Revised draft:
"File-Like ICN Collection (FLIC)"

ICNRG interim meeting, UCL
March 18, 2018

<christian.tschudin@unibas.ch>

In this presentation

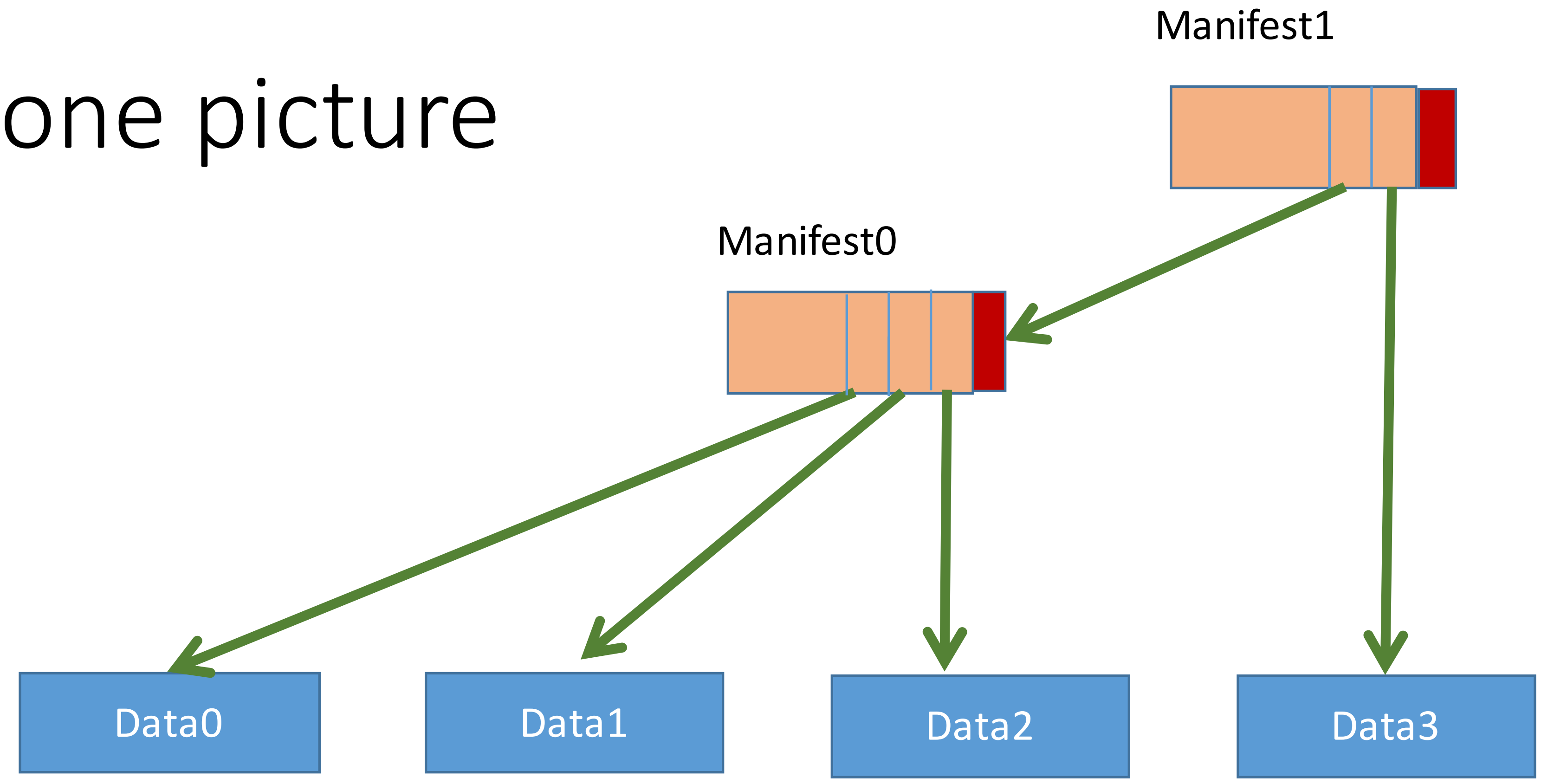
a) FLIC in one picture

b) Different motivations why people may use FLIC:
packet trains *vs* huge data collections

c) Four motivations for a revised draft:

1. Encoding strategies
2. Separate DEK and SEK (**D**ata vs **S**tructure **E**ncryption **K**ey)
3. Implementation complexity of FLIC **en**coding
4. Implementation complexity of FLIC **de**coding

a) FLIC in one picture



- Large data is cut into chunks, persisted independently
- Manifest packets contain: metadata, index table(s), signature. Are also persisted.
- Index table contains “hash pointers” (incl intrinsic name of data or manifest chunk)
- Manifests as an alternative to *chunk naming*, even have “name-less objects”

b) Differences in why people may use FLIC

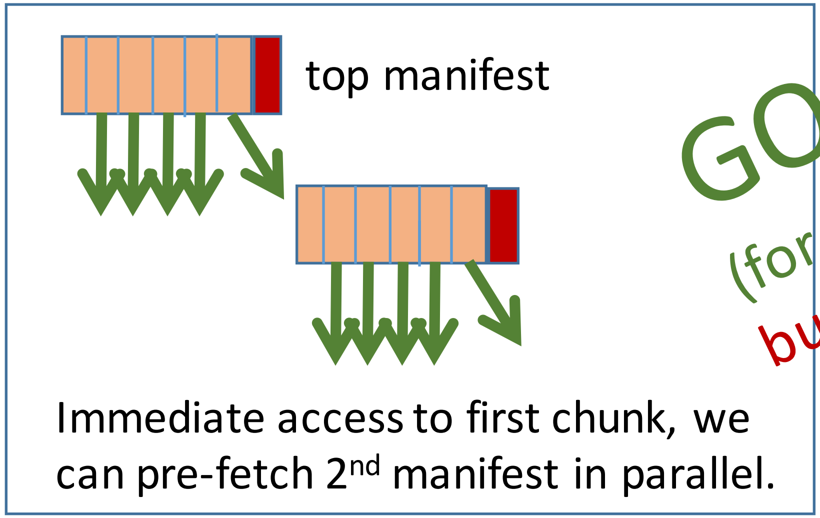
- Transient use:
  - send several signature-less data chunks
  - manifest with a single signature covers all chunks, less run-time effort

- Permanent use:
  - ICN as a global block storage service (PDU == block)
  - large data collections then mapped to ICN blocks
  - collection examples:
    a file, DB table, append-only log, other hash-chained data structures

c.1) Revision b/c of encoding strategies

Seminar students pointed out that B-trees are better (than a transport-optimized manifest encoding) for storing video: we want to "seek"

- "transport-optimized" means:
  A manifest has multiple data pointers to fetch before one has to fetch another manifest



Immediate access to first chunk, we can pre-fetch 2nd manifest in parallel.

GOOD

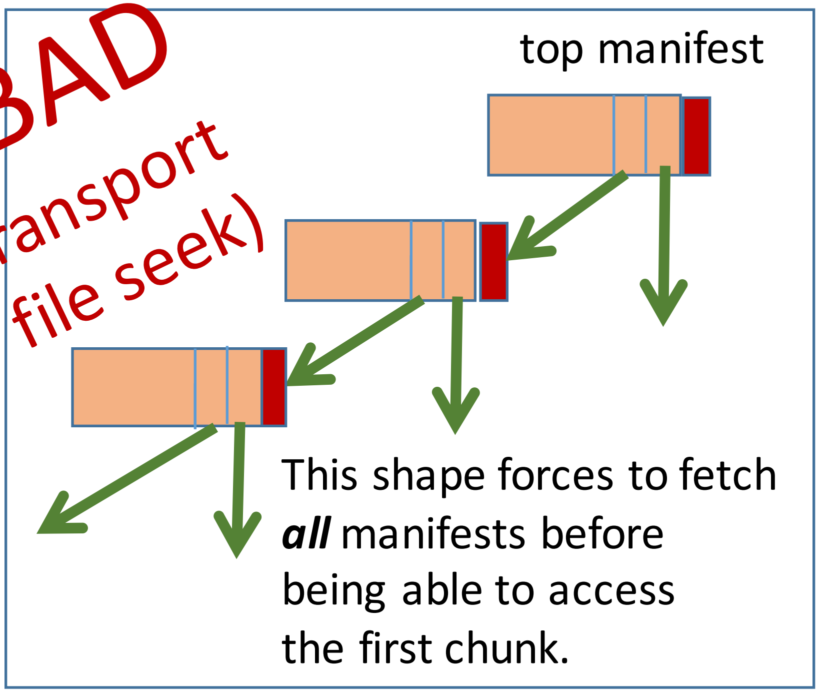
(for transport,

but NOT for file)

BAD

(for transport

and file seek)



This shape forces to fetch ***all*** manifests before being able to access the first chunk.

Goal for draft: better describe trade-offs and preferences for tree shapes.

c.2) Separate DEK and SEK

End-to-end encryption and access control in ICN:

- source encrypts *content* with a DEK (data encryption key)
- access is controlled by selectively handing out the DEK

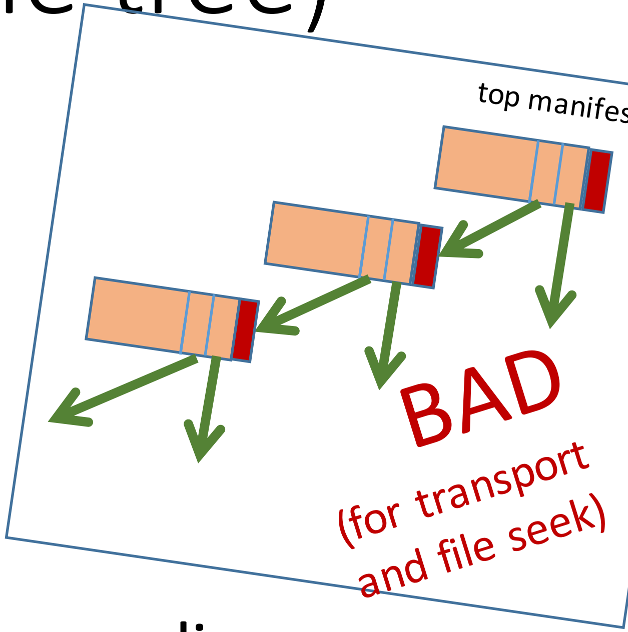
Should manifest packets be encrypted, too?

→Use a different SEK (*structure* encryption key)

- Permits to delegate operations on the tree to third parties, edge nodes, without exposing the (DEK-protected) data

Goal for draft: introduce SEK, perhaps also “ptr to encrypted manifest”

c.3) Encoding complexity (shape of the tree)



From PyCN-lite: easy to write a *default* tree encoder

But encoder needs data structure awareness:

- **“log file” case: append-only of text lines**

  bad: append a new manifest to old manifest, plus link to new line

  good: transport-friendly **reshaping** of the previous tree

  (WITHOUT link to the previous manifest!)

Goal for draft: better discussion of encoding strategies, data structure awareness

c.4) Decoding complexity (metadata)

How useful are the proposed FLIC metadata fields? Ex: pos and # of bytes in a sub-tree

Became wary when writing FLIC decoders (= tree traversers): packets control my effort.
- accidentally wrong metadata (byte position for seeking MUST be accurate)
- deliberate wrong metadata

“Be conservative in what you send, be liberal in what you accept”

a. Drop most of metadata?
   (Manifest consumer has to verify a lot, has to guard against DoS attacks - at the end the SW is perhaps not better off, compared to not having this information at all.)

b. Introduce "attestation" of manifest content, by third parties?

Goal for draft: eliminate all metadata fields having usefulness concerns.