Enabling Decentralized Applications: A Transport Perspective

Varun Patil
PhD Defense

Outline

- Background
  - The role of end-users in today's Internet
  - Named Data Networking
- **State Vector Sync**
- **NDN Distance Vector Routing**
- **Ownly: A Decentralized Application**
- Future Directions

The role of end-users in today's Internet

A BRIEF BACKGROUND

Where are we? – Today's Applications

cen·tral·i·zation of the internet

/ˌsentrələˈzāSH(ə)n, ˌsentrəˌlīˈzāSH(ə)n/

the concentration of **control** of an activity or organization under a single authority.
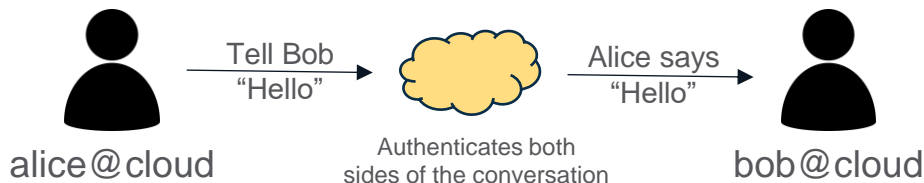
- Applications and services, together with the resulting data, are provided and controlled by a small number of dominant players

- Users depend on cloud; no way to build apps where users fully control their data
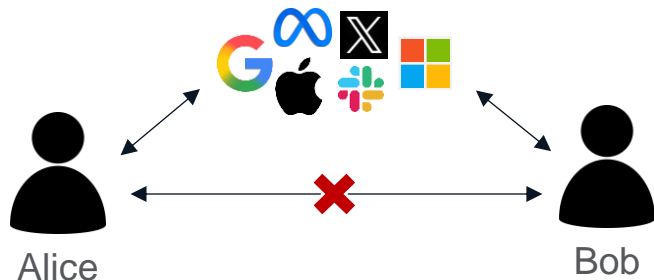
The role of end users in today's Internet

The internet started as a distributed system without a central controller. Consolidation of applications happened over time.

- No direct communication between users – NATs and firewalls
  - User devices in general don't have public addresses
  - Asymmetric – user to cloud services only
- Users are identified by cloud provider-assigned email addresses
  - Security is handled by central servers
  - Inter-user authentication is through the provider

alice@cloud   Tell Bob "Hello" →   Alice says "Hello" →   bob@cloud

Authenticates both sides of the conversation

Mitigating the limitations of TCP/IP

TCP/IP was designed for resource sharing, applications like Telnet.
Modern applications are inherently distributed and many-to-many.

- Point-to-point networking is a poor fit for applications
- IP multicast aimed to help, but has seen no deployment
  - Multiple years of research; many protocols developed
  - Multicast routing never rolled out
  - Disguised architectural change
- Rise of Content Delivery Networks (CDNs)
  - Multicast delivery at application layer
  - Only serve paying customers
  - 3rd party CDNs disguised as application – TLS termination

Efforts for decentralization in recent years

Several platforms, applications and protocols such as Solid, Nostr, Mastodon, and Bluesky are being designed and deployed. These aim to decentralize the control power by new app designs running over TCP/IP

Observations from examining their design and effectiveness in decentralization.

- Naming users by the nodes they attach to ties users to specific servers
    - Users should have their own names
- Using keys as user names leads to further indirection
    - Apps work with meaningful names, not key IDs
- Semantic names help define and enforce security policies
- Application logic should be decoupled from data storage
    - Storage should be a generic resource, used but not controlled by apps

Ma, Xinyu. *Towards Decentralized Applications*. University of California, Los Angeles, 2024.

Named Data Networking

A NEW WAY OF THINKING

Identifying Application Needs

Modern applications are data-centric. An application running over HTTP just requests a piece of data using a URL (i.e. a globally unique identifier for that data)

- Applications only care about getting the desired data
  - Data-centric requirements
- TCP/IP provides host-centric or location-based packet delivery
  - Networking by TCP/IP requires the apps to know *where* the data is
- All communication must also be secured
- Today's practice – many layers of indirection to support this semantic mismatch

Named Data Networking

NDN is a data-centric network architecture – applications request the data they need by name; the network locates and delivers the named and secured data.

- Conceptually: consider bringing HTTP down to the network layer
- Each piece of data is uniquely identified globally, or within the scope of usage
  - Semantically meaningful hierarchical **name** with one or more components
    - Names are application-defined
  - Not a new concept or requirement – URLs

`/alice/hello`
"Hello"
Signed, Alice

Alice

Named Data Networking

NDN is a data-centric network architecture – applications request the data they need by name; the network locates and delivers the named and secured data.

- **Consumers** send **Interest** packets to request data
  - Carries the name of the piece of data
- **Producers** generate named secured **Data** packets, reply to Interests
  - Carries the name and application content
- Network forwards Interests to producers and Data in reverse
  - Name-based forwarding, according to routing protocol

Alice

/alice/hello
"Hello"
Signed, Alice

/alice/hello
"Hello"
Signed, Alice

Bob
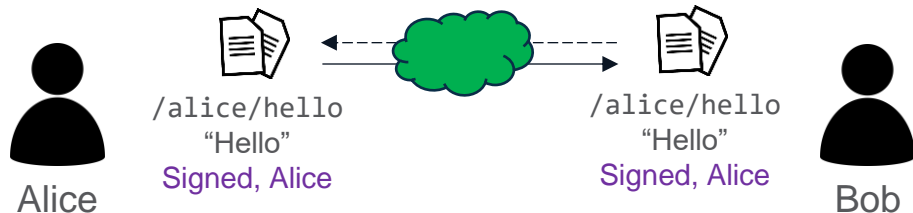
Stateful Forwarding

NDN uses a stateful data plane for forwarding. Interests leave a breadcrumb trail in the network which the Data traces back to the consumer(s).

When a forwarder receives an Interest packet, it –

1. Looks up the name-based **Forwarding Information Base (FIB)**.
   FIB tells the forwarder which paths may bring back the requested data.

2. Adds an entry to the **Pending Interest Table (PIT)**.
   PIT tracks outstanding Interests and their ingress and egress (inter)faces.

3. On receiving the Data reply, the forwarder looks up the PIT.
   Data is forwarded to all ingress faces for the pending Interest.

Named Data Networking
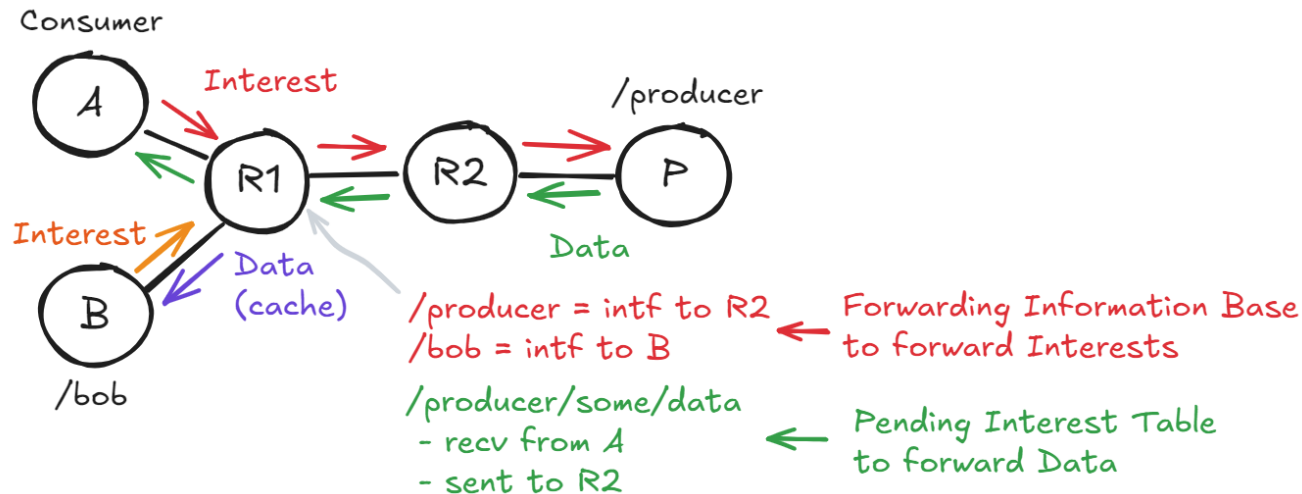
An illustration of name-based stateful forwarding in NDN

Data-centric Security – End-to-End Security

NDN security is fundamentally different from TCP/IP's use of TLS – an authenticated and encrypted pipe.

- Each piece of data is signed by the producer

- Consumers verify the authenticity of the data independently of where it comes from – no connections

- Signed data can be securely cached in the network – built-in CDN function.

Alice

/alice/hello
"Hello"
Signed, Alice

/alice/hello
"Hello"
Signed, Alice

Bob
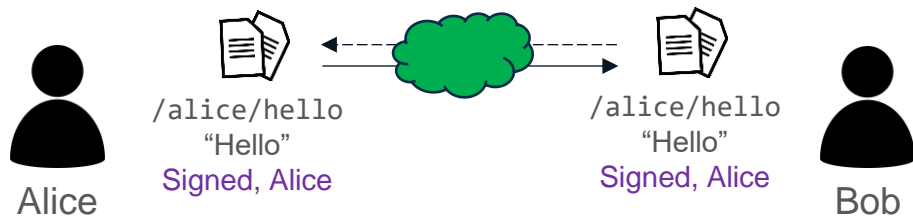
Security Bootstrapping

NDN views a network as made of entities with trust relationships. Each entities assumes an identity through a process of security bootstrapping, with four elements.

- Name

- Trust Anchor

- Certificate of Identity
  - For the entity's identity, e.g. "gmail.com/@alice"

- Security Policies – Trust schema
  - Who can do what
 - Semantic naming enables systematic definition of security policies

Transport Layer in NDN

Transport bridges the functionality gap between what the network offers and what applications need

- Names provide multiplexing and demultiplexing
  - Port numbers are no longer needed

- Applications define the level of reliability desired
  - Retransmit Interests if no data received

- Congestion control at the network layer
  - Enabled by NDN's 2-way packet exchange – feedback loop
  - This was always a network function, bolted to transport layer in IP because IP forwarding is open loop

- To fetch data by name, applications must be aware of the latest available data – the task of NDN's transport layer.

Distributed Dataset Synchronization

- Multiple communicating entities in an application form a **Sync Group**

- Collection of data names produced in the application – **Dataset State**

- When any entity produces data,
  - Sync informs everyone else about existence of this data name
  - Others may fetch this data if desired (using name)

- Sync provides **reliable** *namespace* synchronization
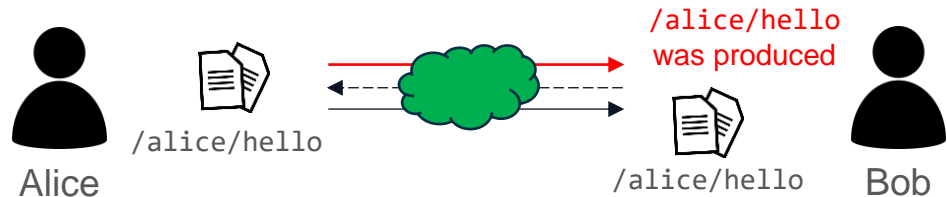
The Development of State Vector Sync

A NEW TRANSPORT PROTOCOL FOR NDN

Goals of Sync Design

These goals directly serve as metrics when evaluating Sync protocols.

- **Reliability** in namespace synchronization – applications need to know what data is available, even if they decide to not fetch it.

- Low synchronization **latency** – Sync should enable low-latency real-time applications to perform well.

- **Resiliency** – Sync should work well in all network conditions, such as high loss rates and intermittent network partitions.

- Low **overhead** – just like any other protocol

Previous Sync Designs

Several NDN Sync protocols were developed before SVS.
From a systematic study, we identify a few common components.

- **Namespace Representation**
  - Directly use application names
  - Sequential naming, i.e. [producer name, seq#]
- **Dataset State Encoding**
  - Compact representation for protocol messages
- **State Change Notification**
  - Multicast long-lived Sync Interests
  - Multicast notification Sync Interests

Sync Protocol Case Study

CHRONOSYNC

Namespace Representation
Sequential Naming

Dataset State Encoding
State Digest

State Change Notification
Long-lived Sync Interest
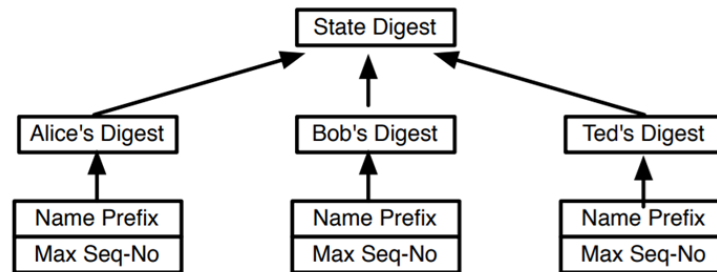
(b) An example of sync data name

Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking," 2013
21st IEEE International Conference on Network Protocols (ICNP), Goettingen, Germany, 2013

Sync Protocol Case Study

SYNCPS

Namespace Representation
    Directly use application names
    Each name has a lifetime in the dataset

Dataset State Encoding
    Invertible Bloom Filter

State Change Notification
    Long-lived Sync Interest
    Data replies carry content

Kathleen Nichols. 2019. Lessons Learned Building a Secure Network Measurement Framework using Basic NDN. In Proceedings of the 6th ACM Conference on Information-Centric Networking (ICN '19).

Sync Protocol Case Study

PSYNC

Namespace Representation
Sequential Naming

Dataset State Encoding
Invertible Bloom Filter

State Change Notification
Long-lived Sync Interest

Sync Interest

Name: /<routable-prefix>/psync/<SL>/<old-IBF>

Sync Data

Name:/<routable-prefix>/psync/<SL>/<old-IBF>/<new-IBF>
Content /<prefix1>/<version> /<prefix2>/<version>

M. Zhang, V. Lehman and L. Wang, "Scalable name-based data synchronization for named data networking," IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, Atlanta, GA, USA, 2017

Identifying Common Issues

- One multicast Sync Interests triggers multiple replies
  - Only one reply is delivered back to the sender
  - Different group members may get different replies
- Reliance on long-lived Interests
  - Persistent state in the network that needs to be refreshed
  - Hard to detect and recover from losses

Designing a New Sync Protocol

STATE VECTOR SYNC

Namespace Representation
   Sequential Naming

Dataset State Encoding
   Raw State

State Change Notification
   Notification Sync Interests
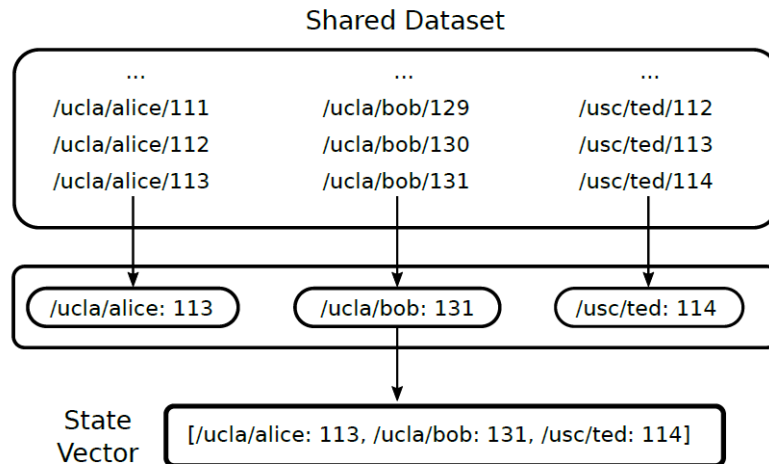
Shared Dataset

...	...	...
/ucla/alice/111	/ucla/bob/129	/usc/ted/112
/ucla/alice/112	/ucla/bob/130	/usc/ted/113
/ucla/alice/113	/ucla/bob/131	/usc/ted/114

/ucla/alice: 113   /ucla/bob: 131   /usc/ted: 114

State Vector: [/ucla/alice: 113, /ucla/bob: 131, /usc/ted: 114]

M. Zhang, V. Lehman and L. Wang, "Scalable name-based data synchronization for named data networking," IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, Atlanta, GA, USA, 2017

SVS Naming and Sync Interest

/<grp-prefix>/v=3/<digest>

(1) (2) (3)

/ucla/cs/irl/chatroom/v=3/c9ff1f50...

(1) (2) (3)

(a) Sync Interest Naming Scheme and Example

/<grp-prefix>/<producer-prefix>/<boot-time>/<seq-no>

(1) (2) (3)

/ucla/cs/irl/chatroom/ucla/cs/alice/t=1745693492/124

(1) (2) (3)

(b) Data Item Naming Scheme Example

```
/<grp-prefix>/v=3/<digest>
Lifetime=1s
Application Parameters
  /<grp-prefix>/v=3
    State Vector
  Signature
```

(c) Structure of an SVS Sync Interest

Sync Interests carry a signed copy of the raw state vector.

Basic Functionality

Sync Interests of SVS carry the entire raw state of the group.
Receiving a single Sync Interest is enough for synchronization.

- Producer **P** increments local sequence number on data production

- SVS generates Sync Interest with updated state vector

- Receivers update their local state
  - Sequence number of **P** changed
  - Inform application about new data
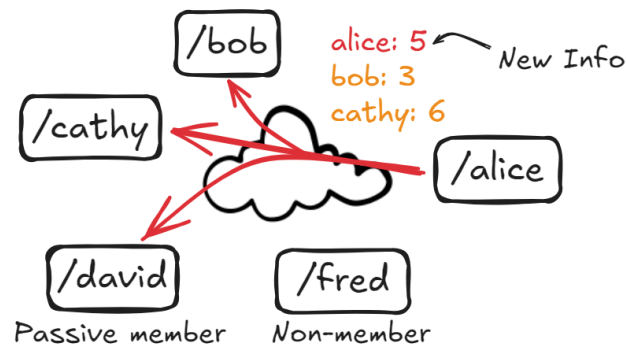
Loss Recovery

Some Sync Interests may be lost due to packet losses. Sync protocols should be resilient to losses, and provide eventual consistency of the namespace dataset state.

- During losses, some node **N** has outdated state

- **N** sends an outdated Sync Interest
  - Received by everyone else
  - Compare with local state

- "Reply" by generating own Sync Interest
  - Contains up-to-date state

- **N** receives this; updates own state

- Periodic Sync Interests to ensure this happens
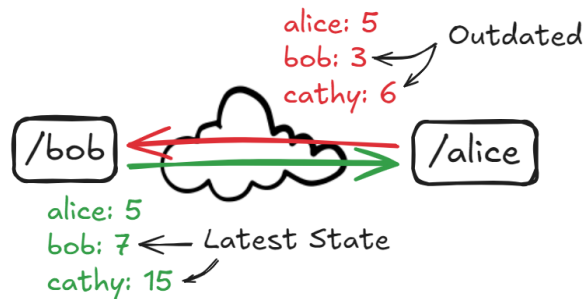
Sync Interest Suppression

Each member of an SVS group has equivalent behavior. Loss recovery may generate too many messages, both periodic and in response to outdated Sync Interests.

- SVS uses a **single** timer for loss recovery

- On receiving or sending up-to-date state
  - Reset timer to periodic timeout (30s ± 10%)

- On receiving outdated state
  - Reset timer to suppression timeout (0 - 200ms)

- Multicast + randomness reduces redundancy

- Suppression at end users, not in the network
  - State vector must be signed by the application
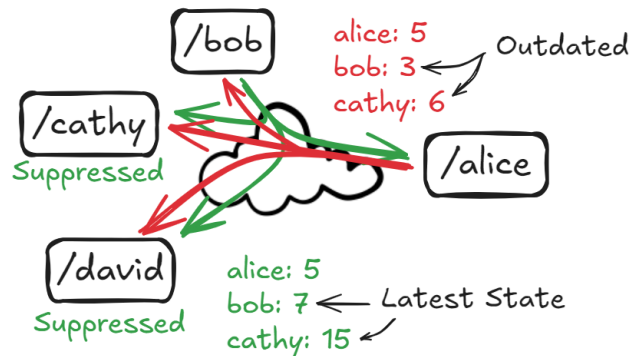
Sync Interest Suppression
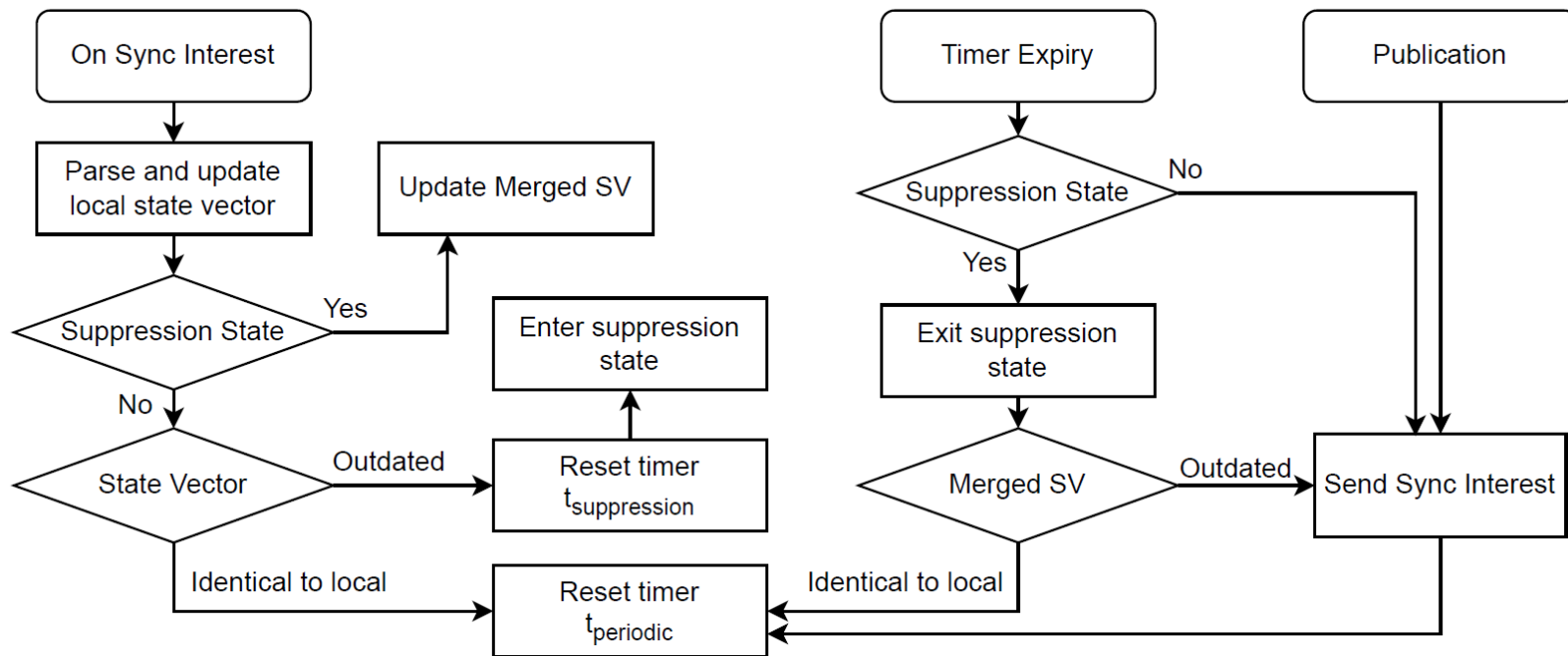
Tuning SVS Timers

The actual values of the periodic and suppression timeout affect SVS performance.
These values must be tuned for the network and application.

- Exponential suppression timer sampling

- Throttling outgoing Sync Interests

- Learning optimum values from the network
    - Suppression timeout proportional to network size
    - Shorter periodic timeout for lossy networks
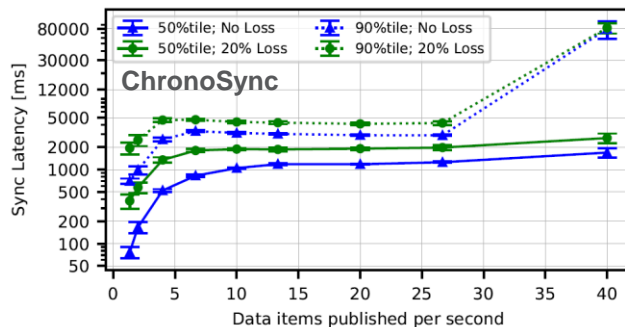   - Trade-off in latency vs overhead
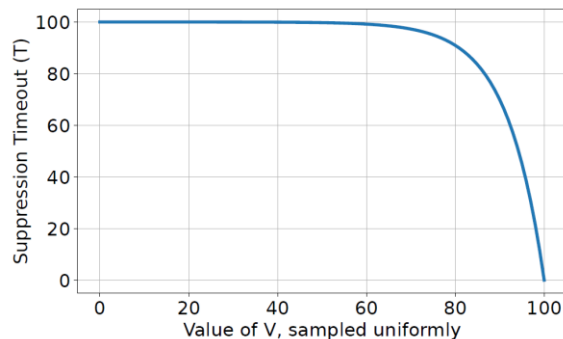
$$T = C \times \left(1 - e^{\frac{V-C}{C/F}}\right)$$

Evaluation

- Emulated 45-node GEANT topology

- Each link has 10ms propagation delay

- 20 participants in a Sync group

- Varying publication frequency

- Varying loss rate

- Latency and overhead metrics
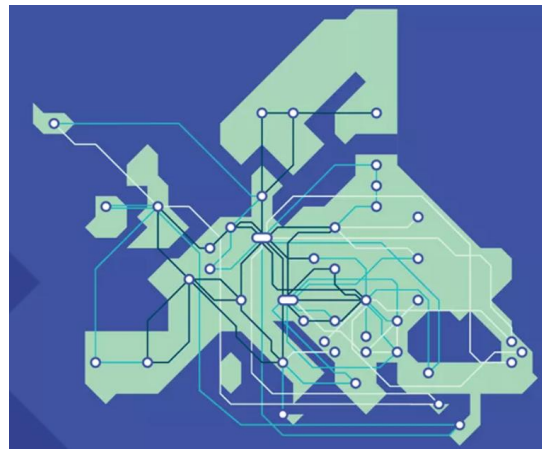
Evaluation – Sync Latency
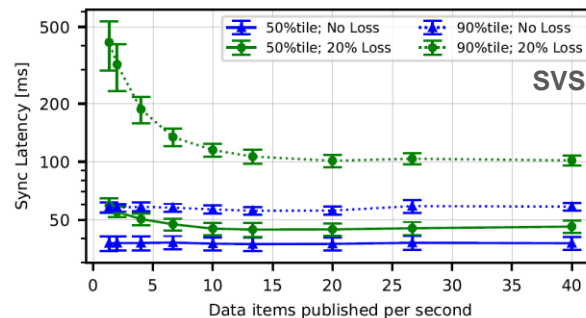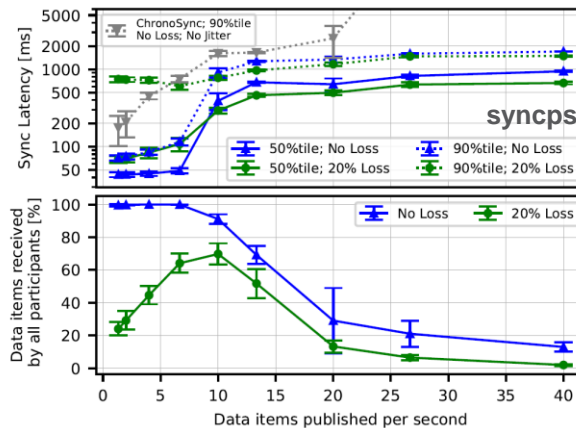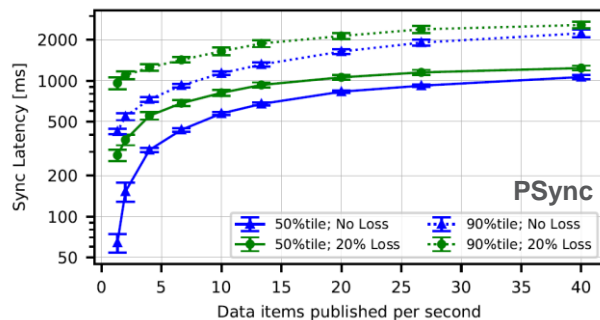
Evaluation – Sync Latency and Overhead
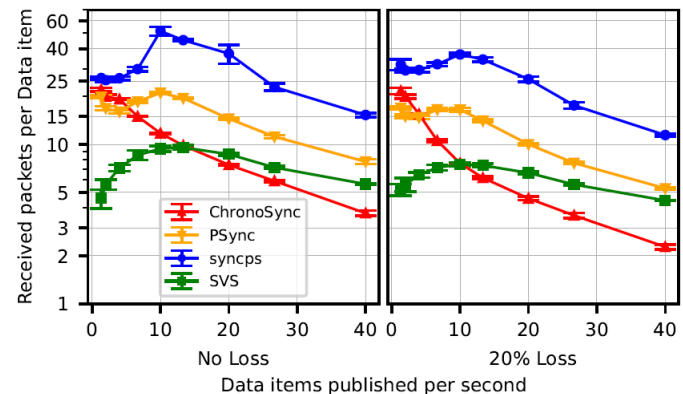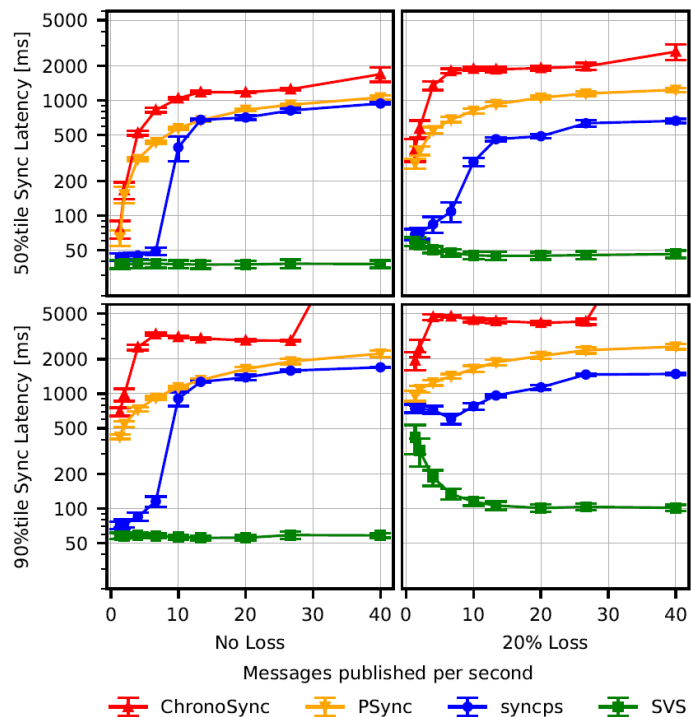
Scaling SVS to Large Groups

Carrying raw state raises the concern of the size of a Sync Interest, which is limited by the network MTU. State vector grows linearly with the number of producers.

- Each state vector is independently usable

- Partial state vectors are also usable
  - Does not need to carry sequence number for all producers
 - Advantage of raw dataset encoding

- What is the optimal way to break the state?
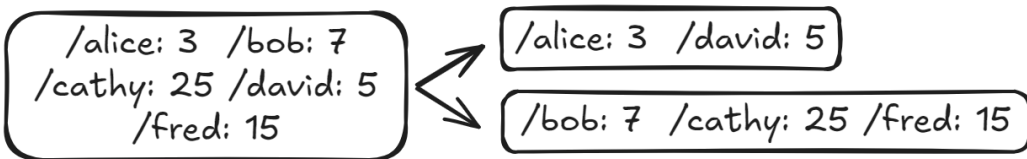
p-SVS Strategies

p-SVS uses partial state vectors for supporting a large number of producers. Each Sync Interest carries only a subset of producers' state.

- **Baseline** – use the entire state vector (only for evaluation)
- **Segmented-Baseline** – segment the entire state and send all
- **Recent** – only include most recent changes
- **Random** – include a random set of producers
- **Random-Recent** – combine these two subsets

p-SVS Evaluation

- Simulated 8 x 8 grid with ndnSIM (ns-3)

- All 64 nodes in one p-SVS group

- 50% packet loss rate (rich connectivity)

- Nodes randomly produce data

- Vary publication rate of the group
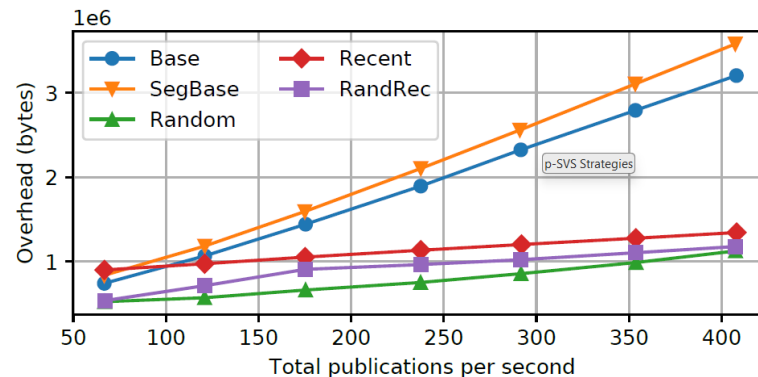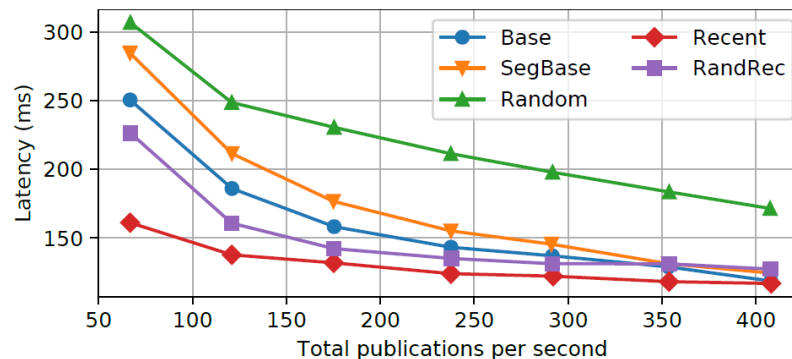
Sequential Naming – SVS Pub/Sub

- Transport needs sequential naming for loss recovery and scalability

- Applications need semantic naming

- Addressing this gap – SVS-PS
  - Map seq# to app name
  - Fetch the app name; may optimize out

- Familiar publish-subscribe interface
  - Hide segmentation and security details
  - publish(name, data)
  - subscribe(prefix, callback)
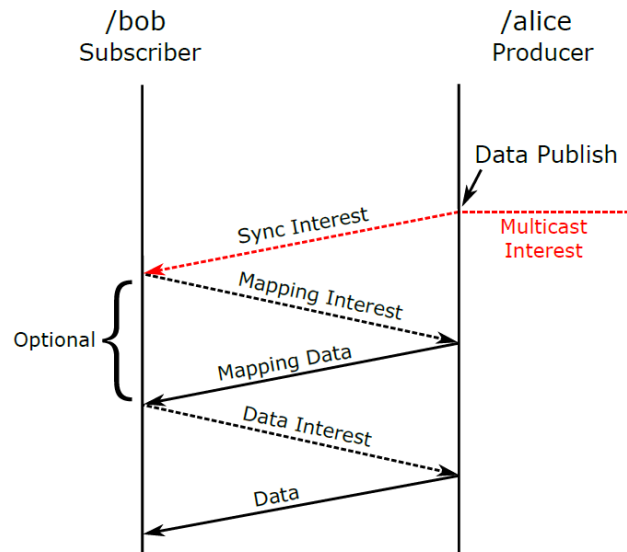
SVS Future Work

- Further evolution of the p-SVS design

- Automated timer tuning and learning

NDN Distance Vector Routing

SCALABLE NAME-BASED ROUTING

Challenges with NDN routing

- Name-based forwarding needs name-based routing

- Each application announces prefixes
  - Data may need to be globally available

- How to scale name-based routing?
  - Can Sync help here?

Two types of reachability

- Reachability to **routers**
  - Scales with number of routers in the network
- Reachability to **end-user prefixes**
  - Scales with number of applications (in NDN)
- Goal – establish reachability to customer prefixes
  - Too many end-user prefixes → scaling challenges
  - Scale NDN routing with 2-level lookup
    - Let routing protocol build FIB for router reachability
   - Use Sync to build prefix to router mapping
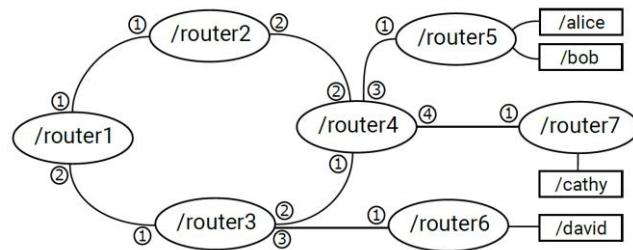
Ideas from existing Intra-AS routing

- In practice, intra-AS routing separates router and prefix reachability
  - IGP: builds the intra-AS router reachability table.
  - iBGP: builds the mapping from prefixes to next-hop routers.

- Packet forwarding performs two lookups
  - Use the BGP table to determine the exit router for a destination prefix.
  - Use the IGP table to determine how to reach that exit router.

- Existing routing protocols do not explicitly or effectively separate the two
  - OSPF: 2 separate LSA types, but both sent together
 - BGP: propagates prefix reachability only

Why Distance Vector?

- Does not need topological map
  - Every router exchanges distance vector *only with neighbors*
  - No need to flood the whole network with updates

- Low overhead
  - Fewer updates – changes are only propagated as far as needed
  - Simpler computation

R2=1, R3=1, R4=1
R5=2, R6=3
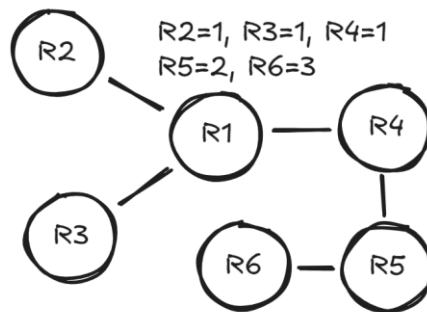
Mitigating routing loops with DV

- RIP (a DV routing protocol) has a bad reputation for creating routing loops

- With adequate topological redundancy, count to next path instead of infinity

- Transient routing loops may still exist, resulting in packet looping in the data plane

- NDN stateful data plane breaks these loops
  - PIT / DNL detect looped packets
  - Forwarding strategy can work around them

Alice=2 .. 5 .. 8

Alice=1 .. 4 .. 7

Alice

R2

R1

R3

Alice=2 .. 3 .. 6
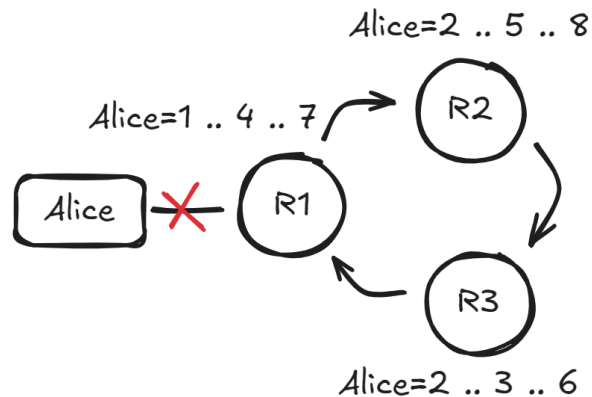
ndn-dv components

- RIB – distance to each router through each interface
  - Establish router reachability
- Advertisement – **same** message broadcast to all neighbors
  - Distance Vector
  - Extra information for multi-path reachability
  - Local broadcast variant of SVS
- Prefix Table – mapping routers to prefixes
 - Establish prefix reachability

ndn-dv example

Destination	Intf (1)	Intf (2)	Intf (3)
/router1	1	3	∞
/router2	2	2	∞
/router4	3	1	∞
/router5	4	2	∞
/router6	∞	∞	1
/router7	4	2	∞

TABLE I: RIB at router 3 in our example.

Destination	Next Hop	Cost	Other
/router1	/router1	1	3
/router2	/router1	2	2
/router3	/router3	0	∞
/router4	/router4	1	3
/router5	/router4	2	4
/router6	/router6	1	∞
/router7	/router4	2	4

TABLE II: Advertisement generated by router 3.
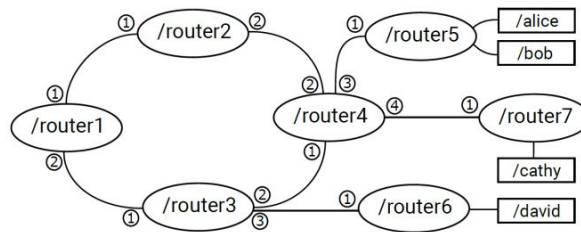
Name Prefix	Exit Router
/alice	/router5
/bob	/router5
/cathy	/router7
/david	/router6

TABLE III: Global prefix table in our example.

| Name Prefix	Next Hops
/alice	intf=2 (cost=2), intf=4 (cost=4)
/bob	intf=2 (cost=2), intf=4 (cost=4)
/cathy	intf=2 (cost=2), intf=4 (cost=4)
/david	intf=3 (cost=1)

TABLE IV: FIB at router 3 in our example.

ndn-dv update algorithm

Algorithm 1 Updating RIB from Advertisements

function COMPUTERIB(neighbors)
    rib   ▷ return value
    for each n **in** neighbors **do**
        if n.advert **is not null then**
            for each entry **in** n.advert **do**
                $cost \leftarrow entry.cost + 1$
                if entry.nexthop **is self then**
                    if entry.other **is not null then**
                        $cost \leftarrow entry.other + 1$
                  else
                    continue
                  end if
                end if
                if $cost \geq max$ **then**
                    continue
                end if
                $rib[entry.dest][n.intf] \leftarrow cost$
            end for
        end if
    end for
end function

Multipath

Poison Reverse

Break Infinity

Prefix Table Sync

- Synchronized with SVS-PS globally
  - Hop-by-hop Sync Interest flooding
- Not affected by topology change
- Two-step Interest forwarding
  - Lookup exit router for the prefix using mapping table
  - Lookup next hop for that router

Name Prefix	Exit Router
/alice	/router5
/bob	/router5
/cathy	/router7
/david	/router6

TABLE III: Global prefix table in our example.

Name Prefix	Next Hops
/alice	intf=2 (cost=2), intf=4 (cost=4)
/bob	intf=2 (cost=2), intf=4 (cost=4)
/cathy	intf=2 (cost=2), intf=4 (cost=4)
/david	intf=3 (cost=1)

TABLE IV: FIB at router 3 in our example.

Routing with Security built-in

A routing daemon is an application in itself; ndn-dv is secured just like any other application running over NDN through NDN security primitives

- Each router undergoes security bootstrapping
  - Trust anchor of network operator
  - Certificate of router's identity
- All updates produced as signed NDN data
  - One advertisement for all neighbors
  - One prefix table update for entire network
- Implementation – LightVerSec trust schema

Design overview of ndn-dv
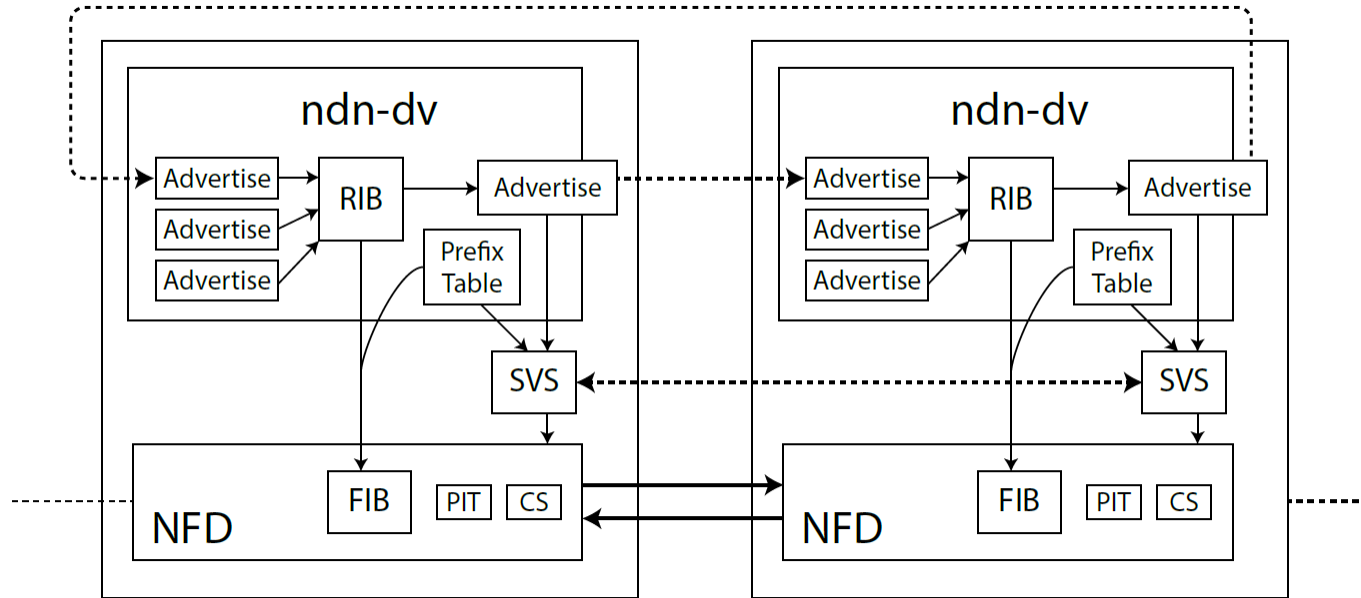
Preliminary Evaluation

- 52-node topology, 50ms delay on each link
  - Emulated in MiniNDN

- 80 randomly setup flows, 100 data interest per second
  - Emulate application behavior

- Mean-Time-To-Failure (MTTF) = 4000s → 300s

- Mean-Time-To-Recovery (MTTR) = 120s

- Measure fraction of satisfied Interests

Evaluation with Retransmissions
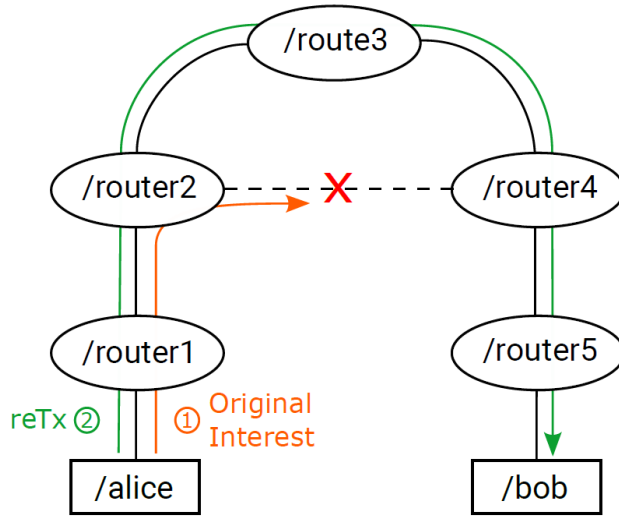
Best route w/ retransmissions
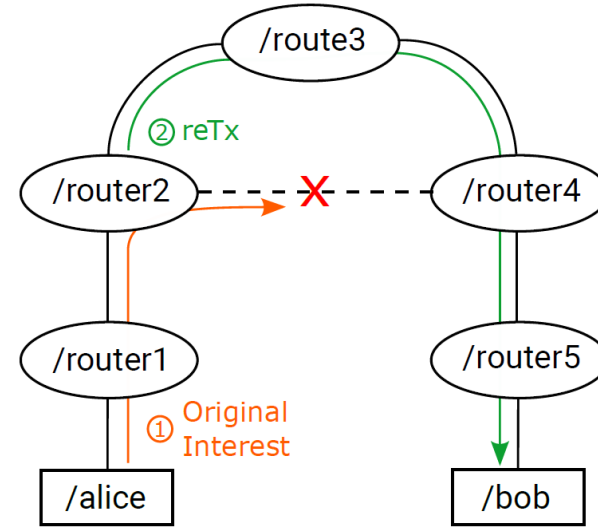
Best two routes strategy
(experimental)

Evaluation
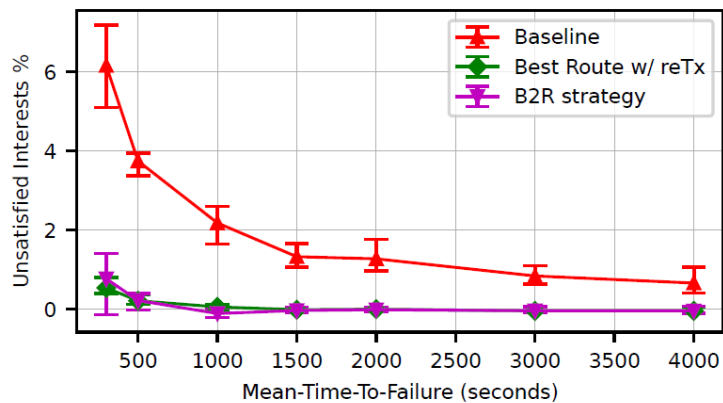
Fig. 7: Fraction of unsatisfied Interests with ndn-dv.
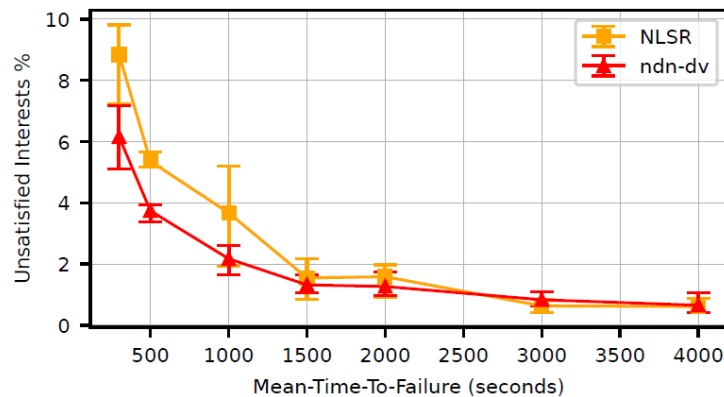
Fig. 8: Baseline comparison of ndn-dv with link-state.

Ownly

DECENTRALIZED WORKSPACE WITH NDN
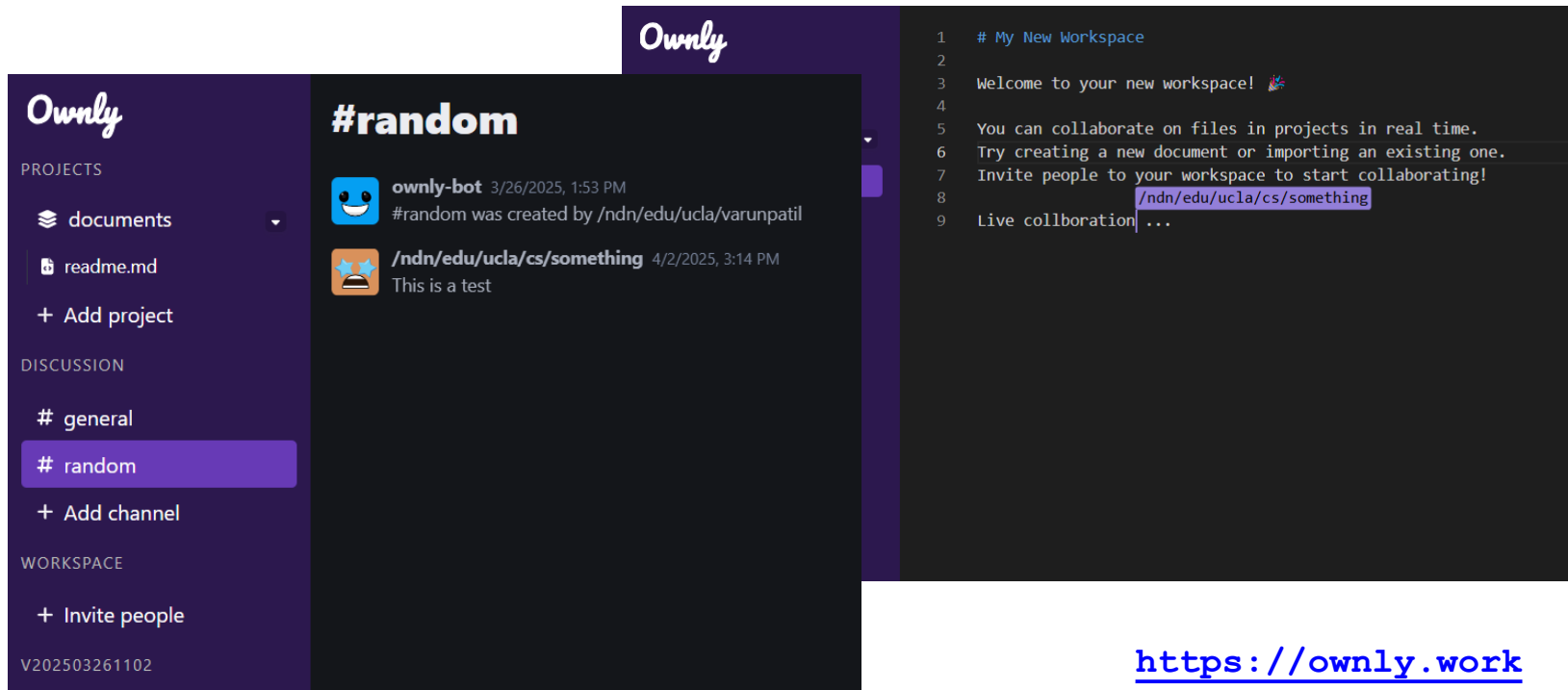
What is Ownly?

- Fully decentralized collaborative workspace run by end users

- Built over the Named Data Networking stack
  - Secured with name-based security primitives
  - Communication based on NDN primitives

- End-to-end security

- Reimplementation of *NDN Workspace* [1]
  - Several components redesigned with lessons learned
  - Focus on end-user usability: overhaul of UI

https://ownly.work

[1] Ma, Xinyu. *Towards Decentralized Applications*. University of California, Los Angeles, 2024.

What can Ownly do?

https://ownly.work

Design and Implementation Goals

- Build a real decentralized application
  - Data-centric – applications handle data rather than channels
  - Secure data directly – eliminate gatekeepers
  - Peer-to-peer – no dependency on central servers
- Implement a generalized set of libraries usable in other applications
  - Implemented as a part of the NDNd standard library (Golang)
- Focus on usability and user experience
  - Minimize human costs

Ownly Architecture
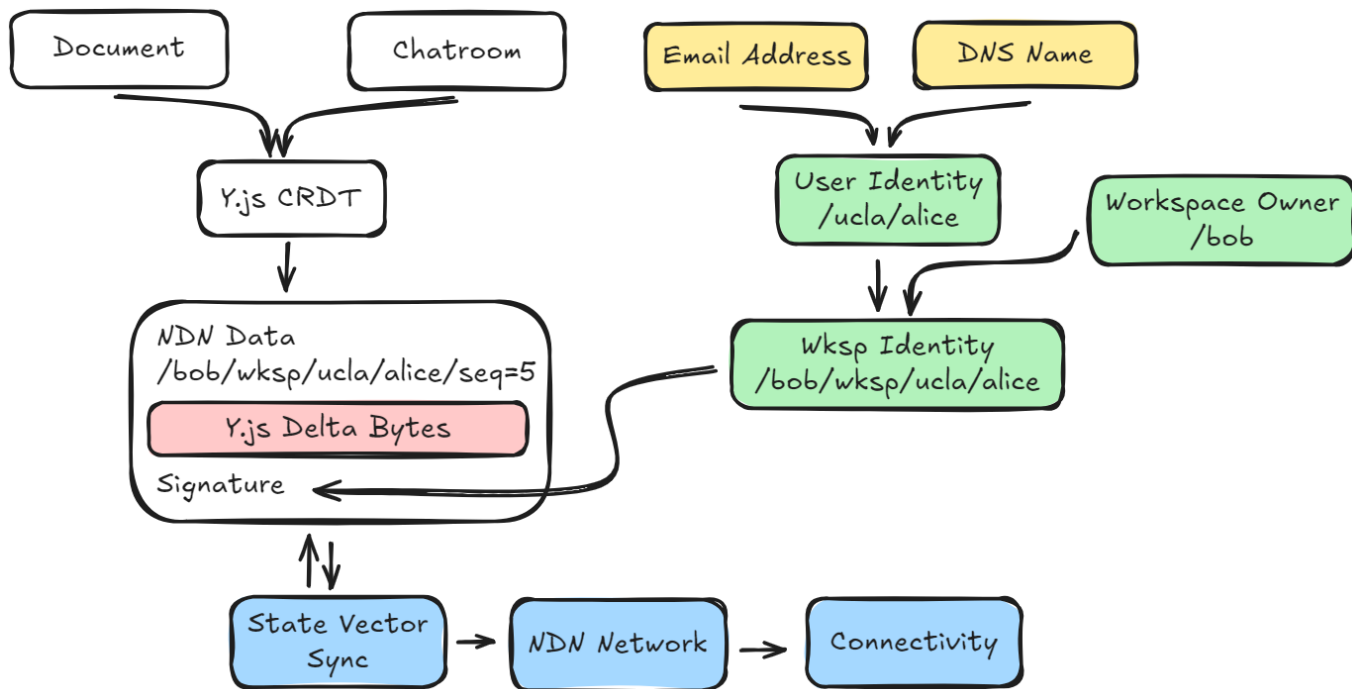
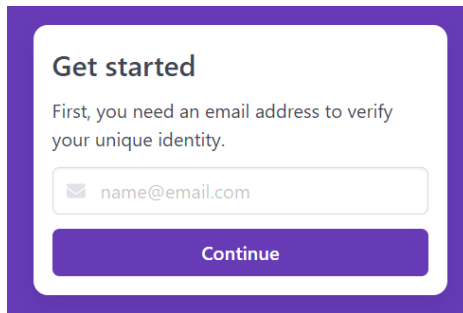# Exchanging App Data Securely: Ingredients

- Naming
  - Each user has a unique name – DNS and derived names
  - Each piece of data is uniquely named and immutable

- Security
  - Each piece of data is directly secured

- Sync
  - Consumers are notified of data production

- Storage
  - Make Data available for consumption all the time

Naming Users and Data

- Using DNS and DNS derived names:
  - DNS name delegation
  - With an assigned DNS name N, one can assign any other names under N
    - "ucla.edu"
- Name each user
  - "gmail.com/@alice"
- Name each application instance
  - "ucla.edu/IRL"
- Name each piece of data
  - "ucla.edu/IRL/gmail.com/@alice/DATA/seq=8"
- Important: well-designed **naming conventions**

Ownly Security Bootstrapping

- Install Trust Anchor out-of-band
  - Bundled with the application code
  - Can be user-configurable

- Reuse external identity for users
  - Email address
  - DNS namespace ownership (planned)
  - Requirement: unique and verifiable

- Verify external identity and issue identity certificate
  - NDNCERT protocol, CA running on global NDN testbed
    - Email Challenge
  - Certificate proves user identity to other Ownly users
  - NDNCERT CA only verifies identity, does not control application

Get started

First, you need an email address to verify your unique identity.

✉ name@email.com

Continue

Security Policies – Trust Schema

- Ownly uses LightVerSec to define trust policy

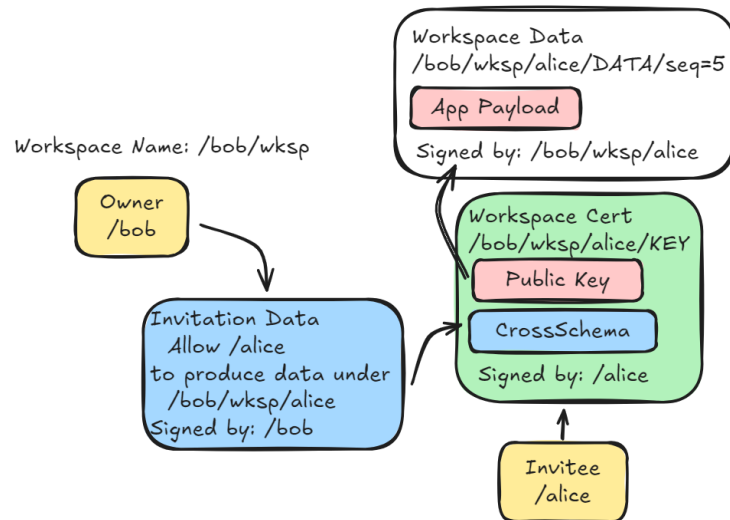- Static schema bundled with compiled application

```
// Only owner can sign all user certificates
// The delegation will happen using a separate CrossSchema
#user_cert: #owner/wksp/#user/#KEY <= #owner_cert
#owner_cert: #owner/wksp/#owner/#KEY <= #owner_id_cert
#owner_id_cert: #owner/#KEY <= #testbed_site_cert | #testbed_root_cert

// Testbed trust model
#testbed_site_cert: /"ndn"/_/_/#KEY <= #testbed_root_cert
#testbed_root_cert: /"ndn"/#KEY

// Project sync group
#proj: #owner/wksp/proj
#proj_data: #proj/#user/_/_ <= #user_cert
#proj_blob: #proj/#user/_/"32=blob"/_ <= #user_cert
```

Dynamic Security Policies – CrossSchema

- Security policies may need to change at runtime
  - E.g. inviting users to a workspace
- Break up schema into smaller pieces
  - These are generated at runtime
  - The schema itself is a signed NDN data
- Producer attaches required schema to NDN Data
  - NDN Data describes how it can be verified
 - KeyLocator + CrossSchema

Use of SVS in Ownly

Transparent In-Network Storage – Sync Repo

- Repo runs as a network-provided service
  - Run by a provider or ISP for a cost
- Application asks repo to join Sync group
  - Repo announces app data prefix
- Repo fetches all data on the group
  - Verify data authenticity before storage
  - Does not decrypt – storage does not see data
  - Makes data available even producer is gone
- Repo tracks the latest data production via Sync
  - "Transparent" – application does not interact with storage
  - Relays Sync Interests to new users for dataset state

Repo
/bob/wksp

Alice
/bob/wksp/alice

/bob/wksp/alice/DATA/...

Consumer
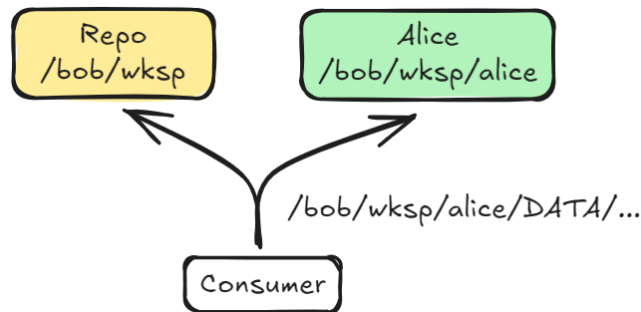
Deployment

- NDN network connectivity using NDN Testbed
  - https://named-data.github.io/testbed/
- Currently uses Testbed NDNCERT infrastructure for identity verification
- Static application hosted on Netlify
  - https://ownly.work
- Has been in active use
  - IRL group meetings for the last 4 months
  - NDN project team weekly call agenda
  - During NDN Community Meeting 2025

Technical Contributions

- Real-world usage of State Vector Sync
  - Key component for decentralized applications

- Improving SVS through feedback
  - Handling multiple instances of the same identity
  - Instance-specific bootstrap time
  - Improvements to the SVS-PS API

- Optimizations, e.g. timers and snapshots

- New security primitives

Ongoing and Future Work

- Completing Encryption Implementation
  - Group key management
  - Key rotation
- Verifying data after certificates expire
  - Ideas from NDN DeLorean
- Snapshot efficiency and rollback
  - Make bootstrapping faster
  - Reduce duplication of data

Conclusion

Summary

- Identifying technical issues driving the centralization
  - Lack of secure peer-to-peer communication
    - The first barrier: Users have no identity
  - Reliance on CDNs for content dissemination

- State Vector Sync
  - Resilient, performant and low-overhead namespace synchronization

- ndn-dv
  - Scalable name-based routing
  - Separating router and prefix reachability

- Ownly
  - A real usable NDN application
  - CrossSchema, Sync Repo

Other Contributions

- Implementation of Named Data Networking Daemon (NDNd)
  - Consolidated NDN implementation in Golang
  - Forwarder, ndn-dv, Sync, standard library, security etc.
  - https://github.com/named-data/ndnd
- Implementation of NDN Developer Tooling
  - NDN-Play as an educational tool
    - https://play.ndn.today
  - GUI for MiniNDN
  - VS Code and Chrome extensions

Acknowledgements

Professors

Prof. Lixia Zhang
Prof. Beichuan Zhang
Prof. Lan Wang
Prof. Alex Afanasyev
Prof. Susmit Shannigrahi
Prof. Jeff Burke

Prof. Songwu Lu
Prof. Harry Xu
Prof. Peter Reiher
Prof. George Varghese
Prof. Todd Millstein
Prof. Susan Ettner

Collaborators

Xinyu Ma
Tianyuan Yu
Mark Theeranantachai
Adam Thieme
Davide Pesavento
Junxiao Shi
Philipp Moll
Sichen Song
Yekta Kocaogullar
Haotian Yi
Adam Chen
Omar Elamri
Brad Lowe
Alexander Lane
Hemil Desai

Master's Students

Nishant Sabharwal [SVS]
Peter Wang [YaNFD]
Alan Chiu [Pub/Sub]
Seiji Otsu [SVS Timers]
Tanmaya Hada [DNL]
Nidhi Panchal [Testbed]
Hope Pegah [NDNCERT]
Evan He [ndn-dv]
Jacob Zhi [Prefix Injection]

Department

Joseph Brown
Juliana Alvarez
Helen Tran

Operant Networks

Scott Gray
Randy King
Andrew Bartels
Shawn Green
Alex Shaffer
Mike Herzig
Roger Jungerman
Bhavesh Bhatkar
Keith Rose

Kathleen Nichols
Van Jacobson

and many more …

Thank You