Distributed Dataset Synchronization in Named Data Networking

Wentao Shang

Final defense

06/01/2017

Research Problem

- Distributed applications require efficient support for multi-party communication
  - Multiple nodes publish and share data

- Named Data Networking (NDN) enables new ways to support multi-party communication through dataset synchronization (sync)
  - Leveraging data-centric network architecture
  - Without centralized server

State of Affairs

- A number of sync protocols have been developed since the start of the NDN project
  - CCNx 0.8 Sync; ChronoSync; iSync; CCNx 1.0 Sync; RoundSync; pSync

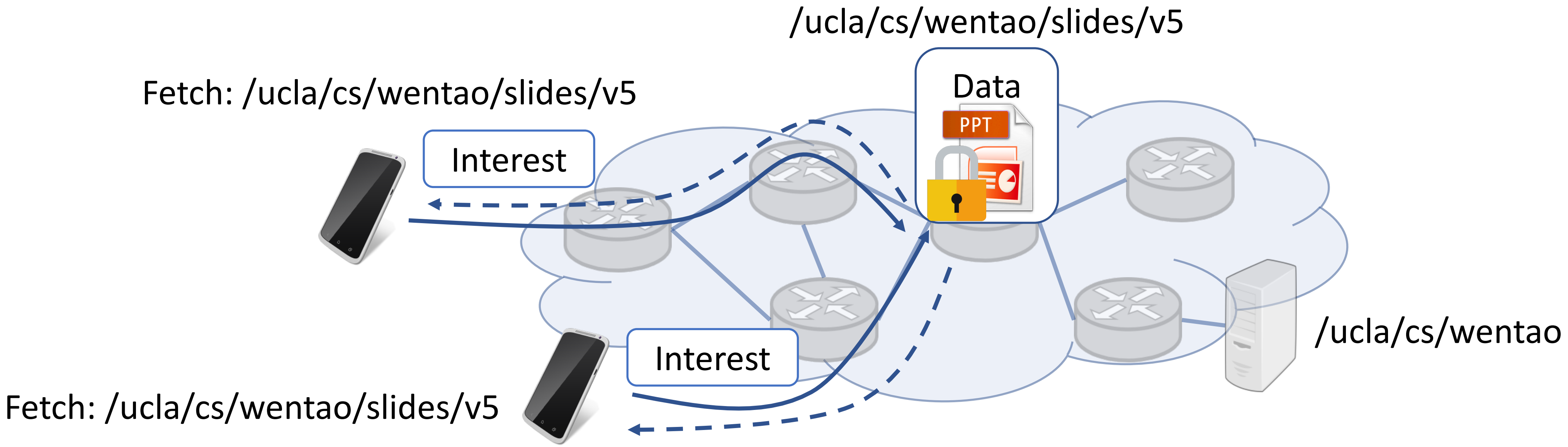
- A number of existing NDN applications run on top of sync
  - CCNx repo: replicated data storage
  - ChronoShare: distributed file sharing
  - ChronoChat: server-less group chat
  - NLSR: link-state routing protocol
  - NDN-RTC: group conferencing
  - Distributed data catalog
  - IoT pub-sub system

Research Objectives

- Understanding the design space of NDN sync
  - Systematic examination of all the existing NDN sync protocols
- Designing a new sync protocol
  - Learning from the design tradeoffs in the existing protocols
  - Supporting new functions not offered by the existing works
  - Applying methods developed in the distributed systems area

NDN Overview

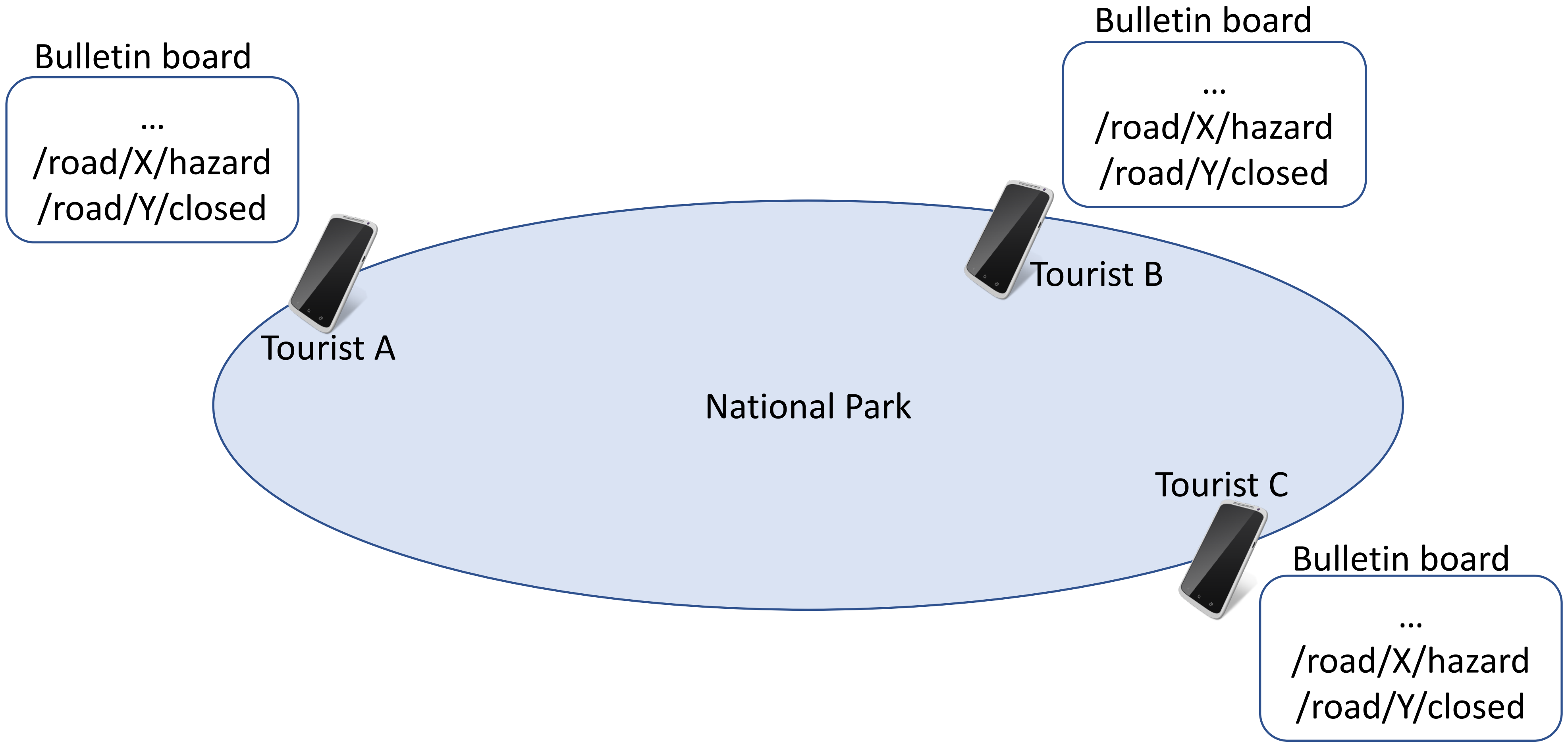
- Unique and secured binding between name and content
  - Name data, and secure data directly
- Name-based data retrieval
  - Stateful Interest-Data exchange
  - Secured data enables in-network storage


/ucla/cs/wentao/slides/v5
Fetch: /ucla/cs/wentao/slides/v5
Data
PPT
Interest
Interest
/ucla/cs/wentao
Fetch: /ucla/cs/wentao/slides/v5

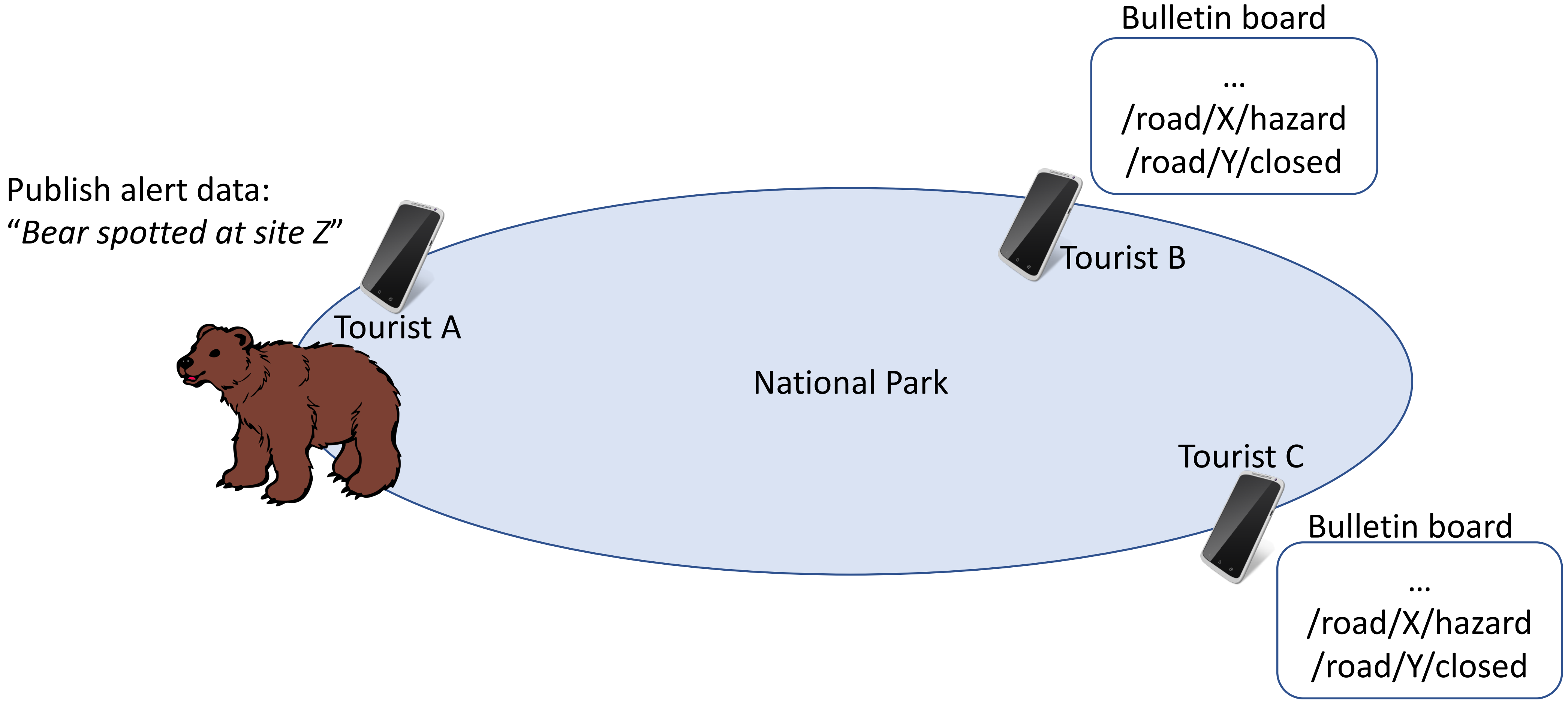
NDN Sync for Multi-Party Communication

- Enable a group of nodes to publish and consume data in a shared dataset
  - Maintain a consistent state of the dataset among the participants
- NDN provides unique binding between name and data → Synchronizing dataset = synchronizing the namespace of the dataset
- Fully utilize NDN’s data-centric communication
  - In-network caching
  - Multicast data delivery

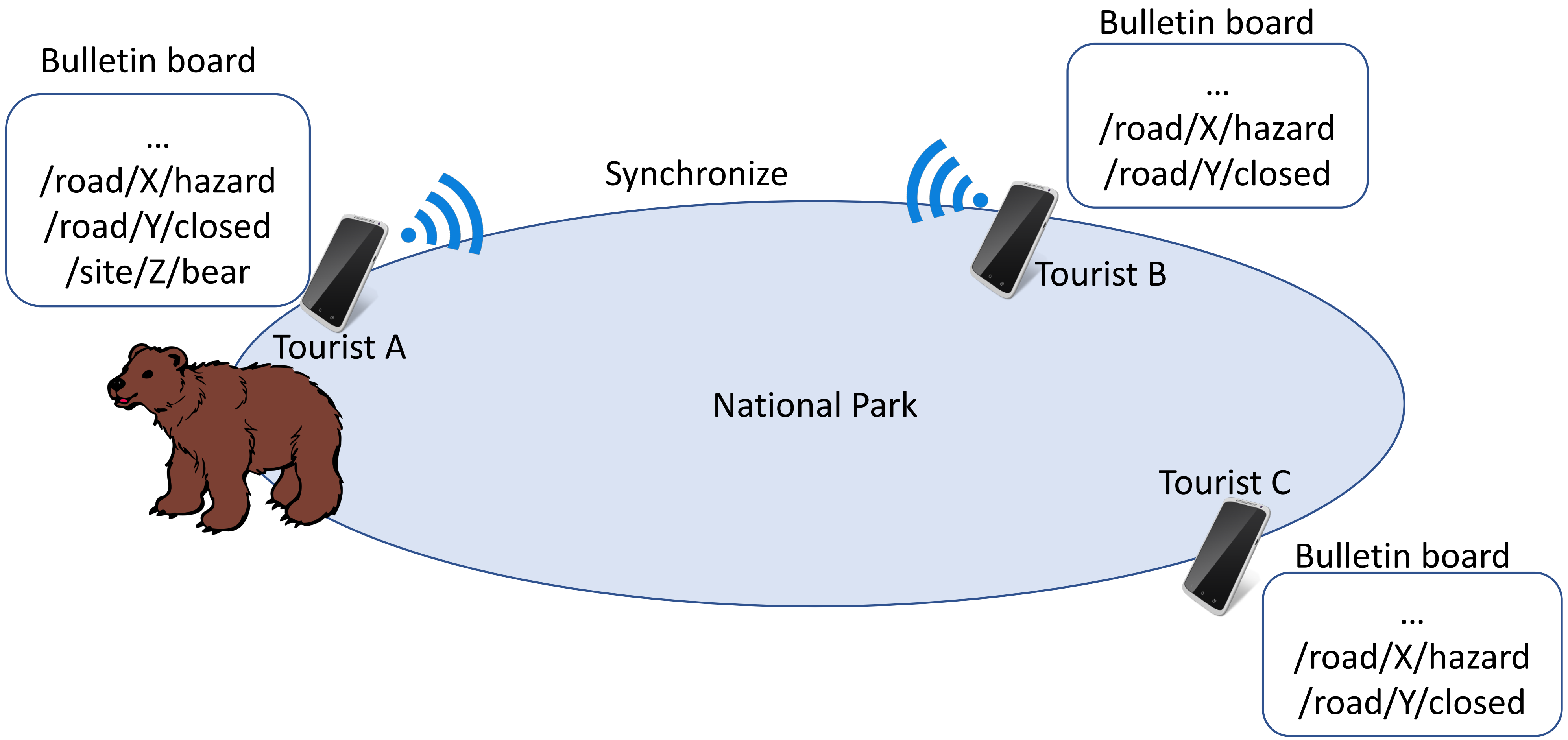
Sync in NDN


Bulletin board
…
/road/X/hazard
/road/Y/closed
Bulletin board
…
/road/X/hazard
/road/Y/closed
Tourist A
Tourist B
National Park
Tourist C
Bulletin board
…
/road/X/hazard
/road/Y/closed

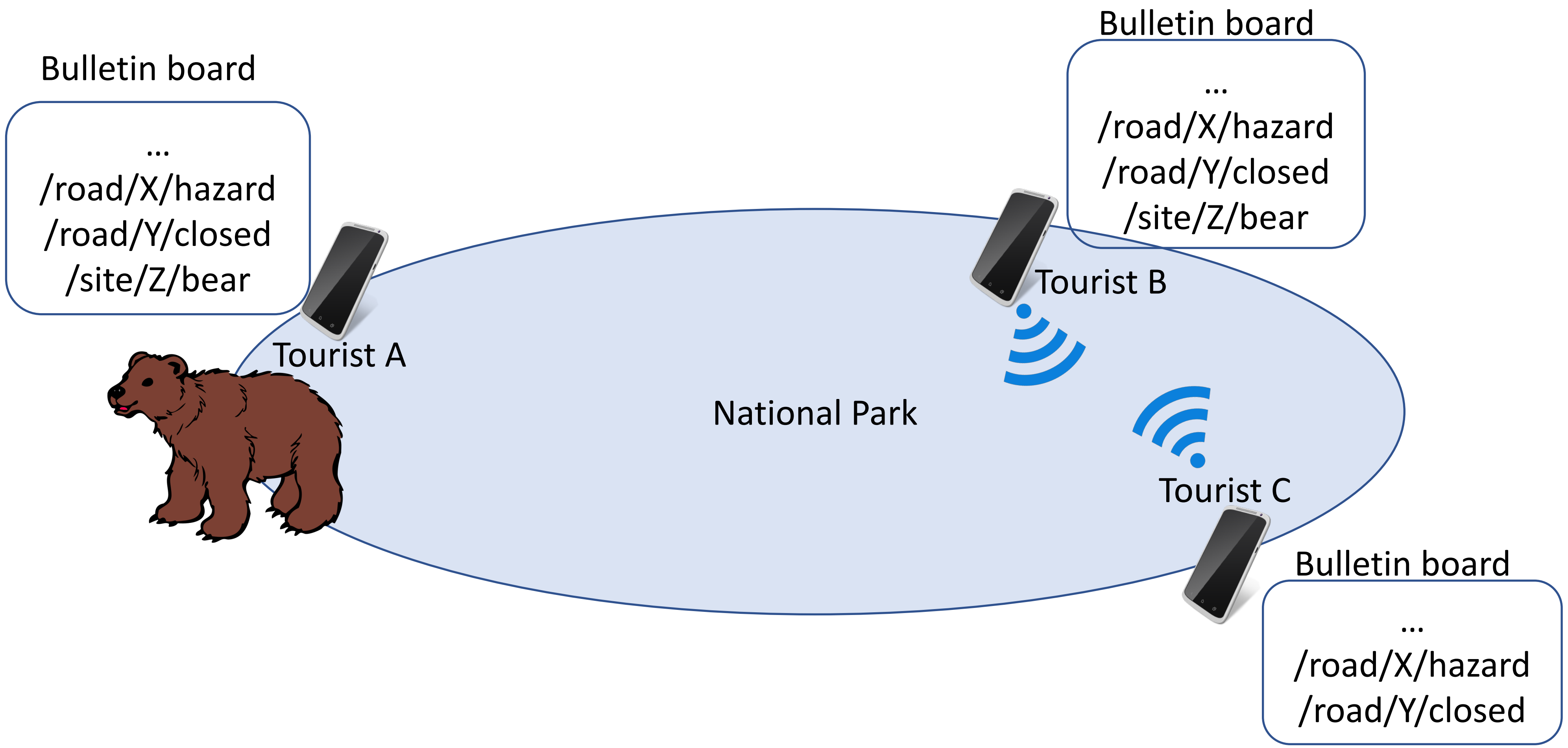
Sync in NDN


Publish alert data:
"Bear spotted at site Z"
Tourist A
Tourist B
Tourist C
National Park
Bulletin board
…
/road/X/hazard
/road/Y/closed
Bulletin board
…
/road/X/hazard
/road/Y/closed

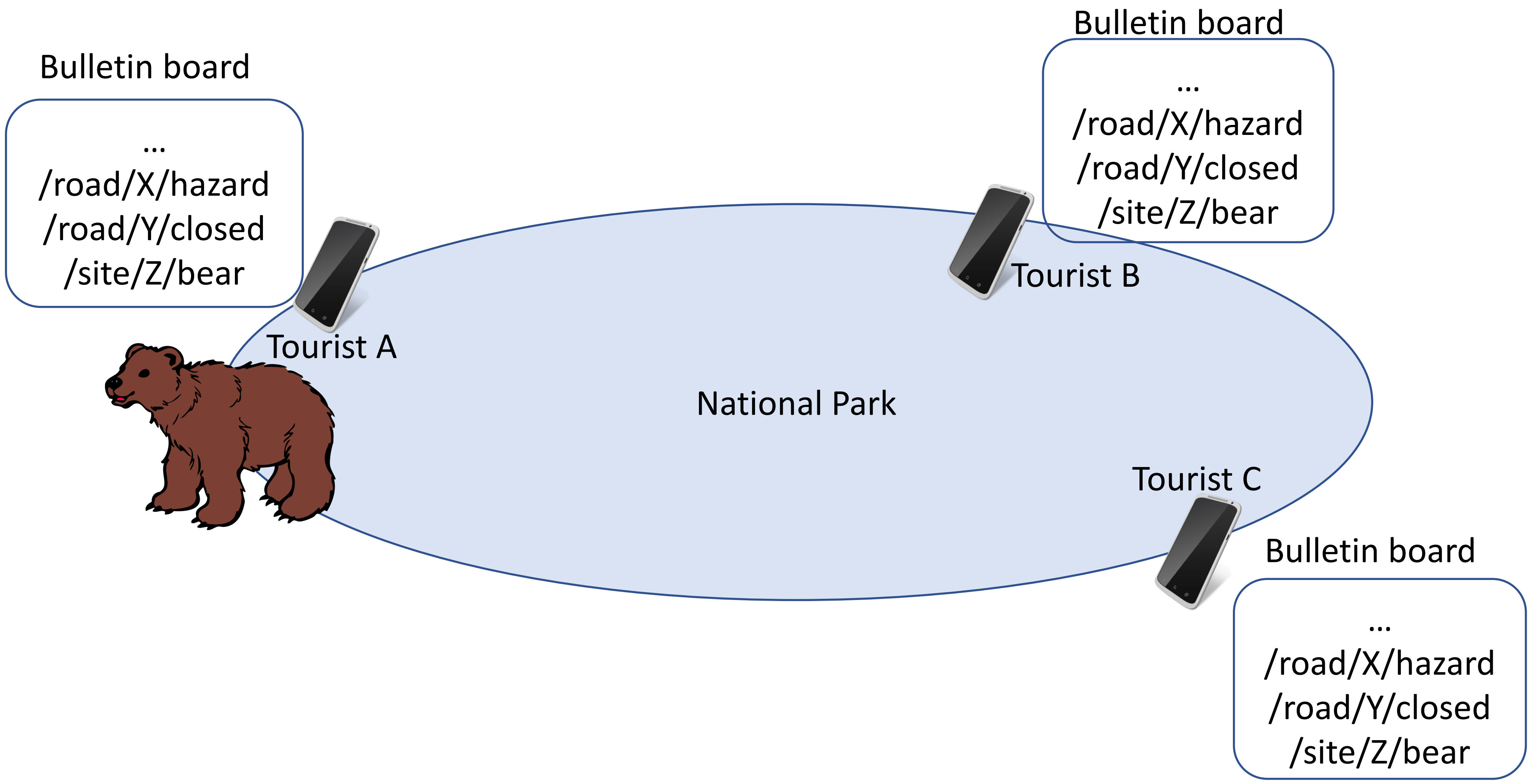
Sync in NDN


Bulletin board
…
/road/X/hazard
/road/Y/closed
/site/Z/bear
Tourist A
Synchronize
Bulletin board
…
/road/X/hazard
/road/Y/closed
Tourist B
National Park
Tourist C
Bulletin board
…
/road/X/hazard
/road/Y/closed

Sync in NDN

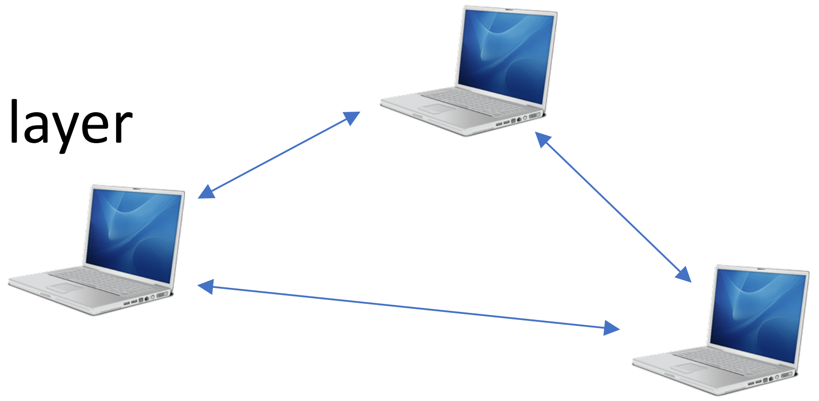

Bulletin board
…
/road/X/hazard
/road/Y/closed
/site/Z/bear
Tourist A
Bulletin board
…
/road/X/hazard
/road/Y/closed
/site/Z/bear
Tourist B
National Park
Tourist C
Bulletin board
…
/road/X/hazard
/road/Y/closed

Sync in NDN

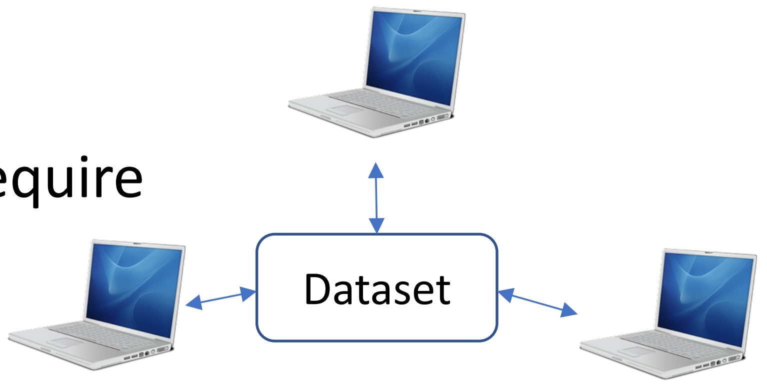

Bulletin board
…
/road/X/hazard
/road/Y/closed
/site/Z/bear
Tourist A
Bulletin board
…
/road/X/hazard
/road/Y/closed
/site/Z/bear
Tourist B
National Park
Tourist C
Bulletin board
…
/road/X/hazard
/road/Y/closed
/site/Z/bear

Comparing NDN Sync with Today's Data Synchronization Solutions

- Traditional Synchronization with TCP/IP networking
  - Network provides point-to-point communication
  - Dataset synchronization achieved at the application layer



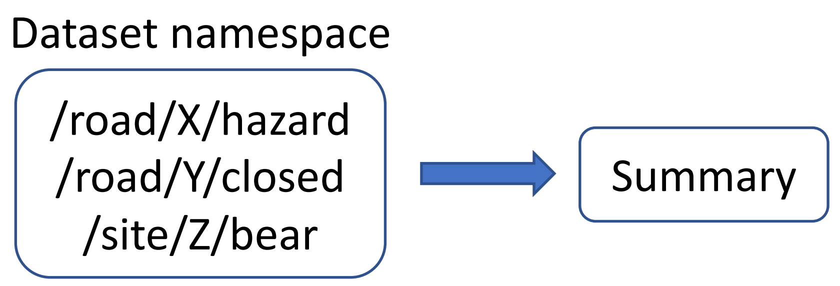
- Sync in NDN
  - Network provides data-centric communication
  - Sync protocol provides data transport service for the application
  - Because of data-centric nature, NDN sync does not require all parties connected to each other all the time



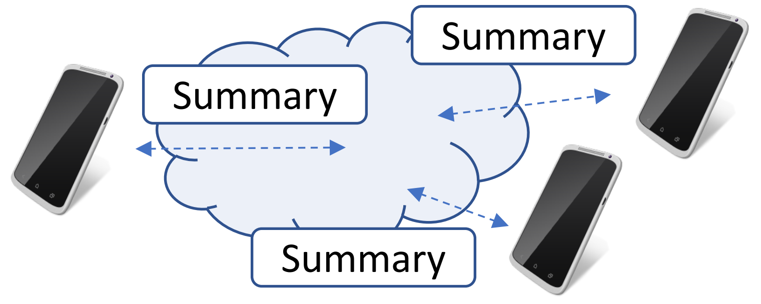
Design Space of NDN Sync Protocols

Common Sync Protocol Framework

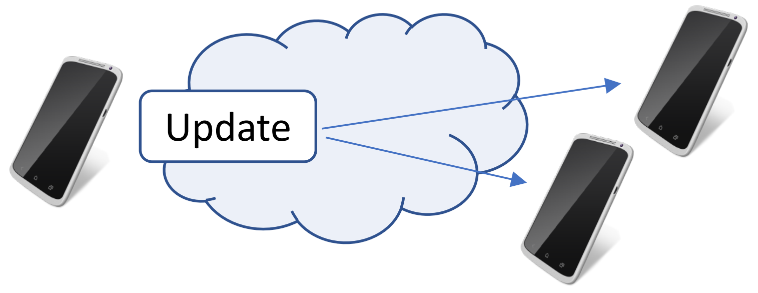
Generate a concise summary of the dataset namespace to be communicated between nodes



Detect and reconcile inconsistency by exchanging the summary periodically



(Optionally) support quick notification to other nodes when publishing new data



Key Design Aspects

- Dataset naming
  - How to name data items in the shared dataset
- Namespace representation
  - How to provide an efficient summary of namespace
- State synchronization mechanism
  - How to make nodes learn about changes ASAP
  - How to detect and reconcile inconsistency caused by various factors

Design Choices in Dataset Naming

- Sync protocol synchronizes application data names directly
  - CCNx 0.8 Sync; iSync; CCNx 1.0 Sync

```
/road/X/hazard
/road/Y/closed
/site/Z/bear
```

- Sync protocol names data by each producer sequentially
  - Encapsulate application names if needed
  - ChronoSync; RoundSync; pSync

```
/TouristA/13:  {/site/Z/bear}
/TouristA/14:  {/site/W/alert}
/TouristB/55:  {/road/X/harzard}
/TouristB/56:  {/road/Y/closed}
```

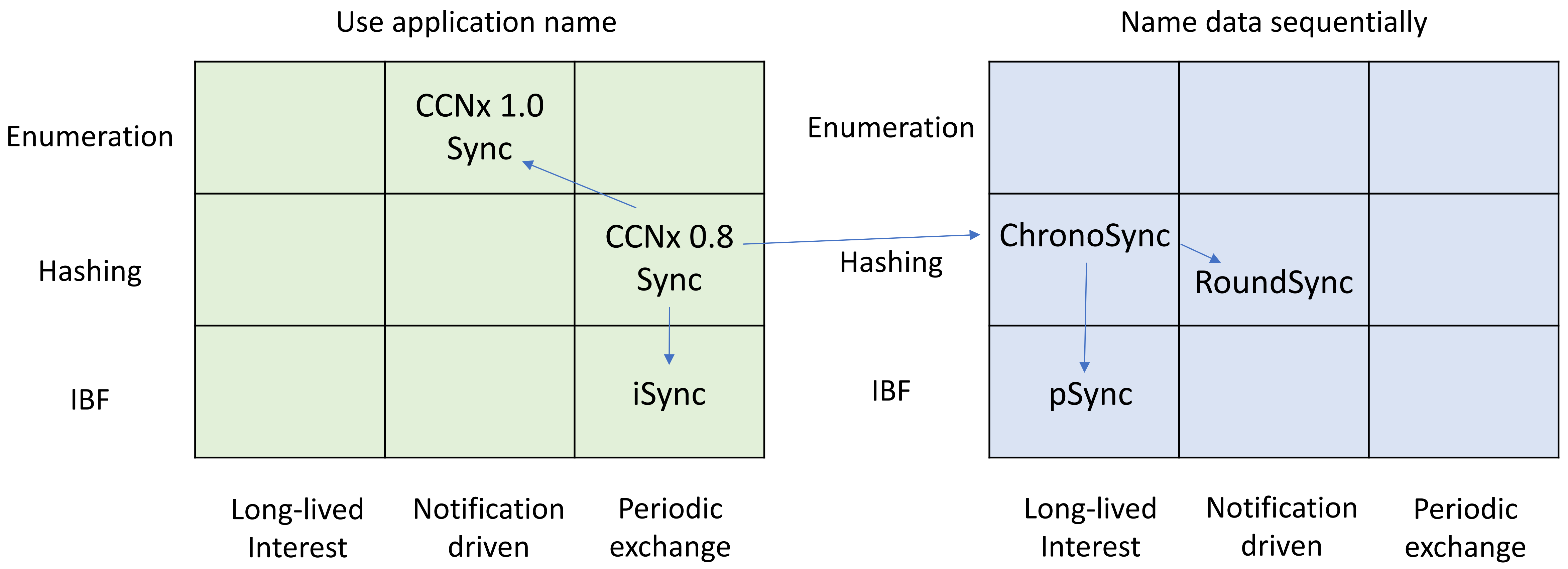
Design Choices in Namespace Representation

- Enumeration
  - Lossless compression in the namespace (or no compression)
  - CCNx 1.0 Sync
- Hashing
  - One-way compression of namespace
  - CCNx Sync; ChronoSync; RoundSync
- Invertible Bloom Filter (IBF)
  - Store and extract individual name hashes
  - iSync; pSync

Design Choices in State Synchronization Mechanism

- Long-lived Interest
  - Nodes maintain pending Interests in the network to solicit changes from others
  - ChronoSync; pSync;
- Notification-driven
  - Nodes inform others about new changes
  - CCNx 1.0 Sync; RoundSync;
- Periodic exchange of dataset summary
  - Nodes exchange their state summary periodically to detect and reconcile inconsistency
  - CCNx 0.8 Sync; iSync

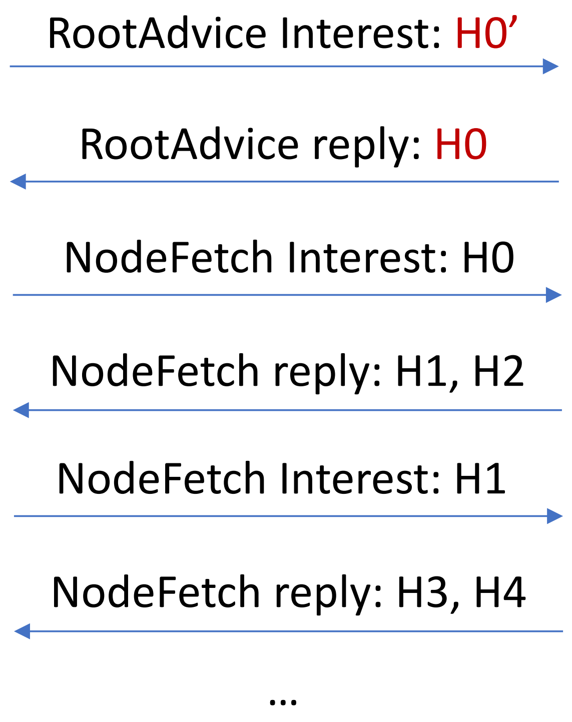
Evolution of Existing Sync Protocols

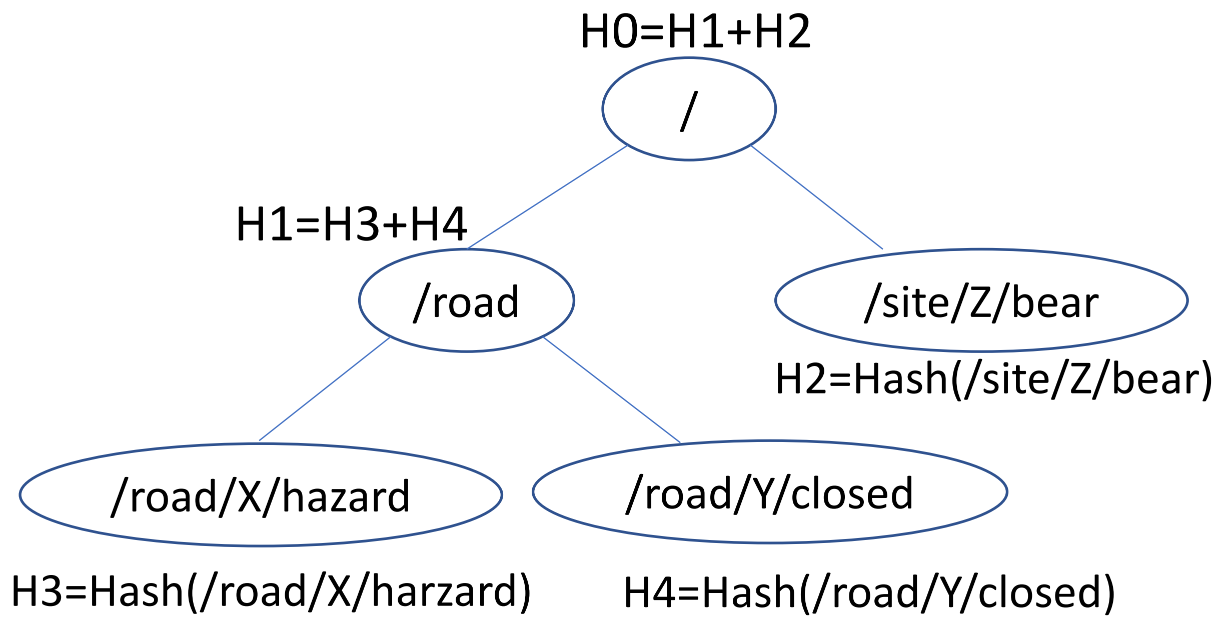


W. Shang et al., "A Survey of Distributed Dataset Synchronization in Named Data Networking", NDN-TR-0053, 2017

CCNx 0.8 Sync

- Summarize dataset namespace using combined hashes over tree structure
- Send Interest with root hash periodically to request different hash(es)
- Take multiple rounds to reconcile the differences



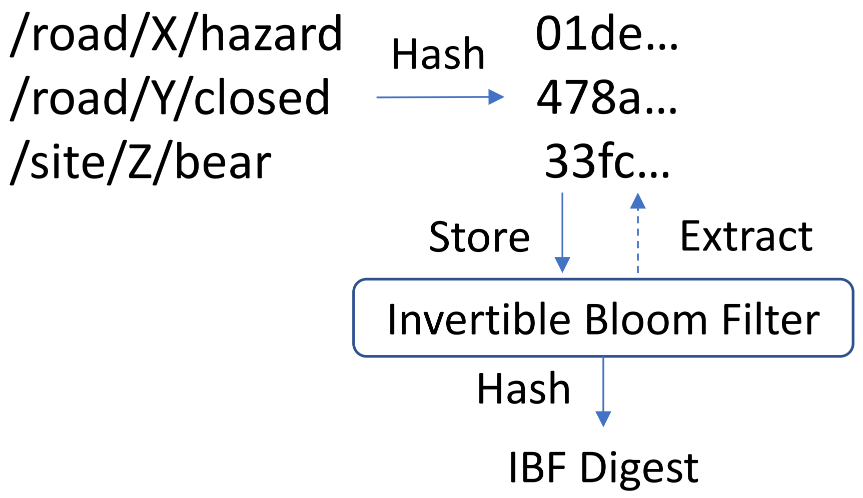


iSync: Improving CCNx 0.8 Sync

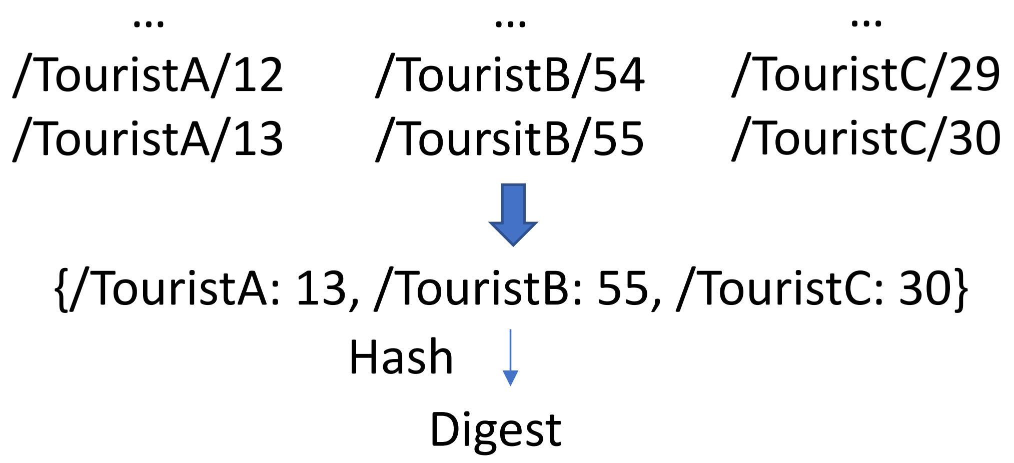
- Use Invertible Bloom Filter (IBF) to summarize the namespace
  - Detect differences using IBF subtraction

- Reduce the synchronization round-trip at the cost of larger namespace representation
  - Exchange only the IBF digest
  - Need extra RTT to retrieve the IBF content

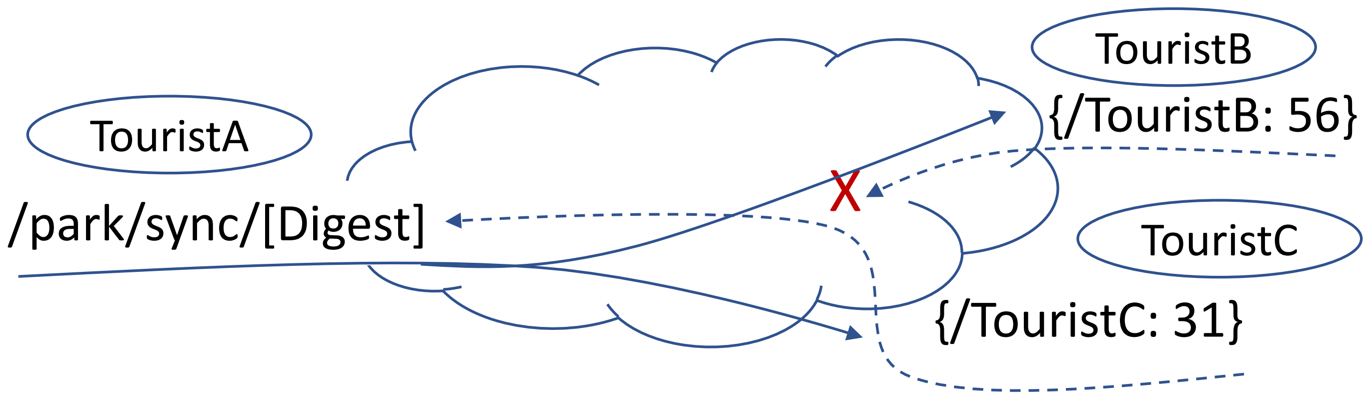
- Both CCNx 0.8 Sync and iSync synchronize via periodic exchange of state summary
  - Add additional delay to learning new data



ChronoSync

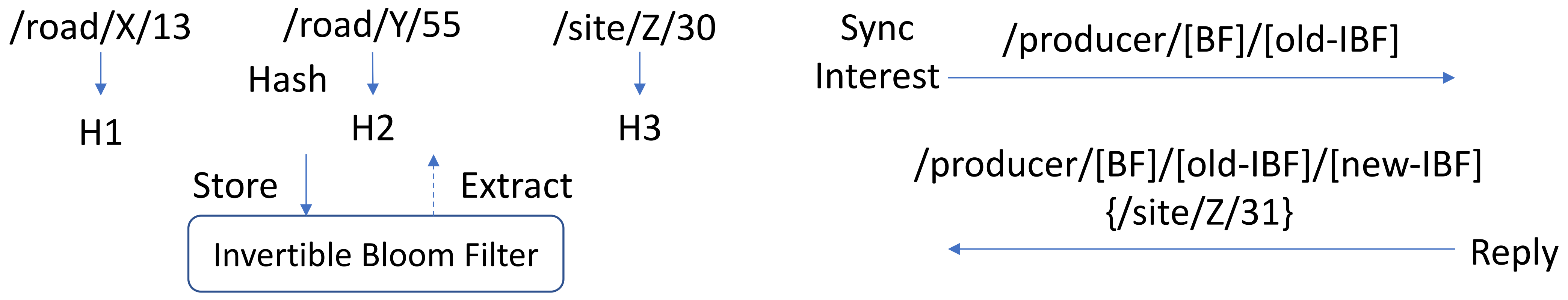


- Name data sequentially
- Summarize the namespace with a digest
- Maintain long-lived Interest in the network to wait for next update
  - Need “exclude filter” to retrieve simultaneous updates by multiple producers
- Interest carries state digest for inconsistency detection
  - Provide a “recovery” mechanism as last resort for repairing state conflict



pSync: Pub-sub over Sync

- Take the sequential name approach from ChronoSync, IBF as representation from iSync
  - IBF stores only each node's latest seq#, so size is determined by the group size
- Each consumer sends long-lived Interest with old IBF to request updates from a producer
  - IBF provides specific information about the consumer's state
  - Producer can reply with new data names directly



Other Sync Protocols

- CCNx 1.0 Sync: another fix to CCNx Sync
  - Enumerate data names in a manifest file
  - Broadcast manifest digest when publishing new data

- RoundSync: a revision to ChronoSync
  - Reduce but not eliminate the simultaneous publishing problem

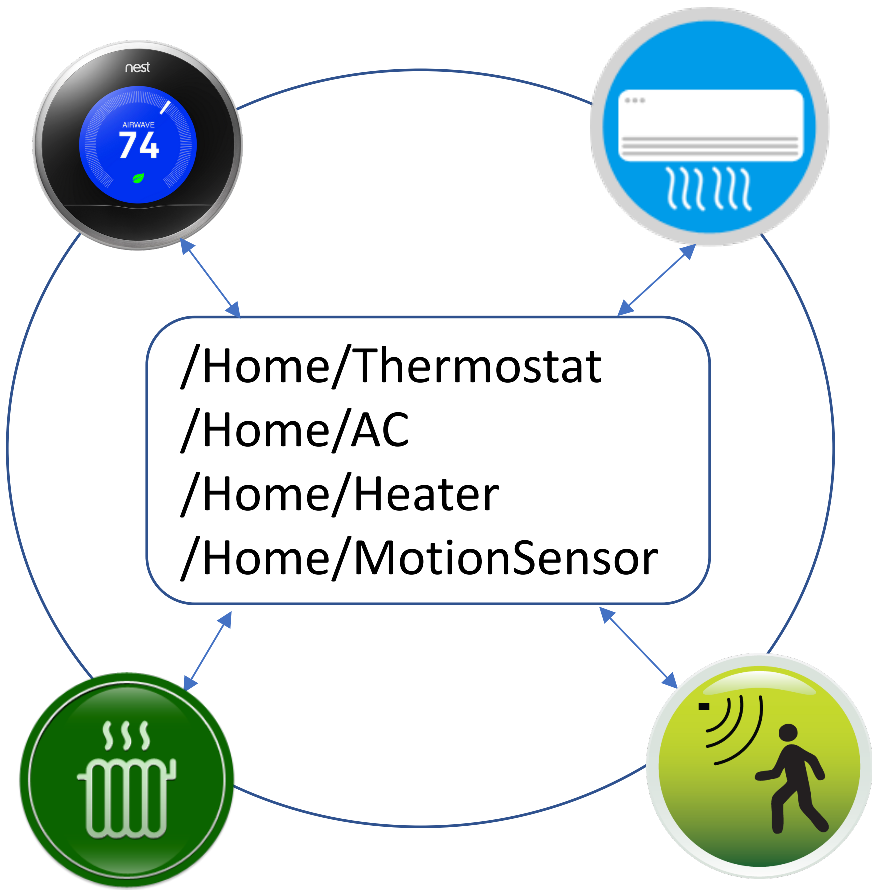
Lessons Learned

- Allowing sync protocol to name data sequentially simplifies the design
  - Only need to synchronize the latest sequence numbers
- Notifications should carry specific update information
  - So that recipients can fetch new data directly, without further exchange to identify the new data
- Avoid using long-lived Interest to fetch new updates
  - A long-lived Interest cannot fetch multiple data produced at the same time
  - Long-lived Interests add burden to network in maintaining Interest path state

VectorSync Protocol

Synchronization with Managed Group

- Many distributed applications require explicit group membership management
- Examples:
  - Resource discovery in IoT networks
  - Routing protocol
- Existing sync protocols do not support membership management
  - Difficult to remove departed nodes from the dataset state

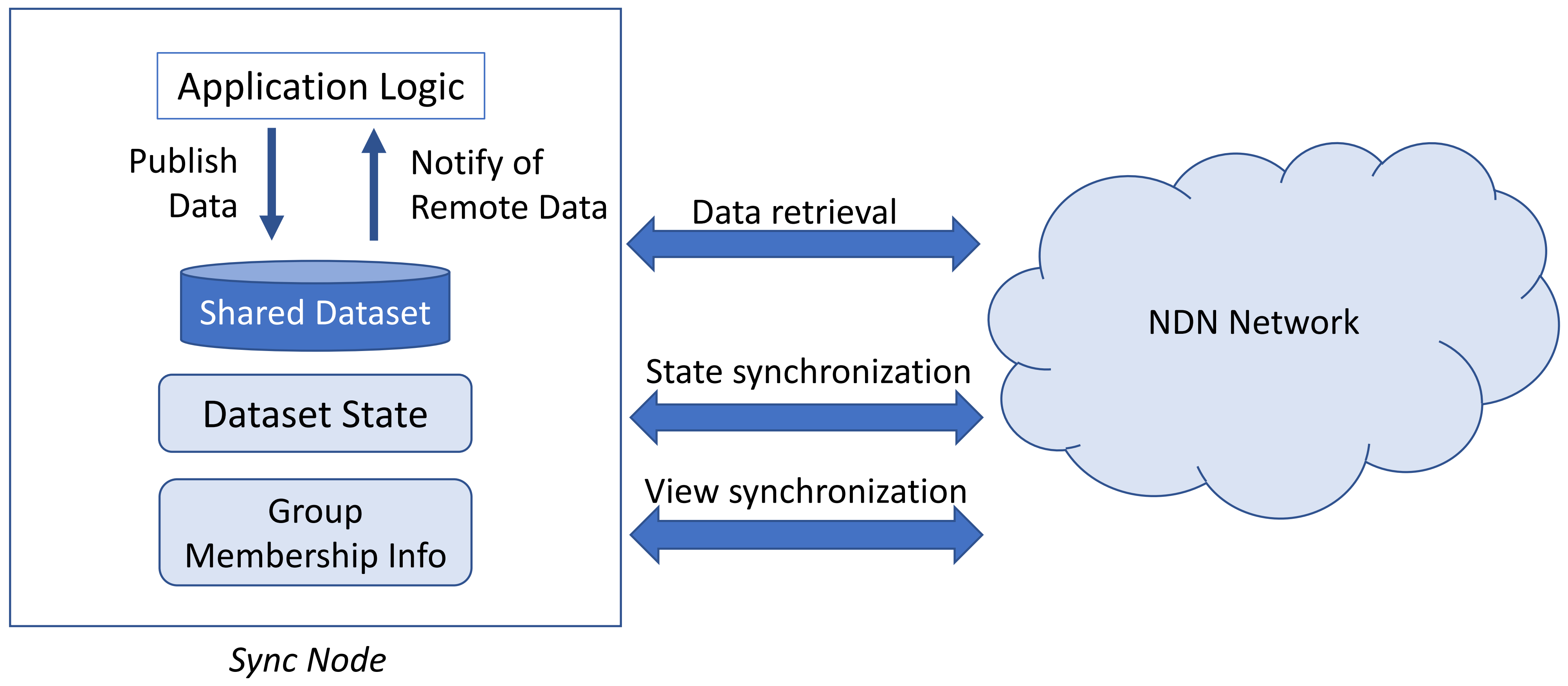


VectorSync Design Highlights

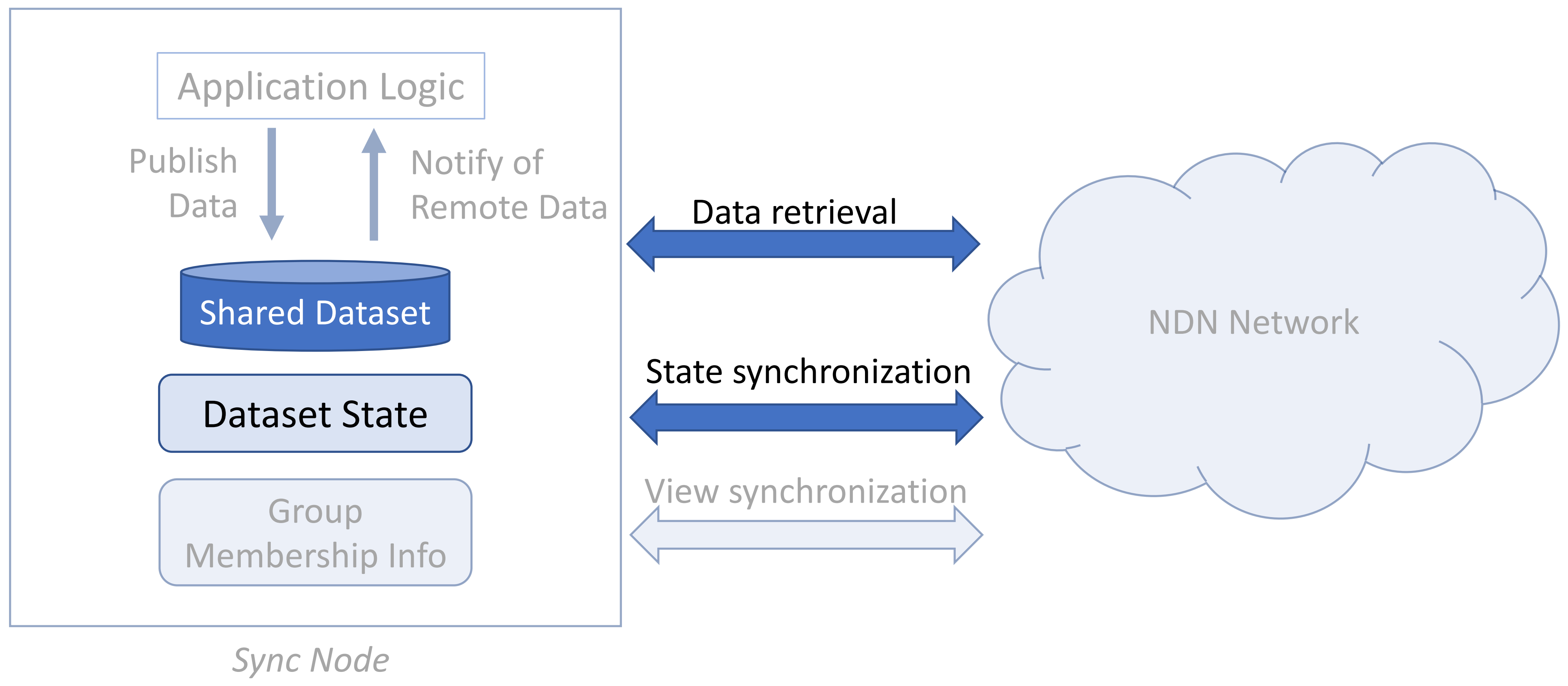
- Maintain a consistently ordered list of group participants (called a view) among all participating nodes
- Utilize a leader-driven process to synchronize the view among all nodes
- Leverage sequential dataset naming to synchronize the dataset using *version vector*
- Adopt notification-driven synchronization with specific update info

D. Parker et al., “Detection of Mutual Inconsistency in Distributed Systems”, 1983

VectorSync Overview



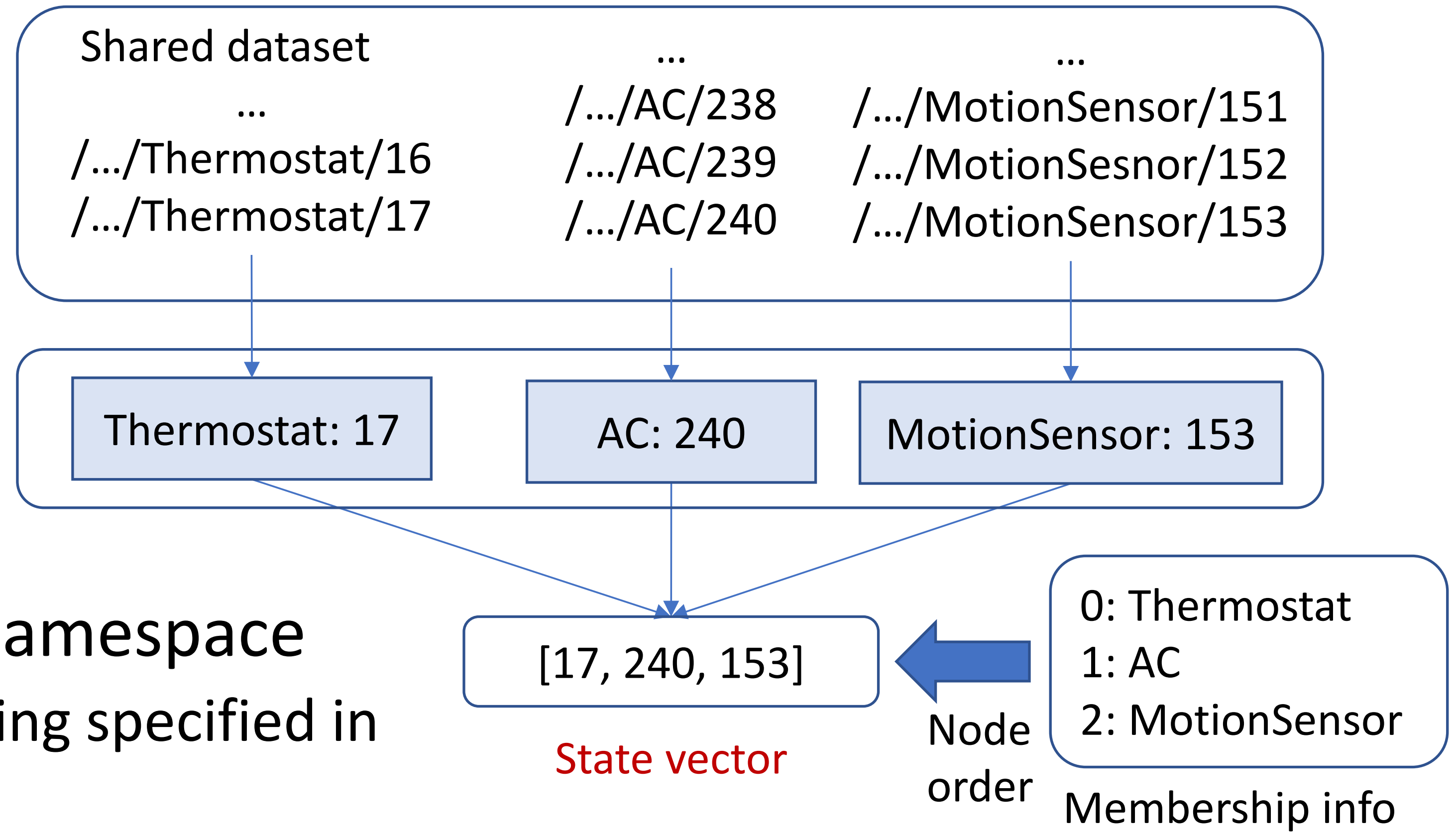
VectorSync Overview



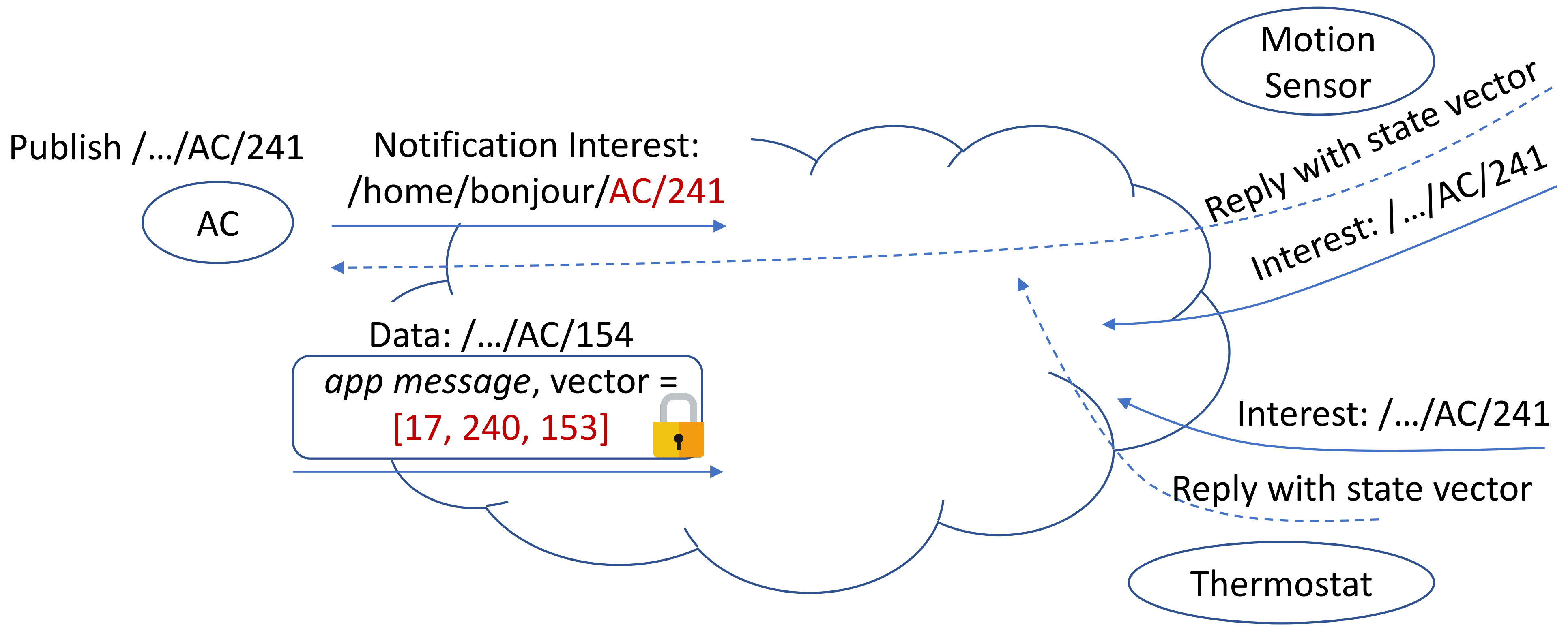
Dataset Namespace

- Sequential data naming
  - Using sequence numbers

- Vector representation of namespace
  - Producer names and ordering specified in the membership info

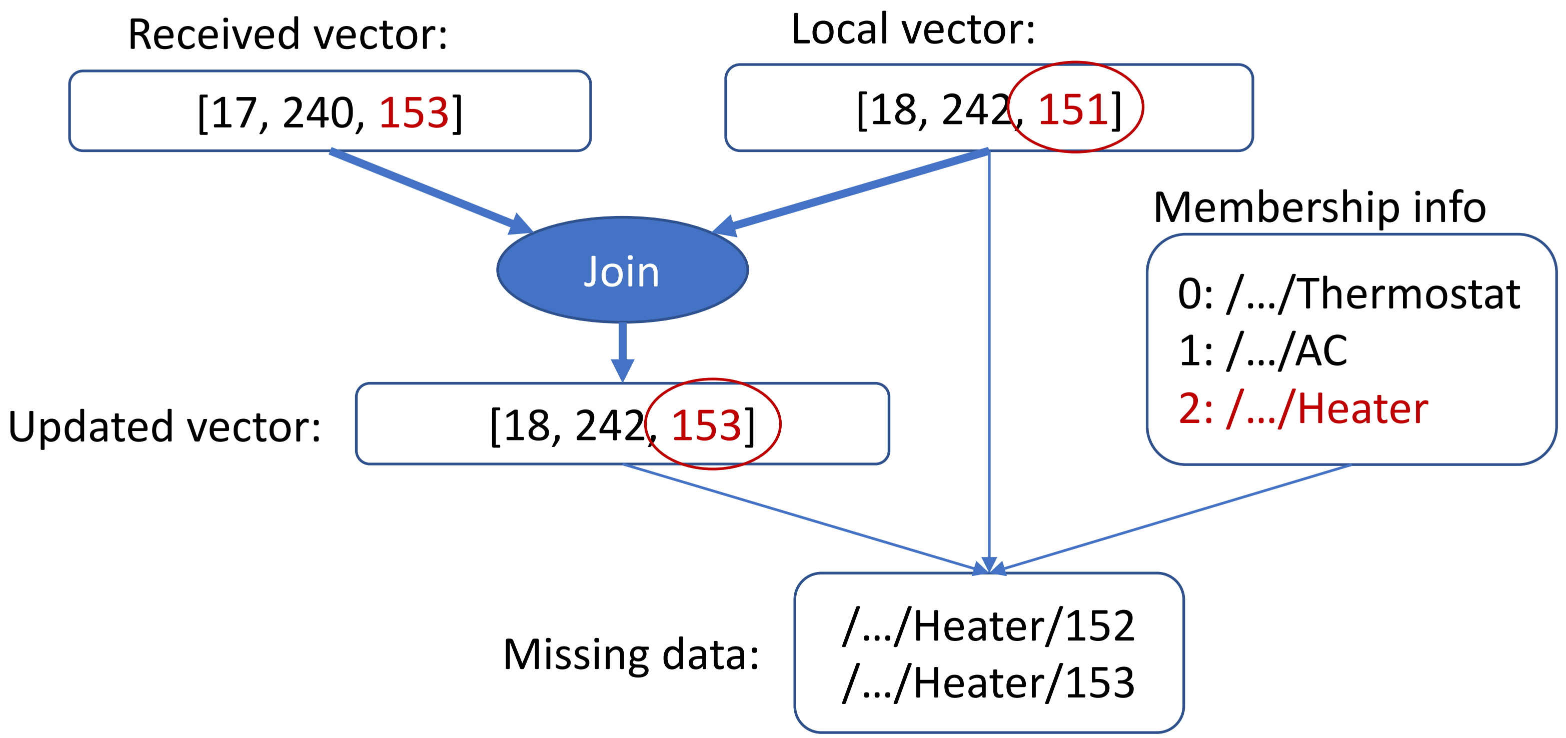


Publishing and Synchronizing Data



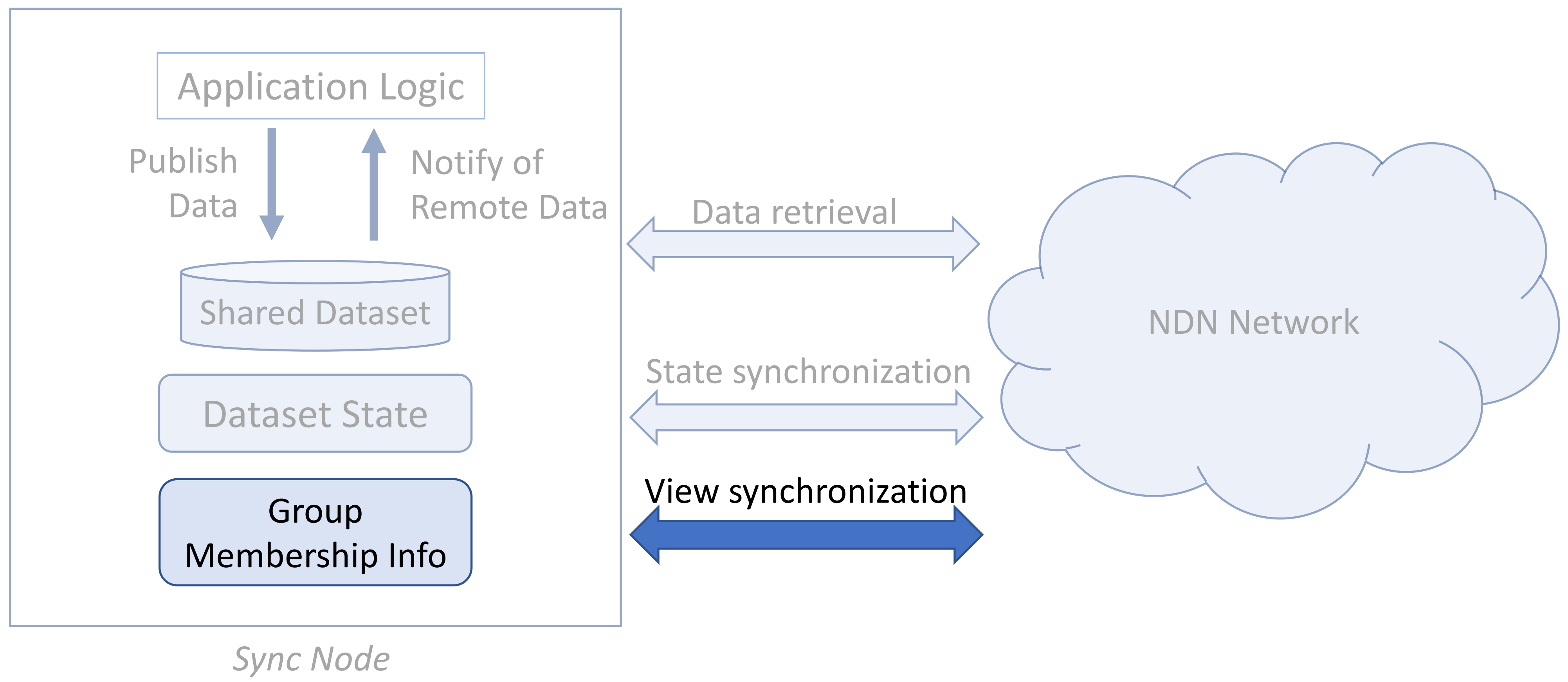
- Multicast notification Interest carries explicit information about changes
- Node data carries full state vector of the publisher, provides causal ordering

Detecting and Reconciling Inconsistency



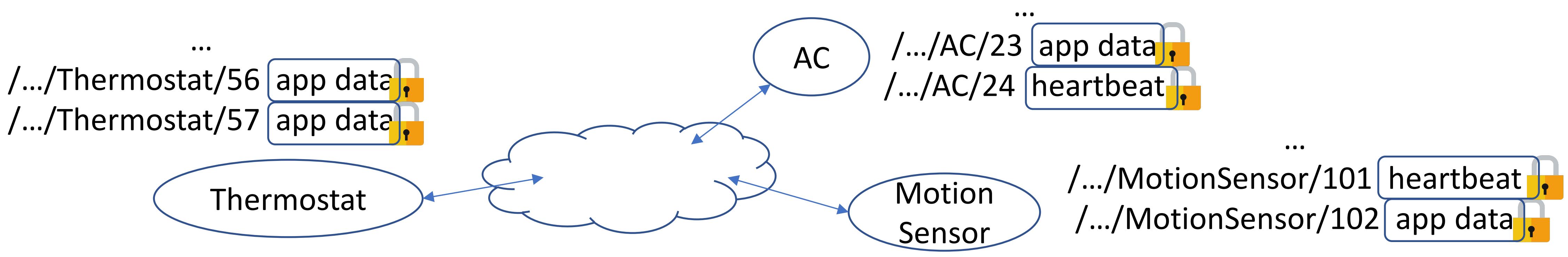
Update the local vector with the result of Join and retrieve missing data

VectorSync Overview



Soft-state Membership

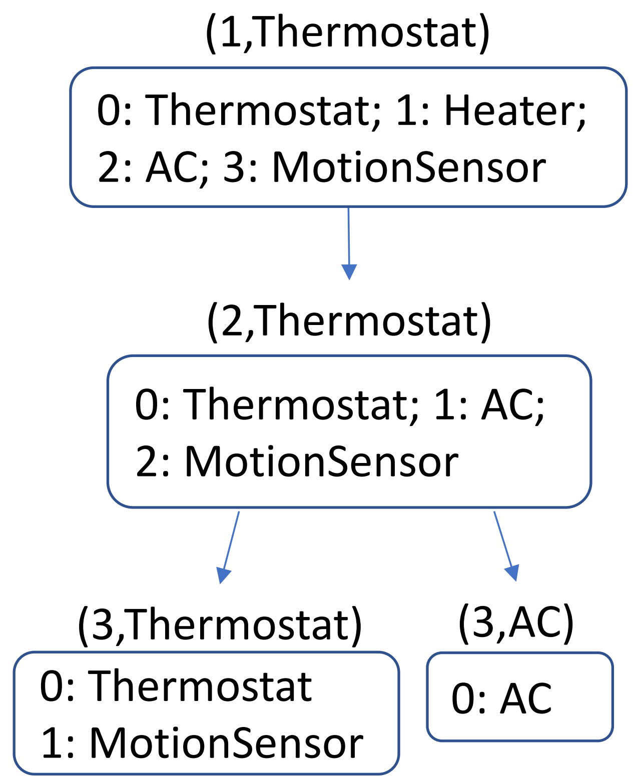
- Nodes refresh their membership by publishing data (in the dataset)
  - Authenticated assertion of existence
- When application is idle, node publishes "heartbeat" data periodically
  - Enable periodic exchange of state vectors
- A node is considered "gone" if no data received from it for certain amount of time
  - Heartbeat period and timeout values decided by application



…
/…/Thermostat/56 app data
/…/Thermostat/57 app data
Thermostat
AC
…
/…/AC/23 app data
/…/AC/24 heartbeat
Motion Sensor
…
/…/MotionSensor/101 heartbeat
/…/MotionSensor/102 app data

Leader-driven Membership Management

- Nodes select a leader to manage the group
- Leader defines and publishes its view of the group
  - Other nodes follow the leader's view
- Leader monitors the group and creates new view when the membership changes
- Views are named sequentially using view number
  - Leader increases the number when creating new view
- Upon network partition, each partition may select its own leader which creates its own view
  - View ID = (view number, leader name)

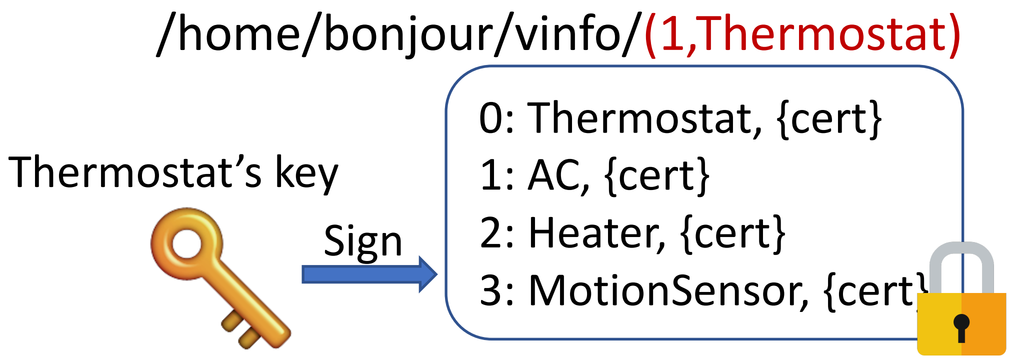

(1,Thermostat)
0: Thermostat; 1: Heater; 2: AC; 3: MotionSensor
(2,Thermostat)
0: Thermostat; 1: AC; 2: MotionSensor
(3,Thermostat)
0: Thermostat 1: MotionSensor
(3,AC)
0: AC

Selecting a leader

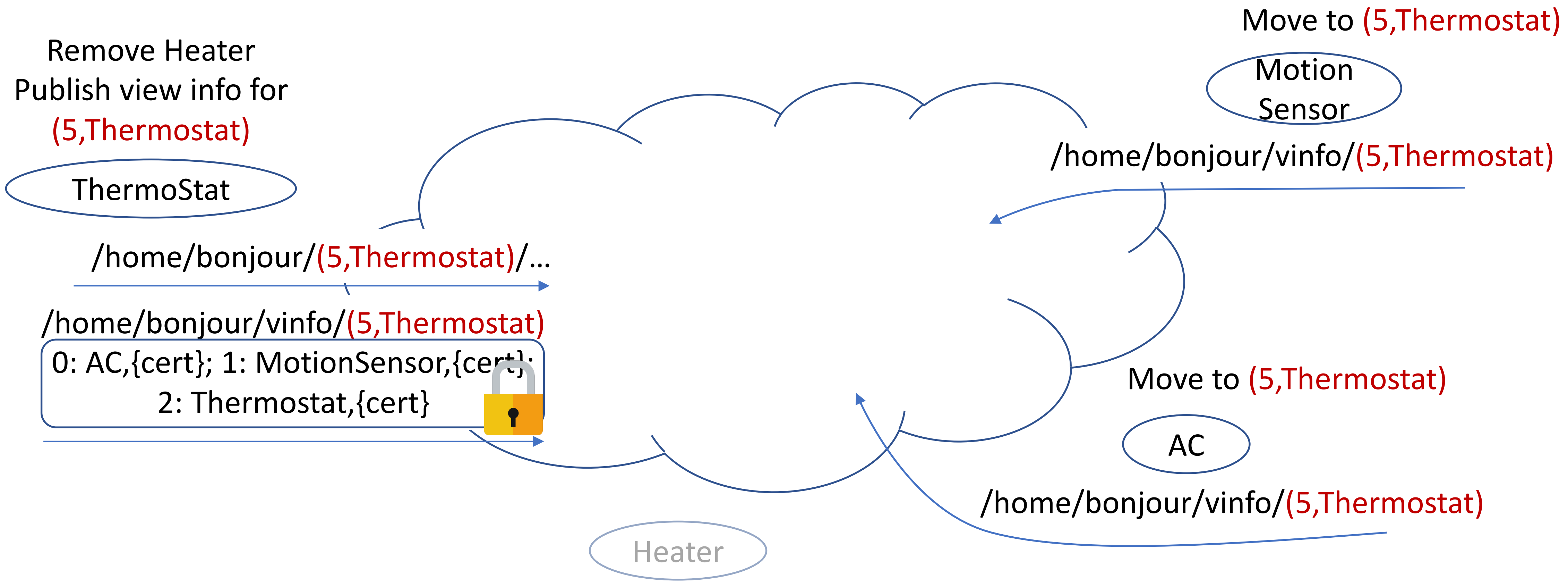
- If the current leader has left, the remaining nodes compete to become the next leader via *random leader selection*
    - When a node detects leader departure, it starts a random wait timer
    - After the timer goes off, the node declares itself the leader and creates a new view
    - If it notices a new view before the timer goes off, it cancels the timer and join the new view
- Other leader selection mechanism can also be used
    - Using pre-configured preference list

Synchronizing the View

- Leader signs and publishes view info as data when creating new view
  - Name: /[multicast-prefix]/vinfo/[view-id]
  - Contains names and certificates of the members
- View ID carried in all notification Interests
- Node fetches the view info if noticing a larger view number
  - Join the new view after receiving the view info
- Node keeps publishing data in its current view before the view synchronization finishes

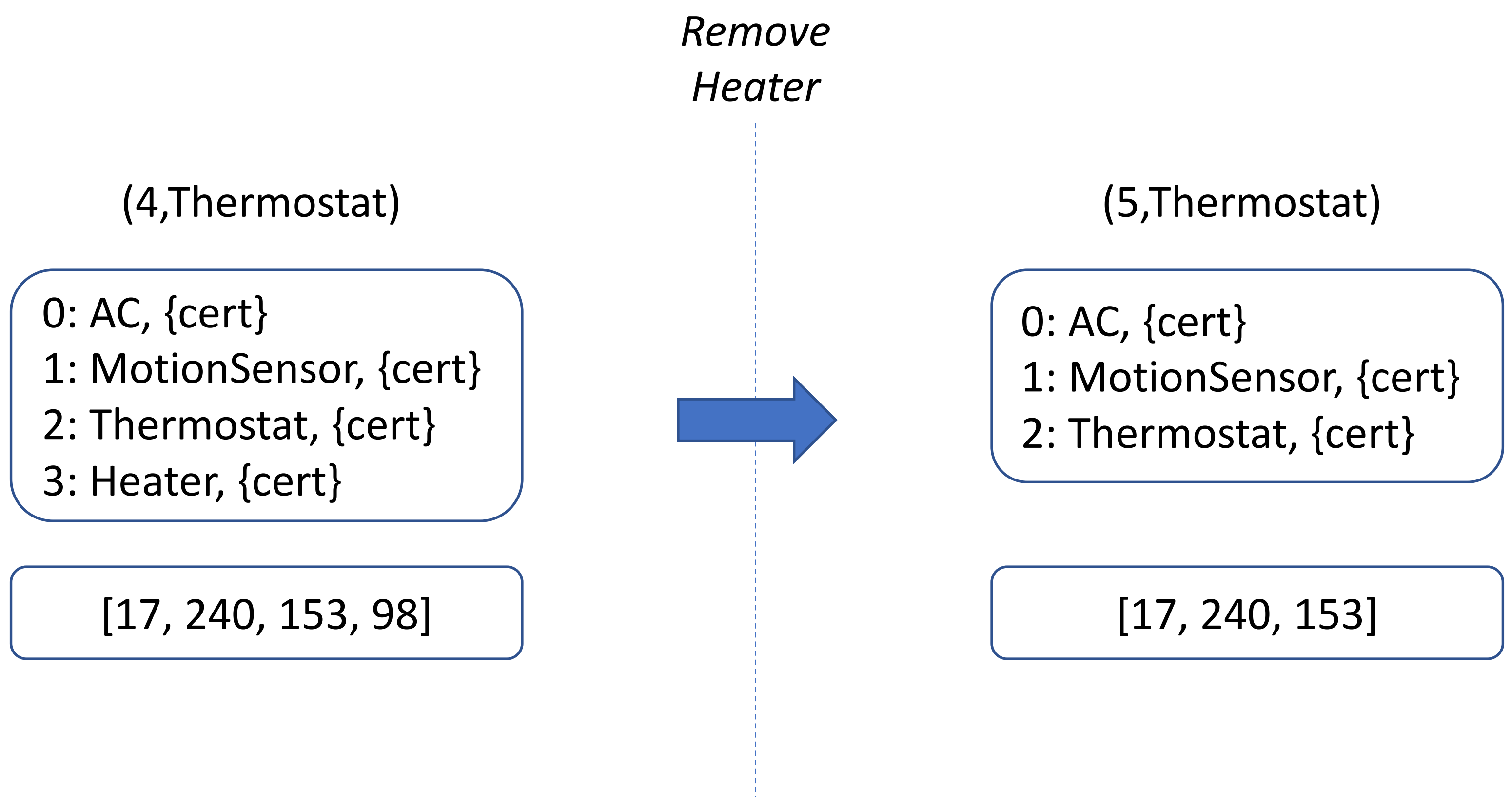


View Synchronization with Single Leader

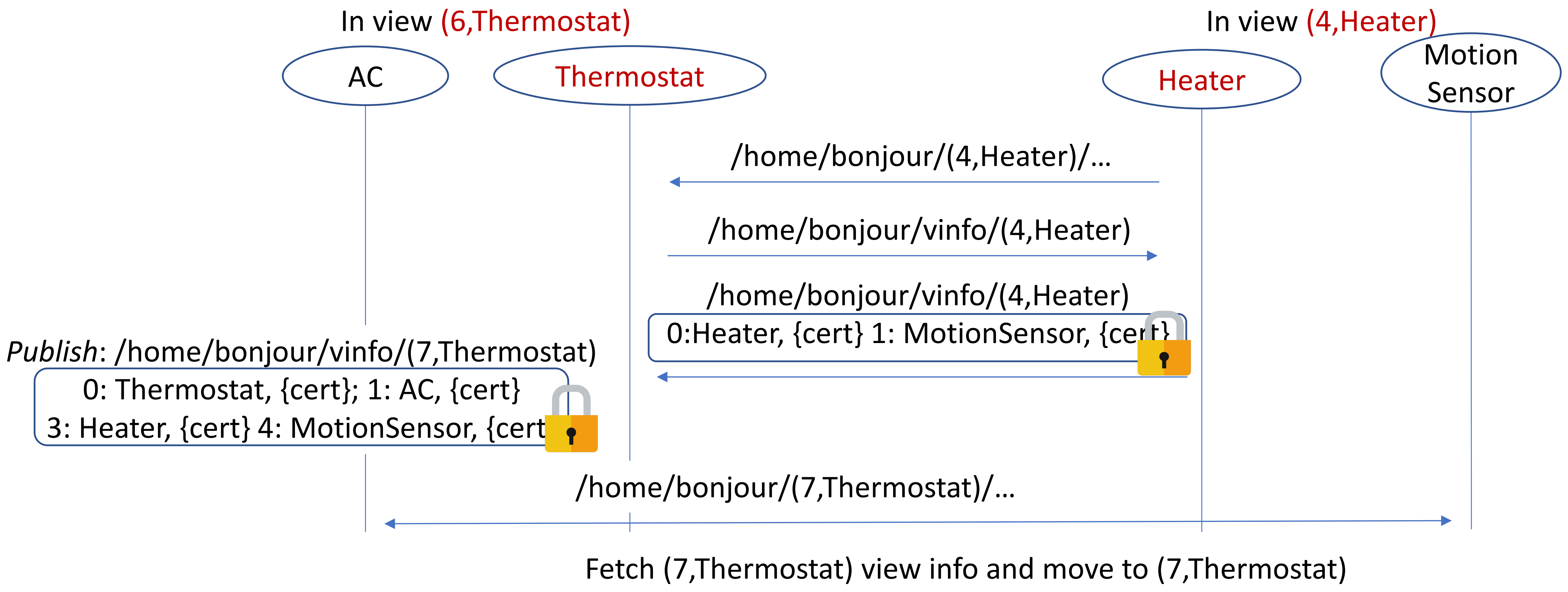


View (4,Thermostat): {0:AC; 1:MotionSensor; 2:Thermostat; 3:Heater}
Remove Heater
Publish view info for
(5,Thermostat)
ThermoStat
/home/bonjour/(5,Thermostat)/…
/home/bonjour/vinfo/(5,Thermostat)
0: AC,{cert}; 1: MotionSensor,{cert};
2: Thermostat,{cert}
Move to (5,Thermostat)
Motion Sensor
/home/bonjour/vinfo/(5,Thermostat)
Move to (5,Thermostat)
AC
/home/bonjour/vinfo/(5,Thermostat)
Heater

Updating State After Membership Change

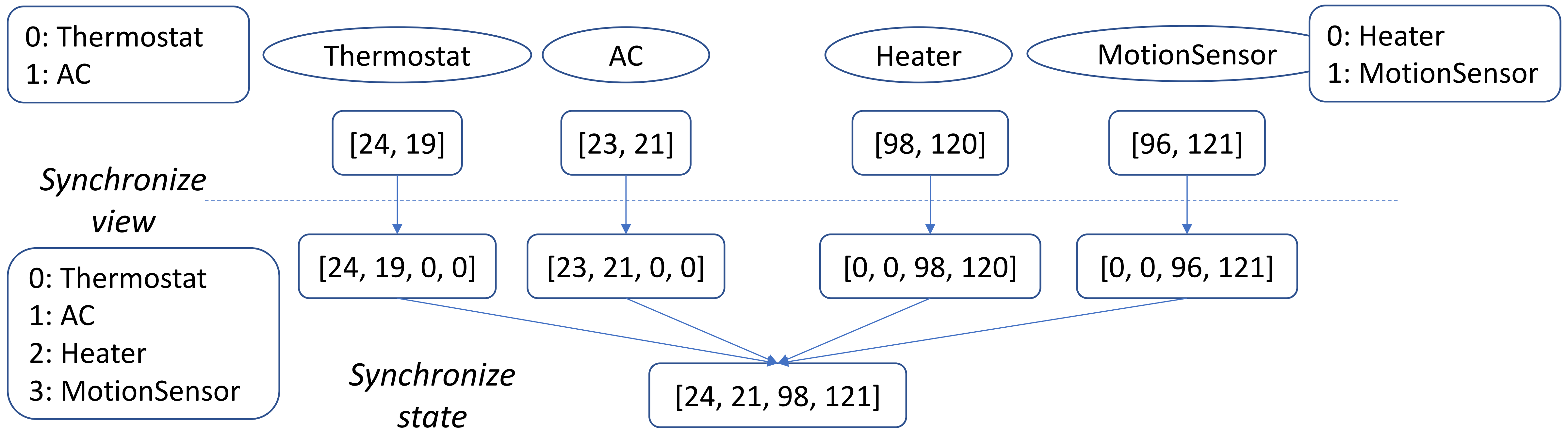


Reconciling Multiple Views

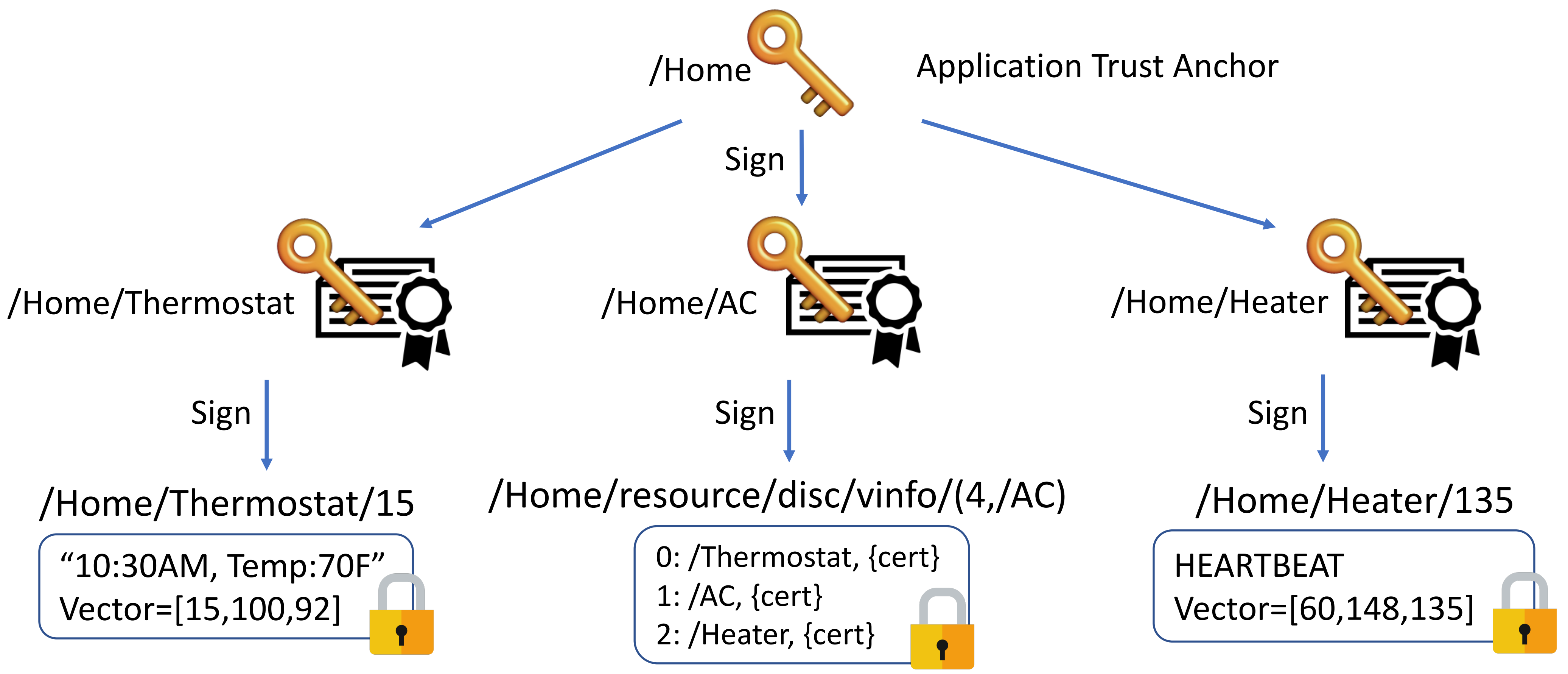

In view (6,Thermostat)
In view (4,Heater)
AC
Thermostat
Heater
Motion Sensor
/home/bonjour/(4,Heater)/…
/home/bonjour/vinfo/(4,Heater)
/home/bonjour/vinfo/(4,Heater)
0:Heater, {cert} 1: MotionSensor, {cert}
Publish: /home/bonjour/vinfo/(7,Thermostat)
0: Thermostat, {cert}; 1: AC, {cert}
3: Heater, {cert} 4: MotionSensor, {cert
/home/bonjour/(7,Thermostat)/…
Fetch (7,Thermostat) view info and move to (7,Thermostat)

After group partition heals, the leader with higher view number or “larger” leader name is responsible for merging the views

Synchronizing State After View Merging



Securing Dataset Synchronization



Summary

- Explicit group membership list (the view) enables the use of version vector as a concise representation of the dataset namespace
- Event-drive notification with explicit information allows nodes to retrieve new data immediately
- Keeping consistent view among all nodes by including the view ID in all notification Interests
    - Nodes can retrieve the view info after receiving new view ID
- Publishing all state info (vector, membership list) as named and secured data using well-defined naming convention

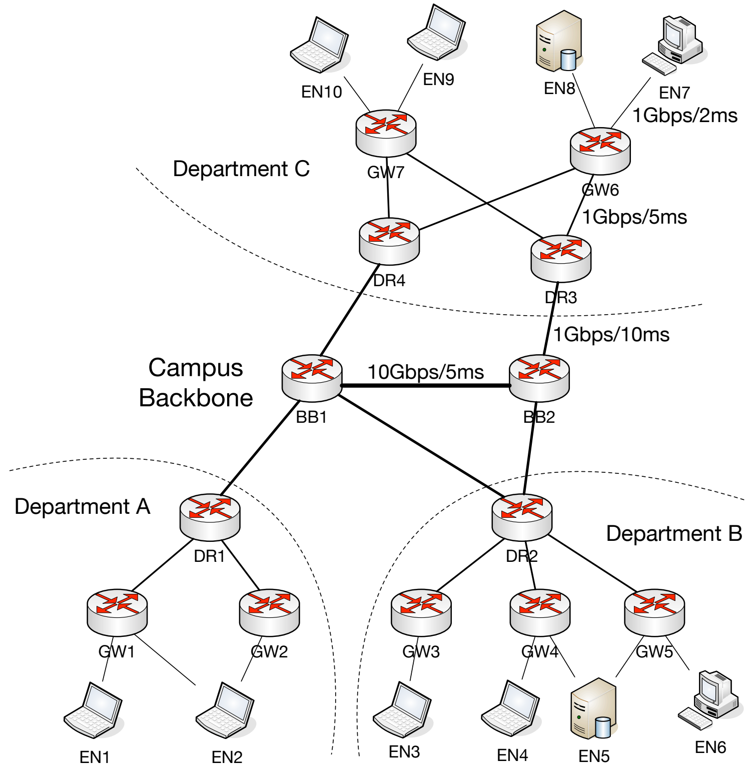
Simulation Study

- Methodology
  - Conduct experiments using ndnSIM (ns3-based network simulator)
  - Compare with ChronoSync using the same application

- Metrics
  - Data dissemination delay: the time needed for any node to receive the data after it is published
  - Synchronization delay: the time needed for the all nodes in the group to receive the data after it is published
  - Network traffic: total number of Interest and Data packets transmitted in the network

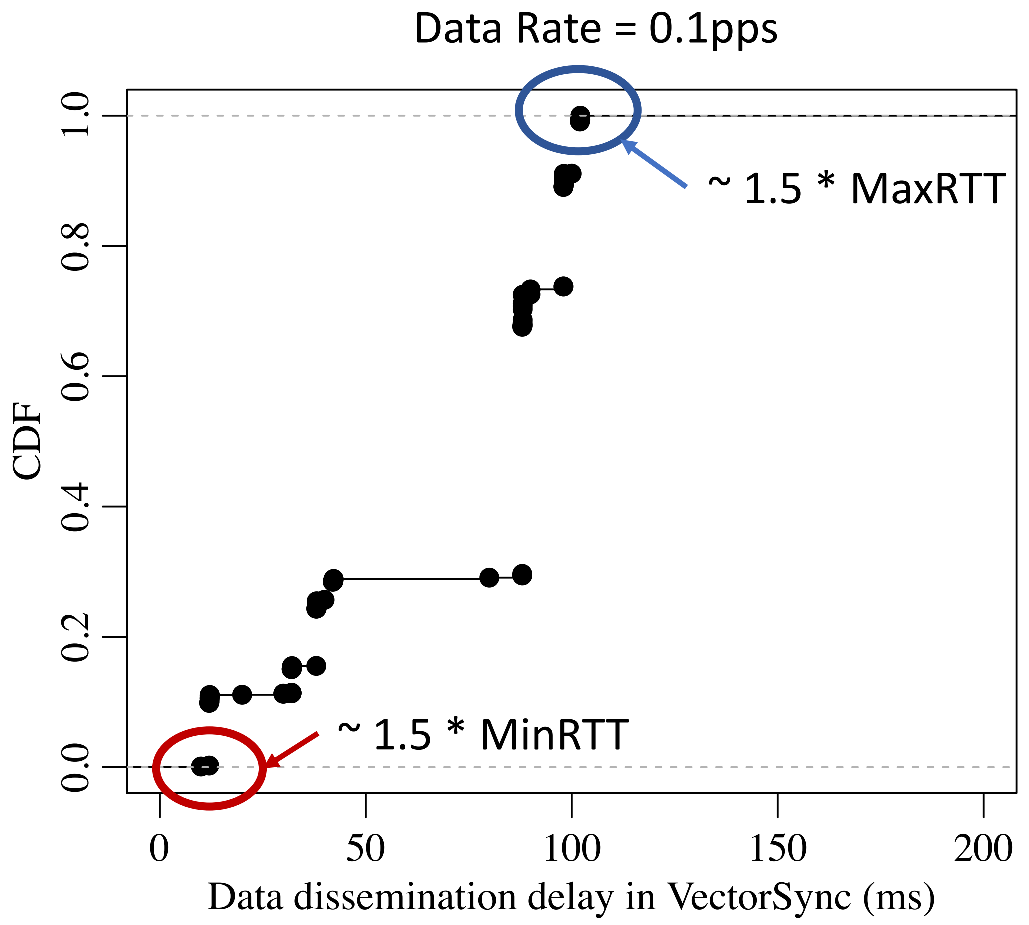
https://github.com/named-data/ChronoSync

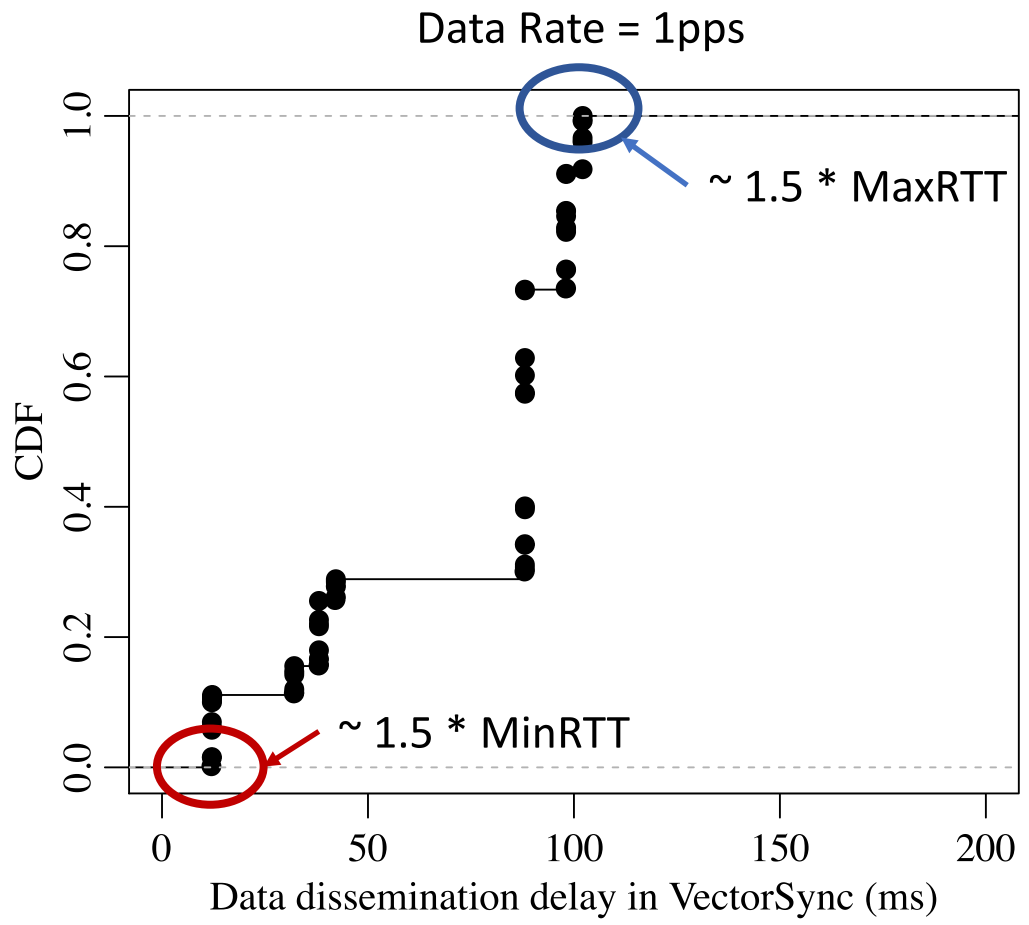
Scenario: small campus network



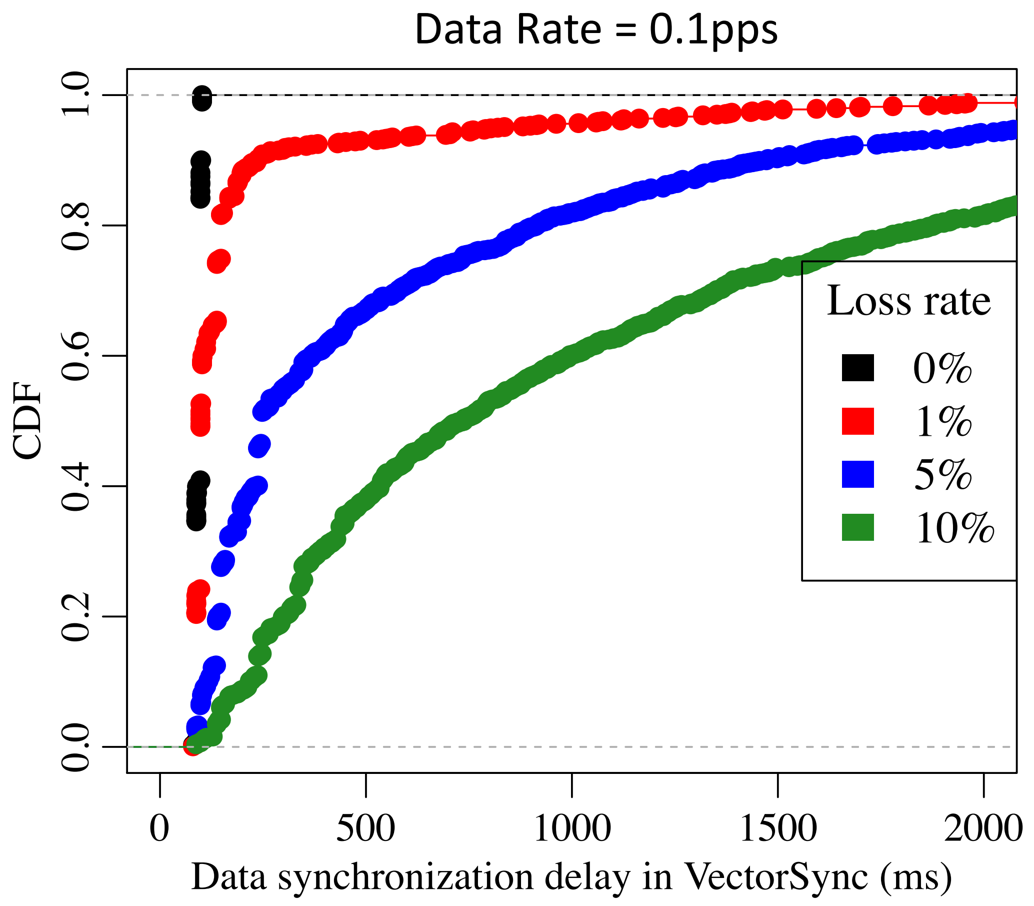
- Application running on 10 edge nodes participating in a sync group
  - RTT range: 8ms ~ 68ms
- Two types of traffic:
  - Low data rate: 10s average inter-arrival time for each
  - High data rate: 1s average inter-arrival time for each

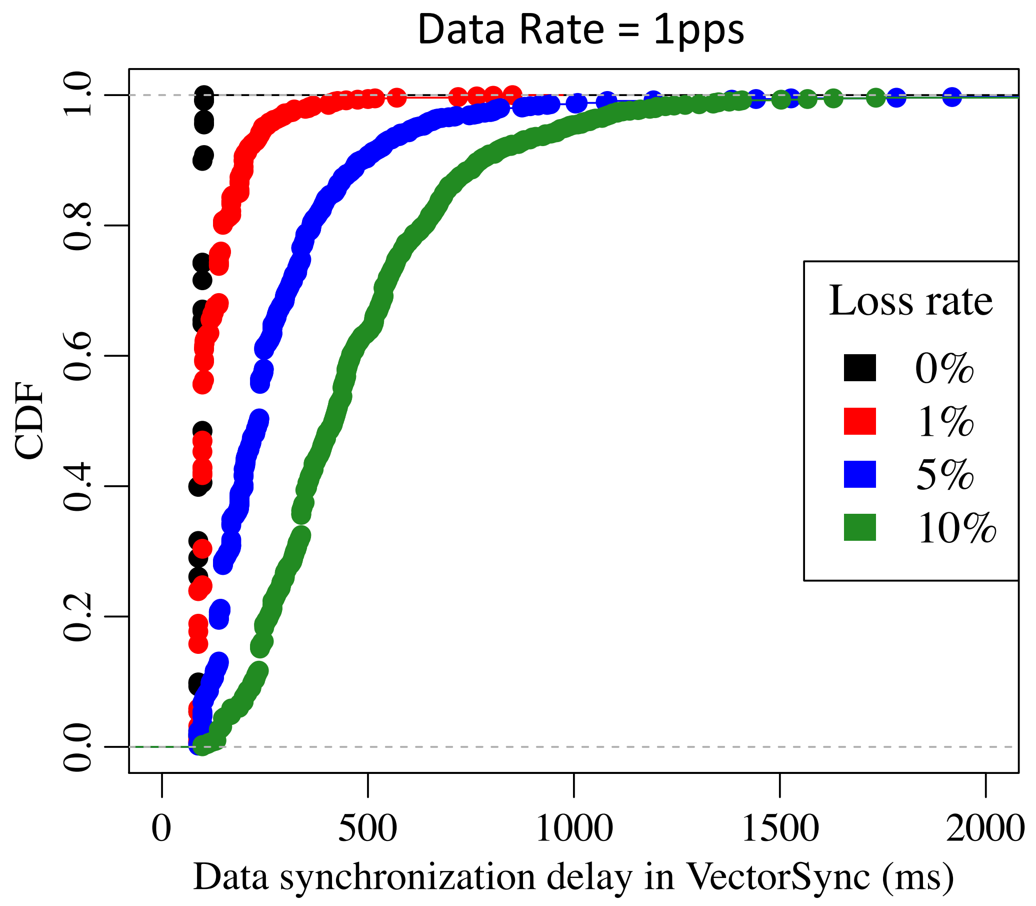
Data Dissemination Delay with No Packet Loss





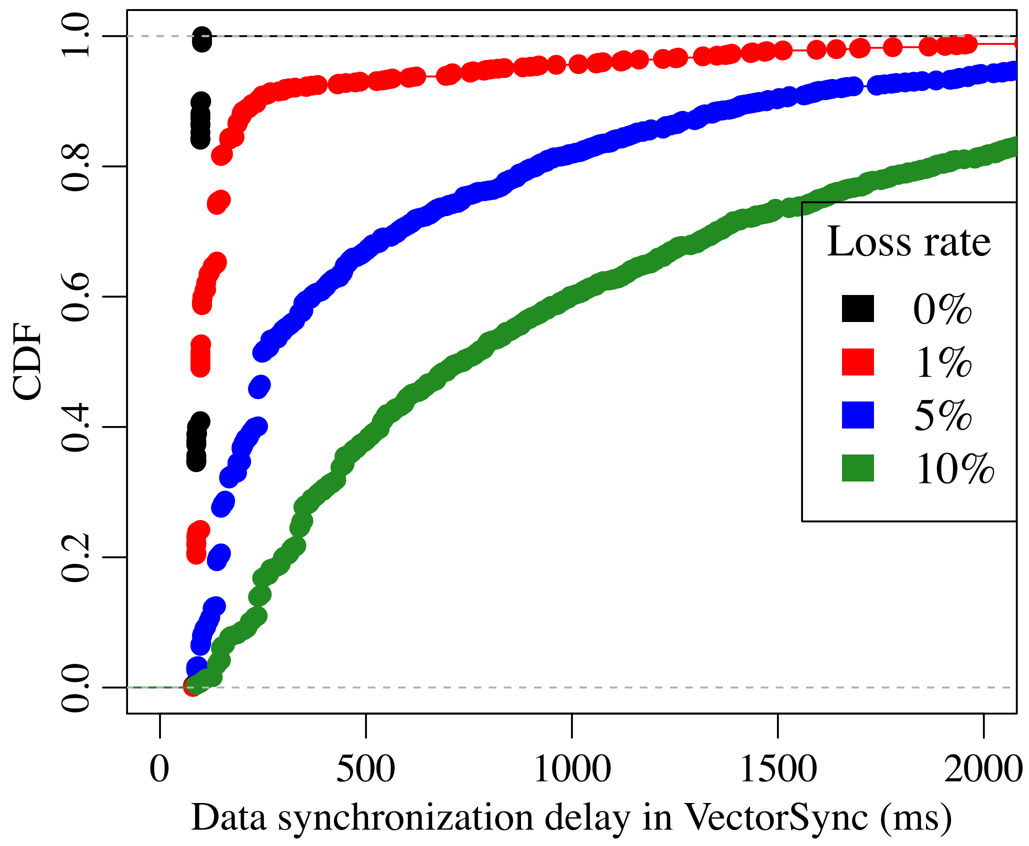
Synchronization Delay in VectorSync

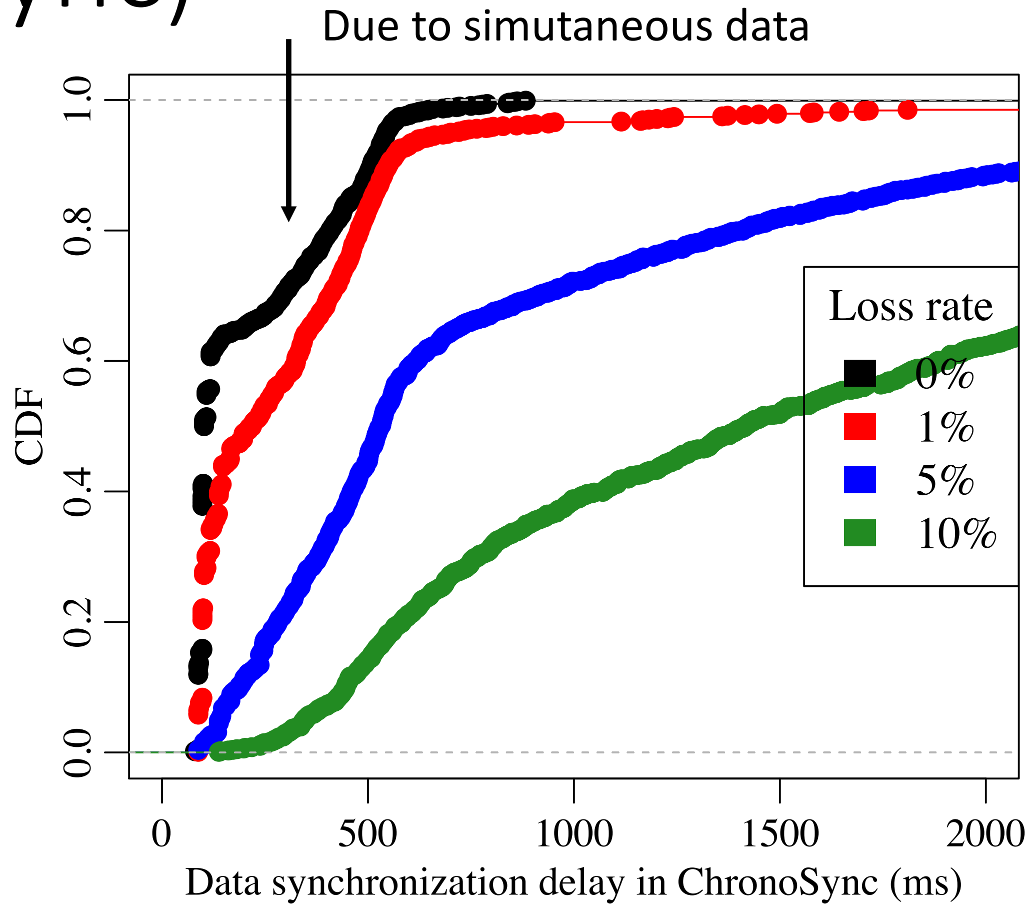




Higher data rate in the group enables VectorSync to recover from packet loss faster because the state vector carried in each data enables inconsistency detection

Synchronization Delay under Low Data Rate (VectorSync vs. ChronoSync)





VectorSync is resilient to simultaneous update because explicit notification allows receivers to fetch the new data immediately

Network Traffic under Low Data Rate (VectorSync vs. ChronoSync)

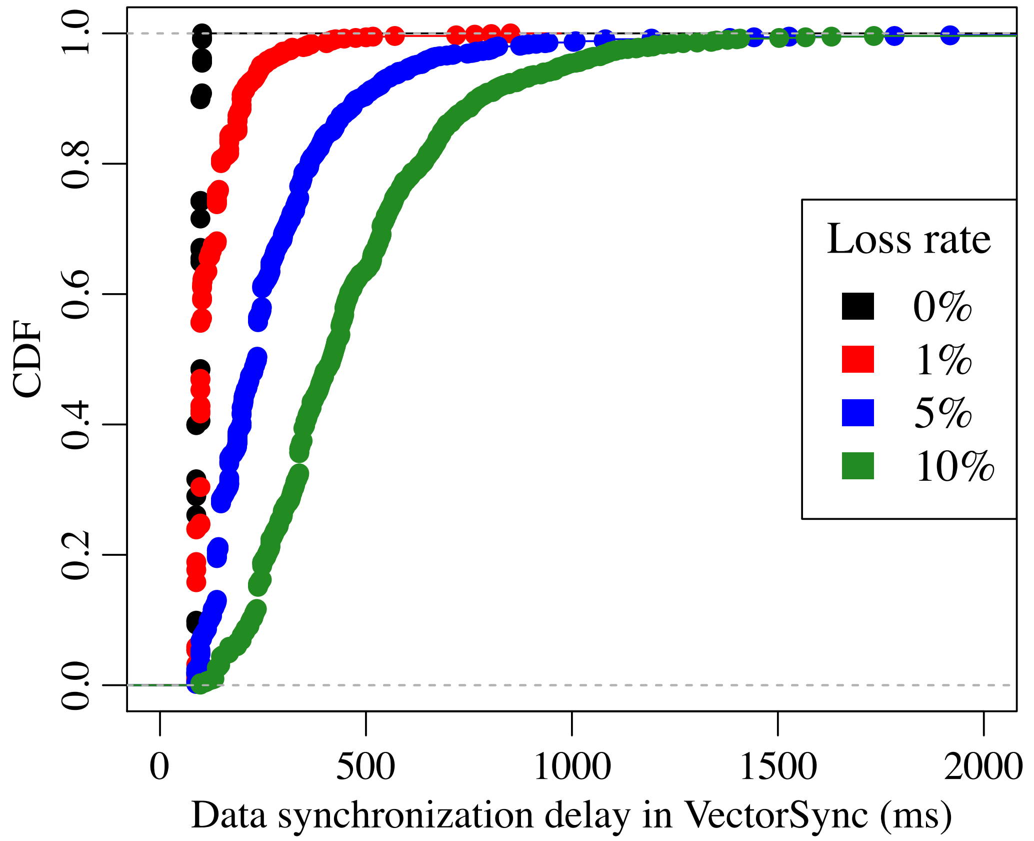
Loss Rate	0%		1%		5%		10%	
Packet Type	Interest	Data	Interest	Data	Interest	Data	Interest	Data
VectorSync	167k	134k	170k	134k	183k	132k	205k	129k
ChronoSync	321k	132k	359k	151k	436k	172k	455k	154k

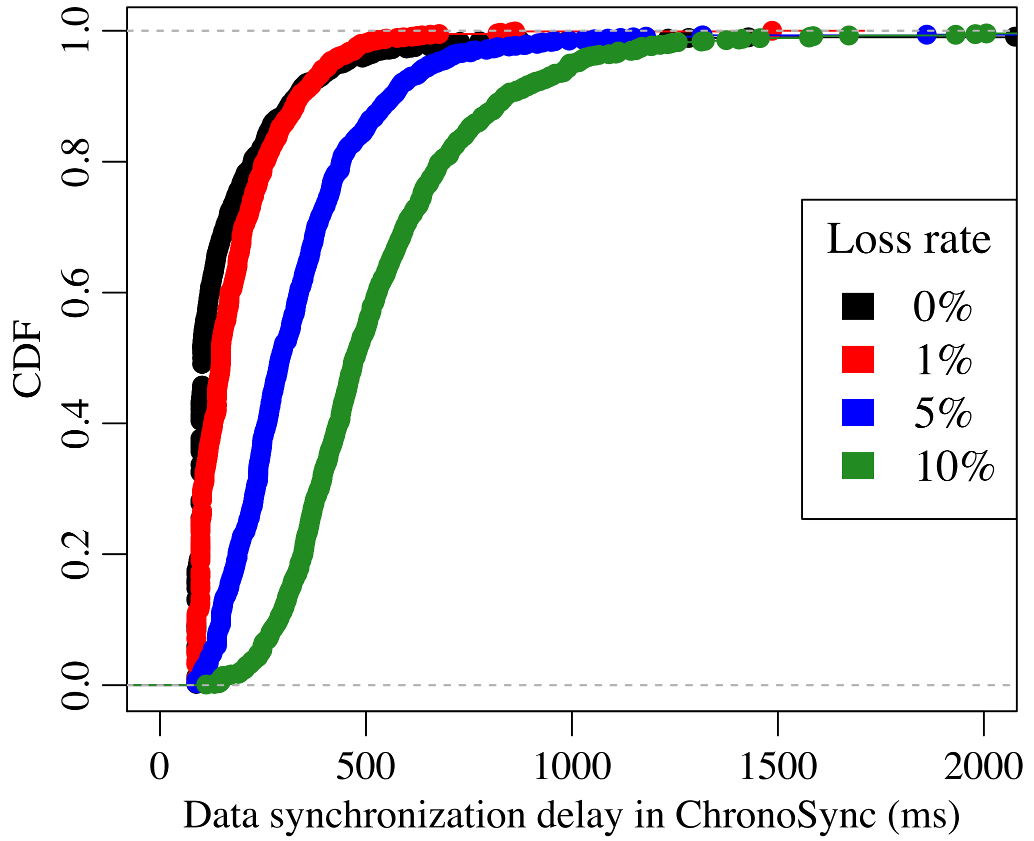
Total number of packets across all links

Main reason for higher Interest volume in ChronoSync:

- Additional multicast Sync Interest for detecting simultaneous updates
- Recovery Interest for repairing conflicting states

Synchronization Delay under High Data Rate





Under high data rate, ChronoSync invokes "recovery" mechanism frequently, which provides similar information as the state vector in VectorSync

Network Traffic under High Data Rate

Loss Rate	0%		1%		5%		10%	
Packet Type	Interest	Data	Interest	Data	Interest	Data	Interest	Data
VectorSync	101k	82k	104k	82k	112k	80k	126k	79k
ChronoSync	1179k	482k	989k	417k	730k	300k	576k	215k

Total number of packets across all links

- High traffic volume in ChronoSync due to frequent invocation of recovery mechanism
- Traffic volume in VectorSync increases only slightly due to Interest retransmission

Summary

- Carrying data name explicitly in the notification Interest enables prompt and efficient data dissemination, even in face of multiple simultaneous data producers
- Carrying state vector in nodes’ data enables detecting and reconciling inconsistency in the dataset namespace
- Higher group data rate enables VectorSync to recover from packet loss more quickly

Evaluating View Synchronization

- Understanding the behavior of view synchronization under node joining/leaving and packet loss
    - How fast the group reacts to membership changes
    - Messaging overhead
    - Additional delay
    - Implications of protocol parameter settings

- Results to be included in the dissertation

Comparison with Existing Protocols

Use application name

	Long-lived Interest	Notification driven	Periodic exchange
Enumeration		CCNx 1.0 Sync	
Hashing			CCNx 0.8 Sync
IBF			iSync

Name data sequentially

	Long-lived Interest	Notification driven	Periodic exchange
Enumeration		VectorSync	
Hashing	ChronoSync	RoundSync	
IBF	pSync		

Comparison with Existing Protocols

	Factors affecting Interest size	Factors affecting Data Content size	Interest Overhead	Min Data Dissemination RTT
CCNx 0.8 Sync	Node hash	Number of child nodes	Periodic	Depending on Interest period + tree walk
CCNx 1.0 Sync	Manifest digest	Total number of names	One per update	2.5
iSync	IBF digest	IBF (depending on number of new data)	Periodic	Depending on Interest period + 3.5 RTT
ChronoSync	State digest (with exclude filter)	Number of names with new sequence numbers	Long-lived Interest	Min is 0.5; can be long with simultaneous data publishing
RoundSync	Round digest (with exclude filter)	Number of names with new seq# in a round	Two per update	Min is 1.5; can be long with simultaneous data publishing
pSync**	IBF (+ subscription list)	IBF + number of new names	Long-lived Interest	1.5
VectorSync	Leader name + node name	State vector (small)	One per update (with heartbeat)	1.5

** pSync does not support group sync

Conclusion

Conclusion

- Distributed dataset synchronization is an important abstraction in NDN for supporting distributed applications
- Our comparative study of the existing sync protocols identified common sync protocol framework and exposed different tradeoffs in the protocol design choices
- We developed VectorSync, a new sync protocol design that overcome the drawbacks of the existing sync protocols
  - Explicit group membership management
  - Explicit new data notification
  - Detecting and reconciling dataset inconsistency using version vector

Future Works

- Exploring different group rendezvous mechanisms
  - DHT
  - Viral propagation
- Applying VectorSync to NDN applications
  - Routing protocol
  - Distributed repository

Publications

NDN Applications

- *NDN.JS: a Javascript Client Library for Named Data Networking*, Infocomm'13
- *NDNFS: an NDN-friendly File System*, NDN-TR-0027, 2014
- *MicroForwarder.js: an NDN Forwarder Extension for Web Browsers*, ICN'16

IoT over NDN

- *Securing Building Management Systems Using Named Data Networking*, IEEE Network, vol.28, no.3, 2014
- *NDN-ACE: Access Control for Constrained Environments over Named Data Networking*, NDN-TR-0036, 2015
- *Challenges in IoT Networking via TCP/IP Architecture*, NDN-TR-0038, 2016
- *Named Data Networking of Things*, IoTDI'16
- *The Design and Implementation of the NDN Protocol Stack for RIOT-OS*, Globecom'16 ICNSRA Workshop
- *Breaking out of the cloud: Local trust management and rendezvous in Named Data Networking of Things*, IoTDI'17

NDN Sync

- *The Design of RoundSync Protocol, NDN-TR-0048, 2017*
- *A Survey of Distributed Dataset Synchronization in Named Data Networking*, NDN-TR-0053, 2017

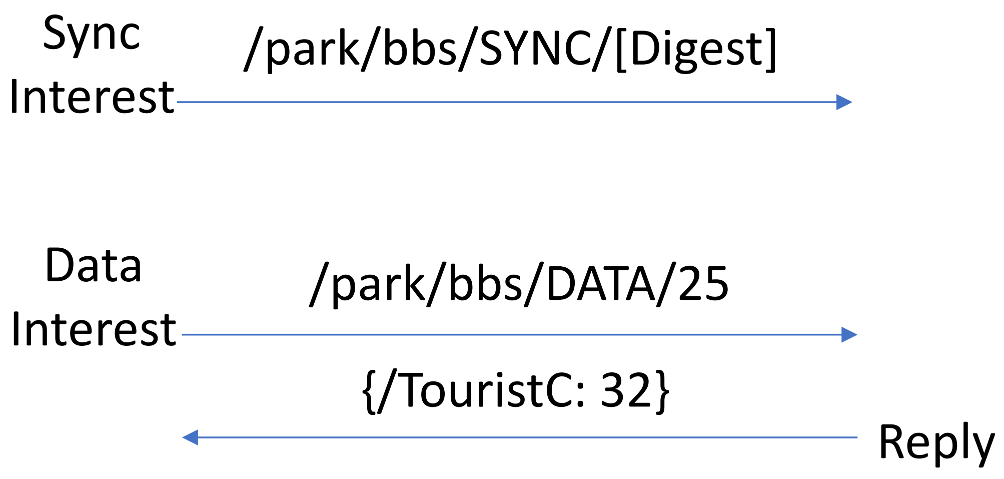
Backup Slides

RoundSync: addressing issues in ChronoSync

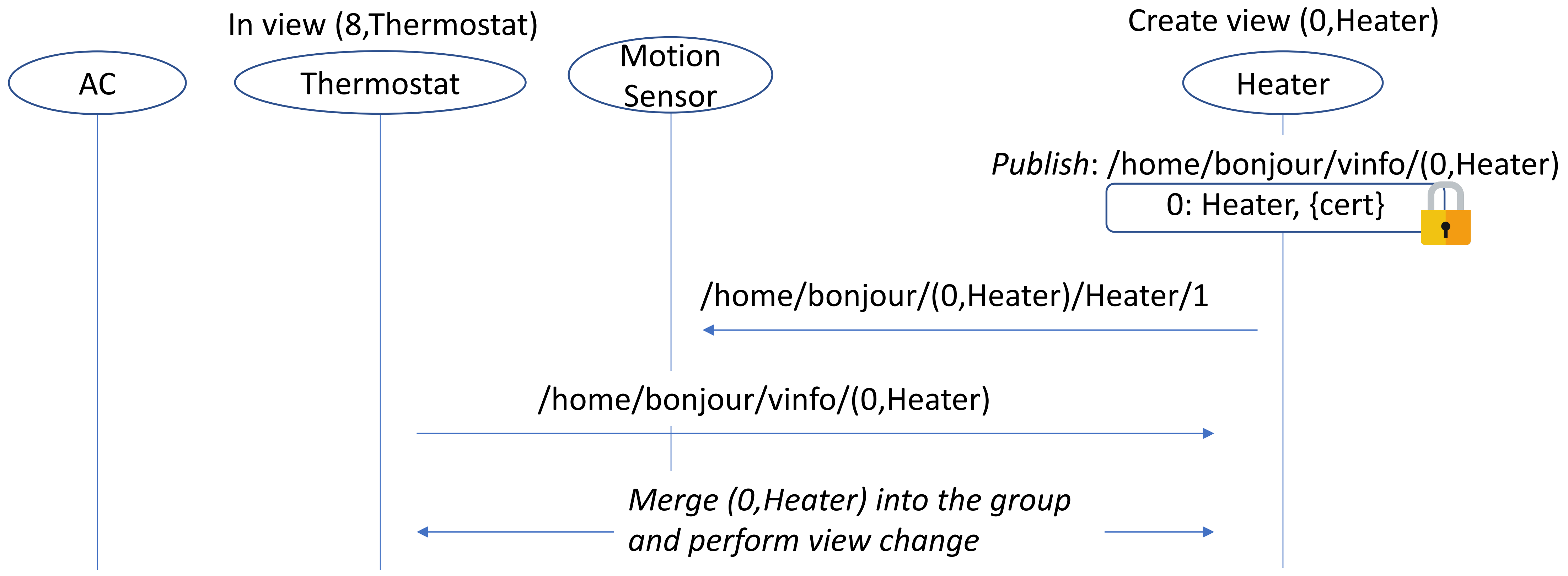
- Divide data publishing into rounds
- Decouple state notification from update retrieval
- Still need special mechanism to retrieve multiple data in a single round

Round Log

RN	Dataset	Digest
	…	
24	{/TouristA/15, /TouristB/60}	D24
25	{/TouristC/32}	D25



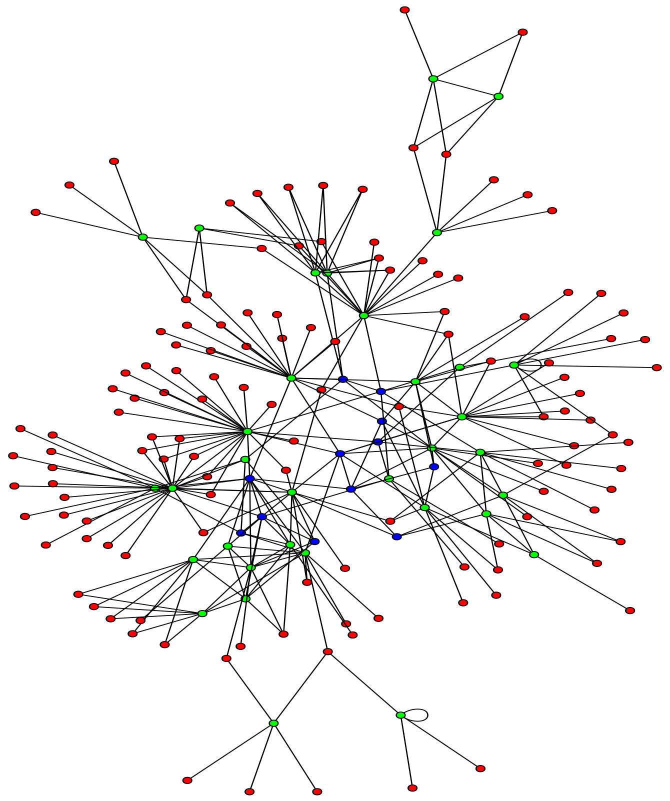
Joining the Group



In view (8,Thermostat)
Create view (0,Heater)
AC
Thermostat
Motion Sensor
Heater
Publish: /home/bonjour/vinfo/(0,Heater)
0: Heater, {cert}
/home/bonjour/(0,Heater)/Heater/1
/home/bonjour/vinfo/(0,Heater)
Merge (0,Heater) into the group and perform view change

Node joining is handled in the same way as view merging

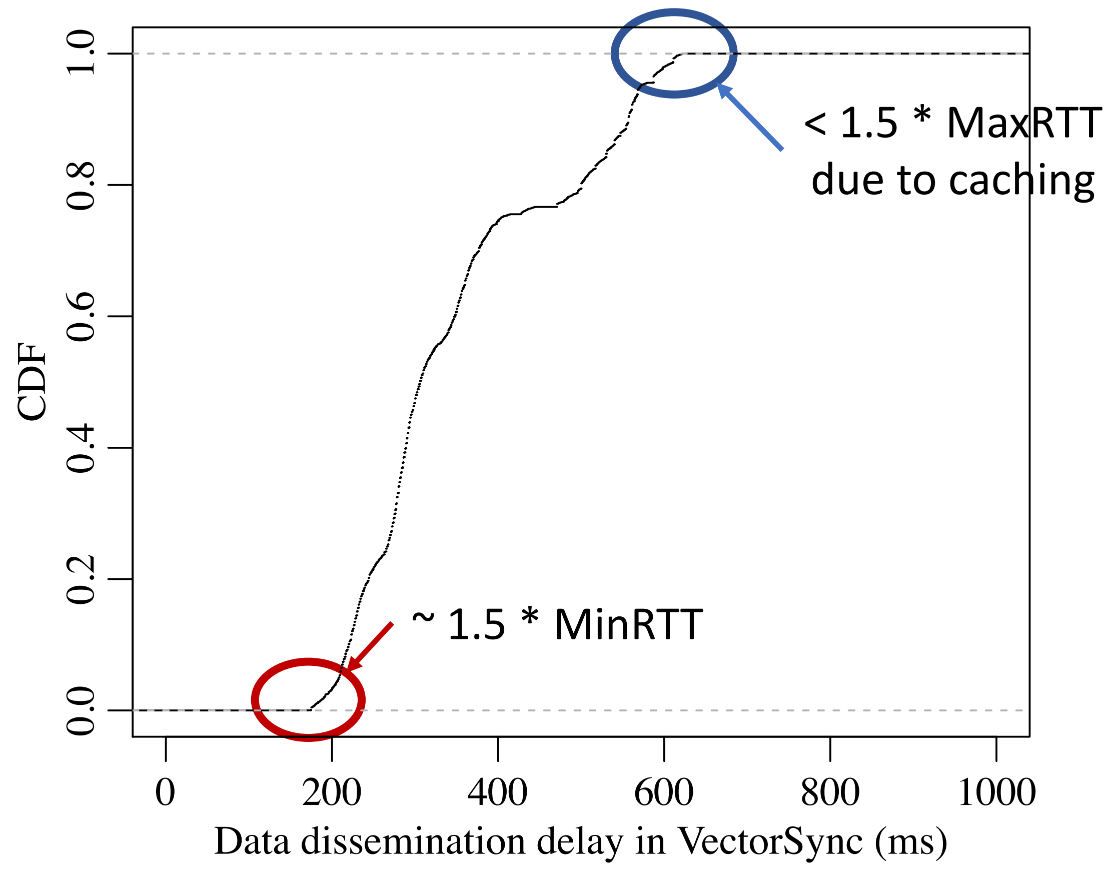
Scenario 2: large ISP network



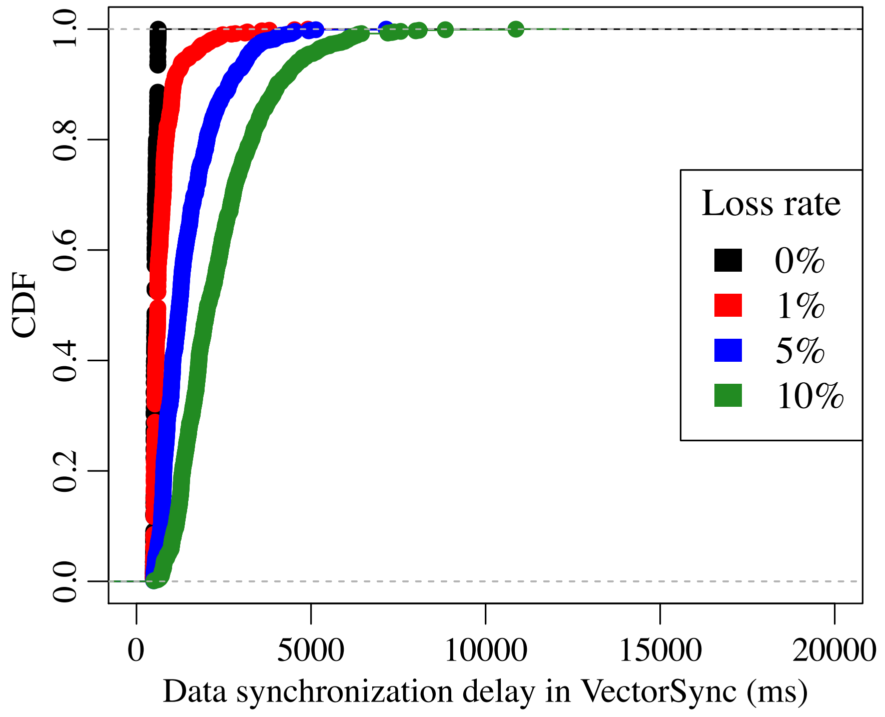
- Randomly pick 10 “leaf” nodes
  - RTT range: 111ms ~ 476ms
- Traffic: 10s inter-arrival time from each node

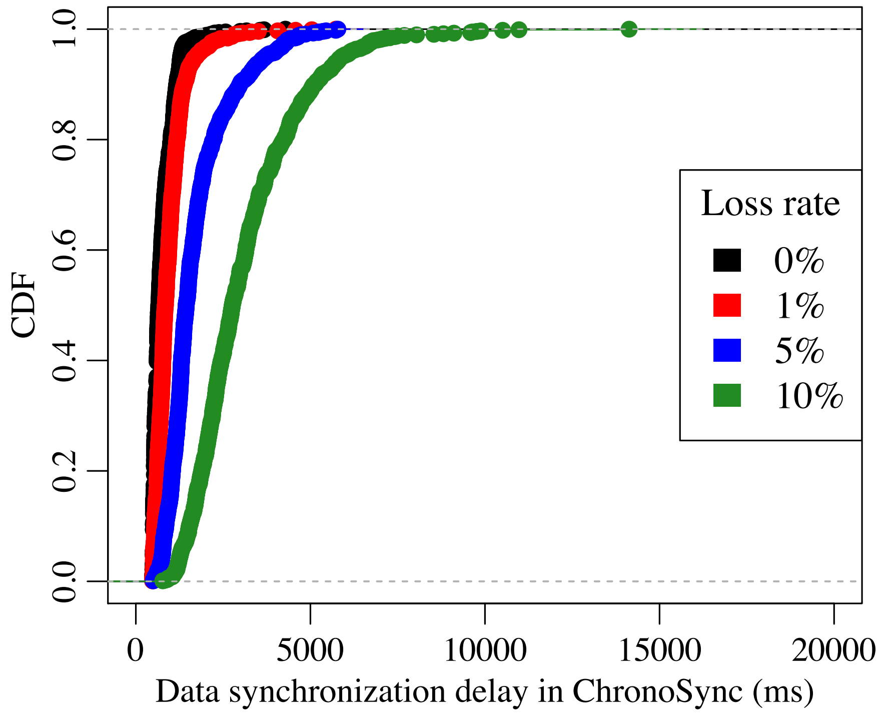
N. String et al., “Measuring ISP topologies with Rocketfuel”, 2004

Data Dissemination Delay with No Packet Loss



Synchronization Delay





Network Traffic

Loss Rate	0%		1%		5%		10%	
Packet Type	Interest	Data	Interest	Data	Interest	Data	Interest	Data
VectorSync	519k	185k	522k	185k	534k	184k	575k	187k
ChronoSync	1224k	248k	1346k	273k	1616k	317k	1647k	294k

Total number of packets across all links

High traffic volume in ChronoSync due to “recovery” mechanism

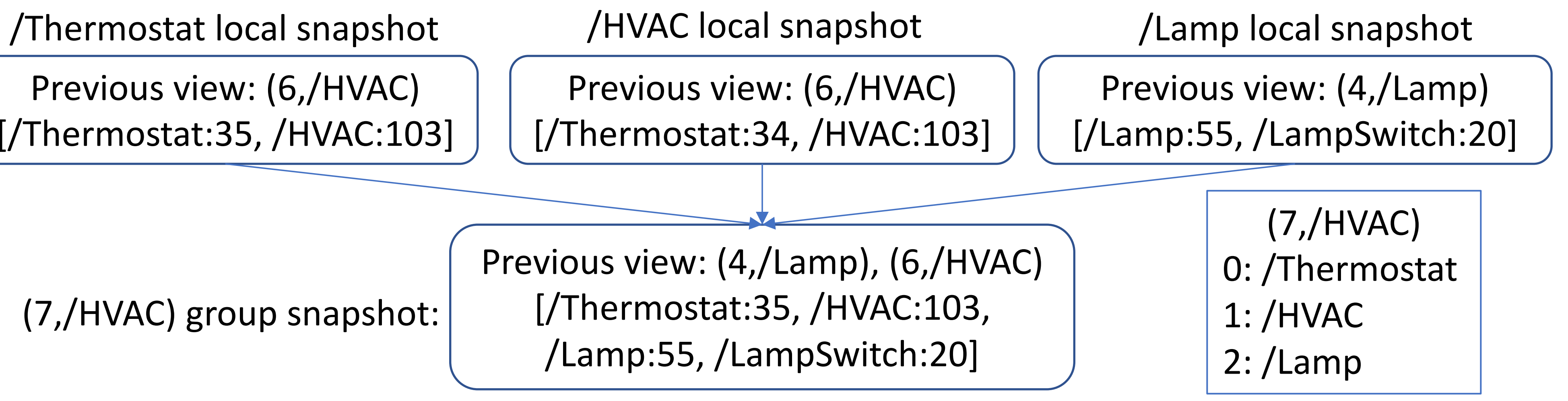
Dataset Snapshot and Permanent Storage

Motivation

- Problem:
  - VectorSync synchronizes among active members
  - Some applications may want to keep all data published in the history
- Solution:
  - The group generates a “snapshot” for the dataset *state* at the beginning of each view
    - A snapshot is a *version vector* covering all the data published in the group before the view starts
  - A dedicated repo collects data based on the snapshot and store permanently
  - New nodes retrieve whole dataset from the repo to bootstrap

Generating Group Snapshot

- Before syncing data in a new view, each node publishes its local snapshot in the shared dataset
  - Local snapshot packets propagated in the group via sync
- Leader computes the group snapshot as the *Join* of local snapshots from *all* nodes in the current view


/Thermostat local snapshot
Previous view: (6,/HVAC)
[/Thermostat:35, /HVAC:103]
/HVAC local snapshot
Previous view: (6,/HVAC)
[/Thermostat:34, /HVAC:103]
/Lamp local snapshot
Previous view: (4,/Lamp)
[/Lamp:55, /LampSwitch:20]
(7,/HVAC) group snapshot:
Previous view: (4,/Lamp), (6,/HVAC)
[/Thermostat:35, /HVAC:103,
/Lamp:55, /LampSwitch:20]
(7,/HVAC)
0: /Thermostat
1: /HVAC
2: /Lamp

Summary

- Effectiveness
    - Synchronizing latest seq# ensures nodes discover all missing data
    - Periodic heartbeat helps recover from packet loss
- Efficiency
    - Explicit data name notification enables faster sync
    - Simultaneous publishing do not interfere
- Scalability
    - Membership management controls state vector size
    - Large groups may adopt compressed encoding (e.g., IBF)