Data Deduplication with FLIC Manifests

NDNComm 2025

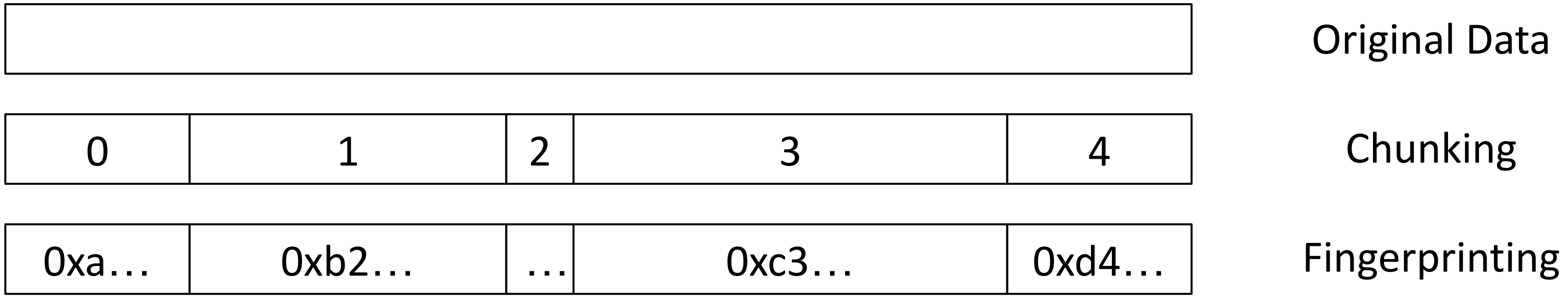
Marc Mosko
SupraLiminal Technologies

https://github.com/mmosko/ccnx-dedup
https://github.com/mmosko/ccnpy
https://datatracker.ietf.org/doc/draft-irtf-icnrg-flic/07

Outline

- Data Deduplication
- FLIC Manifests
- Methodology
- Results
- Conclusion

Data Deduplication [1]



Original Data

0 | 1 | 2 | 3 | 4 — Chunking

0xa… | 0xb2… | … | 0xc3… | 0xd4… — Fingerprinting

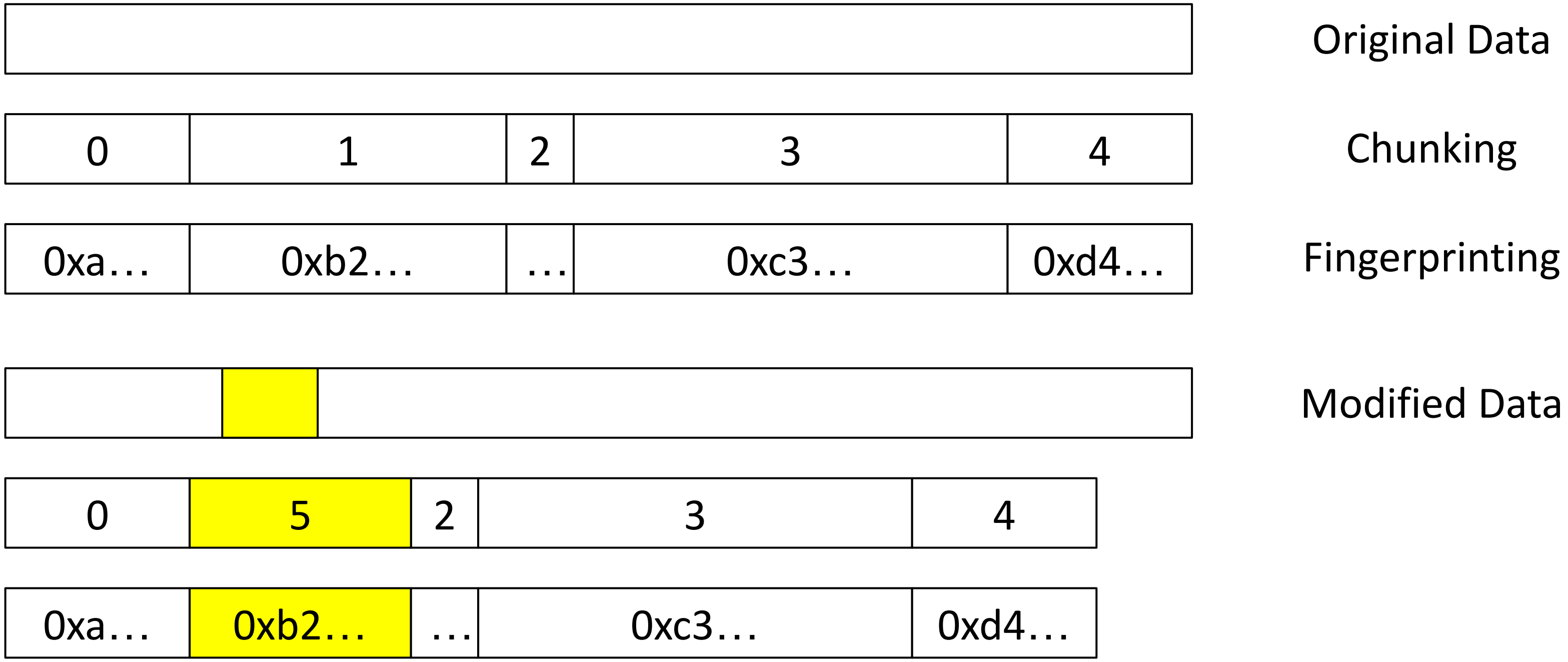
Chunking splits the data at breakpoints.

Fingerprinting uses a fast hash to screen chunks.

A strong hash or byte comparison fully matches chunks.

[1] Karp, R.M. and Rabin, M.O., 1987. Efficient randomized pattern-matching algorithms. IBM journal of research and development, 31(2), pp.249-260.

Deduplication Matching



Original Data

0 | 1 | 2 | 3 | 4

Chunking

0xa… | 0xb2… | … | 0xc3… | 0xd4…

Fingerprinting

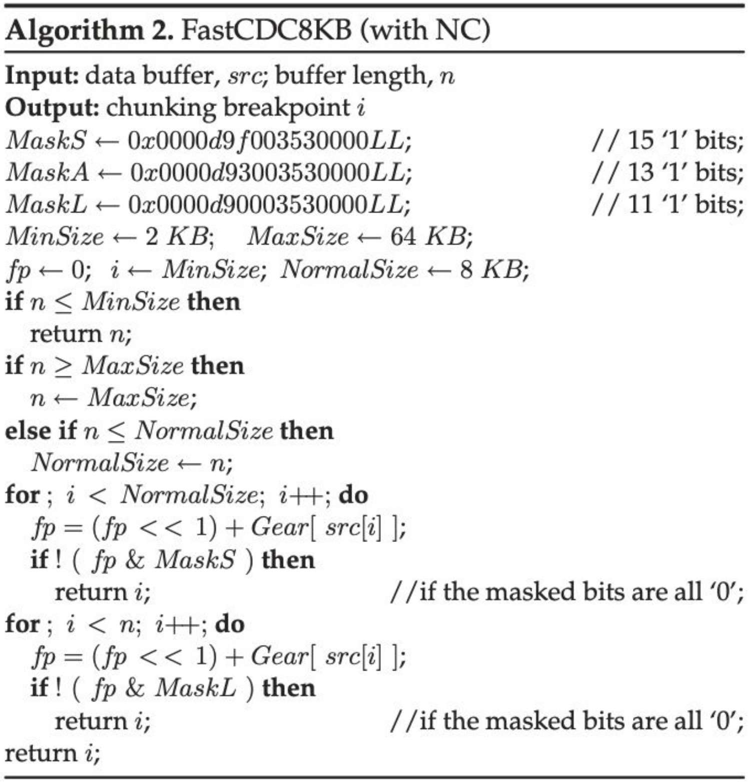
Modified Data

0 | 5 | 2 | 3 | 4

0xa… | 0xb2… | … | 0xc3… | 0xd4…

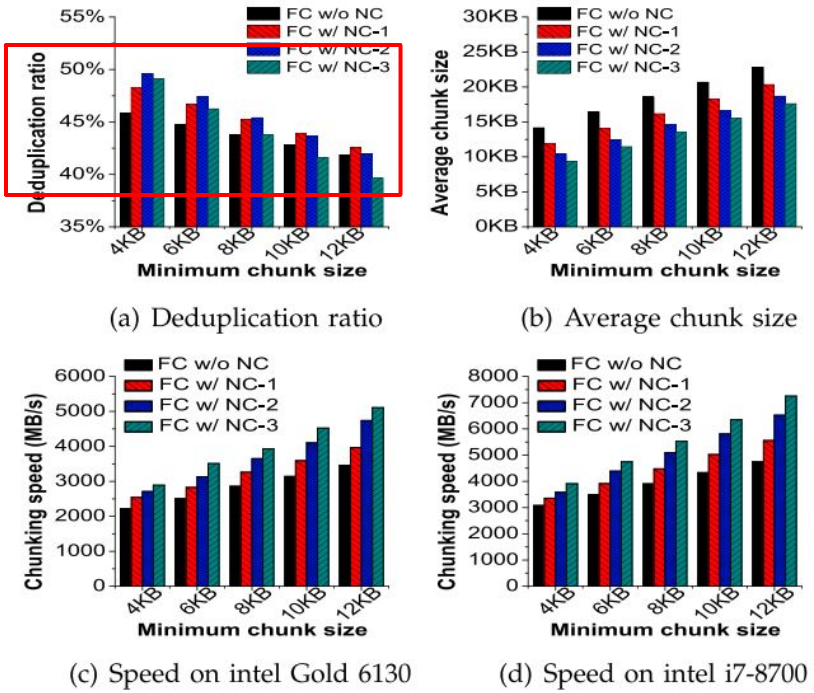
Karp-Rabin fingerprinting accommodates chunks moving

FastCDC [2] and Gear [3]



Algorithm 2. FastCDC8KB (with NC)

```
Input: data buffer, $src$; buffer length, $n$
Output: chunking breakpoint $i$
$MaskS \leftarrow 0x0000d9f003530000LL$;        // 15 '1' bits;
$MaskA \leftarrow 0x0000d93003530000LL$;        // 13 '1' bits;
$MaskL \leftarrow 0x0000d90003530000LL$;        // 11 '1' bits;
$MinSize \leftarrow 2\ KB$;   $MaxSize \leftarrow 64\ KB$;
$fp \leftarrow 0$;  $i \leftarrow MinSize$;  $NormalSize \leftarrow 8\ KB$;
if $n \le MinSize$ then
    return $n$;
if $n \ge MaxSize$ then
    $n \leftarrow MaxSize$;
else if $n \le NormalSize$ then
    $NormalSize \leftarrow n$;
for ; $i < NormalSize$; $i$++; do
    $fp = (fp << 1) + Gear[\ src[i]\ ]$;
    if ! ( $fp$ & $MaskS$ ) then
        return $i$;                  //if the masked bits are all '0';
for ; $i < n$; $i$++; do
    $fp = (fp << 1) + Gear[\ src[i]\ ]$;
    if ! ( $fp$ & $MaskL$ ) then
        return $i$;                  //if the masked bits are all '0';
return $i$;
```



(a) Deduplication ratio

(b) Average chunk size

(c) Speed on intel Gold 6130

(d) Speed on intel i7-8700

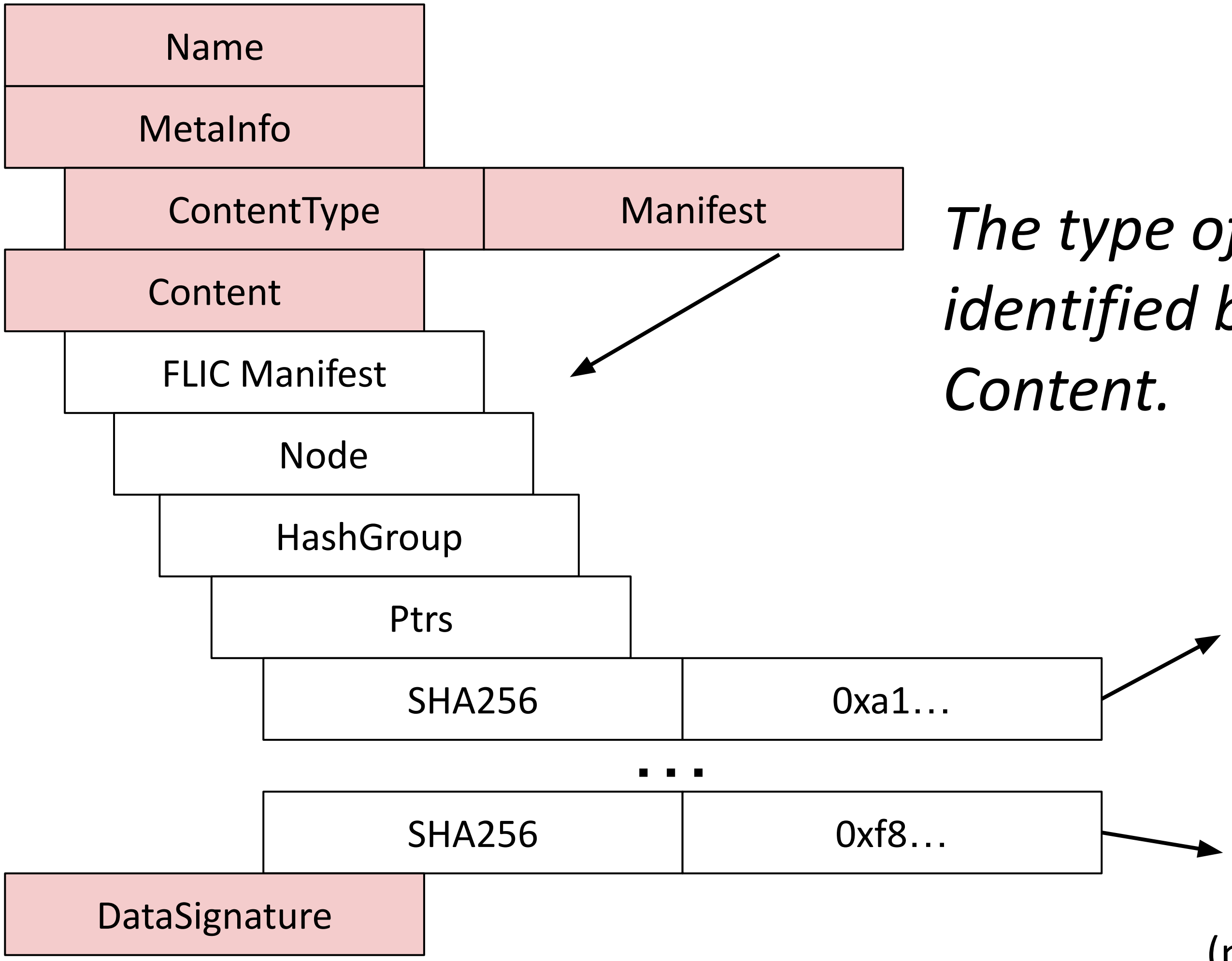
Fig. 13. Evaluation of comprehensive performance of normalized chunking with different normalization levels.

[2] Xia, Wen, Xiangyu Zou, Hong Jiang, Yukun Zhou, Chuanyi Liu, Dan Feng, Yu Hua, Yuchong Hu, and Yucheng Zhang. "The design of fast content-defined chunking for data deduplication based storage systems." IEEE Transactions on Parallel and Distributed Systems 31, no. 9 (2020): 2017-2031.
[3] Xia, W., Jiang, H., Feng, D., Tian, L., Fu, M. and Zhou, Y., 2014. Ddelta: A deduplication-inspired fast delta compression approach. Performance Evaluation, 79, pp.258-272.

FLIC Manifests

- File-Like Information Centric Manifests
  - https://datatracker.ietf.org/doc/draft-irtf-icnrg-flic/07
- Encodes a single object
- Reconstructed by concatenating Data object payloads bytes via pre-order traversal
- Follow-up work
  - Archives manifests of multiple FLICs
  - Object compression
  - Differential manifest encoding

FLIC Manifest Structure



The type of Manifest identified by opening TLV in Content.

app data

child manifest

(pre-order traversal)

Hashes To Names

- *Name Constructor* informs an Interest.
  - Segmented: Use a prefix plus a **T-V number**.
  - Prefix: A common prefix, use implicit hash.
  - Hash: CCNx Nameless Objects.
  - *Locators*, which become NDN routing hints.
  - Other Interest flags.
- A *HashGroup* specifies which NC to use.
- NCs are inherited.

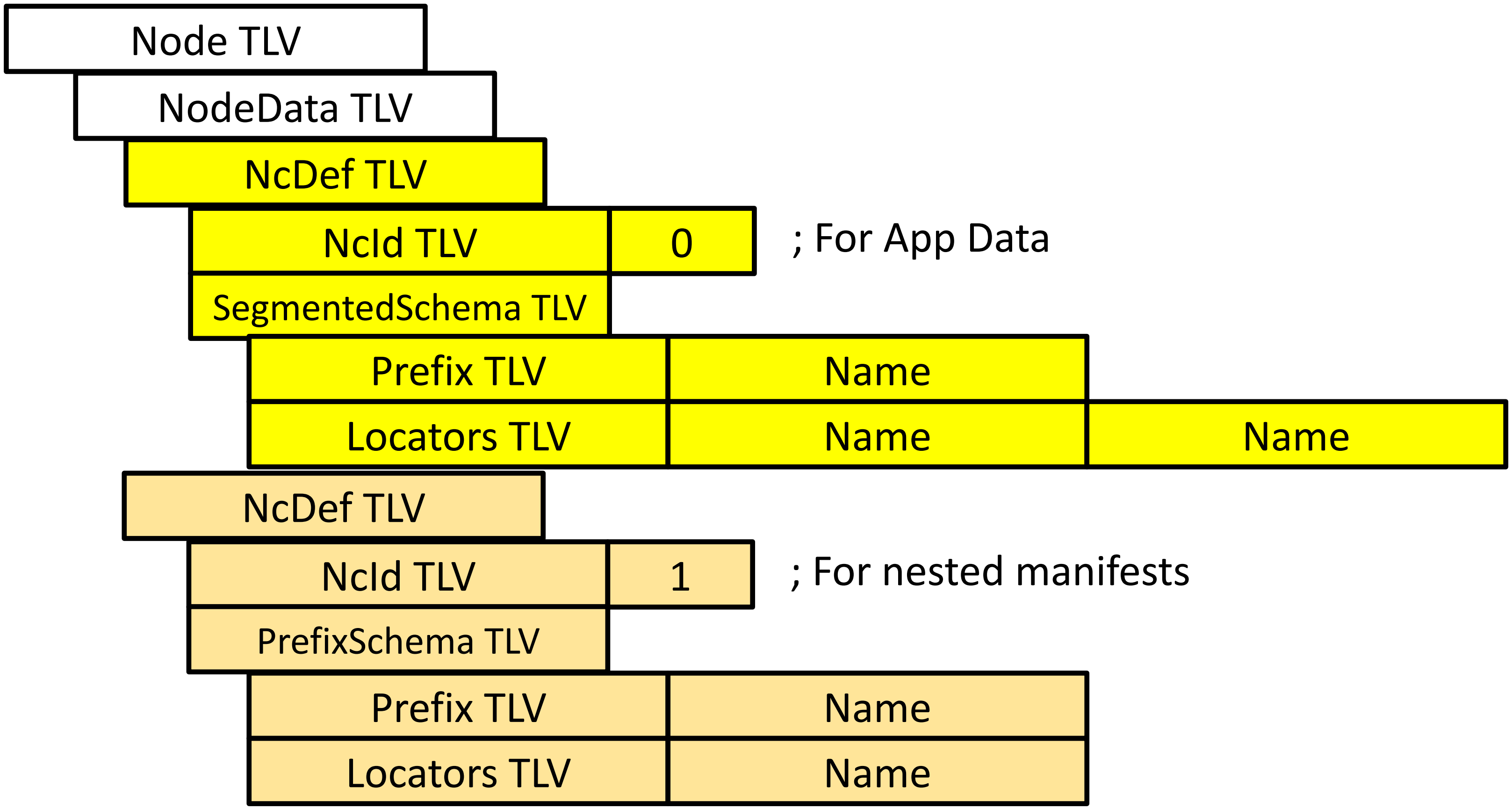
NDN Objectstore Naming

- NDN prefers that each Data object have a unique name (not counting the implicit hash).

```
common prefix + number (+hash) [Segmented]
common prefix (+hash)          [Prefix]
```

- If the manifest has one prefix and the application data has a second prefix, then one must have two (or more) HashGroups.
- Manifests use a unique *ManifestId* name segment not a Segment Number.
- One must traverse the manifest in pointer order.

Unencrypted Manifest With NodeData



Node TLV

NodeData TLV

NcDef TLV

NcId TLV 0 ; For App Data

SegmentedSchema TLV

Prefix TLV Name

Locators TLV Name Name

NcDef TLV

NcId TLV 1 ; For nested manifests

PrefixSchema TLV

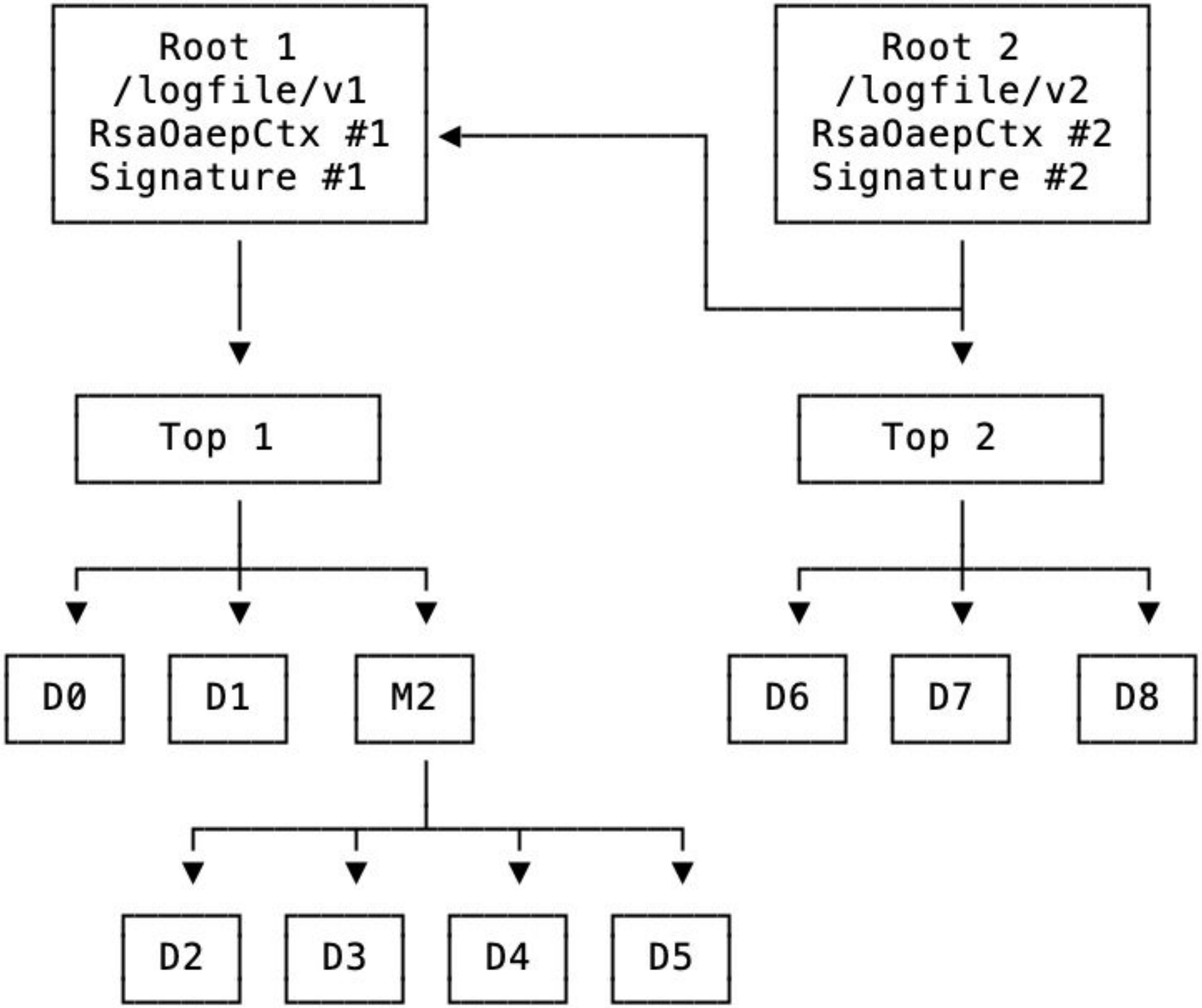
Prefix TLV Name

Locators TLV Name

```
1.   /foo/0x82=10, hash=0x0001¶
2.   /foo/0x82=20, hash=0x0002¶      ; manifests using ManifestId
3.   /foo/0x82=12, hash=0x0003¶
4.   /bar/0x32=0, hash=0x0004¶
5.   /bar/0x32=1, hash=0x0005¶       ; segmented app data
6.   /bar/0x32=2, hash=0x0006¶
```

```
Manifest
  Node
 NodeData
   NcDef NcId 1 SegmentedSchema Name '/foo' SuffixComponentType 0x82
   NcDef NcId 2 SegmentedSchema Name '/bar' SuffixComponentType 0x32
 HashGroup
   GroupData NcId 1 StartSegmentId 10
   AnnotatedPtrs
     PointerBlock
       Ptr HashValue(0x0001)
       Ptr HashValue(0x0002) SegmentIdAnnotation(20)
       Ptr HashValue(0x0003)
 HashGroup
   GroupData NcId 2 StartSegmentId 0
   Ptrs
       HashValue(0x0004)
       HashValue(0x0005)
       HashValue(0x0006)
```

Typical Manifest Structures



Other FLIC Features

- Encryption of Manifest
  - In-place encryption using AES
  - RSA-OAEP key wrapping with salt
  - KDF
- Metadata
  - Subtree size, Subtree digest
  - Annotated pointers
- Algorithm for one-pass bottom-up encoding.
- Growable, re-namable, re-signable.

Experimental Deduplicaton Results

- We applied Fastcdc data deduplication algorithm to a sampling of GNU source code distributions.
  - bison, emacs, and patch
  - (binutils and gcc show similar results)

Methodology

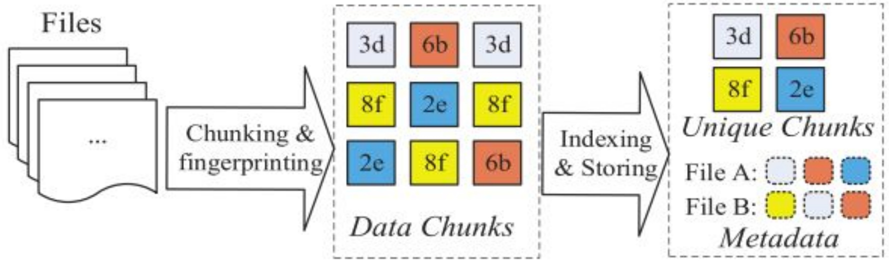


Fig. 1. General workflow of chunk-level data deduplication. [Xia et al.]

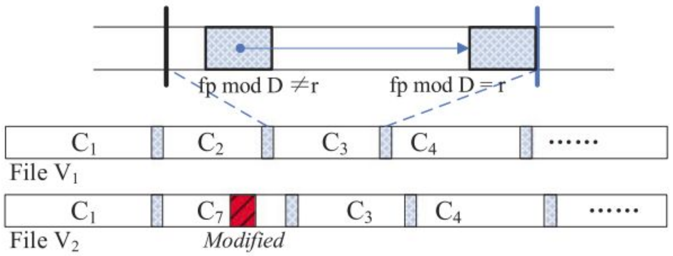


Fig. 2. The sliding window technique for the CDC algorithm. The hash value of the sliding window, *fp*, is computed via the Rabin algorithm (this is the *hashing stage* of CDC). If the lowest $log_2 D$ bits of the hash value matches a threshold value r, i.e., *fp* mod $D = r$, this offset (i.e., the current position) is marked as a chunk cut-point (this is the *hash-judging stage* of CDC). [Xia et al.]

Each dedup chunk is about 2KB - 40KB.

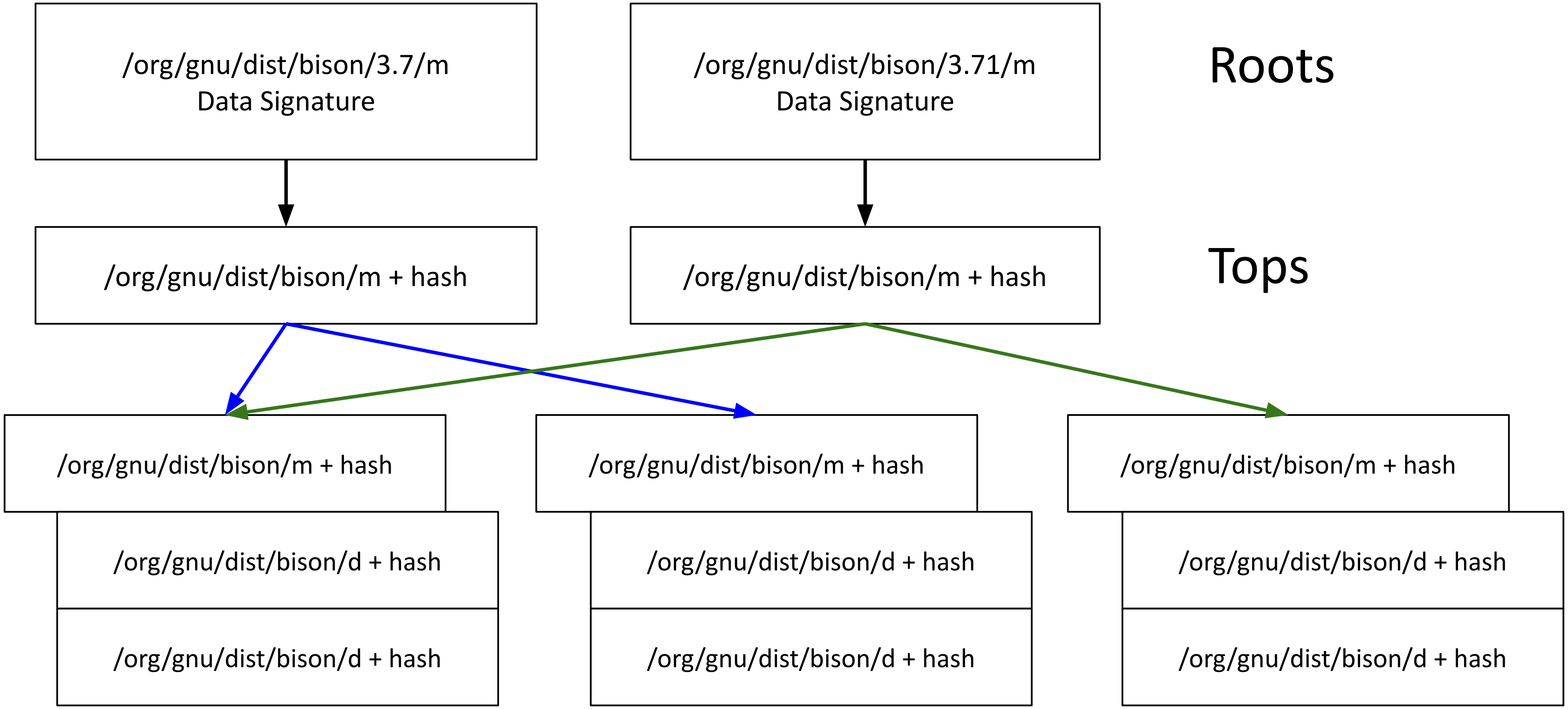
We encode each chunk as it's own mini-FLIC manifest.

Usually 1 FLIC manifest plus 7-30 data objects.

Each *tar* file has its own signed root manifest.

Naming

- The root manifest of each tar file has a name and signature.
- In HASHED
  - CCNx Nameless objects
- In PREFIX
  - /com/objectstore/gnu/d/ (+ hash)
  - /com/objectstore/gnu/m/ (+ hash)
  - differ by implicit hash.



/org/gnu/dist/bison/3.7/m
Data Signature

/org/gnu/dist/bison/3.71/m
Data Signature

Roots

/org/gnu/dist/bison/m + hash

/org/gnu/dist/bison/m + hash

Tops

/org/gnu/dist/bison/m + hash

/org/gnu/dist/bison/d + hash

/org/gnu/dist/bison/d + hash

/org/gnu/dist/bison/m + hash

/org/gnu/dist/bison/d + hash

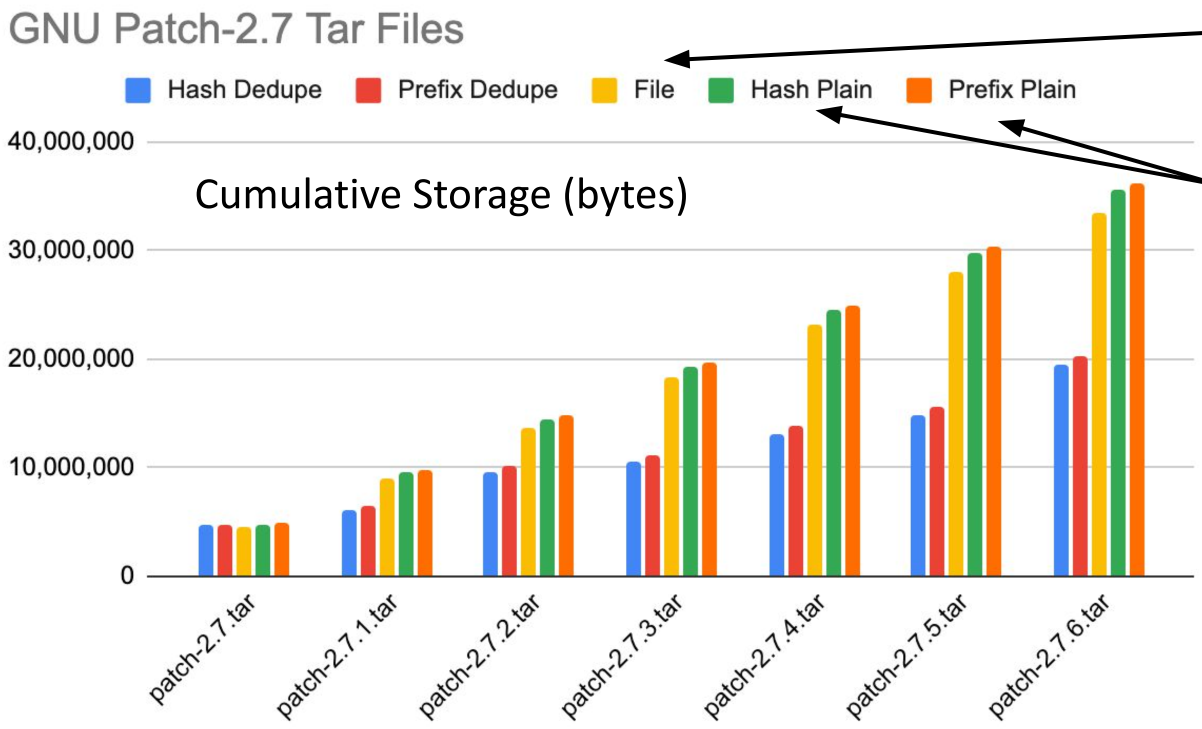
/org/gnu/dist/bison/d + hash

/org/gnu/dist/bison/m + hash

/org/gnu/dist/bison/d + hash

/org/gnu/dist/bison/d + hash

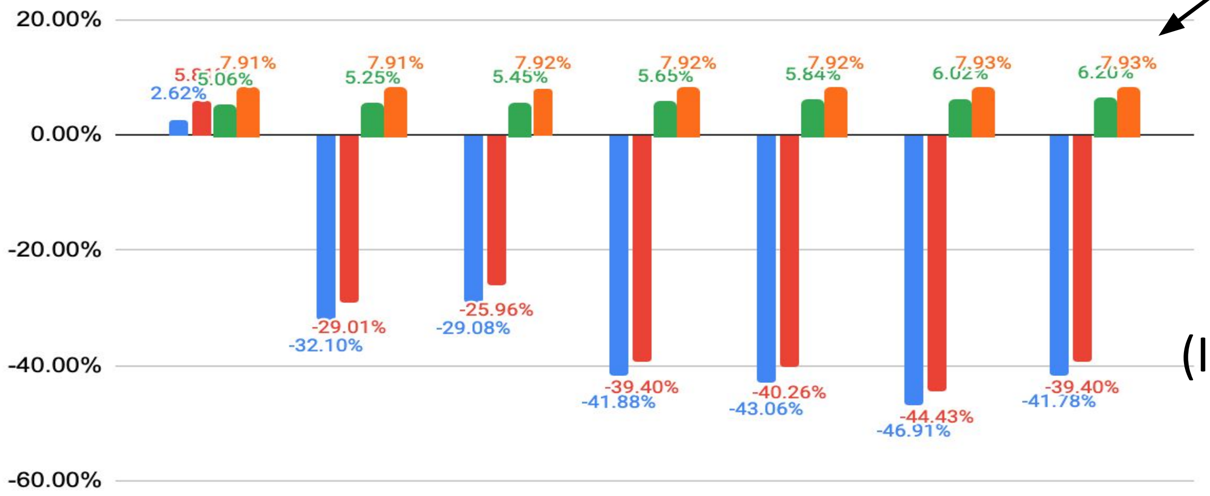
Signed tar file manifest
points to top manifest
points to chunk manifests



Cumulative Storage (bytes)

Bytes of tar file

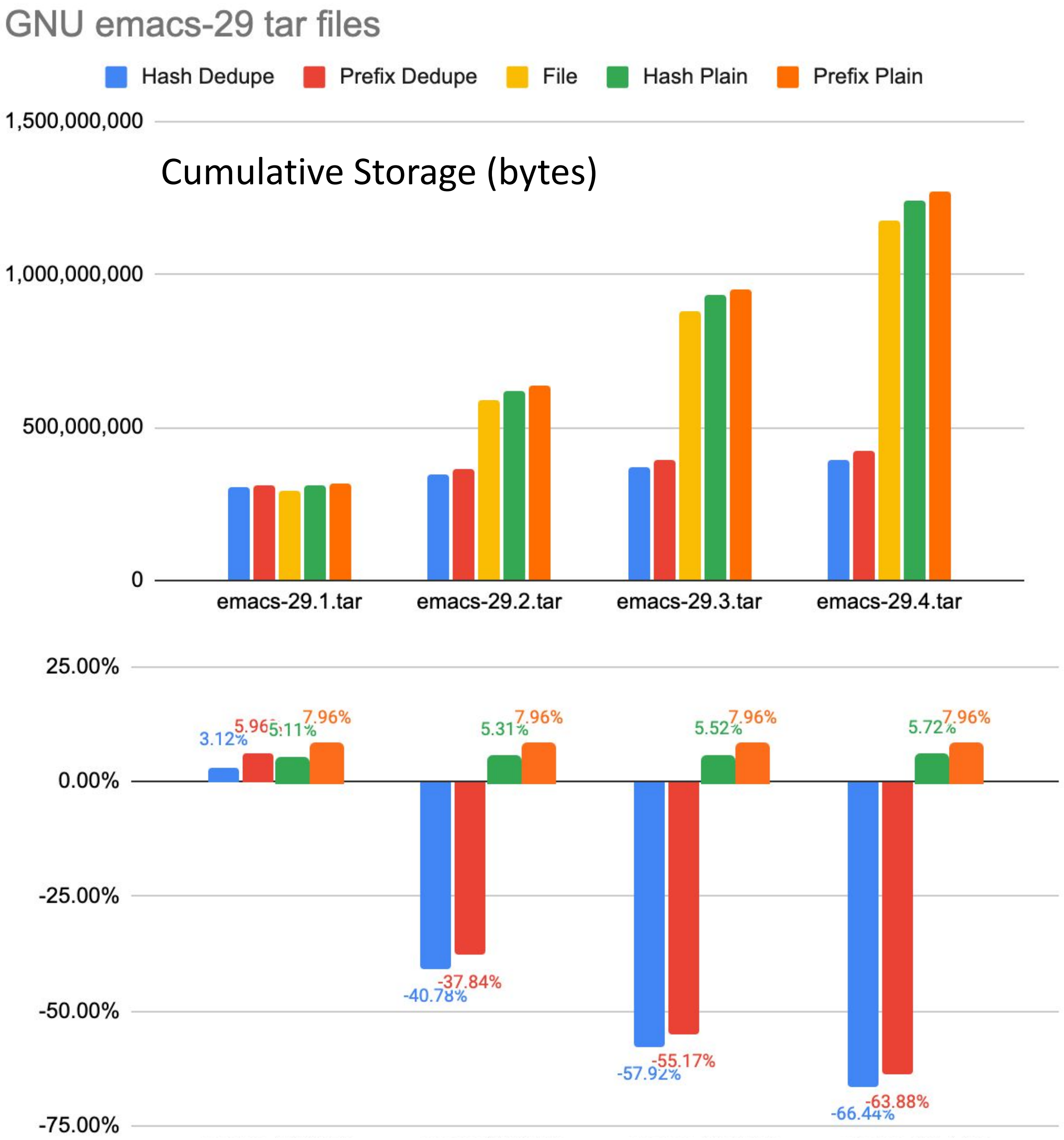
Encoding without Dedup



Packetization overhead without Dedup

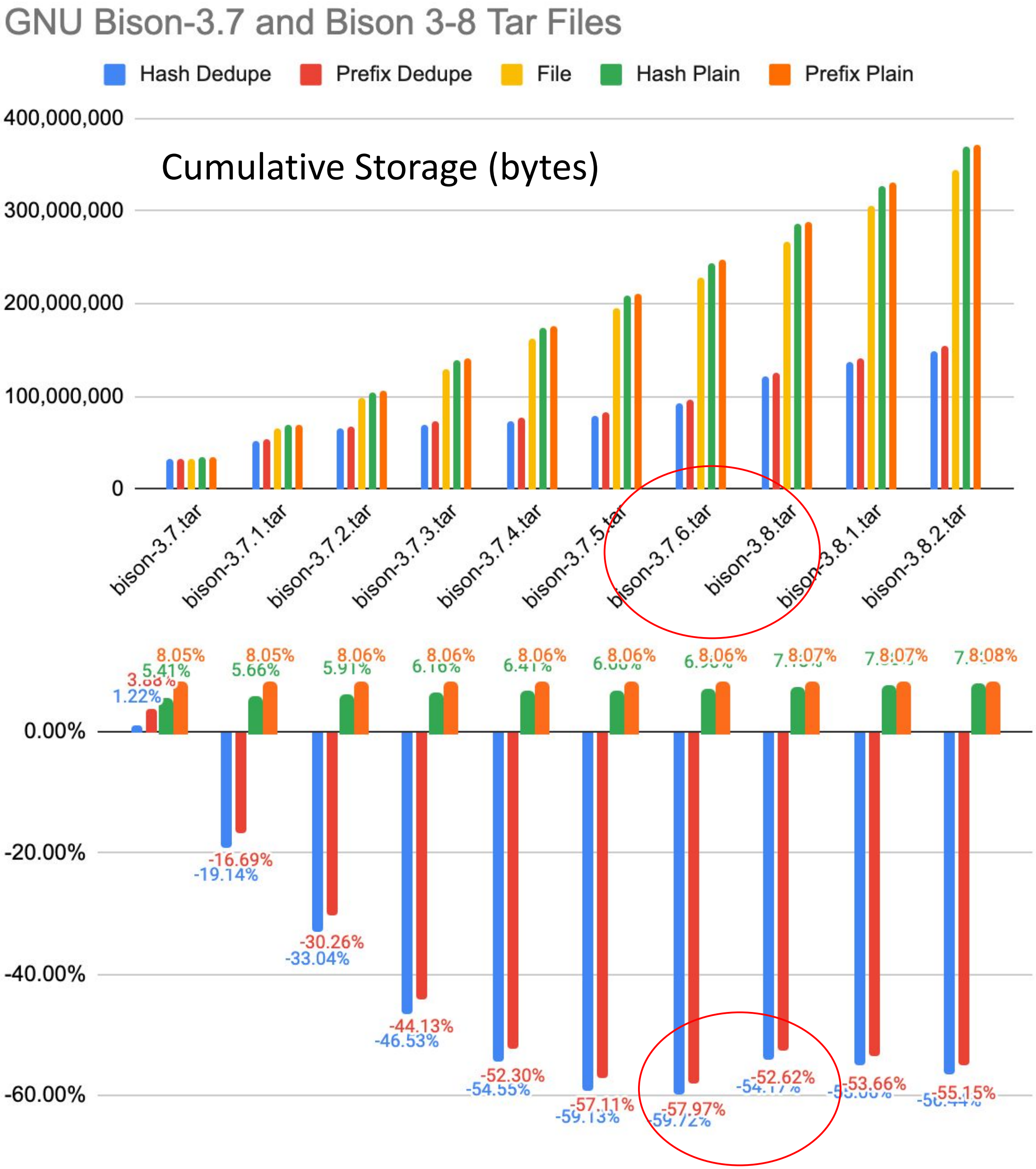
0 = file size

% Savings =
(ICN_size - file_size) / file_size



GNU emacs-29 tar files

Cumulative Storage (bytes)



Transition from 3.7 to 3.8

Savings takes a hit

Experiment Results

- Encoding a series of releases within a project has significant savings.
- Savings
  - 30% to over 60% compared to file size.
  - Extra 5%-8% over ICN packetization.
  - About 2% - 3% Nameless over Prefix.

Unique Naming What-if

- tar root manifest
  - /objstore/gnu/patch/2.7.1/m
- tar internal manifests
  - /objstore/gnu/patch/2.7.1/m/<ManifestId>
- Chunk manifests
  - /objstore/gnu/patch/m/<?>
- Chunk data
  - /objstore/gnu/patch/d/<?>/<seqnum>

Conclusion

- The FLIC -07 draft should be ready for final review.
- We only have experience with CCNx and cefore forwarder.
- We suggested NDN encodings and would like to get NDN implementation experience.
- Upcoming IRTF drafts
  - Archives of multiple FLICs
  - Compression
  - Manifest diffs for updates