Distributed Dataset Synchronization in Mobile Ad Hoc Networks over NDN

*Tianxiang Li*, Spyridon Mastorakis, Xin Xu, Haitao Zhang, Lixia Zhang
UCLA
What is Sync

- Distributed applications require synchronized state
  - chatroom messaging
  - collaborative editing
  - routing protocol

- Key Idea
  - Reconcile *set difference* for a dataset shared among multiple parties
  - Each party has a local *state (view)* of *shared dataset*
  - Goal: All parties share the same *state* of the *shared dataset*
Sync in NDN

- **NDN Sync**
  - A way to implement data oriented multiparty communication protocol

- **Multiparty communication synchronization** of shared dataset
  - Define application-specific data units as items in a shared dataset
  - Synchronize namespace of data units

- **Sync** provides synchronization as a service to NDN applications
  - Keeps application up-to-date about newest dataset state
  - Individual applications fetches content based on need
Basic Functions of NDN Sync (Chatroom Example)

- **State Representation**
  - Namespace design
    - *how to name the chat messages*
  - State encoding
    - *how to encode each user’s shared dataset state*

![Diagram of state representation]

- C’s shared dataset state
  - Message 1 $\rightarrow$ /C/2
  - Message 2 $\rightarrow$ /C/3
  - Message 3 $\rightarrow$ /A/1

Encoded state
Basic Functions of NDN Sync (Chatroom Example)

- **State Representation**
  - Namespace design
    - *how to name the chat messages*
  - State encoding
    - *how to encode each user’s shared dataset state*

- **State change detection**
  - *discovering if any new chat data has been produced*
Basic Functions of NDN Sync (Chatroom Example)

- **State Representation**
  - Namespace design
    - *how to name the chat messages*
  - State encoding
    - *how to encode each user’s shared dataset state*

- **State change detection**
  - *discovering if any new chat data has been produced*

- **State difference identification**
  - *identify the difference in dataset state between nodes*

A’s encoded state  C’s encoded state
Basic Functions of NDN Sync (Chatroom Example)

- **State Representation**
  - Namespace design
    - *how to name the chat messages*
  - State encoding
    - *how to encode each user’s shared dataset state*

- **State change detection**
  - *discovering if any new chat data has been produced*

- **State difference identification**
  - *identify the difference in dataset state between nodes*

- **Fetching missing data**
  - *Receiver-driven data delivery reliability*
State Representation (Namespace Design)

● Sequential Naming
  ○ Data name: unique producer prefix + sequence number
  ○ Dataset namespace: set of [producer prefix + latest data sequence number]
  ○ Sequential naming provides efficient namespace representation

```
Shared
Dataset
State

Namespace
Representation

/A/1
... 
/A/111
/A/112

/B/1
... 
/B/126
/B/127

/C/1
... 
/C/190
/C/191

A: 112
B: 127
C: 191
```
State Encoding approach-1: State Digest

- **State Digest**
  - compress knowledge of dataset into crypto digest
  - hashes each producer's name prefix and latest sequence number
  - compare state digest to detect state inconsistency

*Example of State Digest*

```
hash of each [producer prefix + latest seq no]
```

Example of State Digest
State Encoding approach-2: Invertible Bloom Filter

- Invertible Bloom Filter (IBF)
  - Inserts hashes of each [stream prefix+latest seq no] into cells in IBF
  - IBF supports subtraction operation to identify state difference (IBF1-IBF2)

Example of IBF

Add Data d
(bit representation of d: 1010; h(1010) = 0100)

\[ h_1(d) \]
\[ h_2(d) \]
\[ h_3(d) \]

idSum: 1010 0000 1010 0000 0000 1010
hashSum: 0100 0000 0100 0000 0000 0100
Count: 1 0 1 0 0 1

Example of IBF
State Encoding approach-3: State Vector

- State Vector directly lists the [producer prefix : latest seq no] in a version vector
- Nodes can compare State Vector directly to resolve any state mismatch
- Do not have assumption on underlined connectivity

State Vector Example
State Change Detection

- **Sync Interest**
  - Contains sender’s encoded state
  - Periodically multicasted to advertise sender’s state and detect state change
  - Receiving nodes can identify state difference though encoded state comparison

- **Sync Reply**
  - Contains updated data names or sender’s state vector
MANET Challenge 1

- Intermittent connectivity with mobility
  - Connectivity may be lost quickly
- State divergence is the norm
  - Network partitioned into different clusters
  - Nodes accumulate different state updates

![Diagram of MANET network with intermittent connectivity and state divergence]
Reconcile State Divergence

- **State digest**
  - node fetches complete dataset state information

- **IBF**
  - can deduce difference in dataset namespace (IBF1-IBF2)
  - IBF size limits the amount of state difference which can be recovered (False Positive of IBF)
    - In case of large state divergence only part of the state difference can be decoded

- **State Vector**
  - Directly expresses dataset state
  - Can resolve any degree of state divergence
MANET Challenge 2

- Decoupling state and dataset synchronization
  - Sync provides synchronization as a service to NDN applications
    - State Sync: synchronize knowledge about the latest dataset
    - Dataset Sync: application decides which data to fetch
MANET Challenge 2 (Cont)

- Decoupling state and dataset synchronization causes excessive transmission
  - State and dataset mismatch caused by intermittent connectivity
  - Results in nodes continuously fetching none existent data
  - Mismatched state gets propagated further in the network

1. B didn’t fetch all of A’s data before connection broke
MANET Challenge 2 (Cont)

- Decoupling state and dataset synchronization causes excessive transmission
  - State and dataset mismatch caused by intermittent connectivity
  - Results in nodes continuously fetching nonexistent data
  - Mismatched state gets propagated further in the network
MANET Challenge 2 (Cont)

- Decoupling state and dataset synchronization causes excessive transmission
  - State and dataset mismatch caused by intermittent connectivity
  - Results in nodes continuously fetching nonexistent data
  - Mismatched state gets propagated further in the network

1. B didn’t fetch all of A’s data before connection broke

2. Mismatched state/dataset gets propagated

3. C would keep requesting B for A:4 which B doesn’t have
MANET Challenge 3

- State Change Detection
  - Periodic transmission of Sync Interest creates unnecessary overhead in MANET
    - Nodes are unaware of the connectivity of neighbors
    - Nodes do not know if a node with newer state is reachable
    - Sync Interest carries encoded state which is costly to transmit frequently
  - Possible solutions
    - Using lightweight message for state change detection
State Change Detection (Layer 2 Beacons)

- Utilizing Layer 2 beacons (802.11 ad hoc)
  - Encoding state information (digest) into layer 2 frame
  - Detects neighbors with state difference
    - Trigger Sync Interest transmission
  - Issues
    - Requires interface for Network/MAC layer exchange
      - Existing MAC layer design/implementation is unusable by higher layer
Possible Solutions

● Couple state sync with data sync
  ○ nodes send State Vector based on its actual dataset
  ○ nodes fetch data pieces in sequence, to support sequential namespace representation

● Increase data availability
  ○ e.g. deploy distributed repos in the network
Conclusion

- NDN Sync facilitates distributed multiparty applications
- New insight from trying sync in MANET
  - State Vector offers resilient state divergence recovery under adverse conditions
  - Decoupling State and Dataset Sync causes large amount of excessive Interest
  - Simple, old soft-state works: Periodic notification of state offers most resiliency under adverse condition
- Existing MAC layer design/implementation is unusable by higher layer; a redesign may greatly improve the overall network performance
Thank You
Q&A