ICNRG
Internet-Draft
Updates: 8569, 8609 (if approved)
Intended status: Experimental
Expires: 25 January 2026

M.E. Mosko

H. Asaeda
NICT

24 July 2025

CCNx Content Object Chunking
draft-irtf-icnrg-ccnxchunking-02

Abstract

This document specifies a chunking protocol for dividing a user payload into CCNx Content Objects. It defines a name segment type to identify each sequential chunk number and a Content Object field to identify the last available chunk number. This includes specification for the naming convention to use for the chunked payload and a field added to a Content Object to represent the last chunk of an object. This document updates RFC8569 and RFC8609.

Table of Contents

1. Introduction

CCNx Content Objects [RFC8569] 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 uses state in the Name and in an optional field within the Content Object. A chunked object may also have an external metadata content object that describes the original pre-chunked object.

For example, a video file may be several gigabytes of data. To publish the video, one would divide it up into transport-sized Content Objects. Each Content Object would have a common name prefix plus a chunk number.

CCNx uses two types of messages: Interests and Content Objects [RFC8569]. 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 document adds a ChunkNumber to the CCNx Name Segment Type for conveying the chunk number and an EndChunkNumber to the CCNx Message Type for conveying the last chunk number of the content. It adds a ChunkSize CCNx Message Type to specify that a constant payload size is used to enable a consumer to seek to specific byte locations. It updates [RFC8609]. 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.

1.1. Requirements Language

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

2. Chunking

Chunking, as used in this specification, means serializing user data into one or more chunks, each encapsulated in a CCNx Content Object. A chunk is a contiguous byte range within the user data. One segment in the Name of each Content Object represents the chunk number. A field in the Content Object - 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 MAY use a fixed block size, where only the final chunk MAY use a smaller block size. This is required to allow a reader to seek to a specific byte offset once it knows the block size. If the first (Chunk 0) object has the field ChunkSize, then the series of chunked objects uses a fixed payload size. ChunkSize MUST only appear in Chunk 0, if it is used. The first chunk of user data (a chunk with a Payload TLV) may not be chunk 0.

Because of the possible use of a fixed block size, the inclusion of certain cryptographic fields in the same content objects as user data would throw off the ability to seek. Therefore, it is RECOMMENDED that all required cryptographic data, such as public keys or key name links, be included in the leading chunks before the first byte of user data. User data SHOULD then run continuously. If the publisher included ChunkSize in Chunk 0, then all Payload sizes must be of ChunkSize except the last, which may be smaller.

An advantage of using a Manifest (e.g. FLIC [FLIC]) is that all cryptographic data is in the manifest, which then links to each chunk via a full hash name. This means that the chunked user data can devote all available Content Object bytes after the name to the user data.

This draft introduces a new CCNx Name Segment TLV type, called the ChunkNumber. The ChunkNumber is the serial order of the chunks. It MUST begin at 0 and MUST be incremented by 1. The ChunkNumber is appended to the base name of the user data, and is usually the last name segment.

The value of the ChunkNumber TLV is a network byte order compact encoding of the chunk number.

A name MAY have more than one ChunkNumber name segments. This may happen, for example, if one chunked object is related to another chunked object. Only the last ChunkNumber name segment is significant.

EndChunkNumber is a new Content Object message type. It MUST be included in the Content Object which is the last chunk of user data, where the ChunkNumber TLV value and the EndChunkNumber TLV value MUST be the same. The value of the EndChunkNumber value is the same format as the ChunkNumber name segment. For example, if 3000 bytes of user data is split with a 1200 byte block size, there will be 3 chunks: 0, 1, and 2. The EndChunkNumber is 2.

The EndChunkNumber field MAY be present earlier, such as to assist in pipelining Interests. It MAY be increased by subsequent EndChunkNumber fields.

The last chunk is when the EndChunkNumber equals the right-most ChunkNumber name segment.

When a received Content Object contains both ChunkNumber TLV and EndChunkNumber TLV, and both values are the same, the consumer application will terminate reception of the content.

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 and have no payload. This mechanism for early termination is to accommodate circumstances where a stream ended before expected and the publisher needs to close early.

It may occur that a publisher never includes an EndChunkNumber, and thus has never published the last chunk. This may happen due to an error or network conditions that do not allow finding the last chunk. If a consumer times out trying to retrieve chunks, it SHOULD report an error to the user and terminate. This is similar to a TCP client not hearing a FIN.

If the user data fits within one Content Object and the publisher uses the Chunking protocol, the publisher names the content with "Chunk=0" and includes "EndChunkNumber=0" (assuming the Payload is in the first chunk).

To summarize:

1. Every chunk MUST have a ChunkNumber name segment, beginning at 0 and incrementing by 1.

2. The leading chunks MAY have missing or empty Payload TLVs and convey only cryptographic or other information.

3. An optional ChunkSize TLV MAY be in, and only in, Chunk 0. If it is present, it indicates that all Payloads are the same number of bytes, except the last which may be smaller.

4. The last chunk MUST have an EndChunkNumber TLV and the value MUST be equal to the ChunkNumber TLV value.

5. Content Objects before the last chunk MAY have an EndChunkNumber TLV with the expected last chunk number. These hints MAY be updated in subsequent Content Objects but SHOULD NOT decrease.

6. If the final chunk has a ChunkNumber less than a previously published EndChunkNumber, the publisher SHOULD pad out the chunks with empty Content Objects that have the true EndChunkNumber.

2.1. Cryptographic material

Chunk 0 SHOULD include the public key or key name link used to verify the chunked data. It is RECOMMENDED to use the same key for the whole set of chunked data. If a publisher uses multiple keys, then the public key or key name link for all keys SHOULD be in the leading chunks before any user data. Each subsequent chunk only needs to include the KeyId and signature.

The rational for putting all cryptographic data up front is because the protocol requires using a fixed block size for all user data to enable seeking in the chunked stream.

As noted above, using a Manifest eliminates the need for cryptographic material in the user-data Content Objects.

2.2. Examples

Here are some examples of chunked Names using the Labeled Content Identifier URI scheme in human readable form (ccnx:).

In this example, the content producer publishes a JPG that takes 4 Chunks. The EndChunkNumber is missing in the first content object (Chunk 0), but is known and included when Chunk 1 is published. It is omitted in Chunk 2, then appears in Chunk 3, where it is mandatory.

```
ccnx:/Name=example.com/Name=picture.jpg/Chunk=0  --
ccnx:/Name=example.com/Name=picture.jpg/Chunk=1  EndChunkNumber=3
ccnx:/Name=example.com/Name=picture.jpg/Chunk=2  --
ccnx:/Name=example.com/Name=picture.jpg/Chunk=3  EndChunkNumber=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 EndChunkNumber. Chunks 4, 5, and 6 do not contain any new user data.

```
ccnx:/Name=example.com/Name=talk.wav/Chunk=0  --
ccnx:/Name=example.com/Name=talk.wav/Chunk=1  EndChunkNumber=6
ccnx:/Name=example.com/Name=talk.wav/Chunk=2  --
ccnx:/Name=example.com/Name=talk.wav/Chunk=3  EndChunkNumber=3
ccnx:/Name=example.com/Name=talk.wav/Chunk=4  EndChunkNumber=3
ccnx:/Name=example.com/Name=talk.wav/Chunk=5  EndChunkNumber=3
ccnx:/Name=example.com/Name=talk.wav/Chunk=6  EndChunkNumber=3
```

3. TLV Types

This section specifies the TLV types used by CCNx chunking.

3.1. ChunkNumber

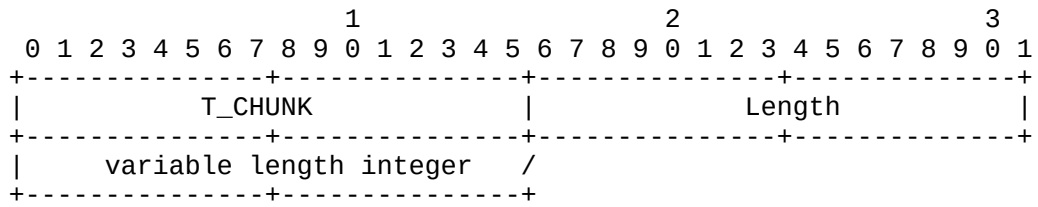
CCNx chunking defines one new CCNx Name Segment type: ChunkNumber.

Type	Abbrev	Name	Description
%x0005	T_CHUNK	Chunk Number (Section 3.1)	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.

Table 1: ChunkNumber

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 ccnx: URI form, it is denoted as "Chunk".



Code	Type name
%x0000	Reserved [RFC8609]
%x0001	T_NAMESEGMENT [RFC8609]
%x0002	T_IPID [RFC8609]
%x0003	T_NONCE [RFC9508]
%x0004	T_VERSION [I-D.ccnxcversioning]
%x0005	T_CHUNK
%x0006-%x000F	Unassigned
%x0010-%x0013	Reserved [RFC8609]
%x0014-0x0FFE	Unassigned
%x0FFF	T_ORG [RFC8609]
%x1000-0x1FFF	T_APP:00 - T_APP:4096 [RFC8609]
%x2000-0xFFFF	Unassigned

Figure 1: CCNx Name Segment Type Namespace

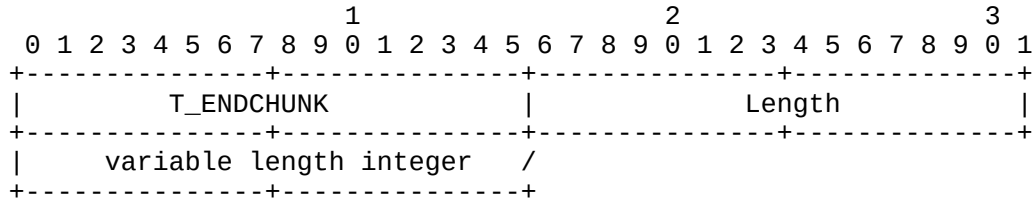
3.2. Message TLVs

CCNx chunking defines two new CCNx Message type: EndChunkNumber and ChunkSize.

Type	Abbrev	Name	Description
%x0007	T_ENDCHUNK	EndChunkNumber (Section 3.2)	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.
%x0008	T_CHUNK_SIZE	ChunkSize (Section 3.2)	The fixed payload size of user data up to the last chunk. It must be a positive integer.

Table 2: 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.



The chunk size number is the same as the end chunk number, except it must be a positive integer.

Code	Type name
%x0000	T_NAME [RFC8609]
%x0001	T_PAYLOAD [RFC8609]
%x0002	T_KEYIDRESTR [RFC8609]
%x0003	T_OBJHASHRESTR [RFC8609]
%x0005	T_PAYLDTYPE [RFC8609]
%x0006	T_EXPIRY [RFC8609]
%x0007	T_ENDCHUNK
%x0008	T_CHUNK_SIZE
%x0009-%x000C	Reserved [RFC8609]
%x000D	T_DISC_REQ [RFC9344]
%x000E	T_DISC_REPLY [RFC9344]
%x0FFE	T_PAD [RFC8609]
%x0FFF	T_ORG [RFC8609]
%x1000-%x1FFF	Reserved [RFC8609]

Figure 2: CCNx Message Type Namespace

4. Acknowledgements

5. IANA Considerations

As per [RFC8126], this section makes an assignment in one existing registry in the "Content-Centric Networking (CCNx)" registry group. The registration procedure is "RFC Required", which requires only that this document be published as an RFC.

5.1. CCNx Name Segment Type Registry

This document defines one message type, T_CHUNK, whose suggested value is %x0005.

5.2. CCNx Message Type Registry

This document defines two message types, T_ENDCHUNK, whose suggested value is %x0007, and T_CHUNK_SIZE, whose value is %0x0008.

6. Security Considerations

This draft does not put any requirements on how chunked data is signed or validated.

7. References

7.1. Normative References

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

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

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

[RFC8569] Mosko, M., Solis, I., and C. Wood, "Content-Centric Networking (CCNx) Semantics", RFC 8569, DOI 10.17487/RFC8569, July 2019, <https://www.rfc-editor.org/info/rfc8569>.

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

7.2. Informative References

[FLIC] Tschudin, C., Wood, C. A., Mosko, M., and D. R. Oran, "File-Like ICN Collections (FLIC)", Work in Progress, Internet-Draft, draft-irtf-icnrg-flic-07, 3 March 2025, <https://datatracker.ietf.org/doc/draft-irtf-icnrg-flic/>.

[I-D.ccnxcversioning] Asaeda, H., Hlaing, H. H., and M. E. Mosko, "CCNx Content Versioning", Work in Progress, Internet-Draft, draft-asaeda-icnrg-ccnxcversioning-00, 3 March 2025, <https://datatracker.ietf.org/doc/draft-asaeda-icnrg-ccnxcversioning/>.

Authors' Addresses

Marc Mosko
Kensington, California 94707
United States of America
Email: marc@mosko.org

Hitoshi Asaeda
National Institute of Information and Communications Technology
4-2-1 Nukui-Kitamachi, Tokyo
184-8795
Japan
Email: asaeda@nict.go.jp