Empowered by Innovation NEC

HydraICN

Scalable Content Exchange in Challenged ICNs

Bilal Gill

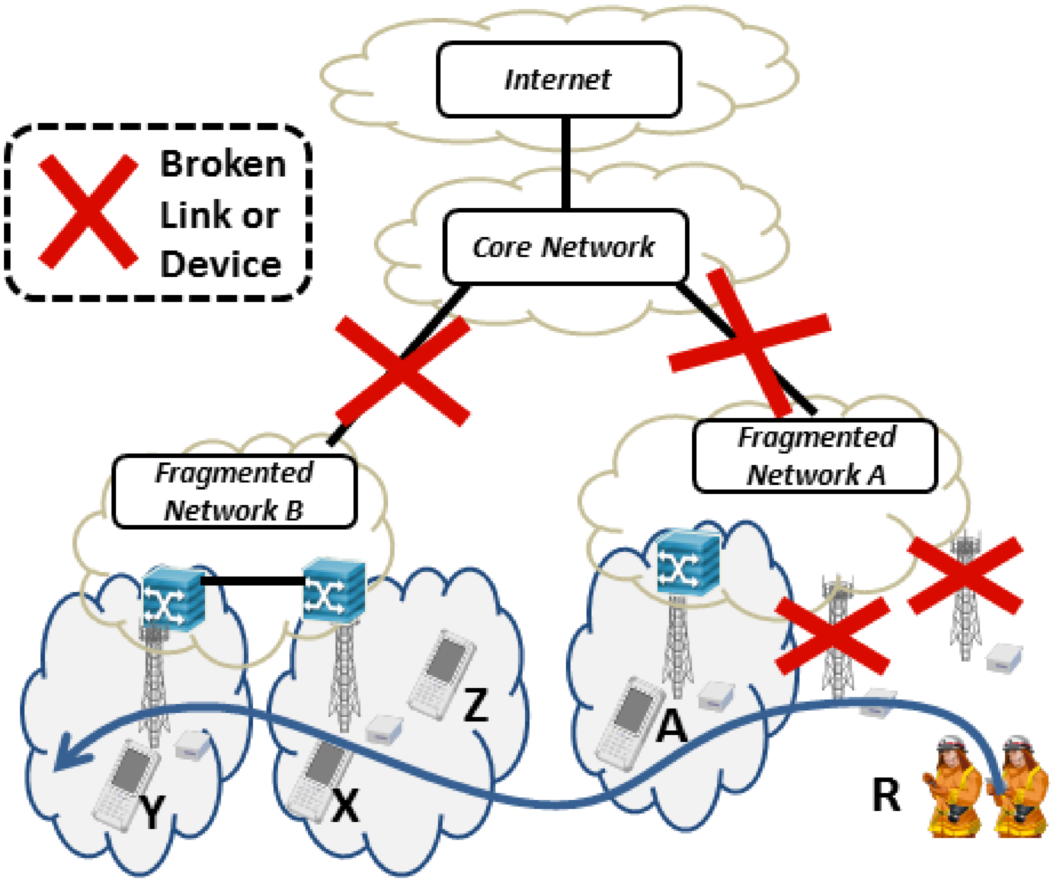
Jan Seedorf

Dirk Kutscher

NEC Laboratories Europe

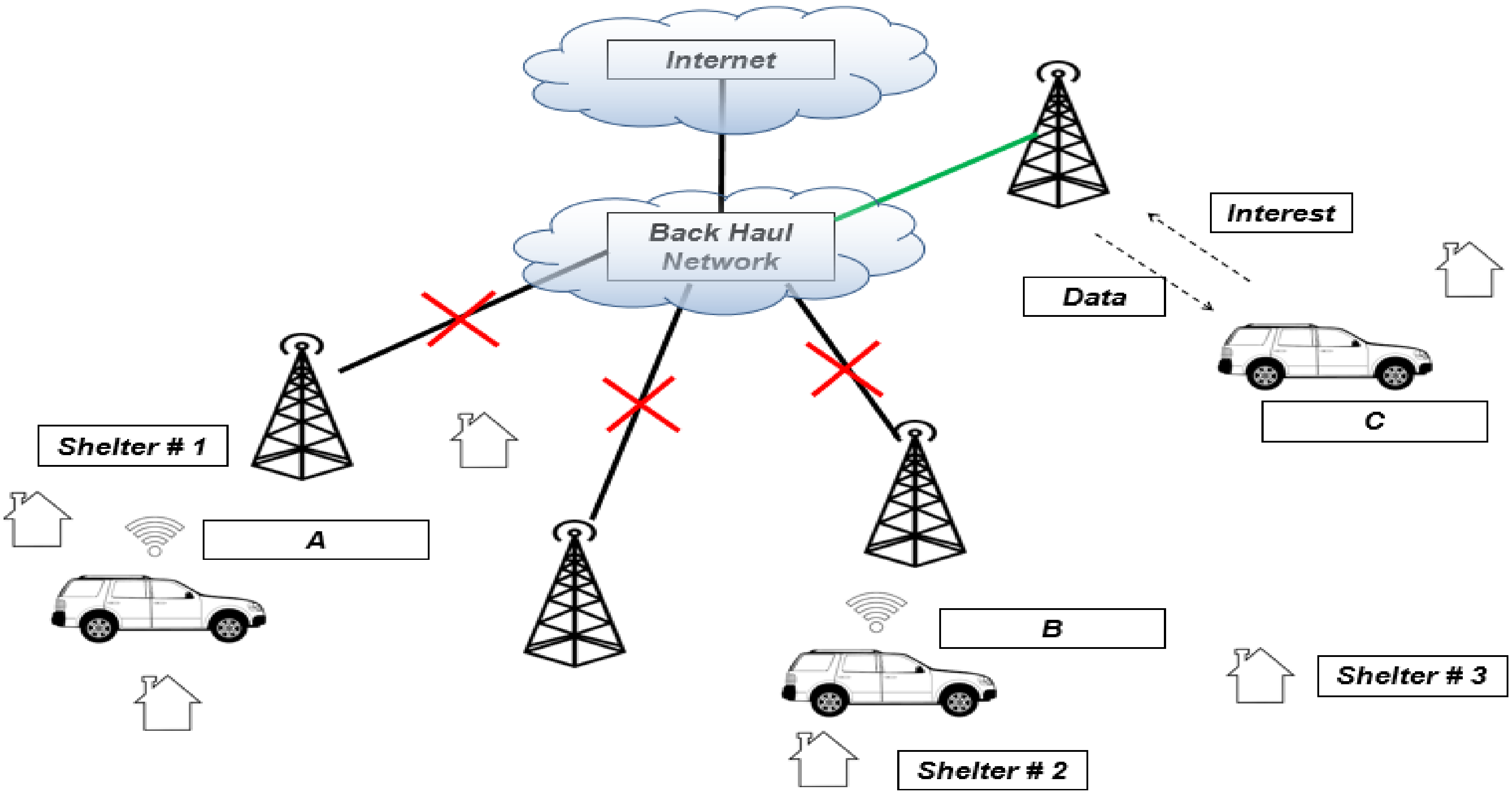
Supported by GreenICN Project

ICN Information Availability in Fragmented Networks



ICN nodes: store-and-forward mode

Data Mules Between Fragmented Communities



Challenges

- Achieving **„optimal" data availability** in fragmented ICNs with intermittent, unpredictable connectivity
- **Allocating storage and networking resources** accordingly
- **Leveraging knowledge about data popularity/relevance** to optimize resource usage
- **No global view on popularity**: need good enough decentralized estimation
- **Estimation algorithm**: balance effectiveness and complexity

Objectives

Distributed counting/aggregating of interests

- Interests serve as popularity counters
- Aggregating interest information without losing too much information

Robust protocol

- Scalable with respect to network size and network lifetimes
- Loop-free to accomodate random, unpredictable movements of ICN nodes

Naïve Approach: Counting Interests

- **Append nonce to each unique Interest message** and accumulate nonces in the network
  - Interest + nonce == unique interest

- **Advantage: can lead to accurate representation** of popularity per Interest at many nodes

- **Disadvantage: Scalability problem**
  - Need to store all nonces per Interest
  - Also: need to exchange complete nonce list when two nodes meet

Our Approach

- Idea: **Aggregate [nonce:count] tuples**
  - Approximate content popularity

- Append nonce to each unique Interest message and **count nonces as a popularity counter**: **[nonce:count]** tuples

- **Data mules maintain sorted list of [nonce:count]** per interest for scalability

- When two data mules meet, they **exchange their Interests**
  - Interests & Popularity estimation gets distributed in network

Our Approach

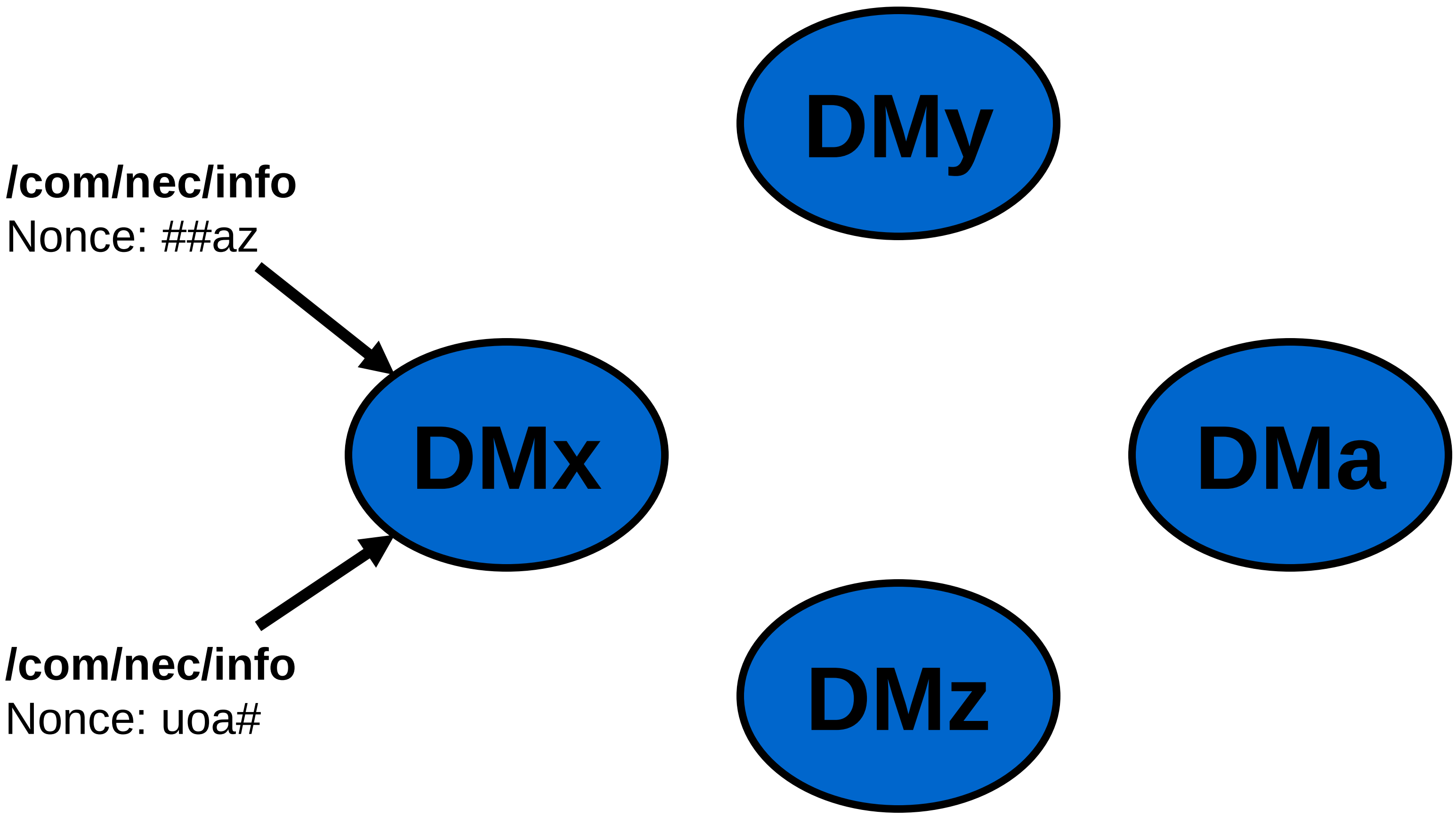
- For each Interest, aggregate **[nonce:count]** tuples as follows:

- **Compare([nonce1, count1] , [nonce2, count2])**

- **IF nonce1 == nonce2**
  - **new_count = MAX(count1, count2) (at both nodes)**

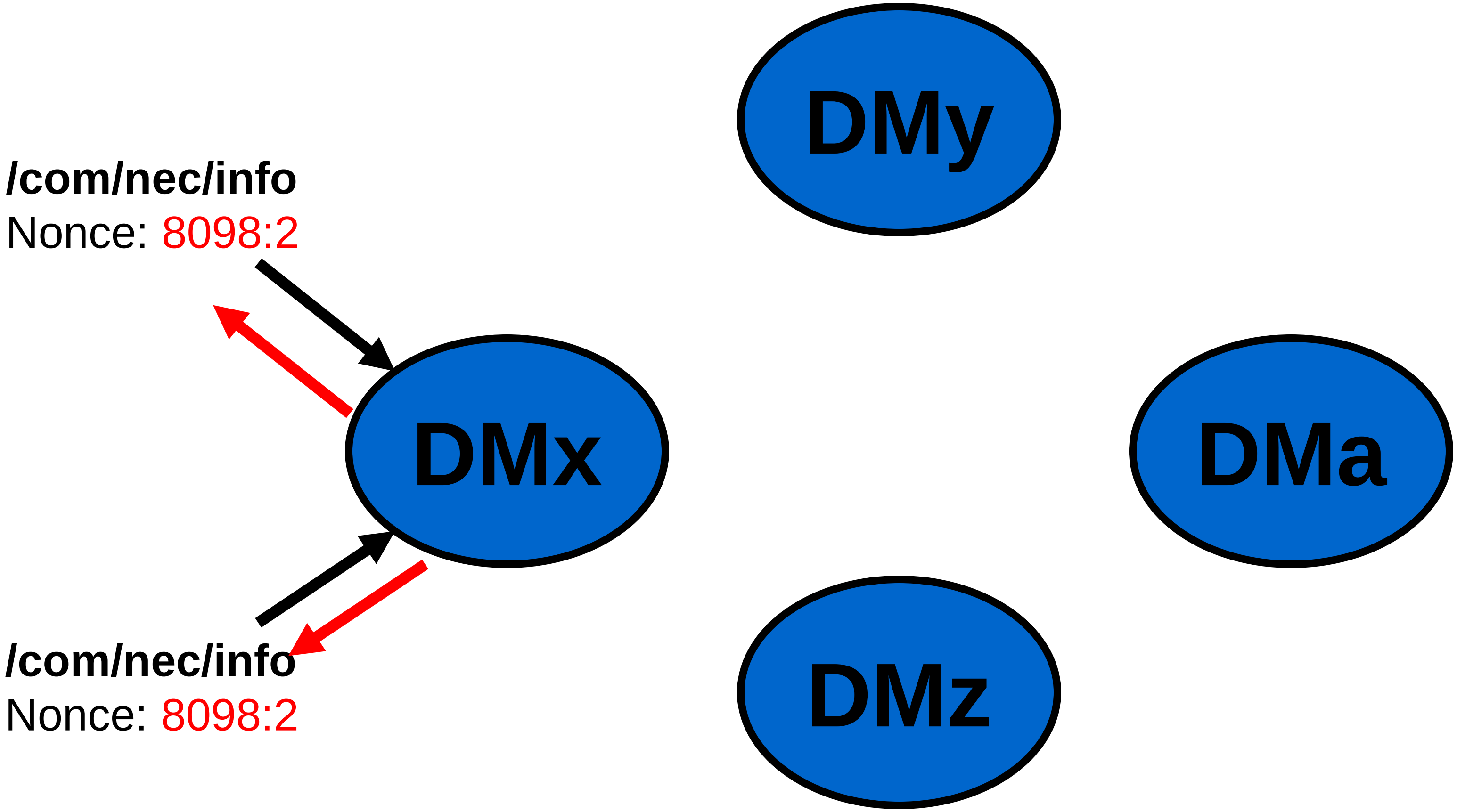
- **IF nonce1 != nonce2**
  - **New_nonce = nonce with largest counter([nonce1, count1], [nonce2, count2])**
  - **New_count = count1 + count2**

Sample Exchange 1



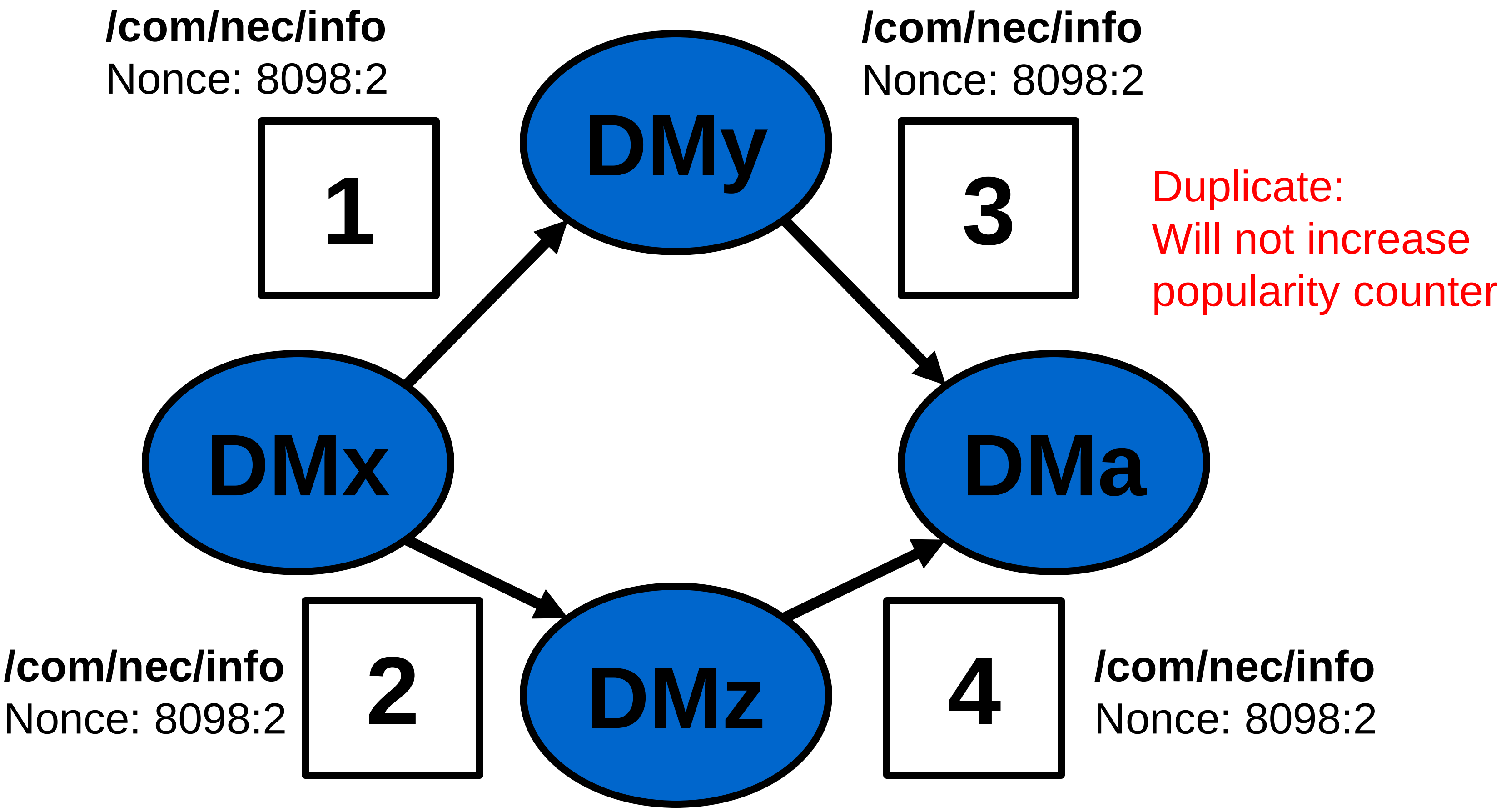
Two end users request the same name from DMx

Sample Exchange 2



End users are assigned a new nonce with count 2 for the same prefix

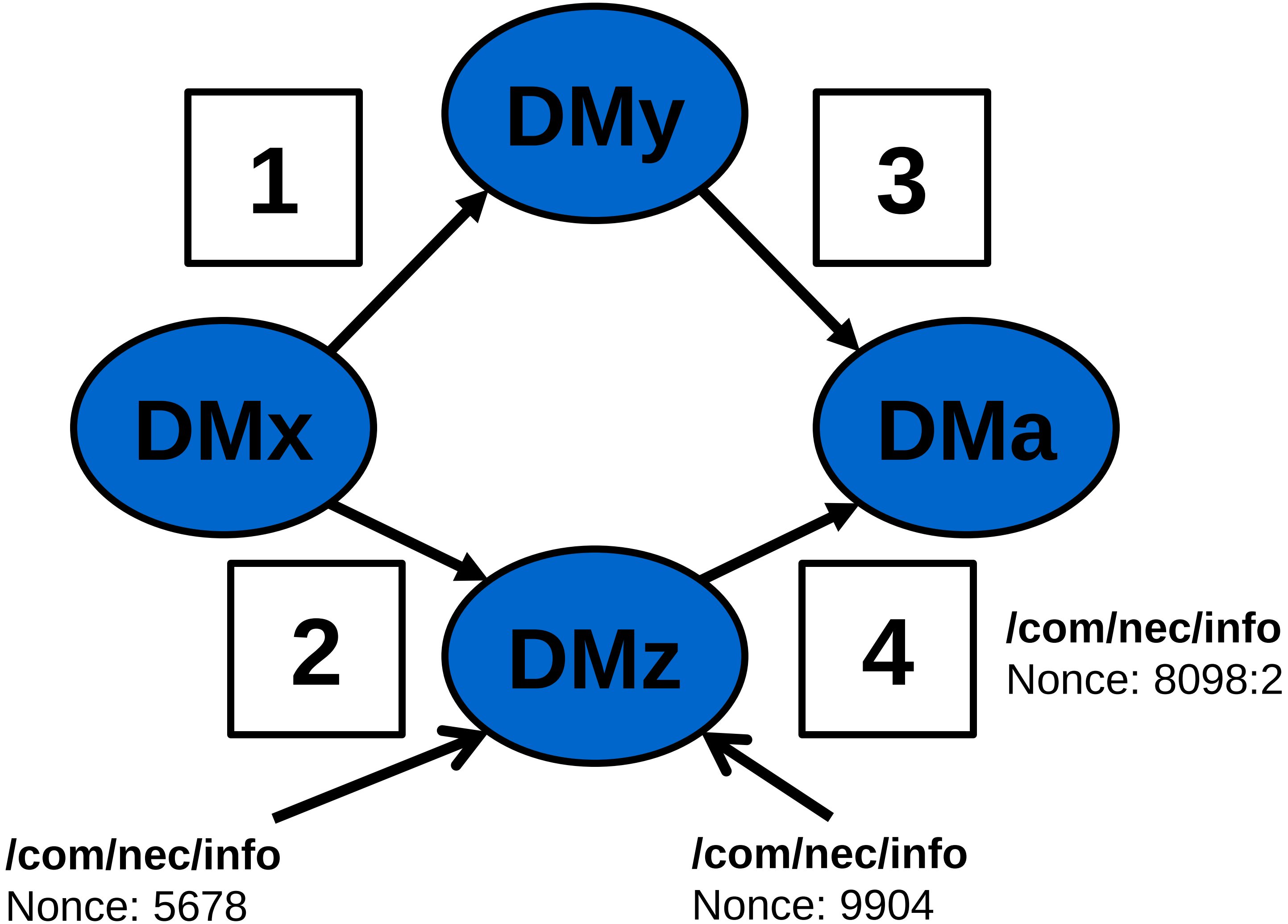
Sample Exchange 3



DMa receives duplicate query (loop) from DMz, thus drops the message (loop detection)

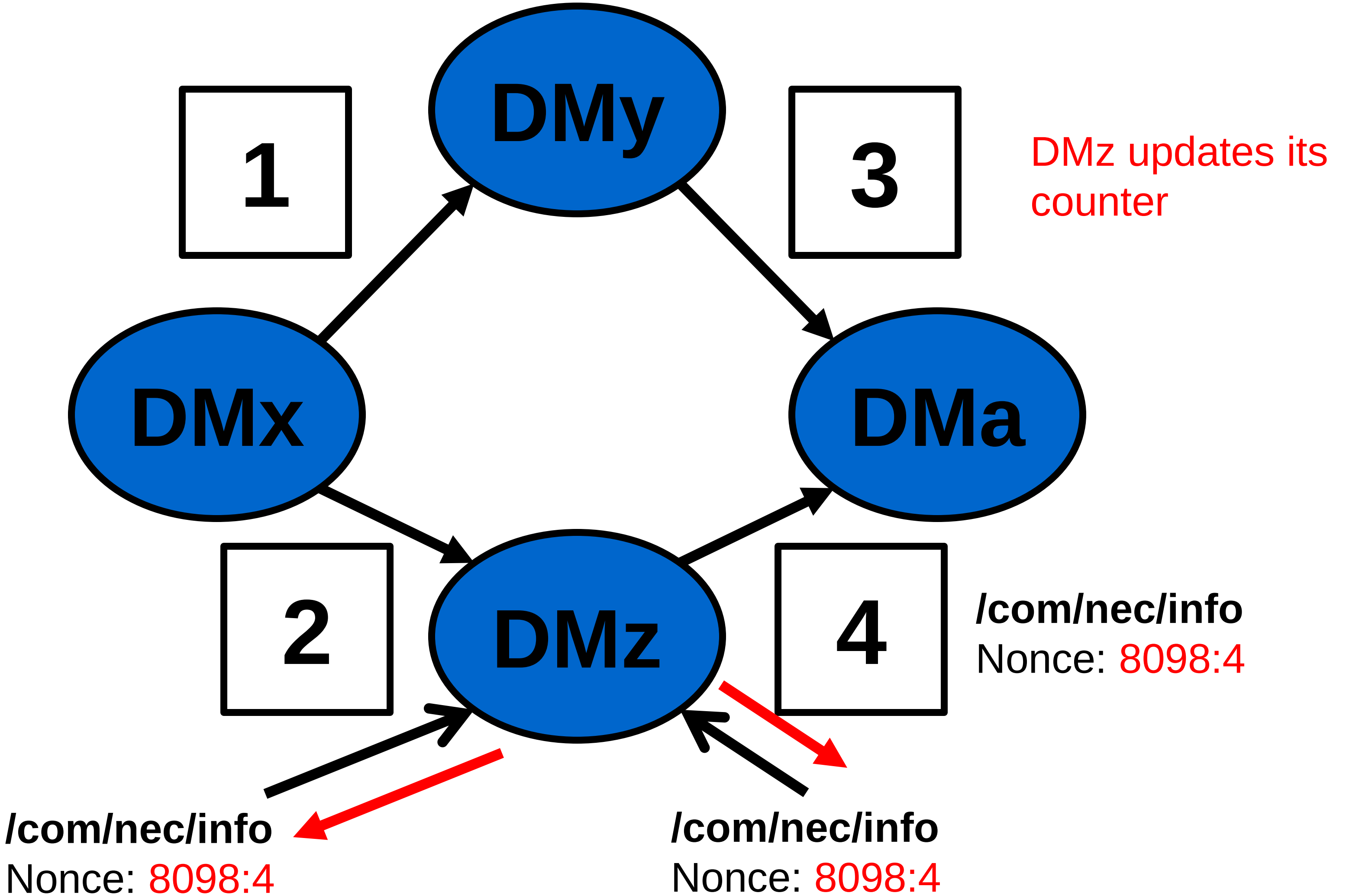
Sample Exchange 4

Two new user requests with same prefix from DMz

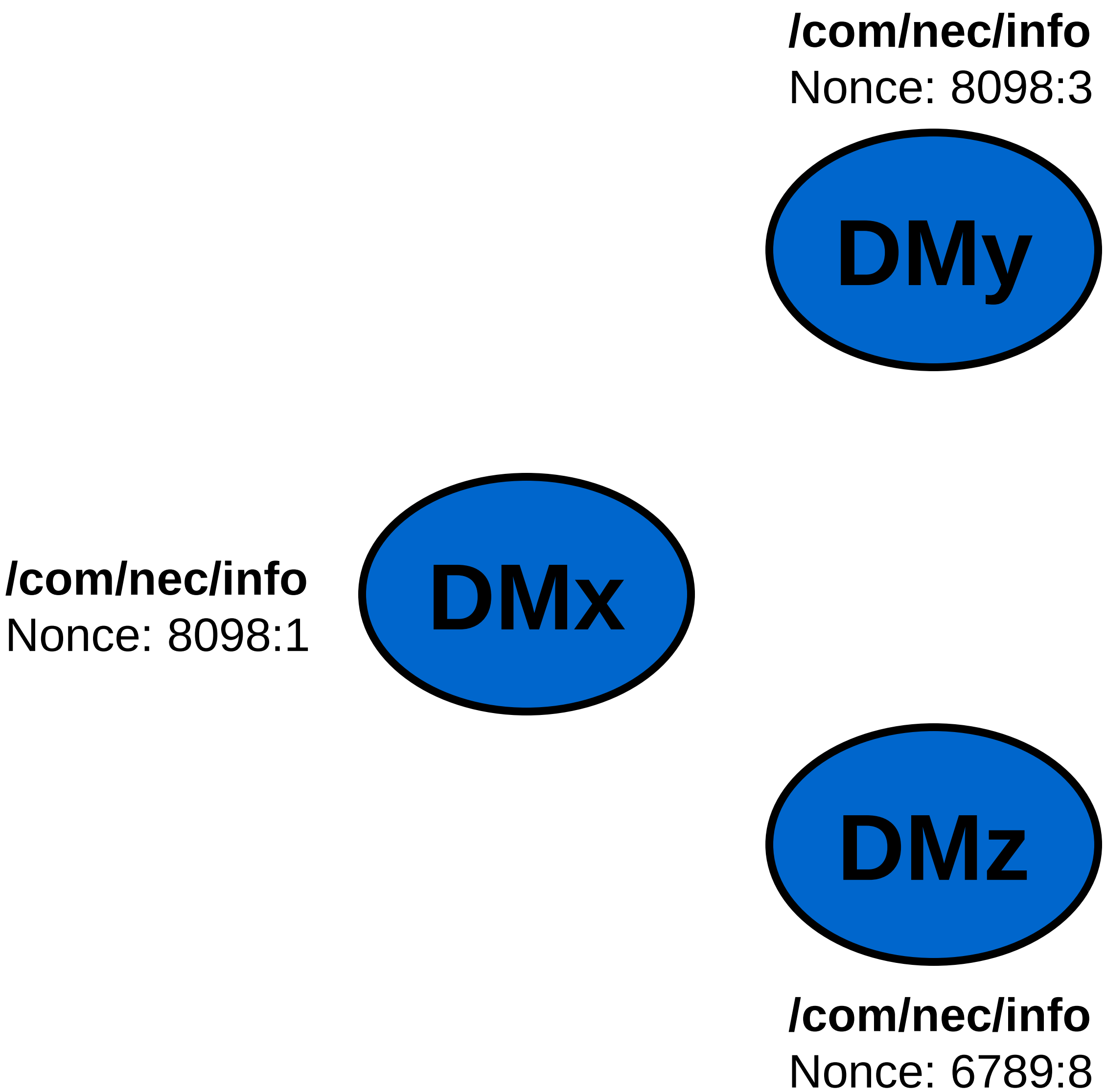


Sample Exchange 5

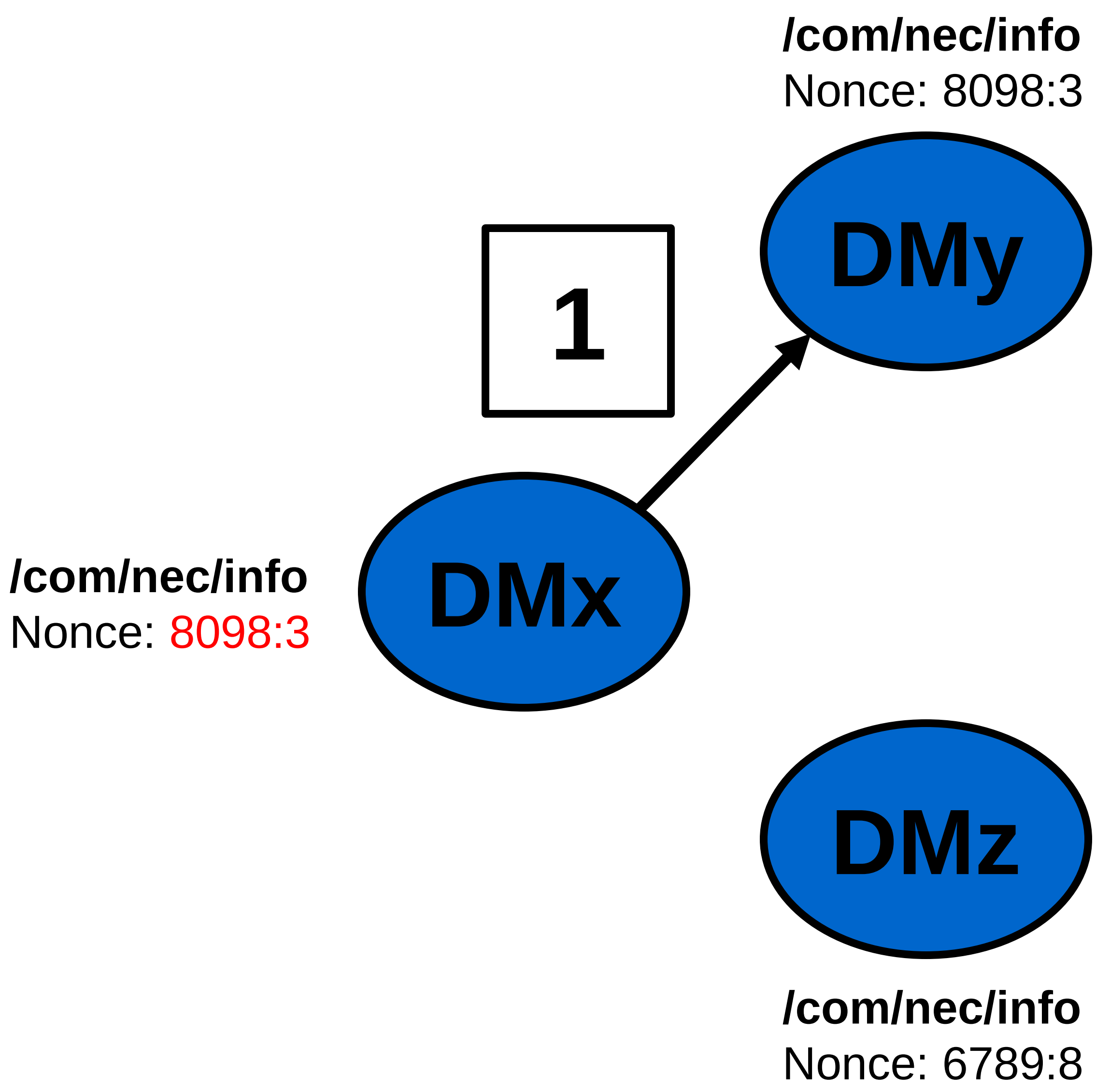
DMz updates its count and sends new once to the end user



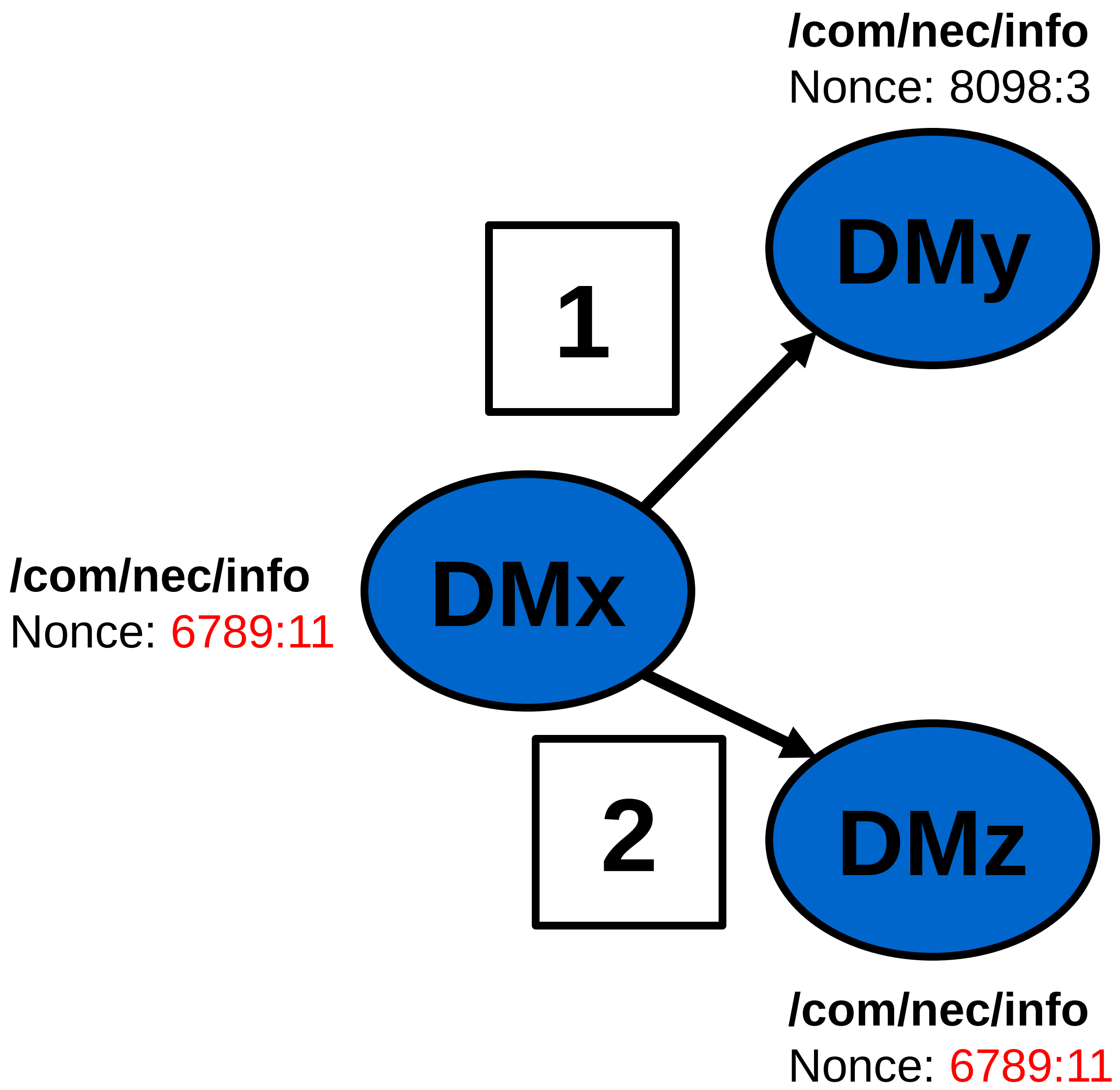
Loop Problem



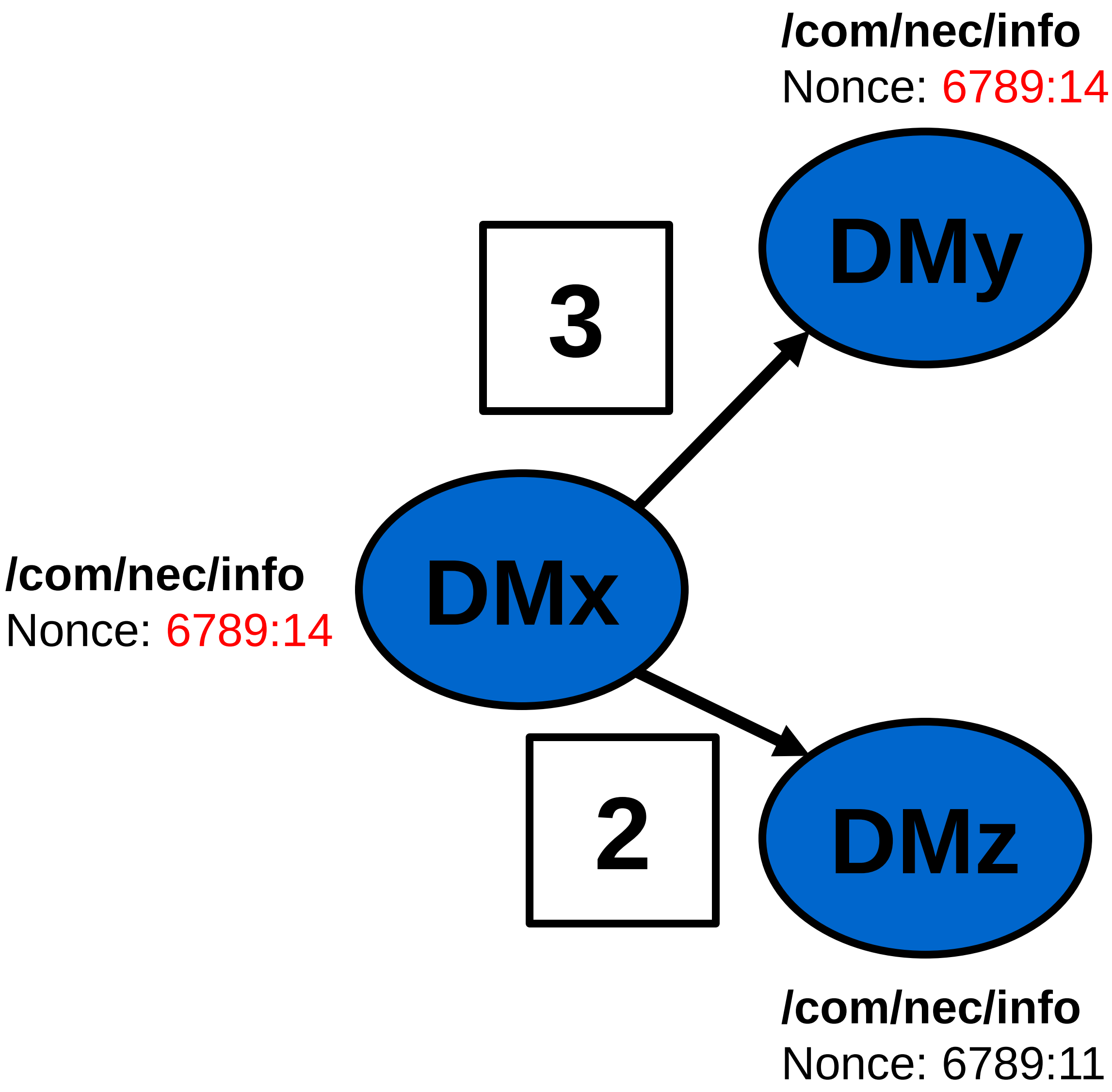
Loop Problem



Loop Problem



Loop Problem



Over-estimating Popularity!

Loop Prevention

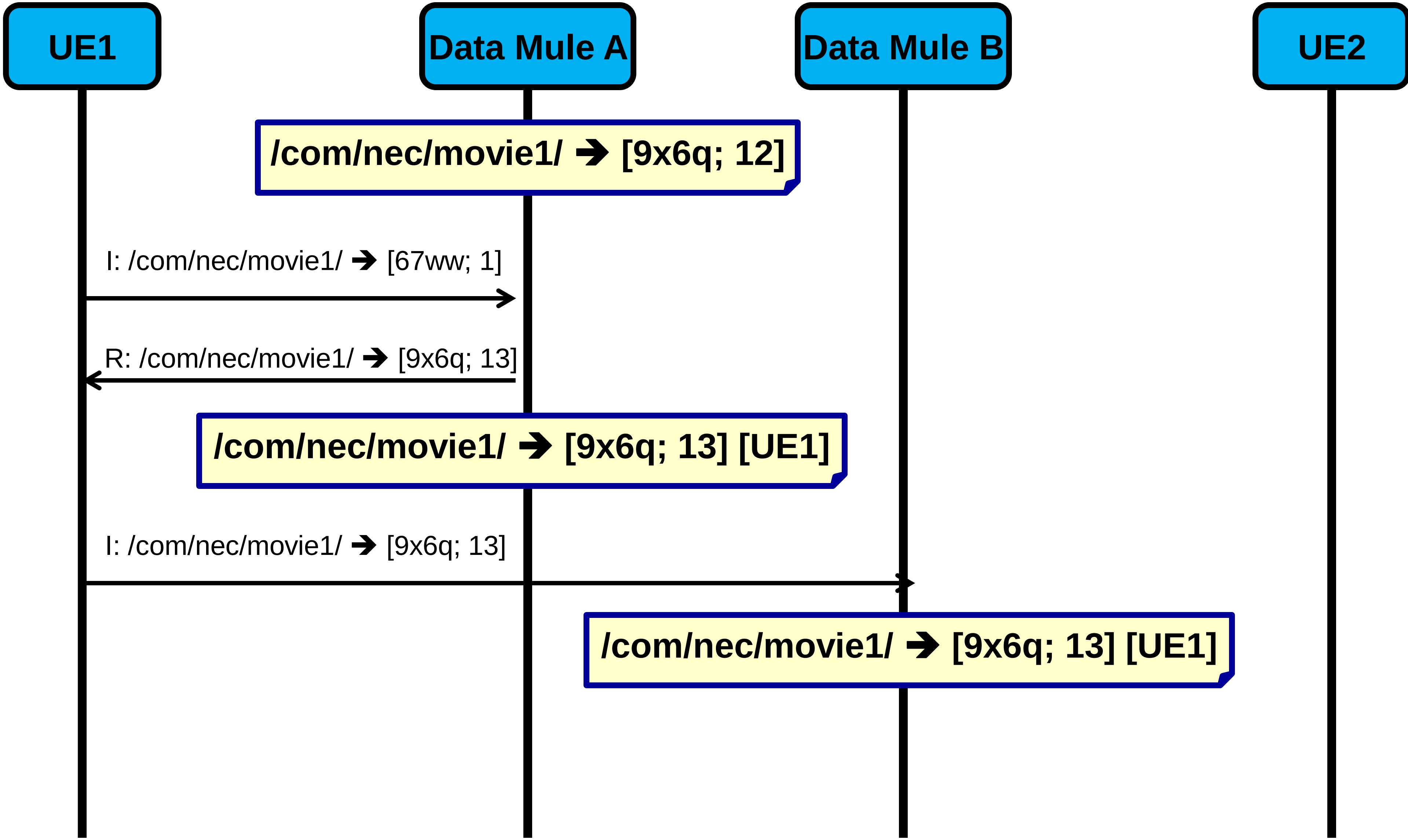
- Data mules keep list of recently encountered mules per Interest

- If a data mule is encountered shortly after the first encounter, counters are not added

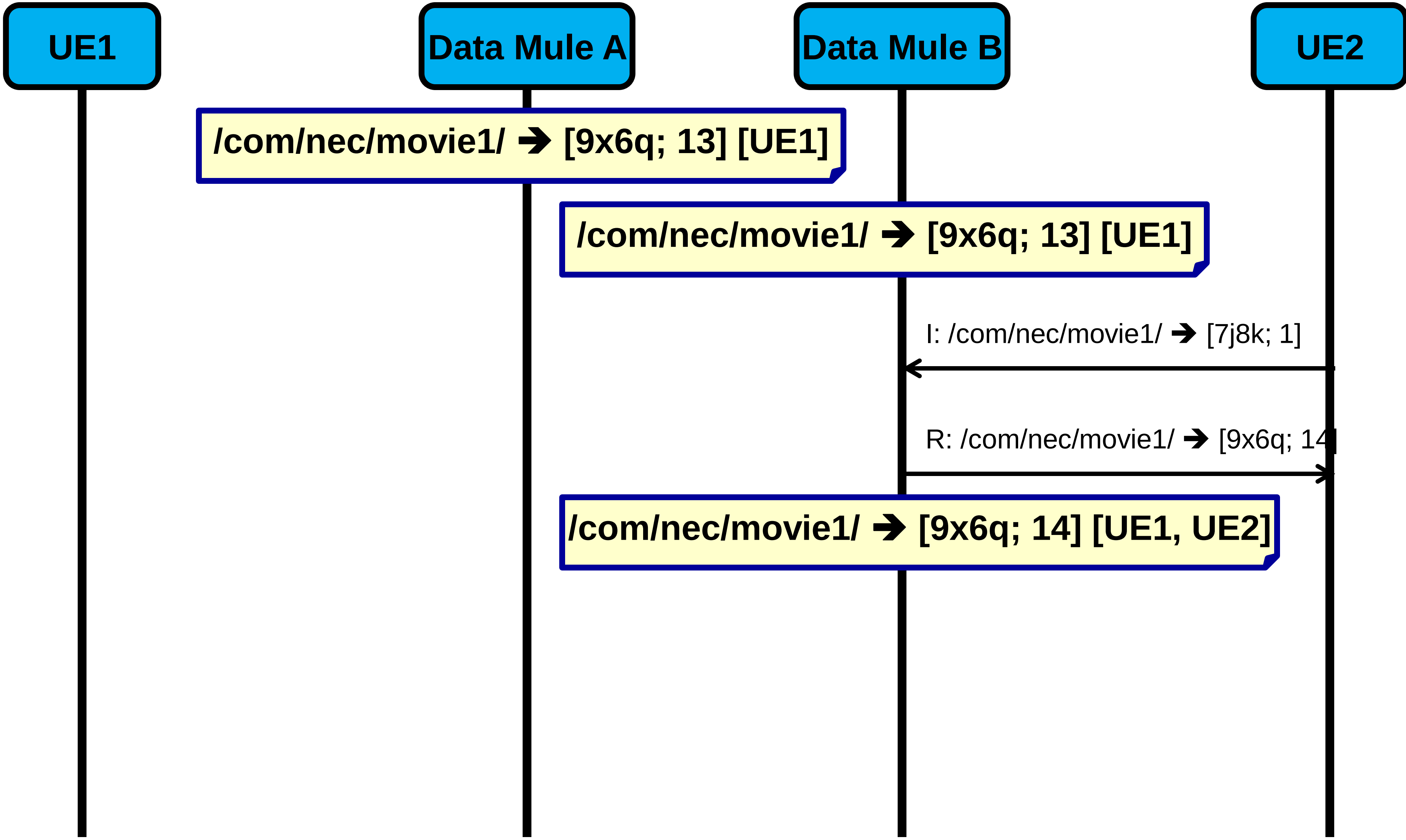
- Instead, max-counter rules is applied at both sides
  - **`nonce1 != nonce2 (AND recently met)`**
    - **`New_nonce = nonce with largest counter([nonce1, count1], [nonce2, count2])`**
    - **`New_count = MAX(count1, count2)`**

- Sliding approach: FIFO list of encountered nodes has limited (configurable) size
  - Can trade off acuracy against memory conservation

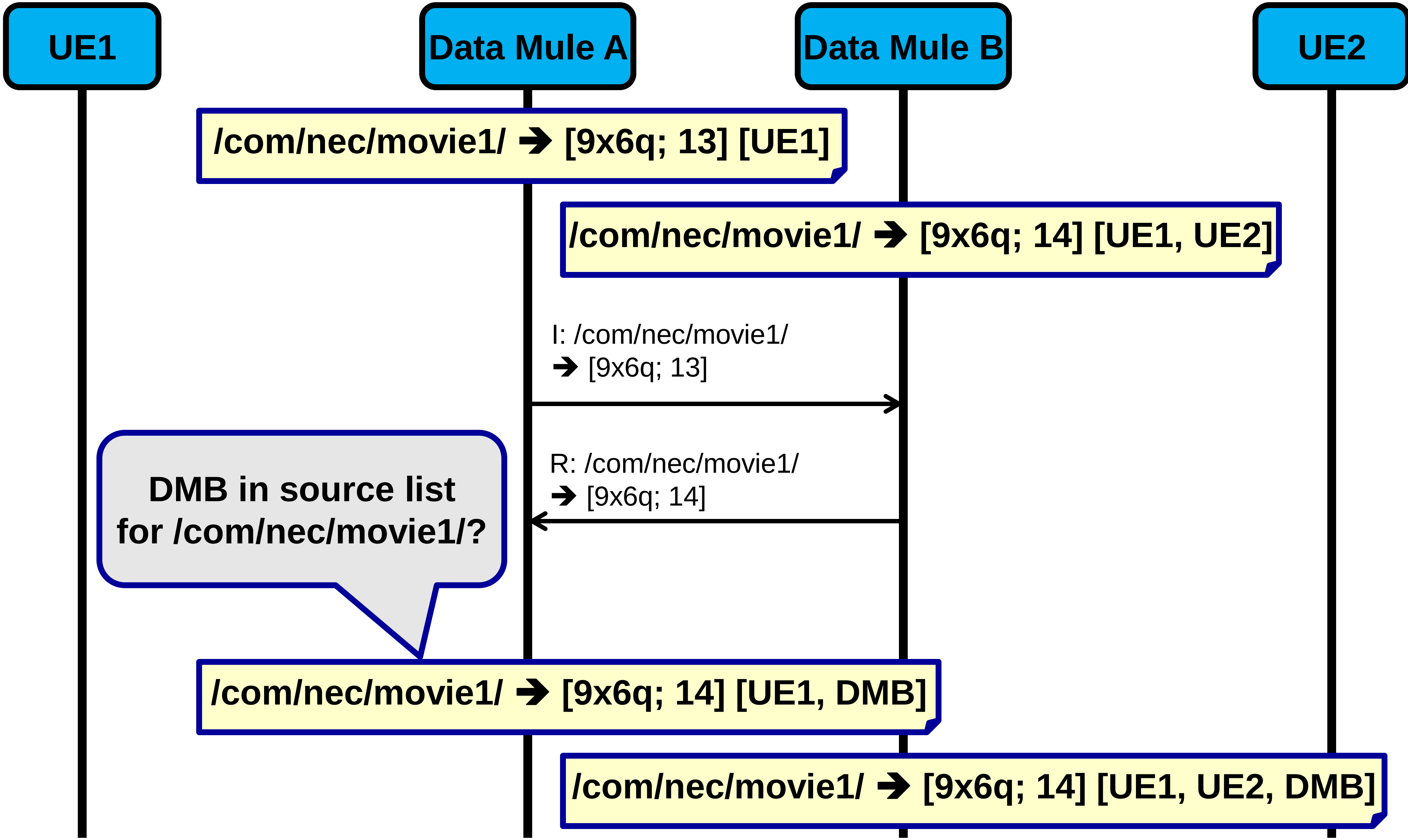
Exchange Scheme (1)



Exchange Scheme (2)



Exchange Scheme (3)



Implementation

- Interests are retransmitted whenever two nodes meet
- Application on top of CCN creates FIB entries for all the interests (names) we want to send to the next node
- The life time of an interest packet is set to infinite
  - Objective: don‘t expire interest
  - PITs can become large...
- Current exchange protocol implemented in TCP (not in CCNx protocol)
- Popularity count for all the prefixes in the PIT
  - Colon separates nonce and count

```
/com/nec/info          8098:4
```

Name/Popularity-Based Prioritization

- Contact time between two data mules could be short
  - Data prioritization scheme needed to optimize exchanges
  - Here: **Leverage popularity estimation for prioritization**

- ***Name-Based Prioritization*** **protocol (NBP)**

1. Node contact: nodes exchange meta data about cached objects
   - List of names, assessed popularity

2. Requesting node generates Interest packets, ordered by „priority“

- Different categorization schemes possible
  - GreenICN: prioritizing by name / prefix (disaster scenario)

Evaluation

- Objective: evaluate accuracy of distributed popularity estimation

- Two Fragmented Communities

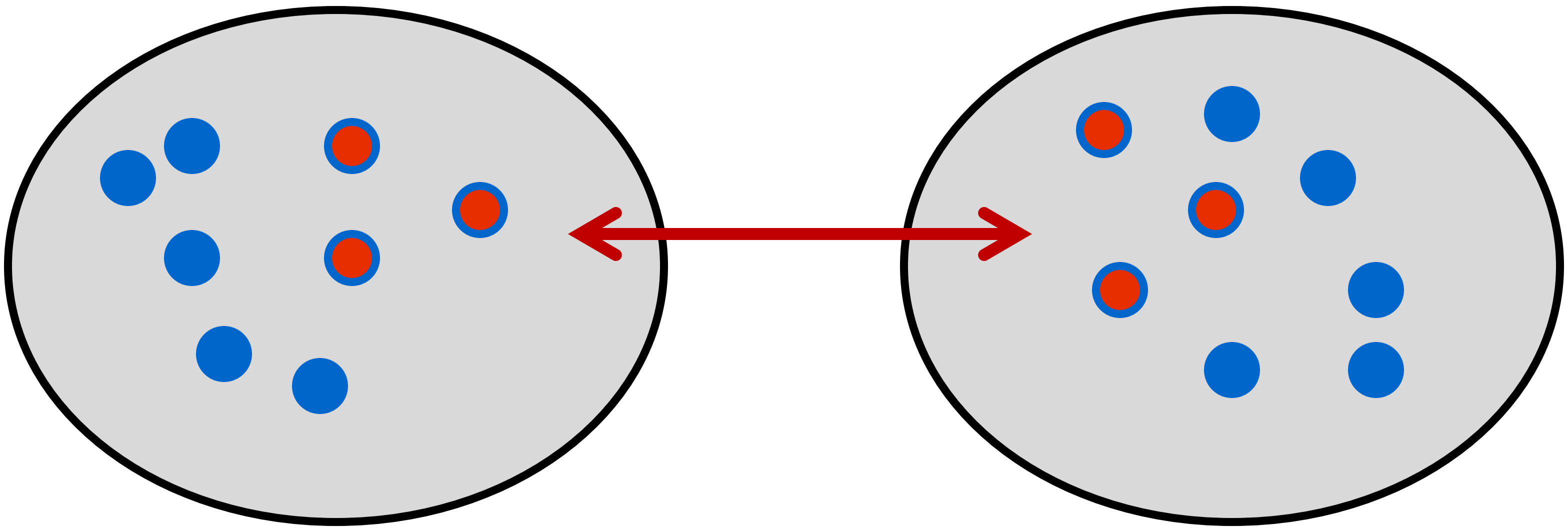
- 100 distinct data objects
  - Queried from three different DMs in each of the fragmented communities

- Data mules meet each other in a random manner and exchange interests

- Zipf distribution for Interests with 0.8 < alpha <0.9

Scenario Diagram

- **3 Nodes per FG which fetch interests from 600 end users**
- **Different sets of names (popular content) in FGs**



Fragmented Community 1 Fragmented Community 2

Measurement Setup

Areas with data mules

- Emulation: Mules meet randomly, no disruption during intermeeting times

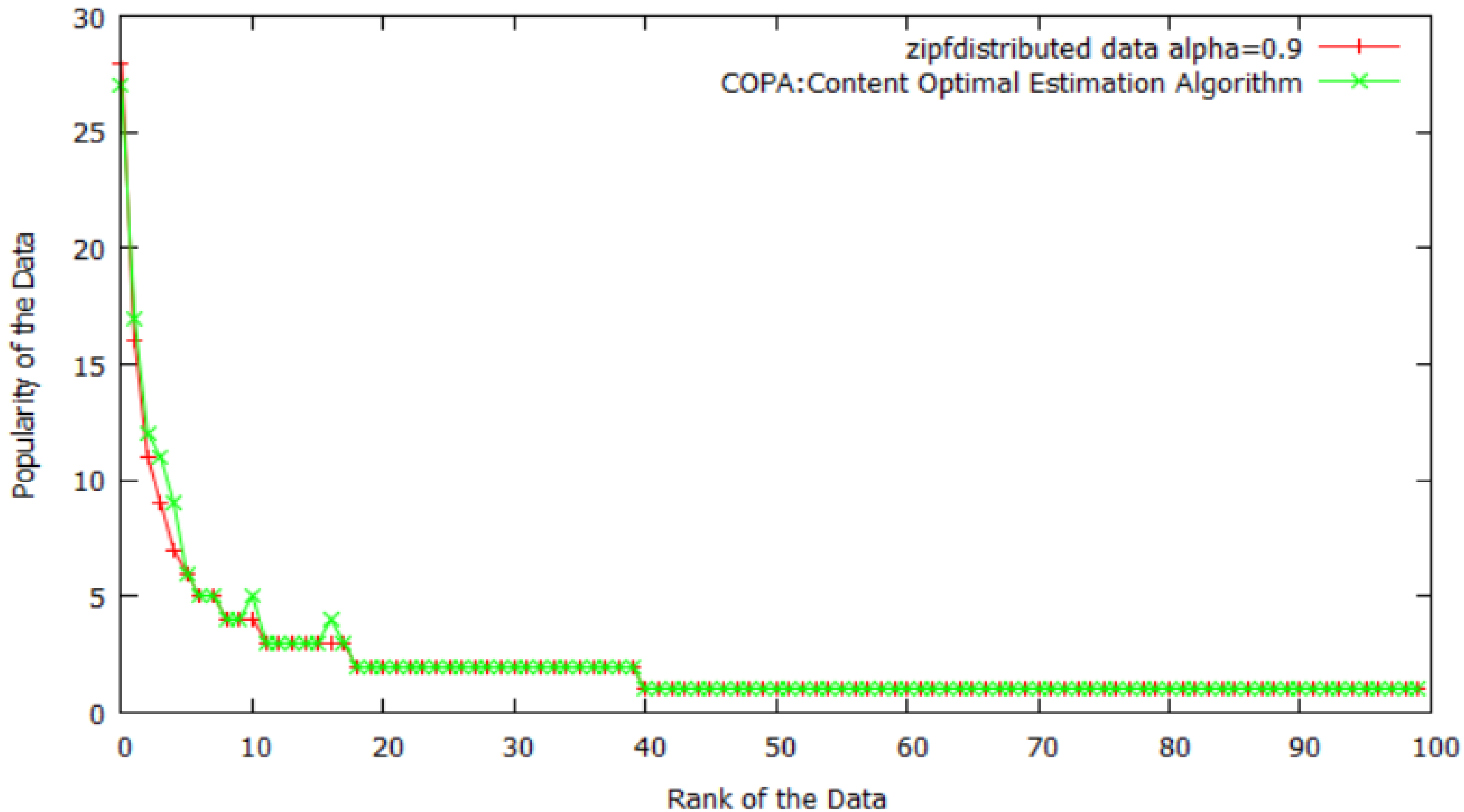
Details of the test setup

- 6 nodes, 2 Fragmented Communities with different set of names
- Three nodes in their respective FG fetch interest from end users
- Meet DM at random times
- 15 contacts between different DM at different times
- Object size: 8 KB
- Approx. 500 Interests issued by users

Test specs

- 10 runs for each of the normalized results
- Each runs takes several minutes

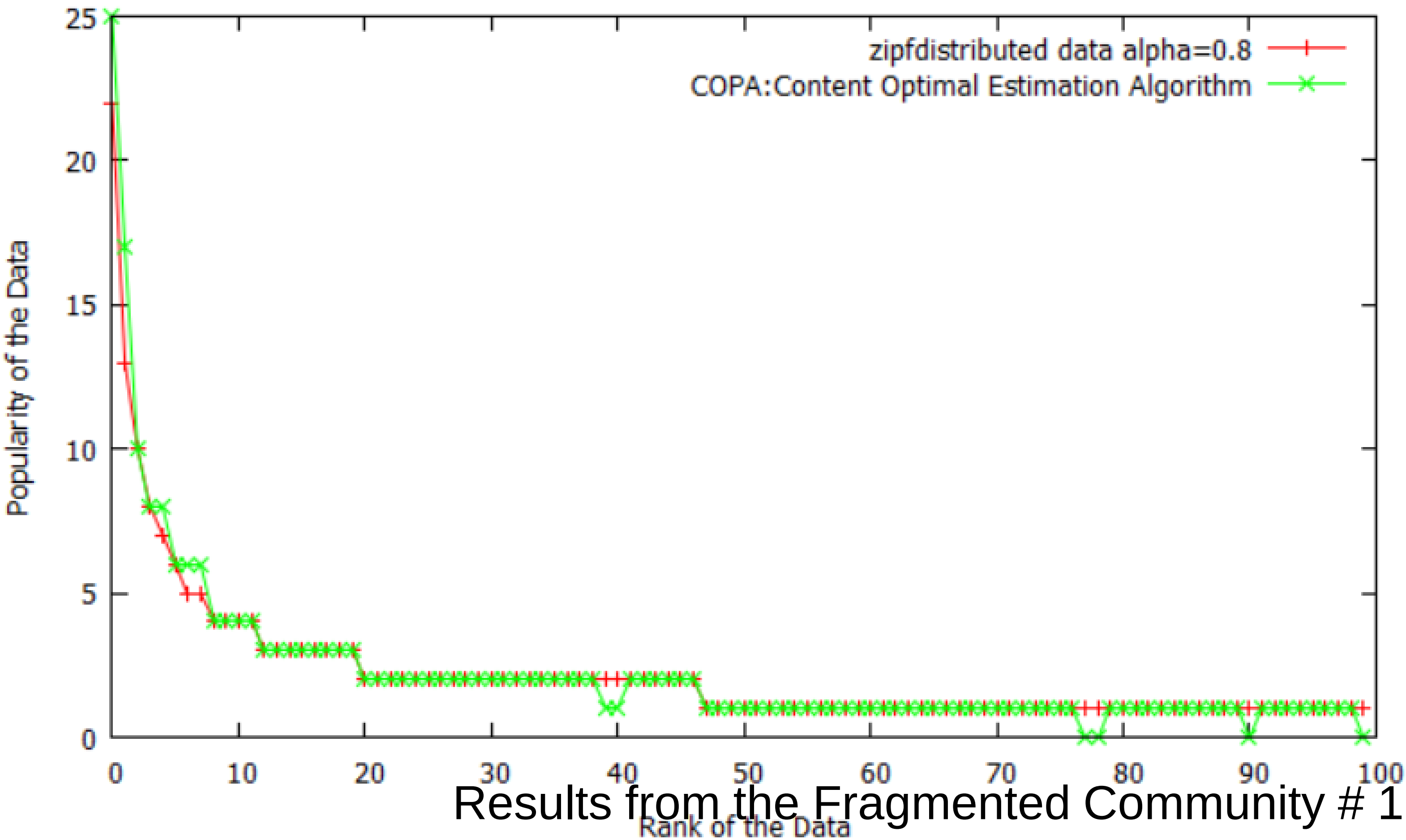
Evaluation (15 Contacts)



Results from the Fragmented Community # 1

Zipf, alpha=0.9
15 contacts

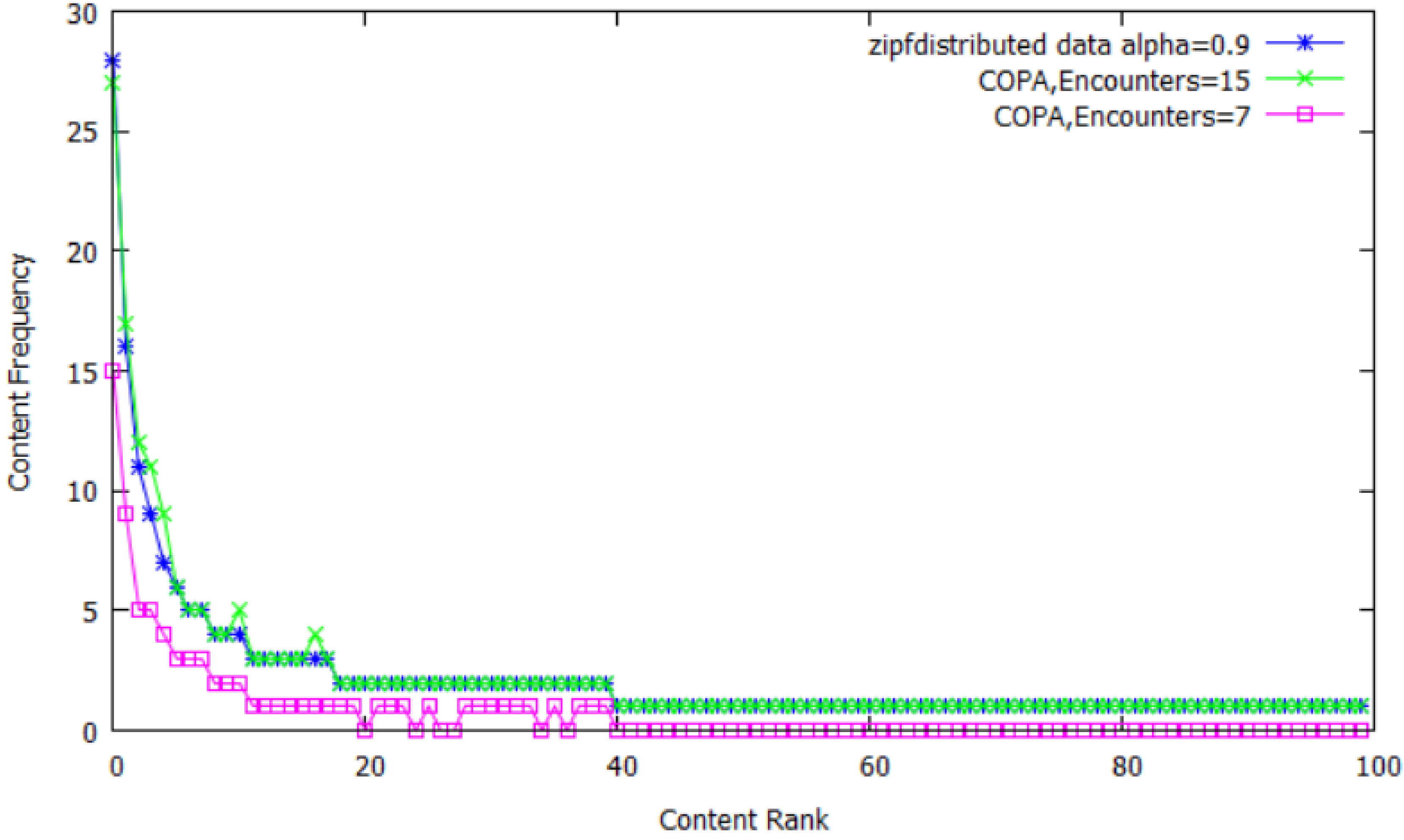
Zipf-Alpha=0.8



Results from the Fragmented Community # 1

Zipf, alpha=0.8
15 contacts

Better Estimation with more Contacts



Zipf, alpha=0.9
7 and 15 contacts

Observations

- More contacts: better accuracy
  - 15 contacts lead to quite good results
  - But: consistent underestimation (for all objects) – so may not be a real problem

- There can be more than one nonce per object name in the network
  - Can lead to slight overestimation

- PITs can become large

- Our approach significantly more memory-efficient than naive approach (thanks to aggregation)

Summary

- **Popularity estimation in ICN** for optimizing performance and availability (here: DTN scenario)
- **Leveraging ICN naming and storage features** – changing CCN semantics (store-carry-forward of interests for long time)
- **Simple scheme – intuitively scalable**
- First evaluations suggest **sufficient accuracy**
- Next Steps
  - Polish implementation, documentation, more experiments
  - Mobile GW implementation

Empowered by Innovation

NEC

www.greenicn.org