Table of Contents

CCNx Semantics

Abstract

This document describes the core concepts of the CCNx architecture and presents a minimum network protocol based on two messages: Interest and Content Object. It specifies the set of mandatory and optional fields within those messages and describes their behavior and interpretation. This architecture and protocol specification is independent of a specific wire encoding.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document describes the principles of the CCNx architecture. It describes the network protocol based on two message types: Interests and Content Objects. The description is not dependent on a specific wire format or particular encodings.

CCNx uses subjective names to identify bytes of payload. The Name combines a routable prefix with an arbitrary suffix assigned by the publisher to a piece of content. The result is a "named payload". This is different from other systems that use only self-certifying names, where the payload name is intrinsically derivable from the payload or its realization in a network object (e.g. a SHA-256 hash of the payload or network object).

The key concept of CCNx is that a subjective name is bound to a fixed payload via cryptographic operations. This implies that the fact that a given publisher bound a certain subjective name to a certain payload can be verified via cryptographic means. For example, a publisher could use a cryptographic hash over the name and payload, sign the hash, and deliver the tuple {Name, Payload, Validation}. Additional information would be included as needed by different validation mechanisms are used. A typical named payload is thus {Name, Payload, ValidationAlgorithm}.

CCNx specifies a network protocol around Interests (request messages) and Content Objects (response messages) to move named payloads. An Interest includes the Name, the desired payload, and two optional restrictions to limit responses to a specific publisher or a specific Content Object. 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: each Name should be hierarchical and is used to route towards an authoritative source. The CCNx Name looks like a URI absolute path and we use URI terminology to describe the absolute path as made up of path segments. In practice it has a binary encoding, not a text string.

The hierarchy of a CCNx Name is used for routing via the longest matching prefix in a Forwarder. The longest matching prefix is computed 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 both 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 to the requester.

There are additional optional attributes in an Interest message that can be used to select between multiple Content Objects with matching Names (it is possible that multiple publishers could issue Content Objects with the same Name). An Interest may carry an optional KeyIdRestriction and / or an optional

ContentObjectHashRestriction. If either or both of these are present, a Forwarder must ensure that they exactly match the corresponding KeyId and computed ContentObjectHash in the Content Object.

As an Interest travels the forward path following the Forwarding Information Base (FIB), it leaves behind state at each Forwarder. This state is stored in the Pending Interest Table (PIT), which tracks the ingress and egress ports as well as the Name, KeyIdRestriction (if one exists), and ContentObjectHashRestriction (if one exists) of each Interest. When a Content Object arrives at the node, it is exactly matched against that tuple to see if it satisfies any Interests in the PIT. If it does, it is returned along the matching Interest's reverse path. If it does not, the Content Object is dropped.

If multiple Interests with the same tuple {Name, KeyIdRestriction, ContentObjectHashRestriction} arrive at a node before a Content Object matching the first Interest comes back, they are grouped in the same PIT entry and their reverse paths aggregated. Thus, one Content Object might satisfy multiple pending Interests.

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

The remainder of this document describes a 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 CCNx 0.8 Interest/Content Object protocols based on exclusions, min/max suffix components, and other selectors described on CCN.org could be built on top of the minimum protocol specified here if desired.

This document is supplemented by these documents:

- CCNx Messages in TLV format: see [ccnx-mosko-tlvmessages-03].
- Fragmentation: see [ccnx-mosko-fragmentation-01] for an end-to-end fragmentation protocol.
- Object Chunking: see [ccnx-chunking-mosko-01] for object chunking protocol.
- URI Name format: see [ccnx-mosko-labeledcontent-02].

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

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

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

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

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

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

TOC

3. Names

A CCNx name is a hierarchical absolute path. Each path segment carries a label identifying 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 into smaller, signed Content Objects.

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

The path segment types KeyId and ContentObjectHash are used to name a Payload - they are not the same as the KeyIdRestriction or ContentObjectHashRestriction. Putting a KeyId in a Name, for example, means that the Name is saying something about that KeyId - it is not imposing a matching restriction to the publisher's KeyId. One might use a KeyId field in a Name segment because the Payload of the Content Object is that key. Likewise, putting a ContentObjectHash type in a Name segment only meanss the Payload is somehow related to that. The Payload, for example, might be bibliographic information about the Content Object identified by that ContentObjectHash. Putting a ContentObjectHash Name segment in a Name will not force a match against a computed ContentObjectHash.

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

Name	Description
Name Segment	A generic path segment that includes arbitrary octets.
Nonce Segment	A nonce in a path segment, 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.
ContentObjectHash	The ContentObjectHash path segment specifies the self-certifying name of a Content Object based on the entire object hash.
Application Components	An 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 need only identify path segment boundaries and be able to compare two path segments (both label and value) for equality. The Forwarder matches paths segment-by-segment against its forwarding table to determine a next hop.

TOC

4. Interests

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

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

An Interest may also have a Payload which carries state about the Interest but is not used to match a Content Object.

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

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

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

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

TOC

5. Content Objects

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

The Name is a mandatory field that identifies the contents.

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

A Content Object carries additional optional information about the Payload. Optional contents include the PayloadType and ExpiryTime. The PayloadType field identifies the type of Content Object: "DATA" implies an opaque payload; "LINK" is a Content Object with an encoded Link as a payload; "NACK" is a negative acknowledgement equivalent to the response "does not exist", etc. If it is present, ExpiryTime is a millisecond timestamp containing the number of milliseconds since the epoch in UTC of when the contents will expire. If it is not present, there is no expiration on the Content Object. Additionally, a publisher or upstream node may include a Recommended Cache Time for Content Objects. This recommendation is a guideline as to the useful lifetime of a Content Object, but may be ignored. Other protocols, such as versioning or chunking, could place other kinds of metadata in the Content Object.

TOC

6. Link

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

TOC

7. Validation

TOC

7.1. Validation Algorithm

The Validator consists of a ValidationAlgorithm that specifies how to verify the message and a ValidationPayload containing the validation output. The ValidationAlgorithm section defines the type of algorithm to use and includes any necessary additional information. The validation is calculated from the beginning of the CCNx Message through the end of the ValidationAlgorithm section. Some Validators contain a KeyId, identifying the publisher authenticating the Content Object. If an Interest carries a KeyIdRestriction, then that KeyIdRestriction must exactly match the Content Object's KeyId. Validation Algorithms fall into three categories: MICs, MACs, and Signatures. Validators using MIC algorithms do not need to provide any additional information; they may be computed and verified based only on the algoritm (e.g. CRC32C). MAC validators require the use of a KeyId identifying the secret key used by the authenticator. Because MACs are usually used between two parties that have already exchanged secret keys via a key exchange protocol, the KeyId may be any agreed-upon value to identify which key is used. Signature validators use publis key cryptography such as RSA, DSA, or Elliptical Curve (EC). The KeyId field in the ValidationAlgorithm identifies the public key used to verify the signature. A signature may optionally include a KeyLocated, as described above, to bundle a Key or Certificate or KeyName.

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

TOC

8. Interest to Content Object matching

A Content Object satisfies an Interest if and only if (a) the Content Object name exactly matches the Interest name, and (b) the ValidationAlgorithm KeyId of the Content Object exactly equals the Interest KeyIdRestriction, and (c) the computed ContentObjectHash exactly equals the Interest ContentObjectHashRestriction, if given. A Content Object does not carry the ContentObjectHash as an expressed field, it must be calculated in network to match against. It is sufficient within an autonomous system to calculate a ContentObjectHash at a border router and carry it via trusted means within the autonomous system. If a Content Object ValidationAlgorithm does not have a KeyId then the Content Object cannot match and Interest with a KeyIdRestriction.

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

9. Protocol Behavior

As an Interest moves through the network following the FIB table based on longest matching prefix, it leaves state at each forwarding node. The state is represented in a Pending Interest Table (PIT). The PIT tracks the Name, KeyIdRestriction, and ContentObjectHashRestriction to be matched by a Content Object. If a second Interest arrives with the same name and selector values, it 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.

When a Content Object arrives at a Forwarder, it is matched against the Interests in the PIT. For each matching Interest, the Content Object is forwarded along the reverse path of that PIT entry and the PIT entry is removed.

A Content Object that does not match a PIT entry is dropped.

A Forwarder may implement a Content Object cache called a Content Store. At the core protocol level, the Content Store must obey similar rules as the Forwarder. If an Interest specifies a ContentObjectHashRestriction, the Content Store should not respond unless it has verified the hash of the Content Object. If the Interest carries a KeyIdRestriction then the Content Store must cryptographically verify the signature or not respond. 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. 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 Validator is invalid, or (b) the ContentObjectHash does not self-certify. Failure (a) can happen because signatures are not checked in-network. Failure (b) can happen 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 ContentObjectHashRestriction and the node received a Content Object that did not match, it is likely that a malicious node is somewhere in the forwarding path. The issuer should attempt re-expressing the Interest. Correctly behaving Forwarders should not reply to a ContentObjectHashRestriction 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, as in the original protocols. Exclusions are not scalable under an attack, as an attack could generate an endless number of objects to exclude.

TOC

10. Acknowledgements

TOC

11. 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

12. Security Considerations

We cover whether a key mechanically verifies a signature, but not the broader issue of trusting a key.

TOC

13. References

TOC

13.1. 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

13.2. Informative References

[CCN] 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	December 3, 2014
Expires: June 6, 2015	

TLV Packet Format

Abstract

This document defines a general format for network packets that uses Type-Length-Value (TLV) encoding. It consists of a fixed header that defines the version and type of message, a set of hop-by-hop headers, and a payload.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document describes a TLV-based packet format for CCNx messages. 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 immediate access to the basic fields needed for parsing. The hop-by-hop header fields can be used to extend the functionality of the static header as needed.

The maximum TLV payload is 64KiB.

In this document, packets are represented as 32-bit wide words using ASCII art. Due to the nested levels of TLV encoding and the presence of optional fields and variable 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 bit widths, 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

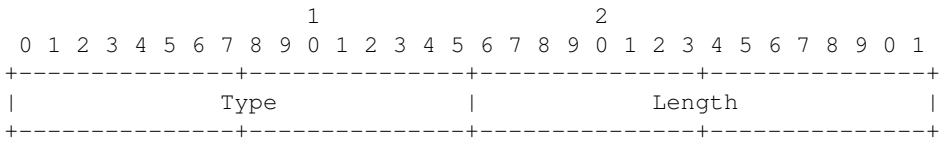
2. Definitions

- TLV: Type-Length-Value. This is the commonly used term to refer to the organization of the Type, Length and Value fields. The Type field may be treated as a label or tag. For example, the type "hop count" implies both the purpose of the field and the type of the data (an unsigned integer)

TOC

3. Type-Length-Value Structure

We use 16-bit Type and 16-bit Length fields to encode TLV based packets. This provides 64K different possible types and value field lengths of up to 64KiB. With 64K possible types, there should be sufficient space for basic protocol types, while also allowing ample room for experimentation, application use, and growth. In the event that more space is needed, either for types or for length, a new version of the protocol would be needed.



No types are defined in this document. A protocol using this format must define the types and type semantics it needs for operation.

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

TLV structures are nestable, allowing the Value field of one TLV structure to contain another TLV structure. The enclosing TLV structure is called the container of the enclosed TLV.

TOC

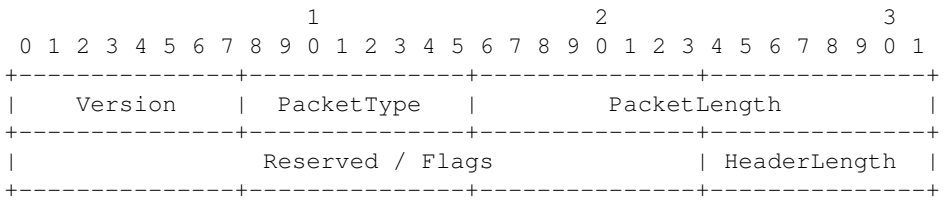
4. Fixed Headers

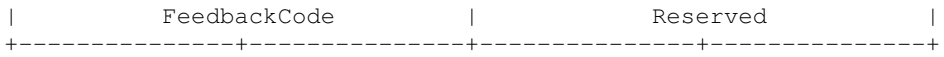
CCNx messages begin with a 12 byte fixed header (in a non-TLV format) followed by a set of hop-by-hop headers (in TLV format) and the protocol message (in TLV format). The HeaderLength field in the fixed header represents the combined length of the fixed and hop-by-hop headers, therefore the beginning of the protocol message is found at "packet start + HeaderLength".

The payload of a CCNx TLV packet is the protocol message itself. The Content Object Hash is computed over the payload only, excluding the fixed and hop-by-hop headers as those may change from hop to hop. Similarly, signed information or Similarity Hashes should not include any of the fixed or hop-by-hop headers. The payload should be self-sufficient in the event that the fixed and hop-by-hop headers are removed.

TOC

4.1. Common Fixed Header





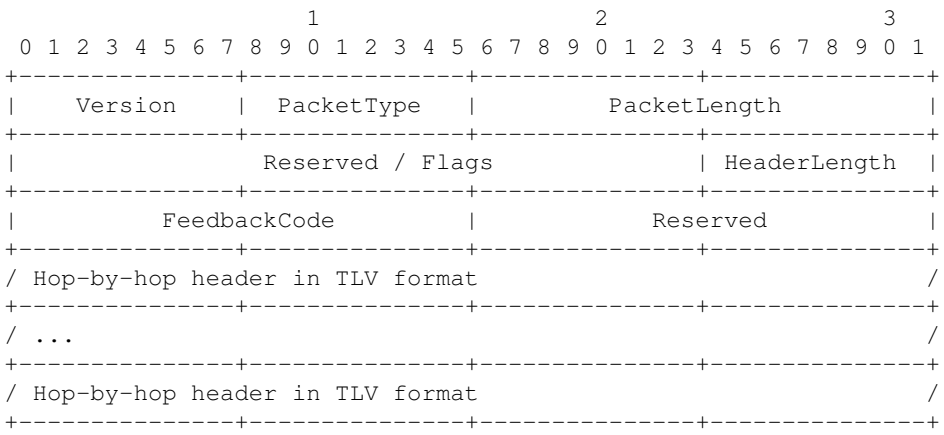
- Version: defines the version of the packet.
- HeaderLength: The length of the fixed header (12 bytes) + hop-by-hop headers. The minimum value is "12".
- PacketType: 0 = Request (Interest), 1 = Response (Content Object), 2 = Feedback, 3 = Control.
- PacketLength: Total octets of packet including all headers (fixed header plus hop-by-hop headers) and protocol message.

TOC

4.2. Hop-by-hop Headers in TLV format

Between the fixed header and the beginning of the CCNx protocol message are the hop-by-hop headers in TLV format. They are for per-hop processed options, such as a DSCP-equivalent field. The hop-by-hop headers are unordered. The order in which they are processed MUST NOT affect the result of processing other hop-by-hop headers.

Optional headers and their semantics are described in [ietf-draft-CCNmessages-mosko-03].



TOC

4.3. Message Body

The CCNx protocol message begins immediately after the hop-by-hop headers in TLV format. The message body is encoded with the same TLV structure as these 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

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

[CCN] 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 December 3, 2014
Expires: June 6, 2015

CCNx Messages in TLV Format

Abstract

This document specifies the encoding of CCNx messages using the TLV Packet specification described in [mosko-tlvpackets-02]. CCNx messages follow the CCNx Semantics specification decsribed in [mosko-semantics-02]. 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 value encodings for the CCNx protocol. This draft describes the mandatory and common optional fields of Interests and Content Objects. Several additional protocols specified in their own documents are in use that extend this specification.

CCNx specifies a network protocol around Interests (request messages) and Content Objects (response messages) to move named payloads. An Interest includes the Name, two optional restrictions to limit responses to a specific publisher or a specific Content Object, and an optional Payload used to compute a response. 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: each Name should be hierarchical and is used to route towards an authoritative source. The CCNx Name looks like a URI absolute path and we use URI terminology to describe the absolute path as made up of path segments.

A full description of the semantics of CCNx messages, providing an encoding-free description of CCNx messages and message elements, may be found in [CCN-mosko-semantics].

In the final draft, the type values will be assigned to be compact. All type values are relative to their parent containers. It is possible for a TLV to redefine a type value defined by its parent. For example, each level of a nested TLV structure might define a "type = 1" with a completely different meaning.

This document specifies:

- The TLV types used by CCNx messages.
- The encoding of values for each type.
- 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-02] for the protocol operation regarding messages and message elements.
- Fragmentation: see [ccnx-mosko-fragmentation-01] for an end-to-end fragmentation protocol.
- Object Chunking: see [ccnx-mosko-chunking-01] for object chunking protocol.
- URI Name format: see [ccnx-mosko-labeledsegments-02].

Packets are represented as 32-bit wide words using ASCII art. Due to the nested levels of TLV encoding and the presence of optional fields and variable 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 bit widths, 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

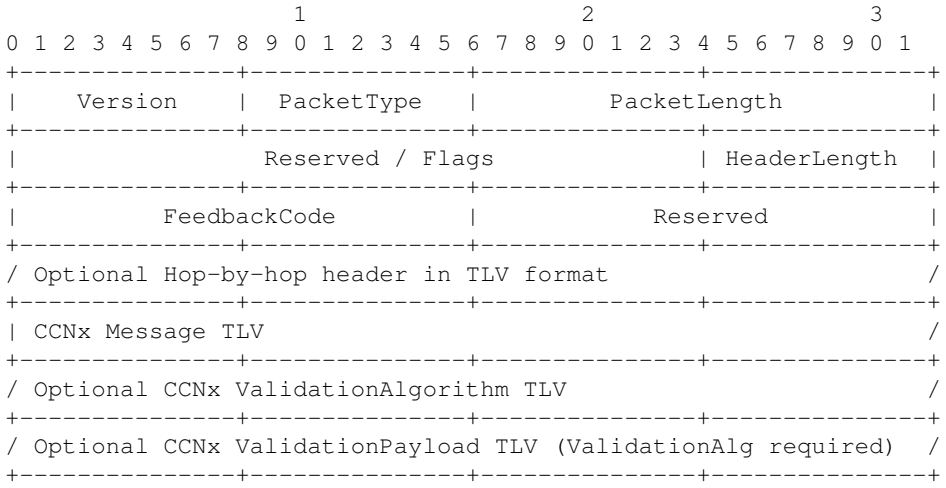
2. Definitions

- 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 message requesting a Content Object with a matching Name and other optional selectors to choose from multiple objects with the same Name. Any Content Object with a Name and optional selectors that matches the Name and optional selectors of the Interest is said to satisfy the Interest.
- Content Object: A data object sent in response to an Interest request. It has an HSVLI Name and a content payload that are bound together via cryptographic means.

TOC

3. Type-Length-Value (TLV) Packets

CCNx over TLV uses the TLV syntax described in [mosko-tlvpackets-02]. Each packet includes a 12 byte fixed header, followed by a set of hop-by-hop headers in TLV format, followed by a payload. The packet payload is a TLV encoding of the CCNx message, followed by the optional Validation TLVs.



After discarding the fixed and hop-by-hop headers the remaining payload should be a valid protocol message. Therefore, the payload always begins with a 4 byte TLV defining the protocol message (whether it is an Interest or a Content Object or a future message type) and its total length.

The embedding of a self-sufficient protocol data unit inside the fixed and hop-by-hop 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:

```
PacketLength = total packet octets
   = HeaderLength + size of CCNx Message + size of ValidationAlgorithm
                    + size of ValidationPayload
```

It is acceptable to have a 0-length payload, in which case all signaling is done in the fixed and hop-by-hop headers and PacketLength = HeaderLength.

The range of bytes protected by the Validation includes the CCNx Message and the ValidationAlgorithm.

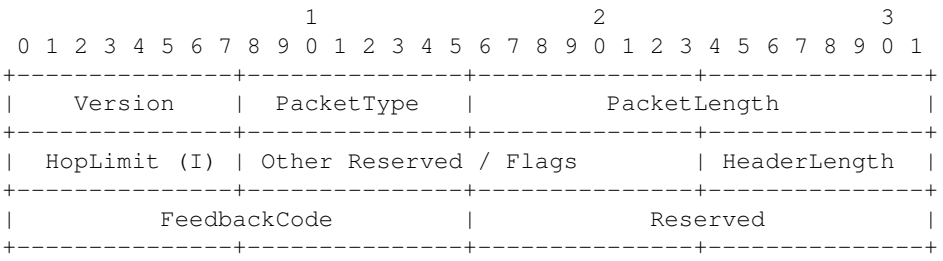
The ContentObjectHash begins with the CCNx Message and ends at the tail of the packet.

TOC

3.1. Fixed Header

The fixed header is 12 bytes and includes the Version, PacketType, PacketLength, HeaderLength, and FeedbackCode along with some flags specific to different PacketTypes. As an example, for Interest messages, the fifth byte contains an Interest HopLimit value. The total length of the hop-by-hop headers plus the length of the fixed header is captured in the HeaderLength field. Therefore the beginning of the protocol message is

at packet start + HeaderLength.



* HopLimit for Interests only

TOC

3.1.1. Packet Type

The PacketType field indicates how the forwarder should process the packet. A Request Packet has PacketType 0, a Response has PacketType 1, a Feedback Packet has PacketType 2, and a Control Packet has PacketType 3. For a Request PacketType, a Forwarder will try to match the request to a response in its cache (ContentStore). If that is not possible, it will either aggregate with an Interest already in the PIT, or store state in a new PIT entry, and forward along the FIB. For a Response PacketType, a Forwarder will match against the PIT for reverse path routing, clear the matched PIT state, and optionally save in the ContentStore. Forwarder behavior upon receiving a PacketType of 2 or 3 will be specified in a later document.

TOC

3.1.2. Interest HopLimit

For an Interest message (i.e. PacketType = 0), the HopLimit is included in the 5th byte of the Fixed Header. The HopLimit is a counter that is decremented with each hop. It limits the distance an Interest may travel on the network. The node originating the Interest may put in any value - up to the maximum of 255 - in network byte order. Each node that receives an Interest with a HopLimit decrements the value upon reception. If the value is 0 after the decrement, the Interest cannot be forwarded off the node.

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

TOC

3.2. Hop-by-hop TLV headers

Hop-by-hop TLV headers are unordered and no meaning should be attached to their ordering. Two hop-by-hop headers - the InterestLifetime for Interests and the RecommendedCacheTime for Content Objects - are described in this document. Additional hop-by-hop headers are defined in higher level specifications

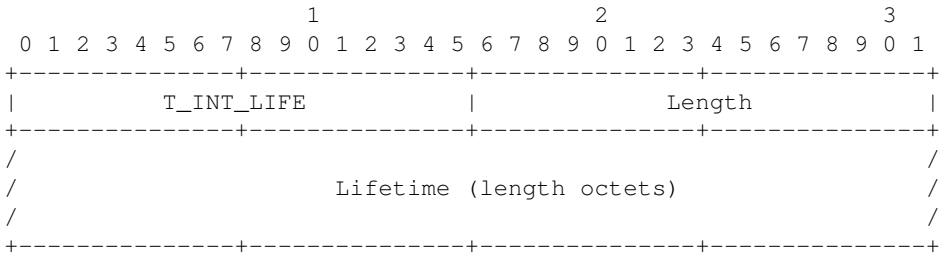
such as the fragmentation and chunking specifications.

TOC

3.2.1. Lifetime

The Interest Lifetime is the time that an Interest should stay pending at an intermediate node. It is expressed in milliseconds as an unsigned, network byte order integer.

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



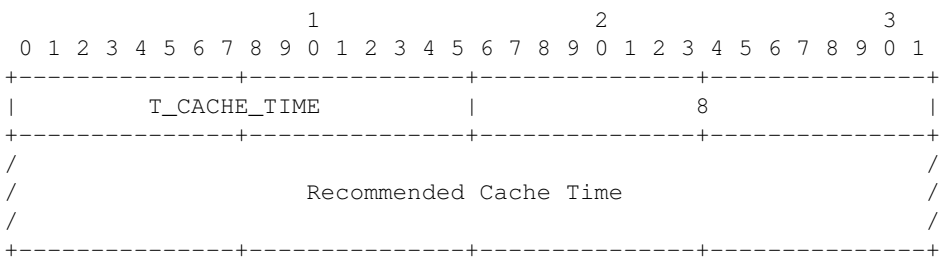
TOC

3.2.2. Recommended Cache Time

The Recommended Cache Time (RCT) is a measure of the useful lifetime of a Content Object as assigned by a content producer or upstream node.. It serves as a guideline to the Content Store cache in determining how long to keep the Content Object. It is a recommendation only and may be ignored by the cache. This is in contrast to the ExpiryTime (described in Section 3.5.2.2.2 (ExpiryTime))which takes precedence over the RCT and must be obeyed.

Because the Recommended Cache Time is an optional hop-by-hop header and not a part of the signed message, a content producer may re-issue a previously signed Content Object with an updated RCT without needing to re-sign the message. There is little ill effect from an attacker changing the RCT as the RCT serves as a guideline only.

The Recommended Cache Time (a millisecond timestamp) is a network byte ordered unsigned integer of the number of milliseconds since the epoch in UTC of when the payload expires. It is a 64-bit field.



TOC

3.3. Top-Level Types

The top-level TLV types listed below exist at the outermost level of a CCNx protocol message.

Type	Abbrev	Name	Description
%x0001	T_INTEREST	Interest (CCNx Message)	An Interest MessageType.
%x0002	T_OBJECT	Content Object (CCNx Message)	A Content Object MessageType
%x0003	T_VALIDATION_ALG	Validation Algorithm (Validation Algorithm)	The method of message verification such as Message Integrity Check (MIC), a Message Authentication Code (MAC), or a cryptographic signature.
%x0004	T_VALIDATION_PAYLOAD	Validation Payload (Validation Payload)	The validation output, such as the CRC32C code or the RSA signature.

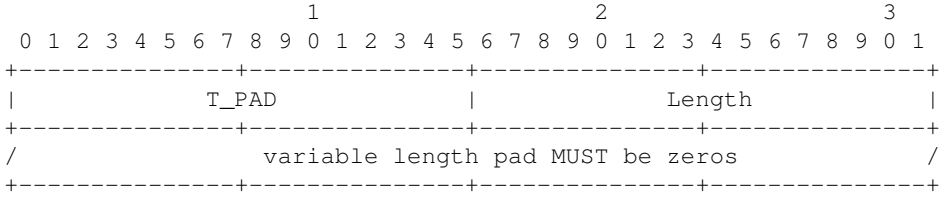
Table 1: CCNx Top Level Types

TOC

3.4. Global Formats

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.

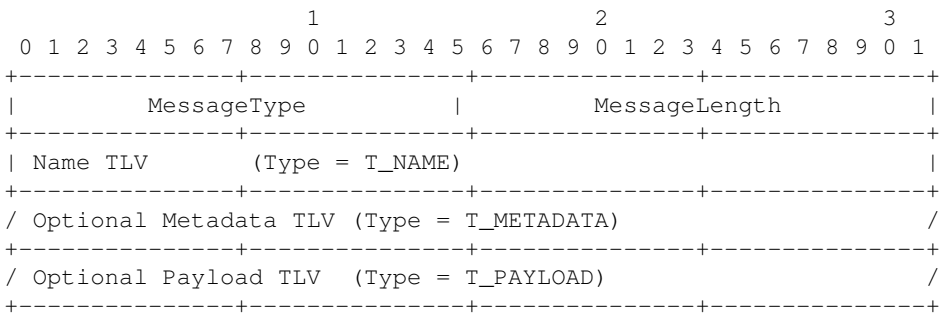
A pad may be inserted after any TLV except within a Name TLV. In the remainder of this document, we will not show optional pad TLVs.



TOC

3.5. CCNx Message

This is the format for the CCNx protocol message itself. The CCNx message is the portion of the packet between the hop-by-hop headers and the Validation TLVs. The figure below is an expansion of the "CCNx Message TLV" depicted in the beginning of Section 3 (Type-Length-Value (TLV) Packets). The CCNx message begins with MessageType and runs through the optional Payload. The same general format is used for both Interest and Content Object messages which are differentiated by the MessageType field. The first enclosed TLV of a CCNx Message is always the Name TLV. This is followed by an optional Metadata TLV and an optional Payload TLV.



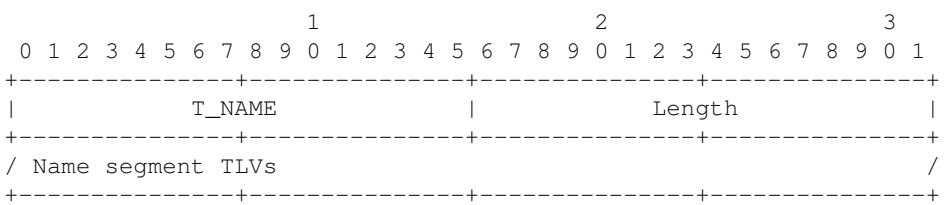
Type	Abbrev	Name	Description
%x0000	T_NAME	Name (Name)	The CCNx Name requested in an Interest or published in a Content Object.
%x0001	T_METADATA	Metadata (Metadata)	A container for protocols to place their own TLVs.
%x0002	T_PAYLOAD	Payload (Payload)	The message payload.

Table 2: CCNx Message Types

TOC

3.5.1. Name

A Name is a TLV encoded sequence of segments. The table below lists the type values appropriate for these Name segments. A Name MUST NOT include PAD TLVs.



Type	Symbolic Name	Name	Description
%x0001	T_NAMESEGMENT	Name segment (Name Segments)	A generic name Segment.
%x0002	T_NAMENONCE	Nonce Segment (Name Segments)	A nonce name segment, to generate a unique name in an Interest.
%x0003	T_NAMEKEY	KeyId (Name Segments)	The digest of a cryptographic key that identifies the publisher.
%x0004	T_OBJHASH	ContentObjectHash (Name Segments)	The ContentObjectHash component of the Name refers to the self-certifying name of a Content Object based on the entire object hash.
%xF000 - %xF0FF	T_APP:00 - T_APP:255	Application Components (Name Segments)	Application-specific payload in a name segment. An application may apply its own semantics to the 256 payload components.

Table 3: CCNx Name Types

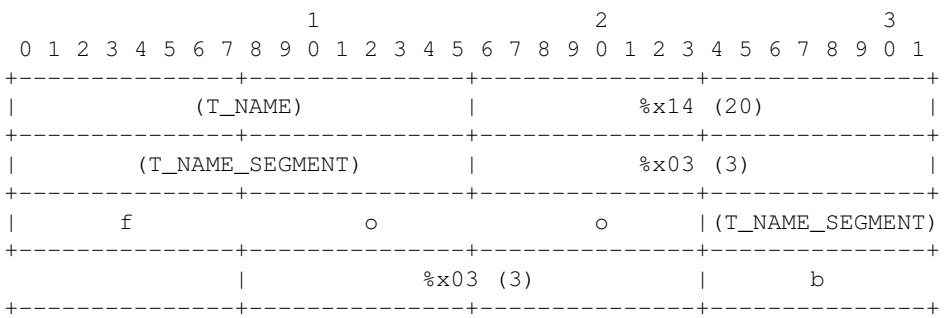
The path segment types KeyId and ContentObjectHash are used to name a Payload - they are not the same as the KeyIdRestriction or ContentObjectHashRestriction. Putting a KeyId in a Name, for example, means that the Name is saying something about that KeyId - it is not imposing a matching restriction to the publisher's KeyId. One might use a KeyId field in a Name segment because the Payload of the Content Object is that key. Likewise, putting a ContentObjectHash type in a Name segment only meanss the Payload is somehow related to that hash. The Payload, for example, might be bibliographic information about the Content Object identified by that ContentObjectHash. Putting a ContentObjectHash Name segment in a Name will not force a match against a computed ContentObjectHash.

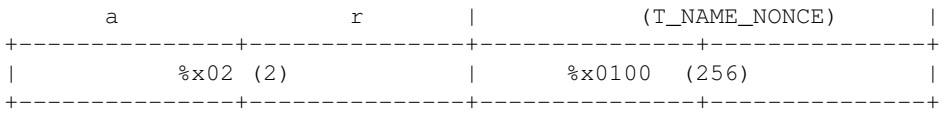
TOC

3.5.1.1. Name Segments

Special application payload name segments are in the range %x1000 - %1FFF. 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 segments.

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





TOC

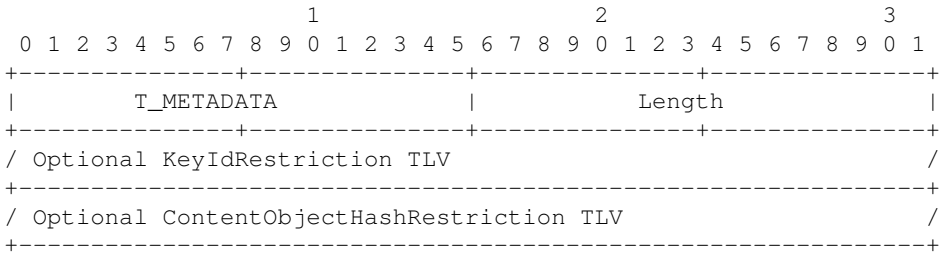
3.5.2. Metadata

The metadata section of a CCNx message is a container for protocol specific TLVs. Each message type (Interest or Content Object) is associated with a set of optional Metadata TLVs. Additional specification documents may extend the types associated with each.

TOC

3.5.2.1. Interest Metadata

There are two Metadata TLVs currently associated with an Interest message: the KeyIdRestriction selector and the ContentObjectHashRestriction selector. These two selectors are used to narrow the universe of acceptable Content Objects that would satisfy the Interest.



Type	Abbrev	Name	Description
%x0001	T_KEYID	KeyIdRestriction (KeyIdRestriction)	An octet string identifying the specific publisher signing key that would satisfy the Interest.
%x0002	T_OBJHASH	ContentObjectHashRestriction (ContentObjectHashRestriction)	The SHA-256 hash of the specific Content Object that would satisfy the Interest.

Table 4: CCNx Interest Metadata Types

TOC

3.5.2.1.1. KeyIdRestriction

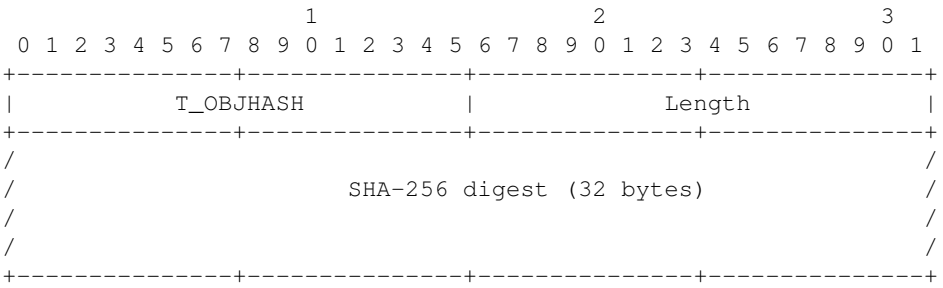
An Interest may include a KeyIdRestriction selector. This ensures that only Content Objects with matching KeyIds will satisfy the Interest. See Section 3.6.1.4.1 (KeyId) for the format of a KeyId.

TOC

3.5.2.1.2. ContentObjectHashRestriction

An Interest may also contain a ContentObjectHashRestriction selector. This is the SHA-256 hash of the Content Object - the self-certifying name restriction that must be verified in the network, if present.

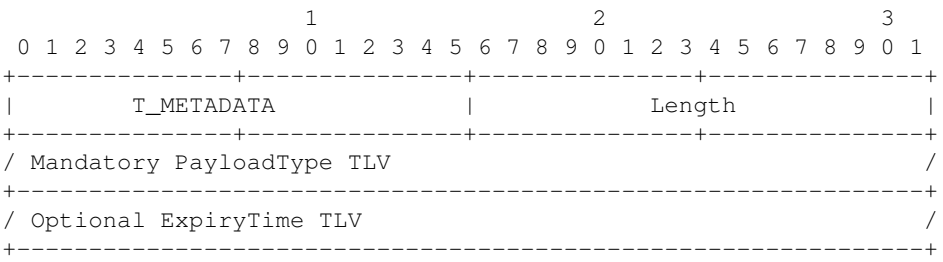
The only acceptable length is 32.



TOC

3.5.2.2. Content Object Metadata

The following metadata TLVs are currently defined for Content Objects: PayloadType (required) and ExpiryTime (optional).



Type	Abbrev	Name	Description
%x0003	T_PAYLDTYPE	PayloadType (PayloadType)	Indicates the type of Payload contents (e.g. 0 = data, 1 = encrypted data, 2 = key, 3 = link, etc.)

%x0004	T_EXPIRY	ExpiryTime (ExpiryTime)	The time at which the Payload expires, as expressed in the number of milliseconds since the epoch in UTC. If missing, Content Object may be used as long as desired.

Table 5: CCNx Content Object Metadata Types

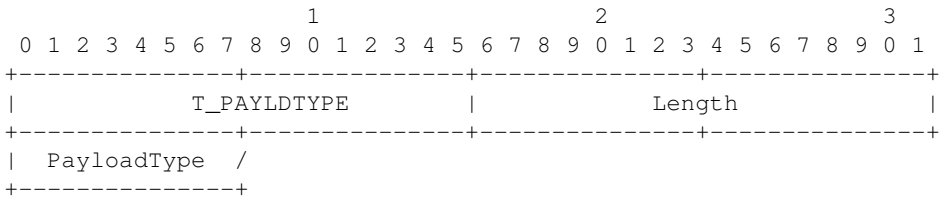
TOC

3.5.2.2.1. PayloadType

The PayloadType is a network byte order integer representing the general type of the Payload TLV.

- 0: Data
- 1: Encrypted Data
- 2: Key
- 3: Link
- 4: Certificate
- 5: Gone (whiteout)
- 6: NACK
- 7: Manifest

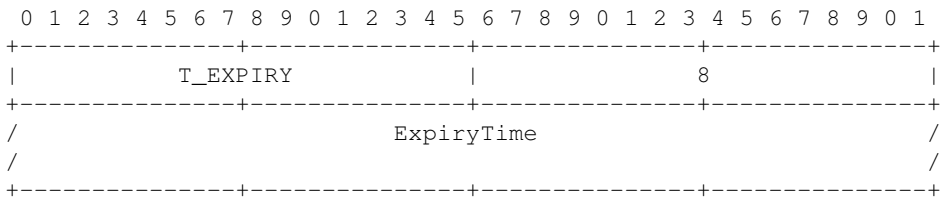
The Data and Encrypted Data types indicate that the Payload of the ContentObject is opaque application bytes. The Key type indicates that the Payload is a DER encoded public key. The Certificate type indicates that the Payload is a DER encoded X.509 certificate. The Link type indicates that the Payload is a Link (Sec 3.6.1.4.5). The Gone type indicates that all previous versions are no longer valid; it is only meaningful when combined with a versioning protocol. NACK indicates that the requested Content Object does not exist. A Manifest type indicates that the Payload is a Manifest (format TBD).



TOC

3.5.2.2.2. ExpiryTime

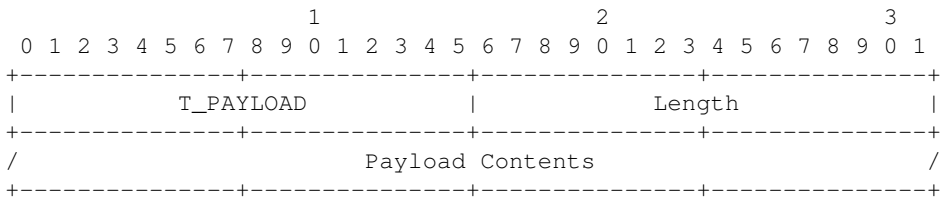
The ExpiryTime is the time at which the Payload expires, as expressed by a timstamp containing the number of milliseconds since the epoch in UTC. It is a network byte order unsigned integer in a 64-bit field. A cache or end system should not respond with a Content Object past its ExpiryTime. Routers forwarding a Content Object do not need to check the ExpiryTime. If the ExpiryTime field is missing, the Content Object has no expressed expiration and a cache or end system may use the Content Object for as long as desired.



TOC

3.5.3. Payload

The Payload TLV contains the content of the packet.



TOC

3.6. Validation

Both Interests and Content Objects have the option to include information about how to validate the CCNx message. This information is contained in two TLVs: the ValidationAlgorithm TLV and theValidationPayload TLV. The ValidationAlgorithm TLV specifies the mechanism to be used to verify the CCNx message. Examples include verification with a Message Integrity Check (MIC), a Message Authentication Code (MAC), or a cryptographic signature. The ValidationPayload TLV contains the validation output, such as the CRC32C code or the RSA signature.

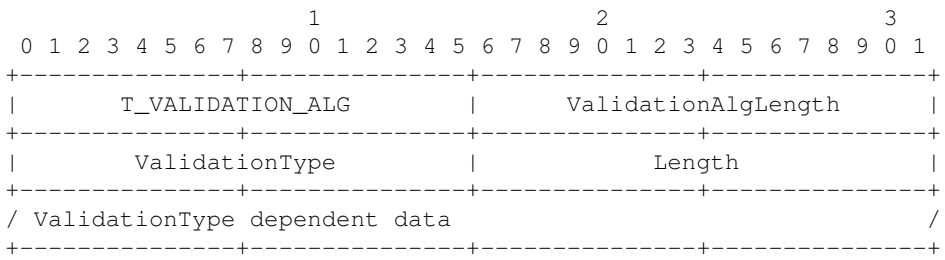
An Interest would most likely only use a MIC type of validation - a crc, checksum, or digest.

TOC

3.6.1. Validation Algorithm

The ValidationAlgorithm is a set of nested TLVs containing all of the information needed to verify the message. The outermost container has type = T_VALIDATION_ALG. The first nested TLV defines the specific type of validation to be performed on the message. The type is identified with the "ValidationType" as shown in the figure below and elaborated in the table below. Nested within that container are the TLVs for any ValidationType dependent data, for example a Key Id, Key Locator etc.

Complete examples of several types may be found in Section 3.6.1.5 (Validation Examples)



Type	Abbrev	Name	Description
%x0001	T_CRC32	CRC32 (Message Integrity Checks)	The Ethernet CRC32 (normal form polynomial 0x04C11DB7).
%x0002	T_CRC32C	CRC32C (Message Integrity Checks)	Castagnoli CRC32 (iSCSI, ext4, etc.), with normal form polynomial 0x1EDC6F41.
%x0003	T_RFC793	RFC793 (Message Integrity Checks)	The TCP checksum.
%x0004	T_HMAC-SHA256	HMAC-SHA256 (Message Authentication Checks)	HMAC (RFC 2104) using SHA256 hash.
%x0005	T_VMAC-128	VMAC-128 (Message Authentication Checks)	VMAC with 128bit tags (http://www.fastcrypto.org/vmac/draft-krovetz-vmac-01.txt).
%x0006	T_RSA-SHA256	RSA-SHA256 (Signature)	RSA public key signature using SHA256 digest.
%x0007	EC-SECP-256K1	SECP-256K1 (Signature)	Elliptic Curve signature with SECP-256K1 parameters (http://www.secg.org/collateral/sec2_final.pdf).
%x0008	EC-SECP-384R1	SECP-384R1 (Signature)	Elliptic Curve signature with SECP-384R1 parameters (http://www.secg.org/collateral/sec2_final.pdf).

Table 6: CCNx Validation Types

TOC

3.6.1.1. Message Integrity Checks

MICs do not require additional data in order to perform the verification. Examples are CRC32, CRC32C, RFC793, etc., that have a "0" length value.

TOC

3.6.1.2. Message Authentication Checks

MACs are useful for communication between two trusted parties who have already shared private keys. Examples include T_HMAC-SHA256 or others. They rely on a KeyId. Some MACs might use more than a KeyId, but those would be defined in the future.

TOC

3.6.1.3. Signature

Signature type Validators specify a digest mechanism and a signing algorithm to verify the message. Examples include T_RSA-SHA256, Elliptic Curve, etc. These Validators require a KeyId and a mechanism for locating the publishers public key (a KeyLocator) - optionally a PublicKey or Certificate or KeyName.

TOC

3.6.1.4. Validation Dependent Data

Different Validation Algorithms require access to different pieces of data contained in the ValidationAlgorithm TLV. As described above, Key Ids, Key Locators, Public Keys, Certificates, Links and Key Names all play a role in different Validation Algorithms.

Following is a table of CCNx ValidationType dependent data types:

Type	Abbrev	Name	Description
%x0009	T_KEYID	SignerKeyId (KeyId)	An identifier of the shared secret or public key associated with a MAC or Signature. Typically the SHA256 hash of the key.
%x000A	T_PUBLICKEYLOC	PublicKeyLocator (KeyName)	A data structure that tells how to locate the public key associated with a signing. It may contain a Public Key, a Certificate, or a Link to a Key.
%x000B	T_PUBLICKEY	Public Key (Public	DER encoded public key.

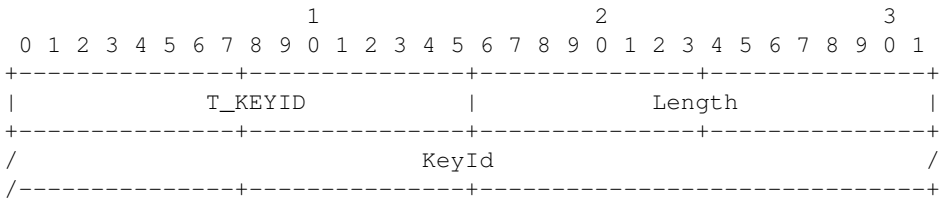
		Key)	
%x000C	T_CERT	Certificate (Certificate)	DER encoded X509 certificate.
%x000D	T_LINK	Link (Link)	A CCNx Link object.
%x000E	T_KEYNAME	KeyName (KeyName)	A CCNx Link object.
%x000F	T_SIGTIME	SignatureTime (SignatureTime)	A millsecond timestamp indicating the time when the signature was created.

Table 7: CCNx Validation Dependent Data Types

TOC

3.6.1.4.1. KeyId

The KeyId is the publisher key identifier. It 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 both public/private key systems and to symmetric key systems.



TOC

3.6.1.4.2. KeyLocator

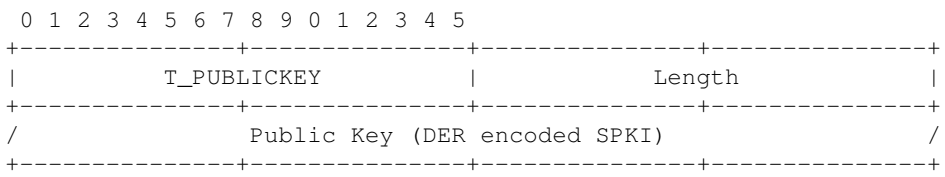
The KeyLocator is an optional field. If it is not present, a node wishing to authenticate a Content Object must have prior knowledge of the Publisher KeyId, or be able to retrieve the corresponding key through external means. The KeyLocator may contain one of: a DER encoded Public Key, a DER encoded X509 Certificate, or a KeyName.

TOC

3.6.1.4.3. Public Key

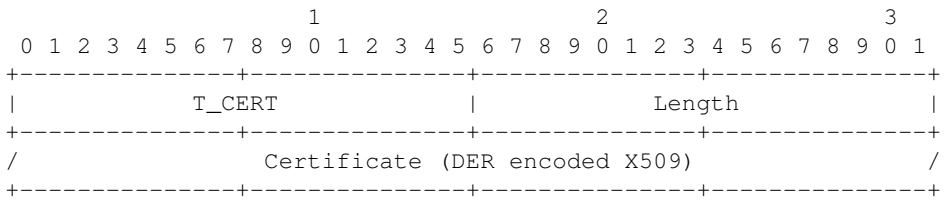
A Public Key is a DER encoded Subject Public Key Info block, as in an X509 certificate.

1



TOC

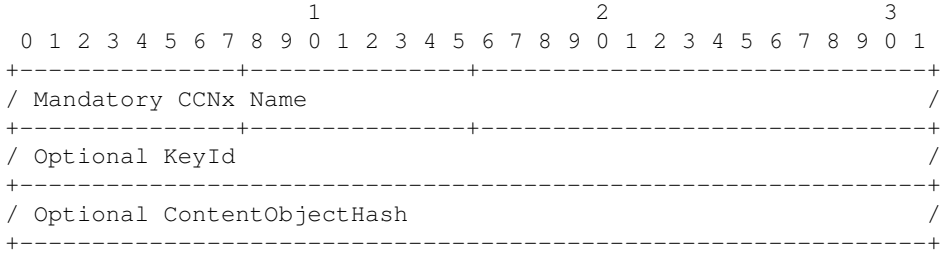
3.6.1.4.4. Certificate



TOC

3.6.1.4.5. Link

A Link is the tuple: {CCNx Name, KeyId, ContentObjectHash}. It may be the payload of a Content Object with PayloadType = "Link". Alternatively, it could be the KeyName field in a KeyLocator.

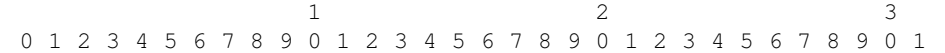


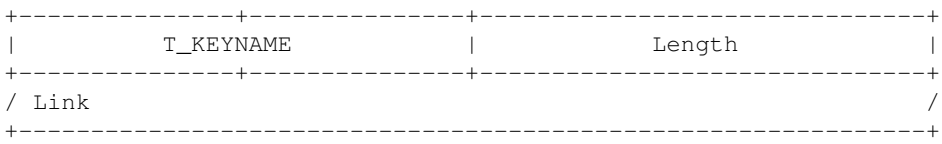
TOC

3.6.1.4.6. KeyName

A KeyName type KeyLocator is a Link.

The KeyName digest is the publisher digest of the Content Object identified by KeyName. It may be included on an Interest's digest restriction. A KeyName is a mandatory Name and an optional KeyId. The KeyId inside the KeyLocator may be included in an Interest's KeyId to retrieve only the specified key.



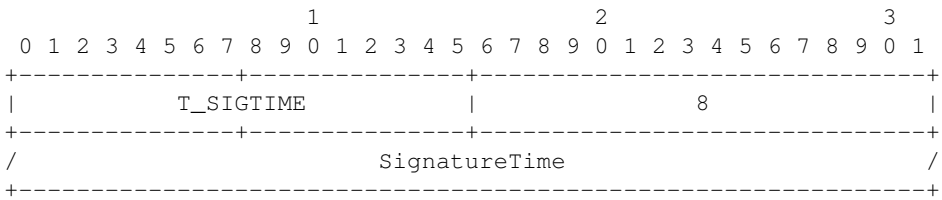


TOC

3.6.1.4.7. SignatureTime

The SignatureTime is a millisecond timesamp indicating the time at which a signature was created. The signer sets this field to the current time when creating a signature. A verifier may use this time to determine whether or not the signature was created during the validity period of a key, or if it occurred in a reasonable sequence with other associated signatures. The SignatureTime is unrelated to any time associated with the actual CCNx Messge, which could have been created long before the signature. The default behavior is to always include a SignatureTime when creating an authenticated message (e.g. HMAC or RSA).

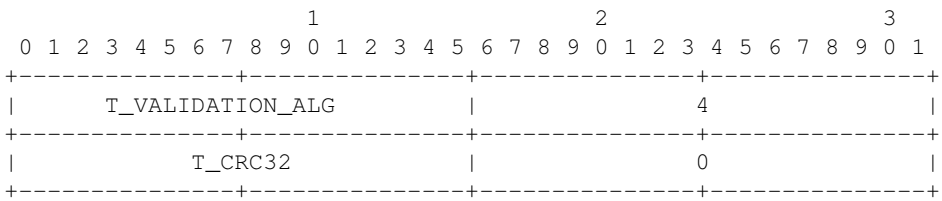
SignatureTime is a network byte ordered unsigned integer of the number of milliseconds since the epoch in UTC of when the signature was created. It is a fixed 64-bit field.



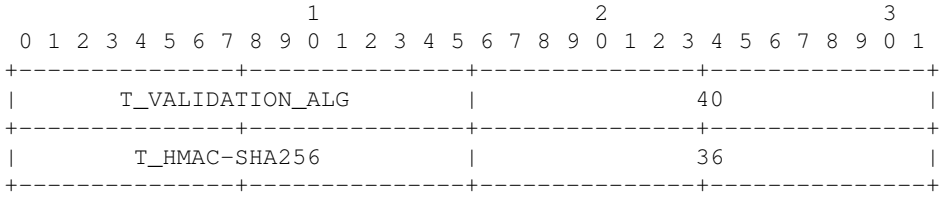
TOC

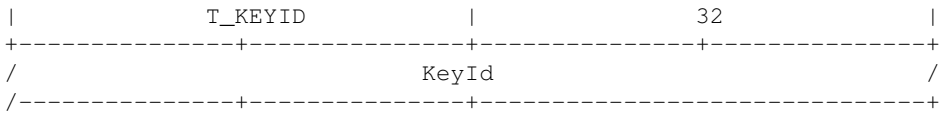
3.6.1.5. Validation Examples

As an example of a MIC type validation, the encoding for CRC32 validation would be:

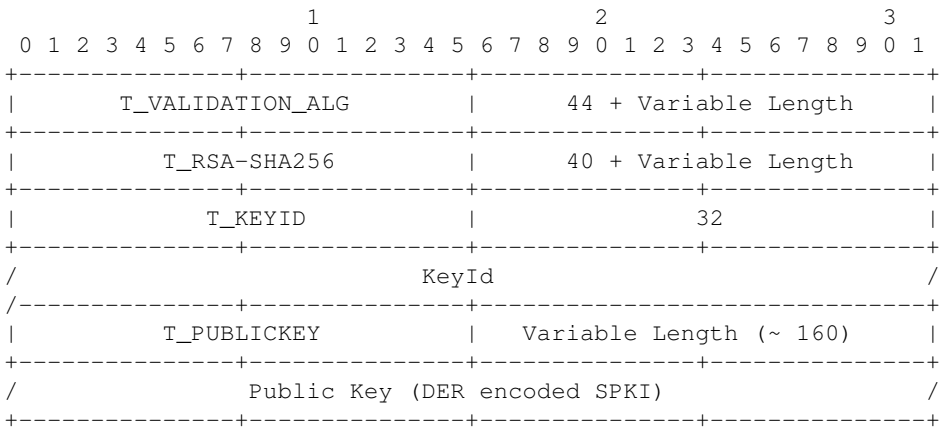


As an example of a MAC type validation, the encoding for an HMAC using a SHA256 hash would be:



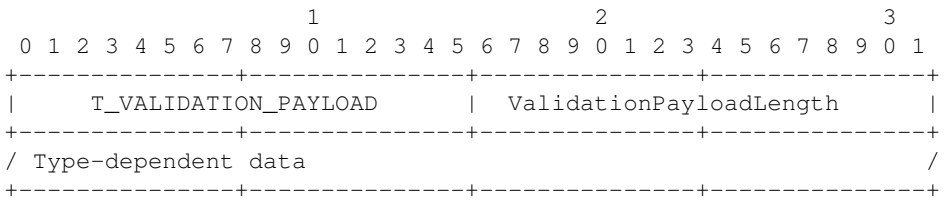


As an example of a Signature type validation, the encoding for an RSA public key signing using a SHA256 digest and Public Key would be:



TOC

3.6.2. Validation Payload



The ValidationPayload contains the validation output, such as the CRC32C code or the RSA signature.

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

[CCN] 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 December 3, 2014
Expires: June 6, 2015

Labeled Segment URIs

Abstract

This document defines an RFC3986 URI compliant identifier in which path segments carry a label. This allows differentiation between unrelated resources with similar identifiers. For example, one resource named "/parc/csl/7" might be referring to the 7th version of "/parc/csl", while another might refer to the 7th page of the resource. Using labeled segments, the two resources could have unambiguous names such as "/parc/csl/version=7" and "/parc/csl/page=7". A URI scheme that specifies the use of labeled segment URIs conforms to the encoding rules presented here.

Status of this Memo

Table of Contents

---

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 that allows an application or protocol to embed labels in path segments, thus disambiguating 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 path segments between different identifier spaces, such as "version" and "page". Other protocols may wish to use a type system such as "/str=parc/int=7" and "/str=parc/str=7". Labeled Segment URIs provide an unambiguous and flexible representation 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 set of common conventions. For example, if one application uses a base 64 encoding of a frame number, e.g. base64(0xbdea), and another uses "ver=" to represent a document version, there is an ambiguity because base64(0xbdea) is the string "ver=".

Labeled Segments defines "ls-segment" as "label[:param]=value", where the value only contains unreserved, percent-encoded, or certain sub-delim characters. In the previous example, one application 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, here 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 absolute 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, an 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 delimiter between the label:param and the value. Within the value,

an equal sign must be percent encoded.

lpv-segment labels and values may be alpha-numeric identifiers or numbers (decimal or hexadecimal). For example, one scheme may define the labels "name", "version", and "frame". A version may be of types "date" or "serial", meaning that the version is either a date 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 associated with "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 "." refers to 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 be used, 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 do the following:

- 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 components 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 December 3, 2014
Expires: June 6, 2015

Labeled Content Information

Abstract

This document describes Labeled Content Information - a labeled segment URI (LS-URI) representation of network data. This scheme, called "lci:", is applicable to network protocols such as Content Centric networks (CCN) 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

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, therefore a URI and CCNx Name are both composed of a URI path, which is a collection of path segments. We do not use the term "name component" as was common in old CCNx. In this document, the word "segment" alone means "path segment."

Labeled Content Information carries a label for each path segment. The contents of each path segment must conform to the label semantics. Example segment types are "Binary Segment", "Name", 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 that 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, KeyLocators, or any other place requiring a name. 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:

- the lci scheme.
- 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. The lci Scheme

This section describes the Labeled Content Information "lci:" scheme for Labeled Names. A Labeled Content Name assigns a semantic type or label to each segment of the hierarchical content Name.

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

Several path 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 CCNx Name types are:

- Name Segment: A generic path segment that includes arbitrary octets.
- Nonce Segment: A nonce in a path segment, often used by name-based protocols to create unique Interest requests.
- KeyId: A binary path segment representing the "fingerprint" of a key. An example might be the SHA-256 hash of the public key of the publisher.
- ContentObjectHash: A binary path segment specifiying the self-certifying name of a Content Object based on the entire object hash.
- 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 protocols, 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 an "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'	Hexadecimal	Name Segment
'Nonce'	Hexadecimal	Nonce segment
'Key Id'	Hexadecimal	Key Id segment
'ContentObjectHash'	Hexadecimal	ContentObjectHash 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/Version=0xA0/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 an 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

[CCN] 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	December 3, 2014
Expires: June 6, 2015	

CCNx End-To-End Fragmentation

Abstract

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

Status of this Memo

Table of Contents

TOC

1. Introduction

This document specifies an end-to-end fragmentation protocol for breaking a Content Object into several smaller pieces in order to fit within a network MTU. The fragmentation protocol does not provide a secure binding of the fragments to the original object. This 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.

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

The principle of operation for fragmentation in CCNx 1.0 is that intermediate systems should not have to fragment packets. An Interest message is always fragmented to the minimum MTU size and the forward path's MTU size is recorded in the Interest message so that a system sending back a Content Object can fragment it to the the correct size for the path. An intermediate system's Content Store may store only pre-fragmented objects and respond only if those fragments satisfy the requirements of an Interest's MTU, otherwise it will be considered a cache miss.

Because the Interest is the path discovery mechanism for Content Objects, all Interests MUST carry an InterestFragmentationHeader that includes information about the forward path's MTU so the reverse path MTU may be 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 than the path MTU discovered, but all fragments except the last MUST be the same size. This ensures that any two fragment streams of the same Content Object with the same MTU have the same representation.

When an end system creates a fragment stream, it generates a 64-bit number for the FragmentStreamID. This number identifies a contiguous stream of fragments and allows an end system to use the FragmentStreamID for reassembly. An intermediate system uses the FragmentStreamID 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 FragmentStreamID. This avoids easy denial-of-service attacks by replying with junk for known fragment stream IDs. A fragmented Content Object carries both its own FragmentStreamID, which SHOULD be based on the ContentObjectHash, and the corresponding Interest FragmentStreamID 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 of the 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 {FragmentID, FragNum}.

If all fragments are not received within a system-dependent timeout, a system re-assembling fragments should 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.

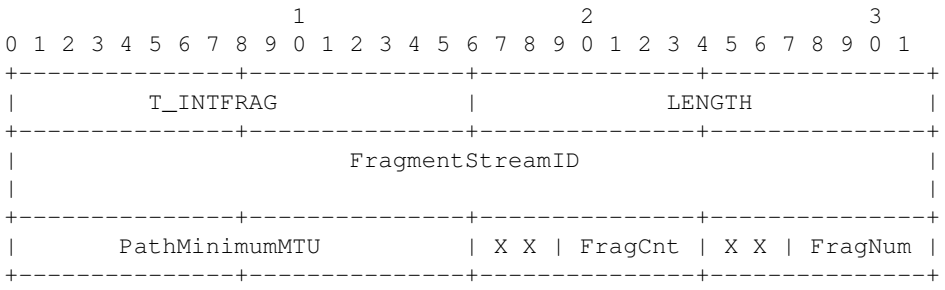
Both fragmented Interests and Content Objects use the fields FragCnt and FragNum. The count is the number of fragments in the message, which has a minimum of 1. FragNum is the 0-based fragment number. Sequential fragment numbers represent sequential byte boundaries of the original object.

TOC

3.1. Interest Header

The field FragmentStreamID identifies a contiguous stream of fragments. It SHOULD be a random number.

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



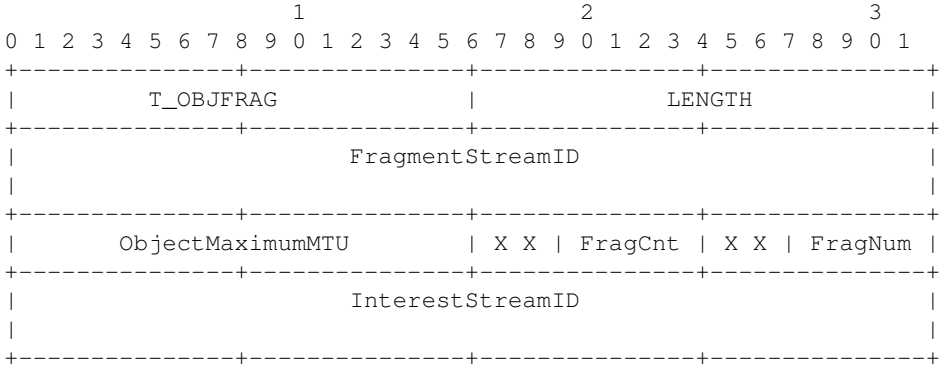
TOC

3.2. Content Object Header

The field FragmentStreamID 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 InterestStreamID 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 ObjectMaximumMTU 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.



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

[CCN] 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 December 3, 2014
Expires: June 6, 2015

CCNx Content Object Chunking

Abstract

This document specifies a chunking protocol for dividing a user payload into CCNx Content Objects. This includes specification for the naming convention to use for the chunked payload, the metadata convention used to store information about the chunked object, add the field added to a Content Object to represent the last chunk of an object.

Status of this Memo

Table of Contents

TOC

1. Introduction

CCNx Content Objects are sized to amortize cryptographic operations over user data while simultaneously staying a reasonable size for transport over today's networks. This means a Content Object is usually within common UDP or jumbo Ethernet size. If a publisher has a larger amount of data to associate with a single Name, the data should be chunked with this chunking protocol. This protocol leverages state in the Name and in an optional field within the Content Object. A chunked object may also have a metadata object that describes the original pre-chunked object.

CCNx uses two types of messages: Interests and Content Objects. An Interest carries the hierarchically structured variable-length identifier (HSVLI), or 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 Name, the object's Payload, and the cryptographic information used to bind the Name to the payload.

This specification adds a new segment to the Name TLV and a new field to the Metadata TLV. It updates [ietf-draft-ccnxmessages]. It also provides guidelines for the usage of the Key Locator in chunked 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. Chunking

Chunking, as used in this specification, means serializing user data into one or more chunks, each encapsulated in a CCNx Content Object. One segment in the Name of that Content Object represents the current chunk number. A field in the Content Object's Metadata - only mandatory in the final chunk - represents the end of the stream. Chunks are denoted by a serial counter, beginning at 0 and incrementing by 1 for each contiguous chunk. The chunking ends at the final chunk. No valid user data exists beyond the final chunk, and reading beyond the final chunk MUST NOT return any user data.

Chunking uses a block size for chunks. The block size may be inferred by the Content size of each Content Object. It is not explicit in the chunking name protocol. It may also be represented in a chunking Metadata object related to the Content Object. Using a consistent block size allows a reader to predict byte offsets.

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

The new Metadata field is the EndChunkNumber. It MUST be included in the Content Object which is the last chunk of an object, but SHOULD be present at the earliest time it is known. The value of the EndChunkNumber should be the network byte order value of the last ChunkNumber

The EndChunkNumber may be updated in later Chunks to a larger value, as long as it has not yet reached the end. The EndChunkNumber SHOULD NOT decrease. If a publisher wishes to close a stream before reaching the End Chunk, it should publish empty Content Objects to fill out to the maximum EndChunkNumber ever published. These padding chunks MUST contain the true EndChunkNumber.

TOC

2.1. Examples

Here are some examples of chunked 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 Chunks. Inside the Metadata, the EndChunkNumber is missing in the first (Chunk 0) object, but is known when Chunk 1 is published so is included in Chunk 1. It is omitted in Chunk 2, then appears in Chunk 3, where it is mandatory.

```
lci:/Name=parc/Name=csl/Name=picture.jpg/Chunk=0
  Protocol Info: No EndChunk
lci:/Name=parc/Name=csl/Name=picture.jpg/Chunk=1
  Protocol Info: EndChunk="3"
lci:/Name=parc/Name=csl/Name=picture.jpg/Chunk=2
  Protocol Info: No EndChunk
lci:/Name=parc/Name=csl/Name=picture.jpg/Chunk=3
  Protocol Info: EndChunk="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 ChunkNumber, stating the correct EndChunk. Chunks 4, 5, and 6 do not contain any new user data.

```
lci:/Name=parc/Name=csl/Name=talk.wav/Chunk=0
lci:/Name=parc/Name=csl/Name=talk.wav/Chunk=1
   Protocol Info: EndChunk="6"
lci:/Name=parc/Name=csl/Name=talk.wav/Chunk=2
lci:/Name=parc/Name=csl/Name=talk.wav/Chunk=3
   Protocol Info: EndChunk="3"
lci:/Name=parc/Name=csl/Name=talk.wav/Chunk=4
   Protocol Info: EndChunk="3"
lci:/Name=parc/Name=csl/Name=talk.wav/Chunk=5
   Protocol Info: EndChunk="3"
lci:/Name=parc/Name=csl/Name=talk.wav/Chunk=6
   Protocol Info: EndChunk="3"
```

TOC

3. Chunking Metadata

Chunking metadata MAY be saved along with a chunked object. If Chunking metadata is saved it SHOULD be saved using a Meta labeled path chunk with the value of chunking.

An example name using SerialNumber versioning is "lci:/Name=parc/Name=picture.jpg/SerialNumber=1/Meta=chunking/SerialNumber=0". This name means there is an object named "lci:/Name=parc/Name=picture.jpg/SerialNumber=1" that conforms to the Chunking protocol. The chunking metadata is stored in a separate Content Object with the example name.

The Chunking Metadata is a JSON Content Object whose content adheres to the following schema. The BlockSize is used for all Chunks except the last. The EndChunk is optional at the time the header is written, if it is not known. Once a chunking is finished a new header could be written with the correct EndChunk.

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.

```
{ "chunking" :
  { "BLOCKSIZE"  : <blocksize>
    [, "FILENAME"   : <original file name, if known>]
    [, "ENDChunk" : <end Chunk, if known>]
    [, "TOTALBYTES" : <total bytes of all Chunks, if known>]
    [, "DIGEST"   : { "ALGORITHM"=<OID>, "DIGEST"=<hexadecimal value> }]
  }
}
```

TOC

4. TLV Types

This section specifies the TLV types used by CCNx chunking.

TOC

4.1. Name Types

CCNx chunking uses one new Name type, for Chunk Number.

Type	Abbrev	Name	Description
%x0010	T_Chunk	Chunk Number (Chunk Number)	The current Chunk Number, is an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.
%x0011	T_Meta	Chunk Metadata (Chunk Number)	Identifies the Content Object on metadata for the prior Name.

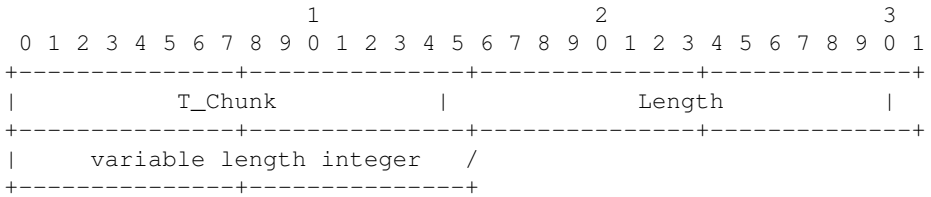
Table 1: Name Types

TOC

4.1.1. Chunk Number

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

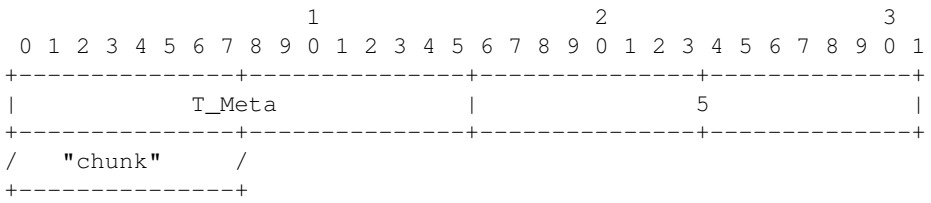
In lci: URI form, it is denoted as "Chunk".



TOC

4.1.2. Chunk Metadata Name Component

A name component that identifies the content object provides chunk metadata about the left-encapsulated name prefix, as described in Section 3 (Chunking Metadata).



TOC

4.2. Protocol Information

CCNx chunking uses one new field in the Metadata for the EndChunkNumber.

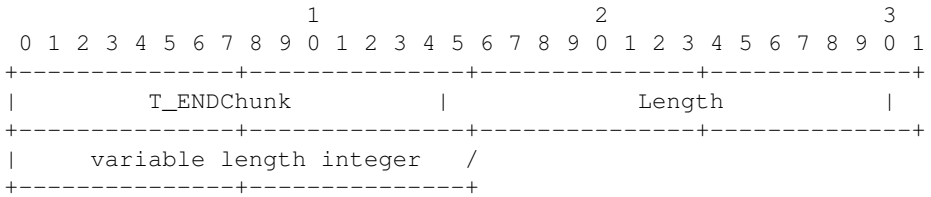
Type	Abbrev	Name	Description
%x0019	T_ENDChunk	EndChunkNumber (Chunk Number)	The last Chunk number, as an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

Table 2: Protocol Information Types

TOC

4.2.1. EndChunkNumber

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



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	December 3, 2014
Expires: June 6, 2015	

CCNx Publisher Clock Time Versioning

Abstract

This document specifies the use of a timestamp as a path segment in a CCNx Name as a versioning specifier. It defines the path segment label, encoding, and semantics.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document specifies the use of 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 publishers.

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 as an RFC3339 UTC string in the Interest date/time format, for example "1985-04-12T23:20:50.52Z". This format allows a direct strcmp() of two strings to determine their time ordering. Note that we allow fractions of a second.

An example Name using this format is "lci:/Name=parc/Name=presentation.pdf/Time=1985-04-12T23:20:50.52Z".

A publisher assigns a timestamp to indicate the time ordering of the prior Name path segments. It does not imply any specific temporal meaning such as the time of content creation or the time of Content Object signature. It is simply used to order a set of objects.

A "GONE" PayloadType 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
%x0012	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 December 3, 2014
Expires: June 6, 2015

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" PayloadType 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
%x0013	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: Informational December 3, 2014
Expires: June 6, 2015

CCNx Label Forwarding (CCNLF)

Abstract

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

Status of this Memo

Table of Contents

TOC

1. Introduction

CCNx Label Forwarding (CCNLF) supplants the normal forwarding rules where a potentially long hierarchically structured variable-length identifier is used for forwarding. In CCNLF, a label representing the routable part of the Name is pre-computed and added to the hop-by-hop headers in the packet. This, along with a another pre-computed label for Interests, allows forwarding nodes to switch Interests and Content Objects without needing to process the entire Name every hop.

For an Interest, a Interest Label is a strong hash of the entire Interest message from the CCNx MessageType through the end of the packet. The hash does not include any fixed or hop-by-hop headers.

If an end-system supports a default route, then an end system may leave the Forwarding Label empty (set to 0) in an Interest. At the first router, it will inspect the Name, compute the best routing prefix for the Interest, and update the Forwarding Label (FL) to that value. As the packet moves through the network the Interest is forwarded without inspecting the Name inside the Interest as long as the FL continues to match a FIB entry without children. If the FIB entry is not found or it has children, then that router must inspect the Name and determine the new FL.

As an Interest moves through the network, the FL may change over time. A Forwarder must record the previous hop's FL in the PIT so that changes may be reversed on the return path. This ensures that the IL and FL exactly equal what the previous hop expects.

When an Interest arrives at a node that may satisfy 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 IL/FL pair in an hop-by-hop header on the Content Object and returns it along the reverse route. The Content Object's IL/FL is matched against the PIT and then sent along the reverse paths. The Forwarder swaps the FL, as needed.

This document specifies:

- The use of a new hop-by-hop header on Interest and Content Object packets for forwarding.
- The Interest Label of an Interest.
- The Forwarding Label of a CCNx Name.
- The forwarding and reverse path forwarding algorithm using the IL/FL.
- The packet formats of CCNx TLV encoded messages.

This document does not cover fragmentation

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. Determining Labels

Label Forwarding relies on each node computing the same forwarding label to encode Name prefixes. One option is for the routing protocol to carry both the FL and the associated Name prefix. In this case, there is no calculation. The routing protocol specifies both the FL and Name. Another option is to use a well-known function to compute the FL from the Name. In this document, we assume that the FL is computed from the Name, so a common hash function is specified

The Interest Label 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 Interest Label uses SHA-256 and is computed from MessageType through the end of the packet.

The Forwarding Label 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 Label 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 a Forwarding Label 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 CCNLF header includes two fields: the Forwarding Label (FL) and the Interest Label (IL). The FL and IL are as described previously.

TOC

3.1. TLV Format

CCNLF uses an hop-by-hop header to encapsulate the IL and FL. The header is 36 bytes long, containing two 16-byte hashes and a 4 byte TLV header.

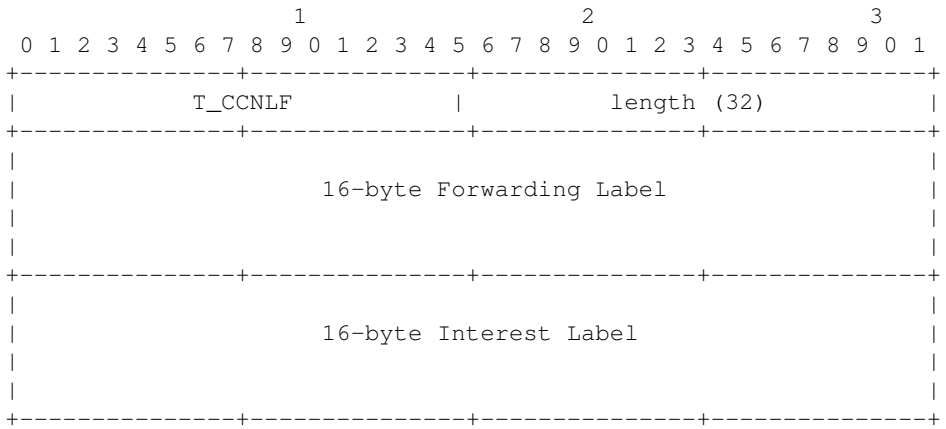


Figure 2: CCNLF hop-by-hop header

TOC

4. Protocol Operation

The CCNLF protocol operates on principle similar to the normal CCNx protocol: 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 exactly equal to the Interest Name, and it must satisfy the various restrictions in the Interest if present. CCNLF speeds up normal CCNx processing by pre-computing the Interest Label and longest-matching-prefix (LMP) Forwarding Label. The assumption is that the LMP Forwarding Label does not change frequently in-route, so few, if any, intermediate systems need to do expensive CCNx Name parsing and hashing.

The key difference between CCNLF and CCNx is that a CCNLF Forwarder does not evaluate the Name when matching content in the Content Store. It uses an exact match on the Interest Label. Additionally, the entries in the Forwarding Information Base (FIB) include Forwarding Labels.

The key difference between CCNLF and CCNx, and in particular the Forwarder behavior, is that a CCNLF Forwarder does not evaluate the Name when matching content in the Content Store. It uses an exact match on the Interest Label and Forwarding Label.

A CCNLF 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 Interest and Forwarding Labels), so one Content Object may be forked to multiple reverse paths.

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

In general, we must match both the IL and FL of an Interest on the return path of a Content Object. This is because a malicious user could put in an IL for /popular/content and an FL for /colluding/site. The Content Object from /colluding/site would have malicious content, but an IL for /popular/content. If Forwarders do not validate that the Content Object matches the full pending Interest and only reverse path forward with the IL, the malicious content pollutes the network. However, if we match on both the IL and FL, then a benign user who asked for /popular/content in the IL and set the IL 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 FL to the more specific header. When a Content Object is returned, an intermediate node will re-swap the FL label. When an intermediate node receives a Content Object, it verifies that it came from the expected direction, based on the PIT entry and IL/FL headers. An exception to this is if an Interest was routed along the default route (an empty FL), in which case the FL header in the Content Object is not swapped, enabling the previous hop to learn the correct FL.

A PIT entry must store the IL (which is invariant through forwarding), the ingress FL, and the egress FL. When a Content Object is received, the Content Object's FL must match the IL and egress FL in the PIT entry. When the Content Object is reverse path forwarded its FL is label swapped with the ingress FL stored in the PIT. It is possible that the PIT must store multiple ingress FL's based on how ingress FL's are rewritten when forwarding.

TOC

4.1. Interest Forwarding

TOC

4.1.1. Initiate Interest

A first node creates a CCNx Interest as normal. It computes the InterestLabel as specified in Section 2 (Determining Labels), putting it in the IL field in the hop-by-hop CCNLF header. If the node knows the proper Forwarding Header, it places that in the FL field.

It then sends the CCNLF packet to the next hop.

TOC

4.1.2. Receive Interest

When a Forwarder receives a CCNLF Interest on an ingress interface, it performs the following actions:

- Drop an Interest if it has a HopLimit of 0.
- Decrement the HopLimit field of the Interest.
- Look up the Interest's IL/FL in the PIT. If no entry exists, create a PIT entry by recording the IL and FL of the Interest and noting the ingress port.
- If the Forwarder implements a Content Store, check the Content Store. (Check Content Store)
- Forward the Interest. (Forward Interest).

TOC

4.1.3. Check Content Store

Lookup the FL in the FIB and determine the longest matching FL (LMFL). If there is no longer matching FL, set LMFL = FL.

Match the IL and LMFL in the Content Store. If there is an exact match, return the Content Object and consume the PIT entry. The returned object carries IL/FL, unless FL was the default route, in which case it carries IL/LMFL.

TOC

4.1.4. Forward Interest

The Forwarder follows these steps to forward the Interest:

- Find the longest matching prefix to the FL in the FIB based on the Name of the Interest and forward out those ports. An Interest should never be forwarded out its ingress port. Call the longest-matching FIB Forwarding Label LMFL and the set of egress interfaces E. As an example, if the FIB is a hash table, look up the FL 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.
- Look up the IL/LMFL in the PIT. If the Interest's HopLimit (as decremented above) is greater than the PIT entries "maximum HopLimit", then set the PIT entry's maximum HopLimit to the Interest's HopLimit. Internally mark the Interest as "HopLimit extended." If the entry is not "HopLimit extended", remove any egress interfaces already used from E.
- Link IL/LMFL to IL/FL, if they are different. This may be a one to many relationship.
- If E is not empty, update the FL in the Interest with the longest matching FIB hash, then forward.

TOC

4.2. Content Object Forwarding

TOC

4.2.1. 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 IL/FL carried in the CCNLF hop-by-hop header of the Interest.

TOC

4.2.2. Forward Content Object

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

- Verify the IL/FL is in the PIT. If not, drop the Content Object.
- Verify that the object arrived from a port over which it was forwarded. If it does not verify, drop the Content Object.
- Verify that the KeyId and ContentObjectHash of the Content Object match the KeyIdRestriction and ContentObjectHashRestriction of the Interest, if present.
- Forward the object along the reverse path, label swapping the object's FL to the reverse path's FL, unless the reverse path FL was the default route (empty) in which case the FL remains unchanged. This is done by following the links from IL/LMFL to IL/FL, if any exist.

- Consume the PIT entries satisfied by the object.

TOC

4.2.3. 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

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

- Forwarders do not evaluate the contents of an Interest, but rely only on the user-provided Interest Label. A use could place a forged IL on an Interest and retrieve content not requested in the actual Interest.

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

[CCN] 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

TOC

Internet Engineering Task Force M. Mosko, Ed.

Internet-Draft PARC

Intended status: Experimental December 3, 2014

Expires: June 6, 2015

Interest Return Network Control Packet

Abstract

This document describes the process whereby a network element that cannot satisfy an Interest may return the Interest to a previous hop for further processing or to inform the Interest origin of the error condition.

Status of this Memo

Table of Contents

TOC

1. Introduction

This document describes the process whereby a network element may return an Interest message to a previous hop if there is an error processing the Interest. The returned Interest may be further processed at the previous hop or returned towards the Interest origin.

This document specifies:

- The TLV encoding and packet format.
- The behavior of nodes creating and receiving Interest Return messages.

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

- The PacketType is set to 2 to indicate a Feedback message.
- The FeebackCode is set to the appropriate value to signal the reason for the return

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

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

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

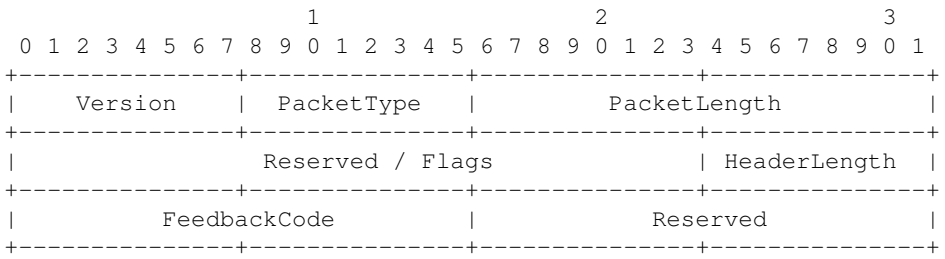
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.) [RFC2119].

TOC

2. Message Format

The Interest Return message looks exactly like the original Iterest message with the exception of the two

modifications mentioned above.



- Version: defines the version of the packet - the same as the original Interest.
- PacketType: 2
- PacketLength: the PacketLength field from the original Interest.
- Reserved / Flags: the same as from the original Interest.
- HeaderLength: the HeaderLength field from the original Interest.
- FeedbackCode: Set to a value that indictes the reason for the return.

The FeedbackCode is divided into two segments:

- Byte 1: Indicator for the type of message being returned. For Interest returns, this is set to 1.
- Byte 2: Reason code (see list of Reason Codes below)

TOC

3. Interest Return Reason Codes

This section defines the Interest Return reason codes introduced in this RFC.

Reason	Name	Description
0	No route to name (No route to name)	The returning Forwarder has no route to the Interest name
1	PIT time exceeded (PIT Time Exceeded)	The returning Forwarder's PIT timed, no answer will come
2	Interest MTU too large (Interest MTU Too Large)	The Interest's MTU does not conform to the required minimum and would require fragmentation.
3	Interest Rejected (Interest Rejected)	The returning Forwarder rejects the Interest for unspecified reasons. These may include internal capacity limits, policy settings, or any other cause not covered by other Reason codes.

Table 1: Interest Return Reason Codes

TOC

4. Protocol

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

TOC

4.1. No route to name

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

If a Forwarder receives a "no route" Interest Return, it SHOULD:

- Look up the Interest in the PIT.
- If it does not exist, ignore the Interest Return
- If it does exist, it MAY do one of the following:
    - Try a different forwarding path, if one exists, and discard the Interest Return, or
    - Consume the PIT state and forward the Interest Return along the reverse path.

TOC

4.2. PIT Time Exceeded

A Forwarder MAY choose to send PIT Time Exceeded messages when a PIT lifetime expires without receiving a matching Content Object. These messages should be used for network testing, and are not recommended for normal use.

If a Forwarder receives a "PIT Time Exceeded" Interest Return, it SHOULD take the same actions as a "No Route" message.

TOC

4.3. Interest MTU Too Large

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

If a Forwarder receives an "Interest MTU Too Large" is SHOULD clear the PIT state and send the Interest Return to the previous hop, updating the Interest Return Authenticator as appropriate.

TOC

4.4. Interest Rejected

If a Forwarder receives an Interest, it MAY reject it for any reason. It SHOULD send an "Interest Rejected" message to the previous hop, though it may rate limit them. An Interest could be rejected for many reasons, such as an administrative policy, too many Interests already in the PIT, too many Interests from the reverse path peer, etc.

If a Forwarder receives an "Interest Rejected" message, it SHOULD take the same actions as for a "No Route" message.

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

Interest return messages without an authenticator should only be used in restricted situations. Cases where an authenticator may be omitted would be in closed, directly connected systems. Internet systems, systems running over IP tunnels, or systems on multiple access media should always use an Interest Return Authenticator.

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