Order Encoded Manifests

Marc Mosko[1]*

Abstract
A CCNx Manifest lists a series of CCNx Content Objects and other Manifests that comprise one logical unit. If one arranges all the Content Objects pointed to by all the manifests, one would have the original logical unit. Usually the Content Objects are ordered from beginning to end, such that the arrangement is simple concatenation of the Content Objects in the order listed in each Manifest, in the order each Manifest is listed – that is, some form of tree. In some cases, a consumer may wish to fast forward or rewind through the encoded object, such as when viewing a movie or reading a document. We describe a Manifest structure to encode the content ordering in the manifest tree.

Keywords
Content Centric Networks – Named Data Networks

[1] *Palo Alto Research Center*
*__Corresponding author__: marc.mosko@parc.com

Contents

Introduction 1
1 Manifests 1
1.1 Examples 1
2 Ordered Manifests 2
2.1 External Metadata 2
Complete root index • Relative indicies
2.2 Extended Manifest Grammar 2
3 Conclusion 3

Introduction

We describe the standard CCNx Manifest, which allows for in-order content retrieval. We then present two methods for random-access to the ordered content. The first method uses external indices associated with the Manifest's metadata. The second method extends the standard Manifest format to include ordering markers.

Ordered content may come in many forms. For example, one could index the location of each frame of a movie. A 2 hour movie at 24 fps would have 172,800 frames, each at a time granularity of just under 42 milli-seconds. Depending on the codec, each frame may be spread over many packets or content objects. In an Order Encoded Manifest, each Manifest would include the starting frame number of the the manifest – that is, the left-most child included under the node's branch. In another example, a manifest encoding a book could include the chapter number or page number in the manifest to allow a user to quickly seek to the desired chapter or page.

1. Manifests

Fig. 1 shows the ABNF grammar for a standard Manifest. A Manifest is a CCNx Content Object, with optional Validation Algorithm and Validation Payload. The Validation Algorithm and Validiation Payload are for integrity checks (e.g. CRC) or authentication (e.g. HMAC or RSA signature). They do not affect the contents of the Manifest. Inside a Content Object, if the PayloadType is `MANIFEST`, then the Payload of the Content Object should be parsed as an encoded `ManifestPayload`.

The `ManifestPayload` is composed of `MetadataSection` and `PayloadSection`. A Metadata section points to elements that describe the PayloadSection, for example it may describe the video encoding used for the payload or the table of contents of a book. Both are encoded as a `SECTION`.

A Section is defined as an optional Access Control List (ACL) – described in [cite] – followed by two arrays. The first array is the `ListOfNames` followed by the `ListOfHashes`. Each entry in ListOfNames indicates its root `MediaName`. The list of media names may be empty if all hashes are based off the current Content Object's Name. The ContentObject Name is always NameIndex 0, so the ListOfNames will always begin with NameIndex 1. If the ContentObject-Name has a ChunkNumber, then that number plus 1 is the implied StartChunk of NameIndex 0. A `NameEntry` specifies a media name, with an optional `StartChunk`. If the StartChunk is present, then the names are assumed to include a Chunk name component. The starting chunk number will be $StartChunk + relativeOrder$, where the *relativeOrder* is the ordinal position of the ListOfHashes entry that uses the NameIndex. Examples are given below in Sec. 1.1. If no StartChunk is present, the name is not chunked (i.e. does not have a "Chunk=" name segment).

1.1 Examples

Fig. 2 shows a Manifest in JSON format.

In this example, the set of generated Interests would be:

```
Interest 0 = { Name = /parc/obj29/chunk=0,
```

Figure 1. ABNF for standard Manifest

```
SignedObject     = ContentObject [ValidationAlg]
                     [ValidationPayload]
ValidationAlg    = <e.g. RSA, HMAC>
ValidationPayload= <e.g. a signature>

ContentObject =  Name CreateTime [ExpiryTime]
                 PayloadType [Payload |
                 ManifestPayload]
Name = <CCNx Name of the Content Object>
CreateTime = <UTC time>
ExpiryTime = <UTC time>
PayloadType= DATA | MANIFEST | <others>
DATA       = 0  <application payload>
MANIFEST   = 6  <Manifest payload>
Payload    = *OCTETS

ManifestPayload = *[MetadataSection |
                  PayloadSection]
MetadataSection = SECTION
PayloadSection  = SECTION

SECTION     = [ACL] [ListOfNames] ListOfHashes
ACL         = LINK
ListOfNames = *(NameEntry)
NameEntry   = [StartChunk] MediaName
ListOfHashes= NameIndex HASH
NameIndex   = OCTET
HASH        = 32(OCTET)

LINK    = TargetName [TargetKeyId] [TargetHash]
TargetName = <ccnx name of link target>
TargetKeyId= <KeyId Restriction>
TargetHash = <ContentObjectHash Restriction>
```

Figure 2. Example (JSON representation)

```
ManifestPayload = {
.ListOfNames = [
 {.StartChunk=1,.MediaName=/parc/obj29/manifest},
 {.StartChunk=0,.MediaName=/parc/obj29} ],
.ListOfHashes = [
 {.NameIndex = 2, .Hash= 0x123},
 {.NameIndex = 1, .Hash= 0xAAA},
 {.NameIndex = 2, .Hash= 0x456},
 {.NameIndex = 1, .Hash= 0xBBB} ] }
```

```
                Hash = 0x123 }
Interest 1 = { Name = /parc/obj29/manifest/chunk=1,
                Hash = 0xAAA }
Interest 2 = { Name = /parc/obj29/chunk=1,
                Hash = 0x456 }
Interest 3 = { Name = /parc/obj29/manifest/chunk=2,
                Hash = 0xBBB }
```

This allows the Manifest to indicate an interleaving of fetching series of objects with different media names and different chunk number sequences. In the example, the first set is for DATA objects (hash 0x123 and 0x456) and the second set is for hierarchical manifests (hash 0xAAA and 0xBBB).

2. Ordered Manifests

The first method uses external indices associated with the Manifest's metadata. The second method extends the standard Manifest format to include ordering markers.

2.1 External Metadata

One method for random-access retrieval from an ordered manifest is to provide an external index. One advantage of the external index is that a Manifest may specify multiple Metadata sections to provide different indices. For example, a book might have a chapter index, a figure index, and a page index.

An external index is a set of standard CCNx Content Objects of PayloadType = Index organized by their own Manifest tree. In general, an external index should not use another external index, but should use an extended manifest grammar for seeking.

2.1.1 Complete root index

The first method is to have the root manifest object include a Metadata section that points to a complete, exhaustive table of contents. The root index would be an array of, for example, video frame number to the nearest manifest containing that frame.

For the case of a 2 hour video at 24 fps, there are 172,800 frames. Each Manifest entry takes 32 bytes per hash, so to index all the frames would take 5.5 MB, plus the encapsulation overhead, so perhaps 8 MB.

2.1.2 Relative indicies

A second method for external metadata is for each Manifest object to include a Metadata section that points to an Index object. Rather than being a complete index, it only gives the ordering information for the direct children of the current manifest. A node would need to walk down the manifest tree to locate a specific point.

The external index would also include a parent pointer to the name of the parent manifest so a node could also walk up the tree. It should also include a pointer to the root manifest (if different than the parent) so a node can quickly skip to the root of the tree. This method could, in fact, use any form of threading through the tree for quicker seeks.

Figure 3. Modified SECTION

```
SECTION     = [ACL] [ListOfNames] ListOfHashes
              [Ordering]
Ordering    = *(OCTET)
ACL         = LINK
ListOfParents= *(ParentEntry)
ParentEntry = [Ordering] LINK
ListOfNames = *(NameEntry)
NameEntry   = [StartChunk] MediaName
ListOfHashes= [Ordering] NameIndex HASH
NameIndex   = OCTET
HASH        = 32(OCTET)
```

2.2 Extended Manifest Grammar

By extending the manifest grammar we allow random access to the content without the extra round trips required to fetch an external metadata object. We need to add pointers to parent objects, so rewinds are efficient, and add ordering data to each manifest, parent, and child entry.

3. Conclusion

We have described three methods to provide random-access to an ordered manifest. The first method is to have a complete index referenced from the root manifest's Metadata section. The second method is to use relative indices from each Manifest. The third method is to extend the Manifest grammar to include ordering attributes in the Manifest such that each Section has its optional starting order, each child link has an optional next order, and each parent has its optional previous order.