ICNRG
Internet-Draft
Intended status: Experimental
Expires: December 31, 2015

M. Mosko
I. Solis
PARC, Inc.

June 29, 2015

CCNx Messages in TLV Format
draft-irtf-icnrg-ccnxmessages-00

Abstract

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

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document specifies a Type-Length-Value (TLV) packet format and the TLV type and value encodings for the CCNx network protocol as specified in [CCNSemantics]. 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.

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

This document specifies:

- o The TLV packet format.
- o The overall packet format for CCNx messages.
- o The TLV types used by CCNx messages.
- o The encoding of values for each type.
- o Top level types that exist at the outermost containment.
- o Interest TLVs that exist within Interest containment.
- o Content Object TLVs that exist within Content Object containment.

This document is supplemented by this document:

- o Message semantics: see [CCNSemantics] for the protocol operation regarding Interest and Content Object, including the Interest Return protocol.

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.

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.

TODO -- we have not adopted the Requirements Language yet.

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 [RFC2119].

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.

3. Type-Length-Value (TLV) Packets

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. Specifically, the TLV types in the range 0x1000 - 0x1FFF are reserved for experimental use. These type values are reserved in all TLV container contexts.In the event that more space is needed, either for types or for length, a new version of the protocol would be needed.

```
                     1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|              Type             |            Length             |
+---------------+---------------+---------------+---------------+
```

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 additional TLV structures. The enclosing TLV structure is called the container of the enclosed TLV.

Type values are context-dependent. Within a TLV container, one may re-use previous type values for new context-dependent purposes.

3.1. Overall packet format

Each packet includes the 8 byte fixed header described below, followed by a set of TLV fields. These fields are optional hop-by-hop headers and the Packet Payload.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Version    |  PacketType   |         PacketLength          |
+---------------+---------------+---------------+---------------+
|           PacketType specific fields          | HeaderLength  |
+---------------+---------------+---------------+---------------+
/ Optional Hop-by-hop header TLVs                               /
+---------------+---------------+---------------+---------------+
/ PacketPayload TLVs                                            /
+---------------+---------------+---------------+---------------+
```

The packet payload is a TLV encoding of the CCNx message, followed by

optional Validation TLVs.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
| CCNx Message TLV                                              /
+---------------+---------------+---------------+---------------+
/ Optional CCNx ValidationAlgorithm TLV                         /
+---------------+---------------+---------------+---------------+
/ Optional CCNx ValidationPayload TLV (ValidationAlg required)  /
+---------------+---------------+---------------+---------------+
```

This document describes the Version "1" TLV encoding.

After discarding the fixed and hop-by-hop headers the remaining PacketPayload should be a valid protocol message. Therefore, the PacketPayload always begins with a 4 byte TLV defining the protocol message (whether it is an Interest, Content Object, or other 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 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.

3.2. Fixed Headers

CCNx messages begin with an 8 byte fixed header (non-TLV format). The HeaderLength field represents the combined length of the Fixed and Hop-by-hop headers. The PacketLength field represents the entire Packet length.

A specific PacketType may assign meaning to the reserved bytes.

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

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+---------------+
   |    Version    |  PacketType   |         PacketLength          |
   +---------------+---------------+---------------+---------------+
   |           PacketType specific fields          | HeaderLength  |
   +---------------+---------------+---------------+---------------+
```

- o Version: defines the version of the packet.

- o HeaderLength: The length of the fixed header (8 bytes) and hop-by-hop headers. The minimum value is "8".

- o PacketType: describes forwarder actions to take on the packet.

- o PacketLength: Total octets of packet including all headers (fixed header plus hop-by-hop headers) and protocol message.

- o PacketType Specific Fields: specific PacketTypes define the use of these bits.

The PacketType field indicates how the forwarder should process the packet. A Request Packet (Interest) has PacketType 0, a Response (Content Object) has PacketType 1, and an InterestReturn Packet has PacketType 2.

HeaderLength is the number of octets from the start of the packet (Version) to the end of the hop-by-hop headers. PacketLength is the number of octets from the start of the packet to the end of the packet.

The PacketType specific fields are reserved bits whose use depends on the PacketType. They are used for network-level signaling.

3.2.1. Interest Fixed Header

If the PacketType in the Fixed Header is "0", it indicates that the PacketPayload should be processed as an Interest message. For this type of packet, the Fixed Header includes a field for a HopLimit as well as Reserved and Flags fields. The Reserved field must be set to 0 in an Interest - this field will be set to a return code in the case of an Interest Return. There are currently no Flags defined, so this field must also be set to 0.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Version    |       0       |         PacketLength          |
+---------------+---------------+---------------+---------------+
|   HopLimit    |   Reserved    |     Flags     | HeaderLength  |
+---------------+---------------+---------------+---------------+
```

3.2.1.1. Interest HopLimit

For an Interest message, 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. 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.

3.2.2. Content Object Fixed Header

If the PacketType in the Fixed Header is "1", it indicates that the PacketPayload should be processed as a Content Object message. A Content Object defines a Flags field, however there are currently no flags defined, so the Flags field must be set to 0.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Version    |       1       |         PacketLength          |
+---------------+---------------+---------------+---------------+
|           Reserved            |     Flags     | HeaderLength  |
+---------------+---------------+---------------+---------------+
```

3.2.3. InterestReturn Fixed Header

If the PacketType in the Fixed Header is "2", it indicates that the PacketPayload should be processed as a returned Interest message. The only difference between this InterestReturn message and the original Interest is that the PacketType is changed to "2" and a ReturnCode is is put into the Reserved octet. All other fields are unchanged. The purpose of this encoding is to prevent packet length changes so no additional bytes are needed to return an Interest to the previous hop. See [CCNSemantics] for a protocol description of this packet type.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Version    |       2       |         PacketLength          |
+---------------+---------------+---------------+---------------+
|   HopLimit    |  ReturnCode   |     Flags     | HeaderLength  |
+---------------+---------------+---------------+---------------+
```

3.2.3.1. InterestReturn HopLimit

This is the original Interest's HopLimit, as received. It is the value before being decremented at the current node.

3.2.3.2. InterestReturn Flags

These are the original Flags as set in the Interest.

3.2.3.3. Return Code

The numeric value assigned to the return types is defined below. This value is set by the node creating the Interest Return.

A return code of "0" is not allowed, as it indicates that the returning system did not modify the Return Code field.

Value	Return Type
1	No Route
2	Hop Limit Exceeded
3	No Resources
4	Path Error
5	Prohibited
6	Congested
7	MTU too large

Table 1: Return Codes

3.3. Hop-by-hop TLV headers

Hop-by-hop TLV headers are unordered and no meaning should be attached to their ordering. Four hop-by-hop headers are described in this document:

Type	Abbrev	Name	Description
%x0001	T_INTLIFE	Interest Lifetime (Section 3.3.1)	The time an Interest should stay pending at an intermediate node.
%x0002	T_CACHETIME	Recommended Cache Time (Section 3.3.2)	The Recommended Cache Time for Content Objects.

Table 2: Hop-by-hop Header Types

Additional hop-by-hop headers are defined in higher level specifications such as the fragmentation specification.

3.3.1. Interest 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. It should still be forwarded, but no reply is expected.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_INTLIFE            |             Length            |
+---------------+---------------+---------------+---------------+
/                                                               /
/                      Lifetime (length octets)                 /
/                                                               /
+---------------+---------------+---------------+---------------+
```

3.3.2. 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 3.6.2.2.2)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.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_CACHETIME          |               8               |
+---------------+---------------+---------------+---------------+
/                                                               /
/                    Recommended Cache Time                     /
/                                                               /
+---------------+---------------+---------------+---------------+
```

3.4. 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 (Section 3.6)	An Interest MessageType.
%x0002	T_OBJECT	Content Object (Section 3.6)	A Content Object MessageType

%x0003	T_VALIDATION_ALG	Validation Algorithm (Section 3.6.4.1)	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 (Section 3.6.4.2)	The validation output, such as the CRC32C code or the RSA signature.

Table 3: CCNx Top Level Types

3.5. Global Formats

3.5.1. Pad

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

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

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

3.5.2. Organization Specific TLVs

Organizations may request proprietary TLV types in the Hop-By-Hop headers section or other TLV containers. The organization then has control of the contents of the Value, which may be its own binary

field or an encapsulated set of TLVs. The inner TLVs, because we use a context-dependent TLV scheme, may be fully defined by the organization.

Organization specific TLVs MUST use the T_ORG type. The Length field is the length of the organization specific information plus 3. The Value begins with the 3 byte organization number derived from the last three digits of the IANA Private Enterprise Numbers([CCNSemantics]), followed by the organization specific information.

Type	Abbrev	Name	Description
%x0FFF	T_ORG	Vendor Specific Information	Information specific to a vendor implementation.

Table 4: Additional CCNx Message Types

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|             (T_ORG)           |     Length (3+value length)   |
+---------------+---------------+---------------+---------------+
|   PEN[0]      |    PEN[1]     |     PEN[2]    |               /
+---------------+---------------+---------------+               +
/                  Vendor Specific Value                        /
+---------------+---------------+---------------+---------------+
```

3.5.3. Link

A Link is the tuple: {CCNx Name, KeyId, ContentObjectHash}. It is a general encoding that is used in both the payload of a Content Object with PayloadType = "Link" and in the KeyName field in a KeyLocator.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
/ Mandatory CCNx Name                                           /
+---------------+---------------+-------------------------------+
/ Optional KeyId                                                /
+---------------------------------------------------------------+
/ Optional ContentObjectHash                                    /
+---------------------------------------------------------------+
```

3.6. 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. 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 Message TLVs and an optional Payload TLV.

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|         MessageType           |         MessageLength         |
+---------------+---------------+---------------+---------------+
| Name TLV       (Type = T_NAME)                                |
+---------------+---------------+---------------+---------------+
/ Optional Message TLVs   (Various Types)                       /
+---------------+---------------+---------------+---------------+
/ Optional Payload TLV  (Type = T_PAYLOAD)                      /
+---------------+---------------+---------------+---------------+
```

Type	Abbrev	Name	Description
%x0000	T_NAME	Name (Section 3.6.1)	The CCNx Name requested in an Interest or published in a Content Object.
%x0001	T_PAYLOAD	Payload (Section 3.6.3)	The message payload.

Table 5: CCNx Message Types

3.6.1. 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.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            T_NAME             |            Length             |
+---------------+---------------+---------------+---------------+
/ Name segment TLVs                                             /
+---------------+---------------+---------------+---------------+
```

Type	Symbolic Name	Name	Description
%x0001	T_NAMESEGMENT	Name segment (Section 3.6.1.1)	A generic name Segment.
%x0002	T_IPID	Interest Payload ID (Section 3.6.1.2)	An identifier that represents the Interest Payload field. As an example, the Payload ID might be a hash of the Interest Payload. This provides a way to differentiate between Interests based on their payloads without having to parse all the bytes of the payload itself; instead using only this Payload ID Name segment
%x1000 - %x1FFF	T_APP:00 - T_APP:4096	Application Components (Section 3.6.1.1)	Application-specific payload in a name segment. An application may apply its own semantics to the 4096 reserved types.

Table 6: CCNx Name Types

3.6.1.1. 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/yo" would be encoded as:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|            (T_NAME)           |           %x14 (20)           |
+---------------+---------------+---------------+---------------+
|        (T_NAME_SEGMENT)       |           %x03 (3)            |
+---------------+---------------+---------------+---------------+
|       f                o               o       |(T_NAME_SEGMENT)
+---------------+---------------+---------------+---------------+
|               |            %x03 (3)           |       b       |
+---------------+---------------+---------------+---------------+
|      a                r       |           (T_NAME_SEGMENT)   |
+---------------+---------------+---------------+---------------+
|           %x02 (2)            |       y       |       o       |
+---------------+---------------+---------------+---------------+
```

3.6.1.2. Interest Payload ID

The InterestPayloadID is a name segment created by the origin of an Interest to represent the Interest Payload. This allows the proper multiplexing of Interests based on their name if they have different payloads. A common representation is to use a hash of the Interest Payload as the InterestPayloadID.

As part of the TLV 'value', the InterestPayloadID contains a one identifier of method used to create the InterestPayloadID followed by a variable length octet string. An implementation is not required to implement any of the methods to receive an Interest; the InterestPayloadID may be treated only as an opaque octet string for purposes of multiplexing Interests with different payloads. Only a device creating an InterestPayloadID name segment or a device verifying such a segment need to implement the algorithms. Because we allow application-specific algorithms and nonces, a device may not be able to verify the name segment. We use the same encoding as RFC 6920 [RFC6920] Binary Format. If the InterestPayloadID is created via a hash, it is encoded exactly as in RFC 6920 Section 6 Binary Format. If the ID is created via application specific means, then we set the high-order Reserved bit (0x80) and use the following table for methods, which are not part of the RFC6920 suite.

0: Application Specific (0x80)

1: Nonce (0x81)

In normal operations, we recommend displaying the InterestPayloadID as an opaque octet string in an LCI scheme, as this is the common denominator for implementation parsing. The InterestPayloadID name segment may be displayed using the RFC6920 format NI scheme, for example as "lci:/name=foo/name=bar/ipid=sha-256-32;f4OxZQ".

The InterestPayloadID, even if it is a hash, should not convey any security context. If a system requires confirmation that a specific entity created the InterestPayload, it should use a cryptographic signature on the Interest via the ValidationAlgorithm and ValidationPayload or use its own methods inside the Interest Payload.

3.6.2. Message TLVs

Each message type (Interest or Content Object) is associated with a set of optional Message TLVs. Additional specification documents may extend the types associated with each.

3.6.2.1. Interest Message TLVs

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

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          MessageType          |         MessageLength         |
+---------------+---------------+---------------+---------------+
| Name TLV                                                      |
+---------------+---------------+---------------+---------------+
/ Optional KeyIdRestriction TLV                                 /
+---------------------------------------------------------------+
/ Optional ContentObjectHashRestriction TLV                     /
+---------------------------------------------------------------+
```

Type	Abbrev	Name	Description
%x0002	T_KEYIDRESTR	KeyIdRestriction (Section 3.6.2.1.1)	An octet string identifying the specific publisher signing key that would satisfy the Interest.
%x0003	T_OBJHASHRESTR	ContentObjectHashRestriction (Section 3.6.2.1.2)	The SHA-256 hash of the specific Content Object that would satisfy the Interest.

Table 7: CCNx Interest Message TLV Types

3.6.2.1.1. KeyIdRestriction

An Interest may include a KeyIdRestriction selector. This ensures that only Content Objects with matching KeyIds will satisfy the Interest. See Section 3.6.4.1.4.1 for the format of a KeyId.

3.6.2.1.2. 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.

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

3.6.2.2. Content Object Message TLVs

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

```
                         1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+---------------+
   |          MessageType          |         MessageLength         |
   +---------------+---------------+---------------+---------------+
   | Name TLV                                                      |
   +---------------+---------------+---------------+---------------+
   / Optional PayloadType TLV                                      /
   +---------------------------------------------------------------+
   / Optional ExpiryTime TLV                                       /
   +---------------------------------------------------------------+
```

Type	Abbrev	Name	Description
%x0005	T_PAYLDTYPE	PayloadType (Section 3.6.2.2.1)	Indicates the type of Payload contents.
%x0006	T_EXPIRY	ExpiryTime (Section 3.6.2.2.2)	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 8: CCNx Content Object Message TLV Types

3.6.2.2.1. PayloadType

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

- 0: Data (possibly encrypted)
- 1: Key
- 2: Link
- 3: Manifest

The Data type 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 Link type indicates that the Payload is a Link (Section 3.5.3). If this field is missing, a "Data" type is assumed. A Manifest type indicates that the Payload is a Manifest (format TBD).

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|             T_PAYLDTYPE       |             Length            |
+---------------+---------------+---------------+---------------+
|  PayloadType  /
+---------------+
```

3.6.2.2.2. ExpiryTime

The ExpiryTime is the time at which the Payload expires, as expressed by a timestamp 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.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_EXPIRY            |               8               |
+---------------+---------------+---------------+---------------+
/                          ExpiryTime                           /
/                                                               /
+---------------+---------------+---------------+---------------+
```

3.6.3. Payload

The Payload TLV contains the content of the packet. It is permissible to have a "0" length. If a packet does not have any payload, this field may be omitted, rather than carrying a "0" length.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|           T_PAYLOAD           |             Length            |
+---------------+---------------+---------------+---------------+
/                        Payload Contents                       /
+---------------+---------------+---------------+---------------+
```

3.6.4. 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 the ValidationPayload 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.

3.6.4.1. 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 3.6.4.1.5

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|        T_VALIDATION_ALG       |      ValidationAlgLength      |
+---------------+---------------+---------------+---------------+
|         ValidationType        |             Length            |
+---------------+---------------+---------------+---------------+
/ ValidationType dependent data                                 /
+---------------+---------------+---------------+---------------+
```

Type	Abbrev	Name	Description
%x0002	T_CRC32C	CRC32C (Section 3.6.4.1.1)	Castagnoli CRC32 (iSCSI, ext4, etc.), with normal form polynomial 0x1EDC6F41.
%x0004	T_HMAC-SHA256	HMAC-SHA256 (Section 3.6.4.1.2)	HMAC (RFC 2104) using SHA256 hash.
%x0005	T_VMAC-128	VMAC-128 (Section 3.6.4.1.2)	VMAC with 128bit tags [VMAC]
%x0006	T_RSA-SHA256	RSA-SHA256 (Section 3.6.4.1.3)	RSA public key signature using SHA256 digest.
%x0007	EC-SECP-256K1	SECP-256K1 (Section 3.6.4.1.3)	Elliptic Curve signature with SECP-256K1 parameters (see [ECC]).
%x0008	EC-SECP-384R1	SECP-384R1 (Section 3.6.4.1.3)	Elliptic Curve signature with SECP-384R1 parameters (see [ECC]).

Table 9: CCNx Validation Types

3.6.4.1.1. Message Integrity Checks

MICs do not require additional data in order to perform the verification. An example is CRC32C that has a "0" length value.

3.6.4.1.2. Message Authentication Checks

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

3.6.4.1.3. Signature

Signature type Validators specify a digest mechanism and a signing algorithm to verify the message. Examples include RSA signature og a SHA256 digest, an Elliptic Curve signature with SECP-256K1 parameters, etc. These Validators require a KeyId and a mechanism for locating the publishers public key (a KeyLocator) - optionally a PublicKey or Certificate or KeyName.

3.6.4.1.4. 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 (Section 3.6.4.1.4.1)	An identifier of the shared secret or public key associated with a MAC or Signature. Typically the SHA256 hash of the key.
%x000B	T_PUBLICKEY	Public Key (Section 3.6.4.1.4.2)	DER encoded public key.
%x000C	T_CERT	Certificate (Section 3.6.4.1.4.3)	DER encoded X509 certificate.

%x000E	T_KEYNAME	KeyName (Section 3.6.4.1.4.4)	A CCNx Link object.
%x000F	T_SIGTIME	SignatureTime (Section 3.6.4.1.4.5)	A millsecond timestamp indicating the time when the signature was created.

Table 10: CCNx Validation Dependent Data Types

3.6.4.1.4.1. KeyId

The KeyId is the publisher key identifier. It is similar to a Subject Key Identifier from X509 [RFC 5280, 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.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|             T_KEYID           |             Length            |
+---------------+---------------+---------------+---------------+
/                             KeyId                             /
/---------------+---------------+-------------------------------+
```

3.6.4.1.4.2. Public Key

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

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

3.6.4.1.4.3. Certificate

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

3.6.4.1.4.4. 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.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|          T_KEYNAME            |             Length            |
+---------------+---------------+-------------------------------+
/ Link                                                          /
+---------------------------------------------------------------+
```

3.6.4.1.4.5. SignatureTime

The SignatureTime is a millisecond timestamp 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 Message, 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.

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+-------------------------------+
   |           T_SIGTIME           |               8               |
   +---------------+---------------+-------------------------------+
   /                         SignatureTime                         /
   +---------------------------------------------------------------+
```

3.6.4.1.5. Validation Examples

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

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+---------------+
   |      T_VALIDATION_ALG         |               4               |
   +---------------+---------------+---------------+---------------+
   |            T_CRC32            |               0               |
   +---------------+---------------+---------------+---------------+
```

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

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+---------------+
   |       T_VALIDATION_ALG        |               40              |
   +---------------+---------------+---------------+---------------+
   |         T_HMAC-SHA256         |               36              |
   +---------------+---------------+---------------+---------------+
   |             T_KEYID           |               32              |
   +---------------+---------------+---------------+---------------+
   /                              KeyId                            /
   /---------------+---------------+-------------------------------+
```

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:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|       T_VALIDATION_ALG        |      44 + Variable Length     |
+---------------+---------------+---------------+---------------+
|          T_RSA-SHA256         |      40 + Variable Length     |
+---------------+---------------+---------------+---------------+
|             T_KEYID           |               32              |
+---------------+---------------+---------------+---------------+
/                            KeyId                              /
/---------------+---------------+-------------------------------+
|          T_PUBLICKEY          |   Variable Length (~ 160)     |
+---------------+---------------+---------------+---------------+
/               Public Key (DER encoded SPKI)                   /
+---------------+---------------+---------------+---------------+
```

3.6.4.2. Validation Payload

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|     T_VALIDATION_PAYLOAD      |  ValidationPayloadLength      |
+---------------+---------------+---------------+---------------+
/ Type-dependent data                                           /
+---------------+---------------+---------------+---------------+
```

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

4. Acknowledgements

5. IANA Considerations

TODO: Work with IANA to define the type space for: Top level types, Hop-by-hop header types, Name segment types, CCNx messages types, Interest message TLV types, Content Object TLV message types, Validation types, and Validation dependent data types.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs [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.

6. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

7. References

7.1. Normative References

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

7.2. Informative References

[CCN] PARC, Inc., "CCNx Open Source", 2007, <http://www.CCNx.org>.

[CCNSemantics] Mosko, M., Solis, I., and M. Stapp, "CCNx Semantics (Internet draft)", 2015, <http://tools.ietf.org/html/draft-mosko-icnrg-ccnxsemantics-00>.

[ECC] Certicom Research, "SEC 2: Recommended Elliptic Curve Domain Parameters", 2010, <http://www.secg.org/sec2-v2.pdf>.

[EpriseNumbers] IANA, "IANA Private Enterprise Numbers", 2015, <http://www.iana.org/assignments/enterprise-numbers/enterprise-numbers>.

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

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, April 2013.

[VMAC] Krovertz, T. and W. Dai, "VMAC: Message Authentication Code using Universal Hashing", 2007, <http://www.fastcrypto.org/vmac/draft-krovetz-vmac-01.txt>.

Authors' Addresses

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
USA

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

Ignacio Solis
PARC, Inc.
Palo Alto, California 94304
USA

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