mGuard: Secure, Real-Time mHealth Data Distribution

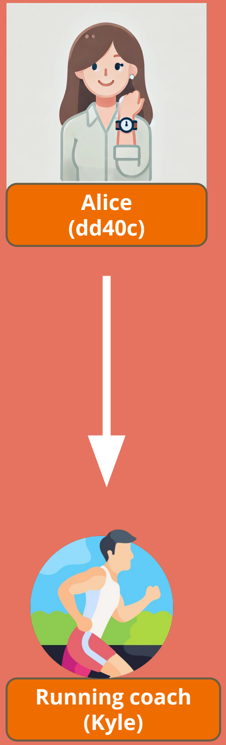
Development Updates and Testbed Experiments

Suravi Regmi, Lan Wang, University of Memphis

Motivation

- Growth of wearable health devices producing sensor data
- mHealth data produced by these devices used for diagnostics, therapeutics
- Existing mHealth system had manual methods for data sharing (e.g. USB)
- Not real time and not secure
- Privacy and access control are crucial for health data

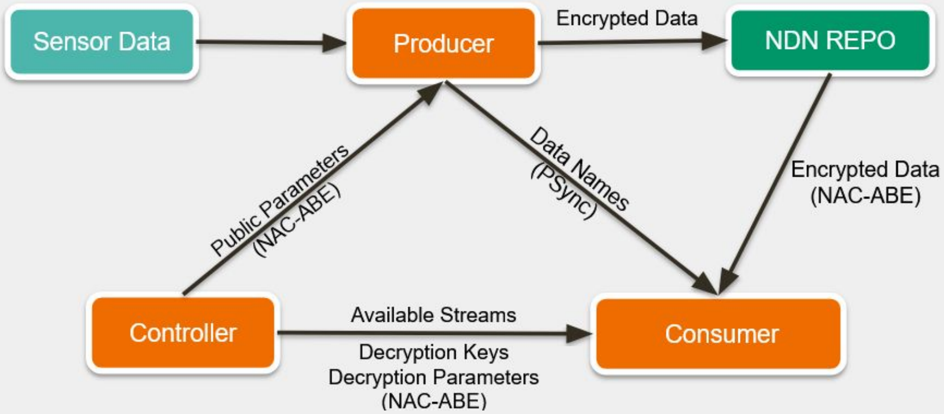
Use Case: Kyle, a running coach, needs real-time access to Alice's(dd40c) accelerometer data for stride analysis.



Alice (dd40c)

Running coach (Kyle)

System Overview



Producer: Receives the data encrypts and publishes

Controller: Controls access policies

Consumer: Subscribes and decrypts authorized data

NDN Repo: Persistent storage

PSync: Synchronizes data availability between producer and consumer

Controller sets Kyle's access policy. Kyle subscribes to /dd40c/motion_sense/accelerometer and retrieves data

Access Control

```
policy-id          1
requester-names    /ndn/org/md2k/kyle
attribute-filters
{
 allow {
       /ndn/org/md2k/dd40c/motion_sense/accelerometer
       /ndn/org/md2k/dd40c/motion_sense/gyroscope
       /ndn/org/md2k/ATTRIBUTE/location/gym
       from “November 1, 2024”
       to “April 1, 2025”
       }
}
```

mGuard uses attribute-based encryption to control access based on:

WHO : the requester is (e.g., Kyle)

WHAT: data is requested (e.g., accelerometer, gyroscope)

WHERE : the data was recorded (e.g., gym)

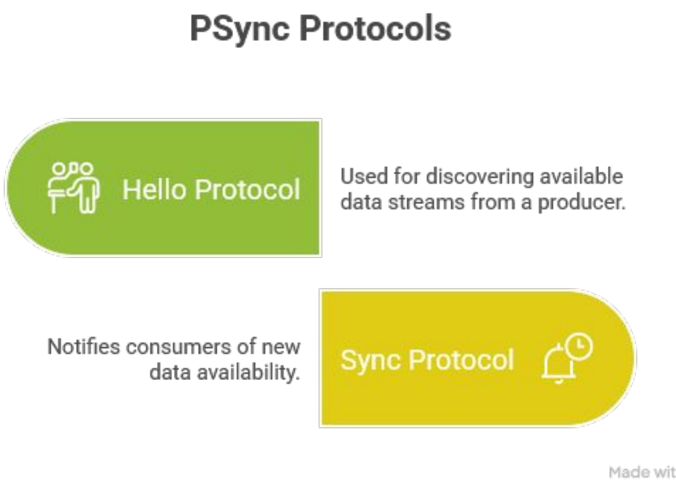
WHEN: the data was generated

Time-Based Enhancement Enables policies like "Kyle may access data generated from November 1, 2024 to April 1, 2025”

Dates need to be converted to unix timestamp before usage with the system

PSync Overview

- PSync enables partial sync , consumers subscribe to specific streams
- Consumers send Hello Interest to get stream list
- Then subscribe using Sync Interests
- Sync updates sent every time there is new data updates
- Consumer fetches data

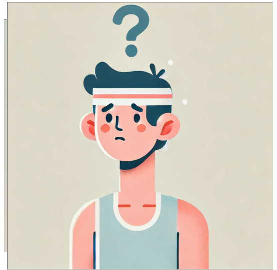


PSync Protocols
Hello Protocol
Used for discovering available data streams from a producer.
Notifies consumers of new data availability.
Sync Protocol
Made wit

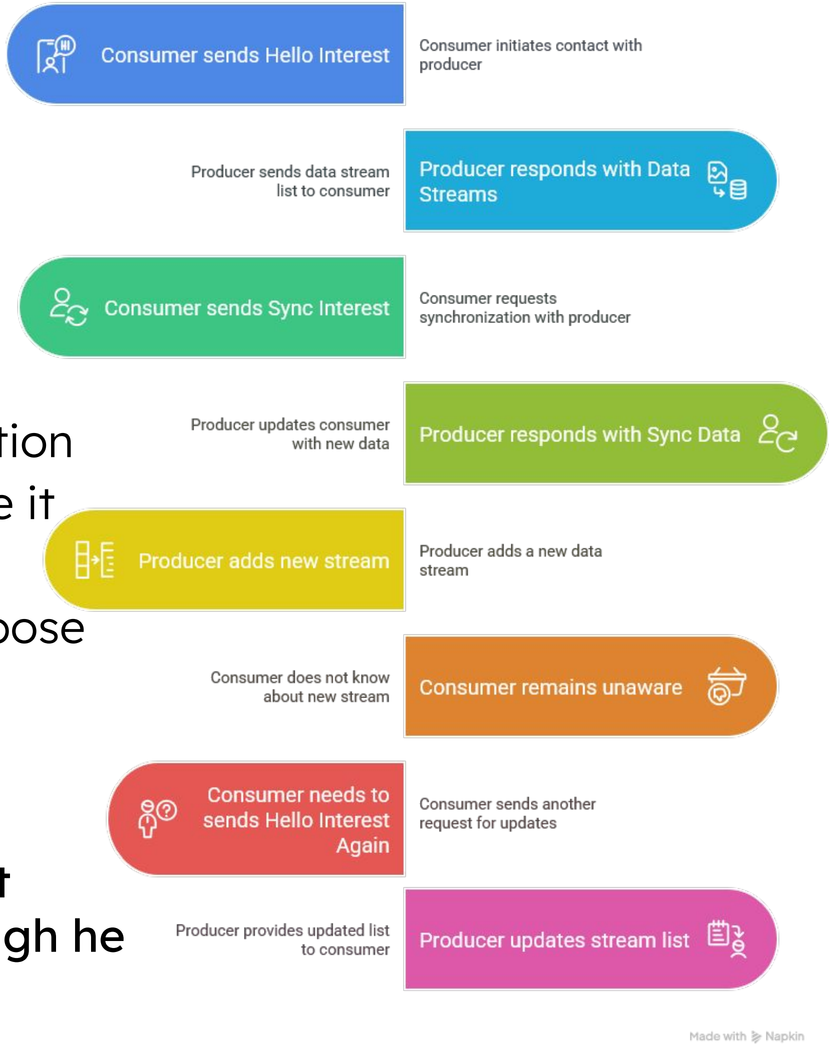
Kyle sees accelerometer stream data and subscribes to it.
Is notified every time there are new updates to that stream.

PSync Issue

- Hello requests used static names -> consumers got cached/outdated lists
- No automatic stream change notification
- User missed new data stream because it wasn't dynamically advertised
- Frequent polling defeated Sync's purpose



Kyle didn't get notified about gyroscope stream. Even though he has access to it.



Consumer sends Hello Interest

Consumer initiates contact with producer

Producer responds with Data Streams

Producer sends data stream list to consumer

Consumer sends Sync Interest

Consumer requests synchronization with producer

Producer responds with Sync Data

Producer updates consumer with new data

Producer adds new stream

Producer adds a new data stream

Consumer remains unaware

Consumer does not know about new stream

Consumer needs to sends Hello Interest Again

Consumer sends another request for updates

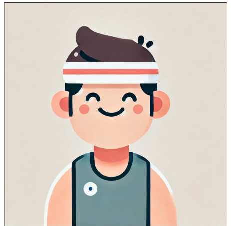
Producer updates stream list

Producer provides updated list to consumer

Made with Napkin

Solution: Default Stream

- Introduced ***/<app_prefix>/default/*** as a default stream
- Combining Hello Protocol and Sync Protocol
- Add a default stream: ndn/org/md2k/mguard/default/<seq>
- Consumers are always subscribed to it.
- Producer publishes stream updates (add/remove) to this stream.
- Default stream seq number updates after each change.
- Sync protocol delivers these updates like any other update.



Kyle is automatically subscribed to default which lets him know hey a new stream is also available for subscription.

Experiments

Topology: Producer (with Repo), controller, and consumer 3 different nodes.
Execution Order: Controller, Producer , consumer start then data generation
Goal: Evaluate system behavior under high data volume with real-time encryption and repo insertion.
Previous Experiments:

- Initially used manual pacing of data generation to avoid system crashes.
- Tests in mini-ndn provided fresh runs and behaviour of system on different nodes.
- Finally to figure out real world issues we also ran the experiment in testbed.

Issues:

- Psync Issues with cached data and no stream update notifications.
- CK(Content Key) used to decrypt data ended up being larger than the data itself.
- After time attribute based encryption it expanded to be 0-4 segments of data packets.
- Time time granularity for CK generation was previously seconds.
- In bulk insert cases for repo insertion there were crashes for large data generation.

Experiments

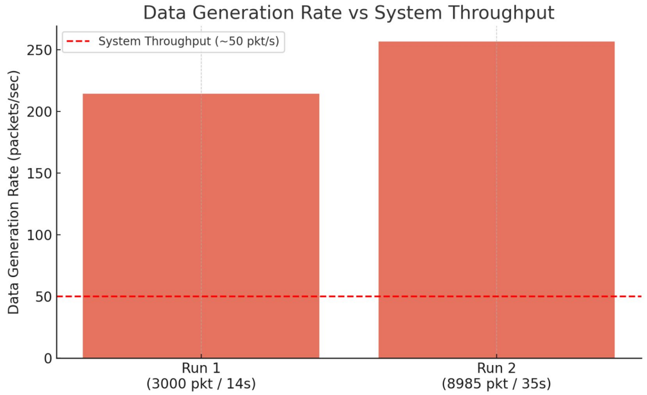
Current System

- Time attribute based encryption and access control
- Psync Protocol Updates.
- CK Granularity Update: Changed from seconds to minutes to reduce the number of CKs and prevent fetch timeouts
- Introduced a token-based scheduler to manage insertion rate
  - 50 tokens per refill
  - Refill interval dynamically adjusted between 200–4000 ms
- System now queues and inserts data reliably under sustained load.

Experiments

Experiment Scale & Throughput

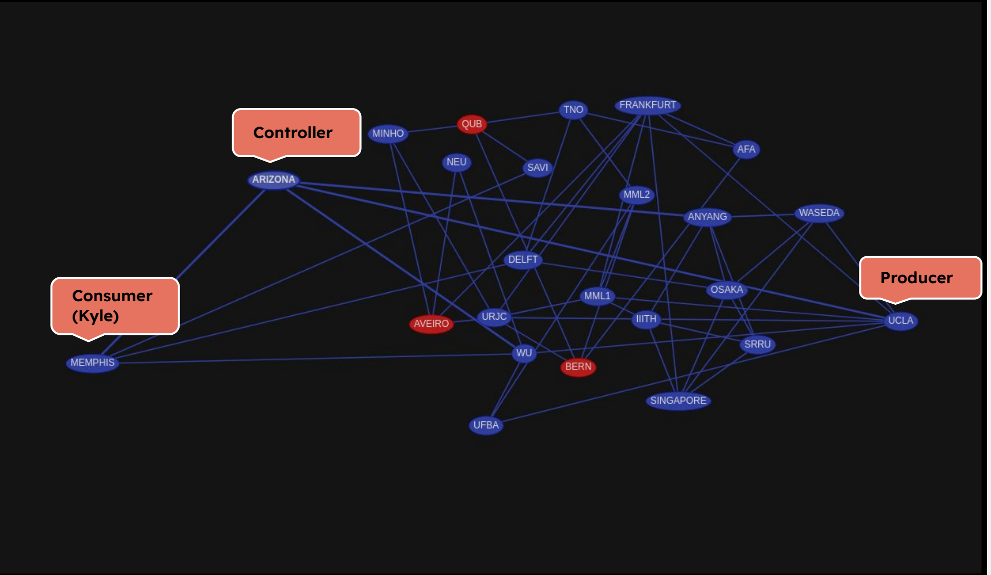
- Batch sizes: 1–5
- Data Streams: 5
- Data per stream: 5–599
- Data packets per run: 3000+
- Data generation:
  - 3000 packets in 14s (~214 pkt/s)
  - 8985 packets in 35s (~256 pkt/s)
- System throughput: ~50 data/sec (generation to storage)
- No crashes — system stable even under backlog



Insights

- Previous set of experiments results we had were for 1-8 data point publication per second.
- Now that's been expanded up to 50 data points
- Queue absorbs bursty input
- Data generation can run at full speed without modification
- Flow control + encryption + repo insert path now robust

Testbed Experiment



Controller

Consumer (Kyle)

Producer

Setup: Producer, controller, and consumer on 3 separate nodes
Experiments: 13 runs

- data: 20–1005 packets
- Manifests: 5–122 data name
- CKs: 20–332 packets

Results:

- Correct Psync Behaviour
- No repo crashes
- Occasional retries for CK fetch
- Correct content fetch

Issues:

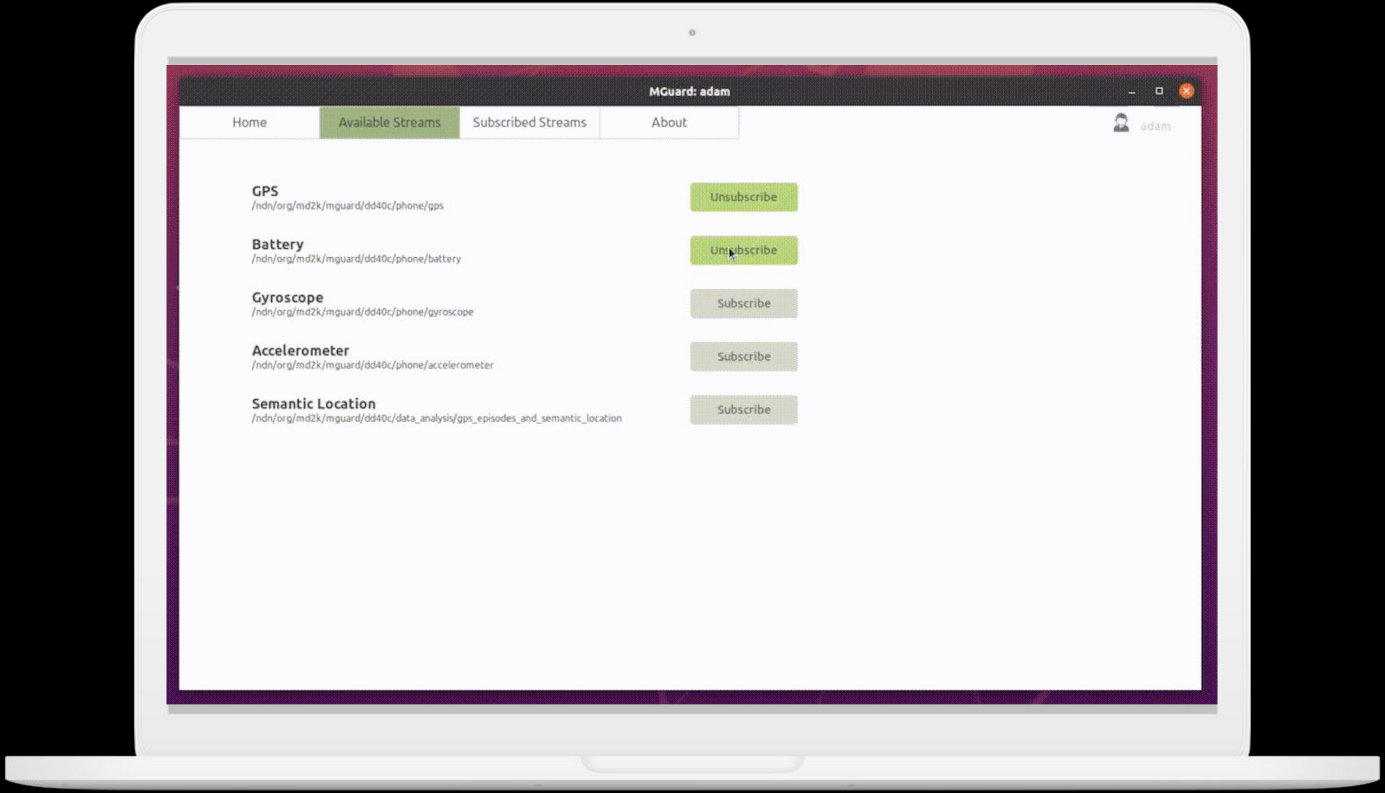
- Partial ck paket being fetched

Next Steps:

- Gather precise performance metrics
- Introduce network variability
- Scale consumers and experiment durations
- Decouple producer and repo to separate nodes.

Next steps

- Extensive experiments on the time attribute based access control.
- Robust error handling on all modules
- Run larger-scale experiments with increased batch sizes and stream counts
- Test with node failures and restarts
- Separate producer and repo for deployment testing
- Improve UI, visualizing streams



GPS
/ndn/org/md2k/mguard/dd40c/phone/gps

Battery
/ndn/org/md2k/mguard/dd40c/phone/battery

Gyroscope
/ndn/org/md2k/mguard/dd40c/phone/gyroscope

Accelerometer
/ndn/org/md2k/mguard/dd40c/phone/accelerometer

Semantic Location
/ndn/org/md2k/mguard/dd40c/data_analysis/gps_episodes_and_semantic_location

Acknowledgement

Advisor: Dr. Lan Wang

Collaborators: Adam Thieme, Saurab Dulal, Tianyuan Yu, Dr. Lixia Zhang

Lab: MD2K Lab, University of Memphis

Funding: NSF Award #2019085

Special Thanks: MD2K team and NDN community for guidance and infrastructure support

Q&A

Any questions?

Thank You!