All-In-One Streams for Content Centric Networks

Marc Mosko and Ignacio Solis
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304
Email: {mmosko, isolis}@parc.com

***Abstract*—To reduce download latency, All-In-One content streams concatenate many CCNx Content Objects in to one named stream so a requester may open up a large Interest window and download a set of related content objects without incurring a round-trip time to first read a manifest. We describe the structure of an All In One Stream, and how the manifest allows the use of the single object stream or the parallel use of independent streams. We also describe how a modular technology called Reconstructable Objects allows an in-path cache to create sections of the All-In-One content stream from cache rather than having to fetch the entire stream from the origin. Modeling shows improved download speeds of up to 57%.**

I. Introduction

As is well known from typical server applications [1], round trip times can add significant latency to content download. For example, with HTTP over TCP, one typically has a 3-way TCP handshake then a GET request before beginning to download the desired HTTP and HTML markup. Then, after parsing the HTML markup, the client can request individual embedded objects. Just as there are efforts to reduce to so-called "zero round trip time" protocols in the IP world, we may devise such protocols for CCNx [2]. All In One Streams is an efficient and secure zero-round trip time protocol that allows a client to skip already cached pieces of the download and to exploit external caching of embedded objects.

All-in-one streams is also an efficient mechanism to handle Manifest content. A CCNx Manifest is similar to an HTML index.html file: it enumerates other content objects that make up the parent object. Normally, one would need to fetch the manifest first, then fetch the constituent pieces. Using an All-In-One stream, the manifest and it contents may be fetch in a zero-round-trip Interest stream.

In a typical CCN deployment, a system encodes user data in to Content Objects. Each network transmissible content object contains a small piece of the original data, say 1500 to 8000 bytes. After a receiver as all the constituent objects, it may reconstruct the original user data. If the receiver wishes to re-transmit the content objects, it must save them, along with the user data, because the content objects are cryptographically signed by the original publisher and the receiver cannot reconstitute that signature if it throws away the objects. This leads to a potentially large set of duplicate data on a user's system to store the content object representation and the original user data representation. We describe a method to avoid the duplicate data using Reconstructable Content Objects. When applied to an All-In-One stream, Reconstructable Objects allows intermediate caches to respond to parts of the stream without needing to fetch the entire stream from the source.

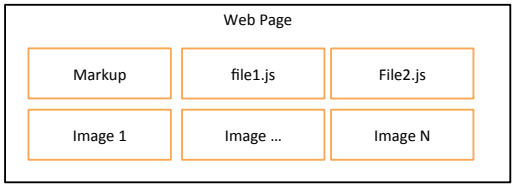

Fig. 1. Typical web page components

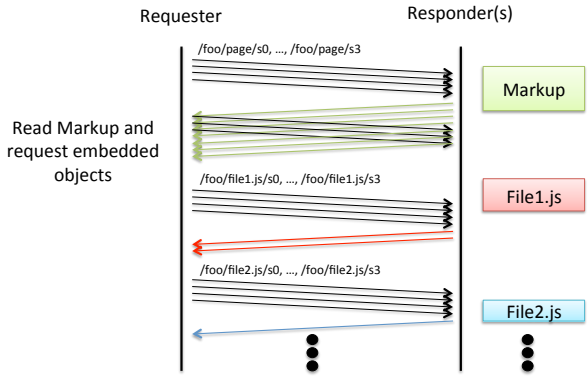

Fig. 2. One-by-one download

II. One-by-one protocols

Conventionally, a CCNx system would issue some initial set of Interest messages to read an piece of content, such as an HTML markup page, then begin downloading the individual components from that page. The download is constrained by some maximum number of outstanding interests that saturate the bottleneck link, typically the last hop. Also, a requester does not know *a priori* how many chunks to request, so for small objects the requester may waste some outstanding interests by over-requesting chunks.

For example, in Fig. 1, a notional web page consists of the markup and objects references from the markup. These subordinate objects may include some javascript files, such as file1.js and file2.js and embedded images. To download the complete web page, a client must first ask for the markup, and then begin parsing the markup for embedded objects to request those objects.

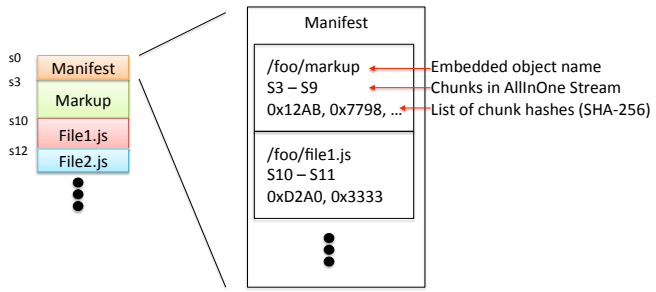

Fig. 3. All In One Manifest

As illustrated in Fig. 2, a conventional download may end up staggering the download of embedded objects because of the need to retrieve the manifest first. Also, because the Requester does not know the number of segments in an object *a priori*, the Requester may open up too large of a window, such as when the Requester issues 3 interests for file1.js when it only needed to issue 2 interests. The extra interest could have been used to request useful content, such as beginning the download for file2.js.

III. All In One Streams

An All In One Stream is a method for a content provider to aggregate a set of content in to one named stream. A Stream consists of an initial Manifest followed by constituent objects. The entire stream is under one chunked namespace, so a requester can open up one large Interest window that downloads the manifest plus all locally served content. The manifest has enough information that a requester could skip specific embedded objects if it already has them.

Fig. 3 shows an example manifest, which is an array of manifest entries. Each entry has the CCNx base name of the embedded object (minus the chunk number), the list of chunk numbers in the All In One stream, and the Content Object Hash of each embedded content object. Enumerating the chunk ranges of each embedded content object allows the requester to determine if the object is already covered by the outstanding Interest window or to skip embedded content objects if the Requester already has an object with the same Content Object Hash. Enumerating the Content Object Hashes of each embedded object allows the requester to open up separate Interest windows for the embedded object and request it by hash.

Fig. 4 shows the ABNF for an All-In-One stream. It begins with a stream Manifest and is followed by embedded Content Objects. The stream manifest enumerates the CCNx base name (name before chunk number) of an entry, the starting chunk number in the embedded stream and the chunk length in the embedded stream. It also enumerates the ContentObjectHash of each embedded object (the inner object, not the All-In-One stream object). At minimum, the first content object of an All In One stream should carry the public key used to sign the stream so intermediate nodes and end systems can verify signatures.

Fig. 4. ABNF for All-In-One Stream

```
Stream = Manifest *(Embed)
Manifest = *(ManifestEntry | ManifestContinuation)
ManifestEntry = Name StartChunk ChunkLength
                [Flags] (ChunkLength)(Hash)
Name = <CCNx Name>
StartChunk = ChunkNumber
ChunkLength = Number
ChunkNumber = Number
Flags = [gzip]
gzip = <embedded object gzip'd>
Hash = <SHA-256 ContentObjectHash>
ManifestContinuation = [PreviousChunk] [NextChunk]
PreviousChunk = ChunkNumber
NextChunk = ChunkNumber
```

The Manifest may be interleaved through the All-In-One stream using ManifestContinuation records which point to the previous Manifest and next Manifest using the All-In-One stream chunk numbering. This organization is similar to an LTFS [3] tape volume, where Index and Data Extent partitions can be sequenced in the LTFS Volume. Rather than include generation numbers in each index, we use the fact that there is a common CCNx name prefix for each chunk of the All-In-One stream, such as with a version name component, to indicate that all indices and data extents belong to the same stream.

The publisher creating an All-In-One stream may organize the manifest to optimize download. For example, the manifest may list only a few initial content object hashes of each embedded object rather than a complete enumeration. Listing only a few initial hashes gives the requester enough information to determine if they have the object and to begin downloading it under its own namespace. Later index sections could then finish the embedded object enumerations.

The publisher may generate the embedded objects two ways. It could create one embed object per embedded chunk. Or, it could create a non-chunk aligned embedding of the content. Either method may be used. If the stream uses compression then one is generally using the second, non-aligned method. If the gzip flag is present in a Manifest entry, it means the embed object uses the Gzip protocol to compress the inner object.

The Interest request could include a "modified since" parameter in the name, so the All In One stream would only include embedded content objects modified after the embedded parameter. It may also included older objects that are newly referenced, for example if an old photo has not been included in a web page manifest for a long time, it could be included in the All In One stream even though it has not been modified since the Interest parameter.

A Requester may request embedded objects under their own name, using a self-certified Content Object Hash name. Downloading the objects under their own name could allow

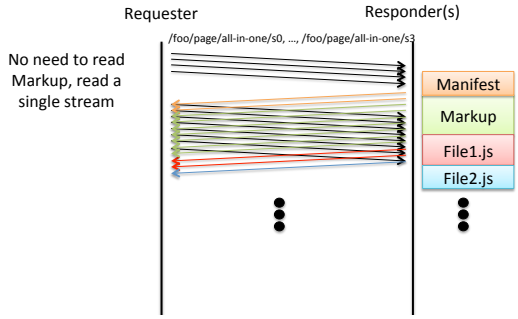

Fig. 5. All In One Manifest

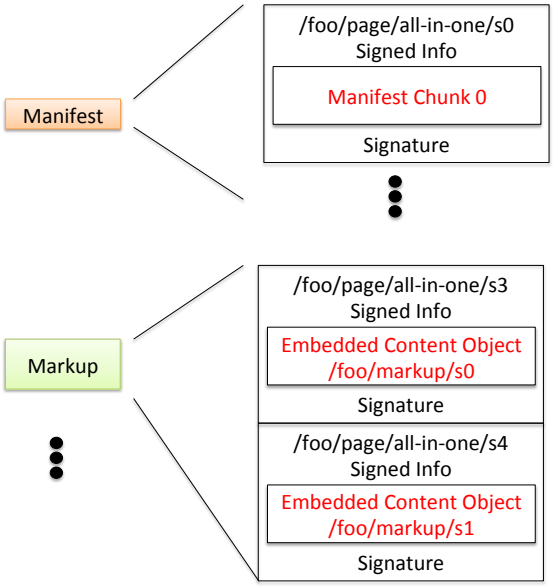

Fig. 6. Example All In One Stream Content Objects

the download to come from well-positioned caches, whereas downloading the embedded objects might need to come from a less optimal source. For example, image files may have very long cache lifetimes, so would be cached in more places, while the web page might have a short cache lifetime because it is frequently updated.

Fig. 6 illustrates an example stream. The manifest is one or more content objects whose payload is the Manifest. It is signed, as normal, for a CCNx Content Object. The next objects in the stream are embedded inside stream objects. This is because the stream objects have their own stream name. Therefore, /foo/page/all-in-one/s3 is a CCNx Content Object that embeds /foo/markup/s0, being the first chunk of the actual markup. Both the wrapping object and the embedded object have their own CCNx signatures. The embed objects could omit the CCNx signature because of the ability to hash chain from the Manifest.

The embedded objects might themselves be All In One streams. For example, one HTML file might reference frames of other HTML or other objects, which could themselves

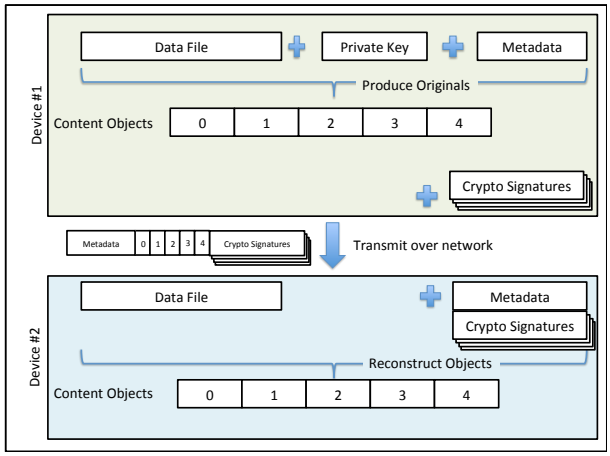

Fig. 7. Reconstructing individual content objects

be organized as an All In One stream. In such a case, the embedded All In One stream is treated like one embedded object in the parent manifest.

IV. All In One Streams and Reconstructable Objects

Reconstructable Content Objects store the original user data as a normal file on a user's system. This is the normal form an application would need to use the data, such as a JPEG or a game's data files. In addition to the user data, the system stores a set of rules. The rules describes how to publish the user data, such as the number of bytes per content object, the timestamps to use, how to name the content objects, and other things that go in each Content Object. With the original data file and the reconstruction rules, a system can verbatim reproduce the original content objects without duplicating the data. Finally, the system stores the original publisher's signature(s) for the data. In a system that uses directly signed content objects, there are many signatures. For a system that uses manifests (secure catalogs), there may only be one or a small number of signatures.

Reconstructable Content Objects consist of several components. There is the underlying user data which is chunked in to one or more Content Objects. The chunking is done via a set of rules embodied in a ruleset. This ruleset provides all information for all content objects except for the cryptographic signatures. Finally, there is a set of cryptographic signatures, one for each signed content object. This is a necessary and sufficient set of objects to construct content objects from only the underlying user data.

Fig. 7 illustrates the process of creating and then reconstructing content objects. An initial device, noted as "Device #1" in the figure, has an original data file. Using a metadata ruleset, it constructs and initial set of content objects. The metadata ruleset specifies how a device fills in all the fields of a content object, such as the Creation Time or when to use an End Segment field, and the format of the CCNx names. The initial device then cryptographically signs each content object.

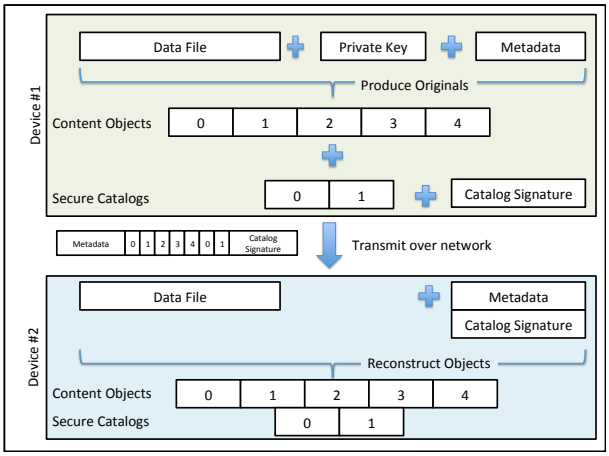

Fig. 8. Reconstructing using secure catalog

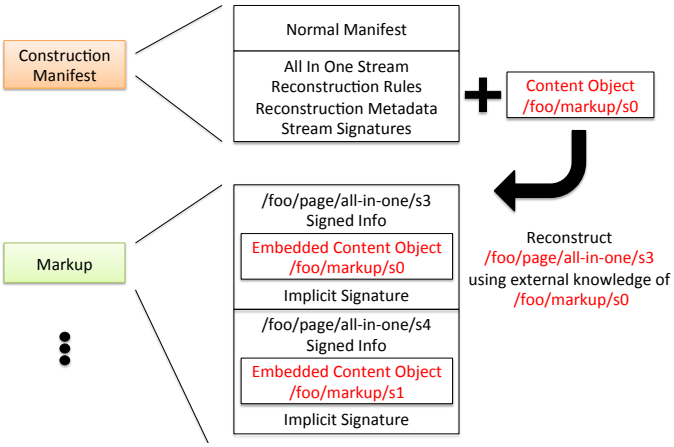

Fig. 9. Reconstructing Stream Content Objects

This creates the set of signatures shown in the figure, which are actually part of each content object.

The initial device in Fig. 7 then transfers the metadata ruleset plus the content objects over the network to a second device, noted as "Device #2". The second device reconstructs the original user data file and saves the metadata ruleset plus the cryptographic signatures. This this small additional data, it may reconstruct the original content objects by following the metadata ruleset and plugging in the saved cryptographic signatures.

In a variation of this system, the first device may use a secure catalog to authenticate the content objects. This scheme, show in Fig. 8, only has a signature on the secure catalog, not on each individual content object. Therefore, the state transferred over the network and saved at the second device is potentially much smaller than the system where each content object is individually signed. In this variation, the rules for creating the secure catalog are in the metadata, so the secure catalog itself does not need to be stores.

A. All-In-One Streams with Reconstruction

A reconstructable content object comprises a set of rules about how to construct content objects from user data, a set of metadata, a set of cryptographic signatures, and the user data. From this set of data, a system may re-construct a CCNx Content Object without having the actual content object. A reconstructable All-In-One stream uses this system to reconstruct wrapped content objects. The original (inner) content object is the "data file" and the reconstruction rules describe how to wrap the inner content object in the All-In-One stream. This allows, for example, a cache to respond to an All-In-One chunk request while storing only the inner wrapped content object and the reconstruction rules.

A forwarder, for example, may already have an embedded object in its Content Store. Using a reconstructable stream allows the forwarder to reconstruct a stream segment from the locally cached embedded object. Fig. 9 shows an example reconstructable All In One stream. The manifest is now a Constructable Manifest, which includes the normal manifest plus Reconstructable Object rules, metadata, and signatures. This allows a system that already has the embedded object, such as /foo/markup/s0, to construct the All In One stream object, such as /foo/page/all-in-one/s3 without having to fetch the wrapper /foo/page/all-in-one/s3.

An advanced forwarder that understands All In One manifests can pro-actively fetch embedded objects before the user asks for them, then use a the reconstructable stream to pass them on to the user. It could cache the embedded objects separately from the wrapped stream objects so in the future it could respond with either content object using its hash-based name.

Note that the Construct Manifest also serves as a secure catalog. It contains all the signatures for the All In One stream, so each subsequent content object in the stream may refer to the Construct Manifest for signature verification.

If the Construction Manifest carries a GUID (such as a SHA-256 hash of the manifest) in the first content object of the manifest, then an intermediate forwarder could detect if its seen the entire All In One stream before. In such a case, the forwarder can immediately satisfy all interests in the All In One Stream based on cached objects or reconstructable streams. A system should only do this if it can verify the signature of the first content object.

The Construction Manifest could specify the cryptographic hashes of subsequent content objects in the All In One stream. This means the subsequent content objects do not carry any signatures, but are verified solely by the signatures in on the Construction Manifest.

V. Analysis

We model the performance of All-In-One streams using several stream size distributions assuming a lossless network and TCP-like window behavior. Each stream contains 3, 10, or 30 embedded objects, where each object is uniformly $1KB \ldots 50KB$. All content objects are segmented at 1500

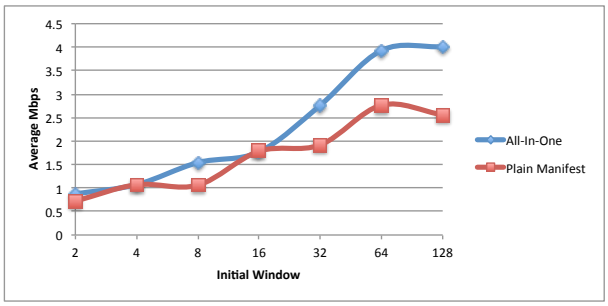

Fig. 10. Avearge Mbps for 86 KB download

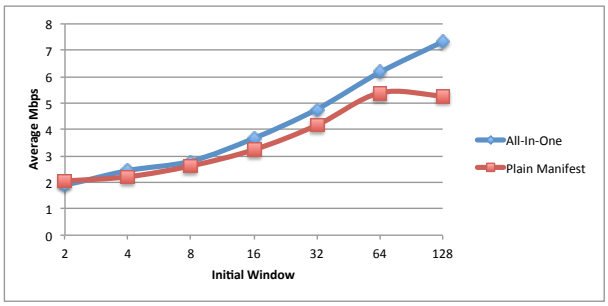

Fig. 11. Avearge Mbps for 296 KB download

bytes (about 1400 bytes of Manifest payload). Only the Manifest chunks are signed. Wrapped objects are not signed, but may be verified based on the Manifest hash. The TCP-like window behavior begins each Content Object download with K initial window Interest packets and doubles until link saturation. We use 10 Mbps as the bottleneck link speed with a total 50 msec one-way latency. We used 5 random simulations per data point and present the average.

We compare a system that uses All-In-One manifests, which maintains a single optimal TCP-like window over the entire stream to a plain Manifest system. In the plain Manifest system, once the first chunk of the Manifest is read, the system can continue reading the referenced objects and we assume the TCP-like window maintains its optimal size. That is, it does not re-start the TCP window for each object. If the flow controller implementation restarts the TCP window per object, the performance will be significantly worse than what we show here. Based on this setup, we expect that the All-In-One stream will show the most improvement for smaller downloads where the advantage of opening up a large initial window gains the most benefit.

Fig. 10 shows the average Mbps of download speed when the system begins with a given initial window. In these runs the average download size is 86 KB. Even with small initial windows, the All-In-One stream can be 50% faster than the plain Manifest version. At large initial window sizes, the benefit is pronounced and consistent, ranging from a 42% to 57% improvement. At an initial window size of 4, the All-In-One stream was 1% worse than the plain manifest.

Fig. 11 shows the average Mbps of download speed where the average download is 296 KB. At an initial window of 2,

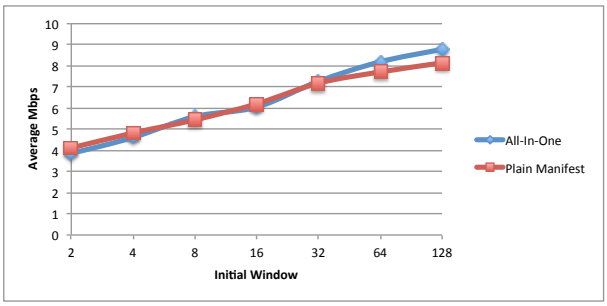

Fig. 12. Avearge Mbps for 905 KB download

the All-In-One stream is 8% worse than a plain manifest, and in all other cases, the All-In-One stream is between 7% better up to 39% better with a large initial window.

Fig. 12 shows the average Mbps of download speed where the average download is 905 KB. At 2, 4, and 16 for the initial window size, the All-In-One stream was between 2% to 6% worse than a plain manifest. At other values, the All-In-One stream is between 3% to 8% better. Clearly, at this size of download the advantage of eliminating the initial round trip to begin reading data is nearly lost.

VI. Conclusion

All In One streams is an optimized download method for content networks. It allows a requester to open up one large Interest window and download a manifest plus related objects without needing to parse the manifest before downloading actual content. The organization of an All In One stream allows the requester to skip embedded objects it already has or to download embedded objects from their manifest name using their own namespace.

Modeling of download of an All-In-One stream shows significant improvement, up to 57% faster, than using just a Manifest and waiting for an extra round trip before downloading content. If the download flow controller restarts the download Interest window per object, the improvement of All-In-One streams would be significantly higher. The benefit of All-In-One streams is highest for smaller downloads, say under 500KB, and becomes small by 1MB, where the benefit of having a large initial download window to avoid an extra round trip begins to vanish.

References

[1] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2. RFC draft-ietf-httpbis-http2-14, July 2014.

[2] Content centric networking specification. http://www.ccnx.org/documentation, 2014.

[3] *Linear Tape File System (LTFS) Format Specification*, version 2.0.0 edition, March 2011.