File-Like ICN Collections (FLIC)

draft-irtf-icnrg-flic-07

IETF 122, Bangkok

Marc Mosko

Dave Oran

Network Systems Research & Design

Outline

- Updates since -06
- Implementation Status
- Experimental Results with Deduplication

Overview of Updates

- Major reorganization for readability and conciseness.
- Updates to RSA key wrapping based on implementation experience.
- IANA considerations updated and complete.
- Significant updates to Security Considerations.

Table of Contents

Table of Contents

Clarifications on Structure

- No unique root. Manifests can point to other manifests or there may be multiple signed roots (e.g. different names or updated keys).
- A FLIC manifest is a DAG, not a tree.

A RECOMMENDED FLIC structure is to use a distinguished root manifest with exactly one pointer to the top manifest of a regular n-ary graph. The root manifest is signed by the publisher and contains links to certificates or public keys. It also specifies the Name Constructors used in the manifest. If using RSA-OAEP key wrapping, it contains the wrapped key. These one-time data structures take significant space, so off-loading them to the root manifest is needed to use a regular n-ary graph for the rest of the manifest. In CCNx, the root manifest is the only one in the graph that needs to have a name if using nameless objects.

Clarification about non-tree FLICs

- A manifest branch may have multiple parents.
- Each parent may have its own NcDef for an NcID.
- The consumer must track the current NcDef as it descends into a branch.
- This is likely moot, as the client should have retrieved all the hashes the first time and they are the exact same objects as in subsequent descents.

ABNF Updates

- Used the IANA symbolic type names
- Main updates are to RSA and KDF

```
AEADCtx = TYPE LENGTH AEADData
AEADData = KeyNum AEADNonce Mode
KeyNum = TYPE LENGTH INTEGER
AEADNonce = TYPE LENGTH 1*OCTET
AEADMode = TYPE LENGTH (AEAD_AES_128_GCM / AEAD_AES_256_GCM /
           AEAD_AES_128_CCM / AEAD_AES_128_CCM)

RsaKemCtx = 2 LENGTH RsaKemData
RsaKemData = KeyId AEADNonce AEADMode WrappedKey LocatorPrefix
KeyId = TYPE LENGTH HashValue; ID of Key Encryption Key
WrappedKey = TYPE LENGTH 4*OCTET 1*OCTET  ; 4-byte salt plus AES key
LocatorPrefix = TYPE LENGTH Link
```

```
AEADCtx = T_AEAD_CTX LENGTH AEADData
AEADData = KeyNum AEADNonce AEADMode [KDFData]
KeyNum = T_KEYNUM LENGTH INTEGER
AEADNonce = T_NONCE LENGTH 1*OCTET
AEADMode = T_AEADMode LENGTH (AEAD_AES_128_GCM / AEAD_AES_256_GCM /
           AEAD_AES_128_CCM / AEAD_AES_128_CCM)
           ; RFC5116 definitions
KDFData = T_KDF_DATA LENGTH KDFAlg [KDFInfo]
KDFAlg = T_KDF_ALG LENGTH INTEGER
    ; IANA "HPKE KDF Identifiers" [RFC9180]
KDFInfo = T_KDF_INFO LENGTH 1*OCTET
    ; Passed to the KDF Info

RsaOaepCtx = T_RSAOAEP_CTX LENGTH RsaOaepData
RsaOaepData = AEADData [RsaOaepWrapper]
RsaOaepWrapper = KeyId [KeyLink] HashAlg WrappedKey
    ; KeyId as pre RFC8609 for CCNx
    ; KeyLink as pre RFC8609 for CCNx
HashAlg = T_HASH_ALG LENGTH alg_number
    ; alg_number from IANA "CCNx Hash Function Types"
WrappedKey = T_WRAPPED_KEY LENGTH RsaOaepEnc{4OCTET 1*OCTET}
    ; Encrypted 4-byte salt plus AES key
```

KDF updates

- Some were in -06.
- All encryption modes allow a KDFData.
- KDFData is a KDFAlg [RFC9180] and the Content-object specific KDFInfo to used in derived key.

```
KDFData = T_KDF_DATA LENGTH KDFAlg [KDFInfo]
KDFAlg = T_KDF_ALG LENGTH INTEGER
    ; IANA "HPKE KDF Identifiers" [RFC9180]
KDFInfo = T_KDF_INFO LENGTH 1*OCTET
    ; Passed to the KDF Info
```

RSA Updates

- The RSA-OAEP mode wraps an AES key inside a manifest.
- Only needed once in a FLIC graph.
- AEADData may include a KDF.
- WrappedKey includes recommended 4-byte salt.
- This usage of RSA-OAEP is intended to be compliant with KTS-OAEP [NIST-800-56Br2] for non-shared RSA keys.

```
RsaOaepCtx = T_RSAOAEP_CTX LENGTH RsaOaepData
RsaOaepData = AEADData [RsaOaepWrapper]
RsaOaepWrapper = KeyId [KeyLink] HashAlg Wrapped Key
    ; KeyId as pre RFC8609 for CCNx
    ; KeyLink as pre RFC8609 for CCNx
HashAlg = T_HASH_ALG LENGTH alg_number
    ; alg_number from IANA "CCNx Hash Function Types"
WrappedKey = T_WRAPPED_KEY LENGTH RsaOaepEnc{4OCTET 1*OCTET}
    ; Encrypted 4-byte salt plus AES key
```

Padding

- FLIC explicitly allows the use of the CCNX 1.0 T_PAD field [RFC8609].
- Padding may obfuscate encrypted manifests and reduce information leakage from byte size patterns.
- The draft recommends padding in 128-byte blocks (upto the maximum MTU).

Security Considerations

- Symmetric key lifespan and scope
  - RSA-OAEP encapsulated symmetric keys are implicitly scoped to the current manifest.
  - General symmetric keys using a KeyNum are agreed out-of-band by publisher and consumer.
  - Recommend caching symmetric keys based on their scope, e.g.
    - (Publisher KeyId, KeyNum) or
    - (Publisher KeyId, Name Prefix, KeyNum)

Implementation Status

- ccnpy [1] now implements the whole spec.
- ccnpy now on pipy [3].
- All encryption modes tested in implementation.
- We have still not finished testing with cefore [2], but that is next up. This is mostly an issue of needing to use named objects. This drove a big update to ccnpy to fully support all the name constructors.

[1] https://github.com/mmosko/ccnpy
[2] https://github.com/cefore/
[3] https://pypi.org/project/ccnpy/

Experimental Deduplicaton Results

- We applied a recent data deduplication algorithm (Fastcdc [1]) to a sampling of GNU source code distributions.
  - binutils, bison, emacs, and patch
- We encoded each Fastcdc segment of a tar file with HASHED (nameless object) CCNx.

[1] 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. https://ranger.uta.edu/~jiang/publication/Journals/2020/2020-IEEE-TPDS(Wen%20Xia).pdf

Methodology

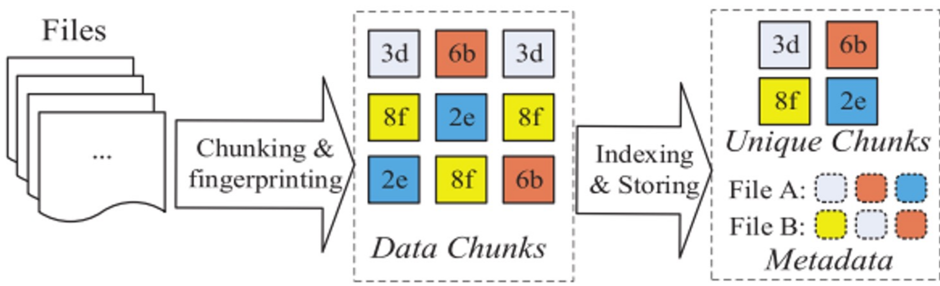


Fig. 1. General workflow of chunk-level data deduplication. [Xia et al.]

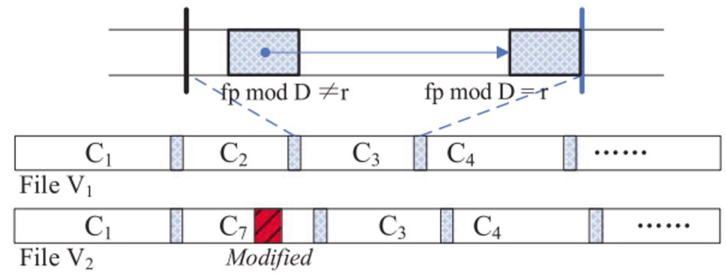
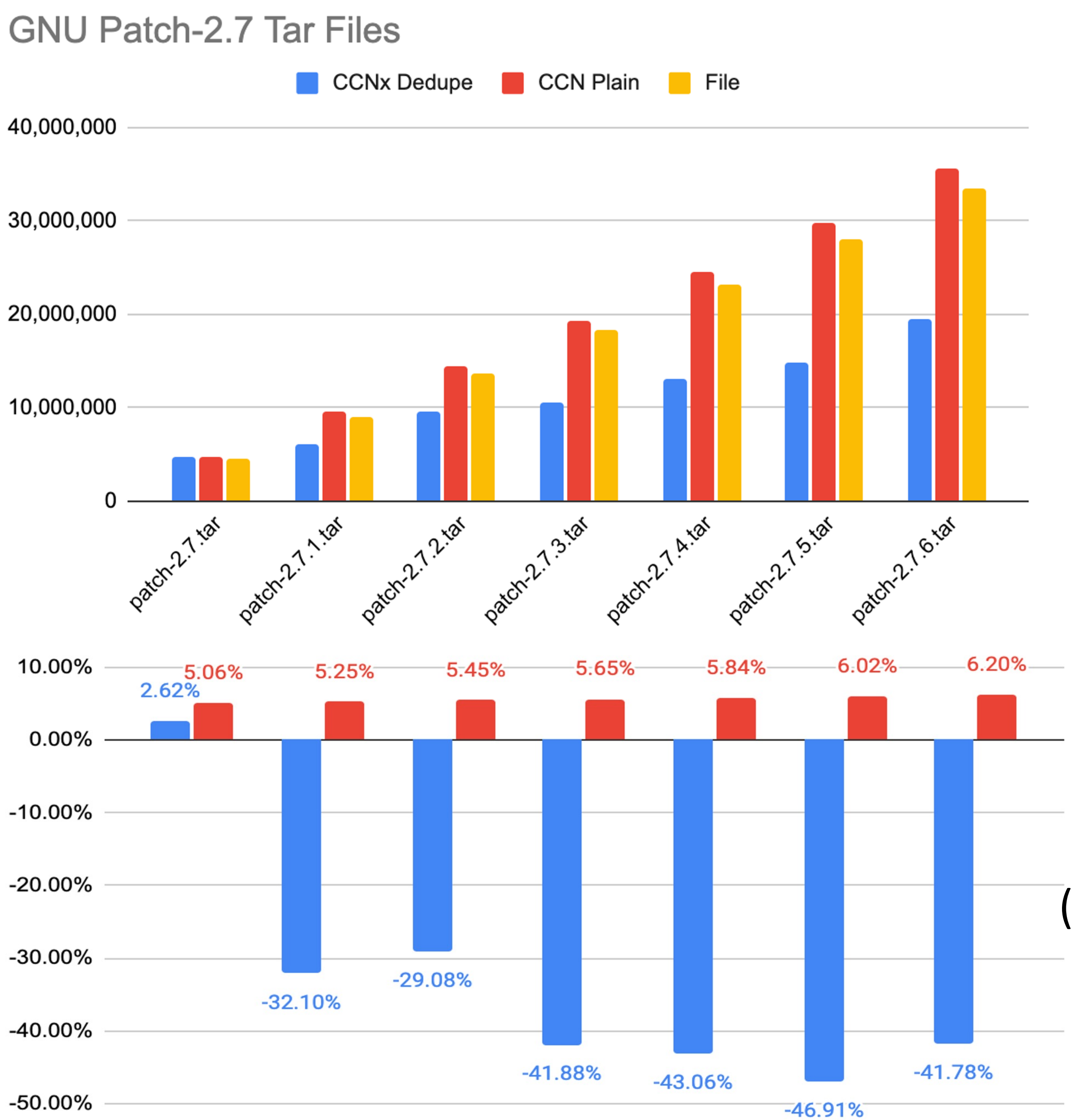
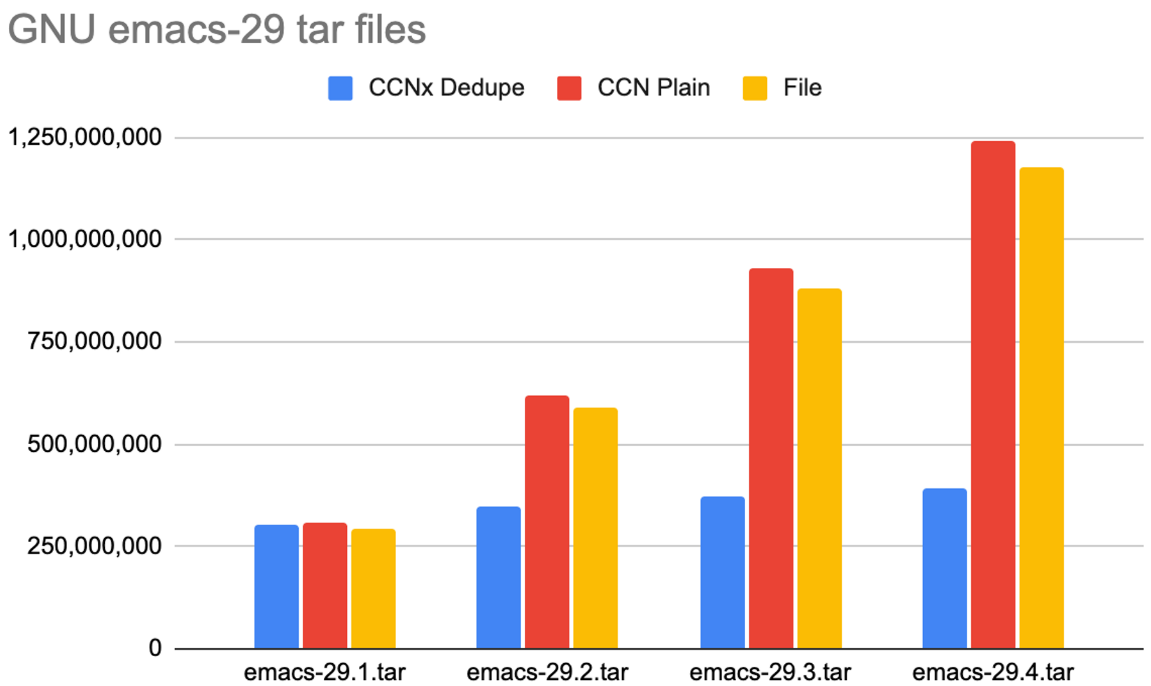


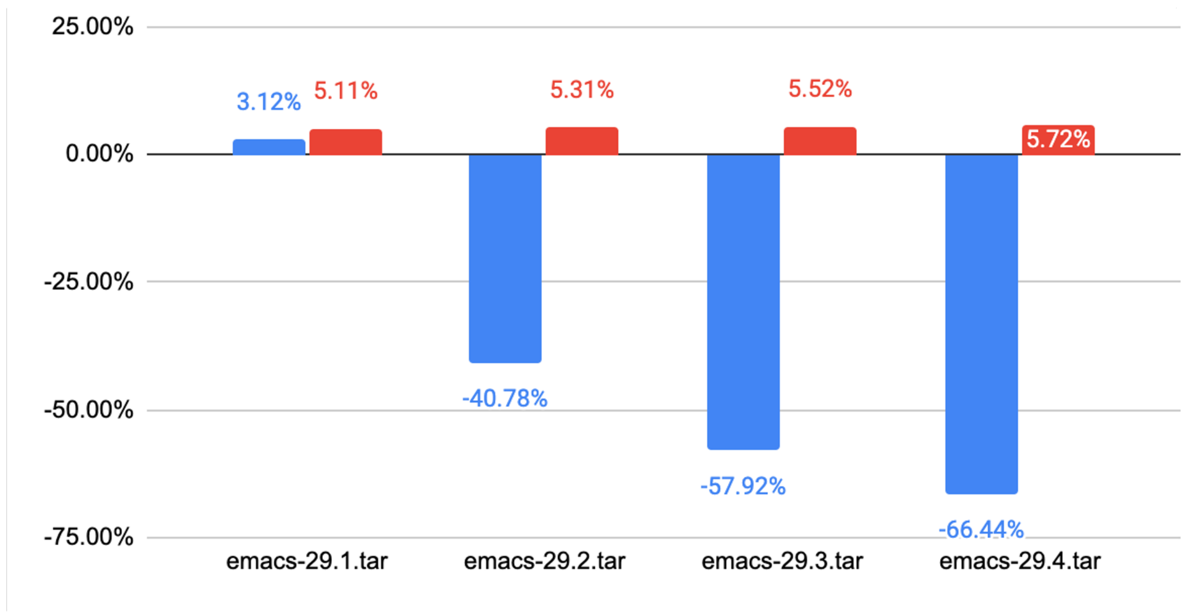
Fig. 2. The sliding window technique for the CDC algorithm. The hash value of the sliding window, *fp*, is computed via the Rabin algorithm (this is the *hashing stage* of CDC). If the lowest $log_2 D$ bits of the hash value matches a threshold value r, i.e., *fp* mod $D = r$, this offset (i.e., the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

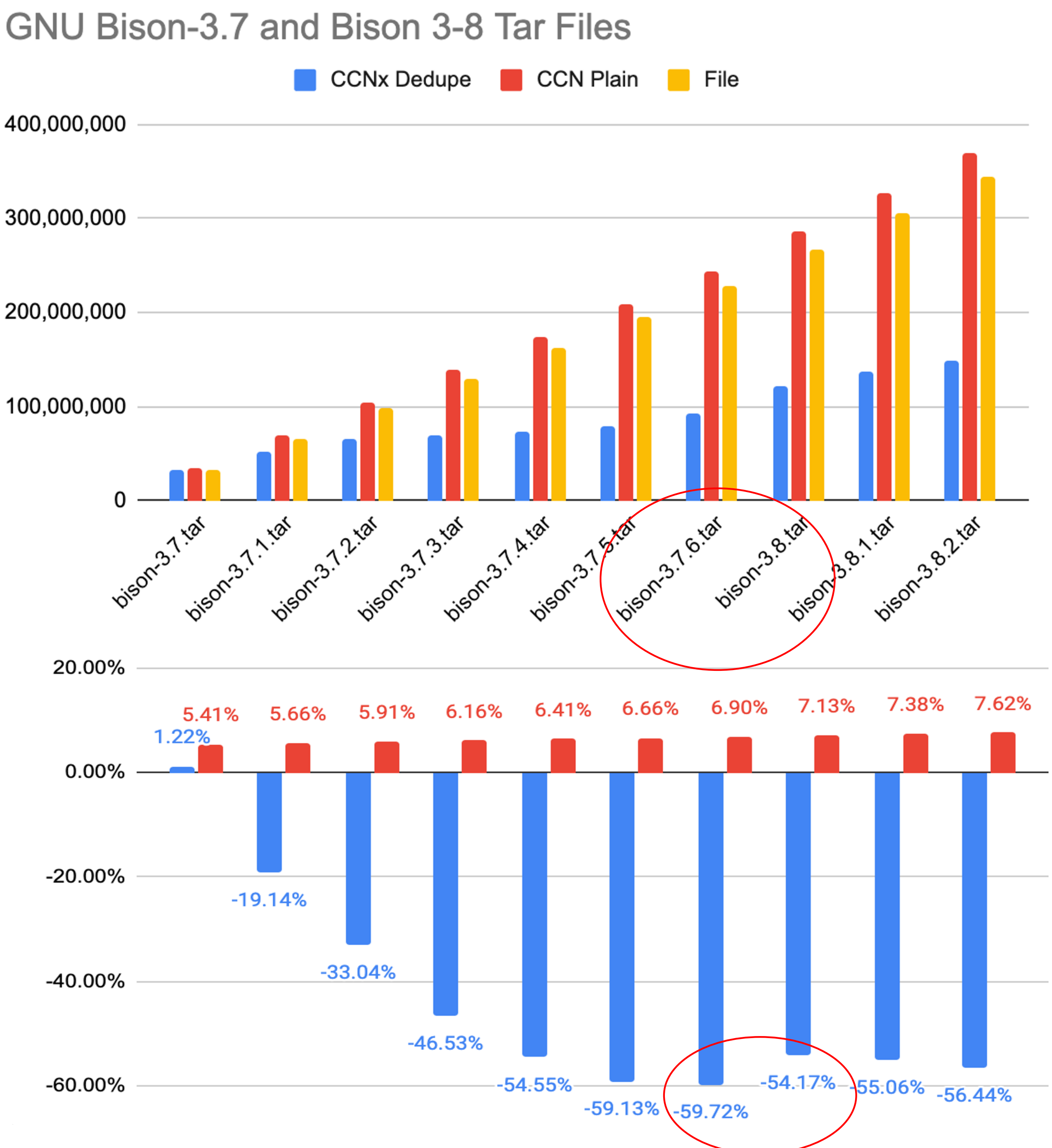
Each dedup chunk is about 2KB - 40KB.
We encode each chunk as it's own mini-FLIC nameless manifest. This is 1 FLIC manifest plus 7-30 data objects (1500 bytes).
Each *tar* file has its own signed root manifest.



% Savings =
(ccnx_size - file_size) / file_size


GNU emacs-29 tar files
CCNx Dedupe
CCN Plain
File
1,250,000,000
1,000,000,000
750,000,000
500,000,000
250,000,000
0
emacs-29.1.tar
emacs-29.2.tar
emacs-29.3.tar
emacs-29.4.tar


25.00%
0.00%
-25.00%
-50.00%
-75.00%
3.12%
5.11%
5.31%
5.52%
5.72%
-40.78%
-57.92%
-66.44%
emacs-29.1.tar
emacs-29.2.tar
emacs-29.3.tar
emacs-29.4.tar



Transition from 3.7 to 3.8

Savings takes a hit

Experiment Results

- Encoding a series of releases within a project has significant size savings.
- Savings of 30% - over 60% compared to the file size. it is even more compared to the non-dedupe CCNx encoding.
  – For example, emacs-29.1 to 29.4
    - the cumulative file size is 1,121.5 MB
    - The cumulative CCNx dedup is 376.3 MB.
- Using named objects would reduce savings.

Conclusion

- The -07 draft should be ready for final review.
- We do not expect any significant changes based on cefore interop experiments.

Q&A