SoK: The Evolution of Distributed Dataset Synchronization Solutions in NDN

Philipp Moll, Varun Patil
{phmoll,varunpatil}@cs.ucla.edu
UCLA
Los Angeles, USA

Lan Wang
lanwang@memphis.edu
University of Memphis
Memphis, USA

Lixia Zhang
lixia@cs.ucla.edu
UCLA
Los Angeles, USA

ABSTRACT

Distributed dataset synchronization, or Sync in short, plays the role of a transport service in the Named Data Networking (NDN) architecture. A number of NDN Sync protocols have been developed over the last decade. In this paper, we conduct a systematic examination of NDN Sync protocol designs, identify common design patterns, reveal insights behind different design approaches, and collect lessons learned over the years. We show that (i) each Sync protocol can be characterized by its design decisions on three basic components – dataset namespace representation, namespace encoding for sharing, and change notification mechanism, and (ii) two or three types of choices have been observed for each design component. Through analysis and experimental evaluation, we reveal how different design choices influence the latency, reliability, overhead, and security of dataset synchronization. We also discuss the relationship between transport and application naming, the implications of namespace encoding for Sync group scalability, and the fundamental reason behind the need for *Sync* Interest multicast.

CCS CONCEPTS

• **Networks** → **Network protocol design**; **Transport protocols**; *Network design principles.*

KEYWORDS

Named Data Networking, Distributed Dataset Synchronization, Sync Protocols, NDN Transport

ACM Reference Format:
Philipp Moll, Varun Patil, Lan Wang, and Lixia Zhang. 2022. SoK: The Evolution of Distributed Dataset Synchronization Solutions in NDN. In *9th ACM Conference on Information-Centric Networking (ICN '22), September 19–21, 2022, Osaka, Japan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3517212.3558092

1 INTRODUCTION

Most people in the ICN community are familiar with Named Data Networking (NDN)'s basic *network* communication model of Interest-Data exchanges, which has been documented in numerous publications [4, 56]. However, a comprehensive overview of NDN *transport* service is missing. This paper aims to fill that void.

The transport services in the NDN architecture differ from that in the TCP/IP architecture in fundamental ways due to two reasons. First, as we explain in §2, NDN moves the three basic functions provided by the transport layer in today's IP network, namely demultiplexing, reliable data delivery, and congestion control, out of transport and into proper places in the NDN protocol stack. Second, different from today's common practice where distributed applications rendezvous at centralized servers through point-to-point transport service, NDN enables entities running distributed applications to communicate *directly* by enabling them to fetch desired data using applications layer names at the network layer. That is, NDN needs a new type of *multiparty* transport service that can enable a group of distributed entities to communicate in a secure, resilient, and reliable way.

The new NDN transport service that emerged from the last ten plus years of NDN experimental research is the namespace synchronization of shared datasets among a group of distributed entities. A variety of NDN Sync protocols, as summarized in a survey [31], have been developed, and the design approaches have evolved over the years. This SoK paper aims to provide a systematic examination of all the existing NDN Sync protocol designs, collect the lessons learned from different design choices, and identify remaining issues and future research directions. We start with a clarification on the role of Sync in the NDN protocol stack (§2), then proceed with identifying the major components in a Sync protocol (§3). We illustrate the impact of different design choices through case studies of five representative Sync protocol designs (§4), then step up a level to compare and contrast the different design decisions made by different protocols, and validate our analysis with comparative evaluation results (§5).

From the above exercise we learn that every Sync protocol faces three basic design decisions: how to represent the dataset namespace containing data produced by all participants, how to encode the dataset namespace in communication, and how to disseminate dataset state changes effectively and efficiently[1]. We discover that a few design patterns are shared among all the Sync protocols (§4), and reveal the necessity of multicasting Sync Interests and the constraint in its use. Using both analysis and experimental evaluation, we show how different design choices influence the latency, reliability, overhead, and security in dataset synchronization. We wrap up the paper with discussions on the relationship between transport and application naming, the implications of namespace encoding for Sync protocol scalability (§6), and a few related research areas (§7), followed by the conclusion and future work (§8).

ICN '22, September 19–21, 2022, Osaka, Japan

ACM ISBN 978-1-4503-9257-0/22/09.
https://doi.org/10.1145/3517212.3558092

[1]Throughout this paper we treat *dataset namespace*, *dataset namespace state*, and *dataset state* as exchangeable terms.

2 THE ROLE OF SYNC IN NDN

Generally speaking, transport services bridge the gap between the services that the network layer provides and the services that applications desire. Today's transport protocols, as exemplified by TCP, convert IP's point-to-point datagram service to reliable byte stream delivery between two application processes, which are identified by a pair of IP addresses and transport port numbers. TCP provides three basic functions:

(1) demultiplexing incoming IP packets to different application processes;
(2) performing network congestion control[2]; and
(3) providing reliable byte stream delivery.

However, to support distributed applications, using TCP's point-to-point connections for multiparty communication requires setting up $n \times n$ TCP connections which is complex and inefficient. Although a few UDP-based reliable multicast protocols, such as NORM [3], have been developed, we note that IP multicast pushes all packets sent to a multicast address to all group members simultaneously, which can be problematic in scenarios where different members may have different resource constraints, consequently some may prefer to get specific subsets of the data first. A more commonly adopted approach is to build a middleware overlay to connect all participants in a distributed applications [27, 43], avoiding the cost of $n \times n$ TCP connections by paying the complexity of setting up and maintaining the overlay, together with increased vulnerability due to the overlay failure.

In an NDN network, the network layer fetches named data chunks. These data chunks carry semantic names that uniquely identify their content, together with cryptographic signatures that bind the names to the content. From NDN's perspective,

(1) NDN uses names for demultiplexing across all protocol layers, thus it does not need additional information from transport header for demultiplexing;
(2) NDN moves network congestion control to the network layer where it belongs [47, 49, 55]; and
(3) as a data-fetching protocol, NDN enables individual data consumers to pick and choose which data piece to fetch and when.

Although NDN's data-centric model facilitates multiparty communication by letting individual participants request desired data, the participants still need to learn the names of all newly produced data by others as soon as possible, so that they can retrieve it promptly if needed by applications.

We identified *distributed dataset namespace synchronization*, or *Sync*, as a basic service to provide reliable synchronization of the names of the shared dataset in an application group, dubbed *Sync group*. Sync has been used as transport service for a number of NDN applications such as a chatroom app [6], a network routing protocol [53], a network management framework [37], and a pub/sub implementation [30].

As a transport layer service running on top of NDN's Interest-Data exchange, Sync offers unique advantages in supporting heterogeneous receivers and diverse data reliability requirements. An important lesson learned from TCP is that all applications desire communication reliability to certain degree, but their definitions of reliability can vary. Based on this lesson, the role of NDN Sync is to synchronize the dataset *namespace* (the collection of data item names in the dataset) among all members in a Sync group. Having learned the published data names, each member can then decide on its own whether and when to retrieve all or some of the data items based on the application's need as well as local resource constraints.

[2]Congestion control was not part of the TCP's original functions [45], but was added later to mitigate the Internet congestion meltdown through utilizing TCP's end-to-end feedback loop [21].

3 THE DESIGN OF NDN SYNC PROTOCOL

This section first presents our view on the Sync protocol design goals and non-goals, and then identifies the major components that make up a Sync protocol.

3.1 Sync Protocol Design Goals

One common need for distributed applications is group membership management, which we believe is best handled by applications, as only applications have the necessary knowledge to determine whether a specific user should/not be accepted into a group. Assuming all group members possess proper identities and certificates, Sync provides the means for them to participate in group communication effectively and efficiently. The design goals below reflect general observations on applications needs.

Reliable dataset *namespace* Sync: This requirement denotes the protocol's ability to deliver all updates of dataset namespace to all the Sync participants. In the absence of permanent network partition, all participants should eventually learn all the data names in the shared dataset.

Low synchronization latency: To make distributed applications perform well, Sync must inform all participants promptly of the shared dataset state changes to meet the low-latency requirement of applications such as online games or conference calls.

Resilient performance: Network environments can range from stable infrastructure networks with a low loss rate at one end of spectrum to ad-hoc wireless networks with intermittent connectivity at the other end. To provide a general transport service, a Sync protocol should work well across the whole spectrum.

3.2 Components in Sync Protocol Design

The Sync protocol development efforts identified three basic design components early on. First, one needs to define a representation of the shared dataset's namespace (*dataset namespace representation*). Second, one needs an efficient way to encode the dataset namespace in a defined data structure to transmit over the network (namespace encoding). Third, one needs to promptly propagate the changes of the shared dataset namespace (*state change notification*). We illustrate each design component below.

Namespace Representation: The design of the namespace representation starts with considering the *namespace* for application-produced data items. We observe two approaches to the dataset namespace representation. The first one uses the application data names directly. Thus the Sync namespace representation is simply a collection of the names of all the data items that have been produced in the shared dataset. The second approach assumes that data items generated by each producer P is named sequentially, which can be represented by a pair of [producer name, seq#], thus the namespace of the entire shared dataset consists of a list of

[producer name, seq#]-pairs, one for each participant. We discuss the implications of each of the two approaches in §6.1.

Dataset State Encoding: The goal of *state encoding* is to convert the shared dataset namespace representation into a compact form for transmission over potentially lossy networks. As we discuss in §6.1, the efficiency of state encoding solutions is directly tied to the dataset namespace representation, and different solutions lead to different design tradeoffs.

Dataset State Change Notification: Since participants fetch Data by names, it is Sync's responsibility to inform all the participants in a Sync group of data production updates as soon as possible. For Sync to work, however, participants in a group must be able to learn the namespace updates without having to name individual participants. Therefore, information about new dataset state needs to be carried using *multicast* Sync Interest packets to the Sync group.

Up to now, we have observed two uses of Sync Interest multicast. The first one lets every participant *pull* new data productions from the group by multicasting a Sync Interest to the group and receiving new changes in reply Data packets (Sync Replies). The second approach lets data producers *notify* all the others in the group by directly including state update information in each Sync Interest, which one does not need to send a data reply.

The need for using Sync Interest multicast deserves further elaboration. NDN is designed with built-in multicast delivery of *Data* packets by forwarding Interest packets, guided by router FIBs, towards the desired Data direction and merging Interests carrying the same request. *Interest* multicast differs from *Data* multicast. While multicast data delivery is inherently supported by NDN's network layer, Sync Interest multicast requires multicast routing support to forward Interests to all the participant in a Sync group. We discuss the necessity of Interest multicast in §6.3.

4 SYNC PROTOCOL DESIGN: CASE STUDIES

More than ten different NDN Sync protocol designs have been developed over the years [31], with later ones improving upon the lessons learned from previous designs. In this section, we examine various design choices made during the evolution of Sync, as observed in five representative Sync protocols (ChronoSync, iSync, PSync, syncps, and SVS), and validate our analysis with evaluation results. We make the comparison in the context of a general use case, looking at metrics such as the synchronization latency and network traffic overhead.

Below we first describe our evaluation setting for all the experiments. Although some of the experiment parameters will be introduced later in the paper, the following summary of evaluation parameter settings aims to offer the reader a convenient place to find all the relevant information in interpreting the evaluation results. We choose Mini-NDN [28] as our evaluation environment and emulate the topology of the GÉANT research network [18] with 45 nodes; each link has unlimited bandwidth and a 10msec propagation delay. In each experiment run, we randomly select 20 nodes to be participants in a Sync group. Our evaluation uses this relatively small group size to illustrate the differences between different Sync protocol design choices; we discuss the impact of Sync group sizes in §6.2. To observe each Sync protocol's performance under different conditions, we vary the loss rate of each link from zero to 20%, and the publishing rate of each Sync group members from 1 data item every 15 seconds to 2 data items per second (in the evaluation plots, the X-axis indicates the data publication rate of the *entire* group).

We experimented with setting the Sync Interest lifetime of ChronoSync, PSync, and syncps, the Sync protocols that deploy *long-lived Sync Interests*, to 1sec, 4sec and 10sec; the results reported in this paper use 1sec as Sync Interest Lifetime. For other configurable parameters such as various timer values, data freshness periods, IBF size etc., we use the suggested default configuration values in individual protocols' implementations. The evaluation results include i) **Sync latency**: the time period between a data item's generation time and the time its notification reaches another member; percentiles for individual values are calculated and reported; ii) **Sync protocol overhead**: for each published data item, the summation of the number of Sync Interest and reply Data packets received at every NDN forwarder including all end nodes; and iii) **reliability**: the percentage of participants which received new publication notifications. Error bars in the figures denote the 95% confidence interval from 10 runs of every emulation setting.

4.1 Encoding Dataset Namespace by Digest

The first Sync protocol, CCNx Sync [46], assumes that the dataset namespace forms a hierarchical name-tree. It computes a digest at each node on the tree from the bottom layer up, and uses the root digest to represent the entire dataset state. *ChronoSync* [58], the second Sync protocol design and the first one being used by actual applications, takes the same approach of using a digest to to represent the dataset state. ChronoSync is the first one to use sequential naming convention described in §3.2, thus its dataset namespace is a list of [producer name, seq#] instead of a name-tree. ChronoSync encodes the state by computing a cryptographic hash over the list to create the *state digest*. Each participant P then multicasts a Sync Interest I which carries its state digest, which informs others in the same Sync group of P's dataset state, and serves as a request for new data produced by other group members. With a goal of fetching next data item as soon as possible while its production time is unpredictable, each group member keeps a pending Sync Interest at each of all other members. We call such Sync Interests *long-lived Interest*, which keep *persistent PIT entry* at every router between every pair of the group members, waiting for next new data item. Under stable conditions where all participants have a synchronized dataset state, they send identical Sync interests which are aggregated at routers. The example in Fig.1 shows that the Sync interests from participants P3 and P5 are aggregated at router R2, and only one is forwarded to P1. In the absence of replies, each P sends Sync Interests periodically (ie., Sync period), with a random delay jitter between 100-500msec,

In Fig. 1a, participant $P1$ multicasts a Sync Interest I_{P1} carrying its state digest. Each receiver of I_{P1} compares the state digest in I_{P1} with its local value. If the two are identical, the receiver and $P1$ have the same dataset state, and the receiver will keep I_{P1} pending locally. When $P3$ produces a new data item, it puts the data in an NDN Data packet and sends it as a reply to the pending I_{P1}. $P3$ also recomputes its state digest, and immediately sends a new

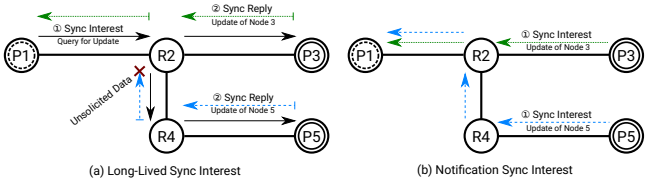

Figure 1: Simultaneous publication issues.

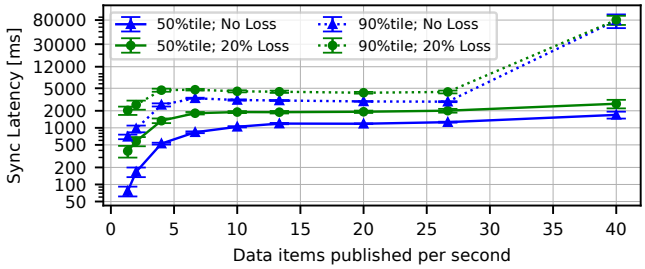

Figure 2: Sync latency of ChronoSync.

Sync interest I_{P3} containing its new state digest. In the absence of packet losses, $P3$'s reply reaches all others in the Sync group. If $P5$ publishes new data *after* both P3's reply to I_{P1} *and* its new Sync interest have been received by everyone in the group, $P5$'s new data will also be successfully received by the whole group.

When an incoming state digest differs from the local value, the receiver is informed of being out of sync with someone, but the digest does not tell the exact namespace differences. ChronoSync lets each participant P maintain a log of recent digests, when a received state digest D_{rec} differs, P checks D_{rec} against it digest log. If P finds D_{rec} in the log, an indication that D_{rec}'s sender lags behind, P will send a reply with its current dataset state; if not, P waits for a random time period for potential incoming Sync replies with a newly produced data item and resolve its puzzle. If no reply is received in time, P multicasts a recovery Interest carrying D_{rec}, hoping the sender of D_{rec}, or whoever has D_{rec}, can reply with its full dataset state.

One cause for unrecognized state digests is packet losses, e.g. $P1$ fails to receive $P3$'s new data item but receives $P3$'s new Sync Interest, which informs $P1$ of a state change but cannot tell $P1$ what update it misses. Another cause is simultaneous data publishing. NDN's flow balance principle states that one Interest retrieves one Data packet. Fig. 1a shows that, if P3 and P5 in the same Sync group produce new data and respond to the same pending Sync Interest simultaneously, R2 forwards one reply and drops the other. Again $P1$ will be unaware of the missing publication until it receives a Sync Interest with an unrecognized state digest. In all cases, it will take time for all participants to reach synchronized dataset state again.

Fig. 2 shows ChronoSync's performance measured by the group synchronization delay. In the absence of packet losses, ChronoSync performs well at low publication rate; as publishing rate increases, simultaneous publications become more likely, which lead to unrecognizable state digests, hence additional latency in resolution. Compounding simultaneous publications with packet losses further deteriorates performance.

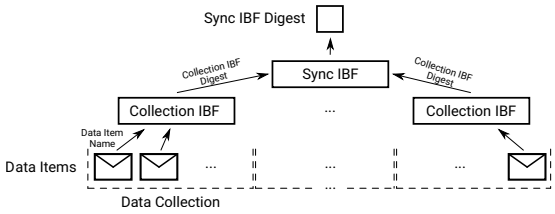

Figure 3: iSync's Multi-Level IBF Structure

4.2 Encoding Dataset Namespace by IBF

The lessons learned from ChronoSync suggest that, to quickly reconcile dataset namespace change, each Sync Interest should carry information to help infer the exact differences. A few followup protocols explored the use of Invertible Bloom Filter (IBF) [13] for this purpose. IBF is a probabilistic and space-efficient encoding for datasets that allows membership queries and set difference calculation. Encoding a dataset namespace in IBF and carrying it in a Sync Interest I enables each recipient R of I to calculate the state difference between itself and I's sender. More specifically, when R receives an IBF (f_1) that differs from its local IBF (f_2), R can extract the elements corresponding to f_2 -f_1, i.e., elements encoded in f_2 but not in f_1. Note that a data name needs to first be hashed into a number and inserted into the IBF. Since this hashing is one-way, R can infer the names it has that I's sender does not, but cannot infer what name(s) are missing in R's local dataset namespace. Below we examine three IBF-based Sync protocols: iSync, syncps, and PSync.

4.2.1 Supporting Application Names by Hierarchical IBF. The iSync protocol [15] was first to use IBF. Its dataset namespace is a collection of *all* application data names. To accommodate large datasets, iSync uses a 2-level structure to encode the dataset namespace as shown in Fig. 3. iSync first divides the whole dataset's publications into multiple *collections*, with each collection encoding its publication names in a *Collection IBF*. The individual collection IBFs are then grouped together to be encoded in a top level *Sync IBF*. iSync then computes the *digest* of the Sync IBF to be carried in multicast Sync Interests.

Similar to ChronoSync, iSync multicasts Sync Interests carrying the digest of the dataset state. However, a fundamental difference between the two is that iSync's Sync Interests do not solicit reply; they serve the purpose of dataset state notification only; i.e. iSync does not use long-lived Interests. Assuming a Sync group of three members, Alice, Bob, and Cathy. When Alice multicasts a Sync Interests I_A,

(1) If Bob detects a difference between the received digest in I_A and its local digest, Bob *fetches* the IBF from Alice. [3]
(2) When Bob receives the returned data packet D containing Alice's Sync IBF, from which Bob can identify the Collection IBF(s) that differs from its own, and retrieves the corresponding Collection IBF C_r from D's sender to compute the set differences between C_r and its local one.[4]
(3) Bob then sends an Interest carrying the set difference from the last step to retrieve the missing data names.

[3] This statement is copied from [15], which did not specify whether Bob unicasts, or multicasts, an Interest to fetch the IBF. Retrieving the IBF from Alice requires I_A including its sender information.
[4] This requires Data packet D contains its sender information.

Note that in the above steps, Bob retrieves information from a *specific node*. Therefore if Cathy sends a Sync interest around the same time as Alice with a different digest, Bob can carry out the dataset reconciliation with Cathy in parallel, enabling iSync to support simultaneous publications.

iSync's original implementation in CCNx no longer works, thus we are unable to run evaluation as we do with other Sync protocols. We note that iSync's design choice of support synchronization of arbitrary application data names implies that a growing dataset requires an increasing IBF size. iSync's 2-level IBF hierarchy could be extended to more levels to handle larger dataset namespace, which would also add additional complexity and round trip delays to the dataset reconciliation process. Two other protocols based on IBF, syncps and PSync, address the dataset namespace scalability in different ways as we explain below.

4.2.2 Limiting Synchronization Time. syncps [37] supports application data names and encodes the dataset namespace in an IBF similar to iSync. Different from iSync, whose Sync Interests carry the *digest* of dataset IBF and retrieves the IBF by sending Interests for data packets, syncps directly appends the dataset IBF to the end of each Sync Interest's name, and multicasts the Interest to the group. However, the size of an Interest packet is limited by network MTU size because NDN Interest packets cannot be fragmented [5], syncps keeps the IBF size under control by removing data names from the dataset after a predefined *lifetime*.

A participant P receiving a Sync Interest computes the dataset difference between the local and received IBFs. If the two are identical, P does nothing. That is, syncps also deploys long-lived Interest as ChronoSync: in the absence of new data generation, every group member refreshes its Sync Interest periodically. If P detects that the Interest sender misses some data items that P has, P sends a *Sync reply*, a data packet, that contains all the missing data items, if the resulting packet size is within the limitation of network MTU; otherwise some data items need to wait for next Sync Interest to be transmitted. Different from other Sync protocols which synchronize dataset namespace, syncps synchronizes the dataset directly – this approach can work well in scenarios where all participants desire all produced data, and all new data items can fit in a single data packet.[5]

Thanks to its use of IBF in namespace encoding, syncps significantly improves the Sync latency as compared to ChronoSync: with 7 data items per sec and without losses, syncps' 90-percentile Sync latency is around 0.1sec while the same measure for ChronoSync is almost 1sec. [6] However, since syncps multicasts Sync Interests to solicit dataset changes as ChronoSync does, it suffers from similar issues in handling simultaneous publications and packet losses (§4.1). Worse yet, if the recovery from simultaneous publications or packet losses takes longer than the predefined data item lifetime, some data items may have been removed from the dataset before being propagated. As a result, syncps does not guarantee reliable dataset synchronization.

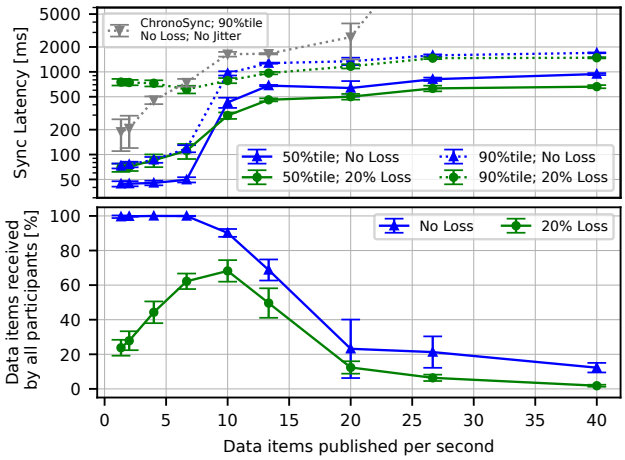

Figure 4: Sync latency and reliability using syncps.

Our evaluation results shown in Fig. 4 confirm that the percentage of data items received by all participants decreases with increasing publication rate. While increasing the data lifetime from the default of 2sec might mitigate some losses, this requires an increase in the size of the IBF, or otherwise risks an increasing rate of false positive IBF lookups.

4.2.3 Encoding Sequential Names in IBF. The PSync [57] protocol supports two modes of operation: *partial sync* and *full sync*. Here we focus on the latter which supports the same dataset namespace synchronization in a group as the other Sync protocols do.

To address the dataset namespace scalability issue, PSync adopts the sequential naming convention. It encodes the list of data names, i.e., /producer-name/latest-seq, in an IBF and carries the IBF in a Sync Interest name as syncps does. Encoding dataset state in IBF allows a PSync participant P to identify the data names it has but the sender of a received Sync Interest I does not. In this case, P sends a Sync Reply to I containing those data names, instead of directly putting the data in the Sync Reply as syncps does.

The high level operations in PSync are similar to those in ChronoSync as shown in Fig. 1a. The adoption of IBF enables PSync to identify state changes more quickly than ChronoSync. The adoption of the sequential naming convention enables it to scale well with the number of total data items, enabling it to encode the entire dataset namespace in an IBF and avoid the namespace scaling issues faced by iSync and syncps, ensuring reliable dataset state synchronization under all the evaluated conditions. However, PSync also multicasts Sync Interests to solicit state changes from anyone in the group using long-lived Interests, which bring costs to network routers and impair the performance when simultaneous publications and packet losses increase, as shown in Fig. 5. Furthermore, PSync's use of IBF also introduces an upper bound with regard to the number of data producers, which we discuss in §6.2.

4.3 Using Dataset State Representation Directly

Carrying an IBF in a Sync Interest I_{IBF} allow senders and receivers of I_{IBF} to identify and reconcile differences between them as long

[5]There is a discrepancy in Sync delay definition between syncps and other Sync protocols: the former synchronizes the dataset while the latter dataset namespace. To minimize the impact of this discrepancy, our evaluations assume that data item sizes are small enough to fit all newly produced data items into a single Sync reply.

[6]Unlike syncps, ChronoSync's design includes a random delay jitter to prevent excessive Sync Interests; to allow for a direct comparison of the protocols, we disabled ChronoSync's delay jitter in Fig. 4.

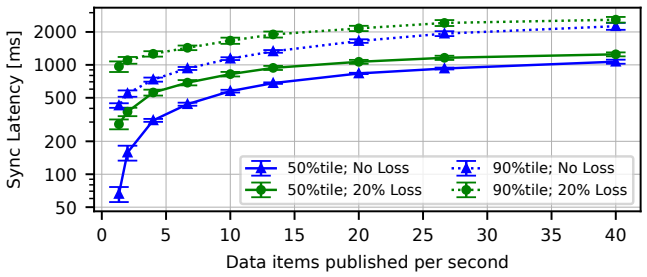

Figure 5: Sync latency of PSync.

Figure 6: Sync latency of State Vector Sync (SVS).

as the differences are within the IBF's encoding limit; otherwise multiple exchanges are needed to resolve the difference. In dynamic mobile environments, the dataset among participants may diverge significantly, multiple exchanges may also be infeasible with ad-hoc encounters.

State Vector Sync (SVS) [29] takes a rather different state encoding approach. Its design adopts the sequential data naming convention, and directly carries the dataset namespace representation, encoded as *[producer name, seq#]*-pairs, in each multicast Sync Interest. This state encoding approach was first proposed by DSSN [54] to support asynchronous communications among IoT devices. However different from DSSN, SVS treats Sync Interests as notifications only as iSync does, without soliciting Sync reply. Each participant sends Sync Interests under two conditions: i) event-driven, to notify others about a recent change, and ii) periodic, to mitigate potential losses of event-driven Sync Interests.

Fig. 1b shows two SVS participants P3 and P5 using event-driven Sync Interests, I_{P3} and I_{P5} respectively, to notify the group of their dataset updates due to a new data item by each. Since I_{P3} and I_{P5} carry different dataset state in their names, they are not merged by R2, thus both will reach P1. Processing the state vectors carried in both I_{P3} and I_{P5} will update P1's local dataset namespace to the latest state, demonstrating a solution to simultaneous publication. The evaluation results in Fig. 6 show that data publishing rate has no impact on SVS Sync latency, and adding packet losses only has a small impact. Increasing publishing rate leads to more frequent event-driven Sync Interests, compensating packet losses, making the Sync latency drop. Our evaluations also confirm that all examined Sync protocols can reliably synchronize the dataset namespace in all tested settings, with syncps as the only exception.

SVS's design decision to carry the raw dataset namespace state in each Sync Interest brings the advantage that each receiver R of a Sync Interest fully understands the carried dataset namespace, independent from R's own state or the number of previously missed messages. However, allowing a received Interest to change one's state opens the door for abuses. SVS prevents such abuse by signing all Sync Interests, as we discuss in §5.3.[7]. In addition, given Interest packet size is limited by network MTU, Sync groups with a large number of producers make it infeasible to carry the entire raw dataset namespace in Sync Interests. We discuss SVS scalability solutions in §6.2.

Two other Sync protocols, PLI-Sync [20] and ICT-Sync [2] were developed in parallel with SVS. Both of them adopt the sequential data naming convention and use Sync Interests to carry state-vector as notifications only, with each differs in its unique ways. PLI-Sync's unique feature is in taking advantage of sequential naming to optimistically *prefetch* the next data item using long-lived *Data Interests*, before being notified of the data item's production. When a participant successfully fetches a new data item, it informs the Sync protocol of the new name, which triggers an event-driven Sync Interest. This allows PLI-Sync to set a longer period for periodic Sync Interests. ICT-Sync utilizes intermediate nodes deployed in the network to aggregate Sync Interests from different participants and carrying different state vectors, which can help reduce both Sync latency and overhead. To minimize the state vector size, ICT-Sync uses numeric producer IDs instead of semantic names, which requires Sync entities to maintain a mapping between the numeric IDs and actual producer names.

5 SYNC PROTOCOL DESIGN SPACE

The previous section described various Sync protocol designs. In this section, we reflect on their design choices to gain a better understanding of the design space. In §5.1 and §5.2, we summarize the existing Sync protocols' design choices with respect to the basic design components identified in §3.2 and evaluate their pros and cons. We review the security implications from certain design choices in §5.3, and provide a summary of Sync protocol design comparison in §5.4.

5.1 State Representation and Encoding

Because a major goal of Sync protocols is to reliably synchronize the shared dataset namespace among participants in distributed applications, the first design decision is how to represent the shared dataset namespace. We have identified two design choices: (i) using application data names and (ii) using sequential naming. For the time being, we assume protocols may use either approach and defer a discussion on the implications of sequential naming to §6.1.

The next design decision is how to encode shared namespace state. Our examination shows four design choices for state encoding: (i) digest-based, as used by ChronoSync; (ii) IBFs, as used by syncps and PSync; (iii) combination of IBF and digest, as used by iSync; and (iv) directly using the namespace representation, as used by SVS.

With digest-based encoding, ChronoSync chooses sequential data naming and computes a digest of all [producer, seq#] pairs, and CCNx computes a digest over an application name-tree [46]. However, as we show in §4, a digest alone cannot directly identify

[7] Sync protocols using pulling do not need to secure their Sync Interests, because it is the Sync replies (NDN Data packets), not Sync Interests, that change receivers state

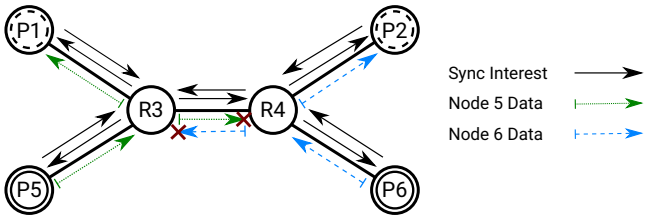

Figure 7: Participants P_5 and P_6 satisfy the same Sync Interest, leading to a state divergence.

namespace differences. iSync is the first to use IBFs for state encoding. It computes the digest of IBF to be carried in Sync Interests, and then uses the digest to retrieve the IBF. This addresses a digest's inability to identify changes in dataset state. However, because iSync chooses application data names as the namespace representation, iSync runs into IBF scalability issue as the total number of data items grows over time. It mitigates this issue by using an IBF hierarchy, which adds protocol complexity, synchronization overhead, and Sync latency.

syncps also uses application data names. It *circumvents* the above scalability problem by setting a time limit on how long data items can be synchronized. Unfortunately doing so leads to unreliable synchronization under adverse conditions (cf. Fig. 4). PSync *eliminates* the scalability problem caused by the number of data items by adopting sequential data naming, which reduces the number of items encoded in IBFs from the total number of data items to the number of producers in a Sync group.

One less-recognized issue with using IBF in Sync is that, when a group member P compares the IBF in a received Sync Interest I_{IBF} with its local IBF to identify the state differences, since IBF encodes numbers (not names directly), P cannot directly infer what names I_{IBF}'s sender has but itself is missing. This information can only be retrieved by P's own Sync Interest, which requires more packet exchanges. In contrast, by directly carrying the dataset namespace in Sync Interests, SVS enables any receiver R of Sync Interest I_{SVS} to interpret the dataset state carried in I_{SVS}, independent from R's local state. The cost is I_{SVS}'s larger size compared to I_{IBF}. We discuss how to mitigate SVS scalability in §6.2.

5.2 State Change Notification

All the NDN Sync protocols, including those not described in this paper, share two design features. First, their Sync Interests carry an encoding of the dataset namespace. Second, every participant in a Sync group *multicasts* its Sync Interests to the group. However, a multicast Sync Interest has one of two semantics: (i) it *pulls* updates from the group; or (ii) it *notifies* others about the sender's dataset state.

As an example, Fig. 7 illustrates how the *pull* semantic is handled in a network. Let us assume that 4 nodes, namely $P1$, $P2$, $P5$, and $P6$, are participants of the same Sync group. When the group is in a steady state, i.e. all participants have identical dataset state, the multicast Sync Interests from different participants are aggregated at routers, form 4 *overlapping* multicast Data delivery trees, with each tree rooted at one Sync participant and stretching its branches to all the others. The tree is maintained by persistent pending

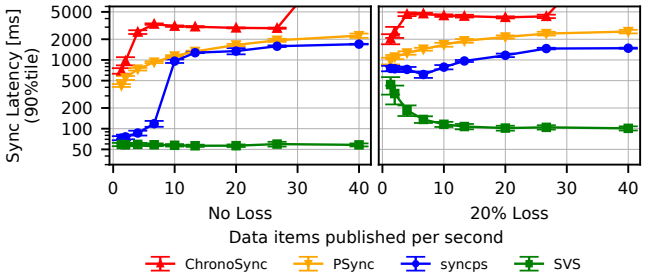

Figure 8: Sync latency across different protocols.

Sync Interests in the PIT of each router on the tree. We make the following observations:

(1) Given the next data production time is unpredictable, to fetch new data quickly, Sync Interests must stay in the PIT of each router along the multicast tree, with the PIT entries being refreshed by periodic Sync Interests before they expire.

(2) When a participant P produces new data D_P, P immediately sends a Sync Reply as a response to the pending Sync Interest, which is multicast-delivered to the group. However, if a Sync Interest from any participant $P1$ is lost, $P1$ will not receive the update about D_P; if another member $P2$ receives the update, $P2$'s next Sync Interest can inform $P1$ of the dataset state change but not the information of D_P if IBF encoding is used.

(3) A *multicast* Sync Interest I_m pulls potential replies from all the members in the group. When multiple recipients reply around the same time, at most one reply can reach I_m's sender. Different participants likely receive different updates based on their distances to data sources, leading to dataset state divergence. The example in Fig. 7 shows that $P5$ and $P6$ each send a reply to the pending multicast Sync Interest. Routers $R3$ and $R4$ receive both replies and drop the second one, thus $P1$ and $P2$ receive different replies. Recovering from this divergence takes additional Sync Interest-Reply exchanges.

On the other hand, using multicast Sync Interests for *notify* semantics, as done by iSync and SVS, removes the above identified issues. As one-way notifications, SVS Sync Interests can have short lifetime since they do not pull replies, thus do not need to stay pending at routers. If a Sync Interest I with the latest dataset state is delivered to all participants, the group is synchronized immediately; if I fails to reach some participants, a future Sync Interest can compensate for all the previous losses. We also note that Sync protocols using IBF encoding cannot use Sync Interests for dataset state notification: a participant P can tell what data items the sender of a Sync Interest has missed, but cannot tell what data items itself is missing. Therefore, Sync protocols using IBF encoding must rely on the pull semantic of Sync Interests, which leads to the above identified issues.

Fig. 8 shows the Sync latency results of the evaluated protocols. In the absence of packet losses, protocols using *pull* semantics (ChronoSync, syncps, PSync) exhibit a low latency at low publishing rates (ChronoSync and PSync's Sync latency include delay jitter, lowering delay jitter lowers Sync latency proportionally). With increased publishing rates, simultaneous publishing becomes more frequent. For protocols with pull semantics, simultaneous publications from multiple participants result in multiple replies

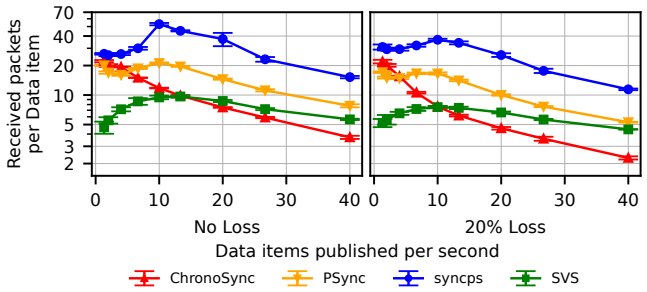

Figure 9: Avg. number of packets received by NDN forwarders including end nodes per published data item.

to the same Sync Interest, all these replies but one get dropped, requiring follow-up Sync Interests to retrieve all the updates (cf. Fig. 1a). Hence, the Sync latency increases with the publication rate for those protocols, while it stays constant for SVS with the *notify* semantics.

5.3 Securing Sync Interests

In NDN, when an Interest is used to retrieve a named piece of Data, it does not cause a state change of the data producer. Sync Interests in SVS notify the group members about latest dataset state, thus they can change receivers' dataset state. To prevent state changes caused by malicious Sync Interests, SVS must authenticate its Sync Interests.

SVS can sign Sync Interests using either the sender's key or a secret group key. While using symmetric group keys ensures identical Sync Interests can be aggregated, this approach requires additional mechanisms to maintain the shared group key. The current SVS design uses senders' keys to sign Sync Interests, preventing Interest aggregation and thereby potentially increasing network traffic. However, Fig. 9 compares network overhead across different Sync protocols, and suggests that the SVS's overhead stays on the lower side. The low overhead of SVS can be attributed to its effective Sync Interest suppression, as described in §4.3[8].

5.4 Sync Protocol Design Summary

Table-1 provides a concise summary of the Sync protocol design choices and consequent impacts on the protocols' reliability, scalability, overhead, latency, and scalability we discussed in this section. There are two additional impacts that are not shown here. First, the sequential naming offers a Sync protocol the scalability in terms of the number of data items and resiliency against packet losses, but it may need to pay the cost of providing a means to map the sequence number to actual data names. Second, all the Sync protocols require network multicast routing support, and supporting a very large number of multicast groups at a large scale has been a well recognized open issue for decades. As mentioned in §8, both issues are part of our future work.

[8]ChronoSync's overhead drops below that of SVS at high publication rate, because its delay jittering between 100-500msec damps Sync interest generation, while SVS multicast an event-driven Sync interest for every new data publication.

6 DISCUSSIONS

In this section we briefly discuss a few remaining issues related to Sync protocol design.

6.1 Data Naming

To achieve reliable namespace synchronization, the dataset state encoded in Sync Interests must be able to reliably convey the dataset state of their senders to the receivers over unreliable networks. This requires a Sync protocol design to convert a collection of general application data names into transport identifiers that are *resilient to losses*. Among the three state encoding approaches, digest, IBF, and direct use of the dataset namespace with sequential naming, digest alone lacks adequate information to assist the namespace synchronization, and using IBF to encode application data names does not scale well. This leaves dataset namespace using sequential naming as the most viable choice. However, applications in general assign semantic, instead of sequential, names to data.

The above conflict reflects the different requirements in data naming between applications and transport service. Unless/until new solutions are discovered, our observation so far suggests that it is essential for transport service to use sequential data naming to meet the goals defined in §3.1. However, this design choice leads to two new issues. First, a Sync protocol using sequential naming must map its sequential naming to application data names. Second and related, when a producer application passes down to Sync a signed NDN data packet D with its original name, Sync informs a remote consumer of the new data item by its sequence number, which it uses to request the data. This requires Sync to encapsulate D with its sequential name and securely bind that name to the content (D). As an example, syncps performs this encapsulation, albeit using IBF as the transport identifier, in the following way: when a participant P1 receives a Sync interest I_{P2} sent by P2, P1 encapsulate all the application data items that P2 misses in a single reply D. On receiving D, P2 extracts the encapsulated application Data items and passes them to the application process for standard NDN Data verification. This works well under the assumption that all group member want all the data at the same time, and all P2's missing data can fit into one Data packet; otherwise multiple rounds of Interest-Data exchange will be needed to synchronize.

A newly developed mobile health data sharing application, mGuard, provides the application data to transport mapping by a different solution [12]. Mobile health data are typically a collection of large amount of small data items. Instead of signing each data item, mGuard supports data authenticity through the use of manifest: collecting the names and hashes of multiple data items into a manifest (also a Data packet) which can be transported through Sync and then fetching/verifying the data items named in the manifest.

In summary, our analysis suggests that mapping application data names to sequential naming for transport is a viable direction, and more work is needed to address the name mapping and data encapsulation security designs. This problem was discussed in [42] and an initial solution developed for a pub/sub overlay over SVS [30].

6.2 Scaling to Large Sync Groups

Assuming Sync protocols adopt the sequential naming convention, we examine how well the dataset namespace encoding can scale

	Dataset Namespace	Sync Reliability	Dataset change notification	Network Overhead	Sync Latency	Protocol Scalability
ChronoSync	Sequential naming	Guaranteed dataset synchronization	Pulling by using long-lived Interest	Periodic refresh to maintain persistent PIT entry per Sync group	0.5 RTT in best case, increase with publication & loss rate	No specific upper bound with the number of data items or producers
syncps	Application data naming	No guarantee on dataset synchronization	Same as above	Same as above	Same as above	Number of data items upper bound by IBF size which is limited by network MTU
PSync	Sequential naming	Guaranteed dataset synchronization	Same as ChronoSync	Same as above	Same as above	Number of producers upper bound by IBF size which is limited by network MTU
SVS	Sequential naming	Guaranteed dataset synchronization	Using Sync Interest as notification	No PIT entry cost; Sync Interests with low refresh period	0.5 RTT when no packet loss, minimal increase with losses	Number of producers upper bound by network MTU, mitigatable (see §6.2)

Table 1: Feature matrix of the evaluated Sync protocols.

with the group size since the encoded namespace is limited by the size of Sync Interests. In general, an IBF with d cells can store $d/1.5$ elements with a low decoding failure probability [13]. PSync uses IBF encoding. It decodes the *differences* between two IBFs instead of decoding each IBF directly, thus its IBF size is proportional to the number of participants with *new* data items since the last synchronization between a node pair. In case of high packet losses or network partitions, the number of differences can be as large as the number of producers. The size of Interest packet sets the upper bound on the number of participants PSync can support in a highly dynamic environment.

SVS carries the raw dataset namespace directly in Sync Interests, thus for the same number of producers, its Sync Interest size can be much bigger than that of PSync. However, communicating raw dataset namespace offers SVS the freedom of putting as many, or as few, [producer-name, seq#]-pairs in a Sync Interest. It can cover the complete state vector by sending multiple Sync Interests, or simply not sending the full list each time. This removes the upper bound on the group size that SVS can support, turning the question to how to choose the entries to put into next Sync Interest. Initial exploration of this approach is reported in [41] with promising results. To provide quick "bootstrapping" support for new members with the complete dataset namespace, a latecomer may pick one participant P listed in the first Sync Interest it receives and fetch the complete list from P.

6.3 Sync Interest Multicast

Deering's seminal paper on IP multicast [10] points out two primary usages of multicast: (i) When an application sends the *same information* to multiple destinations, multicast is more efficient than unicast; and (ii) when an application needs to locate/query, or send information to multiple hosts whose addresses are unknown or changeable, where multicast can serve as a simple, robust alternative to configuration files, name servers, or other binding mechanisms. NDN has usage (i) designed in natively. For (ii), NDN can locate information via network routing/forwarding: network routing announcements inform routers where to locate requested data; in case of a small-scale NDN network without running a routing protocol, data can be located via self-learning [48]. For sending information to multiple entities without identifying their locations, that is the exact function Sync Interest multicast achieves, sending the group state updates to all members in a simple, robust way. That is why all Sync protocol designs multicast Sync Interests as described in §5.2.

One perceived hurdle in rolling out IP multicast is the concern regarding its scalability, a similar concern might arise for Sync Interest multicast, hence challenging the viability of NDN Sync protocols in general. We plan to address this issue in future work.

6.4 Use of the NDN Protocol

At first sight, SVS's use of NDN may seem not following the basic NDN protocol [22]. We discuss two SVS design choices that may have led to this concern.

Sync Interests without soliciting replies: NDN's communication model is retrieving named content chunks using Interest-Data exchange. SVS uses the name of Sync Interest to carry information for synchronization without soliciting Data replies. This design decision is based on the lesson learned from those Sync protocols, such as PSync, that solicit replies: when participants have different dataset states, they send replies to the same multicast Interest with different state updates, as visualized in Fig. 1a, and all but one of these replies get dropped due to NDN's one Interest for one Data packet principle, resulting in prolonged synchronization delay. Thus, we conclude that not soliciting Sync Interest reply is a better design choice. One may be concerned about using Interest flooding as DDoS attacks, which is a general threat to NDN, not specific to SVS's design.

Signed Sync Interests: In general, an Interest packet requests a named content chunk and does not disclose its sender's identity. Some use-cases, e.g., the NFD forwarder configuration [34] requires Interest sender authentication, and the *Signed Interest* format [35] is introduced to meet this requirement, which is used in the SVS design. We note that using Interest signatures does not necessarily disclose the signer's identity. For example, in [36], the signature's key locator field identifies the signing key by the key's fingerprint [16], which enables signature verification for those entities that already know the signers key and does not reveal the signer's identity.

6.5 ALF, NDN Sync, and Message Queues

The concept of Application-Level Framing (ALF) [7] lets applications define semantically meaningful Application Data Units (ADU). These self-contained ADUs may get fragmented for delivery over the network by lower protocol layers, which need to be reassembled back to be processed by applications. Both NDN Sync protocols and TCP/IP-based message queuing frameworks operate based on the concept of ADU, however the two differ in fundamental ways. NDN Sync synchronizes a dataset's state among distributed applications, with the dataset state identifies individual ADUs. Thereby, NDN Sync is taking the role of signaling the existence of ADUs, and NDN network layer retrieves ADUs with the benefit of multicast delivery and in-network caching.

In contrast, TCP/IP-based message queuing frameworks, such as MQTT, ZeroMQ, RabbitMQ, or Kafka [19, 32, 33, 52], are built on top of TCP/IP and work at application layer. Thus they suffer from IP's inefficient point-to-point packet delivery, lack a systematic security solution, and need additional infrastructure deployment, such as MQTT's broker nodes, to support distributed applications.

7 RELATED WORK

File and Folder Synchronization: This area of research has a long history. The rsync algorithm [51] synchronizes files and folders between two nodes. Setting up a central "cloud" server extends file sharing to multiple participants [11, 17, 39, 50]. Peer-to-peer protocols [8, 23] move file sharing towards decentralization by creating application layer overlays. All the above mentioned systems synchronize files via *pairwise* host-to-host communication; multicast dissemination, if used at all, is implemented at the application layer. NDN Sync moves the synchronization process among multiple parties down to the transport layer and increases efficiency by fully utilizing NDN's built-in multicast data delivery and in-network caching, and simplifies security protection.

Distributed Consensus Protocols: One of the most well-known consensus algorithms is Paxos [24] which performs a two-phase commit process among participants. A centralized controller can orchestrate all participants to allow everyone to propose, accept, and learn accepted values. In the absence of a controller, Paxos participants communicate with each other to reaching consensus. This process requires reliable $n \times n$ TCP connections; a costly solution. We note that many extensions of vanilla Paxos have been implemented and deployed in the wild. However, the fundamental issue is that the application layer's focus on data cannot be effectively mapped to the network layer service of connecting nodes. Further, while implementations such as Multi-Ring Paxos [26] can leverage IP multicast [9], achieving multicast in today's Internet is an inherently challenging exercise.

Reliable Multicast: The concept of IP multicast [9] was introduced around the same time as Stonebraker predicted the need for distributed DB systems [40]. Different from the latter, however, issues such as congestion control and reliable delivery hampered wide deployment of multicast [44]. Many reliable multicast solutions [38], including SRM [14], are based on the concept of Application Level Framing (ALF) [7], which suggests that network transport should preserve self-contained Application Data Units (ADUs) that are suitable for cross-layer processing. However, there is a fundamental mismatch between ADUs and multicast groups: the former focuses on *data items*, while the latter on *groups of nodes*. As a result, data is multicast to a predefined group of nodes, and this constraint makes it difficult to achieve efficient data dissemination to a group of nodes with different resource constraints. This mismatch also makes efficient loss recovery difficult: reliable multicast desires re-transmissions of specific ADUs by nearby members [25], while routers have no concept of ADUs and deliver all packets to the whole group.

The ALF concept and receiver-driven multicast ADU delivery provide two basic ingredients in the NDN design: NDN data packets represent ADUs identified by application-level names, and NDN lets consumers fetch desired data items. Doing so removes the aforementioned mismatch between network and upper layers. Sync allows organizing data transport according to application needs, which renders Sync as a framework for efficient data dissemination solutions. Moreover, this leads to the potential of using Sync to address other distributed system synchronization problems, such as those faced by distributed databases and Paxos.

8 CONCLUSION AND FUTURE WORK

This paper identifies a number of shared design patterns in the NDN Sync protocol designs and examines their impact on the latency, reliability, overhead, and security in dataset namespace synchronization. In particular, we notice the adoption of sequential data naming convention and its impact on a Sync protocol's scalability, the use of IBF encoding and its impact on dataset state synchronization, and the use of multicast Sync Interests to either pull or notify the group about dataset state changes, and the associated impact on protocol performance.

The Sync protocol development is still in its early stage. More investments are needed, at least to address two identified issues that we have mentioned earlier. One is Sync's use of sequential naming for application data, which results in data packet encapsulation, which in turn requires its own authentication. An initial solution in our recent work [30] needs further examination to see whether better solutions can be found. The other issue is two scalability concerns raised by Sync protocol designs, scaling with the number of producers in a Sync group, and scaling with the number of Sync groups. To scale better with the number of producers in a Sync group, one can let each SVS Sync interest carry a partial state vector, and a preliminary evaluation has shown promising results in this direction [41]. Scaling with the number of Sync groups is the same problem as scaling with the number of IP multicast groups in global deployment. One potential approach is to build a global overlay among regional Sync aggregators to propagate multicast Sync Interests, an idea suggested in [58]. We plan to further explore the solution space of the above problems.

Our examination of Sync protocol design underscores the important role a well-designed Sync protocol can play in future distributed applications over NDN. By summarizing the lessons learned and identifying the remaining issues, we hope that this work provides a cornerstone for future Sync protocol development efforts.

To foster reproducibility in research, source code used in this work is available under open-source licensing at [1].

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers and the shepherd David Oran for their valuable comments which helped us improve the paper's quality. This work was supported in part by US National Science Foundation under awards 1629769, 1719403, 2019085, and 2126148.

REFERENCES

[1] 2022. Repoducibility Set for "The Evolution of Distributed Dataset Synchronization Solutions in NDN". https://github.com/pulsejet/ndn-sync-eval
[2] Hila Ben Abraham, Jyoti Parwatikar, John DeHart, Adam Drescher, and Patrick Crowley. 2018. Decoupling information and connectivity via information-centric transport. In *ICN 2018 - Proceedings of the 5th ACM Conference on Information-Centric Networking*. 54–66. https://doi.org/10.1145/3267955.3267963
[3] B. Adamson, C. Bormann, M. Handley, and J. Macker. 2009. *NACK-Oriented Reliable Multicast (NORM) Transport Protocol.* RFC 5740.
[4] Alexander Afanasyev, Tamer Refaei, Lan Wang, and Lixia Zhang. 2018. A Brief Introduction to Named Data Networking. In *Proc. of MILCOM.*
[5] Alexander Afanasyev, Junxiao Shi, Lan Wang, Beichuan Zhang, and Lixia Zhang. [n.d.]. *Packet Fragmentation in NDN: Why NDN Uses Hop-By-Hop Fragmentation.* Technical Report NDN-0032. NDN.
[6] Alexander Afanasyev, Zhenkai Zhu, Yingdi Yu, Lijing Wang, and Lixia Zhang. 2015. The Story of ChronoShare, or How NDN Brought Distributed Secure File Sharing Back. In *Proc. of IEEE MASS Workshop on Content-Centric Networks.*
[7] David D. Clark and David L. Tennenhouse. 1990. Architectural considerations for a new generation of protocols. *ACM SIGCOMM Computer Communication Review* (1990), 200–208. https://doi.org/10.1145/99508.99553
[8] Bram Cohen. 2008. The BitTorrent Protocol Specification. https://web.archive.org/web/20140208002821/http://bittorrent.org/beps/bep_0003.html accessed: 2021-05-17.
[9] Steve Deering. 1989. *RFC1112: Host extensions for IP multicasting.* Technical Report.
[10] S. E. Deering. 1988. Multicast Routing in Internetworks and Extended LANs. *SIGCOMM Comput. Commun. Rev.* 18, 4 (aug 1988), 55–64. https://doi.org/10.1145/52325.52331
[11] Dropbox, Inc. 2021. Dropbox Homepage. https://www.dropbox.com/ accessed: 2021-05-17.
[12] Saurab Dulal, Nasir Ali, Adam Thieme, Tianyuan Yu, Siqi Liu, Regmi Suravi, Lixia Zhang, and La Wang. 2022. Building a Secure mHealth Data Sharing Infrastructure over NDN. In *Proceedings of the 9th ACM Conference on Information-Centric Networking.*
[13] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. 2011. What's the difference?: efficient set reconciliation without prior context. In *SIGCOMM.*
[14] Sally Floyd, Van Jacobson, Ching Gung Liu, Steven McCanne, and Lixia Zhang. 1997. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking* 5, 6 (1997), 784–803. https://doi.org/10.1109/90.650139
[15] Wenliang Fu, Hila Ben Abraham, and Patrick Crowley. 2015. Synchronizing namespaces with invertible bloom filters. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS).* 123–134.
[16] J. Galbraith and R. Thayer. 2006. *RFC4716: The Secure Shell (SSH) Public Key File Format.* Technical Report.
[17] Google LLC. 2021. Cloud Storage for Work and Home – Google Drive. https://drive.google.com/ accessed: 2021-05-17.
[18] GÉANT project. 2018. GÉANT topology map. https://www.geant.org/Networks/Pan-European_network/Pages/GEANT_topology_map.aspx accessed: 2021-05-10.
[19] Pieter Hintjens. 2013. *ZeroMQ: messaging for many applications.* O'Reilly Media, Inc.
[20] Yi Hu, Constantin Serban, Lan Wan, Alex Afanasyev, and Lixia Zhang. 2020. PLI-Sync: Prefetch Loss-Insensitive Sync for NDN Group Streaming. (2020). https://www.nist.gov/news-events/events/2020/09/ndn-community-meeting Named Data Networking Community Meeting 2020 (NDNComm'20).
[21] V. Jacobson. 1988. Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols (SIGCOMM '88).* ACM, New York, NY, USA. https://doi.org/10.1145/52324.52356
[22] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *CoNEXT '09: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies.* ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/1658939.1658941
[23] Patrick Kirk. 2003. Gnutella – A Protocol for a Revolution. http://rfc-gnutella.sourceforge.net/
[24] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Transactions on Computer Systems* 16, 2 (May 1998), 133–169. https://doi.org/10.1145/279227.279229
[25] Ching-Gung Liu, Deborah Estrin, Scott Shenker, and Lixia Zhang. 1998. Local Error Recovery in SRM: Comparison of Two Approaches. *IEEE/ACM Trans. Netw.* 6, 6 (Dec. 1998), 686–699. https://doi.org/10.1109/90.748082
[26] Parisa Jalili Marandi, Marco Primi, and Fernando Pedone. 2012. Multi-Ring Paxos. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012).* 1–12. https://doi.org/10.1109/DSN.2012.6263916
[27] N. Mimura, K. Nakauchi, H. Morikawa, and T. Aoyama. 2003. RelayCast: a middleware for application-level multicast services. In *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.* 434–441. https://doi.org/10.1109/CCGRID.2003.1199398
[28] Mini-NDN Authors. 2021. Mini-NDN: A Mininet-based NDN emulator. minindn.memphis.edu/ accessed: 2021-05-10.
[29] Philipp Moll, Varun Patil, Nishant Sabharwal, and Lixia Zhang. 2021. *A Brief Introduction to State Vector Sync.* Technical Report NDN-0073. NDN.
[30] Philipp Moll, Varun Patil, Lixia Zhang, and Davide Pesavento. 2021. Resilient Brokerless Publish-Subscribe over NDN. In *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)* (San Diego, CA, USA). IEEE Press, 438–444. https://doi.org/10.1109/MILCOM52596.2021.9652885
[31] Philipp Moll, Wentao Shang, Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. 2021. *A Survey of Distributed Dataset Synchronization in Named Data Networking.* Technical Report NDN-0053, Revision 2. Named Data Networking. 1–18 pages.
[32] mqtt.org. 2020. MQTT: The Standard for IoT Messaging. https://mqtt.org/ accessed: 2021-07-19.
[33] Neha Narkhede, Gwen Shapira, and Todd Palino. 2017. *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale* (1 ed.). O'Reilly Media, Inc.
[34] NDN Project team. 2018. NFD Management protocol. (2018). https://redmine.named-data.net/projects/nfd/wiki/Management accessed: 2021-07-29.
[35] NDN Project team. 2021. NDN Packet Format Specification version 0.3: Signed Interest. (2021). https://named-data.net/doc/NDN-packet-spec/current/signed-interest.html accessed: 2021-07-29.
[36] Kathleen Nichols. 20121. Trust Schemas and ICN: Key to Secure IoT. In *Proceedings of the 8th ACM Conference on Information-Centric Networking (ICN '21).* Association for Computing Machinery, New York, NY, USA.
[37] 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).* Association for Computing Machinery, New York, NY, USA, 112–122. https://doi.org/10.1145/3357150.3357397
[38] Katia Obraczka. 1998. Multicast transport protocols: a survey and taxonomy. *IEEE Communications Magazine* 36, January (1998), 94–102.
[39] ownCloud GmbH. 2021. ownCloud – share files and folders, easy and secure. https://owncloud.com/ accessed: 2021-05-17.
[40] M. Tamer Ozsu and P. Valduriez. 1991. Distributed database systems: where are we now? *Computer* 24, 8 (1991), 68–78. https://doi.org/10.1109/2.84879
[41] Varun Patil, Sichen Song, Guorui Xiao, and Lixia Zhang. 2022. Poster: Scaling State Vector Sync. In *Proceedings of the 9th ACM Conference on Information-Centric Networking.*
[42] V. Patil and L. Zhang. 2021. Considerations for Higher Level Transports over Sync. (2021). https://www.nist.gov/news-events/events/2021/10/ndn-community-meeting-2021 NDNComm 2021.
[43] Peter R. Pietzuch and Jean Bacon. 2003. Peer-to-Peer Overlay Broker Networks in an Event-Based Middleware. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems* (San Diego, California) *(DEBS '03).* Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/966618.966628
[44] Adrian Popescu, Doru Constantinescu, David Erman, and Dragos Ilie. 2007. A survey of reliable multicast communication. *NGI 2007: 2007 Next Generation Internet Networks - 3rd EuroNGI Conference on Next Generation Internet Networks: Design and Engineering for Heterogeneity* (2007), 111–118. https://doi.org/10.1109/NGI.2007.371205
[45] Jon Postel. 1981. *RFC793: Transmission Control Protocol.* Technical Report.
[46] ProjectCCNx. 2012. CCNx Synchronization Protocol. CCNx 0.8.2 documentation. https://github.com/ProjectCCNx/ccnx/blob/master/doc/technical/SynchronizationProtocol.txt
[47] Klaus Schneider, Cheng Yi, Beichuan Zhang, and Lixia Zhang. 2016. A Practical Congestion Control Scheme for Named Data Networking. In *Proc. of ACM ICN.*
[48] Junxiao Shi, Eric Newberry, and Beichuan Zhang. 2017. On Broadcast-based Self-Learning in Named Data Networking. In *Proceedings of IFIP Networking.*
[49] Sichen Song and Lixia Zhang. 2022. Effective NDN Congestion Control Based on Queue Size Feedback. In *Proceedings of the 9th ACM Conference on Information-Centric Networking.*
[50] Synology Inc. 2021. Synology Drive | Your private cloud for file access and sharing anywhere. https://www.synology.com/en-us/dsm/feature/drive accessed: 2021-05-17.
[51] Andrew Tridgell and Paul Mackerras. 1998. The rsync algorithm. https://rsync.samba.org/tech_report/

[52] VMware Inc. 2021. Messaging that just works – RabbitMQ. https://www.rabbitmq.com/ accessed: 2021-07-19.
[53] L. Wang, V. Lehman, A. K. M. Mahmudul Hoque, B. Zhang, Y. Yu, and L. Zhang. 2018. A Secure Link State Routing Protocol for NDN. *IEEE Access* 6 (jan 2018), 10470–10482.
[54] X. Xu, H. Zhang, T. Li, and L. Zhang. 2018. Achieving Resilient Data Availability in Wireless Sensor Networks. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*.
[55] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2013. A Case for Stateful Forwarding Plane. *Computer Communications: ICN Special Issue* 36, 7 (April 2013), 779–791.
[56] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patric Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *ACM Computer Communication Reviews* (June 2014). http://dx.doi.org/10.1145/2656877.2656887
[57] Minsheng Zhang, Vince Lehman, and Lan Wang. 2017. Scalable Name-based Data Synchronization for Named Data Networking. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*.
[58] Zhenkai Zhu and Alexander Afanasyev. 2013. 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. http://icnp13.informatik.uni-goettingen.de/index.html