NDN-CloudSync: A Secure Storage Service Running on Multiple Clouds

Ronak Badhe*, Emmett Cocke*, Tianyuan Yu*, Tushar Sood**
* UCLA
** TATA Communications

The Problem

Cloud storage services today are provided by hyperscalers with siloed services

- Difficulties In Data Movement and Consistency across Multiple cloud
- High Data Dissemination Overhead
- Slow and Complex Adaptation to Overload and Failures
- Lack of User-Controlled Data Security
- Securing data across cloud boundaries typically requires rigid VPNs or trusting cloud providers with user data encryption keys

What Users Want

- Dominant cloud usage goal among enterprises is to leverage resources from multiple cloud providers
  - Strong desire to get rid of dependency on specific cloud providers and locations
- The goal of NDN-CloudSync
  - Providing user-controlled end-to-end data security
  - Strong consistency of replicated data across multiple clouds
  - Efficient data replication to multiple storages and data dissemination to multiple users
  - Fast adaptation to component failures

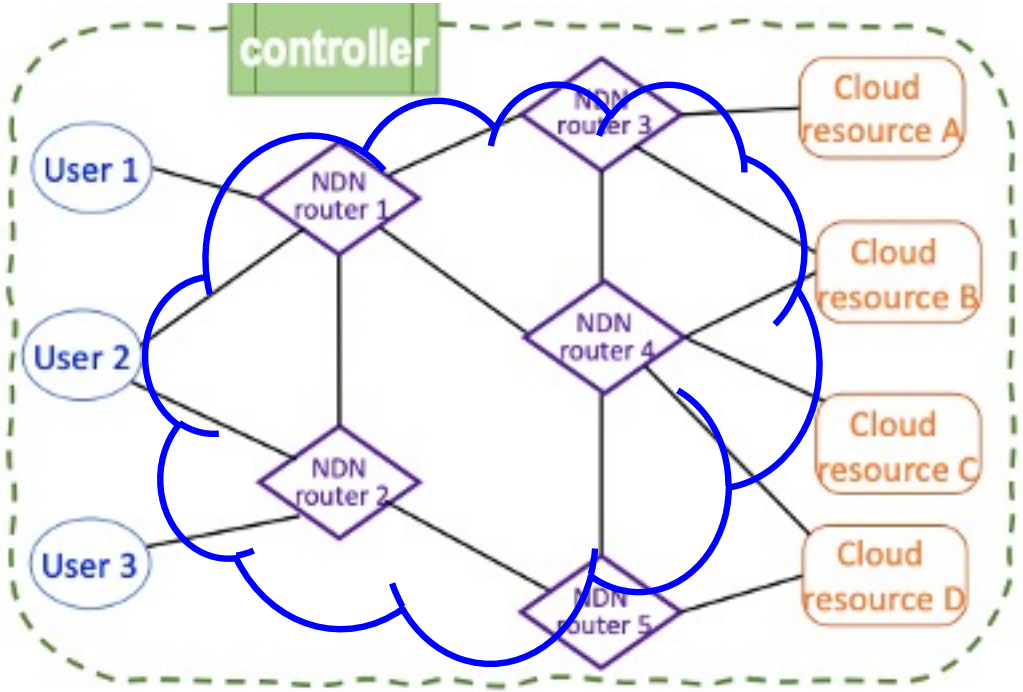
Named Data Networking 101

- An entity: user/host/process that produces and/or consumes data [1]
  - An entity possesses a DNS name, a trust anchor, one or more crypto certificates, and security policies, encoded as trust schemas, from bootstrapping [2, 3]
- NDN treats everything in cyberspace as a piece of named, secured data
  - Data objects, commands to be executed, crypto keys, security policies …
  - Security is bound to data, instead of data containers or channels
  - Data is immutable; changes leads to a different version of the data
- Each administrative domain creates a trust domain and defines trust schemas for all internal entities

[1] “An Overview of Security Support in Named Data Networking”, IEEE Communications Magazine November 2018
[2] “Enabling Plug-n-Play in Named Data Networking”, IEEE MILCOM 2021
[3] “Cornerstone: Automating Remote NDN Entity Bootstrapping”, Asian Internet Engineering Conference 2023

NDN-CloudSync Design

- System controller
- Users: insert and fetch data
  - Insertion may express the degree of replication
- Cloud storage (repos)
- NDN routers, connecting users and repos



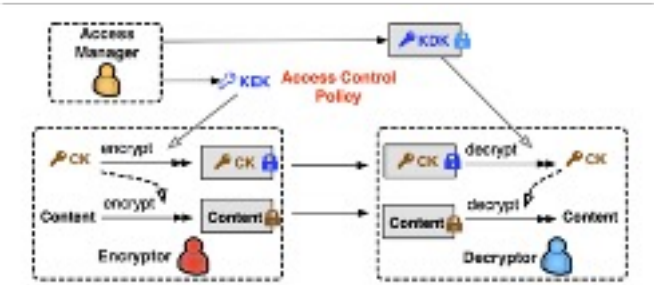
NDN-CloudSync Controller

- Bootstrap users and repos into the CloudSync system
- Serve as the trust anchor for the system
- Define security policies, manage file access control through the management of decryption keys
- Host a Catalog DB which contains the mapping from file names to repos hosting the files

Security

NDNCERT - NDN Certificate Issuance Protocol [4]

- Each entity (user, repo, router) has an identity under the system's trust hierarchy
- Certificates are signed NDN data packets, fetched by name like any other content
- The controller acts as the local CA and issues certs after verification via challenges



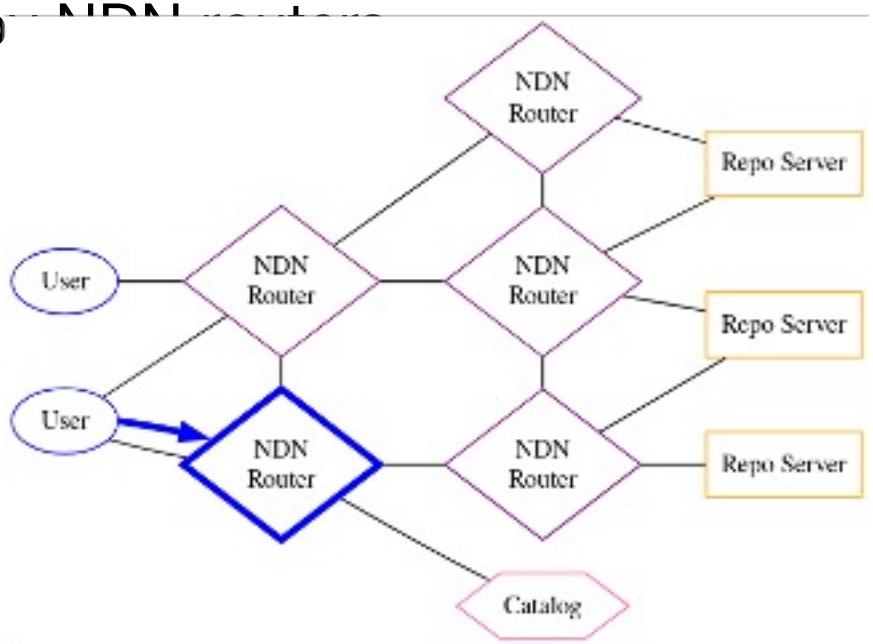
NAC (Name-based Access Control) - automated key management by naming convention [5]

- A user defines a credential namespace scoping who can read what
- Content Key (CK) - fresh per file, encrypts the content
- Key Encryption Key (KEK) - public, used by producers to wrap each file's CK
- Key Decryption Key (KDK) - private, encrypted per authorized consumer so only they can unwrap their copy
- Trust schema validates that enrollment certs chain to the trust anchor

[4] "NAC: Automating Access Control via Named Data", IEEE MILCOM 2018
[5] "NDN Certificate Management Protocol (NDNCERT)", NDN Technical Report NDN-0050 2017

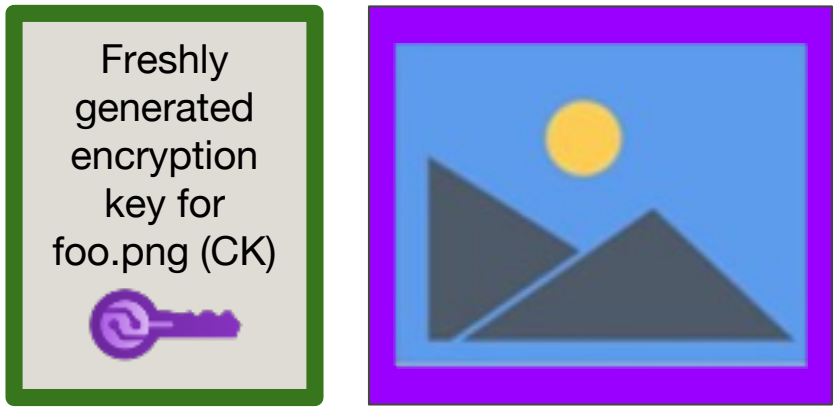
Connectivity

- NDN network consist of router nodes running the NDNd (the Golang forwarder) to form overlay on top of existing UDP/IP network
- Users and servers run NDN-FCH (Find Closest Hub) to establish tunnels to nearb



User File Insertion

1. Produce file with a data name (eg. `/foo.png/v=1778961926914947`)
2. Use NDN security libraries to encrypt and sign each file [4]

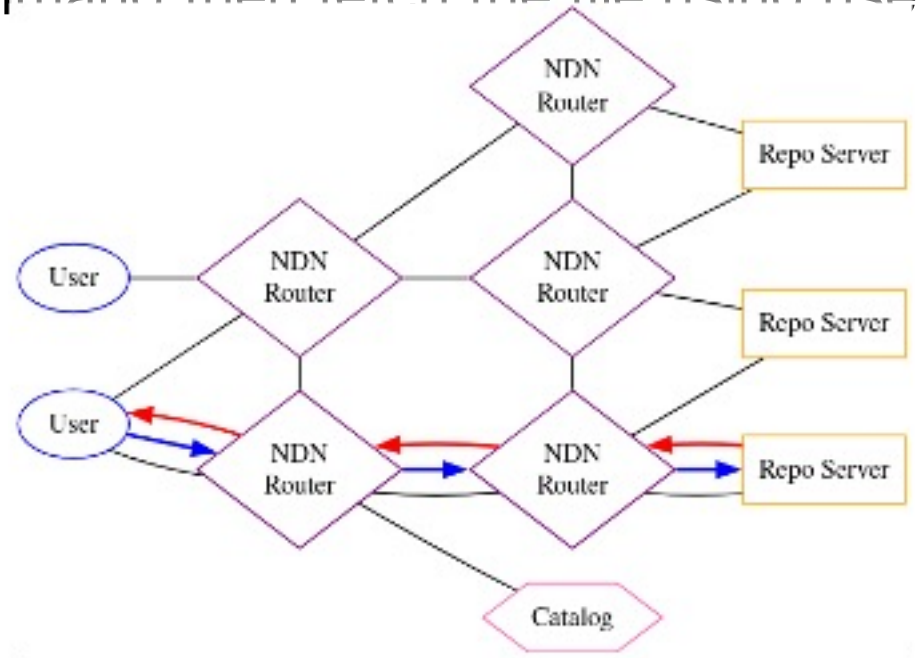


[4] “NAC: Automating Access Control via Named Data”, IEEE MILCOM 2018

User File Insertion

4. Issue *insertion command* to store named, secured file into repo (specifying user name, file name, desired degree of replication)
5. Servers verify command then fetch the file using user name as forwarding hint

Insertion command relies on NDN anycast to send to closest repo server



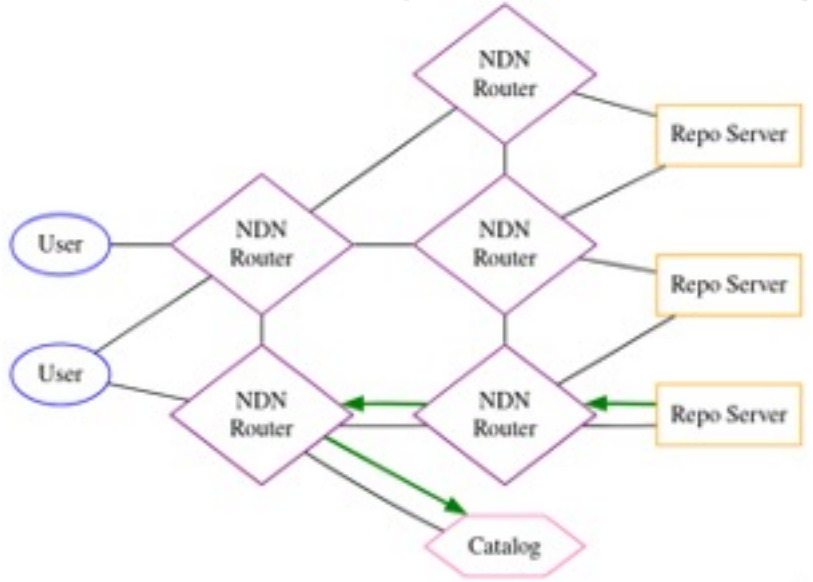
Each repo server register two prefixes

1. Anycast prefix
2. Instance prefix

Repo Server fetch produced file from user

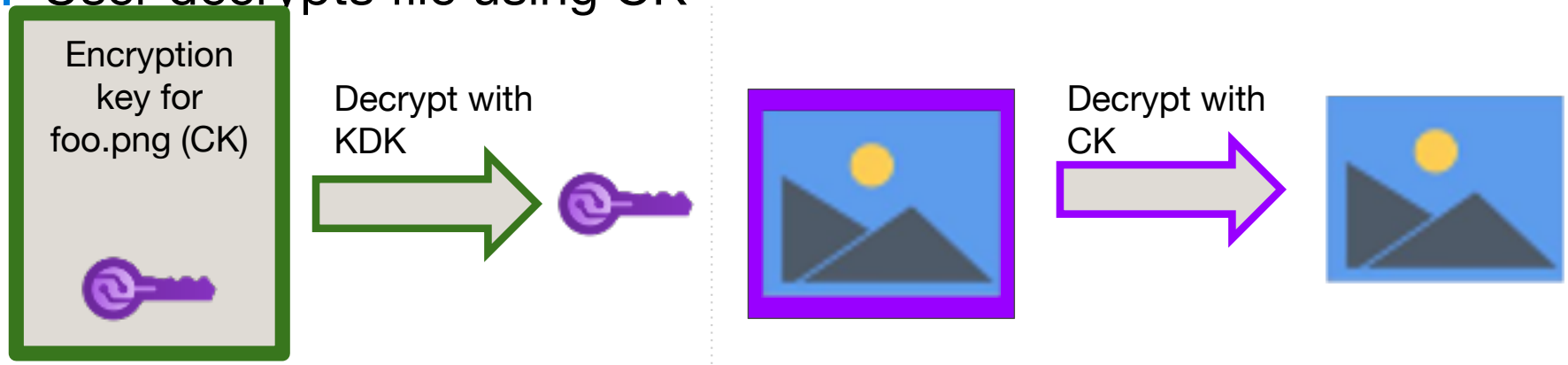
User File Insertion

6. After Server stores encrypted file successfully, send report to catalog
7. Catalog stores in DB `(Server, StoredFileName, FileOwner)`



User File Retrieval

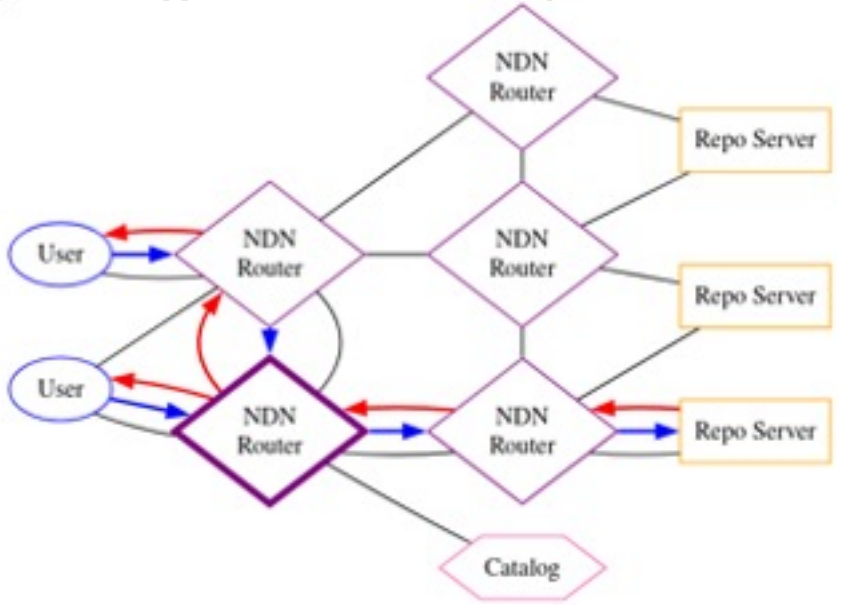
1. User looks up from catalog which repo server stores file
2. User sends out interest for file with specific repo server as forwarding hint
3. User decrypts CK using KDK from NAC
4. User decrypts file using CK



NDN Caching in File Retrieval

- Interest for file solely named with filename + provides repo server name as “forwarding hint”
- Allows NDN cache to work in case of file stored in multiple repo
- Avoids global forwarding table size explosion from 1 prefix per file

File will be served from **intermediate** router’s cache

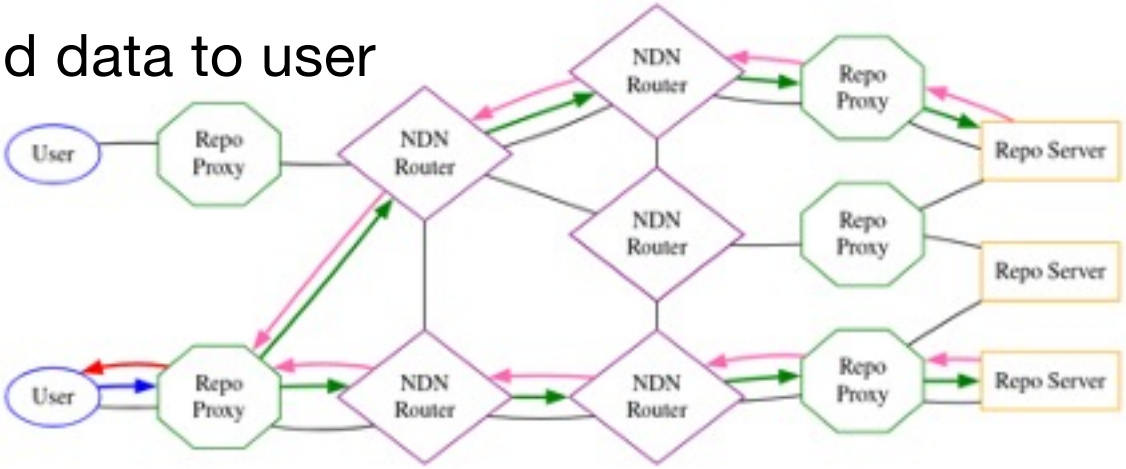


Next Steps - Data Replication

- Add a new repo-proxy service P running on each exit-facing router
- Insertion command interest to P specifies n degrees of replication
- P finds n repo servers and send n insertion command interests
- Each repo sends P a data packet containing ACK upon success
  - The data packet may contain a retry time if file fetch takes longer than expected
- After all ACK, P will send data to user

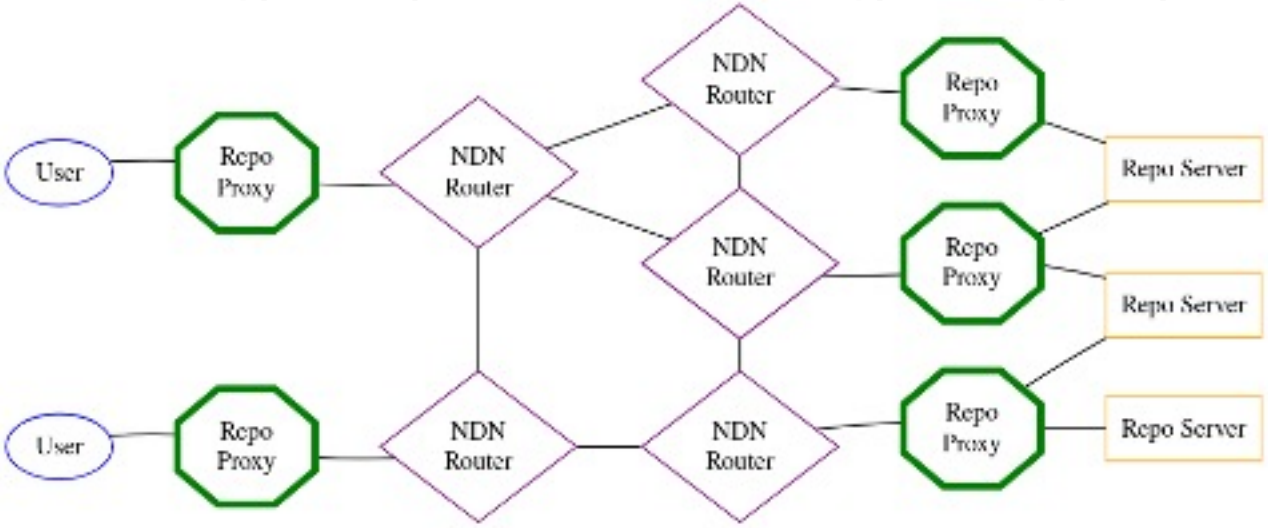
Repo proxy sends n interest commands

Sends data with ACK to user after all servers confirm success



Next Steps - Distributed Catalog

- Use the new repo-proxy service running on each user-facing router
- Host CatalogDB on each user-facing router (replication)
- Upon user interest for a data name, repo-proxy queries local catalog for repo server and attaches to interest as forwarding hint
- Writes to CatalogDB synchronized using SVS group



Next Steps - Compute Replication

- Extend repo-proxy to resource proxy (R-Proxy), hosting both Catalog DB for storage and Catalog DB for compute resources
- When a user issues compute request for n compute resources, the R-Proxy finds n cloud servers which can handle the job

Summary

- NDN-CloudSync achieves its goals by utilizing native NDN functions
  - Providing user-controlled end-to-end data security by securing data and letting users manage decryption keys to control file access
  - Strong consistency of replicated data across multiple clouds via data versioning – NDN's designed-in function
  - Efficient data dissemination to multiple users via NDN data caching
- NDN-CloudSync: demonstrates NDN's utility in meeting industry needs