April 14, 2014 (rev 3)
PARC

CCNx 1.0 Protocol Specification Roadmap

Marc Mosko[1]*

Abstract
The CCNx protocol specification has developed considerably since its early conceptions. A new framing of the protocols results in a layered approach, with individual CCNx related protocols specified in separate RFC-style documents such that a protocol developer or implementer may work on specific protocols. We specify a minimal core set of protocols necessary to realize CCNx via Interests and Content Objects, then add functionality to that core via individual protocol specification documents. This roadmap describes the new protocol architecture and directs the interested reader to specific documents that describe, in detail, the CCNx approach to information centric networking.

Keywords
Content Centric Networks

[1] *Palo Alto Research Center*
***Corresponding author**: marc.mosko@parc.com

Contents

Introduction

The CCNx 1.0 protocols build on a layered approach to protocol specification and construction. The specifications begin with a core description of what is the essence of CCNx, then build upon that to realize a concrete network protocol and to include value-add protocols on top of the minimum specification. The guiding approach is that each document introduce only the minimum necessary functionality to achieve its goal, and build upon earlier document specifications without breaking those specifications.

The result of this approach is that the minimum core set of protocols and specifications are a subset of the original vision [1]. The original CCNx protocols may be built on top of the new core, and in fact many of the higher-level protocols specified in the new layered approach find their roots in those original ideas. The new higher-level protocols, however, have a concrete unambiguous specification, with their concerns separated from other concerns.

This document describes the protocol architecture and directs the reader to specific RFC-style documents with detailed descriptions of individual protocol components.

1. Protocol Layering

This section describes the CCNx 1.0 protocol layering. Not all layers or services described in this section have a full specification yet. Section 2 describes the current set of specifications.

Core Service Profile

1. Interest Name equals Content Object Name.
2. Interest KeyId equals Content Object KeyId.
3. Interest ContentObjectHash equals computed hash of the Content Object.

The CCNx protocols use a request message called an Interest to retrieve payload encapsulated in a Content Object response message. An Interest carries a Name, an optional publisher identifier called the KeyId, and an optional inescapable identity called a ContentObjectHash, which is a secure cryptographic hash of a Content Object message. It includes two operators, Equals and ComputeContentObjectHash, in addition to standard asynchronous message passing primitives. These three variables and two operators are the entirety of the core protocols.

The core set of protocols use exact matching. Content Object names must exactly match an Interest name. If an Interest specified a KeyId for a publisher, that must exactly match the KeyId in the Content Object. If the Interest specifies a ContentObjectHash, then the hash of the Content Object must match the Interest's restriction. Note that equality matching the KeyId is not robust against attacks. One can summarize the service profile of the core protocols as "The network delivers what is requested."

A content store (cache) operating on the core protocols must behave in accordance with those protocols. It must only return content items based on Equals. If a request contains a ContentObjectHash, it must use ComputeContentObjectHash to apply an Equals check. Higher-level caches, that operate on a specific discovery protocol, may use other queries and responses.

Higher-layer protocols add services to the core protocols. Three important higher-layer protocols enable discovery of services, content, and catalogs so an application may request specific content objects using the core protocol.

Discovery Service Profile

1. Service Discovery
2. Content Discovery
3. Secure Catalogs

Service discovery, for example, provides the list of available printers, from which a user may request print services. Content Discovery, between participating caches, enables a user to find distributed content – with CCNx security – without needing connectivity to an authoritative source. Secure catalogs provide name resolution to specific Content Objects so a user can request very specific responses from the network.

The core protocols do not include exclusions. It is up to higher-layer protocols to have a method to work around content they do not like. We propose service and content discovery protocols with methods to work around incorrect content in separate documents. At the core protocol level, exclusions or workarounds for incorrect content must fit into differences in the Name, KeyId, or ContentObjectHash. This approach leaves ample room for experimentation and development of protocols with better performance than the piece-wise linear exclusions as were specified in the previous protocols, such as work done at UCLA [2]. Using other methods, a node requesting a secure catalog could append a nonce to the catalog name causing the request to go to an authoritative source. In peer-to-peer environments, a Content Discovery protocol could have its own name semantics to allow participants to discover the name and ContentObjectHash in near-by caches. A secure network or military network could use an authenticated discovery protocol.

2. Protocol Architecture Documents

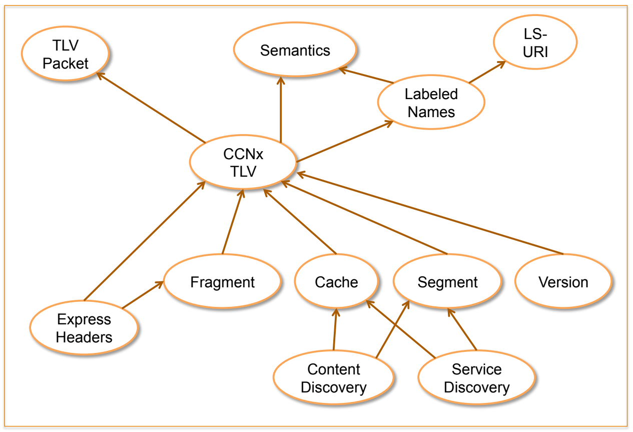

Figure 1. Document Dependency Diagram

The specific higher-layer services available on a device may vary based on the device's capabilities and user configuration. A small sensor, for example, might only implement the core services plus a basic service discovery agent. A high-end router might implement many discovery protocols, caching protocols, routing protocols and enhanced forwarding services beyond simple longest-matching prefix, such as service load balancing.

In addition to the Discovery Services, many other protocols provide valuable additional services. These include, but are not limited to, versioning, large object segmentation, network MTU fragmentation, network-level cache control, and switched forwarding over high-speed cores. Other important network services include routing, trust management, long-term content repositories.

As shown in Fig. 1, there are three top-level protocol documents. In the top-left is the "TLV Packet" which is a new general packet format based on a fixed header, optional TLV headers, and a CCNx message. The top right is "LS-URI" which is a new URI convention for CCNx names that captures the new inherent labels (a little like command markers) of each path segment. The top center is “Semantics” which is the new minimum core protocol for CCNx. It has much simpler rules than the original CCNx protocols, which are be built on top of it.

2.1 Core Protocols

The core document for the CCNx 1.0 specification is “CCNx Semantics” [3]. It describes CCNx at the message level, without tying it to a particular wire format encoding. The semantics of CCNx describes each field in the minimum viable protocol and its function within an Interest or Content Object message.

The document “Labeled Segment URIs” [4] specifies how to represent URI path segments where each path segment contains both a label, which describes the functionality of that path segment, and a value assigned to that label in that path segment. This general specification is the basis for a CCNx specific specification of CCNx URIs.

In “CCNx Labeled Names” [5], we describe the new CCNx name format. This format includes a label specifier for each path segment. The label describes the purpose of that path segment, and is similar to the earlier idea of command markers, except now every path segment has a marker denoting its purpose in the name.

Although CCNB is still a viable network wire format for CCNx, we now specify a TLV scheme as the canonical encoding of a CCNx 1.0 wire format packet. The document “TLV Packet Format” [6] describes the new format as a fixed header and optional network-layer TLV headers, followed by a protocol message. This document does not specify the encoding of an Interest or Content Object, only the overall format of a TLV packet.

The document “CCNx Messages in TLV Format” [7] specifies the exact encoding of a CCNx message (as described in CCNx Semantics) in the TLV packet format. It assigns specific TLV types to specific fields, including the path segments from CCNx Labeled Names. The format includes specifications of security field encodings, such as Key, KeyId, Crypto Suite, and Signature.

2.2 Higher-layer Protocols

Previously “bundled” CCNx protocols, such as segmentation and cache control, are now individual protocols specified in their own documents. The following documents specify these higher-level protocols that build upon the core protocols.

A large user payload is split into multiple Content Objects following the “CCNx Segmentation” [8] protocol. This specification codifies a specific usage of the original CCNx segmentation and removes some ambiguous uses of certain fields. It specifies a new path segment label for segmentation and some new Content Object types for conveying state about segmentation. It also codifies how to write metadata about the segmentation to additional Content Objects related to the the segmented object.

The document “CCNx Content Object Caching” [9] specifies cache control directives that may appear in a Content Object, or its network-level headers, and how a compliant network cache should operate. It specifies a content expiry directive. The content expiry directive puts a termination time on a content object being served from cache.

Because content objects may still be bigger than a network media MTU, we need a fragmentation protocol to encapsulate CCNx messages outside of IP fragmentation. The document “CCNx End-to-End Fragmentation” [10] describes one method of achieving end-to-end fragmentation. This fragmentation scheme does not attempt per-fragment security.

The current versioning scheme for content objects, which is based on the publisher’s clock used as a timestamp in the name, is codified in “CCNx Publisher Time Versioning” [11]. A second versioning scheme, based on sequential numbers assigned by the publisher, is specified in “CCNx Publisher Serial Number Versioning.” [12].

A fast label-switched forwarding technique is described in “CCNx Hash Forwarding” [13]. This technique pre-computes a matching label, like the Similarity Hash, and a Forwarding Hash. An Interest flows through the network based on those values, not the complete name.

The document “CCNx Selector Based Discovery” [14] describes a discovery protocol based on the previous Interest search mechanisms of Interest suffix matching and exclusions. It describes how to encode these search parameters into a single name path segment and use that over the simplified core protocols.

3. Conclusion

CCNx 1.0 presents CCNx in an expanded layered model. A core set of protocols needed for a minimum implementation serve as the basis for many higher-layer protocols. Some of the higher-layer protocols are optional, some may become required, as experimentation and time will tell. The new presentation of CCNx allows designers and implementer to study individual protocols in their own detailed specifications and separates concerns.

References

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

[2] Chenni Qian. Building light control over named data networking. Master's thesis, UCLA, 2011.

[3] CCNx semantics. ccnx-mosko-semantics-00.

[4] Labeled segment uris. icnrg-mosko-labelednames-00.

[5] CCNx labeled segment names. ccnx-mosko-labelednames-00.

[6] TLV packet format. icnrg-mosko-tlvpackets-00.

[7] CCNx messages in TLV format. ccnx-mosko-tlvmessages-00.

[8] CCNx segmentation. ccnx-mosko-segmentation-00.

[9] CCNx content object caching. ccnx-mosko-caching-00.

[10] CCNx end-to-end fragmentation. ccnx-mosko-fragmentation-00.

[11] CCNx publisher time versioning. ccnx-mosko-timeversion-00.

[12] CCNx publisher sequence number versioning. ccnx-mosko-serialversion-00.

[13] CCNx hash forwarding. ccnx-mosko-hashforwarding-00.

[14] CCNx selector based discovery. ccnx-mosko-selectors-00.

Table of Contents

CCNx Semantics

Abstract

This document describes the core concepts of CCNx and presents a minimum network protocol based on Interest and Content Object messages. It specifies the behavior and interpretation of the various fields in those messages, independent of a specific wire encoding.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document describes the nature of CCNx, its principles, and its realization in a network protocol based on Interests and Content Objects. The description is not dependent on a specific wire format or particular encodings.

CCNx uses subjective names to identify bytes of payload. Other systems use so-called self-certifying names, where the payload name is intrinsically derivable from the payload or its realization in a network object. For example, one could use a SHA-256 hash of the payload or a SHA-256 hash of the entire network object. Those are self-certiyfing names because they may be verified anywhere using endogenous attributes of the object. The subjective name is an arbitrary name assigned to the payload. It does not need to be a self-certifying name or have any direct relationship to the payload, although one could use a self-certifying name or something that directly describes the payload. The result is a "named payload".

The essence of CCNx is that a subjective name is bound to a fixed payload via cryptographic operations. This means that by some secure computable means, one can verify that a given publisher bound a certain subjective name to a certain payload. One such means is to use a cryptographic hash over the name and payload, then sign the hash and deliver the tuple {name, payload, signature}; there may need to be additional information if one could use different digest and signature algorithms, or to identify the key used to produce the signature. A are realistic named payload is thus {name, payload, name authenticator, signature}.

CCNx specifies a network protocol around request messages, called Interests, and response messages, called Content Objects, to move named payloads. An Interest includes the name of the desired payload and the Content Object response carries a matching name and the specified payload. Matching a Content Object to an Interest is an exact match on the name. The CCNx network protocol of Interests and Content Objects imposes a restriction on names: the name should be hierarchical and is used to route towards an authoritative source. The CCNx name, in such an instance, looks like a URI absolute path, and we use URI terminology to describe the absolute path as made up of path segments.

The hierarchy of a CCN name is used for routing via longest matching prefix in a forwarder. The longest matching prefix is done path segment by path segment in the hierarchical path name, where each path 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) path segment. Some forwarders may use more advanced matching rules that allow longest matching prefix and shorter prefixes.

Another central 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 upstream from it.

As an Interest travels the forward path, following the Forwarding Information Base (FIB), it leaves behind state at each forwarder. This state is called the Pending Interest Table (PIT), which tracks the ingress ports of an Interest and the egress ports of the Interest. It must store the interest Name, KeyId, and ContentObjectHash. When a Content Object arrives, it is matched against that tuple to see if it satisfies any Interests. If it does, it is returend along the Interest reverse path. If a Content Object does not satisfy an Interst, it is dropped.

If multiple interests with the same tuple {Name, KeyId, ContentObjectHash} arrive before a matching Content Object comes back, they are grouped in the same PIT entry and their reverse paths aggregated. Thus, one Content Object may satisfy multiple pending Interests.

Because multiple publishers could issue Content Objects with the same name, additional optional attributes in the Interest select between multiple matching names. An Interest, in addition to the required Name, also carries an optional KeyId and an optional Self-Certified Name. If the KeyId is present, a forwarder should ensure the responding Content Object has the same KeyId. If a self-certified name is present, a forwarder should match those names.

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

The remainder of this document describes named payload, and the Interest/Content Object network protocol behavior in detail. It does not tie named payload or the Interest/Content Object protocol to a specific network encoding, such as CCNB or TLV. Additional outside specifications describe network encoding, packet formats, and higher-layer protocols built on top of the core Interest/Content Object protocol. The existing ccnx.org interest/object protocols based on exclusions, min/max suffix components, and other selectors may be built on top of the minimum protocol specified here.

This document is supplemented by these documents:

- CCNx Messages in TLV format: see [ietf-draft-ccnxmessages-mosko-03].
- Fragmentation: see [ietf-draft-ccnxfagment-mosko-01] for an end-to-end fragmentation protocol.
- Object Segmentation: see [ietf-draft-ccnxsegment-mosko-01] for object segmentation protocol.
- Object Caching: see [ietf-draft-ccnxcaching-mosko-01] for object caching.
- URI Name format: see [ietf-draft-labeledcontent-mosko-01].

TOC

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].

TOC

2. Named Payload

CCNx supports several cryptographic means to bind a name to payload. One method is to use a public key signature. Another is to use HMAC and shared secrets. Other schemes could be added in the future. Signatures are over specific wire format encodings.

A Content Object contains a section called the Name Authenticator. This section contains a mandatory KeyId and mandatory Crypto Suite specifier. It may also contain a Key Locator that aids in finding the key associated with the KeyId. One type of Key Locator is a Public Key Locator to identify public keys. Other types of Key Locators could include a secure key distribution protocol for symmetric HMAC keys.

The KeyId is an octet string that identifies the key used to sign the content object. 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, such as from the SHA-256 hash of the key. It applies to public/private key systems and to symmetric key systems using HMAC.

The Crypto Suite identifies the digest algorithm used to digest the content object (everything except the signature block) for purposes of signing. It also identifies the algorithm used to sign the digest. The crypto suite specifies both the signature hash function and the signing algorithm. An example would be "RSA Signing with SHA-256 signature digest."

A Public Key Locator may be one of (a) the signer's public key, (b) the signer's certificate, or (c) a KeyName that points to where the signer's key is.

A Key inside a Public Key Locator 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 Key Id of that name's publisher. The name points to a Key or Certificate. The KeyName signer key id is of the signer of the target name's content object, not of the target key or certificate.

TOC

3. Names

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

The path segment labels specified in this document are given in the table below. UTF-8 Segment and Binary Segment are general path segments, 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 path segments of any type. Routing protocol policy and local system policy may limit what goes in to forwarding entries, but there is no restriction at the core level. An Interest routing protocol, for example, may only allow UTF-8 segments, or a mix of UTF-8 and binary path segments. A load balancer or compute cluster may route through additional component types, depending on their services.

Name	Description
UTF-8 Segment	A generic path segment that complies to UTF-8 encoding. Meant for human display.
Binary Segment	A generic path segment that includes arbitrary octets.
Nonce Segment	A nonce in a path segment, which is often used by name-based protocols to create unique Interest requests.
KeyId	A KeyId, as used in Interests or Content Objects, typically a SHA-256 hash of the key.
Content Object Hash	The Content Object Hash path segment specifies the self-certifying name of a content object based on the entire object hash.
Payload Hash	The Payload Hash name component specifies the self-certifying name of the payload of a content object.
Application Components	Application-specific payload in a path 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 Types

At the lowest level, a forwarder does not need to understand the semantics of path segments; it must only identify path segment boundaries and be able to compare two path segments for equality, which includes label and value equality. The forwarder matches paths segment-by-segment against its forwarding table to determine a next hop.

TOC

4. Interests

An Interest is composed of the tuple {Name, KeyId, ContentObjectHash, Scope, Lifetime}. 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.

The KeyId restriction specifies the key used to sign a content object.

ContentObjectHash is a restriction that specifies a self-certified name of the content object that matches the Interest.

Scope determines how far an Interest should propagate. The choices are "localhost", "1-hop neighborhood", and "unlimited". This field does not affect matching of content objects.

The Lifetime 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 is, for example, 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. This field does not affect the matching of content objects.

TOC

5. Content Object

A Content Object is the tuple {Name, Name Authenticator, Protocol Information, Payload, Signature Block}.

The Name is a mandatory field that identifies the contents.

TOC

5.1. Name Authenticator

The Name Authenticator is a mandatory field. It contains two mandatory fields, the signer's KeyId and the Crypto Suite. It may contain an optional Key Locator.

A Public Key Locator provides information about the key identified in the KeyId. It may (a) include the actual key, (b) include a certification with the actual key, or (c) provide one step of indirection to an outside content object with the key or certificate.

Other types of key locators, for example a secret sharing protocol for symmetric keys, may be introduced in the future.

TOC

5.2. Protocol Information

The Protocol Information is a optional structured container for information about the content object. It includes a place for name-based protocol to store state that does not belong in the name. An optional member is Content Information. Other protocols, such as versioning or segmentation, could place data in their own containers.

TOC

5.3. Payload

The Payload of a content object is higher-level payload. It may or may not be encrypted, based on outside information or a protocol information header.

TOC

5.4. Signature Block

The Signature Block contains the cryptographic signature of the content object. It contains the single field called Signature Octets.

The Signature Octets is an opaque octet string of the cryptographic signature.

TOC

6. Interest to Content Object matching

In an Interest, the fields Name, KeyId, and ContentObjectHash, determine which content objects match the Interest. If the KeyId or ContentObjectHash is missing, then any Content Object matches that missing field. It is a necessary condition to exactly match all present fields. A Content Object does not carry the ContentObjectHash as an expressed field, it must be calculated in network to match against it.

It is sufficient within an autonomous system to calculate a ContentObjectHash at a boarder router and carry it in a trusted means within the autonomous system.

An Interest cannot be used to enumerate or iterate over the contents of a Content Store by specifying an indefinite prefix that does not identify a particular content object.

It may be common for an Interest to include name components that will match forwarding entries to index services, search services, or content repositories. These network services may apply different matching rules than the forwarder and return information appropriate for their service.

TOC

7. KeyName Target Objects

A KeyName in a first Content Object is the CCNx name of a Content Object that contains the key or certificate that authenticates the first object's signature. We call the object pointed to by the KeyName the target object.

The first type of KeyName target is a content object that contains the public key that signed the first object. The target object contains the following fields of interest: {target KeyId, target key locator, target payload, target signature}. The target object must be verified via the target signature using the key corresponding to the target KeyId. Here are the supported cases:

1. The target KeyId equals the first object KeyId, and the target payload contains the key that verifies both the target signature and the first object signature. If such a target contains a KeyLocator, it must include the same Key as in the payload.
2. The target KeyId does not equal the first object KeyId. It MAY include a KeyLocator with the Key corresponding to the target KeyId. It MUST NOT have a KeyName. If it does not include a Key Locator, then the verifier must know the target KeyId. If the target object verifies, then the verifier should look at the enclosed key in the payload. The SHA-256 hash of the payload key must match the first object's KeyId.

The second type of KeyName target is 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 key locator 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 key locator's key. It must be issued by a trusted authority, preferably specifying the valid namespace of the key in the distinguished name.

TOC

8. Protocol Behavior

As an Interest moves through the network, it leaves state at each forwarding node. The state is represented in a Pending Interest Table (PIT). The PIT tracks the Name, KeyId, and ContentObjectHash to be matched by a Content Object. If a second Interest arrives with the same name and constraints, it is 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. If an Interest has a Scope, it should not be forwarded outside the scope constraints.

If a second similar interest arrives at a PIT and it has a larger scope than the prvious Interest, the Interest with the larger scope should be forwarded.

When a Content Object arrives at a forwarder, it is matched against 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 a PIT entry is dropped.

A forwarder may implemenent a Content Object cache, called the Content Store. At the core protocol level, the Content Store should obey similar rules as the forwarder. If an Interest specifies a ContentObjectHash, the Content Store should not respond unless it has verified the hash of a Content Object. The verification of a Content Object hash should be done locally. It should match names and KeyId with equality. More advanced systems could implement discovery protocols and offer a richer access method to the Content Store or other object respositories.

If the issuer of an Interest receives a Content Object that matches a pending interest, but in some respect does not satisfy the interest, the issuer must take corrective action. The Content Object can fail to satisfy a matching Interest because (a) the signature is invalid, or (b) the ContentObjectHash does not self-certify. Failure (a) can happen because signatures are not checked in-network. Failure (b) can happened because some forwarder either did not compute it correctly, did not check, or is malicious.

The issuer of an Interest may use several strategies to work around a failure. If the Interest included a ContentObjectHash and the node received a content object that did not match, it is likely there is a malicious node somewhere in the forwarding path. The issuer should attempt re-expressing the Interest. Correctly behaving forwarders should not reply to a ContentObjectHash restriction with an incorrect Content Object. If the issuer of an Interest receives a Content Object with a signature failure, but a matching keyid, then there is likely a malicious node injecting incorrectly signed Content Objects. The issuer should re-express its Interest, perhaps changing the name in the interest to avoid the incorrect content.

We expect that higher-level protocols should incorporate methods for nodes to work around incorrect content. The core protocols do not allow for exclusions, like in the original protocols. Exclusions are not scalable under an attack, as an attack could generate an endless number of objects to exclude.

TOC

9. Acknowledgements

TOC

10. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

11. Security Considerations

We do not address trusting a key, only if a key mechanically verifies a signature.

TOC

12. References

TOC

12.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

12.2. Informative References

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

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

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014
Expires: October 15, 2014	

TLV Packet Format

Abstract

This document defines a general format for network packets that uses Type-Length-Value (TLV) encoding. It consists of an fixed header that defines the version and type of message, a set of optional fields and a payload. Specific protocols that want to make use of this format should be defined in a separate document.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document describes a TLV-based general packet format that can be used to construct network protocols. The format is suitable for use directly over a MAC layer, or encapsulated within a network or transport protocol.

This document specifies:

- A TLV encoding
- A packet format with mixed fixed headers and TLV fields

The TLV packet format is designed for protocol flexibility. The static header provides imediate access to the basic fields needed for parsing. The optional per-hop header fields can be used to extend the functionality of the static header.

The maximum TLV packet size is 64KiB.

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

TOC

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].

TOC

2. Definitions

- TLV: Type-Length-Value. This is the common used term to refer to the organization of a type field, a length field and the value field. In practice the type is not use purely as a type, it is often treated as a label or tag. The type field normally carries semantic and syntactic meaning. The type of "hop count", for example, implies both the purpose of the field as well as the format of the data (e.g. an unsigned integer)

TOC

3. Type-Length-Value Structure

To encode TLV packets we use a 16-bit type and 16-bit length TLV structure. That gives 64K types with a possible length of 64KiB.

With 64K types there should be sufficient space for the basic protocol while allowing ample room for experimentation, application use and growth. A length field of 16 bits is also sufficient to represent the length for any field carried in a single packet. In the event that more space is needed, either for types or for length then the protocol would need to be versioned.

```
                     1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|             type              |            length             |
+---------------+---------------+---------------+---------------+
```

No types are defined in this document. A protocol that wants to use this format must define the types it needs for operation and their requirements.

The length field is the length of the Value in octets. It does not include the length of the Type and Length fields. A zero length TLV is permissible.

TOC

4. Fixed Headers

CCNx TLV messages begin with a non-TLV header followed by per-hop processed TLVs before the ccnx message. The per-hop processed TLVs are covered in the "hdrlen" field, so to find the beginning of the protocol message TLV, one moves to "packet start + 8 + hdrlen". The quantity "8" skips over the fixed length header.

Signed information or Similarity Hashs should not include any of the fixed header or per-hop TLVs. The content object hash should not include the fixed header or per-hop headers, as those may change hop-to-hop. The payload of a CCNx TLV should stand alone and be self-sufficient if the fixed header and per-hop TLVs are removed.

TOC

4.1. Common Header

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|      ver      |   msg type    |         payload length        |
+---------------+---------------+---------------+---------------+
|           reserved            |         header length         |
+---------------+---------------+---------------+---------------+
```

- ver: defines the version of the packet.
- header length: The length of optional per-hops headers. The minimum value is "0".
- msg type: 0 = object, 1 = interest, 2 = control.
- payload length: Total octets following the header (fixed header plus optional headers).

TOC

4.2. Per-hop TLVs

Between the fixed header and the beginning of the CCNx message are per-hop TLVs. These TLVs follow the normal 16b/16b format of other TLVs. They are for per-hop processed options, such as a DSCP-equivalent field, or perhaps a loop-preventing Nonce. Per-hop TLVs are unordered, and order MUST NOT affect the processing of other per-hop TLVs.

Separate documents define per-hop TLVs and their semantics.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|      ver      |   msg type    |        payload length         |
+---------------+---------------+---------------+---------------+
|           reserved            |         header length         |
+---------------+---------------+---------------+---------------+
/ Optional per-hop TLVs                                         /
+---------------+---------------+---------------+---------------+
/ ...                                                           /
+---------------+---------------+---------------+---------------+
/ Optional per-hop TLVs                                         /
+---------------+---------------+---------------+---------------+
```

TOC

4.3. Message Body

The CCNx message begins immediately after the per-hop headers. The message body is encoded with the same TLV structure as the per-hop headers.

TOC

5. Acknowledgements

TOC

6. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

7. Security Considerations

A payload length of 0 may be valid for some message types whose meaning is derived soley from the fixed header and per-hop headers.

TOC

8. References

TOC

8.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

8.2. Informative References

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

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

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

Marc Mosko (editor)
PARC
Palo Alto, California 94304
USA
Phone: +1 650-812-4405
Email: marc.mosko@parc.com

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014
Expires: October 15, 2014	

CCNx Messages in TLV Format

Abstract

This document specifies the encoding of CCNx messages inside a TLV format. The TLV format follows the TLV Packet specification. CCNx messages follow the CCNx Semantics specification. This document defines the TLV types used by each message element and the encoding of each value.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document specifies the TLV types and the semantics of values for the CCNx protocols operating over CCNx TLV syntax. This draft describes the mandatory and common optional fields of Interests and Content Objects. Several common additional protocols are in use that extend this specification, and specified in their own documents.

CCNx uses two types of messages: Interests and Content Objects. An Interest carries a hierarchically structured variable-length identifier (HSVLI), also called the "name", of a Content Object and serves as a request for that object. If a network element sees multiple interests for the same name, it may aggregate those interests. A network element along the path of the Interest with a matching Content Object may return that object, satisfying the Interest. The Content Object follows the reverse path of the Interest to the origin(s) of the Interest. A Content Object contains the HSVLI, the object's payload, and cryptographic information used to bind the HSVLI to the payload.

A full description of the semantics of CCNx messages, as used in this document, see [ccnx-mosko-semantics], which provides an encoding-free description of CCNx messages and message elements.

In this document, the Type values have not been compacted, so there are some gaps. In a final draft, the type values will be assigned to be compact, with a small number of global values and others being container-specific.

This document specifies:

- The TLV types used by CCNx messages.
- The encoding of values for each type.
- Global types that exist in all containments.
- Top level types that exist at the outermost containment.
- Interest TLVs that exist within Interest containment.
- Content Object TLVs that exist within Content Object containment.

This document is supplemented by these documents:

- Message semantics: see [ccnx-mosko-semantics-00] for the protocol operation regarding messages and message elements.
- Fragmentation: see [ccnx-mosko-fragmentation-00] for an end-to-end fragmentation protocol.
- Object Segmentation: see [ccnx-mosko-segmentation-00] for object segmentation protocol.
- Object Caching: see [ccnx-mosko-cachine-00] for object caching.
- URI Name format: see [ccnx-mosko-labelednames-00].

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

TOC

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].

TOC

2. Definitions

- HSVLI: Hierarchically structured variable length identifier, also called a Name. It is an ordered list of path segments, which may be variable length octet strings. In human-readable form, it is represented in URI format as lci:/path/part. There is no host or query string.
- Name: see HSVLI
- Interest: A request for a Content Object that specifies a HSVLI name and other optional selectors to choose among multiple objects with the same name. Any Content Object whose name matches the Interest name and selectors satisfies the Interest.
- Content Object: A data object sent in response to an Interest. It has a HSVLI name and a Contents payload that are bound together via a cryptographic signature.

3. Type-Length-Value Structure

CCNx over TLV uses the TLV syntax of [mosko-tlvpackets-01]. In particular, there is a fixed header followed by per-hop TLVs, followed by a payload. The payload is a TLV of the encoded CCN message. The first TLV in the message is the Name TLV.

The fixed header and per-hop TLVs may be discarded and the resulting payload should be a valid protocol message. Therefore, the payload always begins with a 4 byte TLV defining the message and its total length.

The embedding of a self-sufficient protocol data unit inside the fixed header and per-hops headers allows a network stack to discard the headers and operate only on the embedded message. The following relations hold for a packet with a single message:

```
payload_length      = message_length + 4
msg_type            = message_type
total packet octets = header length + payload_length
```

It is acceptable to have a 0-length payload, in which case all signaling is done in the fixed header and per-hop headers.

Interest and Content objects begin with a non-TLV header followed by per-hop processed TLVs before the Name. The per-hop processed TLVs are covered in the "header length" field, so to find the beginning of the Name TLV, one moves to "packet start + 8 + header length".

An Interest Similarity Hash begins with the "message type" and ends at the earliest of the tail of the packet, or the beginning of the Interest Lifetime or Interest Nonce.

The hash of a content object for the purposes of the signature begins with the "message type" and ends just before the signature block.

The content object hash begins with the "message type" and ends at the tail of the packet.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|      ver      |   msg type    |        payload length         |
+---------------+---------------+---------------+---------------+
|           reserved            |        header length          |
+---------------+---------------+---------------+---------------+
/ Optional per-hop TLVs                                         /
/                                                               /
/                                                               /
/                                                               /
+---------------+---------------+---------------+---------------+
|         message type          |        message length         |
+---------------+---------------+---------------+---------------+
|            T_NAME             |         name length           |
+---------------+---------------+---------------+---------------+
/                                                               /
/                          name value                           /
/                                                               /
+---------------+---------------+---------------+---------------+
/ Message type specific TLVs                                    /
```

```
+---------------+---------------+---------------+---------------+
```

TOC

4. Global Types

CCNx over TLV uses the following global type values. A global type value is a value that appears at the top level of a message. All other type values are specific to their container.

If a container re-uses a global type value, for example a KeyLocator contains a Name, the same type value MUST be used for a Name, if it has the same semantics as a Name.

Type	Abbrev	Name	Description
%x0000	T_NAME	Name (Content Name)	Content Name, encoded in TLV style [cite]
%x007F	T_PAD	Pad (Pad)	A protocol may chose to word-align fields by using a Pad type.
%xE000 - %xEFFF	T_EXPIMENT	Experimental	Experimental fields

Table 1: CCNx Global Type Values

TOC

5. Top-Level Types

Top-level TLVs exist at the outermost level of a CCNx message in TLV format.

Type	Abbrev	Name	Description
%x0001	T_INTEREST	Interest (Interests)	An Interest protocol message.
%x0002	T_OBJECT	Content Object (Content Objects)	A content object protocol message

Table 2: CCNx Top Level Types

TOC

6. Per-hop TLVs

As stated in the TLV syntax, per-hop TLVs are unordered and no meaning may be attached to their ordering.

Type	Abbrev	Name	Description
%x0001	T_NONCE	Interest Nonce (Interest Nonce)	A large random number used for loop detection.
%x0002	T_HOPLIMIT	Interest Hop Limit (Interest Nonce)	An unsigned integer encoded in 2 bytes that limits the number of hops.

Table 3: CCNx Per-hop Types

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|       ver     |    msg type   |         payload length        |
+---------------+---------------+---------------+---------------+
|             header length     |           reserved            |
+---------------+---------------+---------------+---------------+
/ Optional Interest Nonce TLV                                   /
+---------------------------------------------------------------+
/ Optional Interest Hop Limit TLV                               /
+---------------------------------------------------------------+
```

TOC

6.1. Interest Nonce

A Nonce is used to distinguish unique repetitions of the same Interest. A requester, for example, may issue an Interest, timeout, and then re-issue the same Interest. To indicate that it is not a duplicate the requester uses a different Nonce.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_NONCE            |             length            |
+---------------+---------------+---------------+---------------+
/                                                               /
/                           Nonce (length octets)               /
/                                                               /
+---------------+---------------+---------------+---------------+
```

TOC

6.2. Interest HopLimit

The hop limit is a decremented counter that limits the distance an Interest may travel. The node originating the Interest may put in any value, up the the maximum, in network byte order. Each node that receives an Interest with a hop limit decrements the value on reception. If the value is 0 after decrement, the Interest cannot be forwarded off the system.

It is an error to receive an Interest with a 0 hop-limit.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_HOPLIMIT           |            length             |
+---------------+---------------+---------------+---------------+
|           Hop Limit           |
+---------------+---------------+
```

TOC

7. Global Type Formats

TOC

7.1. Pad

The pad type may be used by protocols that prefer word-aligned data. The size of the word may be defined by the protocol. Padding 4-byte words, for example, would use a 1-byte, 2-byte, and 3-byte length. Padding 8-byte words would use a (0, 1, 2, 3, 5, 6, 7)-byte length.

A pad may be inserted after any other TLV except in a Name. In the remainder of this document, we will not show optional pad TLVs.

Name

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|             T_PAD             |            length             |
+---------------+---------------+---------------+---------------+
/                 variable length pad MUST be zeros             /
+---------------+---------------+---------------+---------------+
```

TOC

7.2. Content Name

The value of the Name is a TLV encoded sequence of name components. A Name MUST NOT include PAD TLVs.

Name

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_NAME             |            length             |
+---------------+---------------+---------------+---------------+
/                                                               /
/                     Variable length path                      /
/                                                               /
+---------------+---------------+---------------+---------------+
```

Short Type	Long Type	Name	Description
%x0001	T_UTF8	UTF-8 encoded path segment (Name Components)	A generic name component that complies with UTF-8 encoding. It specifies the routable prefix and user-specified name components.
%x0002	T_BINARY	Binary path segment (Name Components)	A generic name component with arbitrary octets. It specifies the routable prefix and user-specified name components.
%x0003	T_NAMENONCE	Name Nonce (Name Components)	A nonce name component, promoted out of the %xC1.N%x00 namespace.
%x0004	T_NAMEKEY	Publisher Key Digest (Name Components)	The SHA-256 digest of a cryptographic key that identifies the publisher.
%x0006	T_META	Metadata (Name Components)	A Metadata component specifies that the object named by the succeeding name components represents metadata about the prior name. The value of the metadata name component signifies the type of metadata object, such as Segmentation or Bibliographic or Dublin Core.
%x0007	T_OBJHASH	Content Object Hash (Name Components)	The Content Object Hash name component specifies the self-certifying name of a content object based on the entire object hash.
%x0008	T_PAYLOADHASH	Payload Hash (Name Components)	The Payload Hash name component specifies the self-certifying name of the payload of a content object.
%xF000 - %xF0FF	T_APP	Application Components	Application-specific payload in a name component. An application may apply its own semantics to the

(Name Components)	256 payload components.

Table 4: CCNx Name Types

TOC

7.2.1. Name Components

Special application payload name components are in the range %xF000 - %F0FF. These have application semantics applied to them. A good convention is to put the application's identity in the name prior to using these name components.

For example, a name like "lci:/foo/bar/Nonce=256" would be encoded as:

Name

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            (T_NAME)           |           %x18 (24)           |
+---------------+---------------+---------------+---------------+
|            (T_UTF8)           |           %x03 (3)            |
+---------------+---------------+---------------+---------------+
|       f               o               o       |    (T_NCOMP)
+---------------+---------------+---------------+---------------+
      %x01      |              %x03 (3)         |       b
+---------------+---------------+---------------+---------------+
       a               r        |           (T_NONCE)           |
+---------------+---------------+---------------+---------------+
|           %x02 (2)            |      %x0100  (256)            |
+---------------+---------------+---------------+---------------+
```

TOC

7.2.2. Content Object and Payload Hashes

The Content Object Hash and Payload Hash name components represent a cryptograph hash that self-ceritfies the content object. The content object hash is over the entire content object, including the signature, and uniquely identifies the exact content object. The payload hash is only over the payload the content object.

These hashes are defined as SHA-256. The only accepted length is 32.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|  T_OBJHASH or T_PAYLOADHASH   |             length            |
+---------------+---------------+---------------+---------------+
/                                                               /
/                    Hash value (length octets)                 /
/                                                               /
+---------------+---------------+---------------+---------------+
```

TOC

7.3. Interests

An Interest is a fixed header, optional per-hop headers, an Interest type and length container, followed by the Name. It may then have optional Selectors and Lifetime.

The Interest is organized such that the optional Lifetime is at the end. This allows a similarity check on the bytes up to, but not including the lifetime.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|                          Fixed Header                         |
|                                                               |
+---------------+---------------+---------------+---------------|
/ Per-hop TLVs                                                  /
+---------------+---------------+---------------+---------------+
|           T_INTEREST          |         message length        |
+---------------+---------------+---------------+---------------+
|             T_NAME            |          name length          |
+---------------+---------------+---------------+---------------+
/                                                               /
/                          name value                           /
/                                                               /
+---------------------------------------------------------------+
/ Optional Interest KeyId TLV                                   /
+---------------------------------------------------------------+
/ Optional ContentObjectHash TLV                                /
+---------------------------------------------------------------+
/ Optional Scope TLV                                            /
+---------------------------------------------------------------+
/ Optional AllowedResponseType TLV                              /
+---------------------------------------------------------------+
/ Optional Interest Lifetime TLV                                /
+---------------------------------------------------------------+
```

Type	Abbrev	Name	Description
%x0001	T_KEYID	Signer Key ID (KeyId restriction)	An octet string that identifies the key used to sign the contents.
%x0002	T_OBJHASH	ContentObjectHash (Content Object Hash)	The SHA-256 hash of the Content Object message in TLV format.
%x0003	T_SCOPE	Interest Scope (Interest Scope)	The Scope of an Interest.
%x0004	T_ART	Allowed Response Type (Allowed Response Type)	A bitfield indicating the allowed types of response, such as dynamic content or no cache. (old name

			was Answer Origin Kind)
%x0005	T_INT_LIFE	Interest Lifetime (Interest Lifetime)	Interest Lifetime, in milli-seconds

Table 5: CCNx Interest Types

TOC

7.3.1. Interest Scope

The Interest Scope limits the propagation of an Interest in slightly different ways than a Hop Limit.

0 = Local cache only, 1 = local cache or local apps, 2 = anything local or 1-hop neighbor cache or local apps. Other values are undefined and should be ignored.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_SCOPE            |            length             |
+---------------+---------------+---------------+---------------+
|    Scope      |
+---------------+
```

TOC

7.3.2. Allowed Response Type

The Allowed Response Type in an Interest determines the desired types of responses. It is a binary OR of these flags. If the field is missing, any response type is allowed.

0x00 = Unrestricted, 0x01 = Cached Responses Only, 0x02 = Dynamically generated Only

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|             T_ART             |            length             |
+---------------+---------------+---------------+---------------+
|  AllowedType  |
+---------------+
```

TOC

7.3.3. KeyId restriction

An interest selector may include a KeyId restriction to include only content objects with the matching digest.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_KEYID            |             length            |
+---------------+---------------+---------------+---------------+
/                                                               /
/                     KeyId (length octets)                     /
/                                                               /
/                                                               /
+---------------+---------------+---------------+---------------+
```

TOC

7.3.4. Content Object Hash

The SHA-256 hash of the Content Object. This is the self-certifying name restriction that must be verified in the network, if present.

The only acceptable length is 32.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_OBJHASH           |             length            |
+---------------+---------------+---------------+---------------+
/                                                               /
/                     SHA-256 digest (32 bytes)                 /
/                                                               /
/                                                               /
+---------------+---------------+---------------+---------------+
```

TOC

7.3.5. Interest Lifetime

The Interest Lifetime expresses how long an Interest should stay pending at an intermediate node. It is express in milli-seconds as a non-negative big-endian integer.

A value of 0 (encoded as 1 byte %x00) indicates the Interest does not elicit a Content Object response.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_INT_LIFE           |             length            |
+---------------+---------------+---------------+---------------+
/                                                               /
/                    lifetime (length octets)                   /
/                                                               /
+---------------+---------------+---------------+---------------+
```

TOC

7.4. Content Objects

A Content Object contains four mandatory sections, some of which contain optional TLVs.

The Content Object Digest of a Content Object is defined as a SHA-256 beginning at the start of the message (T_OBJECT) and ending at the end.

The signing hash is defined over the oject from the start of the message (T_OBJECT) up to but not including the signature.

Content Object

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|                          Fixed Header                         |
|                                                               |
+---------------+---------------+---------------+---------------|
/ Per-hop TLVs                                                  /
+---------------+---------------+---------------+---------------+
|            T_OBJECT           |         message length        |
+---------------+---------------+---------------+---------------+
|            T_NAME             |          name length          |
+---------------+---------------+---------------+---------------+
/                                                               /
/                          name value                           /
/                                                               /
+---------------------------------------------------------------+
/ Mandatory Name Authenticator TLV                              /
+---------------------------------------------------------------+
/ Optional Protocol Information TLV                             /
+---------------------------------------------------------------+
/ Optional Contents TLV                                         /
+---------------------------------------------------------------+
/ Mandatory Signature Block TLV                                 /
+---------------------------------------------------------------+
```

Type	Abbrev	Name	Description
%x0001	T_KEYID	Signer Key ID (Signer KeyId)	An octet string that identifies the key used to sign the contents.
%x0002	T_NAMEAUTH	Name Authenticator (Name Authenticator)	Crytpographic information to bind the name, contents, and signature.
%x0003	T_PROTOINFO	Protocol Information (Name Authenticator)	A container for protocols to place their own containers.

%x0004	T_CONTENTS	Content (Content)	Binary data, the contents of a Content Object. Certain types of content objects have specified Contents, such as a Link or Collection.
%x0005	T_SIGBLOCK	Signature Block (Signature)	Contains the Signature Bits, optional Witness, and optional Digest Algorithm.
%0006	C_SUITE	Crypto Suite (Crypto Suite)	A short identifier of the digest and signing algorithms.
%x0007	T_PUBKEYLOC	Public Key Locator (Key Name)	A container for the public key locator (a key, a cert, or a keyname)
%x0008	T_KEY	Key (Key)	DER encoded public key.
%x0009	T_CERT	Certificate (Certificate)	DER encoded X509 certificate.
%x000A	T_KEYNAME_KEYID	The Singer Key ID of the KeyName signer (Key Name)	The signer key id of the signer of the Key Name.
%x000B	T_OBJINFO	Object Information (Object Information)	A Protocol Information container with information about the contents.
%x000C	T_OBJTYPE	Content Object Type (Content Type)	Indicates the type of Contents (data, key, link, etc.)
%x000D	T_CREATE	Creation Time (Createtion Time)	A milli-second since epoch in UTC timestamp when the object was created. Should be omitted or set to "0" if the creation time is not known at signing time.
%x000E	T_SIGBITS	Signature Bits (Signature Bits)	A signature, contents varies depending on algorithm.

Table 6: CCNx Content Object Types

TOC

7.4.1. Name Authenticator

The name authenticator contains the cryptographic information necessary to bind the Name to the Contents via the Signature.

Name Authenticator must include a Signer Key ID, the Crypto Suite, and an optional Key Locator. The supported type of Locator is a Public Key Locator.

The Signer Key ID is similar to a Subject Key Identifier from X509 [RFC 3820, Section 4.2.1.2]. It should be derived from the key used to sign, such as from the SHA-256 hash of the key. It applies to public/private key systems and to symmetric key systems.

The Key Locator is an optional field. If it is not present, a node wishing to authenticate a content object must have prior knowledge of the Publisher Key ID, or be able to retrieve the corresponding key through external means. The Key Locator may contain one of : A DER encoded key, a DER encoded X509 certificate, or a KeyName.

A KeyName is a mandatory Name and an optional KeyId. The KeyId inside the Key Locator may be included in an Interet's KeyId to retrieve only the specified key.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_NAMEAUTH          |            length             |
+---------------+---------------+---------------+---------------+
/ Mandatory Signer KeyId TLV                                    /
+---------------+---------------+---------------+---------------+
/ Mandatory CryptoSuite TLV                                     /
+---------------+---------------+---------------+---------------+
/ Optional Public Key Locator                                   /
+---------------------------------------------------------------+
```

TOC

7.4.1.1. Signer KeyId

The publisher key identifier.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|             T_KEYID           |            length             |
+---------------+---------------+---------------+---------------+
/                            Key ID                             /
/---------------+---------------+-------------------------------+
```

TOC

7.4.1.2. Crypto Suite

The Crypto Suite is a short notation indicating the digest algorithm and encryption used to sign a Content Object. The Cipher Suite is a variable-length number evaluated in network-byte order.

If an implementation cannot parse or does not implemented a specified algorithm, it MUST silently discard the content object.

An implementation may choose which cipher suites, if any. The initial list of Crypto Suites is:

Network Byte Order Code	Suite
%x00	SHA-256 with RSA.

%x01	SHA-256 with Symmetric Key.

Table 7: Crypto Suites

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_SUITE            |             length            |
+---------------+---------------+---------------+---------------+
|         Crypto Suite          /
+---------------+---------------+
```

TOC

7.4.1.3. Public Key Locator

The Public Key Locator may be one of a Key, a Certificate, or a KeyName.

TOC

7.4.1.3.1. Key

A Key is a DER encoded Subject Publick Key Info block, as in an X509 certificate.

```
                     1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+---------------+---------------+---------------+---------------+
|             T_KEY             |             length            |
+---------------+---------------+---------------+---------------+
/                      Key (DER encoded SPKI)                   /
+---------------+---------------+---------------+---------------+
```

TOC

7.4.1.3.2. Certificate

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_CERT             |             length            |
+---------------+---------------+---------------+---------------+
/                 Certificate (DER encoded X509)                /
+---------------+---------------+---------------+---------------+
```

TOC

7.4.1.3.3. Key Name

A key name type key locator is a mandatory Name TLV followed by an optional Digest key digest.

The KeyName digest is the publisher digest of the content object identified by KeyName. It may be included an an Interest's digest restriction.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|          T_KEYNAME            |            length             |
+---------------+---------------+-------------------------------+
/ Mandatory Name TLV                                            /
+---------------------------------------------------------------+
/ Optional KeyName Digest                                       /
+---------------------------------------------------------------+
```

TOC

7.4.2. Object Information

The Object Information contains the state of name-based protocols and metadata about the content object.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_OBJINFO           |            length             |
+---------------+---------------+---------------+---------------+
/ Optional Content Metadata TLV                                 /
+---------------------------------------------------------------+
/ Optional Protocol-specific TLVs                               /
+---------------------------------------------------------------+
```

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|        T_CONTENT_META         |            length             |
+---------------+---------------+---------------+---------------+
/ Mandatory Content Type TLV                                    /
+---------------------------------------------------------------+
/ Optional Creation Time TLV                                    /
+---------------------------------------------------------------+
```

TOC

7.4.2.1. Content Type

The Content Type is a network byte order integer encoded in the shortest length, representing the general type of the Contents TLV.

- 0: Data

- 1: Encrypted Data
- 2: Gone (whiteout)
- 3: Key
- 4: Link
- 5: NACK

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_OBJTYPE          |            length             |
+---------------+---------------+---------------+---------------+
|  content type /
+---------------+
```

7.4.2.2. Createtion Time

A milli-second timestamp is a big-endian ordered integer of the number of milli-seconds since the epoch in UTC of when the contents were created, like a POSIX file creation time. It is in a fixed 64-bit network byte order field.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_CREATE            |            length             |
+---------------+---------------+---------------+---------------+
/                            milli-seconds                      /
/                                                               /
+---------------+---------------+---------------+---------------+
```

7.4.2.3. Protocol Specific TLVs

Protocol TLVs are a place for name-based protocols to put state about the protocol inside the signed content object. Each protocol must use an assigned or experimental protocol TLV. The payload of the TLV is protocol-specific.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|         <protocol tlv>        |            length             |
+---------------+---------------+---------------+---------------+
/                            payload                            /
+---------------+---------------+---------------+---------------+
```

7.4.3. Signature

The Signature Block includes the cryptographic signature of the content object.

A Signature Block MUST contain a Signature Bits TLV.

A Content Object without a mandatory field that is understood by an implementation, and that verifies, should be silently discarded.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_SIGBLOCK          |            length             |
+---------------+---------------+---------------+---------------+
/ Mandatory Signature Bits TLV                                  /
+---------------------------------------------------------------+
```

TOC

7.4.3.1. Signature Bits

The cryptographic signature using the signature's digest algorithm

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_SIGBITS           |            length             |
+---------------+---------------+---------------+---------------+
/                            Signature                          /
+---------------+---------------+---------------+---------------+
```

TOC

7.4.4. Content

Opaque content bytes.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_CONTENTS          |            length             |
+---------------+---------------+---------------+---------------+
/                             Content                           /
+---------------+---------------+---------------+---------------+
```

TOC

8. Acknowledgements

TOC

9. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

10. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

11. References

TOC

11.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

11.2. Informative References

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

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

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in

RFCs," BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force — M. Mosko, Ed.
Internet-Draft — PARC
Intended status: Experimental — April 13, 2014
Expires: October 15, 2014

Labeled Segment URIs

Abstract

This document defines an RFC3986 URI compliant identifier in which path segments carry a label. This allows differentiation between resources with otherwise similar identifiers that are not related. For example, one resource could be named "/parc/csl/7" meaning the 7th version of "/parc/csl", while another could mean the 7th page of the resource. With labeled segments, the two resources would have unambiguous names, such as "/parc/csl/version=7" and "/parc/csl/page=7". A URI scheme that specifies the use of labeld segment URIs conforms the the encoding rules presented here.

Status of this Memo

Copyright Notice

Table of Contents

TOC

1. Introduction

A Labeled Segment is an URI (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) [RFC3986] compliant convention for an application or protocol to embed labels in path segments to disambiguate the resource indentified by the path. Labeled Segment URIs also allow for query and fragment components to follow the Labeled Segment form.

Some protocols may wish to disambiguate name components between different identifier spaces, such as "version" and "page". Some protocols may wish to use a type system, such as "/str=parc/int=7" and "/str=parc/str=7". Labeld Segment URIs provide an unambiguous and flexible represenation in systems that allow resources with otherwise similar names.

It is not sufficient to leave the determination of type to application-specific conventions. In a networked system with multiple applications accessing resources generated by other applications, there needs to be a common understanding of conventions. For example, if one application uses a base 64 encoding of a frame number, say base64(0xbdea) and another uses "ver=" to represent a document version, then there is an ambiguity because base64(0xbdea) is the string "ver=".

Labeled Segments defines an "ls-segment" as "label[:param]=value", where the value only contains unreserved, percent-encoded, or certain sub-delim characters. In the previous example, one protocol would say "/frame=%BD%EA" and the other would say "/ver=".

TOC

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].

TOC

2. Labeled Segments

This section describes the formal grammar for Labeled Segments, using ABNF (Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 2008.) [RFC5234] notation. We do not impose restrictions on the length of labels or values. The semantics of values are URI scheme specific, we only describe the meta-structure of labeled segments. We begin by reviewing some definitions from [RFC3986] (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) that define an aboslute path URI.

```
URI           = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
hier-part     = "//" authority path-abempty
                / path-absolute
                / <other path types>
path-absolute = "/" [ segment-nz *( "/" segment ) ]
segment       = *pchar
segment-nz    = 1*pchar
pchar         = unreserved / pct-encoded / sub-delims / ":" / "@"
query         = *( pchar / "/" / "?" )
fragment      = *( pchar / "/" / "?" )
pct-encoded   = "%" HEXDIG HEXDIG
unreserved    = ALPHA / DIGIT / "-" / "." / "_" / "~"
reserved      = gen-delims / sub-delims
gen-delims    = ":" / "/" / "?" / "#" / "[" / "]" / "@"
sub-delims    = "!" / "$" / "&" / "'" / "(" / ")"
                    / "*" / "+" / "," / ";" / "="
```

Labeled Segments defines a new segment type that provides unambiguous representation of a segment's label and its value. We define the top-level LS-URI as the same form as a URI, wherein each part conforms to the Label Segment grammar, which is a subset of the URI grammar.

```
LS-URI           = scheme ":" ls-hier-part ["?" ls-query]
                   ["#" fragment]
ls-hier-part     = ["//" authority] ls-path-absolute
ls-path-absolute = "/" [ ls-segment *( "/" ls-segment ) ]
ls-segment       = lpv-segment / v-segment
lpv-segment      = label [":" param] "=" s-value
v-segment        = s-value
label            = alpha-t / num-t
param            = alpha-t / num-t
s-value          = *(s-pchar)

ls-query         = *1 ( lpv-component / v-component
                        *( "&" (lpv-component / v-component) ) )
lpv-component    = label [":" param] "=" q-value
v-component      = q-value
q-value          = *(q-pchar)

alpha-t          = ALPHA *(ALPHA / DIGIT)
num-t            = dec-t / hex-t
dec-t            = 1*(DIGIT)
hex-t            = "0x" 1*(HEXDIG)
ls-pchar         =  unreserved / pct-encoded / ls-sub-delims
s-pchar          = ls-pchar / ":" / "@" / "&"
q-pchar          = ls-pchar / ":" / "@" / "/"
ls-sub-delims    = "!" / "$" / "'" / "(" / ")"
                       / "*" / "+" / "," / ";"
```

A Labeled Segment URI (LS-URI) contains a scheme that uses Labeled Segments, an optional authority, a labeled segment absolute path (ls-path-aboslute), an optional labeled segment query (ls-query), and a fragment. The authority is URI scheme specific and the fragment is independent of the URI scheme.

The labeled segment path is composed of zero or more labeled segments (ls-segment). Each ls-segment may be either a label-param-value tuple (lpv-segment) or a value singleton (v-segment). A v-segment is an un-labeled segment. A particular LS-URI scheme MUST define how unlabeled segments are processed, and MAY disallow them. An lpv-segment specifies a label, optional parameter for the label, and the segment value.

lpv-segment values come from the s-pchar set, which excludes the "=" equal sign. This means that the only equal sign in a path segment must be the delimeter between the label:param and the value. Within the value, an equal sign must be percent encoded.

lpv-segment lables and values may be alpha-numeric identifiers or numbers (decimal or hexidecimal). For example, one scheme may define the labels "name", "version", and "frame". A version may be of types "date" or "serial", meaning a date is used as the version, or a monotonic serial number. Some examples of resulting LS-URIs are: "/name=parc/name=csl/version:date=20130930" or "/name=alice_smith/version:serial=299". The parameters may also indicate an instance of a label, such as "/name=books/year:1=1920/year:3=1940", where there are scheme or application semantics to "year:1" and "year:3".

lpv-segment labels and parameters may also be numbers. For example, a protocol with a binary and URI representation may not have pre-defined all possible labels. In such cases, it could render unknown labels as their binary value, such as "/name=marc/x2003=green".

The ls-query component is a non-hierarchical set of components separated by "&". Each ls-query component is either a lpv-component or a v-component, similar to segments. They are based on q-value, which uses q-pchar that excludes "&", but includes "/". This allows an LS-URI scheme to use type query parameters.

Labeled Segments allow for dot-segments "." and ".." in a v-segment. They operate as normal. A single dot "." means the current hierarchy level, and may be elided when the URI is resolved. Double dot ".." segments pop off the previous non-dot-segment. An lpv-segment with a value of "." or ".." is not a dot-segment. It means that the value of the given label is "." or "..". For example /a=parc/b=csl/.. is equivalent to "/a=parc/b=csl", but the LS-URI "/a=parc/b=csl/c=.." does not contain a dot-segment.

TOC

3. URI comparison

An LS-URI scheme MUST specify the normalization rules to use, following the methods of Section 6 (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) [RFC3986]. At minimum, an LS-URI scheme SHOULD:

- Normalize unrestricted percent-encodings to the unrestricted form.
- Normalize num-t to either dec-t or hex-t.
- If the scheme allows for value-only segments or query componets and interprets them as a default type, they should be normalized to having the type specified.

- If the scheme allows for undefined labels and represents them, for example, as num-t, then it should normalize all labels to their corresponding num-t. If "name", for example, is known to be %x50 in a binary encoding of the URI, then all labels should be compared using their numeric value.

TOC

4. Acknowledgements

TOC

5. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

6. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

7. References

TOC

7.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

7.2. Informative References

[RFC3552]	Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” BCP 72, RFC 3552, July 2003 (TXT).
[RFC3986]	Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” STD 66, RFC 3986, January 2005 (TXT, HTML, XML).
[RFC5226]	Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).
[RFC5234]	Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” STD 68, RFC 5234, January 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014
Expires: October 15, 2014	

Labeled Content Information

Abstract

This document Labeled Content Information, which is a labeled segment URI (LS-URI) representation of network data. This scheme, called "lci:" is applicable to network protocols such as Content Centric networks (CCNx) and Named Data Networks (NDN). Labeled Content Information applies specific labels to each path segment of a URI to disambiguate between resources with similar names. There is a specific set of segment labels with label semantics.

Status of this Memo

Copyright Notice

Table of Contents

TOC

1. Introduction

In this document, we use URI (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) [RFC3986] terminology, so a URI and CCNx Name are made up of a URI path, and the path is made up of path segments. We do not use the term "name component" as is common in CCNx. We use the term "content object segment" to mean the CCNx concept of segment, as opposed to a URI "path segment." The word "segment" alone means "path segment."

Labeled Content Information carry a label for each name segment. The contents of each segment must conform to the label semantics. Common segment types are "UTF-8 Segment", "Binary Segment", and "KeyId".

We use Labeled Segment URIs as the canonical human-readable representation. There is an unambiguous, one-to-one correspondence between an absolute LS-URI path and a Labeled Name. Relative URI representations are removed during encoding, so no relative name ends up in wire format. Some labels are URIs are IRI (Duerst, M. and M. Suignard, “Internationalized Resource Identifiers (IRIs),” January 2005.) [RFC3987] compatible.

Labeled Names shall be used everywhere a Name is used in CCNx, such as in the name of an Interest or Content Object. They are also used in Links, Key Locators, or any other place. When encoded for the wire, a binary representation is used, depending on the specific wire format codec, which is outside the scope of this document.

This document specifies:

- Name labels.
- A canonical URI representation.

Formal grammars use the ABNF (Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 2008.) [RFC5234] notation.

TOC

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].

TOC

2. Labeled Names

This section describes the Labeled Content Information "lci:" schema for labeled names. A Labeled Content name assigns a semantic type to each segment of the hierarchical content name.

Unless otherwise specified, a name segment is an arbitrary sequence of octets.

Several segment labels are binary unsigned integers. These are always encoded as variable length sequences of 1 or more octets in network byte order using the shortest representation (i.e. no leading %x00). The value of "0" is encoded as the single byte of "%x00". A zero-length sequence must be interpreted as "not present." There is no limit to the number of bytes in the octet sequence.

The types are:

- UTF-8 Segment: A general name component that conforms to UTF-8. It is a sequence of 0 or more octets.
- Binary Segment: A general name component as an octet string. It is a sequence of 0 or more octets.
- Nonce: A binary component representing a unique value to differentiate names, sometimes used in an Interest.
- Key: A binary component representing the SHA-256 digest of a key. These are sometimes used as the hash of the public key of the publisher.
- ObjectHash: A binary component representing the SHA-256 hash of a content object.
- Meta: a binary type indicates the suffix identifies metadata about a parent object.
- Application Type N: An application may use application-specific parameters, numbered as integers, where N is from 0 to a system-specific maximum, not less than 255. These are represented as "App:1=value", for example.

It is common for an information centric networking protocol, such as CCNx or NDN, to use a binary on-the-wire representation for messages. Such protocol, if they use the lci: scheme, must have an appropriate codec that unambiguously represents Labeled Content Information in the chosen wire format. Relative dot-segments should not occur in the wire format, they should be resolved before encoding.

TOC

3. URI Representation

Typed Names use a standard RFC 3986 representation following the LS-URI convention. A path segment consists of any "unreserved" characters plus percent-encoded characters. Reserved characters must be percent encoded.

Within an absolute path, each segment consists of a "ls-segment" (c.f. LS-URI). A labeled segment is a type and a name component value, with a URI representation of "type=value". The "type=" portion may be omitted if it is type "N" (Name).

Some name types take a parameter, such as the Application types. They are represented as "A:nnn=value", where the "nnn" is the application type number and value is the name component.

The Authority, Query, and Fragment sections of a URI are not used. If provided, they are ignored.

Dot-segments (relative name components) are resolved when the URI is converted to a Typed Name. The "." dot-segment is removed. The ".." dot-segment is removed along with the previous non-dot-segment.

Type	Display	Name
'Name'	UTF-8	UTF-8 Segment
'Binary'	Hexadecimal	Binary Segment
'Nonce'	Hexadecimal	Nonce
'Key'	Hexadecimal	Key
'Meta'	Hexadecimal	Metadata segment
'App:0' - 'App:255'	Hexadecimal	Application Component

Table 1: Labeled Content Information Types

TOC

3.1. Examples

```
A name /foo/bar.
   lci:/Name=foo/Name=bar
   lci:/foo/Name=bar
   lci:/foo/bar

A name /foo/bar with key %xA0.
   lci:/Name=foo/Name=bar/Key=0xA0

A name /foo/bar with version %xA0 and Nonce 0x09.
   lci:/foo/bar/Key=0xA0/Nonce=0x00/../Nonce=0x09

A name /foo/.., where the ".." is a literal name component,
not a relative dot-segment.
```

```
   lci:/foo/Name=..

A name /foo/bar with applications type 0 "hello"
and application type 1 "world".
   lci:/Name=foo/Name=bar/App:0=hello/App:1=world
```

TOC

4. lci: URI comparison

While most comparisons are done using a wire format representation of a lci: URI, some applications may compare Labeled Content Information using their URI representation. This section defines the rules for comparing lci: URIs using the methods of Section 6 (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) [RFC3986]

Comparing typed name URIs must be done with:

- Syntax-based normalization
- Case normalization: normalize the representation of percent encodings. lci: does not use the host portion of the URI, and should be ignored if present.
- Percent encoding normalization: Percent encodings of unreserved characters must be converted to the unreserved character.
- Path segment normalization: dot-segments must be resolved first.
- Scheme-based normalization: The authority should be removed and the path represented as an absolute path.
- Protocol-based normalization: Should not be done. A trailing slash indicates a zero-length terminal name component and signifies a different name.
- typed-name-segment normalization: All segments should be presented with their type, do not elide the "N=" for Name components.
- Binary unsigned integer normalization: remove any leading %x00 from numbers, leaving only the terminal %x00 for "0".
- type parameters: they must have their percent encodings normalized. If they are integers, such as for the 'A' type, they must not have leading zeros.

TOC

5. IRI Considerations

International Resource Identifiers extend the unreserved character set to include characters above U+07F and encode them using percent encoding. This extension is compatible with the lci: schema. It applies only to the "value" portion of an ls-segment.

The canonical name is determined by the URI representation of the IRI, after applying the rules of Section 3.1 of [RFC3987] (Duerst, M. and M. Suignard, “Internationalized Resource Identifiers (IRIs),” January 2005.) and resolving dot-segments. The canonical name thus includes the URI representation of language markers, including the bidirectional components.

The value of a UTF-8 Name segment should be interpreted using IRI rules, including bidirectional markers. They may be displayed using localized formats.

Binary unsigned integer types are not interpreted under IRI rules, they are specifically percent encoded numbers. They may be displayed using a localized format.

TOC

6. Acknowledgements

TOC

7. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

8. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

9. References

TOC

9.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

9.2. Informative References

[CCNx]	PARC, Inc., “CCNx Open Source,” 2007.
[RFC3552]	Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” BCP 72, RFC 3552, July 2003 (TXT).
[RFC3986]	Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” STD 66, RFC 3986, January 2005 (TXT, HTML, XML).
[RFC3987]	Duerst, M. and M. Suignard, “Internationalized Resource Identifiers (IRIs),” RFC 3987, January 2005 (TXT).
[RFC5226]	Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).
[RFC5234]	Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” STD 68, RFC 5234, January 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014
Expires: October 15, 2014	

CCNx Content Object Caching

Abstract

This document specifies caching policies for CCNx Content Objects. It follows the similar basic principles as HTTP/1.1 [RFC2616 Section 13.2].

Status of this Memo

Table of Contents

TOC

1. Introduction

CCNx Content Objects may have short-term or long-term value, and may be cached accordingly at intermediate network nodes. Because each CCNx Content Object is signed, nodes that validate signatures may be assured of each object's provenance. This specification describes how the headers and fields within a Content Object may steer caching behavior. It specifies both a hard expiry time. These cache directives do not apply to packets in-flight, but to nodes originating content objects from a cache.

CCNx uses two types of messages: Interests and Content Objects. An Interest carries a hierarchically structured variable-length identifier (HSVLI), also called the "name", of a Content Object and serves as a request for that object. If a network element sees multiple interests for the same name, it may aggregate those interests. A network element along the path of the Interest with a matching Content Object may return that object, satisfying the Interest. The Content Object follows the reverse path of the Interest to the origin(s) of the Interest. A Content Object contains the HSVLI, the object's payload, and cryptographic information used to bind the HSVLI to the payload.

This specification adds new Per-Hop TLVs, Interest Selector TLVs, and Content Object TLVs related to caching. It updates [ietf-draft-ccnxmessages-02].

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

TOC

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].

TOC

2. Expiry Caching in CCNx

These rules only apply to content objects that carry a MaxAge TLV. If the content producer has not specified a MaxAge, caches may hold a Content Object for as long as their local policies allow.

Because CCNx Content Objects are immutable, there is no re-validation of objects with a "server", a term ill-defined in CCNx. If a content producer -- the entity that creates and signes a content object -- puts a short validity period on an object, it does not mean it will be revalidated, it means it will be purged from network caches. The producer cannot re-publish the object with a new validity period, that would result in a cryptographically different content object. Therefore, a content producer should put the maximum validity period in a content object and not use staleness except to invalidate objects past their shelf life.

CCNx object caching has less flexabilty around staleness than HTTP/1.1. A client may not request stale object. It may specify a minimum freshness, that is that the resulting content object have at lest min-fresh seconds left before it becomes stale.

A content producer MAY specify in a per-hop header that a content object SHOULD NOT be cached. An intermediate system should only store the content object long enough to satisfy pending interests for it. This directive takes precedence over other cache directives.

A content producer MAY specify a MaxAge in the Protocol Information, indicating the validity period of the content object. It is allowed to have a MaxAge -- which is a constraint on the data -- and the no-cache directive in the per-hop headers -- which is a constrain on caches.

TOC

2.1. Unsynchronized Time

If a content publisher does not have reasonable time synchronization to UTC, it MUST follow these rules for Unsynchronized Time.

It MUST NOT put a Creation Time or Modification Time TLV in the Content Object.

If it wishes to control the lifetime of the Content Object, it MUST place a MaxAge TLV in the Signed Information and an Age TLV in the per-hop headers. The MaxAge TLV is measured in relative seconds. The Age field is measured in relative seconds. It may set the MaxAge TLV to the desired value and MUST set the Age TLV to 0.

Each hop that processes a Content Object with an Age field must increment the Age field by the approximate time it spent on the system. If the system is simply forwarding the Content Object, it may increment it by +0 (unless it knows it was longer). If the intermediate system is caching the object, it must increment it by approximately the number of seconds the object resided on the system.

A Content Object is stale if the Age > MaxAge. After a content object is stale, a cache SHOULD purge it from its memory and MUST NOT respond to an Interest with the object.

A forwarder MAY NOT check freshness. It is only required to increment the Age if the forwarding took a significant amount of time.

An end system that issued an Interest should accept a Content Object with 0 or more seconds of age remaining, but should not forward them if the object is stale.

TOC

2.2. Synchronized Clocks

If a content object publisher (or the signer) has a reasonblely syncrhonized clock to UTC, it MAY populate the Creation Time and Modification Time fields of a content object's Signed Information. A node without a reasonably synchronized clock, MUST NOT populate those fields.

Unlike HTTP/1.1, there is no Expires value. The content producer controls cache lifetime solely with the MaxAge parameter.

The "origin_time" is the maximum of Creation Time and Modification Time. If both fields are missing, the producer did not have a synchronized clock and the methods of Unsynchornized Clocks MUST be followed.

If both the content producer and cache have reasonably sychronized clocks, the "date_value" is calcuated as "now - max(creation_time, modification_time)" from the signed information. A missing field is considered "0".

```
date_value = max(creation_time, modification_time)
```

Because we have both "age_value" (from the Age header) and our locally computed age based on synchronized clocks, we use the "corrected_value", which uses the maximum of the age as computed by previous caches and our local age.

```
corrected_received_age = max(now - date_value, age_value)
```

A content object is "fresh" if

```
response_is_fresh = (corrected_receive_age <= max_age )
```

TOC

3. TLV Types

This section specifies the TLV types used by CCNx Content Object caching.

TOC

3.1. Per-hop TLVs

Type	Abbrev	Name	Description

%06	T_CACHE	Object Caching (Object Caching)	Per-hop directives on object caching.
%07	T_AGE	Object Age (Age)	The approximate age, in milli seconds, of the content object.

Table 1: CCNx Per-hop header types

TOC

3.1.1. Object Caching

Specifies per-hop behavior for handling a Content Object. It is the binary OR of these flags:

0x01 = Do not cache

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+
|    T_CACHE    |     length    |    handling   |
+---------------+---------------+---------------+
```

TOC

3.1.2. Age

Age is the number of seconds the object has been cached in the network. It is an unsigned 4-byte integer in network byte order.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    T_CHILD    |     length    |              age
+---------------+---------------+---------------+---------------+
                                |
+---------------+---------------+
```

TOC

3.2. Interest Selectors

CCNx Content Object caching adds the MinimumFreshness TLV to Interest Selectors. This specifies the minimum remaining freshness of a content object to be returned satisfying the Interest.

Type	Abbrev	Name	Description
%05	T_MINFRESH	Minimum Freshness (Minimum Freshness)	The minimum seconds of remaining lifetime.

Table 2: Interest Selectors

TOC

3.2.1. Minimum Freshness

Specifies the minimum freshness of a content object that satisfies the interest. The default value is "0", meaning anything up to a just-expired objet.

This parameter MAY NOT be evaluated by a fast-path forwarder. A long-term cache MUST evaluate this paramter when selecting a content object to return.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+--------------+
|   T_MINFRESH  |     length    |    variable legnth integer   /
+---------------+---------------+---------------+--------------+
```

TOC

3.3. Content Object Signed Information

CCNx Content Object caching adds the MaxAge TLV to the Signed Information of a content object.

Type	Abbrev	Name	Description
%05	T_MAXAGE	Maximum Age (Minimum Freshness)	The maximum seconds of remaining lifetime.

Table 3: Content Object Signed Information

TOC

3.3.1. Minimum Freshness

Specifies the minimum freshness of a content object that satisfies the interest. If not present, the maximum caching time is unbounded, governed only by local cache policies.

It is encoded as a big-endian unsigned integer using the minimum number of bytes. The value of "0" (meaning no caching) is encoded as the single byte %x00.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+--------------+
|    T_MAXAGE   |     length    |    variable legnth integer   /
+---------------+---------------+---------------+--------------+
```

TOC

4. Acknowledgements

TOC

5. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

6. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

7. References

TOC

7.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

7.2. Informative References

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” RFC 2616, June 1999 (TXT, PS, PDF, HTML, XML).

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

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014
Expires: October 15, 2014	

CCNx End-To-End Fragmentation

Abstract

This document specifies and end-to-end fragmentation protocol for breaking a Content Object in to several smaller pieces to fit within a network MTU. The fragmentation protocol does not provide a secure binding of the fragments to the original object, which is left to the receiving endpoint. Midpoints may only serve fragments from cache if they have assembled and verified the complete content object.

Status of this Memo

Table of Contents

1. Introduction

This document specifies and end-to-end fragmentation protocol for breaking a Content Object in to several smaller pieces to fit within a network MTU. The fragmentation protocol does not provide a secure binding of the fragments to the original object, which is left to the receiving endpoint. Midpoints may only serve fragments from cache if they have assembled and verified the complete content object.

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

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].

2. Protocol Description

The principle of operation for fragmentation is that intermediate systems should not have to fragment packets. This is achieved by an Interest always fragmented to the minimum MTU and recording the forward path's MTU in the Interest, so a system sending back a Content Object may fragment it to the path's size, or smaller. An intermediate system's content store may store only pre-fragmented objects and respond only if those fragments satisfy an Interest's MTU, otherwise it may be considered a cache miss.

Because the Interest is the path discovery mechanism for content objects, all interests MUST carry an Interest Fragmentation Header so the reverse path MTU is known at the node responding with a Content Object.

The minimum MTU required is 1280 octets. Any system implementing a physical layer with a smaller MTU must implement link fragmentation, for example using a PPP layer over the small MTU network.

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

When an end system creates a fragment stream, it generates a 64-bit number for the Fragment Stream ID. This number identifies a contiguous stream of fragments. An end system uses the Fragment Stream ID for reassembly. An intermediate system uses the Fragment Stream ID of a Content Object to ensure that only one stream of Content Object fragments follow a reverse PIT entry.

A system SHOULD use a random number for an interest's Fragment Stream ID. This avoid easy denial of service attacks by replying with junk for known fragment stream IDs. A fragmented content object carries both its own Fragment Stream ID, which SHOULD be based on the content object hash, and the corresponding Interest Fragment Stream ID to facilitate matching on the reverse PIT path.

If the Maximum Path MTU of a Content Object fragment is larger than the supported MTU on an egress interface, the fragment stream should be dropped on that interface, even if some fragments fit within the MTU.

Fragments are identified by a serial counter FragNum, which ranges from 0 - 63. Forwarders and end systems should drop duplicate fragments, identified by the tuple {Fragment ID, FragNum}.

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

TOC

2.1. Interest Fragmentation

If an Interest does not fit with 1280 bytes, then it must be fragmented to fit within 1280 bytes. There is no path MTU discovery for Interests.

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

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

A forwarding node must examine the Interest name to determine its forwarding. This requires that the forwarding node re-assemble the front of the Interest to examine the name. In a typical case, this means that the node must receive fragment 0 to have enough prefix name components to compute the route. A system MAY discard out-of-order fragments after fragment 0 during this re-assembly, and once fragment 0 arrives and the system constructs a PIT entry with the routing, it should send a control message along the Fragment Stream ID's reverse path to cause the source to resend the interest stream, which can now be forwarded out of order. Or, it may buffer out-of-order fragments.

A system that receives an Interest encapsulated in a packet larger than 1280 octets must discard it.

TOC

2.2. Content Object Fragmentation

When forwarding a Content Object along the reverse path of the PIT, a fragment stream may only be forwarded along reverse PIT entries for which it satisfies the reverse path minimum MTU.

A PIT entry should only be removed once all fragments of a fragment stream pass through, or it times out. Because the FragCnt is limited to 63, a system may match a first stream's Fragment ID and use a single 64-bit mask.

A Content Object is fragmented based on the Interest minimum path MTU. It carries an "Maximum Fragment MTU" field set to the maximum fragment size, which must be no more than an Interest's minimum path MTU. Because a fragment stream may only satisfy PIT entries with larger or equal minimum path MTU, all fragments must carry the Object's fragmentation size. An intermediate node may, for example, receive the last fragment first, so even if fragments were packed to maximum size, the forwarder could not infer which PIT entries the object satisfies without know the fragment stream's fragmentation size

---

TOC

3. Packet Formats

End-to-end fragmentation uses a network-level TLV header for fragmentation. There is one header for Interests and one header for Content Objects.

Both fragmented Interests and Content Objects use the fields "FragCnt" and "FragNum". The count is the number of fragments in the message, which has a minimum of 1. FragNum is the 0-based fragment number. Sequential fragment numbers represent sequential byte boundaries of the original object.

TOC

3.1. Interest Header

The field "Fragment Stream ID" identifies a contiguous stream of fragments. It SHOULD be a random number.

The field "Path Minimum MTU" is updated per-hop to measure the minimum path MTU of the interest's reverse path.

```
                    1                   2                   3
```

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_INTFRAG            |             LENGTH            |
+---------------+---------------+---------------+---------------+
|                        Fragment Stream ID                     |
|                                                               |
+---------------+---------------+---------------+---------------+
|       Path Minimum MTU        | X X | FragCnt | X X | FragNum |
+---------------+---------------+---------------+---------------+
```

TOC

3.2. Content Object Header

The field "Fragment Stream ID" identifies a contiguous stream of fragments for the Content Object. It SHOULD be derived from the content object's hash, so two objects with the same Fragment Stream ID represent the same fragmented object.

The field "Interest Stream ID" is the Fragment ID of the corresponding Interest that the object is answering. This allows PIT matching without having to reconstruct the content object. It makes content objects specific to a given Interest similarity hash.

The field "Object Maximum MTU" is the maximum size of any fragment in the fragment stream. This allows a forwarder to match a content object fragment stream against a reverse path MTU size and not send a fragment stream that will not fit down a path.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_OBJFRAG            |             LENGTH            |
+---------------+---------------+---------------+---------------+
|                        Fragment Stream ID                     |
|                                                               |
+---------------+---------------+---------------+---------------+
|       Object Maximum MTU      | X X | FragCnt | X X | FragNum |
+---------------+---------------+---------------+---------------+
|                        Interest Stream ID                     |
|                                                               |
+---------------+---------------+---------------+---------------+
```

TOC

4. Cache Considerations

Objects should be reassembled before sending from cache, to ensure all fragments exist at the cache.

Single fragment Interests may be satisfied from cache. A system may choose to reassemble Interests to try and answer from cache. If a cache miss, the original fragment stream should be forwarded.

TOC

5. Acknowledgements

TOC

6. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

7. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

8. References

TOC

8.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

8.2. Informative References

[CCNx]	PARC, Inc., “CCNx Open Source,” 2007.
[RFC3552]	Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” BCP 72, RFC 3552, July 2003 (TXT).
[RFC5226]	Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014
Expires: October 15, 2014	

CCNx Content Object Segmentation

Abstract

This document specifies a segmentation protocol for user payload into CCNx Content Objects, and the naming convention to use for such segmented payload. It adds a field to a Content Object to represent the last segment of an object. It also specifies a metadata convention to store information about the segmented object.

Status of this Memo

Table of Contents

TOC

1. Introduction

CCNx Content Objects are fashioned to amortize cryptographic signatures over user data while staying within reasonable size for transport over today's networks. This means a Content Object is usually within common UDP or jumbo Ethernet size. If a user has a larger amount of data to write to a single name, the data is segmented with this segmentation protocol. This protocol uses state in both the Name and in a Content Object field. A segmented object may also have a metadata object that describes the original object before it was split up.

CCNx uses two types of messages: Interests and Content Objects. An Interest carries a hierarchically structured variable-length identifier (HSVLI), also called the "name", of a Content Object and serves as a request for that object. If a network element sees multiple interests for the same name, it may aggregate those interests. A network element along the path of the Interest with a matching Content Object may return that object, satisfying the Interest. The Content Object follows the reverse path of the Interest to the origin(s) of the Interest. A Content Object contains the HSVLI, the object's payload, and cryptographic information used to bind the HSVLI to the payload.

This specification adds new Content Name TLV and Protocol Info TLV. It updates [ietf-draft-ccnxmessages]. It also provides guidelines for usage of the Key Locator in segmented objects.

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

TOC

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].

TOC

2. Segmentation

Segmentation, as used in this specification, means serializing user data into one or more content object segments, each encapsulated in a CCNx Content Object. One name path segment represents the current segment number. A field in the Protocol Information, only mandatory in the final segment, represents the end of the stream. Segments are denoted by a serial counter, beginning at 0 and incrementing by +1 for each contiguous segment. The segmentation ends at the final segment. No valid user data exists beyond the final segment, and reading beyond the final segment MUST NOT return any user data.

Segmentation uses a Block Size for segments. The block size may be inferred by the Content size of each content object. It is not explicit in the segmentation name protocol. It may also be represented in a Segmentation metadata object related to the Content Object. Using a consistent block size allows a reader to predict byte offsets.

The new name component is the "Segment Number" name component. It is a serial counter beginning at 0 and incrementing by 1 for each segment of the user data. Given the base name of an object and the data to segment, the Segment Number is appended to the base name.

The new Protocol Information field is the "End Segment Number". It MUST be in the last segment of a content object, but SHOULD be present at the earliest time it is known. The value of the End Segment Number should be the exact labeled name path segment of the "Segment Number", including the Type, Length, and Value fields.

The End Segment Number may be updated in later segments to a larger value, so long as it has not reached the end yet. The End Segment Number SHOULD NOT decrease. If a publisher wishes to close a stream before reaching the End Segment, it should publish empty content objects to fill out to the maximum End Segment Number ever published. These padding segments MUST contain the true end segment number.

TOC

2.1. Examples

Here are some examples of segmented names using the Labeled Content Identifier URI scheme in human readable form (lci:).

In this example, the content producer publishes a JPG that takes 4 segments. Inside the Protocol Information, the End Segment Number is missing in the first (segment 0) object, but is known when segment 1 is published so is included in Segment 1. It is omitted in Segment 2, then appears in segment 3, where it is mandatory.

```
lci:/Name=parc/Name=csl/Name=picture.jpg/Segment=0
  Protocol Info: No EndSegment
lci:/Name=parc/Name=csl/Name=picture.jpg/Segment=1
  Protocol Info: EndSegment="Segment=3"
lci:/Name=parc/Name=csl/Name=picture.jpg/Segment=2
  Protocol Info: No EndSegment
lci:/Name=parc/Name=csl/Name=picture.jpg/Segment=3
  Protocol Info: EndSegment="Segment=3"
```

In this example, the publisher is writing an audio stream that ends before expected so the publisher fills empty content objects out to the maximum segment number, stating the correct end segment. Segments 4, 5, and 6 do not contain any new user data.

```
lci:/Name=parc/Name=csl/Name=talk.wav/Segment=0
lci:/Name=parc/Name=csl/Name=talk.wav/Segment=1
```

```
   Protocol Info: EndSegment="Segment=6"
lci:/Name=parc/Name=csl/Name=talk.wav/Segment=2
lci:/Name=parc/Name=csl/Name=talk.wav/Segment=3
   Protocol Info: EndSegment="Segment=3"
lci:/Name=parc/Name=csl/Name=talk.wav/Segment=4
   Protocol Info: EndSegment="Segment=3"
lci:/Name=parc/Name=csl/Name=talk.wav/Segment=5
   Protocol Info: EndSegment="Segment=3"
lci:/Name=parc/Name=csl/Name=talk.wav/Segment=6
   Protocol Info: EndSegment="Segment=3"
```

TOC

3. Segmentation Metadata

Segmentation metadata MAY be saved along with a segmented object. If segmentation metadata is saved it SHOULD be saved using a Meta labeled path segment with the value of Segmentation.

An example name is "lci:/Name=parc/Name=picture.jpg/Meta=Segmentation". An example name using SerialNumber versioning is "lci:/Name=parc/Name=picture.jpg/SerialNumber=1/Meta=Segmentation/SerialNumber=0"

The segmentation Metadata is a JSON content object whose content adheres to the following schema. The BlockSize is used for all segments except the last. The EndSegment is optional at the time the header is written, if it is not known. Once a segmentation is finished a new header could be written with the correct EndSegment.

The optional DIGEST key is over the entire user contents. It must specify the algorithm used for the digest in OID form and express the digest in Hexadecimal.

The TOTALBYTES field is the total user bytes, if known. If it is only known at a later time, the metadata object may be updated with a new SerialNumber when the value is known.

The header MUST have the same KeyId -- and be signed by the same key -- as the content object to which it refers. If the content object is encrypted, the header MUST use the same encryption.

```
{ "SEGMENTATION" :
  { "BLOCKSIZE"  : <blocksize>
    [, "FILENAME"   : <original file name, if known>]
    [, "ENDSEGMENT" : <end segment, if known>]
    [, "TOTALBYTES" : <total bytes of all segments, if known>]
    [, "DIGEST"   : { "ALGORITHM"=<OID>, "DIGEST"=<hexadecimal value> }]
  }
}
```

TOC

4. TLV Types

This section specifies the TLV types used by CCNx Segmentation.

TOC

4.1. Name Types

CCNx Segmentation uses one new Name type, for Segment Number.

Type	Abbrev	Name	Description
%x0010	T_SEGMENT	Segment Number (Segment Number)	The current segment number, is an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

Table 1: Name Types

TOC

4.1.1. Segment Number

The current segment number, as an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

In lci: URI form, it is denoted as "Segment".

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+--------------+
|           T_SEGMENT           |            length            |
+---------------+---------------+---------------+--------------+
|     variable length integer   /
+---------------+---------------+
```

TOC

4.2. Protocol Information

CCNx Segmentation uses one new field in the Protocol Information for the End Segment Number.

Type	Abbrev	Name	Description
%x0019	T_ENDSEGMENT	End Segment Number (Segment Number)	The last segment number, as an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

Table 2: Protocol Information Types

TOC

4.2.1. End Segment Number

The ending segment number, as an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+--------------+
|         T_ENDSEGMENT          |            length            |
+---------------+---------------+---------------+--------------+
|     variable length integer   /
+---------------+---------------+
```

TOC

5. Acknowledgements

TOC

6. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

7. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

8. References

TOC

8.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

8.2. Informative References

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” RFC 2616, June 1999 (TXT, PS, PDF, HTML, XML).

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

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014

CCNx Publisher Clock Time Versioning

Abstract

This document specifies using a timestamp as a path segment component in a CCNx name as a versioning specifier. It defines the path segment label, the encoding, and the semantics.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document specifies using an RFC 3339 UTC timestamp in a CCNx name as a version identifier. It specifies a new name segment label and a TLV encoding. The use of a timestamp in a name to denote a version is limited to clock synchronization and in general should not be used to compare versions between multiple publisher.

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

TOC

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].

TOC

2. Protocol Description

A timestamp in a CCNx name path segment indicates an ordering on names based on the UTC timestamp. The timestamp is encoded in network byte order using the minimum number of bytes. The value of "0" is represented as the single byte %x00.

A publisher assigns a timestamp to indicate the time ordering of the prior name path segments. It does not imply a specific meaning, such as the time of content creation or the time of content object signature. It's only specified meaning is a time ordering of names.

A "GONE" ContentType means that this version is a terminal version. All prior versions should be interpreted as deleted. A user, however, may publish more "DATA" after the terminal version, if he decides to un-delete it.

Type	Name
'Time'	UTC Timestamp, in RFC 3339 format for human-readable format, of milliseconds since the epoch.

Table 1: Labeled Content Information Types

Type	Symbol	Name	Description
%x0010	T_TIME	UTC Timestamp	UTC timestamp in network byte order.

Table 2: CCNx Name Types

TOC

3. Acknowledgements

TOC

4. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

5. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

6. References

TOC

6.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

6.2. Informative References

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

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

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014
Expires: October 15, 2014	

CCNx Publisher Serial Versioning

Abstract

This document specifies using a serial number to indicate versions in name path segments. A serial number is an increasing unsigned integer with an increment of 1. Therefore, given one name with a serial version number, one may compute the next version number.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document specifies using a serial number in a CCNx name as a version identifier. It specifies a new name segment label and a TLV encoding. The use of a serial number in a name to denote a version is limited to coordination among publishers if an attempt is made to use a serial number as a distributed ordering.

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

TOC

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].

TOC

2. Protocol Description

A serial number in a CCNx name path segment indicates an ordering on names based on the unsigned integer. The serial number is encoded in network byte order using the minimum number of bytes. The value of "0" is represented as the single byte %x00.

The serial number must be incremented by 1 for consecutive versions of the prior name path segments. There is no maximum serial number, it is limited only by the number of bytes put in to the name component.

A "GONE" ContentType means that this version is a terminal version. All prior versions should be interpreted as deleted. A user, however, may publish more "DATA" after the terminal version, if he decides to un-delete it.

Type	Name
'Serial'	Serial number, displayed as an Integer.

Table 1: Labeled Content Information Types

Type	Symbol	Name	Description
%x0011	T_SERIAL	Serial Number	Serial Number, incrementing by +1.

Table 2: CCNx Name Types

TOC

3. Acknowledgements

TOC

4. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

5. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

6. References

TOC

6.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

6.2. Informative References

[CCNx]	PARC, Inc., "CCNx Open Source," 2007.
[RFC3552]	Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations," BCP 72, RFC 3552, July 2003 (TXT).
[RFC5226]	Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Experimental	April 13, 2014
Expires: October 15, 2014	

CCNx Selector Based Discovery

Abstract

CCNx selector based discovery uses exclusions and interest name suffix matching to discover content in the network. Participating nodes may respond with matching content objects from cache using an encapsulation protocol. This document specifies the available selectors, their encoding in a name path segment, and the encapsulation protocol.

Status of this Memo

Table of Contents

TOC

1. Introduction

Content Discovery is an important feature of CCNx. This document specifies a discovery mechanism that uses a name path segment to encode a discovery query in an Interest. Participating nodes may reply with a Content Object from cache if it matches the encoded query. The query uses exclusions to work around incorrect responses.

This document specifies a new name label for selector query. It also specifies a new type of Content Object that encapsulates another Content Object. The Encapsulation Object is used to return a content object with a longer name than in an interest. It is a standard Content Object, but its signature will not verify. The encapsulated object's signature should verify.

Note that Selector discovery is not needed when asking for a Content Object by its ContentObjectHash, as there should only ever be one match for that.

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

TOC

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].

TOC

2. Protocol Description

Selector based discovery uses four query variables to discover content. These selectors are encoded as a single name path segment affixed to an Interest name. The selectors operate on the prefix up to, but not including the selector name path segment.

The selectors are:

- MinSuffixComponents: the minimum number of additional name path segments a matching content

object must have in its name, with the Content Object Hash appended as the terminal path segment. The default value is 0.
- MaxSuffixComponents: The maximum number of additional name path segments a matching content object may have in its name, with the Content Object hash appended as the terminal path segment. The default value is unlimited.
- ChildSelector: Answer with the left-most or right-most child.
- Exclusions: A set of range and singleton exclusions to eliminate Content Objects. The exclusions match against the name path segment that would immediately follow the Interest prefix prior to the Selector path segment. They are matched against a Content Object name with the Content Object Hash appended as terminal path segment.

A node using Selector discovery appends a Selector name path segment to the end of the Interest name. Even if no selectors are used, the Selector path segment is added to the end, which indicates to a participating node that it should apply Selector based matching to the Interest.

A node receiving a Selector Interest should match against the Content Store using the selector rules. Based on the sort order, it should pick the appropriate Content Object, if any, and return it in an Encapsulation Object. If no Content Objects match, the Interest should be forwarded as normal.

An Encapsulation Object is a Content Object that matches the Selector Interest and whose payload is the ``discovered'' content object. The ContentType of an Encapsulation Object is "ENCAP". The discovered content object's name should be a suffix of the Interest name (prior to the selector name path segment). The KeyId of the Encapsulation Object should be set the same as the discovered object. The Signature of the Encapsulation Object should be set to zeros. The Crypto Suite of the Encapsulation Object should be set to the same as the discovered object.

TOC

3. Name Labels and TLV types

Type	Name
'Selectors'	The value is a binary field of the TLV encoding of the selectors.

Table 1: Selector Name Label

Type	Symbol	Name	Description
%x0001	T_MINSUFFIX	Selectors: Min Suffix Components	Minimum number of additional name components after given name to match (0 default if missing).
%x0002	T_MAXSUFFIX	Selectors: Max Suffix Components	Maximum number of additional name components after given name to match (unlimited default is missing).
%x0003	T_CHILD	Selectors: Child	0 = left, 1 = right (default)

		Selector	
%0004	T_EXCLUDES	Excludes	Encloses ExcludeComponents
%x0005	T_EX_SINGLE	Exclude Singleton	Exclude a single name path segment.
%x0006	T_EX_RANGE	Exclude Range	Exclude an inclusive range, beginning at this value and continuing through the next Singleton, or to infinity if omitted on the last entry.

Table 2: CCNx Name Types

TOC

3.1. Child Selector

If there are multiple choices to answer an Interest, the Child Selector specifies the desired ordering of responses. %x00 = leftmost, %x01 = rightmost.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+
|    T_CHILD    |     length    |   selector    |
+---------------+---------------+---------------+
```

TOC

3.2. Interest Min(Max)SuffixComponents

The Min and Max suffix components are encoded as a minimum-length unsigned integer in network byte order number inside the value. A "0" is represented as a single byte %0x00. A length 0 value is interpreted the same as the type not being present.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|     type      |     length    |                               /
+---------------+---------------+                               /
/                      Min(Max)SuffixComponents                /
/                                                               /
+---------------+---------------+---------------+---------------+
type = T_MINSUFFIX or T_MAXSUFFIX
```

TOC

3.3. Interest Excludes

Interest Excludes specify a set of singletons and ranges to exclude when matching content object names to an Interest. They match the name component immediately following the last component of the Interest name. The excludes must be sorted in ascending order, using the normal Name sorting rules.

The normal name sorting rules use a shortlex algorithm. A name component A is less than a name component B iff A is fewer bytes, or the byte size being equal, A is lexicographically sorted before B, where the most significant byte is first.

The zero-length name component is the minimum name component. If present, it must be the first Exclude component.

An exclude may contain either an Exclude Range type or an Exclude Singleton type. An Exclude Range type means the given value starts an inclusive exclusion range that ends at the next Singleton or at infinity if it is the last exclude component. An Exclude Singleton means to exclude the exact value given.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_EXCLUDES           |            length             |
+---------------+---------------+---------------+---------------+
/ Zero or more exclude-components                               /
+---------------------------------------------------------------+

exclude-components = *component [start-range-tlv]
component = (start-range-tlv singleton-tlv) / singleton-tlv
```

EXAMPLES

TOC

3.3.1. Exclude Singleton

A singleton exclude component means to exclude a name path segment exactly matching the given value.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_EX_SINGLE          |            length             |
+---------------+---------------+---------------+---------------+
/                        TLV name segment                       /
+---------------+---------------+---------------+---------------+
```

TOC

3.3.2. Exclude Range

A Range exclude means to exclude the from the given value up to an including the next Singleton. If the Range is the last component in the Exclude, it means to exclude to infinity.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_EX_RANGE           |            length             |
+---------------+---------------+---------------+---------------+
/                       TLV name segment                        /
+---------------+---------------+---------------+---------------+
```

TOC

4. Acknowledgements

TOC

5. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

6. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 (Rescorla, E. and B. Korver, “Guidelines for Writing RFC Text on Security Considerations,” July 2003.) [RFC3552] for a guide.

TOC

7. References

TOC

7.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

7.2. Informative References

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

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

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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

TOC

Internet Engineering Task Force	M. Mosko, Ed.
Internet-Draft	PARC
Intended status: Informational	August 2014
Expires: February 2, 2015	

CCNx Hash Forwarding (CCNxHF)

Abstract

This document describes a method to use fixed size, flat byte strings to forward CCNx packets with Hierarchically Structured Variable Length Identifiers (HSVLI), thus simplifying the work done at a packet forwarder. A first byte string, called the Similarity Hash (SH), represents the query in an Interest. The Similarity Hash remains invariant as a packet moves through the network. A second byte string, called the Forwarding Hash (FH), represents the longest matching prefix in the routing tables that matches the Interest name. The Forwarding Hash may change hop-by-hop if the underlying routing tables change, such that it always represents the best match at the previous hop. A Content Object, sent in response to an SH/FH Interest, carries the SH/FH header along the return path so the Content Object may be forwarded along the proper path.

Status of this Memo

Table of Contents

TOC

1. Introduction

CCNx Hash Forwarding (CCnxHF) supplants the normal forwarding rules where a long hierarchically structured variable-length identifier is used for forwarder. Instead, a hash value of the routable part of the name is pre-computed and added to the per-hop TLVs. This, along with a another pre-computed hash for Interests, allows forwarding elements to switch Interests and Content Objects without needing to process the entire name every hop.

For an Interest, a Similarity Hash is a strong hash of the entire Interest message, excluding the Interest lifetime. The hash does not include any fixed or per-hop TLV headers.

If an end-system supports a default route, then an end system may leave the forwarding hash empty in an Interest. At the first router, it will inspect the name and compute the best routing prefix for the Interest. The router will update the forwarding hash (FH) to that value. As the packet moves through the network, as long as the FH continues to match a FIB entry without children, the Interest is forwarded without inspecting the name inside the Interest. If the FIB entry is not found or it has children, then that router must inspect the name and determine the new FH.

As an Interest moves through the network, the FH may change over time. A forwarder, in the PIT, must remember the previous hop's FH, if it changes, so it may label-swap in that previous FH on the return path. This ensures that the SH and FH exactly equal what the previous hop expects.

When an Interest arrives at a node that may answer it, that node must inspect the whole interest and ensure that the Content Object returned truly matches the inner Interest. If it does, the responding node puts the Interest's SH/FH pair on the Content Object and returns it along the reverse route. The Content Object's SH/FH is matched against the PIT and then sent along the reverse paths. The forwarder label swaps the FH, as needed.

This document specifies:

- The use of a new header on Interest and Content Object packets for forwarding.
- The Similarity Hash of an Interest.
- The Forwarding Hash of a CCNx name.
- The forwarding and reverse path forwarding algorithm using the SH/FH.

- The packet formats of CCNx TLV encoded messages.

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

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].

2. Hash Function

Hash Forwarding relies on each node using the same hash function to encode name prefixes and compute similarity hashes. Therefore, we specify the hash function and its usage for Hash Forwarding.

The Similarity Hash is only computed by the source node, and optionally verified by an authoritative source node generating content or responding from a long-term repository. The Similarity Hash uses SHA-256. The Similarity Hashes will be different for ccnb or tlv wire format, as some fields have different representations. The forwarding hashes will also differ between CCNB and TLV formats due to the use of labeled path segments in TLV-encoded names.

The Forwarding Hash is used and possibly computed by forwarding nodes based on entries in their FIB table. Speed of computation is important, and collision resistance only needs to be good enough to distinguish between allowed routing names. The Forwarding Hash uses FNV-1a 128-bit (Landon Curt Noll, “FNV Hash,” 2013.) [FNV] with the standard FNV_offset and FNV_prime:

```
FNV_prime  = 2**88 + 2**8 + 0x3B
           = 309,485,009,821,345,068,724,781,371
           = 0x00000000 01000000 00000000 0000013B

FNV_offset = 144,066,263,297,769,815,596,495,629,667,062,367,629
           = 0x6C62272E 07BB0142 62B82175 6295C58D
```

Figure 1: FNV-1a constants

To compute the Similarity Hash over ccnb, hash the Interest packet from the start to the earliest of the Lifetime or Nonce or FaceId or the end of the Interest. If the hash is not to the end of the Interest, include an extra %x00 byte so the ccnb looks well-formed.

To compute the Similarity Hash over tlv, do not include the Lifetime or Nonce TLVs, if present at the end of the Interest.

To compute a Forwarding Hash over a CCNx name, run the FNV-1a 128-bit over each name component, in cumulative order, to the desired number of components.

TOC

3. Header Format

The CCNxHF header includes two fields: the Forwarding Hash (FH), and the Similarity Hash (SH). The FH and SH are as described previously.

TOC

3.1. TLV Format

CCNxHF uses a per-hop TLV, which encapsulates the SH and FH. The header is 32 bytes long, containing two 16-byte hashes.

CCNxHF usually uses HopLimit based loop prevention, so that per-hop header is present also.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_CCNXHF           |          length (32)          |
+---------------+---------------+---------------+---------------+
|                                                               |
|                    16-byte Forwarding Hash                    |
|                                                               |
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
|                    16-byte Similarity Hash                    |
|                                                               |
|                                                               |
+---------------+---------------+---------------+---------------+
```

Figure 2: CCNxHF per-hop header

TOC

4. Content Control Message Protocol

The Content Control Message Protocol (CCMP) serves a similar function as ICMP does in an IP network: it conveys information about network state.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|     %xA4      |            length             |    reserved   |
+---------------+---------------+---------------+---------------+
|   ccmp_type   |   ccmp_code   |                               |
+---------------+---------------+---------------+---------------+
/                      CCMP Body (varies by type                /
+---------------+---------------+---------------+---------------+
```

Figure 3: CCNxHF Packet

TOC

4.1. CCMP Resend

Resend Interest is sent to ask the downstream to re-send a complete interest or interest fragment stream, identified by the SH/FH/Fragment Stream ID.

```
                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|     %xA4      |            length             |    reserved   |
+---------------+---------------+---------------+---------------+
|   ccmp_type   |   ccmp_code   |                               |
+---------------+---------------+---------------+---------------+
|                                                               |
|                     16-byte Forwarding Hash                   |
|                                                               |
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
|                     16-byte Similarity Hash                   |
|                                                               |
|                                                               |
+---------------+---------------+---------------+---------------+
|                       Fragment Stream ID                      |
|                                                               |
+---------------+---------------+---------------+---------------+
```

Figure 4: CCNxHF Packet

TOC

4.2. Fragmentation

CCNxHF uses End-to-End fragmentation, with the follow differences.

The Fragment Stream ID is always random.

Fragments are identified by a serial counter FragNum, which ranges from 0 - 63. Forwarders and end systems should drop duplicate fragments, identified by the tuple {SH, FH, Fragment ID, FragNum}.

Content Objects match an Interest in the PIT based on the SH/FH matching, not the "Interest Fragment Stream ID" in the fragmentation header.

If a fragmented Interest's FH does not exactly match the LMP in the FIB, then the forwarding node must examine the Interest name and compute a new FH. This requires that the forwarding node re-assemble the front of the Interest to examine the name. In a typical case, this means that the node must receive fragment 0 to have enough prefix name components to compute the new FH. A system may discard all fragments after fragment 0, and once fragment 0 arrives and the system constructs a PIT entry with the proper FH, it should send a CCMP Resend Interest along the Fragment Stream ID's reverse path to cause the source to resend the interest stream, which can now be forwarded out of order.

TOC

5. Protocol Operation

```
                          +----+     +----+     +---+
                      /---| R1 |----| R2 |----| B |
    +---+      +----+     +----+     +----+     +---+
    | A |------| R0 |                   |
    +---+      +----+     +----+     +----+     +---+
                     \----| R3 |----| R4 |----| C |
                          +----+     +----+     +---+
```

Figure 5: Example topology

The CCNxHF protocol follows similar principles as normal CCNx: a node issues an Interest for a Content Object and receives back at most one Content Object per Interest it sends. The Content Object's name must be equal to or a suffix of the Interest name, and it must satisfy the various selectors in the Interest. CCNxHF speeds up normal CCNx processing by pre-computing the similarity hash and longest-matching-prefix (LMP) forwarding hash. The assumption is that the LMP forwarding hash does not change frequently in-route, so few, if any, intermediate systems need to do an expensive CCNx name parsing and hashing.

Figure 5 (Example topology) shows an example topology used in the following to illustrate protocol behavior. Nodes A,B,C are end nodes. Nodes R0 - R4 are routers executing CCNxHF.

CCNxHF differs from CCNx, and in particular the "ccnd" behavior, in the following key ways:

- A forwarder does not evaluate the name or selectors when matching content in the Content Store. It uses an exact match on the Similarity Hash.
- A forwarder does not check signatures, which ccnd always did.
- A forwarder does not calculate the implicit hash. Only end nodes calculate an implicit hash if necessary.
- Because a forwarder only uses an exact match on the Similarity Hash, there may be less Interest aggregation in-network.
- For TLV, CCNxHF does not use nonces, but relies on the Hop Limit field and acyclic routing tables to avoid loops.

A CCNxHF forwarder must maintain several data structures. The form of those data structures is implementation dependent, but we provide notional descriptions to facilitate protocol description.

The Pending Interest Table (PIT) tracks outstanding interests the forwarder has seen, for which the forwarder is awaiting a response. It also aggregates similar interests (Interest with the same Similarity Hash), so one Content Object may be forked to multiple reverse paths. The PIT must track the interfaces out which an Interest has been sent and ensure that similar Interests are not sent multiple times out the same interfaces.

The PIT must ensure that similar interests can flow in all directions, at most one time. A forwarder, for example, with three interfaces 1, 2 and 3, may forward an interest from interface 1 out interfaces 2 and 3. At a later time, it receives a similar interest from interface 2. It must forward that interest out interface 1, but not 3.

The PIT entry should also track the maximum hop limit forwarded. If an Interest with a longer hop limit arrives, it should be forwarded even if it is identical to a previously forwarded Interest.

The Content Store (CS) is an optional component. It stores recently seen or high-value Content Objects so later requests for the same object can be answered without forwarding an Interest. The cache policy and retention policy are beyond the scope of this document, which describes one storage and retrieval mechanism.

The Forwarding Information Base (FIB) contains the Interest forwarding routes. A routing protocol maintains the FIB. The entries in the FIB are Forwarding Hashes.

In general, we must match both the SH and FH of an Interest on the return path of a Content Object. This is because a malicious user could put in an SH for /popular/content and an FH for /colluding/site. The content object form /colluding/site would have malicious content, but an SH for /popular/content. If forwarders do not validate that the content object matches the full pending interest and only reverse path forward with the SH, the malicious content pollutes the network. However, if we match on both the SH and FH, then a beginning user who asked for /popular/content in the SH and set the SH to /popular would not get the content from /colluding/site.

To summarize the behavior of forwarding, an Interest is switched based on its Forwarding Header. If an intermediate node has a more specific route, it may update the FH to the more specific header. When a Content Object is returned, an intermediate node will re-swap the FH label. When an intermediate node receives a Content Object, it verifies that it came from the expected direction, based on the PIT entry and SH/FH headers. An exception to this is if an interest was routed along the default route (an empty FH), the FH header in the Content Object is not swapped.

A PIT entry must store the SH, which is invariant in forwarding, the ingress FH, and the egress FH. The egress FH must match a content object's FH when received, and the ingress FH is label swapped to the Content Object when it is reverse path forwarded. It is possible that the PIT must store multiple ingress FH's based on how ingress FH's are rewritten when forwarding.

TOC

5.1. Interest Forwarding

TOC

5.1.1. Initiate Interest

A first node creates a CCNx Interest as normal. It encapsulates the Interest in a CCNxHF header. It computes the Similarity Hash as specified in Section 2 (Hash Function), putting it in the SH field. If the node knows the proper Forwarding Header, it places that in the FH field.

If the original interest has a Scope set, the node will set the Hop Limit to 1 for scopes 0 and 1 and to 2 for scope 2.

No special action is taken in the Interest for the Answer Origin Kind

It then sends the CCNxHF packet to the next hop.

A first node may learn the Forwarding Header several ways:

- Hash the first name component.
- Use a directory service.
- Use the FH returned in a Content Object from a previous Interest for the same prefix.
- Encode the Forwarding Header in a CCNxHF specific link format.

TOC

5.1.2. Receive Interest

When a forwarder receives a CCNxHF Interest on an ingress interface, it performs the following actions:

- Drop an Interest with a Hop Limit of 0.
- Decrement the Hop Limit field of the Interest.
- Lookup the SH/FH in the PIT. If no entry exists, create a PIT entry, then proceed to step Check Content Store (Check Content Store). To create a PIT entry, record the SH and FH of the Interest and note the ingress port.
- Proceed to step Forward Interest (Forward Interest).

TOC

5.1.3. Check Content Store

If a forwarder implements a Content Store, it should follow a procedure similar to this. If it does not implement a Content Store, proceed to step Forward Interest (Forward Interest).

Lookup the FH in the FIB, and determine if there is a more specific route FH'. If not, set FH' = FH.

Match the SH and FH' in the Content Store. If there is an exact match, return the content object and consume the PIT entry. The returned object carries SH/FH, unless FH was the default route, in which case it carries SH/FH'. Otherwise, proceed to step Forward Interest (Forward Interest).

TOC

5.1.4. Forward Interest

- If the Interest's Hop Limit is 0, do not forward.
- Lookup the FH in the FIB.
    - Find the longest matching prefix in the FIB, based on the name of the Interest, then forward out those ports. Never forward an interest back out the port it came in on. Call the longest-matching FIB forwarding hash FH' and the set of egress interfaces E.
    - As an example, if the FIB is a hash table, look up the FH as the key. If the entry exists and has no children, use that entry. If it has children, examine the children to determine if a longer match is possible.
- Remove the interest's ingress interface from E.
- Lookup the SH/FH' in the PIT.
    - If the Interest's hop limit (as decremented above) is greater than the PIT entries "maximum hop limit", then set the PIT entry's maximum hop limit to the Interest's hop limit. Internally mark the interest as "hop limit extended."
    - Not "hop limit extended", remove any egress interfaces already used from E.
    - Link SH/FH' to SH/FH, if they are different. This may be a one to many relationship.
- If E is not empty, update the FH in the interest with the longest matching FIB hash, then forward.

TOC

5.2. Content Object Forwarding

TOC

5.2.1. Content Producer

If an end system content producer receives an Interest, it may create a Content Object that satisfies the body of the Interest and return it along the reverse path. The returned object must carry the SH/FH received in the Interest.

An end system may verify that the SH is properly calculated to match the body of the Interest.

TOC

5.2.2. Forward Object

An intermediate system receiving a Content Object will perform these actions:

- Verify the SH/FH is in the PIT, otherwise, drop it.
- Verify that the object arrived from a port over which it was forwarded, or at minimum over which the egress FH could have been forwarded. If it does not verify, drop the content object.
- Forward the object along the reverse path, label swapping the object's FH to the reverse path's FH, except if the reverse path FH was the default route (empty) in which case do not change the FH. This is done by following the links from SH/FH' to SH/FH, if any exist.
- Consume the PIT entries satisfied by the object.

TOC

5.2.3. Receive Object

An end system receiving a content object should verify that the content object actually satisfies the original Interest. It should verify the integrity of the content object's hash and signature.

TOC

6. Acknowledgements

TOC

7. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.) [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

TOC

8. Security Considerations

- Forwarders do not evaluate the contents of an Interest, but rely only on the user-provided Similarity Hash. A use could place a forged SH on an Interest and retrieve content not requested in the actual Interest.
- Forwarders do not check signatures.
- Forwarders do not verify that a Forwarding Hash corresponds to the embedded Interest Name, except in some situations as described in the protocol.
- The Martian check on SH/FH matching on the reverse path does not detect a malicious man-in-the-middle, only off-path users trying to inject content.

Fragmentation Attacks:

- CCNxHF fragmentation does not allow overwriting fragments, so IPv4/IPv6 fragmentation attacks that depend on overwriting headers do not apply.
- Exhausting buffer space with partial fragment streams are still possible attacks.
- Fragment 0 of an Interest should carry enough of the name to make a forwarding decision. A possible attack is to send fragments beyond 0 of an SH/FH that does not match a PIT entry, causing the forwarder to queue the fragments for path calculation and exhausting the buffer space.
- An attacker could send Interests with many different Minimum Path MTUs, such as from 1280 through 65535, in reverse order, causing forwarders to forward all the Interests upstream. However, at the first hop of a properly executing node, these will all be coalesced to the upstream interface's MTU, such as 1500 bytes or 9000 bytes. If an authoritative publisher customizes its fragmentation per Interest MTU, then this could cause many duplicate objects in response. However, a Content Store or publisher is free to send any matching cached object with MTU less than or equal to the specified Minimum Path MTU. Therefore, if the authoritative publisher has issued a 1280-byte fragment stream or 1500-byte fragment stream, those streams would satisfy the attack and be returned without causing new, later fragments to be issued.
- An attacker could inject Content Object fragments with the same Fragment ID as a known stream. This would require a man-in-the-middle attack. Such an attack would not be detected except at the end system re-assembling the fragments, as they would fail signature verification. Because a forwarder only forwards at most one copy per FragNum of a Fragment ID, a re-assembling system does not receive a multiplicity of fragments with the same FragNum, unless it has multiple interfaces. If a system receives a multiplicity of fragments in the same stream and same FragNum, it must combinatorially try them all to find the proper combination that re-constitutes the original packet, or it may drop the whole stream as corrupt.
- One may use hash chains and signed authenticator fragments to avoid the combinatorial re-assembly of fragments, but this is outside the current scope.

TOC

9. References

TOC

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

TOC

9.2. Informative References

[CCNx] PARC, A Xerox Company, "CCNx Open Source," 2007.

[FNV] Landon Curt Noll, "FNV Hash," 2013.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations," BCP 72, RFC 3552, July 2003 (TXT).

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," BCP 26, RFC 5226, May 2008 (TXT).

TOC

Author's Address

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