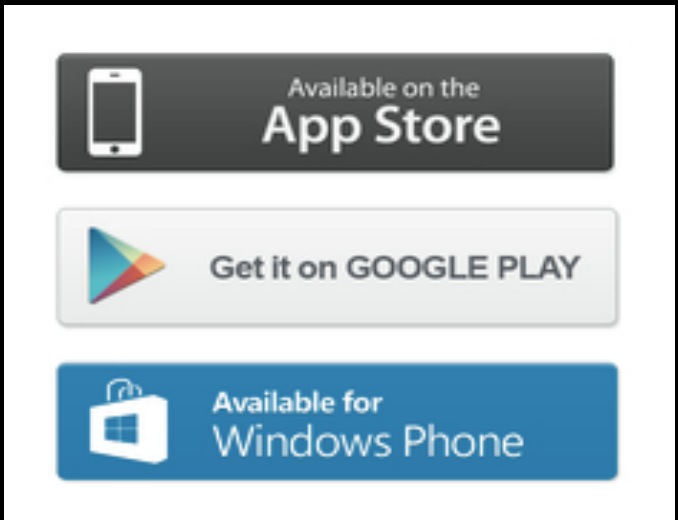
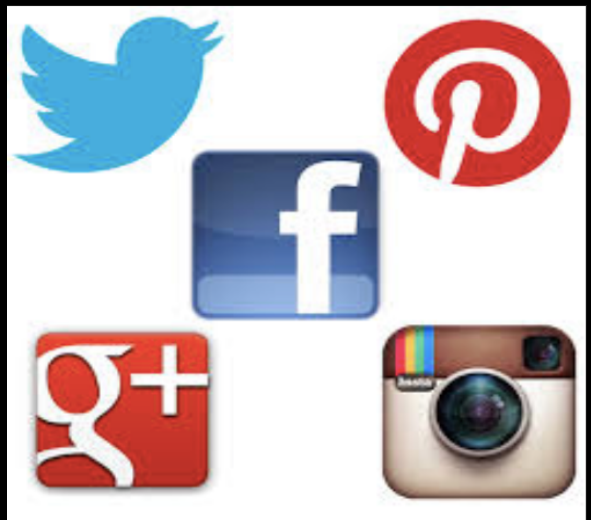
PartialSync: Synchronizing a Partial Namespace in NDN

Minsheng Zhang, Lan Wang
University of Memphis

Application Sync Scenarios

- AppStore: server has millions of apps, each user's phone has a (different) subset of apps
- Facebook: each user interested in a set of feeds
- Subscribe to New York Time news updates
- Twitter, Google+ and etc.
- How to achieve this synchronization efficiently?





Problem Definition

- Data in a data stream: <prefix>/<version>
- Producer: N data streams, P = {p_1, p_2, …, p_N}, p_i is a name prefix
- Consumer: M data streams, Q = {q_1, q_2, …, q_M}, q_i in P & M <= N, Q is *Subscription List*
  - *Subscription list* may change over time
- How to synchronize consumer with producer?

Challenges

- Producer:
  - large number of data streams
  - multiple producers, e.g., replicated repos
- Consumer:
  - any subset of producer’s data streams
  - large number of consumers

Naive Solutions

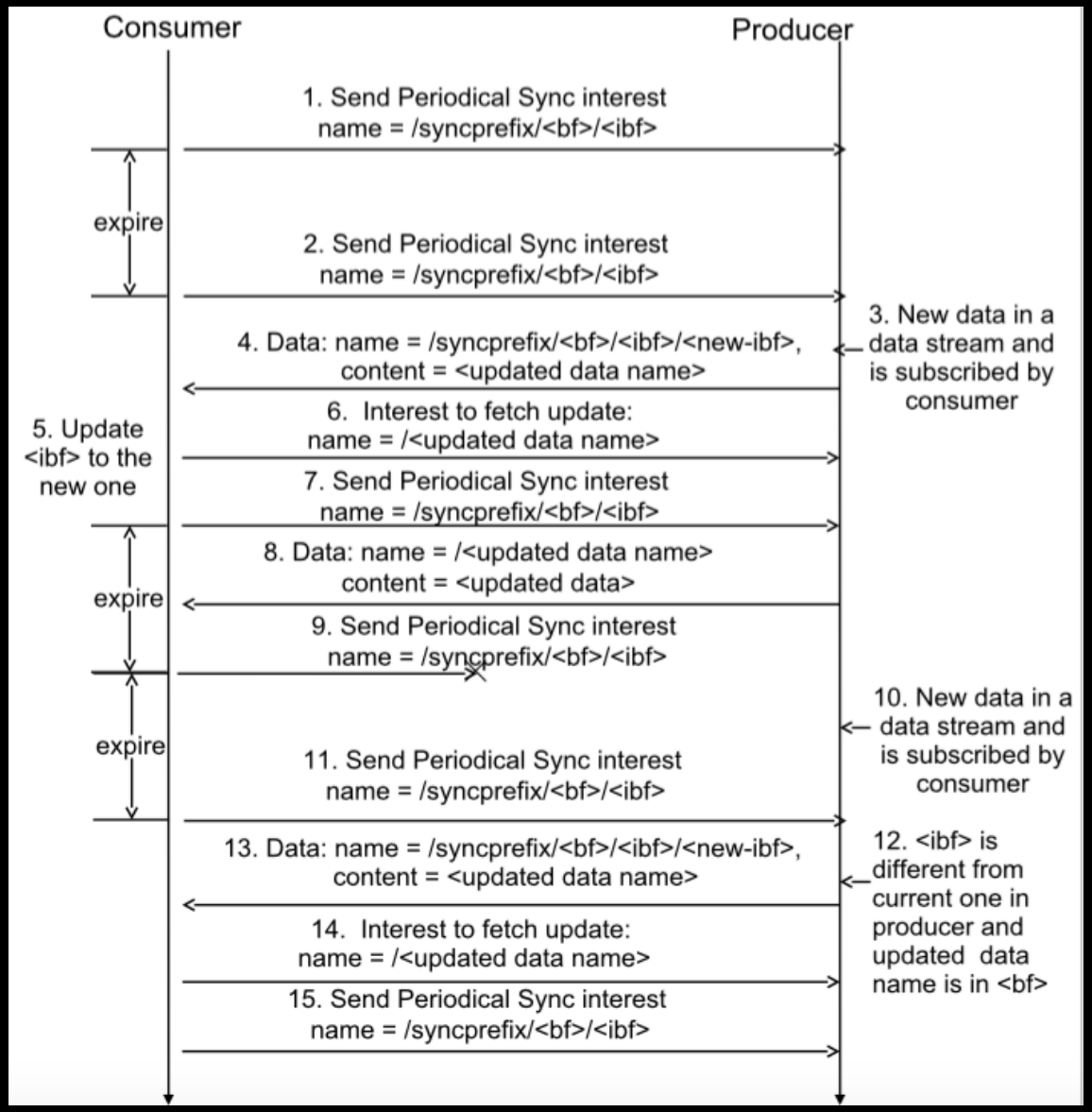
- Consumer sends a pending Interest for each prefix periodically
  - Problem: too many Interests if consumer has a large subscription list

- Consumer syncs all name prefixes the producer has similar to ChronoSync.
  - Two methods:
    - For each different consumer, generate a new sync group with all producers based on consumer's dataset
    - Consumer learns whole dataset and keep all these information sync'ed

Our Solution

- Producer: finding out what has changed
  - Encodes its data streams' status in Invertible Bloom Filter (IBF) and sends IBF in Sync Reply
    - Consumer sends back old IBF in Sync Interest
  - Get changes by calculating difference between current IBF and old IBF in Sync Interest
- Consumer: telling producer its subscription list
  - Encodes the subscribed prefixes in Bloom Filter (BF)
  - Sends BF in Sync Interest
    - Producer queries this BF and determines if an update should be sent to a consumer

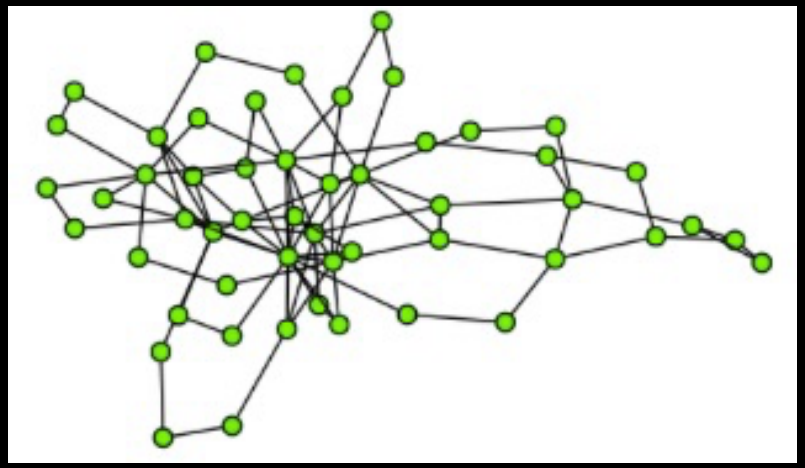
Protocol design



- Case 1: Producer generates new data
  - Check each consumer's sync interest and determine whether it is subscribed by them
- Case 2: Producer receives sync interest
  - Producer calculates changes = current IBF - old IBF
  - Sends sync reply if any change is in consumer's subscription list (BF)

Experiment setup

- Topology: Sprint point of presence
  - 52 nodes, 84 links
  - 1 producer: node with the smallest maximum delay
  - 51 consumers: all nodes other than producer



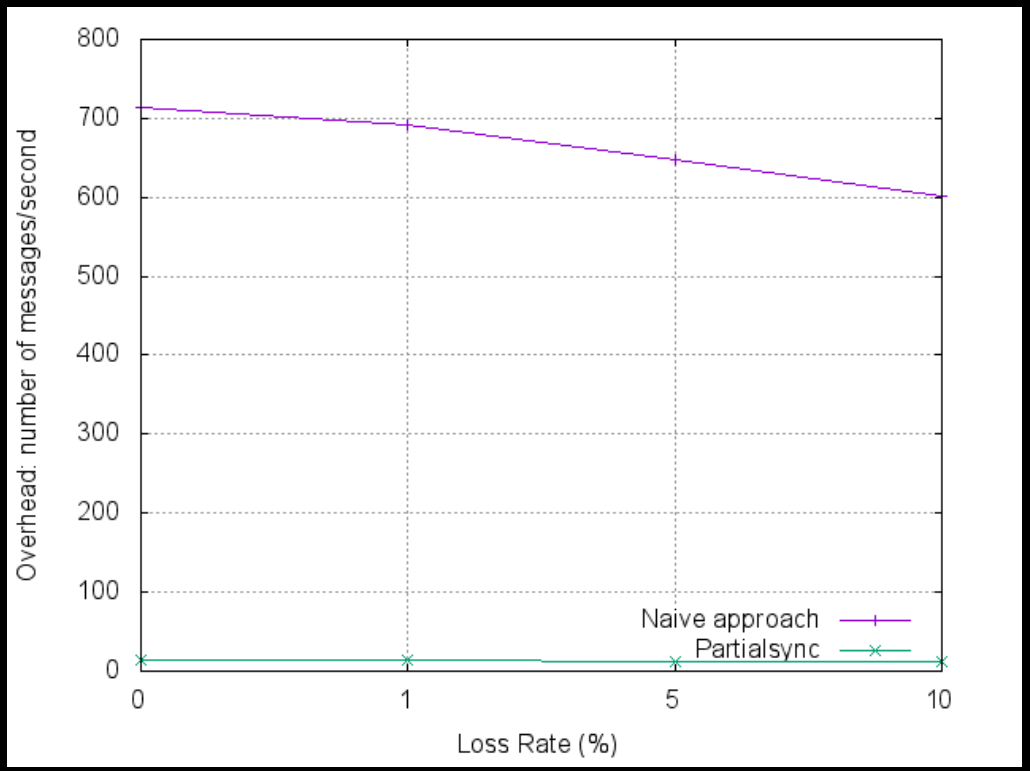
- Different sizes of subscription listen for consumer: 10, 10000
- Different loss rate: 1%, 5% and 10% loss rate
- Platform: mini-ndn

Evaluation

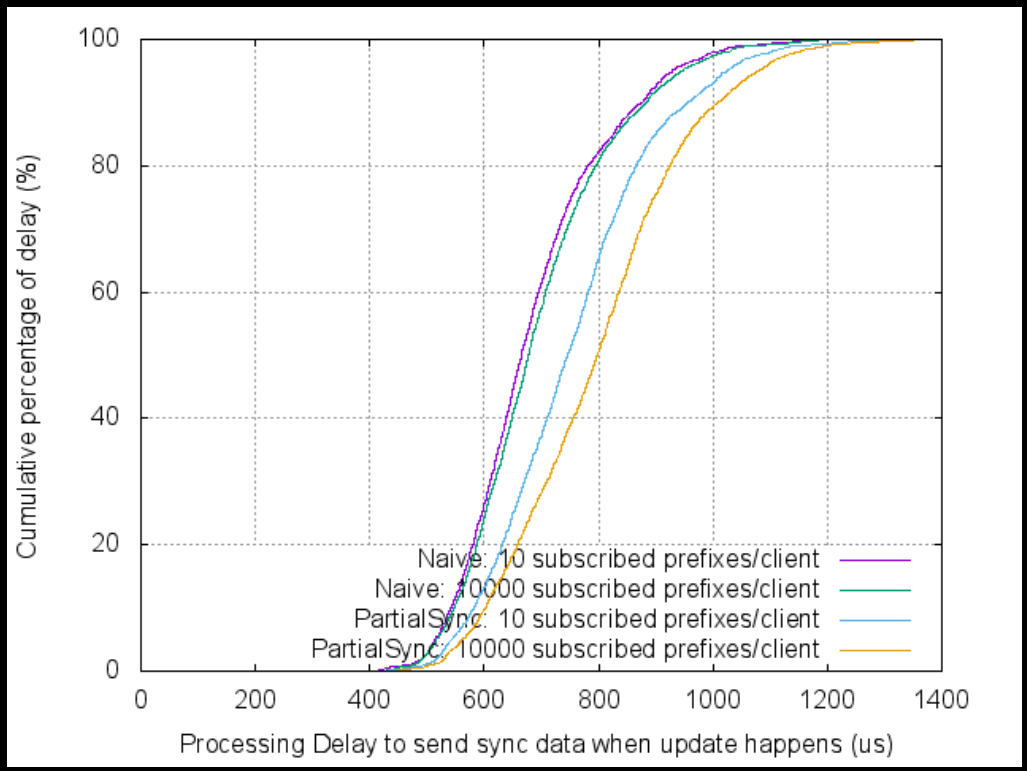
- Comparison methodology
  - Naive approach: for each prefix in subscription list, consumer sends pending Interest to fetch data with next version in the data stream
- Metrics
  - Delay:
    - Delay to get update notification from producer
    - Delay to fetch new data (including notification delay)
  - Message overhead: Number of total interests in the network

Results

Message Overhead



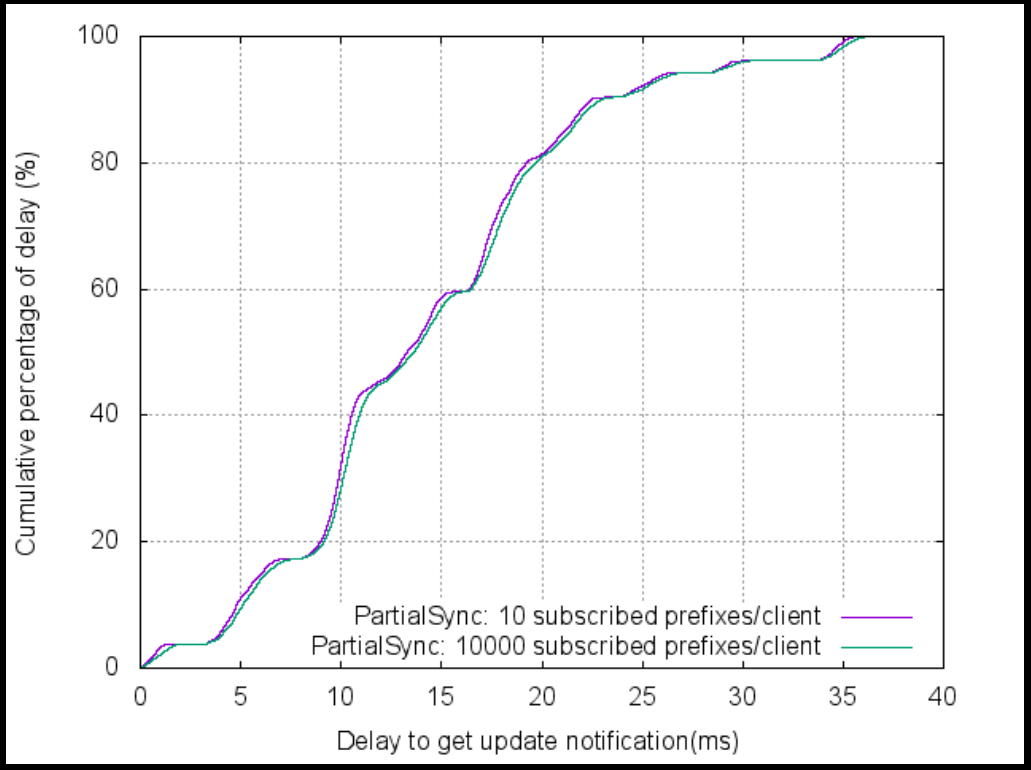
Processing delay



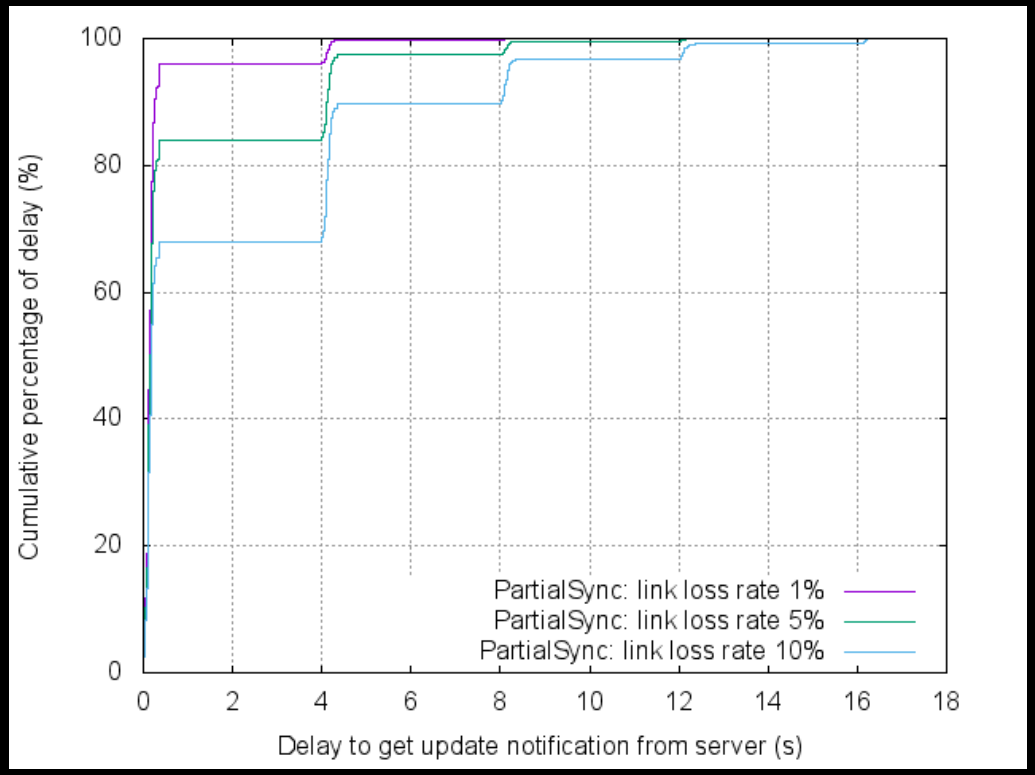
Results (Cont.)

Delay to get update notification

Different size of subscribe list



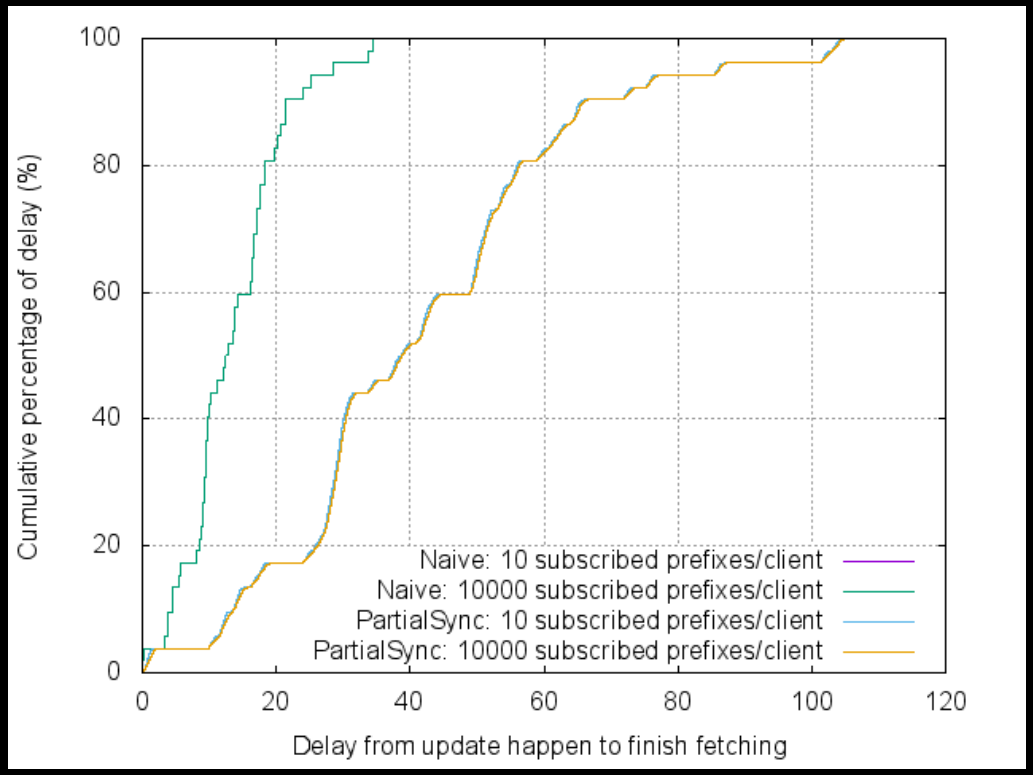
Different loss rate



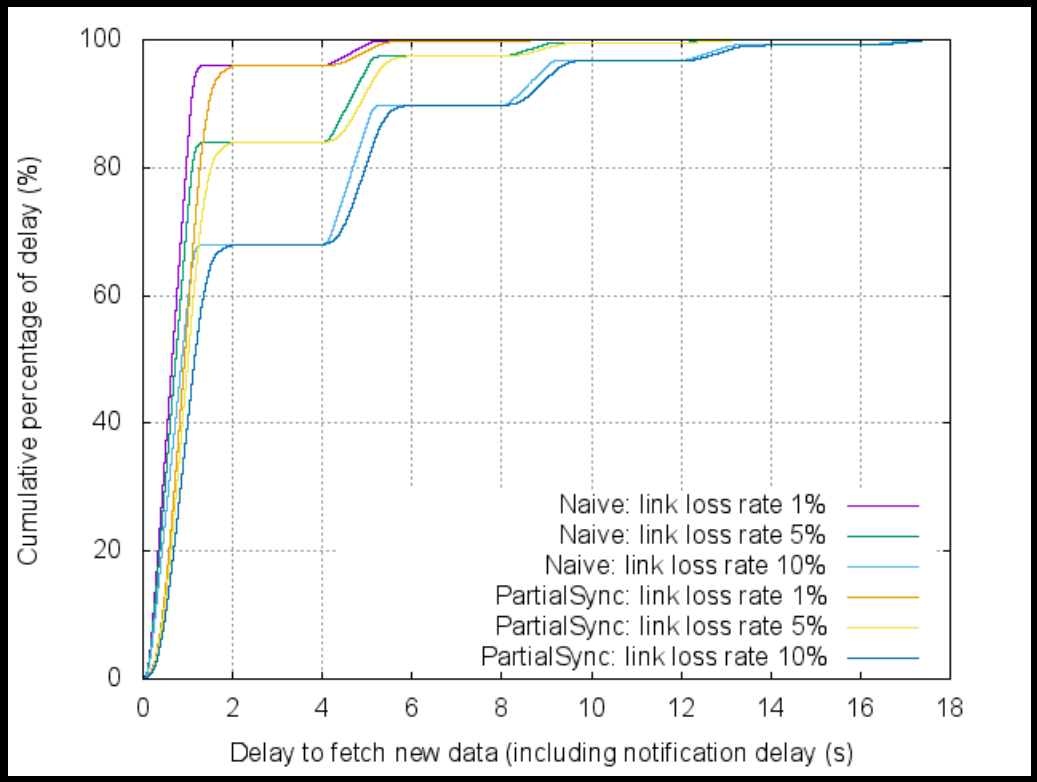
Results (Cont.)

Delay to fetch new data (including notification delay)

Different size of subscribe list



Different loss rate



Related Work

- Chronosync[1]
  - Use digest tree to synchronize participants in a group
  - geared toward full synchronization: all nodes keep knowledge of the whole data streams
- CCNx Sync[2] and iSync[3,4]
  - pair-wise synchronization protocol
  - iSync uses IBF and has lower message overhead than CCNx Sync and ChronoSync
- Existing approaches are not as efficient as PartialSync for large number of subscribers with different and overlapping interest
  - either generate many different Sync groups
  - or sync all the data streams

Conclusion

- PartialSync has significantly lower message overhead than naive solution
- Size of subscription list has little impact on delay and processing overhead of PartialSync
- Under loss condition, PartialSync has similar delays as naive solution

References

1.Zhu, Zhenkai, and Alexander Afanasyev. "Let's chronosync: Decentralized dataset state synchronization in named data networking." *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013.

2. Content centric networking (CCNx) project website. http://www.ccnx.org.

3. Fu, Wenliang, Hila Ben Abraham, and Patrick Crowley. "iSync: a high performance and scalable data synchronization protocol for named data networking." *Proceedings of the 1st international conference on Information-centric networking*. ACM, 2014.

4. Wenliang Fu, Hila Ben Abraham, and Patrick Crowley, “Synchronizing Namespaces with Invertible Bloom Filters,” In Proceedings of the 11th ACM/IEEE Symposium on Architectures for networking and communications systems (ANCS), May 2015.