CCNx Semantics
draft-irtf-icnrg-ccnxsemantics-03

Abstract

This document describes the core concepts of the CCNx architecture and presents a minimum network protocol based on two messages: Interest, Content Object. 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 InterestReturn, whereby one system can return an Interest message to the previous hop due to an error condition. It indicates to the previous hop that the current system will not respond to the Interest.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes the principles of the CCNx architecture. It describes the network protocol based on two message types: Interests and Content Objects. The description is not dependent on a specific wire format or particular encodings. This section introduces the main concepts of CCNx, which are further elaborated in the remainder of the document.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.) [RFC2119].

1.2. Protocol Overview

CCNx is a request/response protocol to fetch chunks of data based on a name. Each chunk of data may be directly signed (e.g. RSA, EC) or protected by a message integrity check (MIC) or message authentication check (MAC), or indirectly protected via hash chains, or even unprotected. The identifying name is hierarchical and includes both a routable prefix and trailing application-dependent data. Because each chunk (or larger indirectly protected block) may provenance from a signature or MAC, we no longer need to rely on host identities, such as derived from TLS certificates, for data provenance. In this sense, CCNx fundamentally protects the data; it does not rely on the pipe. There are several options for data confidentiality, discussed later.

As a request/response protocol, CCNx may be carried over many different transports. In use today are Ethernet, TCP, UDP, 802.15.4, GTP, GRE, DTLS, TLS, and others. While the specific wire format of CCNx may vary to some extent based on transport, the core principles and behaviors of CCNx outlined in this document should remain fixed.

CCNx uses subjective names to identify bytes of payload. The Name combines a routable prefix with an arbitrary suffix assigned by the publisher to a piece of content. The result is a "named payload". This is different from other systems that use only self-certifying names, where the payload name is intrinsically derivable from the payload or its realization in a network object (e.g. a SHA-256 hash of the payload or network object). In human-readable form, we represent names as a ccnx: (Mosko, M. and C. Wood, “The CCNx URI Scheme (Internet draft),” 2016.) [CCNxURI] scheme URI [RFC3986] (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.), though the canonical encoding should be octet strings. In this respect, we speak of a name being made up of hierarchical path segments, which is the URI terminology.

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) in to 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 bound to a fixed payload via cryptographic operations. This implies that the fact that a given publisher bound a certain subjective name to a certain payload can be verified via cryptographic means. For example, a publisher could use a cryptographic hash over the name and payload, sign the hash, and deliver the tuple {Name, Payload, Validation}. Additional information would be included as needed by specific validation mechanisms. Therefore, we divide Validation in to a ValidationAlgorithm and a ValidationPayload. The ValidationAlgorithm has information about the crypto suite and parameters. In particular, the ValidationAlgorithm usually has a field called KeyId which identifies the public key used by the validation, when applicable. The ValidationPayload is the output of the validation algorithm, such as a CRC value, an HMAC output, or an RSA signature.

In addition to the essential Name, Payload, and Validation sections, a CCNx user may need to include some other signaling information. This could include a hint about the type of Payload (e.g. application data, a cryptographic key, etc.) or cache control directives, etc. We will call this extra signaling information ExtraFields.

A Named Payload is thus the tuple {{Name, ExtraFields, Payload, ValidationAlgorithm}, ValidationPayload}, where all fields in the inner tuple are covered by the validation algorithm.

CCNx specifies a network protocol around Interests (request messages) and Content Objects (response messages) to move named payloads. An Interest includes the Name -- which identifies the desired response -- and two optional limiting restrictions. The first restriction on the KeyId and it limits responses to those signed with a ValidationAlgorithm KeyId field equal to the restriction. The second is the ContentObjectHash restriction. It limits the response to one where the cryptographic hash of the entire named payload is equal to the restriction.

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 path 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 (non-hierarchical) 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 Forwarding Information Base (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 PIT table is not mandated by the specification.

The notional PIT table stores the last hop of an Interest plus its Name and optional restrictions. This is the data required to match a Content Object to an Interest (see Section 10 (Interest to Content Object matching)). 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.

If multiple Interests with the same tuple {Name, KeyIdRestriction, ContentObjectHashRestriction} 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 aggregated (see Section 5.1 (Interest Aggregation)). Thus, one Content Object might satisfy multiple pending Interests.

In CCNx, higher-layer protocols often become so-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 InterestReturn. 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. Some options may use the namespace while others may use a structure such as a Manifest. This document does not address these options at this time.

The remainder of this document describes a named payload and the Interest/Content Object network protocol behavior in detail.

TOC

2. Protocol

CCNx is a request/response protocol, where a request is called an Interest and a response is called a ContentObject. CCNx also uses a 1-hop control message called InterestReturn. These are, as a group, called CCNx Messages.

TOC

2.1. Message Grammar

The CCNx message ABNF (Crocker, D., Ed. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 2008.) [RFC5234] grammar is show 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.

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 6.1 (Cache Control)).
- 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 6.2 (Content Object Hash)).
- ExpiryTime: An absolute time after which the content object should be considered expired (see Section 6.1 (Cache Control)).
- InterestField: These are optional fields that may appear in an Interest message.
- KeyIdRestr: The KeyId Restriction. A Content Object must have a KeyId with the same value as the restriction.
- ObjHashRestr: The Content Object Hash Restriction. A content object must hash to the same value as the restriction using the same HashType. The ObjHashRestr MUST use SHA256-32.
- KeyId: An identifier for the key used in the ValidationAlg. For public key systems, this should be the SHA-256 hash of the public key. For symmetric key systems, it should be an identifer agreed upon by the parties.
- KeyLink: A Link (see Section 7 (Link)) that names how to retrieve the key used to verify the ValidationPayload. A message SHOULD NOT have both a KeyLink and a PublicKey.
- Lifetime: The approximate time during which a requester is willing to wait for a response, usually measured in seconds. It is not strongly to the network round trip time, though it must be larger.
- Name: A name is made up of a non-empty first segment followed by zero or more additional segments, which may be of 0 length. Path segments are opaque octet strings, and are thus case-sensitive if encoding UTF-8. An Interest MUST have a Name. A ContentObject MAY have a Name (see Section 10 (Interest to Content Object matching)).
- Payload: The message's data, as defined by PayloadType.
- PayloadType: The format of the Payload. If missing, assume DataType. DataType means the payload is opaque application bytes. KeyType means the payload is a DER-encoded public key. LinkType means it is a Link (see Section 7 (Link)).
- PublicKey: Some applications may wish to embed the public key used to verify the signature within the message itself. The PublickKey is DER encoded. A message SHOULD NOT have both a KeyLink and a PublicKey.
- RelTime: A relative time, measured in milli-seconds.
- ReturnCode: States the reason an Interest message is being returned to the previous hop (see Section 12.2 (ReturnCode Types)).
- SigTime: The absolute time (UTC milliseconds) when the signature was generated.
- 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.

```
Message        := Interest / ContentObject / InterestReturn
Interest       := [Lifetime] BodyName [Validation]
ContentObject  := [CacheTime / ConObjHash] BodyOptName [Validation]
InterestReturn:= ReturnCode Interest
BodyName       := Name Common
BodyOptName    := [Name] Common
Common         := *Field [Payload]
Validation     := ValidationAlg ValidatonPayload

Name           := FirstSegment *Segment
FirstSegment   := 1* OCTET
Segment        := 0* OCTET

ValidationAlg := RSA-SHA256 HMAC-SHA256 CRC32C
```

```
ValidatonPayload := 1* OCTET
RSA-SHA256     := KeyId [PublicKey] [SigTime] [KeyLink]
HMAC-SHA256    := KeyId [SigTime] [KeyLink]
CRC32C         := [SigTime]

AbsTime        := 8 OCTET ; 64-bit UTC msec since epoch
CacheTime      := AbsTime
ConObjField    := ExpiryTime / PayloadType
ConObjHash     := Hash ; The Content Object Hash
DataType       := "1"
ExpiryTime     := AbsTime
Field          := InterestField / ConObjField
Hash           := HashType 1* OCTET
HashType       := SHA256-32 / SHA512-64 / SHA512-32
InterestField := KeyIdRestr / ObjHashRestr
KeyId          := 1* OCTET ; key identifier
KeyIdRestr     := 1* OCTET
KeyLink        := Link
KeyType        := "2"
Lifetime       := RelTime
Link           := Name [KeyIdResr] [ObjHashRestr]
LinkType       := "3"
ObjHashRestr   := Hash
Payload        := *OCTET
PayloadType    := DataType / KeyType / LinkType
PublicKey      := ; DER-encoded public key
RelTime        := 1* OCTET ; msec
ReturnCode     := ; see Section ZZZ
SigTime        := AbsTime
```

Figure 1

TOC

2.2. Consumer Behavior

To request a piece of content for a given {Name, [KeyIdRest], [ObjHashRestr]}, a consumer creates an Interest message with those values. It MAY add a validation section, typically only a CRC32C. If a consumer uses a MAC or a signature, it SHOULD include a name component to signify the signature and prevent the Interest from being aggregated with other Interests or satisfied by a Content Object that has no relation to the validation.

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

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

The network MAY send to the consumer an InterestReturn 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 ContentObject. The consumer SHOULD:

- Ensure the content object is properly formatted.
- Verify that the returned Name matches a pending request. If the request also had KeyIdRestr and ObjHashRest, it should also validate those properties.
- If the content object is signed, it should cryptographically verify the signature. 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 to not be before the time the Interest was generated.
- If the content object is signed, it 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 [schematized trust].
- It MAY cache the content object for future use, up to the ExpiryTime if present.
- A consumer MAY accept a content object off the wire that is expired. It may happen that a packet expires while in flight, and there is no requirement that forwarders drop expired packets in flight. The only requirement is that content stores or caches or producers not respond with an expired content object.

TOC

2.3. Publisher Behavior

TOC

2.4. Forwarder Behavior

TOC

3. Named Payload

CCNx supports several cryptographic means to bind a Name to a payload. Interest and Content Object messages include a section called the ValidationAlgorithm, which specifies the algorithm to use to verify the binding. Several validation algorithms are supported including specific Message Integrity Checks (MICs), Message Authentication Codes (MACs), and Signature types. These are over specific wire format encodings. Additional schemes could be added in the future. Interest and Content Object messages also include a section called the ValidationPayload which contains the validation output.

The KeyId is an optional field in the ValidationAlgorithm. It is an octet string that identifies the key used to sign the Content Object. It uniquely identifies the publisher. It is similar to a Subject Key Identifier from X509 [RFC 3280, Section 4.2.1.2]. It should be derived from the key used to sign, for example SHA-256 hash of the key. It applies to both public/private key systems and to symmetric key systems using HMAC.

A PublicKeyLocator is an optional field in the ValidationAlgorithm. It may be one of (a) the signer's public key, (b) the signer's certificate, or (c) a KeyName that points to the location of the signer's key or certificate.

A Key inside a PublicKeyLocator is a public key corresponding to the signer's private key. Examples would be PEM or DER encodings. The exact encoding is up to the wire format.

A Certificate is an X.509 certificate of the signer's public key. Examples would be PEM or DER encodings of the certificate. The exact encoding is up to the wire format.

A KeyName is a CCNx Name and optional signer's KeyId of that name's publisher. The CCNx Name points to a Key or Certificate. The KeyName signer KeyId is of the signer of the target name's Content Object, not of the target key or certificate.

A SignatureTime is an optional field in the ValidationAlgorithm. It may be used as part of the signature validation check to ensure the signature was generated around the expected time. In particular, if the public key is conveyed in a certificate with a validity period, the verifying system may enforce that the signature came from a corresponding period. Some verifiers might determine a signature without a SignatureTime to be invalid.

TOC

4. 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.

The name segment labels specified in this document are given in the table below. Name Segment is a general name segment, typically occurring in the routable prefix and user-specified content name. Other segment types are for functional name components that imply a specific purpose.

A forwarding table entry may contain name segments of any type. Routing protocol policy and local system policy may limit what goes into forwarding entries, but there is no restriction at the core level. An Interest routing protocol, for example, may only allow binary name segments. A load balancer or compute cluster may route through additional component types, depending on their services.

Name	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 paths segment-by-segment against its forwarding table to determine a next hop.

TOC

4.1. Name Examples

This section uses the CCNx URI (Mosko, M. and C. Wood, “The CCNx URI Scheme (Internet draft),” 2016.) [CCNxURI] representation of CCNx names .

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

TOC

5. Interests

An Interest is composed of the tuple {Name, Metadata, Payload, Validator}. These fields are defined below. Only the Name is mandatory. Other fields, if missing, should be interpreted as "do not care".

Name is a hierarchical path that identifies the resource. It is matched as described above.

An Interest may also have a Payload which 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 table 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 GUID or a nonce. An optional Metadata field defines the IPID field so other systems could verify the IPID, such as when it is derived from a hash of the payload. No system is required to verify the IPID.

An Interest may contain Validation fields including a ValidationAlgorithm section describing the type of validator to use and the ValidationPayload fields containing the output of the validation. Typically this would only be a MIC - a crc, checksum, or digest.

An Interest contains additional fields with information about the query. Two fields - the KeyIdRestriction and ContentObjectHashRestriction - serve as selectors to help identify the specific Content Object that should be returned.

The Interest Lifetime element specifies the maximum number of milliseconds a Forwarder should retain the Interest in its PIT. A lifetime of "0" means the requester does not expect a response from the Interest - it serves as a type of notification. The lifetime is only a guideline for a Forwarder, which may keep an Interest for a shorter or longer time, based on local conditions and system policy. It may change hop to hop if the Interest is delayed for any signifiant amount of time. It is measured in millisecond resolution, so in fast switching it normally would not need to change. This field does not affect the matching of Content Objects.

The Interest HopLimit element is a counter that is decremented with each hop. It limits the distance an Interest may travel. The node originating the Interest may put in any value - up to the maximum - in network byte order. Each node that receives an Interest with a HopLimit decrements the value upon reception. If the value is 0 after the decrement, the Interest cannot be forwarded off the node. The PIT entry should also track the maximum HopLimit forwarded. If an Interest with a longer HopLimit arrives, it should be forwarded even if it is identical to a previously forwarded Interest.

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. Other implementations may define additional optional headers, for example Nonces for loop detection, headers for Differentiated Services Code Points (DSCP), or Flow Labels.

TOC

5.1. Interest Aggregation

INTERST AGGREGATION RULES

TOC

6. Content Objects

A Content Object is composed of the same tuple as an Interest: {Name, Metadata, Payload, Validator}.

The Name is an optional field that identifies the contents. If the name does not exist, then the ContentObject can only be matched by a ContentObjectHashRestriction.

The Payload of a Content Object holds the upper layer payload. It may be encrypted, based on outside information or a protocol information header.

The optional Metadata PayloadType field identifies the type of Content Object: "DATA" implies an opaque payload; "LINK" is a Content Object with an encoded Link as a payload. "DATA" is the default.

The optional field ExpiryTime is a millisecond timestamp containing the number of milliseconds since the epoch in UTC of when the contents will expire. If it is not present, there is no expiration on the Content Object. The ExpiryTime should be part of the signed information covered by the Validator, if present.

A publisher or upstream node may include a Recommended Cache Time for Content Objects. It is represented as the number of milliseconds since the epoch in UTC as the desired minimum time to keep the content object in cache. This recommendation is a guideline as to the useful lifetime of a Content Object, but may be ignored. The RecommendedCacheTime should not be covered by the Validator, if present, as nodes may change the value based on their caching. The ExpiryTime takes precedence over the RecommendedCacheTime, if both exist.

Other protocols, such as versioning or chunking, could place other kinds of metadata in the Content Object.

TOC

6.1. Cache Control

CACHE CONTROL RULES

TOC

6.2. Content Object Hash

CCNx allows an Interest to restrict a response to a specific hash. The hash covers the message body and the validation section, thus it is specific to a given signature. Because it is part of the matching rules (see Section 10 (Interest to Content Object matching)), 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 SHOULD only be used within a trusted domain. The header SHOULD be removed when crossing a trust boundary.

TOC

7. Link

A Link is the tuple {Name, KeyId, ContentObjectHashRestriction}. The information in a Link comprises the fields the fields of an Interest which would retrieve the Link target. A Content Object with PayloadType = "Link" is an object whose payload is a Link. This tuple may be used as a KeyName to identify a specific object with the certificate wrapped key.

TOC

8. Hashes

Several protocol fields use cryptographic hash functions, which must be secure against attack and collisions. Because these hash functions change over time, which better ones appearing and old ones falling victim to attacks, it is important that a CCNx protocol implementation support 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 (Mosko, M. and I. Solis, “CCNx Messages in TLV Format (Internet draft),” 2016.) [CCNMessages] specification should be taken as the definition of acceptable hash functions and uses.

TOC

9. Validation

TOC

9.1. Validation Algorithm

The Validator consists of a ValidationAlgorithm that specifies how to verify the message and a ValidationPayload containing the validation output. 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. The ValidationPayload is the actual cryptographic bytes, such as a CRC value or an HMAC value or a signature value.

Some Validators contain a KeyId, identifying the publisher authenticating the Content Object. If an Interest carries a KeyIdRestriction, then that KeyIdRestriction MUST exactly match the Content Object's KeyId.

Validation Algorithms fall into three categories: MICs, MACs, and Signatures. Validators using MIC algorithms do not need to provide any additional information; they may be computed and verified based only on the algorithm (e.g. CRC32C). MAC validators require the use of a KeyId identifying the secret key used by the authenticator. Because MACs are usually used between two parties that have already exchanged secret keys via a key exchange protocol, the KeyId may be any agreed-upon value to identify which key is used. Signature validators use public key cryptography such as RSA, DSA, or Elliptical Curve (EC). The KeyId field in the ValidationAlgorithm identifies the public key used to verify the signature. A signature may optionally include a KeyLocator, as described above, to bundle a Key or Certificate or KeyName. MAC and Signature validators may also include a SignatureTime, as described above.

A PublicKeyLocator KeyName points to a Content Object with an X509 certificate in the payload. In this case, the target KeyId must equal the first object's KeyId. The target KeyLocator must include the public key corresponding to the KeyId. That key must validate the target Signature. The payload is an X.509 certificate whose public key must match the target KeyLocator's key. It must be issued by a trusted authority, preferably specifying the valid namespace of the key in the distinguished name.

TOC

10. Interest to Content Object matching

A Content Object satisfies an Interest if and only if (a) the Content Object name, if given, exactly matches the Interest name, and (b) the ValidationAlgorithm KeyId of the Content Object exactly equals the Interest KeyIdRestriction, if given, and (c) the computed ContentObjectHash exactly equals the Interest ContentObjectHashRestriction, if given.

The matching rules are given by this predicate, which if it evaluates true means the ContentObject matches the Interest. Ni = Name in Interest (may not be empty), Ki = KeyIdRestriction in the interest (may be empty), Hi = ContentObjectHashRestriction in Interest (may be empty). Likewise, No, Ko, Ho are those properties in the ContentObject, where No and Ko may be empty; Ho always exists. 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 ContentObjectHashRestriction in the Interest specifies an unsupported hash algorithm, then no ContentObject can match the Interest so the system should drop the Interest and MAY send an InterestReturn 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 ContentObject while the next two terms depend on the existence of the attribute in the Interest. The last term is the Nameless Object restriction 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 ContentObjectHash as an expressed field, it must be calculated in network to match against. It is sufficient within an autonomous system to calculate a ContentObjectHash 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 KeyIdRestriction.

TOC

11. Request/Response Protocol

As an Interest moves through the network following the FIB table based on longest matching prefix, it leaves state at each forwarding node. The state is represented in a notional Pending Interest Table (PIT). The PIT tracks the Name, KeyIdRestriction, and ContentObjectHashRestriction to be matched by a Content Object. If a second Interest arrives with the same Name and selector values, it may be aggregated with the existing pending Interest. If the second Interest extends the Lifetime of the pending Interest, it should be forwarded to extend the life of downstream Interests.

When a Content Object arrives at a Forwarder, it is matched against the Interests in the PIT. For each matching Interest, the Content Object is forwarded along the reverse path of that PIT entry and the PIT entry is removed. A Content Object that does not match any Interest in the PIT is dropped.

A Forwarder may implement a Content Object cache called a Content Store. At the core protocol level, the Content Store must obey similar rules as the Forwarder. If an Interest specifies a ContentObjectHashRestriction, the Content Store SHOULD NOT respond unless it has verified the hash of the Content Object. If the Interest carries a KeyIdRestriction then the Content Store MUST cryptographically verify the signature or not respond.

Note that performing digital signature verification in response to an Interest is a vector for Denial of Service (DoS). Two cooperating nodes surrounding a forwarder may maliciously request and then respond with fake content to (a) populate the cache with fake content and (b) force the forwarder to perform wasteful verification operations. Implementations that support signature verification must consider the ramifications of such on-path attacks.

TODO: Specify what to do in case of failure.

TOC

12. 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 InterestReturn 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 [CCNMessages] (Mosko, M. and I. Solis, “CCNx Messages in TLV Format (Internet draft),” 2016.).

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 InterestReturn may take corrective actions or may propagate its own InterestReturn to previous hops as indicated in the reverse path of a PIT entry.

TOC

12.1. 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 InterestReturn and the reserved byte in the Interest header is used as a Return Code. The numeric values for the PacketType and ReturnCodes are in [CCNMessages] (Mosko, M. and I. Solis, “CCNx Messages in TLV Format (Internet draft),” 2016.).

TOC

12.2. ReturnCode Types

This section defines the InterestReturn ReturnCode introduced in this RFC. The numeric values used in the packet are defined in [CCNMessages] (Mosko, M. and I. Solis, “CCNx Messages in TLV Format (Internet draft),” 2016.).

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

Table 3: Interest Return Reason Codes

TOC

12.3. 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 same same path multiple times. For example, it could keep track of which next hops it has tried and not re-use them.

If a forwarder tries an alternate route, it may receive a second InterestReturn, possibly of a different type than the first InterestReturn. 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. 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.

TOC

12.3.1. 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.

TOC

12.3.2. 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.

TOC

12.3.3. 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" is SHOULD NOT try alternate paths. It SHOULD propagate the Interest Return to its previous hops.

TOC

12.3.4. No Resources

If a Forwarder receives an Interest and it cannot process the Interest due to lack of resources, it MAY send an InterestReturn. A lack of resources could be the PIT table is too large, or some other capacity limit.

TOC

12.3.5. 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.

TOC

12.3.6. Prohibited

A forwarder may have administrative policies, such as access control lists, that prohibit receiving or forwarding an Interest. If a forwarder discards an Interest due to a policy, it MAY send a Prohibited InterestReturn to the previous hop. For example, if there is an ACL that says /parc/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.

12.3.7. Congestion

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

12.3.8. 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 InterestReturn to the previous hop.

12.3.9. Malformed Interest

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

13. Acknowledgements

14. IANA Considerations

This memo includes no request to IANA.

15. Security Considerations

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 send the Interest in the Interest Return to that node and not allow anyone to negate Interest messages.

If two peers require authenticated messaging, they must use an external mechanism.

TOC

16. References

TOC

16.1. 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.

TOC

16.2. Informative References

[CCN] PARC, Inc., “CCNx Open Source,” 2007.

[CCNMessages] Mosko, M. and I. Solis, “CCNx Messages in TLV Format (Internet draft),” 2016.

[CCNxURI] Mosko, M. and C. Wood, “The CCNx URI Scheme (Internet draft),” 2016.

[RFC3552] Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005.

[RFC5234] Crocker, D., Ed. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008.

TOC

Authors' Addresses

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
USA
Phone: +01 650-812-4405
Email: marc.mosko@parc.com

Ignacio Solis
TBD
TBD, California TBD

USA
Email: isolis@igso.net

Christopher Wood
PARC, Inc.
Palo Alto, California 94304
USA
Email: christopher.wood@parc.com