ICNRG
Internet-Draft
Intended status: Experimental
Expires: July 13, 2015

M. Mosko
PARC, Inc.

January 9, 2015

CCNx Content Object Chunking
draft-mosko-icnrg-ccnxchunking-00

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 13, 2015.

Copyright Notice

Table of Contents

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.

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. 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.

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"
```

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> }]
  }
}
```

4. TLV Types

This section specifies the TLV types used by CCNx chunking.

4.1. Name Types

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

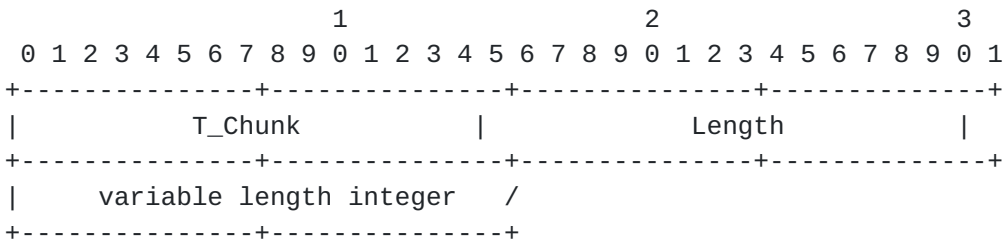
Type	Abbrev	Name	Description
%x0010	T_Chunk	Chunk Number (Section 4.1.1)	The current Chunk Number, is an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.
%x0011	T_Meta	Chunk Metadata (Section 4.1.1)	Identifies the Content Object on metadata for the prior Name.

Table 1: Name Types

4.1.1. Chunk Number

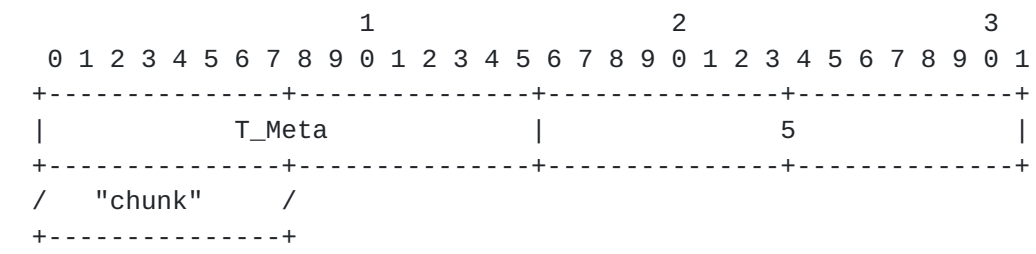
The current chunk number, as an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

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



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.



4.2. Protocol Information

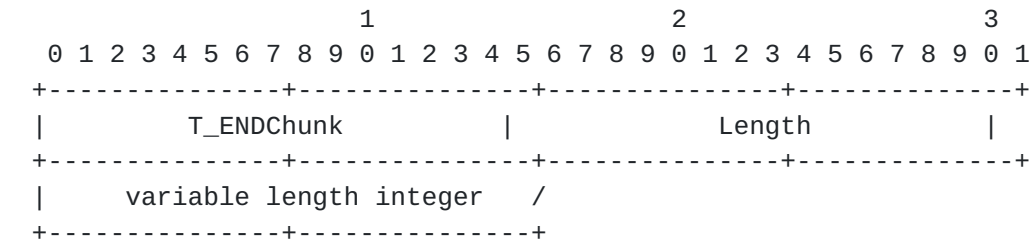
CCNx chunking uses one new field in the Metadata for the EndChunkNumber.

Type	Abbrev	Name	Description
%x0019	T_ENDChunk	EndChunkNumber (Section 4.1.1)	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

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.



5. Acknowledgements

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 [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.

7. Security Considerations

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

8. References

8.1. Normative References

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

8.2. Informative References

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

[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.

[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.

Author's Address

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

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