Difference Based Content Networking

Marc Mosko[1]*

Abstract
Difference Based Content Networking (DBCN) uses diffs between versions to optimize the amount of data sent on the network for version updates of data files. This paper describes several variations on DBCN based on binary diffs, content object diffs, byte offset diffs, and chunk catalog diffs. Existing CCNx versioning methods use total replication of data file bytes between versions, resulting in poor efficiency in data transfer.

Keywords
Content Centric Networks – Named Data Networks

[1] *Palo Alto Research Center*
***Corresponding author**: marc.mosko@parc.com

Contents

Introduction

Diff-based content networking (DBCN) uses differences between versions to reduce the amount of data transferred on the network. For example, if a user updates a 10 Mbyte file with a 1 Kbyte change, it would be encoded as the original 10 Mbyte file and a second 1 Kbyte diff. If a remote user already has the 10 Mbyte file, she would only need to transfer the 1 Kbyte diff.

DBCN is similar to versioned file systems, or in some ways journaled file systems. There is a ground truth of the original object, then a series of diffs. At some point, a new ground truth may be written to avoid needing a large number of diffs. Different implementations of DBCN may use different strategies for writing new ground truths or consolidating diffs to optimize object transfer.

DBCN uses secure catalogs, sometimes called Aggregated Signing Objects (ASOs). This allows a user to limit cryptographic signing to only the secure catalog (ASOs) rather than sign every content object that constitute the underlying data.

DBCN exploits the use of secure catalogs to efficiently encode version differences by referencing the secure catalog of an earlier version and then indicating the differences to the new version. There are many different ways of using diff-based encoding. This article describes several alternatives, including content diffs and ASO diffs. In the following, we describe using a "binary diff" at times. This means measuring the byte location of a difference and indicating the new bytes that should replace the old bytes. For text-based data, a standard text diff could be used instead, with the proper indication of the diff type in the encoding.

The preferred method of DBCN is described in Section 5. This method combines data reduplication techniques with differentially-encoded secure catalogs. Earlier sections describe other variations the provide background as to why Section 5 is the preferred method.

Unless otherwise indicated, a secure catalog does not need to reference the immediately preceding secure catalog to form the secure catalog tree. It may skip back to earlier versions, or not include one at all.

1. Diff-encoded content objects

In the first variation of diff-encoded content objects, as shown in Fig. 1, a data file (or memory) is partitioned into a set of content objects and those content objects are named with sequential numbers. This is a traditional way to generate CCNx content objects from an underlying data file.

Each content object would be named with its sequence number, such as `/parc/csl/papers.doc/v0/s0`, where `s0` indicates segment 0 of version 0. Each content object has an implicit Content Object Hash, being the SHA-256 cryptographic hash of the content object. This allows exact retrieval of a matching content object with cryptographic verification that the object received is the object desired.

The secure catalog would have a name such as `/parc/csl/paper.doc/v0`, where the component `v0` indicates the first version. The catalog enumerates the Content Object Hash of each constituent content object in order. This allows complete reconstruction of the data file using only the signature on the secure catalog.

A new version, such as `/parc/csl/paper.doc/v1`, has a secure catalog that points to the previous version (`/parc/csl/paper.doc/v0`) and the new content objects that make up the binary diff. The binary diff is a structured list of the binary differences from `v0` to `v1`, such that

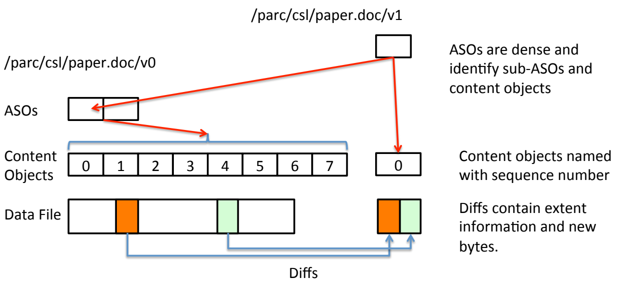

Figure 1. Differential data using binary diff encoding

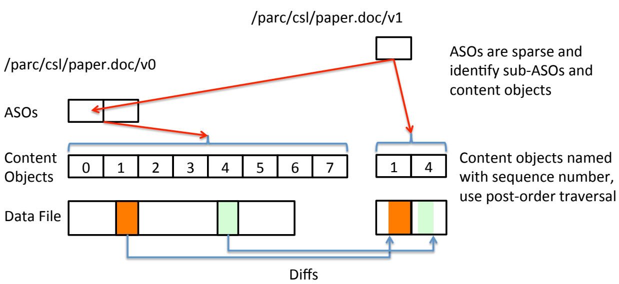

Figure 2. Differential data using content object diffs

one could "patch" the older version to get the newer version. Due to insertions or deletions in the new version, this may be an inefficient representation.

In this method, the encoded content objects only contain the minimum number of difference bytes between the two versions and annotations inside the content object payload describe where those differences occur in the previous version.

2. Content Object diffs

In this variation, instead of doing binary diffs on the data file and encoding that diff, this variation uses diffs on the content object sequence numbers. Fig. 2 shows an example where the original data files encodes to 8 content objects and the diff to version 1 replaces two of those content objects with new objects.

To reconstruct a version, one does a post-order traversal of the secure catalog and only uses the right-most occurrence of a content object sequence number.

As with the previous variation, this method may incur high overhead of bytes are inserted, causing a right-shift of content objects. If bytes are removed, one could easily elide those bytes using a single content object, which may be empty if all bytes in the previous version of that object are replaced.

In this variation, because the diff is done at the content object level, each content object must contain all the bytes being replaced in the previous content objects. As such, if a small number of bytes are changed, for example 128 bytes in an 8KB content object, all 8KB of the new version must ben encoded in the replacement content object.

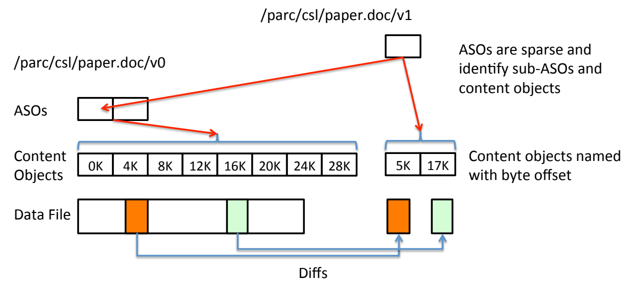

Figure 3. Differential data with byte offsets

3. Byte-offset Content Object diffs

In this variation, shown in Fig. 3, uses byte offsets to label where the bytes inside a content object should be placed in the previous version. The payload of the content object indicates if it represents an "insert" or "replace" or "deletion" operation.

To reconstruct a version, one does a post-order traversal of the secure catalog tree and maintains an interval graph of the data file.

4. Secure Catalog for Chunk Enumeration

In this variation, depicted in Fig. 4, a data file is broken up in to chunks, using data de-duplication technology. Chunks usually vary from 4KB to 16KB, depending on the data and the technology used.

Each chunk is named by its cryptographic hash, such as its SHA-256 name. In one variation, the names appear similar to /parc/csl/paper.doc/<chunk_hash>, where <chunk_hash> is the 32-byte hash. In another variation, chunks may be kept under a higher-level chunk repository, such as /parc/<chunk_hash>.

The secure catalog has a versioned name, such as /parc/csl/paper.doc/v0, where v0 indicates it is the first version. The secure catalog then enumerates all chunk hashes in order. The secure catalog only needs to name the hashes to the extent needed to find them. For example, if the system stores the chunks under /parc/csl/papers.doc, the secure catalog only needs to state this in one place, then the remainder of the entries are only the 32-byte chunk names.

For the next version, such as /parc/csl/papers.doc/v1, the new secure catalog enumerates the new set of chunks.

5. Diff-based Secure Catalogs for Chunk Enumeration

This variation is also based on chunks, as shown in Fig. 5. In this case, the secure catalog is a diff of previous secure catalogs. This allows easy insertion or removal of chunks and

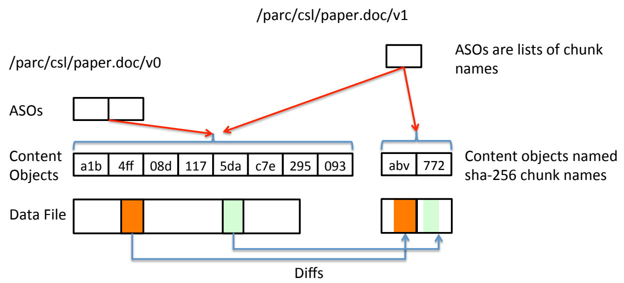

Figure 4. Chunk de-duplication encoding

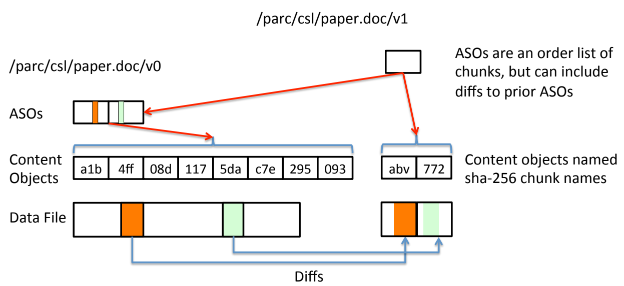

Figure 5. Chunk de-duplication encoding with ASO diffs

because of the differential encoding, the secure catalogs for subsequent versions may be small.

Diffs do not need to be solely to the previous version's catalog. The secure catalog diff may point to any number of earlier versions indicating the diffs to those versions. The final secure catalog is constructed via a post-order traversal of the secure catalog tree.

6. Conclusion

Diff Based Content Networking offers several efficient ways to encode data over versions that do not require transmitting all the bytes of an object with each version. Furthermore, it leverages perviously-signed secure catalogs to reduce the signing overhead of publishing a new version.