Presented at IEEE International Conference on Communications (ICC): Next-Generation Networking and Internet Symposium 2021

PLI-Sync: Prefetch Loss-Insensitive Sync for NDN Group Streaming

Yi Hu
Machine Learning and Data Analytics Research
Perspecta Labs
Basking Ridge, NJ, USA
yhu@perspectalabs.com

Constantin Serban
Machine Learning and Data Analytics Research
Perspecta Labs
Basking Ridge, NJ, USA
cserban@perspectalabs.com

Lan Wang
Computer Science Department
University of Memphis
Memphis, TN, USA
lanwang@memphis.edu

Alex Afanasyev
School of Computing and Information Sciences
Florida International University
Miami, FL, USA
aa@cs.fiu.edu

Lixia Zhang
Computer Science Department
UCLA
Los Angeles, CA, USA
lixia@cs.ucla.edu

***Abstract*—In this paper we explore solutions to robust group communication in disadvantaged wireless networks that exhibit low bandwidth, high packet losses, and frequent or even permanent network partitions. More specifically, we propose a group communication protocol based on Named Data Networking (NDN). By design, NDN's in-network caching and stateful forwarding plane can help improve data delivery robustness in disadvantaged networks, but when content is generated with dynamic rates, an efficient, low-overhead synchronization protocol is needed to inform group members to fetch newly generated content promptly. In this paper, we describe the Prefetch Loss-Insensitive Sync (PLI-Sync) protocol for group communication in highly disadvantaged networks. PLI-Sync combines optimistic content pre-fetching with new content notification via sync, and addresses the challenges of how to distinguish wireless packet losses from mobility-induced disconnections, and between data availability and retrievability. Our evaluations show that leveraging the interplay between optimistic pre-fetching and a low rate sync protocol can significantly reduce communication overhead compared to relying on sync protocols alone, while maintaining low data fetching latency and robust delivery in a variety of wireless conditions and traffic load settings.**

***Index Terms*—Named Data Networking, NDN, Sync, Group Communication, Wireless Networks, Lossy Networks, Disconnected Intermittent Links, DIL**

I. Introduction

Named Data Networking (NDN) [1]–[3] offers significant advantages in supporting distributed applications over the traditional IP-based networks. NDN makes named, secured data the centerpiece of communication, where consumers request desired data by name and the network satisfies these requests. All data items are cryptographically signed and encrypted as needed, hence providing inherent security, while the network provides efficient data distribution using a stateful data plane that offers request aggregation, in-network caching, path discovery, loop suppression, and other features.

NDN benefits become more pronounced in disadvantaged wireless networks that are often infrastructure-less, ad hoc, and deployed on-demand anytime and anywhere. Such networks are characterized by i) varying bandwidth availability due to a multitude of factors, including limited spectrum, dynamic node movements, and adverse operational conditions; ii) high packet loss with different characteristics; and iii) disconnected and intermittent links (DIL), often leading to frequent network partitions as well as more pathological cases of permanently partitioned networks.

NDN's advantage in disadvantaged wireless networks stems from its ability to retrieve data packets by names from anywhere, as well as to support in-network data caching. Therefore when end-to-end network paths are constantly disrupted, data packets can be retrieved from the caches on intermediary nodes. In the context of group communication, data retrieved by some group members increases the success of others in retrieving the same data. Even when a network is permanently partitioned, node mobility or network regrouping allows data to flow from producers to consumers without relying on end-to-end paths, significantly increasing data delivery robustness and maintaining application functionality.

While the above NDN benefits for data transfer in wireless networks is straightforward when dealing with stable, or otherwise known, data generation patterns, dynamically generated content poses specific challenges. Since data retrieval in NDN is a consumer-driven process, consumers need to know the specific names, or at least the prefixes, of available data. Thus prompt fetching of dynamically generated data requires notifying all the consumers about a) the production of new data, and b) the names of the new data. Once this information is conveyed, typical consumer application-driven data retrieval can be applied, bringing about the previously mentioned benefits. This notification task is provided by an NDN synchronization (or Sync in short) protocol [4], which keeps all the consumers informed of the latest data production. However, a Sync protocol also brings with it a certain amount of communication overhead.

This paper describes Prefetch Loss-Insensitive Sync (PLI-

Sync), specifically designed for dynamic data streaming among a group of users in a disadvantaged wireless network. PLI-Sync takes a novel approach of *combining an optimistic content prefetching with a state vector sync protocol*. By utilizing the sync protocol and the data prefetching in a complementary way, PLI-Sync can keep data fetching delay low while significantly lowering the sync protocol communication overhead, under the conditions of heavy packet losses, network partitions, and variable data generation rates.

The rest of this paper is organized as follows. §II presents the application model and its requirements. §III discusses the design of the PLI-Sync protocol, and §IV reports the evaluation scenario and the performance of the PLI-Sync. We conclude the paper with §VI.

II. Application Model

The application model we consider in this paper assumes that individual data items are produced dynamically in real-time according to a stochastic process. Figure 1 (upper part) shows the data production model, where data items are generated at highly variable time intervals. Note that our stochastic data production model subsumes simpler cases where the data productions follow either constant rate or otherwise deterministic patterns.

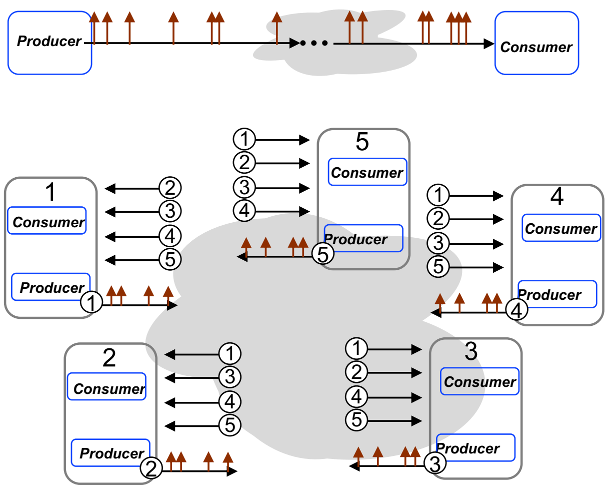

Fig. 1. Stochastic Stream Data Production and Consumption Application Model (top: a single producer and consumer pair; bottom: a group with multiple producer/consumer-combos)

We define an application group as a set of producers and consumers. A stream of data is defined as the data generated by a single producer. Figure 1 (bottom) shows the combined producer-consumer model in a group. The model assumes that each *node* may host applications that produce and/or consume one or multiple streams of data. While there could be cases where nodes are producer-only or consumer-only as shown in Figure 1 (top), the group producer-consumer combo case is our dominant use case.

Our application model allows dynamic group membership. Nodes may join or leave the group at different times, and the exact group membership may not be known a priori. Given the DIL conditions in the network, a specific producer-consumer combo group might only last for a short duration of time.

There are multiple applications and use cases that fit the above model. We list three of them below. One such class of apps are Situational Awareness apps used by emergency response teams, where data items are produced pseudo-periodically to report the individual view of each team member and to be consumed by everyone in the group. Another app is a group chat, where all the members of the group produce and consume chat messages from each other. And the third application use case is video and audio group simulcast, where team members broadcast their audio/video data to all the other team members. The PLI-Sync described in this paper is designed for supporting, and has been integrated with, all the above three application use cases.

III. PLI-Sync Design

Figure 2 shows the major components of the PLI-Sync Streaming Node communication stack. Each node hosts an application instance. The application may interact with the communication stack using multiple streams: one or more producer streams and one or more consumer streams per group member.

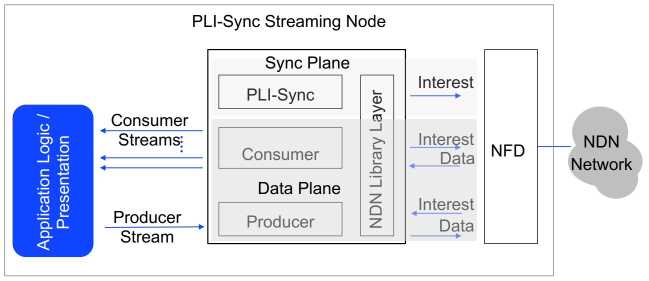

Fig. 2. Overview of the PLI-Sync Streaming Node

The communication layer consists of a data plane component and a sync plane component. The data plane consists of a Producer component which receives data interests and serves the requested data, and a Consumer component which issues interests for data in all the relevant streams and processes the received data. The sync plane is responsible for synchronizing the data production state of all members within the group, enabling the consumer to fetch newly generated data promptly.

A PLI-Sync Streaming Node also contains a local NDN Forwarding Daemon (NFD), representing the entry point into an NDN network composed of a mix of streaming nodes and (pure) NDN forwarders in an arbitrary proportion, depending on the specific deployment.

A. *Producer Streams*

The Producer component (Figure 3) receives a continuous stream of application messages and transforms them into a stream of NDN data packets, following a naming scheme of the form:

“`/<prefix>/<n>/<NodeId>/stream/<streamNum>/<seqNum>`”

where i) node number "`<NodeId>`" identifies a specific source node issuing the data; ii) the stream number "`<streamNum>`" represents a *specific* stream of data generated by that node; for example, a video file and the location data of the same node are represented by two different streamNum's; and iii) the seqNum is a monotonically increasing number, one for each newly produced data packet. "`<prefix>`" represents an application and "`<n>`" identifies a group. The producer may save produced data packets in a local repo, which can then be served upon receiving matching interests.

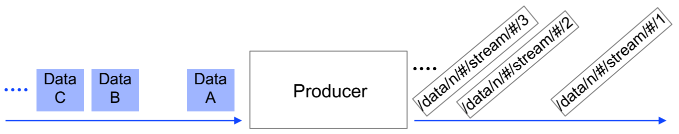

Fig. 3. PLI-Sync producer names incoming application messages.

B. *Consumer Prefetching*

The consumer component issues interests for data items in an increasing order of the sequence number. One key aspect of our solution is that, upon receiving a piece of data, a consumer optimistically issues an interest for the next sequence number into the stream, *before* receiving notification of that piece of data's production. This data interest will be pending in the network, until the producer generates the next data item, or until the Interest's lifetime expires.

An important parameter of the prefetching mechanism is the value of the interest lifetime for data prefetching interests. Usually an interest's lifetime reflects expected network delay as well as the needs of the consumer; for prefetching interests, the lifetime value is also needs to reflect the producer's data production rate. To address this need, each consumer runs an estimation module that predicts the production rate using a weighted average of the interest satisfaction time over a given time window, where the weight and the time window length can be configured. In addition, whenever a prefetch interest expires, a consumer also consult the sync plane, which we will explain next, as the consumer may have received sync interests after the prefetch interest was sent and before it expired to learn whether new data has been produced. Such information helps the consumer to adjust the lifetime value for subsequent prefetching interests. The adjustment of the interest lifetime can also be performed via a number of other approaches; a more in-depth discussion of this topic is beyond the scope of this paper.

This prefetching design enables prompt data retrieval of sequentially generated data packets at the estimated pace of the producer's stochastic process. However, this scheme suffers from the following two drawbacks if applied alone.

a) Producer stream discovery: prefetching data from existing streams alone does not provide a discovery mechanism to signal when a new node joins the group and starts streaming data, or when an existing node initiates a new data stream. This signaling function is important to support dynamic group membership, where nodes can join or leave (temporarily or permanently) at any point, making it difficult for consumers to determine when to start and when to stop pulling data for each stream.

b) Operations under lossy conditions: When the network exhibits significant losses, a consumer has difficulty in telling whether a data prefetching interest's expiration is due to the requested data not being produced yet, or due to the interest or the returned data getting lost in the network. If the former is assumed while the latter is true, the consumer risks getting stuck without retransmitting the interest. However, if the latter is assumed while the former is true, issuing spurious interests for data not yet produced may significantly increase the network overhead. Notwithstanding this overhead, prefetching may also lead to stream throughput collapse even under modest network losses: a consumer may spend a significant amount of time waiting on interest timeout for lost packets, unaware of the state of the producer at the given moment in time. In our initial trials we observed significant throughput collapse even for relatively small ($\approx$10%) end-to-end packet loss rate. The collapse in stream throughput is further amplified by confounding DIL conditions and network partitioning events.

C. *Sync Plane*

To address the two issues identified above, PLI-Sync deploys a state vector sync protocol. The state vector is propagated using NDN interests: each producer generates periodic sync interests, regardless of whether new data is generated. The period is significantly lower than the average data generation rate which is determined either empirically or in a data-driven manner. The format of the sync interests for a group of an application is as follows: "`/<prefix>/<n>/<NodeId>/sync/<seqNum>/<vector_state>`", where "`<vector_state>`" = "`(streamNum, nodeId, SeqNum), ..., (streamNum, nodeId, SeqNum)`". This name encodes the issuing node id, the sync prefix, and a vector which enumerates all data streams of all the node in the group with their latest sequence number. The node id is used to facilitate the authentication of the interest packets: all PLI-Sync interests are signed using a private key corresponding to the application-defined security trust schema. This is necessary since such interests carry important information whose abuse could adversely impact the protocol.

The sync interests fulfill the following two roles.

a) Stream discovery: Sync interests inform all the consumers of the existing streams, including any newly created ones by either current or newly joined nodes, and the current data state of each stream, i.e., the last sequence number.

b) Prefetch advise: The issuer of a Sync interest informs the rest of the group its current view regarding the state of all data productions in the group. The sync interest helps the data plane's prefetch mechanism distinguish packet losses from data availability. Whenever a sync interest reflects a newer sequence number for a give stream, the consumer can confidently fetch the new data, based on the stream latency and reliability requirements.

The period length of sync interests represents a tradeoff between network overhead and maximum tolerable data fetching latency. The choice of the sync period is also dependent on the number of nodes in the network (or partition): if each node generates sync interest at a given rate, a larger group means a higher count on the total sync interests generated compared to a smaller group. In a network of 50 nodes, we configured the sync interest rate to be 6 to 10 times lower than the nodes' average data generation rates.

D. Data and Sync Plane Interplay

Figure 4 shows the interplay between the data plane and the sync plane in PLI-Sync. PLI-Sync derives latest data availability information from the following sources: i) sync interests received over the network from other PLI-Sync instances; ii) the local producer, reflecting the local stream states; and iii) the local consumer, reflecting the latest data fetched (optimistically or not) for all the streams in its group. This last aspect distinguishes PLI-Sync from previous NDN sync protocols in which consumers only fetch data that is known to have been published. Therefore, the previous sync protocols *supply* new data information to the local consumer but do not *obtain* such information from the consumer. In contrast, PLI-Sync makes the local consumer an important source of latest data state discovery. As we show in §IV, the consumer-contributed sync state can significantly reduce the overhead of sync protocol while maintaining low data fetching latency.

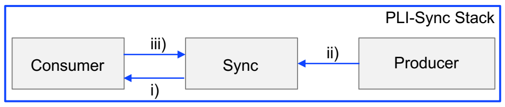

Fig. 4. Data Plane and Sync Plane Interactions

Another important semantic distinction between PLI-Sync and most other sync protocols is that each node N uses sync interests to report *the latest data packet of each stream that N has obtained*, instead of reporting the latest sequence number of all the new data. Therefore whenever a consumer finds that a stream has produced new data, it proceeds to fetch the corresponding data aggressively, knowing with high confidence that the data should be available nearby, e.g., from the sync-issuing node. This design departure from most other NDN Sync protocols reflects a difference in design priorities. The earlier Sync designs take into consideration the cases where not every consumer would want all the data, especially when it is under resource constraints (e.g. under poor connectivity, or running low in local memory or battery power). However, the first priority in the PLI-Sync design is resilient data delivery when operating with intermittent connectivity and frequent network partitions. An extreme case is shown in Figure 5: as time advances from t_0 to t_1, the network is permanently partitioned, thus no path exists between node A and nodes B and C. This situation is common under severe DIL or high mobility conditions, where propagating the dataset state alone would

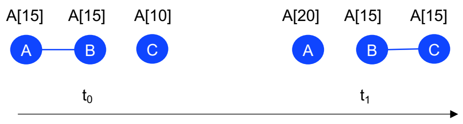

Fig. 5. Operations in Partitioned Networks

not help. At time t_0, Node A and B can communicate, their data states get synchronized (in this case Node B synchronizes with Node A stream state sequence 15), and Node B retrieves the data items up to "`/prefix/n/A/stream/15`" via fetch in data plane, and a copy of all retrieved data is cached in the content store of Node B. Later on at time t_1, when Node B is in contact with Node C, Node C synchronizes its state with that of Node B, and hence it becomes aware of the Node A stream state sequence 15. As a result, Node C is able to fetch the data item up to "`/prefix/n/A/stream/15`" from the network (i.e., from the NFD of Node B). This functionality is enabled by the semantic of the PLI-Sync state vector backed by direct and eager fetching in the data plane, as well as by the PLI-Sync node design (Figure 2) where each participating node has an NFD instance that caches consumed data locally.

Although this example might seem a rare pathological situation (or at best contrived) it is typical of DIL networks where links are temporary permitting only a few packet exchanges before going away. Effective communication under such conditions becomes essential for network functionality.

IV. Performance Evaluation

We implemented the PLI-Sync software stack using multiple versions of NFD and ndn-cxx library (0.6.2 to 0.6.6). We ported and used the stack on different platforms (Ubuntu 16.04, Ubuntu 18.04, Android 7.2, Android 8.0) in deployments with up to 100 devices, using a variety of wireless networks. The evaluation results described in this section had been performed in a hybrid environment, consisting of virtual machines hosting the PLI-Sync software stack and application, and an ns-3 simulation-in-the-loop network model. The results presented below are based on a 50-node network shown in Figure 6. This network consists of five groups of ten mobile devices (n1-n10, ..., n41-n50), connected via WiFi 802.11n to an Access Point (AP1-5), and each AP is connected via LTE to a central LTE Router. The APs are configured to operate at a HtMCS6 rate of 58.5Mbps shared between ten associated nodes, whereas the LTE network is configured to operate at a rate of 100Gbps shared among all five LTE links.

Each phone was represented by a Ubuntu18.04 host running the PLI-Sync Streaming Node software stack in Figure 2, including an NFD instance. Each of the AP nodes and the LTE Router node also ran an NFD instance, but without running the app or PLI-Sync software stack. The application, emulating a generic Situational Awareness app, generated a stream of data items with a bimodal distribution alternating between light episodes (data size 696B, inter-arrival-time uniform 9~11s, 1~50 data items) and heavy communication episodes (data size 1783B inter-arrival-time uniform 1~5s, 1~10 data items). Data generation was independent at each node.

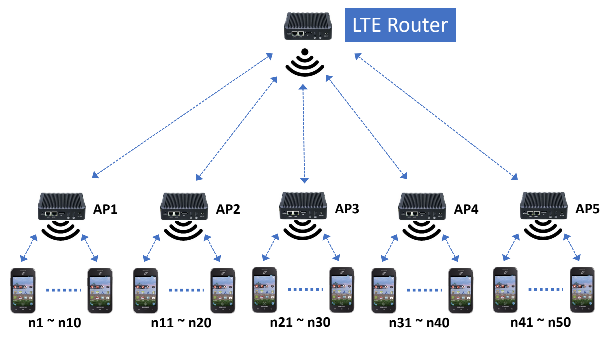

Fig. 6. 50-Node Evaluation Network

The NDN network was configured with opportunistic caching, using UDP unicast faces between each mobile device and its AP, and between each AP and the LTE Router. We did not use broadcast NDN faces, due to poor (or no) support and performance for either WiFi and LTE. In all experiments we used a multicast forwarding strategy for all prefixes.

We used the following main metrics for our evaluation:

- *Delivery latency*: defined as the difference between the data generation time and the data reception time.
- *Overall Interest to Data Ratio (OIDR)*: the ratio between the total number of interests sent (data and sync planes) and the number of data items received. It measures the interest overhead in the sync plane and data plane.
- *Data Interest to Data Ratio (DIDR)*: the ratio between the number of data interests sent and the number of data items received. It measures the performance in the data plane.
- *Sync Interest to Data Ratio (SIDR)*: the ratio between the number of sync interests sent and the number of data items received; it measures sync interest overhead.
- *Sync ACK to Data Ratio (SADR)*: the ratio between the number of sync ACK data sent and the number of data items received; it measures sync ACK data overhead.[1]
- *Efficiency (EFF)*: the total bytes of received data at all end nodes, divided by the total sent bytes at all nodes (the latter consisting of total bytes of sync interests sent, total bytes of sync ACK sent, total bytes of data interest sent, total bytes of data sent).
- *Catch up latency*: defined as the data fetching delay *after* additional data becomes retrievable following mobile node movements.

A. Heavy Loss Conditions

In the first scenario we considered a network with heavy packet losses. Each packet generated by a mobile device is subject to a 20% loss on the uplink path to the AP, and an additional 20% packet loss rate on the downlink from the AP to each receiver node. The loss is applied using ns3 rate error model on a per-packet basis. There is no additional LTE link loss. In this scenario, we analyze the performance of PLI-Sync and compare it with the performance of PSync [5] (in Full Sync Mode), and PLI-Sync *without* feeding information from the consumer into the sync plane; we dub this setup *PLI-Sync-noconsumer*. PLI-Sync-noconsumer is also configured for a node to send additional sync interests when a received sync interest does not have the latest data production state about itself. Consequently, the data retrieval latency for PLI-Sync, PLI-Sync-noconsumer, and PSync are similar in this scenario as shown in Figure 7. The slightly larger maximum data retrieval latency for PLI-Sync-noconsumer is due to the reliance on the sync interests to learn the latest data production state. PSync displays a step-like latency pattern because it was configured with a sync interest lifetime of one second.[2]

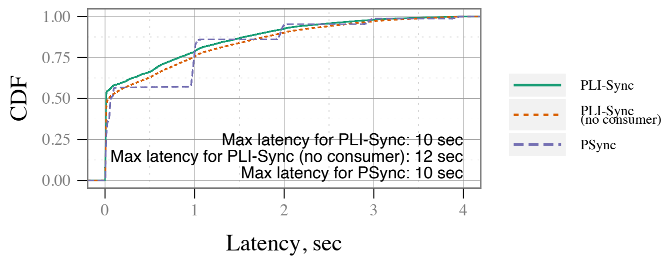

Fig. 7. Data Retrieval Latency

TABLE I
METRICS UNDER HEAVY LOSS

-	OIDR	DIDR	SIDR	SADR	EFF
PLI-Sync	1.88	1.87	0.06	0	13.4
PLI-Sync (no cons.)	2.66	2.08	0.58	0	3.0
PSync	4.88	1.68	0.18	3.02	0.45

The overhead and efficiency measures for single long-running experiments (12 hours each) are summarized in Table I. As expected, PLI-Sync has the smallest overall overhead ratio. The difference in overhead between PLI-Sync and PLI-Sync-noconsumer can be attributed to the feedback between consumer and the sync plane: knowledge of new data from a node's data plane helps speed up other nodes' data discovery, thus, saves the overhead of sending extra sync interests. The Data Interest to Data Ratio is the smallest for PSync because in this case data is retrieved only after nodes are informed of data's existence, in contrast to PLI-Sync's optimistic retrieval which results in extra-overhead due to the retrieval attempts prior to data production. The Sync Interest to Data Ratio is the smallest for PLI-Sync, since it is at constant rate and low frequency (60s). This overhead is larger for PLI-Sync-noconsumer. The Sync ACK to Data Ratio is only applicable to PSync, since PLI-Sync does not respond to sync interests with sync ACK. Finally, it can be observed that the efficiency is much higher for PLI-Sync as compared to PLI-Sync-no-consumer and PSync, thanks to both its optimistic data fetching and the data plane discovery contributions to sync.

[1]PLI-Sync does not generate sync ACK data; PSync used in the comparison does.

[2]This value was hand-picked for the experiment to balance overhead and data fetching delay. Larger values would prolong the fetching delay, while smaller values increase sync overhead.

B. Network Partitioning

In the second scenario, in addition to the 20% loss in WiFi reception, we created a permanently partitioned network in the following way. After the network is fully connected for the first 100s, nodes under AP4 and AP5 get disconnected from LTE, creating their own network partition. At time 400s, nodes under AP3 get disconnected as well, hence creating a network partitioned three-way. At time 700s, AP3 reconnects with the [AP4, AP5] partition, enabling the nodes under AP3 to ferry data generated by nodes under [AP1, AP2] during the time period [100s, 400s] to [AP4, AP5]. Then nodes n21-n30 along with AP3 continue to repeat the swing pattern between a two-partitioned and a three-partitioned network.

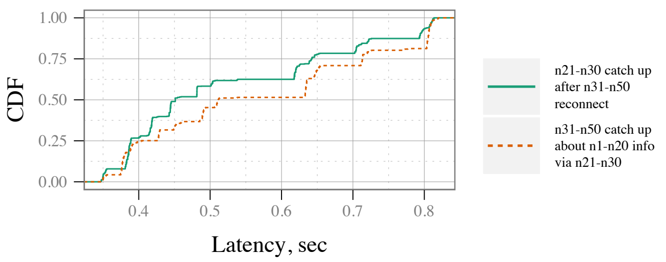

Fig. 8. Disconnected Mode Retrieval Latency

Figure 8 shows the CDF for catch up latency for PLI-Sync operating in the disconnected mode, where the periodic sync interest rate of a producer remains one per 60s. The solid green line highlights the discovery latencies at nodes n21-n30 of data produced by nodes n31-n50 following a reconnection. The dashed red line measures the latency of discovery at nodes n31-n50 of data produced by nodes n1-n20 following a reconnection with nodes n21-n30. It can be observed that the latency is under one second in both cases.

The overhead and efficiency measures are qualitatively similar to those obtained in connected mode (OIDR=2.19, DIDR=2.18, SIDR=0.06, SADR=0, and EFF=14.6), demonstrating that the protocol has stable performance and low overhead across a wide range of network conditions, achieving its design goal.

V. Related Work

A number of NDN sync protocols have been developed [4]–[10]. ChronoSync [7] adopts a cryptographic digest data structure to encode the dataset state. This representation is not semantically meaningful, thus it cannot directly show the difference in the dataset state when multiple nodes make changes to the dataset states. This reduces the protocol efficiency, when compared with PLI-Sync, especially in networks with disconnected and intermittent links (DIL). iSync [6] and PSync [5] use an invertible Bloom Filter (IBF) [11] to encode dataset state, relying on the IBF subtraction operation to infer the state difference. IBF by design has certain limitations on the amount of information it can decode with a single operation due to the probability of false positive errors [11]. As such, multiple rounds of exchanges may be required to resolve differences, which can be expensive or infeasible under DIL conditions. VectorSync [8], DDSN [12], and SVS [13] overcome this problem by directly reporting the latest data production of all the group members to improve resiliency, a feature that we borrow in PLI-Sync. PLI-Sync makes a unique contribution by combining data plane and sync plane in data information discovery with significantly increased efficiency.

VI. Conclusion

The results shown in this paper demonstrate that PLI-Sync can provide robust data delivery in DIL environments, with a low data fetching delay and a low protocol overhead. Among the topics not covered in this paper are the impact of routing and forwarding on the performance of PLI-Sync, the effects of network congestion on data retrievability, as well as aspects related to the predictability of stream data production. These topics are left for future research.

Acknowledgement

This work was supported in part by DARPA under contract HR0011-17-C-0113. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement ”A” (Approved for Public Release, Distribution Unlimited).

References

[1] V. Jacobson, D. K. Smetters, J. Thronton, M. F. Plass, N. H. Briggs, and R. Braynard, “Network Named Content,” *CoNEXT*, 2009.

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, “Named Data Networking,” *ACM Computer Communication Reviews*, June 2014.

[3] A. Afanasyev, T. Refaei, L. Wang, and L. Zhang, “A brief introduction to Named Data Networking,” in *Proc. of MILCOM*, Oct. 2018.

[4] P. Moll, W. Shang, Y. Yu, A. Afanasyev, and L. Zhang, “A Survey of Distributed Dataset Synchronization in Named Data Networking,” Tech. Rep. NDN-0053, Revision 2, Named Data Networking, March 2021.

[5] M. Zhang, V. Lehman, and L. Wang, “Scalable Name-based Data Synchronization for Named Data Networking,” in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[6] W. Fu, H. Ben Abraham, and P. Crowley, “isync: a high performance and scalable data synchronization protocol for named data networking,” in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pp. 181–182, 2014.

[7] Z. Zhu and A. Afanasyev, “Let’s ChronoSync: Decentralized dataset state synchronization in Named Data Networking,” in *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP 2013)*, (Goettingen, Germany), Oct. 2013.

[8] W. Shang, A. Afanasyev, and L. Zhang, “VectorSync: Distributed dataset synchronization over Named Data Networking,” Technical Report NDN-0056, NDN, Mar. 2018.

[9] P. Heras-Quirós, E. Castro, W. Shang, Y. Yu, S. Mastorakis, A. Afanasyev, and L. Zhang, “The design of roundsync protocol,” in *Technical Report NDN-0048*, 04 2017.

[10] X. Xu, H. Zhang, T. Li, and L. Zhang, “Achieving resilient data availability in wireless sensor networks,” in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2018.

[11] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, “What’s the difference?: efficient set reconciliation without prior context,” in *SIGCOMM*, 2011.

[12] T. Li, Z. Kong, S. Mastorakis, and L. Zhang, “Distributed Dataset Synchronization in Disruptive Networks,” in *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, 2019.

[13] “Specification and API Description of the StateVectorSync (SVS) Protocol.” by NDN Project Team, available online at https://github.com/named-data/StateVectorSync.