Stateless Forwarding in Information Centric Networking

AYTAC AZGIN AND RAVISHANKAR RAVINDRAN

{aytac.azgin,ravi.ravindran}@huawei.com

Stateful Forwarding

Default CCN/NDN operation uses **stateful forwarding**

- Pending Interest Tables (PITs) store information on received requests:
  - Content name
  - Incoming/outgoing interfaces → Tell how to forward Data pkts
  - Nonces (if implemented) → Identify duplicate/new requests
  - Timeout values → Limit storage overhead by purging entries for failed requests

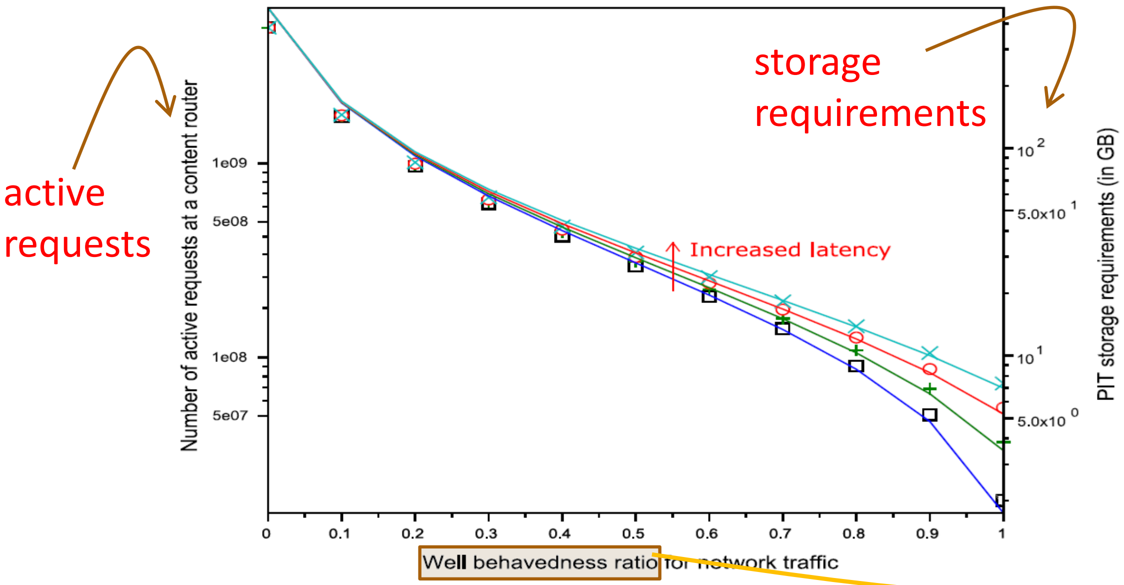
Stateful forwarding has multiple purposes

- Aggregate incoming requests → e.g., same name, different incoming interface and nonce values
- Prevent attacks targeting a content name → as requests targeting the same name are suppressed at the edge
- Create breadcrumbs for the Data packets → received Data packets are checked with PIT entries for a match

Motivation for Stateless Forwarding

What are the main concerns for stateful forwarding?

- Aggregation is limited to edges → `not necessary everywhere`
- Shown to not fully prevent attacks → `may use other means to provide security`
- Introduces additional overhead: **storage** and **processing**



Ratio of Interests that return a Data packet

Problem: Interests with no Data return
What happens: Entries are stored within PIT, until timeout (~4s)

Observations:
(1) Increased memory requirements to represent worst-case scenario
(2) Increased latency to access entries

What remains is the breadcrumb advantage

- replicated using stateless forwarding, **using in-packet filters**

Design Objectives for Stateless Forwarding

We can summarize the basic design objectives as follows:

- Limit forwarding state to domain-based or globally shared forwarding strategy and remove per-request dependency
- Reduce processing and storage requirements at ICN routers without relaxing the security considerations
- Allow for easier transition towards enabling future networking architectures (for instance, ICN over P4)

These objectives can be achieved using in-packet filters, which carry reverse-path information, with vertically-integrated or horizontally-integrated designs

$$\begin{array}{r} \text{Received} \\ + \ \text{Local} \\ \hline \text{Sent} \end{array}$$

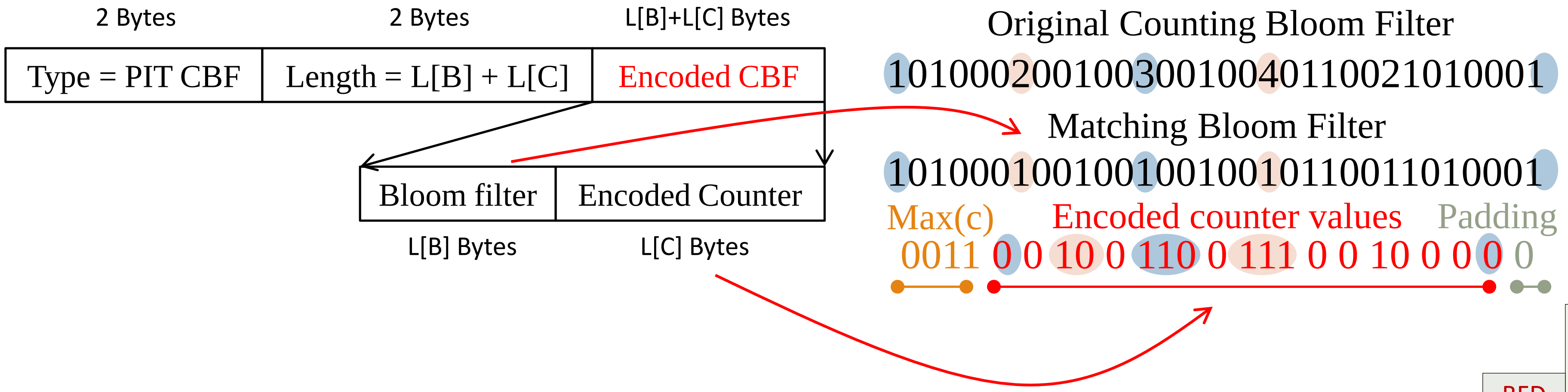
$$F(\ \text{Received}\ \text{Local}\ ) = \text{Sent}$$

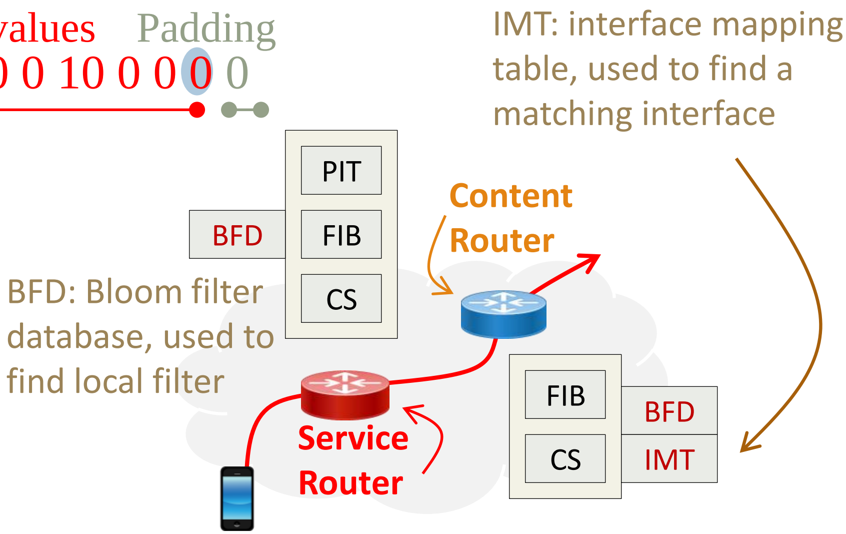
F: Some transformation function

Vertical Design Choice: Counting Bloom Filter

Classic Bloom filter is not a desirable option due to no modification along reverse-path and false positives, which can introduce significant overhead

Filter header consists of **constant sized Bloom filter component** and **variable-sized encoded counter**





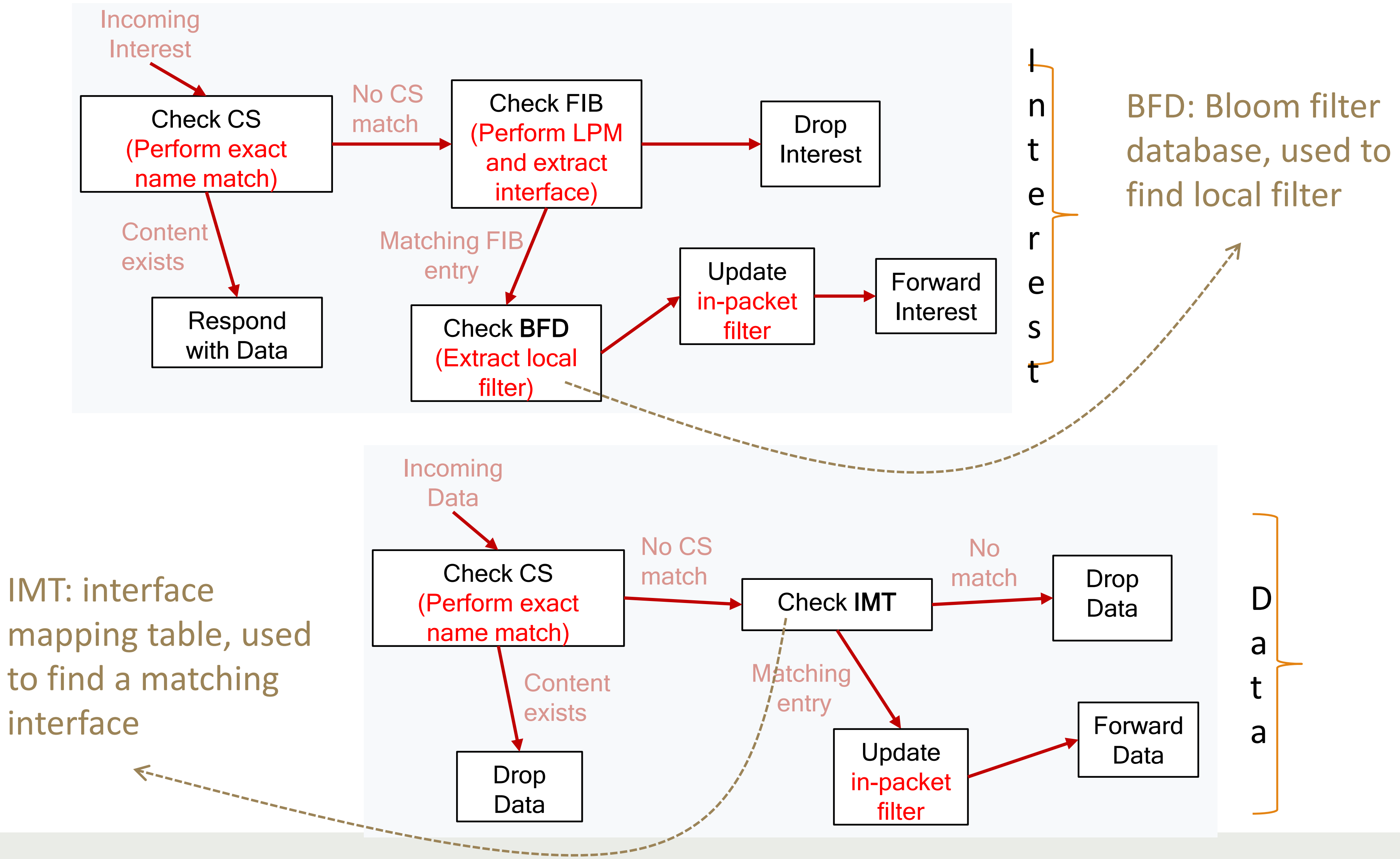
Use of CBF allows update along reverse path

ICN routers perform look-up, update and forward operations

Implement CBF processor for static operations

"pit/LESS: Stateless Forwarding in Content Centric Networks", A. Azgin, R. Ravindran, and G.Q. Wang, IEEE Globecom, 2016.

Vertical Design Choice: Packet Processing Flow



Horizontal Design Choice: Interleaved Labels

Design objective is to remove dependency of in-packet filter on Bloom filters

- provide same advantages as a Bloom filter based design while **avoiding false positives** with minimal added complexity

Utilize **integrated multi-label forwarding** to address the complexity of more advanced BF-based designs, while increasing the robustness in terms of security

Each ICN router implements a **Local Transform Filter (LTF)**

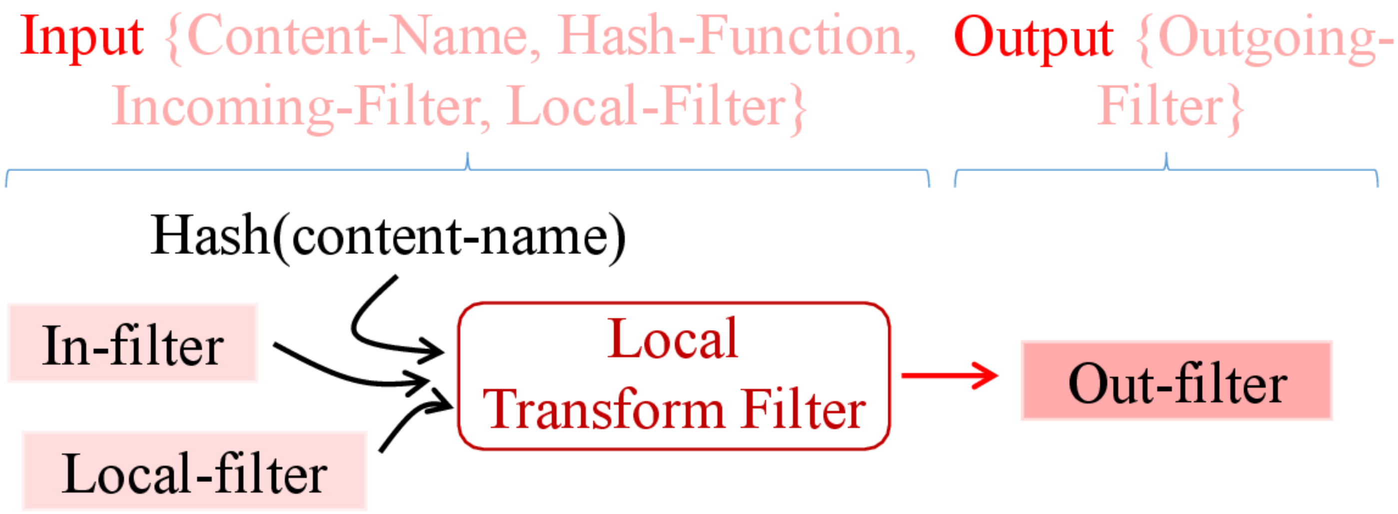
- modify in-packet filters for received Interest and Data packets

Each ICN router also carries a **Filter Database (FDB)**

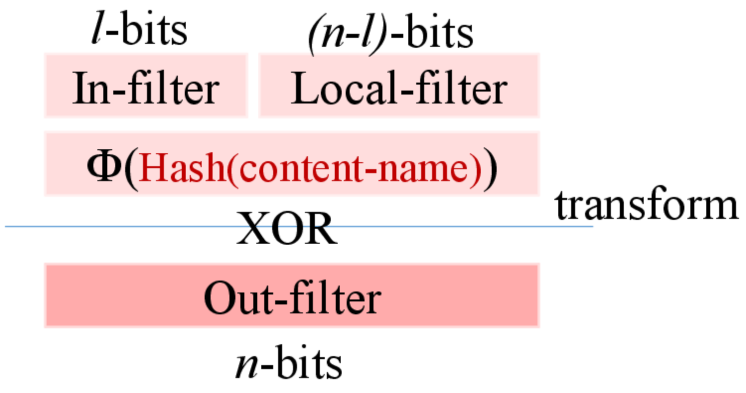
- carry the mappings between interfaces and local filters

As filter implementation is decentralized, each ICN router can insert a dynamic set of control bits to the selected filter for improved robustness

Horizontal Design Choice: Local Transform Filter



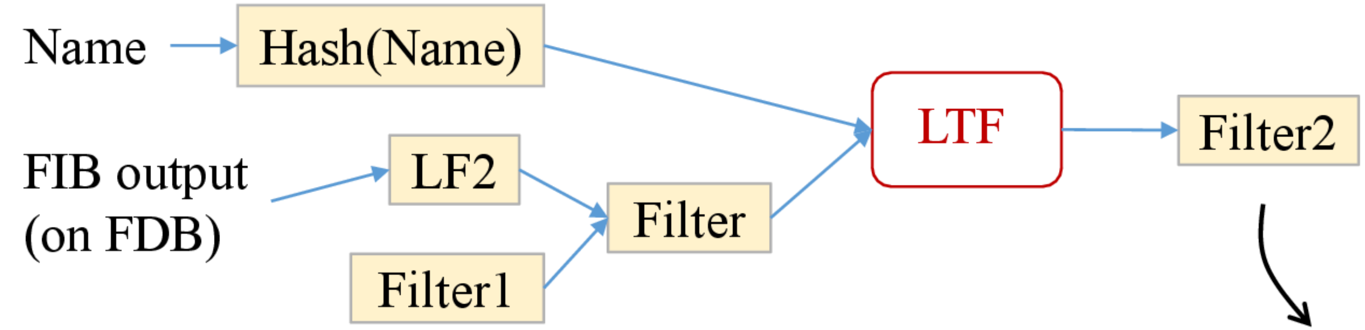
Assume XOR operation for the Local Transform Filter



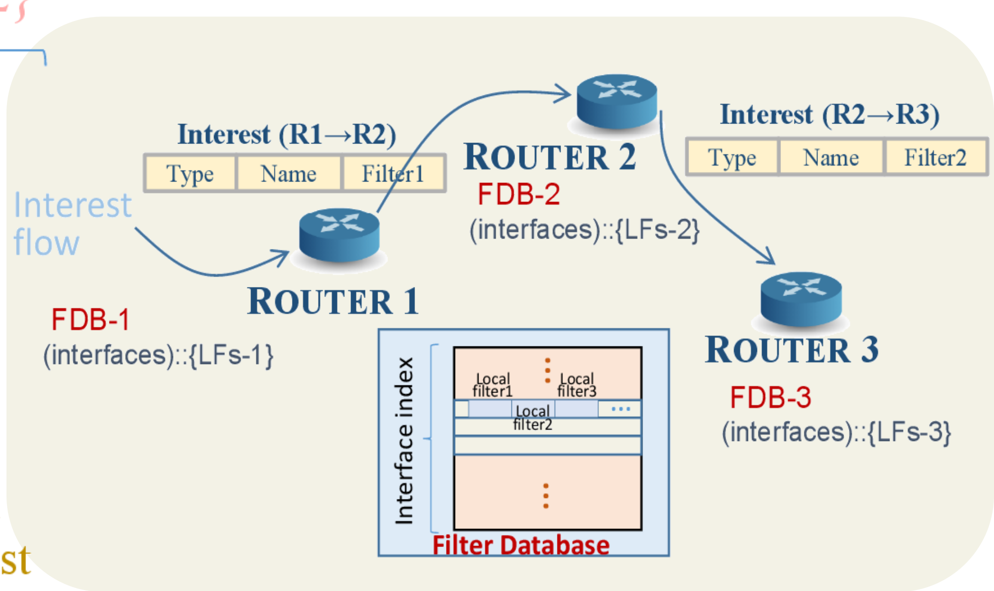
Horizontal Design Choice: Interest Processing

Router 2 - Interest Operations

Input {Name, Hash, Filter1, LF2} Transform Output {Filter2}



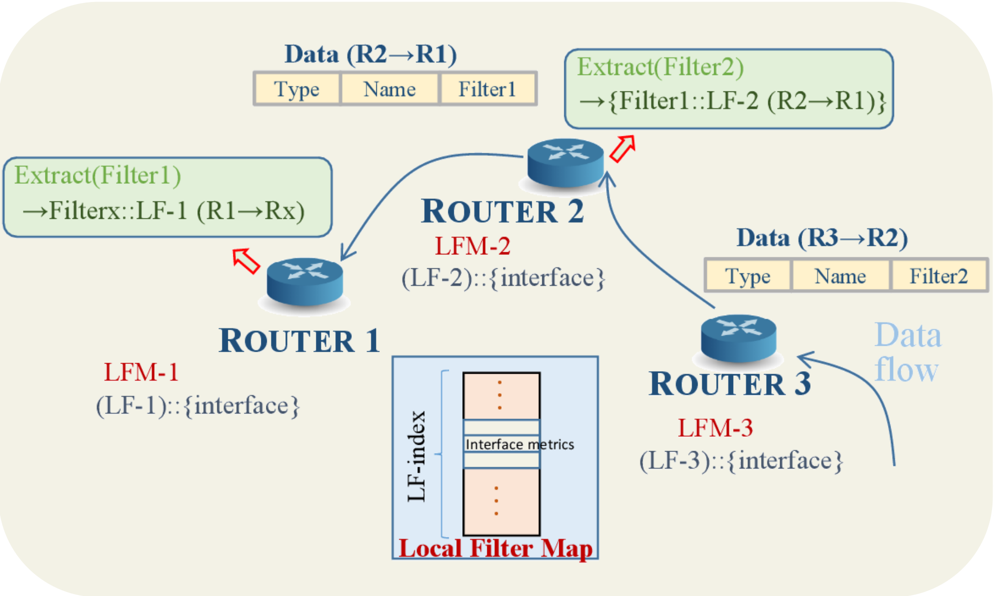
Filter2 replaces Filter1 for the outgoing Interest packet at Router 2 towards Router 3

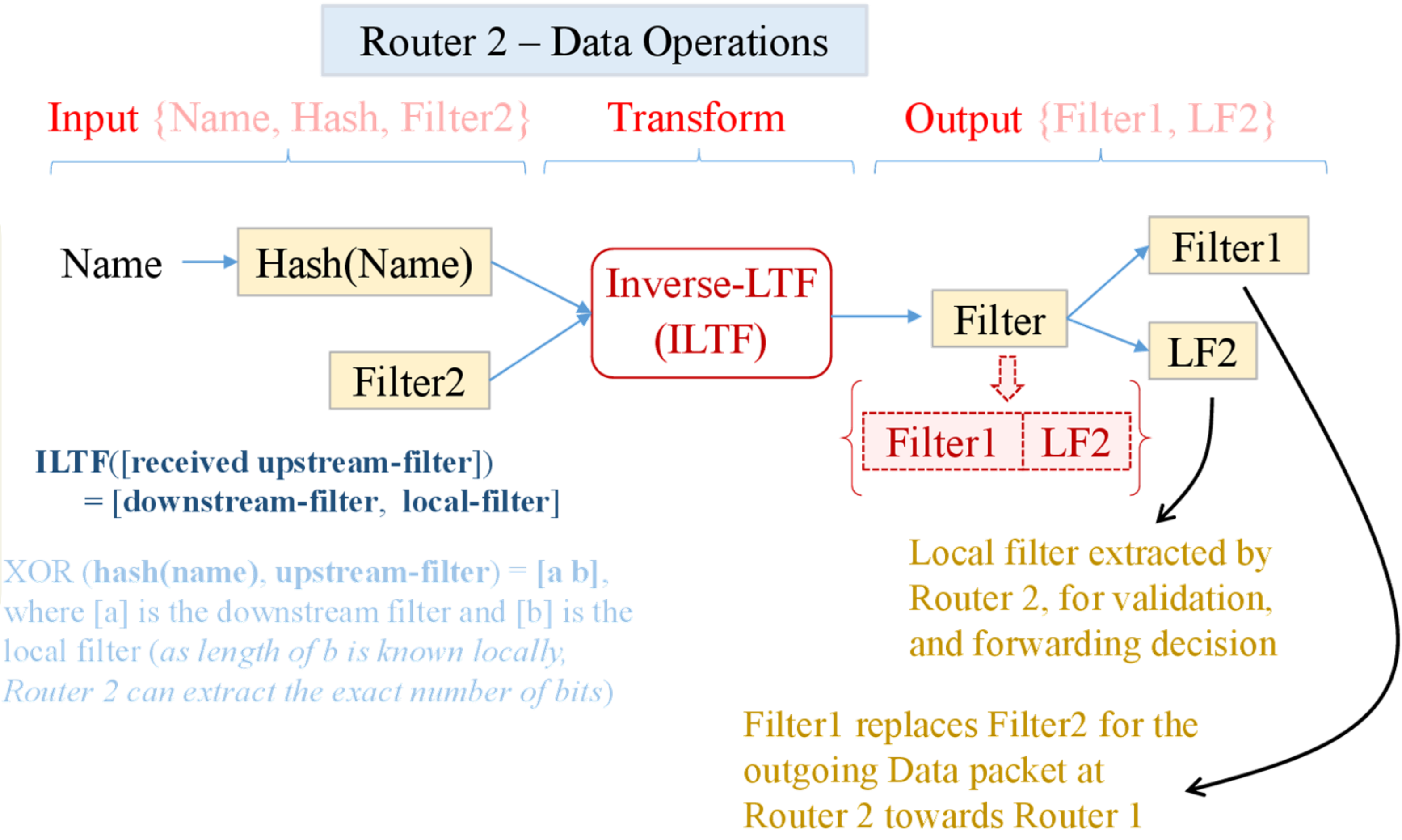


LTF(hash, [**local-filter**, **received downstream-filter**])
= [**upstream-filter**]

We concatenate the received in-packet filter (**Filter1**) in the Interest with the local filter (**LF2**) of size fl >= fl_{min}, then use the **hash output** for the content name to **apply transform on concatenated [Filter1 LF2]**

Horizontal Design Choice: Data Processing





Local Filter Map can be considered as a reformatted version of FDB to enable efficient reverse mapping

Discussion on Common Limitation: Path Failure

Both solutions suffer from the same problem: cannot properly handle link/node failures

- link/node failure typically leads to packet drops as path information is lost

Vertical design choice:

- **link failure**; without knowledge on alternate path's filter, need to use an alternate means to forward the data packet, longer paths increase the impact of false positives
- **node failure**; without having access to an ICN router's filter database, cannot determine the next hop beyond the next hop

Horizontal design choice:

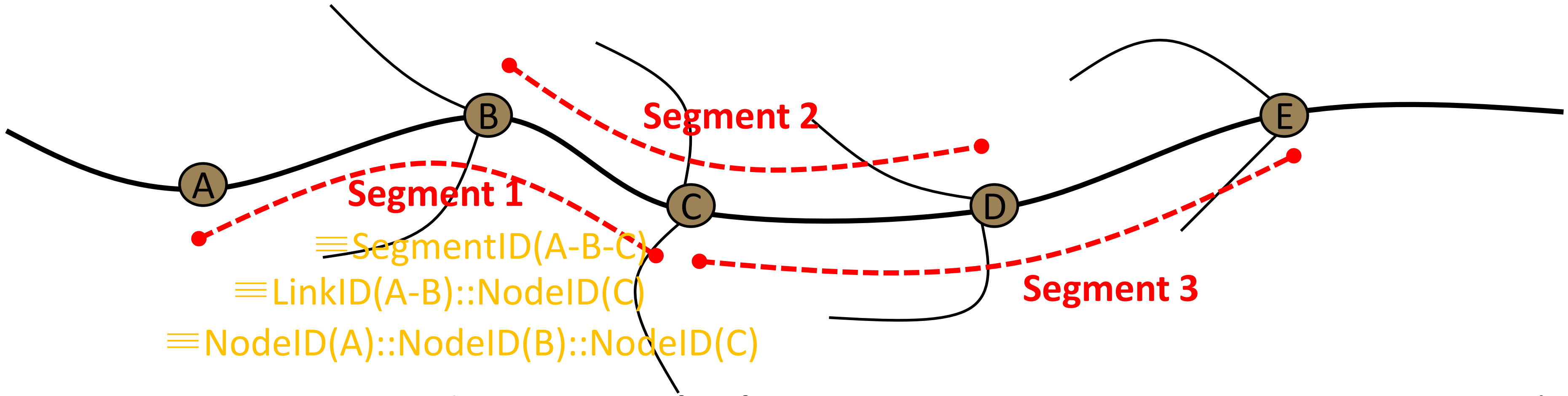
- **link failure**; similar to above (need an alternate means)
- **node failure**; as labels are interleaved, without having access to a node's LTF parameters, cannot recover the path information

How to Support Fast Path Recovery with Stateless Forwarding?

Objective is **to create a secure on-demand source route on the fly** by **utilizing locally transformed path segment identifiers** to create the stateless path

- also continue to address privacy concerns without exposing path information

Store-and-pass path-segment information during path setup using interleaved path segment identifiers



Routers create a path-**segment identifier database (SID)** to include all k-hop path-segments, where k=2 (SID, as intended to include unique path-segment identifiers, may not be necessary and not used for scalability reasons)

Basic Architecture to Support Fast Path Recovery

Type=I	Name	LocalPathFilter(A,C)

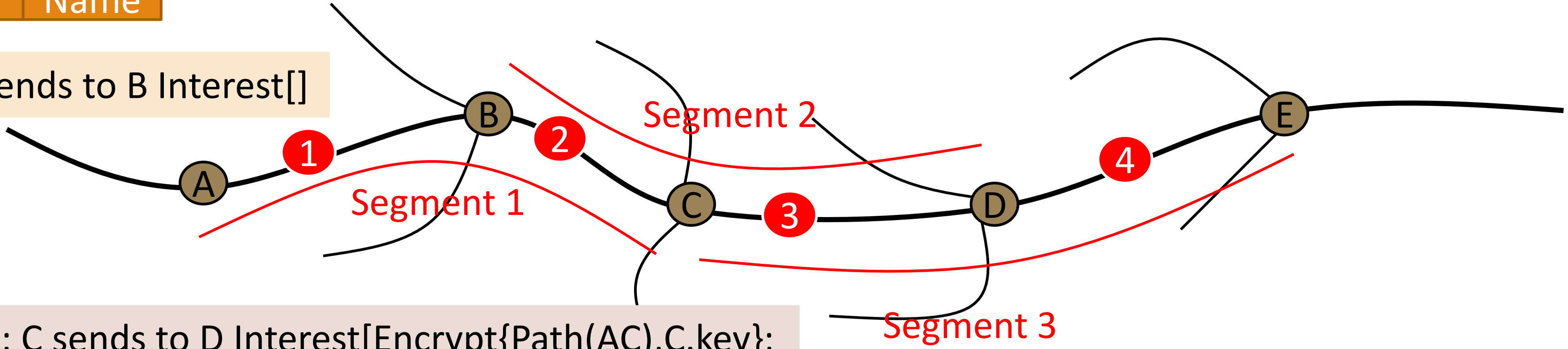
or

Type=I	Name	LPF(A,C)

Step2: B sends to C Interest[Encrypt{Path(AC),C.key}]

Type=I	Name

Step1: A sends to B Interest[]



Step3: C sends to D Interest[Encrypt{Path(AC),C.key};
Encrypt{Path(BD),D.key}]

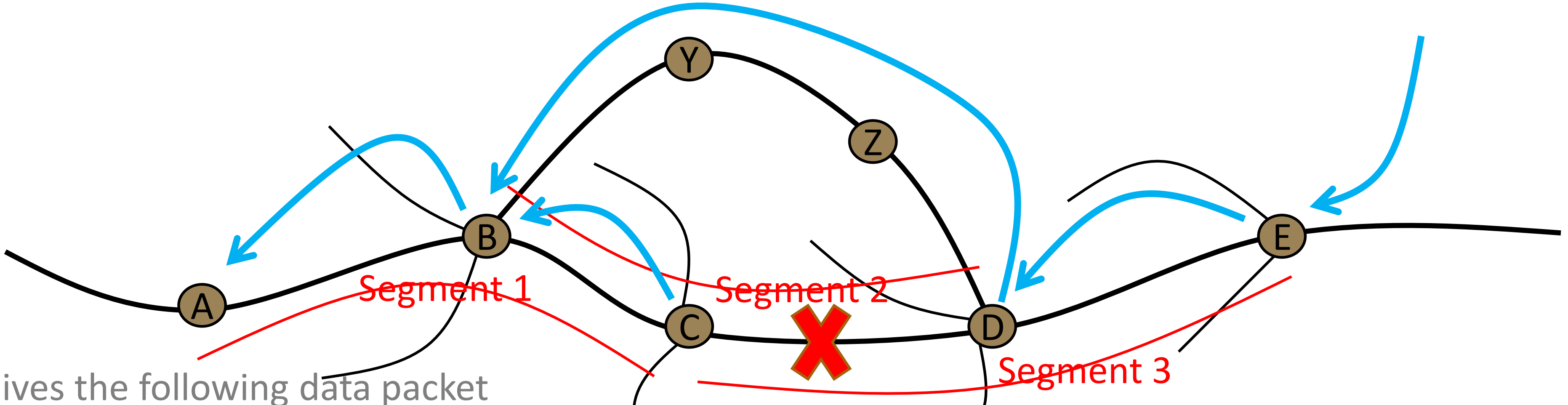
Type=I	Name	LPF(A,C)	LPF(B,D)

LPF: local path filter

Step4: D sends to E Interest[Encrypt{Path(AC),C.key};
Encrypt{Path(BD),D.key};
Encrypt{Path(CE),E.key}]

Type=I	Name	LPF(A,C)	LPF(B,D)	LPF(C,E)

Path Recovery during Data Packet Forwarding



0. Router D receives the following data packet

Type=D	Name	LPF(A,C)	LPF(B,D)

1. Router D extracts LPF(B,D), and decrypts it using its private key

2. Router D extracts information on Router B and Router D; Router E's identifier indicates the previous hop as Router C (same information can also be forwarded by Router E, as separate filter entry)

3. Path(C,D) is broken, so Router D identifies an alternative path to forward Data packet to Router B over Path(B,Y,Z)

4. Router D can include a new path filter of LPF*(B,Y,Z), a non-encrypted path filter, identifying, path and end-host Router B, in case of further failures, packet is forwarded to target Router B through the alternative path(s)

0. Router E receives the following data packet

Type=D	Name	LPF(A,C)	LPF(B,D)	LPF(C,E)

1. Router E extracts the LPF(C,E), and decrypts it using its private key

2. Router E extracts information on Router C and Router E; Router E's identifier indicates the previous hop as Router D

3. If path is operational, Router E sends the packet to D (may or may not include information on C) after removing LPF(C,E) (or replacing with info on Router C)