnCode     = OCTET ; see Section 10.2
   SigTime        = AbsTime
   Vendor         = PEN *OCTET
   PEN            = 1*OCTET ; IANA "Private Enterprise Number"
```

Figure 1: CCNx Message ABNF Grammar

2.2. Consumer Behavior

To request a piece of content for a given {Name, [KeyIdRest], [ContentObjectHashRestr]} tuple, a consumer creates an Interest message with those values. It MAY add a validation section, typically only a CRC32C. A consumer MAY put a Payload field in an Interest to send additional data to the producer beyond what is in the name. The name is used for routing and may be remembered at each hop in the notional PIT to facilitate returning a Content Object; storing large amounts of state in the name could lead to high memory requirements. Because the payload is not considered when forwarding an Interest or matching a Content Object to an Interest, a consumer SHOULD put an Interest Payload ID (see Section 3.2) as part of the name to allow a forwarder to match Interests to Content Objects and avoid aggregating Interests with different payloads. Similarly, if a consumer uses a MAC or a signature, it SHOULD also include a unique segment as part of the name to prevent the Interest from being aggregated with other Interests or satisfied by a Content Object that has no relation to the validation.

The consumer SHOULD specify an InterestLifetime, which is the length of time the consumer is willing to wait for a response. The InterestLifetime is an application-scale time, not a network round-trip time (see Section 2.4.2). If not present, the InterestLifetime will use a default value (2 seconds).

The consumer SHOULD set the Interest HopLimit to a reasonable value or use the default 255. If the consumer knows the distances to the producer via routing, it SHOULD use that value.

A consumer hands off the Interest to its first forwarder, which will then forward the Interest over the network to a publisher (or replica) that may satisfy it based on the name (see Section 2.4).

Interest messages are unreliable. A consumer SHOULD run a transport protocol that will retry the Interest if it goes unanswered, up to the InterestLifetime. No transport protocol is specified in this document.

The network MAY send to the consumer an Interest Return message that indicates the network cannot fulfill the Interest. The ReturnCode specifies the reason for the failure, such as no route or congestion. Depending on the ReturnCode, the consumer MAY retry the Interest or MAY return an error to the requesting application.

If the content was found and returned by the first forwarder, the consumer will receive a Content Object. The consumer uses the following set of checks to validate a received Content Object:

- The consumer MUST ensure the Content Object is properly formatted.

- The consumer MUST verify that the returned Content Object matches one or more pending Interests as per Section 9.

- If the Content Object is signed, the consumer SHOULD cryptographically verify the signature as per Section 8. If it does not have the corresponding key, it SHOULD fetch the key, such as from a key resolution service or via the KeyLink.

- If the signature has a SigTime, the consumer MAY use that in considering if the signature is valid. For example, if the consumer is asking for dynamically generated content, it should expect the SigTime not to be before the time the Interest was generated.

- If the Content Object is signed, the consumer SHOULD assert the trustworthiness of the signing key to the namespace. Such an assertion is beyond the scope of this document, though one may use traditional PKI methods, a trusted key resolution service, or methods like [trust].

- The consumer MAY cache the Content Object for future use, up to the ExpiryTime if present.

- The consumer MAY accept a Content Object off the wire that is expired. A packet Content Object may expire while in flight; there is no requirement that forwarders drop expired packets in flight. The only requirement is that Content Stores, caches, or producers MUST NOT respond with an expired Content Object.

2.3. Publisher Behavior

This document does not specify the method by which names populate a FIB table at forwarders (see Section 2.4). A publisher is either configured with one or more name prefixes under which it may create content or it chooses its name prefixes and informs the routing layer to advertise those prefixes.

When a publisher receives an Interest, it SHOULD:

- Verify that the Interest is part of the publisher’s namespace(s).

- If the Interest has a Validation section, verify it as per Section 8. Usually an Interest will only have a CRC32C, unless the publisher application specifically accommodates other validations. The publisher MAY choose to drop Interests that carry a Validation section if the publisher application does not expect those signatures, as this could be a form of computational denial of service. If the signature requires a key that the publisher does not have, it is NOT RECOMMENDED that the publisher fetch the key over the network unless it is part of the application's expected behavior.

- Retrieve or generate the requested Content Object and return it to the Interest's previous hop. If the requested content cannot be returned, the publisher SHOULD reply with an Interest Return or a Content Object with application payload that says the content is not available; this Content Object should have a short ExpiryTime in the future or not be cacheable (i.e., an expiry time of 0).

2.4. Forwarder Behavior

A forwarder routes Interest messages based on a Forwarding Information Base (FIB), returns Content Objects that match Interests to the Interest's previous hop, and processes Interest Return control messages. It may also keep a cache of Content Objects in the notional Content Store table. This document does not specify the internal behavior of a forwarder, only these and other external behaviors.

In this document, we will use two processing pipelines: one for Interests and one for Content Objects. Interest processing is made up of checking for duplicate Interests in the PIT (see Section 2.4.2), checking for a cached Content Object in the Content Store (see Section 2.4.3), and forwarding an Interest via the FIB. Content Store processing is made up of checking for matching Interests in the PIT and forwarding to those previous hops.

2.4.1. Interest HopLimit

Interest looping is not prevented in CCNx. An Interest traversing loops is eventually discarded using the hop-limit field of the Interest, which is decremented at each hop traversed by the Interest.

A loop may also terminate because the Interest is aggregated with its previous PIT entry along the loop. In this case, the Content Object will be sent back along the loop and eventually return to a node that already forwarded the content, so it will likely not have a PIT entry anymore. When the content reaches a node without a PIT entry, it will be discarded. It may be that a new Interest or another looped Interest will return to that same node, in which case the node will return a cached response to make a new PIT entry, as below.

The HopLimit is the last resort method to stop Interest loops where a Content Object chases an Interest around a loop and where the intermediate nodes, for whatever reason, no longer have a PIT entry and do not cache the Content Object.

Every Interest MUST carry a HopLimit. An Interest received from a local application MAY have a 0 HopLimit, which restricts the Interest to other local sources.

When an Interest is received from another forwarder, the HopLimit MUST be positive, otherwise the forwarder will discard the Interest. A forwarder MUST decrement the HopLimit of an Interest by at least 1 before it is forwarded.

If the decremented HopLimit equals 0, the Interest MUST NOT be forwarded to another forwarder; it MAY be sent to a local publisher application or serviced from a local Content Store.

A RECOMMENDED HopLimit-processing pipeline is below:

- If Interest received from a remote system:
  - If received HopLimit is 0, optionally send Interest Return (HopLimit Exceeded), and discard Interest.
  - Otherwise, decrement the HopLimit by 1.
- Process as per Content Store and Aggregation rules.
- If the Interest will be forwarded:
  - If the (potentially decremented) HopLimit is 0, restrict forwarding to the local system.
  - Otherwise, forward as desired to local or remote systems.

2.4.2. Interest Aggregation

Interest aggregation is when a forwarder receives an Interest message that could be satisfied by the response to another Interest message already forwarded by the node, so the forwarder suppresses forwarding the new Interest; it only records the additional previous hop so a Content Object sent in response to the first Interest will satisfy both Interests.

CCNx uses an Interest aggregation rule that assumes the InterestLifetime is akin to a subscription time and is not a network round-trip time. Some previous aggregation rules assumed the lifetime was a round-trip time, but this leads to problems of expiring an Interest before a response comes if the RTT is estimated too short or interfering with an Automatic Repeat reQuest (ARQ) scheme that wants to retransmit an Interest but a prior Interest overestimated the RTT.

A forwarder MAY implement an Interest aggregation scheme. If it does not, then it will forward all Interest messages. This does not imply that multiple, possibly identical, Content Objects will come back. A forwarder MUST still satisfy all pending Interests, so one Content Object could satisfy multiple similar Interests, even if the forwarder did not suppress duplicate Interest messages.

A RECOMMENDED Interest aggregation scheme is:

- Two Interests are considered "similar" if they have the same Name, KeyIdRestr, and ContentObjectHashRestr, where a missing optional field in one must be missing in the other.

- Let the notional value InterestExpiry (a local value at the forwarder) be equal to the receive time plus the InterestLifetime (or a platform-dependent default value if not present).

- An Interest record (PIT entry) is considered invalid if its InterestExpiry time is in the past.

- The first reception of an Interest MUST be forwarded.

- A second or later reception of an Interest similar to a valid pending Interest from the same previous hop MUST be forwarded. We consider these a retransmission request.

- A second or later reception of an Interest similar to a valid pending Interest from a new previous hop MAY be aggregated (not forwarded). If this Interest has a larger HopLimit than the pending Interest, it MUST be forwarded.

- Aggregating an Interest MUST extend the InterestExpiry time of the Interest record. An implementation MAY keep a single InterestExpiry time for all previous hops or MAY keep the InterestExpiry time per previous hop. In the first case, the forwarder might send a Content Object down a path that is no longer waiting for it, in which case the previous hop (next hop of the Content Object) would drop it.

2.4.3. Content Store Behavior

The Content Store is a special cache that is an integral part of a CCNx forwarder. It is an optional component. It serves to repair lost packets and handle flash requests for popular content. It could be prepopulated or use opportunistic caching. Because the Content Store could serve to amplify an attack via cache poisoning, there are special rules about how a Content Store behaves.

1. A forwarder MAY implement a Content Store. If it does, the Content Store matches a Content Object to an Interest via the normal matching rules (see Section 9).

2. If an Interest has a KeyId restriction, then the Content Store MUST NOT reply unless it knows the signature on the matching Content Object is correct. It may do this by external knowledge (i.e., in a managed network or system with prepopulated caches) or by having the public key and cryptographically verifying the signature. A Content Store is NOT REQUIRED to verify signatures; if it does not, then it treats these cases like a cache miss.

3. If a Content Store chooses to verify signatures, then it MAY do so as follows. If the public key is provided in the Content Object itself (i.e., in the PublicKey field) or in the Interest, the Content Store MUST verify that the public key's hash is equal to the KeyId and that it verifies the signature (see Section 8.4). A Content Store MAY verify the digital signature of a Content Object before it is cached, but it is not required to do so. A Content Store SHOULD NOT fetch keys over the network. If it cannot or has not yet verified the signature, it should treat the Interest as a cache miss.

4. If an Interest has a Content Object Hash restriction, then the Content Store MUST NOT reply unless it knows the matching Content Object has the correct hash. If it cannot verify the hash, then it should treat the Interest as a cache miss.

5. It must obey the cache control directives (see Section 4).

2.4.4. Interest Pipeline

1. Perform the HopLimit check (see Section 2.4.1).

2. If the Interest carries a validation, such as a MIC or a signature with an embedded public key or certificate, a forwarder MAY validate the Interest as per Section 8. A forwarder SHOULD NOT fetch keys via a KeyLink. If the forwarder drops an Interest

due to failed validation, it MAY send an Interest Return (Section 10.3.9).

3. Determine if the Interest can be aggregated as per Section 2.4.2. If it can be, aggregate and do not forward the Interest.

4. If forwarding the Interest, check for a hit in the Content Store as per Section 2.4.3. If a matching Content Object is found, return it to the Interest's previous hop. This injects the Content Store as per Section 2.4.5.

5. Look up the Interest in the FIB. Longest Prefix Match (LPM) is performed name segment by name segment (not byte or bit). It SHOULD exclude the Interest's previous hop. If a match is found, forward the Interest. If no match is found or the forwarder chooses not to forward due to a local condition (e.g., congestion), it SHOULD send an Interest Return message as per Section 10.

2.4.5. Content Object Pipeline

1. It is RECOMMENDED that a forwarder that receives a Content Object check that the Content Object came from an expected previous hop. An expected previous hop is one pointed to by the FIB or one recorded in the PIT as having had a matching Interest sent that way.

2. A Content Object MUST be matched to all pending Interests that satisfy the matching rules (see Section 9). Each satisfied pending Interest MUST then be removed from the set of pending Interests.

3. A forwarder SHOULD NOT send more than one copy of the received Content Object to the same Interest previous hop. It may happen, for example, that two Interests ask for the same Content Object in different ways (e.g., by name and by name and KeyId), and that they both come from the same previous hop. It is normal to send the same Content Object multiple times on the same interface, such as Ethernet, if it is going to different previous hops.

4. A Content Object SHOULD only be put in the Content Store if it satisfied an Interest (and passed rule #1 above). This is to reduce the chances of cache poisoning.

3. Names

A CCNx name is a composition of name segments. Each name segment carries a label identifying the purpose of the name segment, and a value. For example, some name segments are general names and some serve specific purposes such as carrying version information or the sequencing of many chunks of a large object into smaller, signed Content Objects.

There are three different types of names in CCNx: prefix, exact, and full names. A prefix name is simply a name that does not uniquely identify a single Content Object, but rather a namespace or prefix of an existing Content Object name. An exact name is one that uniquely identifies the name of a Content Object. A full name is one that is exact and is accompanied by an explicit or implicit ConObjHash. The ConObjHash is explicit in an Interest and implicit in a Content Object.

Note that a forwarder does not need to know any semantics about a name. It only needs to be able to match a prefix to forward Interests and match an exact or full name to forward Content Objects. It is not sensitive to the name segment types.

The name segment labels specified in this document are given in Table 1. Name Segment is a general name segment, typically occurring in the routable prefix and user-specified content name. Interest Payload ID is a name segment to identify the Interest's payload. Application Components are a set of name segment types reserved for application use.

Type	Description
Name Segment	A generic name segment that includes arbitrary octets.
Interest Payload ID	An octet string that identifies the payload carried in an Interest. As an example, the Payload ID might be a hash of the Interest Payload. This provides a way to differentiate between Interests based on the payload solely through a name segment without having to include all the extra bytes of the payload itself.
Application Components	An application-specific payload in a name segment. An application may apply its own semantics to these components. A good practice is to identify the application in a name segment prior to the application component segments.

Table 1: CCNx Name Segment Types

At the lowest level, a forwarder does not need to understand the semantics of name segments; it need only identify name segment boundaries and be able to compare two name segments (both label and value) for equality. The forwarder matches names segment by segment against its forwarding table to determine a next hop.

3.1. Name Examples

This section uses the CCNx URI [ccnx-uri] representation of CCNx names. Note that as per the message grammar, an Interest must have a Name with at least one name segment that must have at least 1 octet of value. A Content Object must have a similar name or no name at all. The FIB, on the other hand, could have 0-length names (a default route), or a first name segment with no value, or a regular name.

Name	Description
ccnx:/	A 0-length name, corresponds to a default route.
ccnx:/NAME=	A name with 1 segment of 0 length, distinct from ccnx:/.
ccnx:/NAME=foo/APP:0=bar	A 2-segment name, where the first segment is of type NAME and the second segment is of type APP:0.

Table 2: CCNx Name Examples

3.2. Interest Payload ID

An Interest may also have a Payload field that carries state about the Interest but is not used to match a Content Object. If an Interest contains a payload, the Interest name should contain an Interest Payload ID (IPID). The IPID allows a PIT entry to correctly multiplex Content Objects in response to a specific Interest with a specific payload ID. The IPID could be derived from a hash of the payload or could be a Globally Unique Identifier (GUID) or a nonce. An optional Metadata field defines the IPID field so other systems can verify the IPID, such as when it is derived from a hash of the payload. No system is required to verify the IPID.

4. Cache Control

CCNx supports two fields that affect cache control. These determine how a cache or Content Store handles a Content Object. They are not used in the fast path; they are only used to determine if a Content Object can be injected onto the fast path in response to an Interest.

The ExpiryTime is a field that exists within the signature envelope of a Validation Algorithm. It is the UTC time in milliseconds after which the Content Object is considered expired and MUST no longer be used to respond to an Interest from a cache. Stale content MAY be flushed from the cache.

The Recommended Cache Time (RCT) is a field that exists outside the signature envelope. It is the UTC time in milliseconds after which the publisher considers the Content Object to be of low value to cache. A cache SHOULD discard it after the RCT, though it MAY keep it and still respond with it. A cache MAY also discard the Content Object before the RCT time; there is no contractual obligation to remember anything.

This formulation allows a producer to create a Content Object with a long ExpiryTime but short RCT and keep republishing the same signed Content Object over and over again by extending the RCT. This allows a form of "phone home" where the publisher wants to periodically see that the content is being used.

5. Content Object Hash

CCNx allows an Interest to restrict a response to a specific hash. The hash covers the Content Object message body and the validation sections, if present. Thus, if a Content Object is signed, its hash includes that signature value. The hash does not include the fixed or hop-by-hop headers of a Content Object. Because it is part of the matching rules (see Section 9), the hash is used at every hop.

There are two options for matching the Content Object Hash restriction in an Interest. First, a forwarder could compute for itself the hash value and compare it to the restriction. This is an expensive operation. The second option is for a border device to compute the hash once and place the value in a header (ConObjHash) that is carried through the network. The second option, of course, removes any security properties from matching the hash, so it SHOULD only be used within a trusted domain. The header SHOULD be removed when crossing a trust boundary.

6. Link

A Link is the tuple {Name, [KeyIdRestr], [ContentObjectHashRestr]}. The information in a Link comprises the fields of an Interest that would retrieve the Link target. A Content Object with PayloadType of "Link" is an object whose payload is one or more Links. This tuple may be used as a KeyLink to identify a specific object with the certificate-wrapped key. It is RECOMMENDED to include at least one of either KeyId restriction or Content Object Hash restriction. If neither restriction is present, then any Content Object with a matching name from any publisher could be returned.

7. Hashes

Several protocol fields use cryptographic hash functions, which must be secure against attack and collisions. Because these hash functions change over time, with better ones appearing and old ones falling victim to attacks, it is important that a CCNx protocol implementation supports hash agility.

In this document, we suggest certain hashes (e.g., SHA-256), but a specific implementation may use what it deems best. The normative CCNx Messages [RFC8609] specification should be taken as the definition of acceptable hash functions and uses.

8. Validation

8.1. Validation Algorithm

The Validator consists of a ValidationAlgorithm that specifies how to verify the message and a ValidationPayload containing the validation output, e.g., the digital signature or MAC. The ValidationAlgorithm section defines the type of algorithm to use and includes any necessary additional information. The validation is calculated from the beginning of the CCNx Message through the end of the ValidationAlgorithm section (i.e., up to but not including the validation payload). We refer to this as the validation region. The ValidationPayload is the integrity value bytes, such as a MAC or signature.

The CCNx Message Grammar (Section 2.1) shows the allowed validation algorithms and their structure. In the case of a Vendor algorithm, the vendor may use any desired structure. A Validator can only be applied to an Interest or a Content Object, not an Interest Return. An Interest inside an Interest Return would still have the original validator, if any.

The grammar allows multiple Vendor extensions to the validation algorithm. It is up to the vendor to describe the validation region for each extension. A vendor may, for example, use a regular signature in the validation algorithm, then append a proprietary MIC to allow for in-network error checking without using expensive signature verification. As part of this specification, we do not allow for multiple Validation Algorithm blocks apart from these vendor methods.

8.2. Message Integrity Codes

If the validation algorithm is CRC32C, then the validation payload is the output of the CRC over the validation region. This validation algorithm allows for an optional signature time (SigTime) to timestamp when the message was validated (calling it a "signature" time is a slight misnomer, but we reuse the same field for this purpose between MICs, MACs, and signatures).

MICs are usually used with an Interest to avoid accidental in-network corruption. They are usually not used on Content Objects because the objects are either signed or linked to by hash chains, so the CRC32C is redundant.

8.3. Message Authentication Codes

If the validation algorithm is HMAC-SHA256, then the validation payload is the output of the HMAC over the validation region. The validation algorithm requires a KeyId and allows for a signature time (SigTime) and KeyLink.

The KeyId field identifies the shared secret used between two parties to authenticate messages. These secrets SHOULD be derived from a key exchange protocol such as [ccnx-ke]. The KeyId, for a shared secret, SHOULD be an opaque identifier not derived from the actual key; an integer counter, for example, is a good choice.

The signature time is the timestamp when the authentication code was computed and added to the messages.

The KeyLink field in a MAC indicates how to negotiate keys and should point towards the key exchange endpoint. The use of a key negotiation algorithm is beyond the scope of this specification, and a key negotiation algorithm is not required to use this field.

8.4. Signature

Signature-validation algorithms use public key cryptographic algorithms such as RSA and the Elliptic Curve Digital Signature Algorithm (ECDSA). This specification and the corresponding wire encoding [RFC8609] only support three specific signature algorithms: RSA-SHA256, EC-SECP-256K1, and EC-SECP-384R1. Other algorithms may be added in through other documents or by using experimental or vendor-validation algorithm types.

A signature that is public key based requires a KeyId field and may optionally carry a signature time, an embedded public key, an embedded certificate, and a KeyLink. The signature time behaves as normal to timestamp when the signature was created. We describe the use and relationship of the other fields here.

It is not common to use embedded certificates, as they can be very large and may have validity periods different than the validated data. The preferred method is to use a KeyLink to the validating certificate.

The KeyId field in the ValidationAlgorithm identifies the public key used to verify the signature. It is similar to a Subject Key Identifier from X.509 (Section 4.2.1.2 of [RFC5280]). We define the KeyId to be a cryptographic hash of the public key in DER form. All implementations MUST support the SHA-256 digest as the KeyId hash.

The use of other algorithms for the KeyId is allowed, and it will not cause problems at a forwarder because the forwarder only matches the digest algorithm and digest output and does not compute the digest (see Section 9). It may cause issues with a Content Store, which needs to verify the KeyId and PublicKey match (see Section 2.4.3); though in this case, it only causes a cache miss and the Interest would still be forwarded to the publisher. As long as the publisher and consumers support the hash, data will validate.

As per Section 9, a forwarder only matches the KeyId to a KeyId restriction. It does not need to look at the other fields such as the public key, certificate, or KeyLink.

If a message carries multiples of the KeyId, public key, certificate, or KeyLink, an endpoint (consumer, cache, or Content Store) MUST ensure that any fields it uses are consistent. The KeyId MUST be the corresponding hash of the embedded public key or certificate public key. The hash function to use is the KeyId's HashType. If there is both an embedded public key and a certificate, the public keys MUST be the same.

A message SHOULD NOT have both a PublicKey and a KeyLink to a public key, as that is redundant. It MAY have a PublicKey and a KeyLink to a certificate.

A KeyLink in a first Content Object may point to a second Content Object with a DER-encoded public key in the PublicKey field and an optional DER-encoded X.509 certificate in the payload. The second Content Object's KeyId MUST equal the first object's KeyId. The second object's PublicKey field MUST be the public key corresponding to the KeyId. That key must validate both the first and second object's signature. A DER-encoded X.509 certificate may be included in the second object's payload and its embedded public key MUST match the PublicKey. It must be issued by a trusted authority, preferably specifying the valid namespace of the key in the distinguished name. The second object MUST NOT have a KeyLink; we do not allow for recursive key lookup.

9. Interest to Content Object Matching

A Content Object satisfies an Interest if and only if (a) the Content Object name, if present, exactly matches the Interest name, (b) the ValidationAlgorithm KeyId of the Content Object exactly equals the Interest KeyId restriction, if present, and (c) the computed Content Object Hash exactly equals the Interest Content Object Hash restriction, if present.

The KeyId and KeyIdRestr use the Hash format (see Section 2.1). The Hash format has an embedded HashType followed by the hash value. When comparing a KeyId and KeyIdRestr, one compares both the HashType and the value.

The matching rules are given by this predicate, which, if it evaluates true, means the Content Object matches the Interest. Ni = Name in the Interest (may not be empty), Ki = KeyIdRestr in the Interest (may be empty), and Hi = ContentObjectHashRestr in the Interest (may be empty). Likewise, No, Ko, and Ho are those properties in the Content Object, where No and Ko may be empty; Ho always exists (it is an intrinsic property of the Content Object). For binary relations, we use "&" for AND and "|" for OR. We use "E" for the EXISTS (not empty) operator and "!" for the NOT EXISTS operator.

As a special case, if the Content Object Hash restriction in the Interest specifies an unsupported hash algorithm, then no Content Object can match the Interest, so the system should drop the Interest and MAY send an Interest Return to the previous hop. In this case, the predicate below will never get executed because the Interest is never forwarded. If the system is using the optional behavior of having a different system calculate the hash for it, then the system may assume all hash functions are supported and leave it to the other system to accept or reject the Interest.

```
(!No | (Ni=No)) & (!Ki | (Ki=Ko)) & (!Hi | (Hi=Ho)) & (E No | E Hi)
```

As one can see, there are two types of attributes one can match. The first term depends on the existence of the attribute in the Content Object while the next two terms depend on the existence of the attribute in the Interest. The last term is the "Nameless Object" restriction that states that if a Content Object does not have a Name, then it must match the Interest on at least the Hash restriction.

If a Content Object does not carry the Content Object Hash as an expressed field, it must be calculated in network to match against. It is sufficient within an autonomous system to calculate a Content Object Hash at a border router and carry it via trusted means within the autonomous system. If a Content Object ValidationAlgorithm does not have a KeyId, then the Content Object cannot match an Interest with a KeyId restriction.

10. Interest Return

This section describes the process whereby a network element may return an Interest message to a previous hop if there is an error processing the Interest. The returned Interest may be further processed at the previous hop or returned towards the Interest origin. When a node returns an Interest, it indicates that the previous hop should not expect a response from that node for the Interest, i.e., there is no PIT entry left at the returning node.

The returned message maintains compatibility with the existing TLV packet format (a fixed header, optional hop-by-hop headers, and the CCNx Message body). The returned Interest packet is modified in only two ways:

- The PacketType is set to Interest Return to indicate a Feedback message.
- The ReturnCode is set to the appropriate value to signal the reason for the return.

The specific encodings of the Interest Return are specified in [RFC8609].

A forwarder is not required to send any Interest Return messages.

A forwarder is not required to process any received Interest Return message. If a forwarder does not process Interest Return messages, it SHOULD silently drop them.

The Interest Return message does not apply to a Content Object or any other message type.

An Interest Return message is a 1-hop message between peers. It is not propagated multiple hops via the FIB. An intermediate node that receives an Interest Return may take corrective actions or may propagate its own Interest Return to previous hops as indicated in the reverse path of a PIT entry.

10.1. Message Format

The Interest Return message looks exactly like the original Interest message with the exception of the two modifications mentioned above. The PacketType is set to indicate the message is an Interest Return, and the reserved byte in the Interest header is used as a Return Code. The numeric values for the PacketType and ReturnCodes are in [RFC8609].

10.2. ReturnCode Types

This section defines the Interest Return ReturnCode introduced in this RFC. The numeric values used in the packet are defined in [RFC8609].

Name	Description
No Route (Section 10.3.1)	The returning forwarder has no route to the Interest name.
HopLimit Exceeded (Section 10.3.2)	The HopLimit has decremented to 0 and needs to forward the packet.
Interest MTU too large (Section 10.3.3)	The Interest's MTU does not conform to the required minimum and would require fragmentation.
No Resources (Section 10.3.4)	The node does not have the resources to process the Interest.
Path error (Section 10.3.5)	There was a transmission error when forwarding the Interest along a route (a transient error).
Prohibited (Section 10.3.6)	An administrative setting prohibits processing this Interest.
Congestion (Section 10.3.7)	The Interest was dropped due to congestion (a transient error).
Unsupported Content Object Hash Algorithm (Section 10.3.8)	The Interest was dropped because it requested a Content Object Hash restriction using a hash algorithm that cannot be computed.
Malformed Interest (Section 10.3.9)	The Interest was dropped because it did not correctly parse.

Table 3: Interest Return Reason Codes

10.3. Interest Return Protocol

This section describes the forwarder behavior for the various Reason codes for Interest Return. A forwarder is not required to generate any of the codes, but if it does, it MUST conform to this specification.

If a forwarder receives an Interest Return, it SHOULD take these standard corrective actions. A forwarder is allowed to ignore Interest Return messages, in which case its PIT entry would go through normal timeout processes.

- Verify that the Interest Return came from a next hop to which it actually sent the Interest.
- If a PIT entry for the corresponding Interest does not exist, the forwarder should ignore the Interest Return.
- If a PIT entry for the corresponding Interest does exist, the forwarder MAY do one of the following:
  - Try a different forwarding path, if one exists, and discard the Interest Return, or
  - Clear the PIT state and send an Interest Return along the reverse path.

If a forwarder tries alternate routes, it MUST ensure that it does not use the same path multiple times. For example, it could keep track of which next hops it has tried and not reuse them.

If a forwarder tries an alternate route, it may receive a second Interest Return, possibly of a different type than the first Interest Return. For example, node A sends an Interest to node B, which sends a No Route return. Node A then tries node C, which sends a Prohibited Interest Return. Node A should choose what it thinks is the appropriate code to send back to its previous hop.

If a forwarder tries an alternate route, it should decrement the Interest Lifetime to account for the time spent thus far processing the Interest.

10.3.1. No Route

If a forwarder receives an Interest for which it has no route, or for which the only route is back towards the system that sent the Interest, the forwarder SHOULD generate a "No Route" Interest Return message.

How a forwarder manages the FIB table when it receives a No Route message is implementation dependent. In general, receiving a No Route Interest Return should not cause a forwarder to remove a route. The dynamic routing protocol that installed the route should correct the route, or the administrator who created a static route should correct the configuration. A forwarder could suppress using that next hop for some period of time.

10.3.2. HopLimit Exceeded

A forwarder MAY choose to send HopLimit Exceeded messages when it receives an Interest that must be forwarded off system and the HopLimit is 0.

10.3.3. Interest MTU Too Large

If a forwarder receives an Interest whose MTU exceeds the prescribed minimum, it MAY send an "Interest MTU Too Large" message, or it may silently discard the Interest.

If a forwarder receives an "Interest MTU Too Large" response, it SHOULD NOT try alternate paths. It SHOULD propagate the Interest Return to its previous hops.

10.3.4. No Resources

If a forwarder receives an Interest and it cannot process the Interest due to lack of resources, it MAY send an Interest Return. A lack of resources could mean the PIT is too large or that there is some other capacity limit.

10.3.5. Path Error

If a forwarder detects an error forwarding an Interest, such as over a reliable link, it MAY send a Path-Error Interest Return indicating that it was not able to send or repair a forwarding error.

10.3.6. Prohibited

A forwarder may have administrative policies, such as access control lists (ACLs), that prohibit receiving or forwarding an Interest. If a forwarder discards an Interest due to a policy, it MAY send a Prohibited Interest Return to the previous hop. For example, if there is an ACL that says "/example/private" can only come from interface e0, but the forwarder receives one from e1, the forwarder must have a way to return the Interest with an explanation.

10.3.7. Congestion

If a forwarder discards an Interest due to congestion, it MAY send a Congestion Interest Return to the previous hop.

10.3.8. Unsupported Content Object Hash Algorithm

If a Content Object Hash restriction specifies a hash algorithm the forwarder cannot verify, the Interest should not be accepted and the forwarder MAY send an Interest Return to the previous hop.

10.3.9. Malformed Interest

If a forwarder detects a structural or syntactical error in an Interest, it SHOULD drop the Interest and MAY send an Interest Return to the previous hop. This does not imply that any router must validate the entire structure of an Interest.

11. IANA Considerations

This document has no IANA actions.

12. Security Considerations

The CCNx protocol is an L3 network protocol, which may also operate as an overlay using other transports such as UDP or other tunnels. It includes intrinsic support for message authentication via a signature (e.g., RSA or elliptic curve) or message authentication code (e.g., HMAC). In lieu of an authenticator, it may instead use a message integrity check (e.g., SHA or CRC). CCNx does not specify an encryption envelope; that function is left to a high-layer protocol (e.g., [esic]).

The CCNx message format includes the ability to attach MICs (e.g., CRC32C), MACs (e.g., HMAC), and signatures (e.g., RSA or ECDSA) to all packet types. This does not mean that it is a good idea to use an arbitrary ValidationAlgorithm, nor to include computationally expensive algorithms in Interest packets, as that could lead to computational DoS attacks. Applications should use an explicit protocol to guide their use of packet signatures. As a general guideline, an application might use a MIC on an Interest to detect unintentionally corrupted packets. If one wishes to secure an Interest, one should consider using an encrypted wrapper and a protocol that prevents replay attacks, especially if the Interest is being used as an actuator. Simply using an authentication code or signature does not make an Interest secure. There are several examples in the literature on how to secure ICN-style messaging [mobile] [ace].

As an L3 protocol, this document does not describe how one arrives at keys or how one trusts keys. The CCNx Content Object may include a public key or certificate embedded in the object or may use the KeyLink field to point to a public key or certificate that authenticates the message. One key-exchange specification is CCNxKE [ccnx-ke] [mobile], which is similar to the TLS 1.3 key exchange except it is over the CCNx L3 messages. Trust is beyond the scope of an L3 protocol and left to applications or application frameworks.

The combination of an ephemeral key exchange (e.g., CCNxKE [ccnx-ke]) and an encapsulating encryption (e.g., [esic]) provides the equivalent of a TLS tunnel. Intermediate nodes may forward the Interests and Content Objects but have no visibility inside. It also completely hides the internal names in those used by the encryption layer. This type of tunneling encryption is useful for content that has little or no cacheability, as it can only be used by someone with the ephemeral key. Short-term caching may help with lossy links or mobility, but long-term caching is usually not of interest.

Broadcast encryption or proxy re-encryption may be useful for content with multiple uses over time or many consumers. There is currently no recommendation for this form of encryption.

The specific encoding of messages will have security implications. [RFC8609] uses a type-length-value (TLV) encoding. We chose to compromise between extensibility and unambiguous encodings of types and lengths. Some TLV encodings use variable-length "T" and variable-length "L" fields to accommodate a wide gamut of values while trying to be byte efficient. Our TLV encoding uses a fixed-length 2-byte "T" and 2-byte "L". Using a fixed-length "T" and "L" field solves two problems. The first is aliases. If one is able to encode the same value, such as %x02 and %x0002, in different byte lengths, then one must decide if they mean the same thing, if they are different, or if one is illegal. If they are different, then one must always compare on the buffers, not the integer equivalents. If one is illegal, then one must validate the TLV encoding, every field of every packet at every hop. If they are the same, then one has the second problem: how to specify packet filters. For example, if a name has 6 name components, then there are 7 T's and 7 L's, each of which might have up to 4 representations of the same value. That would be 14 fields with 4 encodings each, or 1001 combinations. It also means that one cannot compare, for example, a name via a memory function as one needs to consider that any embedded "T" or "L" might have a different format.

The Interest Return message has no authenticator from the previous hop. Therefore, the payload of the Interest Return should only be used locally to match an Interest. A node should never forward that

Interest Payload as an Interest. It should also verify that it sent the Interest in the Interest Return to that node and not allow anyone to negate Interest messages.

Caching nodes must take caution when processing Content Objects. It is essential that the Content Store obey the rules outlined in Section 2.4.3 to avoid certain types of attacks. CCNx 1.0 has no mechanism to work around an undesired result from the network (there are no "excludes"), so if a cache becomes poisoned with bad content, it might cause problems retrieving content. There are three types of access to content from a Content Store: unrestricted, signature restricted, and hash restricted. If an Interest has no restrictions, then the requester is not particular about what they get back, so any matching cached object is OK. In the hash-restricted case, the requester is very specific about what they want and the Content Store (and every forward hop) can easily verify that the content matches the request. In the signature-restricted case (often used for initial manifest discovery), the requester only knows the KeyId that signed the content. It is this case that requires the closest attention in the Content Store to avoid amplifying bad data. The Content Store must only respond with a Content Object if it can verify the signature; this means either the Content Object carries the public key inside it or the Interest carries the public key in addition to the KeyId. If that is not the case, then the Content Store should treat the Interest as a cache miss and let an endpoint respond.

A user-level cache could perform full signature verification by fetching a public key or certificate according to the KeyLink. That is not, however, a burden we wish to impose on the forwarder. A user-level cache could also rely on out-of-band attestation, such as the cache operator only inserting content that it knows has the correct signature.

The CCNx grammar allows for hash-algorithm agility via the HashType. It specifies a short list of acceptable hash algorithms that should be implemented at each forwarder. Some hash values only apply to end systems, so updating the hash algorithm does not affect forwarders; they would simply match the buffer that includes the type-length-hash buffer. Some fields, such as the ConObjHash, must be verified at each hop, so a forwarder (or related system) must know the hash algorithm; it could cause backward compatibility problems if the hash type is updated. [RFC8609] is the authoritative source for per-field-allowed hash types in that encoding.

A CCNx name uses binary matching whereas a URI uses a case-insensitive hostname. Some systems may also use case-insensitive matching of the URI path to a resource. An implication of this is that human-entered CCNx names will likely have case or non-ASCII symbol mismatches unless one uses a consistent URI normalization to the CCNx name. It also means that an entity that registers a CCNx routable prefix, say "ccnx:/example.com", would need separate registrations for simple variations like "ccnx:/Example.com". Unless this is addressed in URI normalization and routing protocol conventions, there could be phishing attacks.

For a more general introduction to ICN-related security concerns and approaches, see [RFC7927] and [RFC7945].

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

13.2. Informative References

[ace] Shang, W., Yu, Y., Liang, T., Zhang, B., and L. Zhang, "NDN-ACE: Access Control for Constrained Environments over Named Data Networking", NDN Technical Report NDN-0036, December 2015, <http://new.named-data.net/wp-content/uploads/2015/12/ndn-0036-1-ndn-ace.pdf>.

[befrags] Mosko, M. and C. Tschudin, "ICN "Begin-End" Hop by Hop Fragmentation", Work in Progress, draft-mosko-icnrg-beginendfragment-02, December 2016.

[ccn-lite] Tschudin, C., et al., "CCN-lite", University of Basel, 2011-2019, <http://ccn-lite.net>.

[ccnx-ke] Mosko, M., Uzun, E., and C. Wood, "CCNx Key Exchange Protocol Version 1.0", Work in Progress, draft-wood-icnrg-ccnxkeyexchange-02, March 2017.

[ccnx-registry] IANA, "Content-Centric Networking (CCNx)", <https://www.iana.org/assignments/ccnx>.

[ccnx-uri] Mosko, M. and C. Wood, "The CCNx URI Scheme", Work in Progress, draft-mosko-icnrg-ccnxurischeme-01, April 2016.

[chunking] Mosko, M., "CCNx Content Object Chunking", Work in Progress, draft-mosko-icnrg-ccnxchunking-02, June 2016.

[cicn] FD.io, "Community ICN (CICN)", February 2017, <https://wiki.fd.io/index.php?title=Cicn&oldid=7191>.

[dart] Garcia-Luna-Aceves, J. and M. Mirzazad-Barijough, "A Light-Weight Forwarding Plane for Content-Centric Networks", International Conference on Computing, Networking, and Communications (ICNC), DOI 10.1109/ICCNC.2016.7440637, February 2016, <https://arxiv.org/pdf/1603.06044.pdf>.

[eprise-numbers] IANA, "IANA Private Enterprise Numbers", <https://www.iana.org/assignments/enterprise-numbers>.

[esic] Mosko, M. and C. Wood, "Encrypted Sessions In CCNx (ESIC)", Work in Progress, draft-wood-icnrg-esic-01, September 2017.

[flic] Tschudin, C. and C. Wood, "File-Like ICN Collection (FLIC)", Work in Progress, draft-tschudin-icnrg-flic-03, March 2017.

[mobile] Mosko, M., Uzun, E., and C. Wood, "Mobile Sessions in Content-Centric Networks", IFIP Networking Conference (IFIP Networking) and Workshops, DOI 10.23919/IFIPNetworking.2017.8264861, June 2017, <https://dl.ifip.org/db/conf/networking/networking2017/1570334964.pdf>.

[ndn] UCLA, "Named Data Networking", 2019, <https://www.named-data.net>.

[nnc] Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., and R. Braynard, "Networking Named Content", Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, DOI 10.1145/1658939.1658941, December 2009, <https://dx.doi.org/10.1145/1658939.1658941>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC7927] Kutscher, D., Ed., Eum, S., Pentikousis, K., Psaras, I., Corujo, D., Saucez, D., Schmidt, T., and M. Waehlisch, "Information-Centric Networking (ICN) Research Challenges", RFC 7927, DOI 10.17487/RFC7927, July 2016, <https://www.rfc-editor.org/info/rfc7927>.

[RFC7945] Pentikousis, K., Ed., Ohlman, B., Davies, E., Spirou, S., and G. Boggia, "Information-Centric Networking: Evaluation and Security Considerations", RFC 7945, DOI 10.17487/RFC7945, September 2016, <https://www.rfc-editor.org/info/rfc7945>.

[RFC8609] Mosko, M., Solis, I., and C. Wood, "Content-Centric Networking (CCNx) Messages in TLV Format", RFC 8609, DOI 10.17487/RFC8609, July 2019, <https://www.rfc-editor.org/info/rfc8609>.

[selectors] Mosko, M., "CCNx Selector Based Discovery", Work in Progress, draft-mosko-icnrg-selectors-01, May 2019.

[terminology] Wissingh, B., Wood, C., Afanasyev, A., Zhang, L., Oran, D., and C. Tschudin, "Information-Centric Networking (ICN): CCN and NDN Terminology", Work in Progress, draft-irtf-icnrg-terminology-04, June 2019.

[trust] Tschudin, C., Uzun, E., and C. Wood, "Trust in Information-Centric Networking: From Theory to Practice", 25th International Conference on Computer Communication and Networks (ICCCN), DOI 10.1109/ICCCN.2016.7568589, August 2016, <https://doi.org/10.1109/ICCCN.2016.7568589>.

Authors' Addresses

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
United States of America

Phone: +01 650-812-4405
Email: marc.mosko@parc.com

Ignacio Solis
LinkedIn
Mountain View, California 94043
United States of America

Email: nsolis@linkedin.com

Christopher A. Wood
University of California Irvine
Irvine, California 92697
United States of America

Phone: +01 315-806-5939
Email: woodc1@uci.edu