NDN Technical Report NDN-0076. http://named-data.net/techreports.html
Revision 1: April 2022 (to be published)

The Evolution of Distributed Dataset Synchronization Solutions in NDN

Abstract

Distributed dataset synchronization, or Sync in short, plays the role of a transport service in the Named Data Networking (NDN) architecture. In this paper, we conduct a systematic examination of NDN Sync protocol designs, identify common design patterns, reveal the 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. In addition, we discuss the relationship between transport and application naming, the implications of namespace encoding for Sync group scalability, and support for Interest multicast.

CCS Concepts

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

Keywords

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

1 Introduction

Most people in the ICN community are familiar with NDN's basic *network* communication model of Interest-Data exchanges, which has been documented in numerous publications. However, it is hard to find a comprehensive overview of NDN *transport* services. 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 practice where distributed applications rendezvous at centralized servers through point-to-point transport service, NDN aims to leverage its data-centric nature with semantic naming at the network layer to enable distributed entities to communicate *directly*, without relying on centralized servers. That is, NDN needs a new type of multiparty transport service that can enable a group of distributed entities to communicate in a secure, reliable, and resilient way.

The new NDN transport service that emerged from the last ten years of NDN research is the namespace synchronization of shared datasets among a group of distributed entities. A variety of NDN Sync protocols have been developed [33], and the designs have evolved over the years. This SoK paper aims to provide a systematic examination of all the existing NDN Sync protocol designs and the lessons learned from different design choices, and to identify remaining issues and future 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 exemplify the impact of different design choices through case studies of representative Sync protocol designs (§4), and then step up a level to compare and contrast the different design decisions of different protocols and validate our analysis with comparative evaluation results (§5).

From the above exercise we learn that each Sync protocol faces three major design decisions: how to represent the dataset namespace containing data produced by all participants, how to encode the dataset namespace in communication[1], and how to disseminate dataset state changes effectively and efficiently. We discover that a few design patterns are shared among the Sync protocols (see §4) and reveal the need for multicasting Sync Interests and the limitation 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 (§8).

2 The Role of Sync in NDN

Generally speaking, transport services bridge the gap between the services that applications desire and the services the network layer provides. Today's transport protocols, as exemplified by TCP, convert IP's point-to-point datagram service to reliable data delivery between two application

Conference'17, July 2017, Washington, DC, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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

processes, which are identified by a pair of IP addresses and transport port numbers. TCP provides three basic functions:
(1) demultiplexing incoming packets from IP to different application processes;
(2) providing reliable byte stream delivery; and
(3) performing network congestion control[2].

Applications running over TCP avoid the burden of supporting reliable data delivery on their own. However, building multiparty distributed applications using TCP's point-to-point connections requires setting up $n \times n$ TCP connections which is complex and inefficient. While a few reliable UDP multicast protocols such as NORM [3] have been developed, we note here that support for IP multicast is generally limited. Futher, the nature of host-to-group nature of IP multicast requries necessarily distributing the entire dataset to all group participants, which can be inefficient in scenarios where certain subsets of group members only require certain subsets of the dataset.

In Named Data Networking (NDN) [53], the network layer fetches named content chunks. These data chunks carry semantic names that uniquely identify their content, as well as 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 require a transport header to carry information for demultiplexing;
(2) NDN allows different applications to pick and choose different reliability definitions; and
(3) NDN moves network congestion control to the network layer where it belongs [47, 52].

NDN's data-centric model facilitates multiparty communication by letting applications directly request the desired data using data names, but it remains difficult for application developers to build distributed applications directly using Interest-Data exchange provided by the network layer. For example, a participant in a multiparty application needs to know what data is produced by others as soon as possible, in order to retrieve it promptly as needed by the application.

The examination of several pilot NDN applications, such as tools for multiparty file sharing [4], audio and video conferencing [18, 57], and group messaging [56] showed one commonality of all these distributed applications – participants share and collect distributed datasets. Hence, all those applications share the need for a simple, flexible, and resilient synchronization service. *Distributed dataset synchronization*, or *Sync* provides such a service, acting as a transport layer abstraction on top of NDN Interest-Data exchange.

[2]Congestion control was not part of the TCP's functions when published in 1981 [45], but added later to mitigate the Internet congestion meltdown [23].

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

Sync is NDN's transport to serve applications. 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 that Sync should meet.

Supporting diverse data reliability requirements: An important lesson learned from TCP is that, although all applications desire reliable data delivery, the definition 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 participants of a Sync group. Having learned the published data names, each participant may decide whether and when to retrieve all or some of the data items based on the application's need.

Reliable dataset *namespace* Sync: This requirement denotes the protocol's ability to deliver every dataset namespace update to all 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 changes to meet the low-latency requirement of applications such as online games.

Resilient performance: Networking environment may range from stable infrastructure networks with a low loss rate to ad-hoc wireless networks with intermittent connectivity. To provide a general transport service, a Sync protocol should work well in both stable infrastructure networks and mobile ad-hoc networks.

3.2 Components in Sync Protocol Design

Sync protocol development efforts identified three 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 that allows transmission over the network (namespace encoding). Third, each participant needs to notify the others of any changes it makes to the shared dataset (*state change notification*). We illustrate each design component below.

Namespace Representation: Each participant maintains the dataset namespace locally, i.e., the participant's local view of all data items generated so far. 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 all the names in the shared dataset. The second approach assumes that data by each producer P can be named sequentially. Thus all the data items produced by P can be represented by a pair of [producer name, seq#], and the namespace of the entire shared dataset is simply a list of [producer name, seq#]-pairs, one for each participant. We discuss the implications of both approaches in §6.1.

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. One design requirement is an efficient encoding of the entire shared dataset namespace. As we discuss in §6.1, state encoding solutions are directly tied to the dataset namespace representation, and different solutions lead to different design tradeoffs.

State Change Notification: With the data-centric NDN protocol, participants fetch Data by names. It is Sync's responsibility to inform all paricipants in a Sync group of data production as soon as possible. For Sync to work, however, participants in a group cannot fetch each other's namespace updates without *naming individual participants*. Therefore, the dataset state updates exchange need to be carried using *multicast* Sync Interest packets to the Sync group.

Up to now, we have observed two approaches. The first one lets every participant *pull* new namespace changes by multicasting a Sync Interest and receiving new changes in reply Data packets (Sync Replies). The second approach lets data producers *notify* all others about changes by multicasting a Sync Interest.

The use of Sync Interest multicast deserves a further clarification: 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. Note that *Interest* multicast is different from *Data* multicast. While Data multicast is inherently supported, Interest multicast requires multicast routing support to forward Interests to all participant of a Sync group. We further discuss this issue in §6.3,

4 Sync Protocol Design: Case Studies

Over ten different NDN Sync protocol designs have been developed over the years [33], with later designs making improvements based on lessons learned from previous ones. In this section, we examine the design choices of 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.

We choose Mini-NDN [30] as our evaluation environment and emulate the topology of the GÉANT research network [19] with 45 nodes; each link has unlimited bandwidth and a 10msec propagation delay. In each experiment run, we randomly select 20 nodes (excluding 4 central nodes) to be participants in a Sync group[3]. To observe each Sync protocol's performance in stable and unstable environments, we vary the loss rate of each link from zero to 20%. We vary the publishing rate of each Sync participant from 1 data item every 15 seconds to 2 data items per second (in evaluation plots, the X-axis indicates the **data publication rate** of the *whole* group). We experimented with setting the Sync Interest lifetime of ChronoSync, PSync, and syncps to 1sec, 4sec and 10sec; the results reported in this paper use 1sec as Sync Interest Lifetime. Default settings are used for other Sync protocol parameters. Error bars in the figures denote the 95% confidence interval from 10 runs for every emulation setting. The presented 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 new publication notifications received by all participants.

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* [56], as the second NDN Sync protocol, takes the same approach of using a digest to to represent the dataset state. However ChronoSync follows the 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 carrying its state digest to inform others in the same Sync group of its own dataset state. Under stable conditions where all participants have

[3]This study reports the evaluation of Sync protocols' performance with a specific number of Sync nodes; experimentation with different numbers are ongoing, with initial results conforming to that reported in this paper.

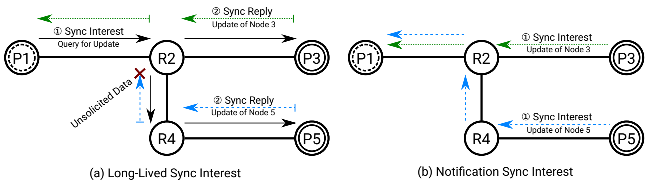

Figure 1: A simple scenario to demonstrate simultaneous publication issues.

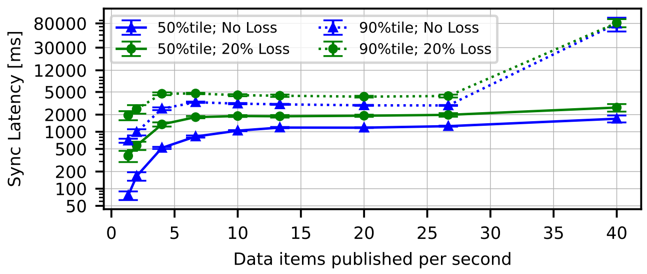

Figure 2: Sync latency of ChronoSync.

synchronized dataset state, they issue 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) before the previous one expires, 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 recomputes the state digest and sends an NDN Data packet in reply to the pending I_{P1}, and also immediately sends a new 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 cannot identify the exact namespace differences from the digest. 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 to see if D_{rec}'s sender lags behind. If P finds D_{rec} in the log, it sends a reply with it current dataset state; if not, it waits for a random time (delay jitter) for potential incoming Sync replies that may 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.

One cause for unrecognized state digests 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 Sync Interest simultaneously, R2 forwards the reply from P3 (due to a shorter path), and drops the one from P5. Worse yet, if P3's reply is lost between P1 and R2, P1 will be unaware of the new publications until it sends the next Sync Interest, or receives a Sync Interest with updated state digest. In either case, the Sync latency is increased proportionally to the Sync period length.

Fig. 2 shows ChronoSync's performance. In the absence of packet losses, ChronoSync performs well at low publication rate; when 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. Under the condition of 40 data items published per second and 20% loss rate, the 90-percentile Sync latency is around 80sec. When the Sync period is varied between 1 and 10sec, no significant impact on Sync latency or overhead is observed.

4.2 Encoding Dataset Namespace by IBF

The lessons learned from ChronoSync suggest that, to quickly reconcile dataset namespace differences, 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 only infer the names it has that I's sender does not, but R 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 the first to use IBF. Its dataset namespace is a collection of general 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*.

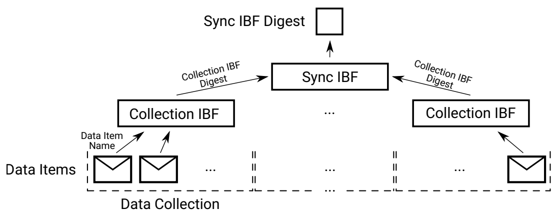

Figure 3: iSync's Multi-Level IBF Structure

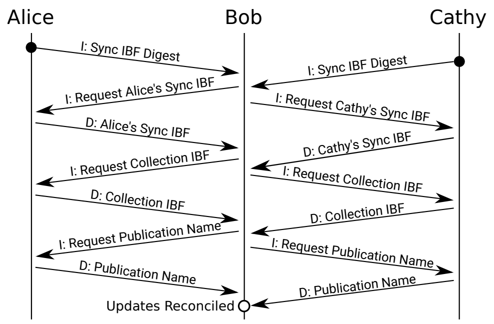

Figure 4: Simultaneous publishing in iSync

iSync then computes the digest of the Sync IBF to be carried in multicast Sync Interests.

iSync's use of Sync Interests to multicast the digest of the dataset state resembles the ChronoSync design, however there is a fundamental difference between the two: iSync uses Sync Interests do not solicit reply; they serve the purpose of dataset state notification only. Assuming a Sync group of three members, Alice, Bob, and Cathy. When Alice multicasts a Sync Interests I_A as shown in Fig.4,

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

Note that in all the above three steps, Bob retrieves information from a *specific node*. Therefore if Cathy sends a Sync interest around the same time as Alice, Bob can carry out the dataset reconciliation with Cathy in parallel, enabling iSync to support simultaneous publications as shown in Fig. 4. This ability is based on the assumption that every IBF is maintained by *a specific publisher*.

iSync's original implementation on CCNx no longer working, we are unable to run evaluation as we do with other

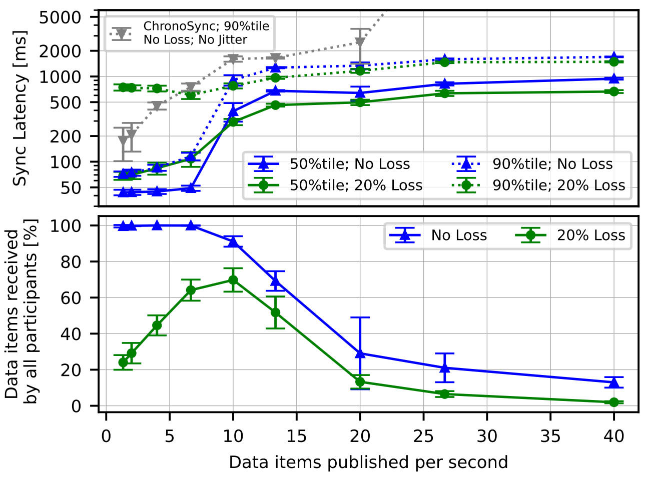

Figure 5: Sync latency and reliability using syncps.

Sync protocols. We view iSync's support of general application data names as a plus, however, this design choice also 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 more 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 Circumventing Namespace Scalability Issues by Limiting Synchronization Time Similar to iSync, syncps [39] supports application data names and encodes the dataset namespace in an IBF. To keep the IBF size under control, syncps removes data names from the dataset after a predefined lifetime. Different from iSync which retrieves IBF in data packets, syncps appends the IBF to the end of each Sync Interest's name and multicasts the Interest to the group. A participant P receiving a Sync Interest computes the dataset difference between the local and received IBFs. If the Interest sender misses any data, P sends a reply that contains the missing data. Different from other Sync protocols which synchronize dataset namespace, syncps synchronizes the dataset directly – this approach can work well for scenarios where all Sync participants need all the produced data.

Like ChronoSync, syncps multicasts Sync Interests to solicit dataset changes, thus it shares similar issues in handling simultaneous publications and packet losses (Section 4.1). Moreover, if the recovery from simultaneous publications or packet losses takes longer than the predefined data item lifetime, some data items may be removed from the dataset before being synchronized. As a result, syncps only provides a weak consistency model in contrast with the other Sync protocols, all of which are eventually consistent. This is confirmed by our evaluation results shown in Fig. 5, where a data item's lifetime is set to syncps's default value of 2sec. In the absence of losses, the percentage of data items received by all the participants drops from 100% to below 20% as the publishing rate increases; with packet losses, the reliability gets even worse. These losses may be mitigated by increasing the data lifetime; this may however lead to an increase in the size of the IBF, or an increased rate of false positives as the degree of filling increases, due to the probabilistic nature of the IBF.

However, 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 (where syncps still maintains a high delivery percentage) and without losses, syncps' 90-percentile Sync latency is around 0.1sec while the same measure for ChronoSync[4] is almost 1sec.

4.2.3 Encoding Sequential Names in IBF The PSync [54] 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.

The high level operations in PSync are similar to those in ChronoSync as shown in Fig. 1a, including the delay jittering in sending periodic Sync interests. Because PSync also multicasts Sync Interests to solicit state changes from anyone in the group, its performance is also affected by simultaneous publishing and packet losses as shown in Fig. 6. PSync reliably synchronizes the dataset state under all evaluated conditions; its adoption of the sequential naming convention enables it to encode the entire dataset namespace in an IBF, avoiding the namespace scaling issue faced by iSync (see discussions in §6.2 regarding the upper bound on feasible namespace size).

4.3 Using Dataset State Representation Directly

Carrying IBF in a Sync protocol message M allow senders and receivers of M to identify and reconcile differences between them as long as the differences are within the IBF's encoding limit; otherwise multiple exchanges are needed to resolve the difference. In dynamic mobile environments, the

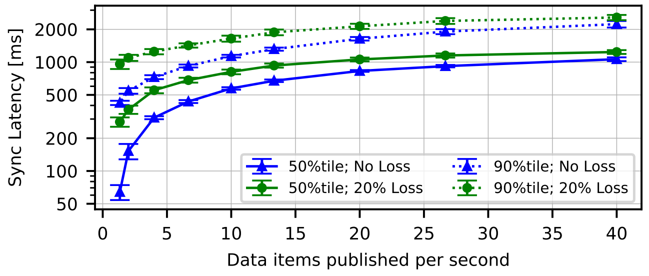

Figure 6: Sync latency of PSync.

dataset among participants can diverge significantly, multiple exchanges may also be infeasible during ad hoc encounters. Adopting the sequential data naming convention, State Vector Sync (SVS) [31] takes a different state encoding approach – directly carrying the entire dataset namespace representation, encoded as *[producer name, seq#]*-pairs, in each multicast Sync Interest, which was initially proposed in VectorSync [48] and DSSN [51] (not discussed in this paper).

Similar to iSync, SVS treats Sync Interests as notifications, without soliciting 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 messages. 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, they are not merged by R2 and will both reach P1. After P1 processes the state vectors carried in I_{P3} and I_{P5}, its local dataset namespace is updated to the latest state, demonstrating a solution to simultaneous publication. The evaluation results in Fig. 7 show that the publishing rate has no impact on Sync latency, and packet losses only make a low impact. A higher publishing rate leads to higher number of event-driven Sync Interests which compensate packet losses, therefore the Sync latency drops with the publishing rate. We also note that our evaluation confirms that all examined Sync protocols can reliably synchronize the dataset namespace in all tested settings, with syncps as exception.

SVS's design decision to encode 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, NDN Interests are not secured by default; 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. Also, given Interest packet size limitations, carrying the raw dataset namespace in Sync Interests may become infeasible with increasing group size. We discuss potential scaling solutions in §6.2.

[4]syncps's design does not include a delay jitter; to allow for a direct comparison of the protocols, we disabled ChronoSync's delay jitter in Fig. 5.

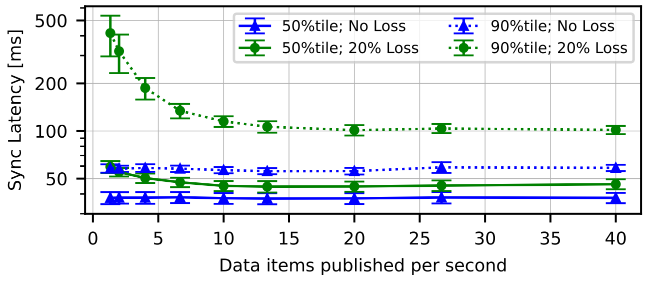

Figure 7: Sync latency of State Vector Sync (SVS).

The Sync protocols PLI-Sync [22] and ICT-Sync [2] were developed in parallel with SVS. Both 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 by long-lived *data Interests*, before being notified of the data item. When a participant successfully fetches a new data item, it informs the Sync protocol of the new name, allowing Sync to set a longer Sync period. ICT-Sync utilizes intermediate nodes deployed in the network to aggregate Sync Interests from different participants 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. We now 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 then review the security implications from certain design choices in §5.3.

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 the shared namespace state. We have observed 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, a single digest or a digest hierarchy is computed from the dataset namespace. While ChronoSync chooses sequential data naming, one could also compute a digest over an application name-tree as CCNx does [46]. However, as we show in §4, a digest alone cannot directly identify namespace differences.

iSync chooses application data names as the namespace representation and is the first to use IBFs for state encoding. As the dataset grows in size, iSync faces an IBF scalability issue. It mitigates this issue by using an IBF hierarchy, which adds protocol complexity, synchronization overhead, and Sync latency. syncps *circumvents* the above scalability problem by specifying a time limit on how long data items are synchronized. Unfortunately doing so leads to unreliable synchronization (cf. Fig. 5). PSync *eliminates* the above scalability problem by adopting sequential data naming as the namespace representation, which reduces the number of items encoded in IBFs from the number of data items to the number of producers.

IBF identifies state differences by comparing a received IBF with the local one. However, since an IBF can only encode numbers (not names directly), a node cannot directly infer what names another node has but it is missing. This leads to more message exchanges and longer Sync latency. In contrast, by directly carrying the dataset namespace in Sync Interests, SVS enables anyone receiving a Sync Interest to interpret the carried namespace, independent from the local dataset state. However, the dataset namespace can be larger in size compared to using IBF encoding, thus carrying the full dataset namespace has an impact on SVS's scalability. We discuss this issue with potential solutions in §6.2.

5.2 State Change Notification

With iSync as an exception,[5] all the other 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. A multicast Sync Interest in this set of protocols has one of the two semantics: (i) it *pulls* updates from the group; or (ii) it *notifies* others about the sender's dataset state.

As an example, Fig. 8 illustrates how the *pull* semantic is handled in a network. Let us assume that 4 nodes are participants of the same Sync group. When the group is in a steady state, i.e. all participants have identical dataset state,

[5]As described in §4.2.1, iSync puts the Sync IBF digest in each Sync interests and multicast it to the group as one-way notifications; when dataset state difference is detected, an iSync node performs a few rounds node-to-node communication to reconcile the difference (Fig.4), assuming that each Collection IBF is managed by a *specific node*.

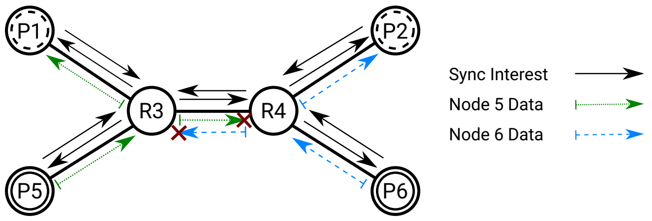

Figure 8: Participants P_5 and P_6 satisfy the same Sync Interest, leading to a state divergence.

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

(1) Given the next namespace update time is unpredictable, Sync Interests may 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 send a Sync Reply as the 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 for potential replies from all the members in the group, but when multiple recipients reply, at most one reply can reach I_m's sender. Different participants likely receives different updates based on their distances to data sources, leading to dataset state divergence. The example in Fig. 8 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.

Using multicast Sync Interests for *notify* semantics, as done by SVS, removes the above-identified issues. As one-way notifications, SVS Sync Interests can have short lifetime as they do not pull replies and do not need to stay pending at forwarders. 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 previously losses.

However, we note that Sync protocols using IBF encoding cannot use Sync Interests for dataset state notification: a participant P can tell what the sender of a Sync Interest has missed, but cannot tell what itself is missing. Therefore, Sync protocols using IBF encoding must rely on the pull semantic of Sync Interests, which leads to above-identified issues.

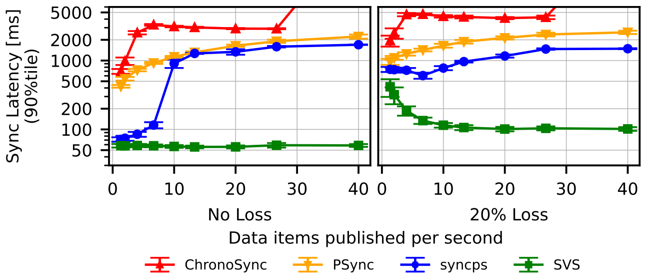

Figure 9: Sync latency across different protocols.

Fig. 9 shows the Sync latency results of the evaluated protocols. Without packet loss, protocols using *pull* semantics (syncps, PSync, ChronoSync) exhibit a low latency at low publishing rates[6]. With an increasing publishing rate, simultaneous publishing becomes more frequent. For protocols with pull semantics, simultaneous publications result in multiple participants generating replies for the same Sync Interest, leading to Sync replies getting dropped and requiring follow-up Sync Interests to retrieve all dataset updates (cf. Fig. 1a). Hence, the Sync latency increases for those protocols, while it stays constant for SVS with *notify* semantics.

5.3 Securing Sync Interests

In NDN, Interests retrieve a named piece of Data without causing a state change of the producer. In SVS, Sync Interests notify the latest dataset state and can thereby change receivers' dataset state. To counteract malicious state changes, Sync Interests must be authenticated in this case.

Sync Interests can be signed using either the sender's key or a secret group key. The first approach makes Sync Interests no longer aggregatable. The second approach can keep Sync Interests aggregatable but requires additional mechanisms for maintaining the shared group key; moreover, if a rogue participant in the group injects bogus Sync Interests, they cannot be distinguished from other participants' messages. The current SVS design takes the first approach. SVS tries to keep the number of Sync Interests low via Interest suppression (see §4.3), Sync Interest aggregation may only bring a minor gain, while using sender signatures keeps the design simple and improves security aspects. Fig. 10 shows the protocol overhead comparison and suggests that the SVS's overhead with unaggregatable Sync interest remains low[7].

It is also worth noting that Sync Interests in syncps and PSync do not change the recipient's dataset namespace (only

[6]ChronoSync and PSync's Sync latency include delay jitter (see §4.1). Evaluations with lower delay jitter show Sync latency lowered proportionally.
[7]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 a Sync interest for every new publication.

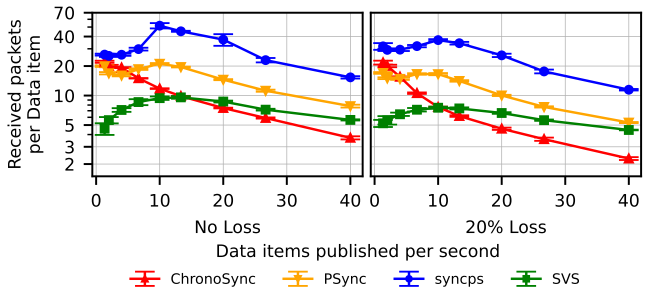

Figure 10: Average number of Sync Interest and reply packets received by NDN forwarders including end nodes in relation to each published data item.

replies to Sync Interests do), therefore, according to their designs, are not secured.

6 Discussions

In this section we discuss two identified issues in Sync protocol designs: data naming and Sync protocol scalability, and briefly touch on a related topic of multicast routing support.

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 requirement cannot be met without converting a collection of general application data names into transport identifiers that are resilient to losses. Although the three state encoding approaches, digest, IBF, and direct use of the dataset namespace with sequential naming, all qualify as transport identifiers, digest lacks adequate information to assist the namespace synchronization, and using IBF to encode application names does not scale. This leaving dataset namespace using sequential naming, either encoded in IBF or directly carried in Sync interests, as the viable choice at this time. 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, we observe that Sync deems sequential data naming essential to meet the goals defined in §3.1. This observation leads to two new issues. First, a Sync protocol should provide a mapping between its sequential naming and original application data names to hide the former from applications. We note that sequential naming retains the semantic data name prefix of a producer and only abbreviates the lower part by a unique number. We also point out that, different from application data names, transport data identifiers, even by using sequence numbers, are not permanent: it is infeasible for Sync to maintain an ever increasing mapping table between application names and corresponding sequential names, and a sequence number will eventually wrapping around, losing its uniqueness.

Second, and related, when a producer application passes down a signed NDN data packet D_{new} with its original names to Sync, a remote consumer learns, and requests, the new data item by its sequence number. This requires Sync to encapsulate D_{new} with its sequential name and securely bind that name to the content (D_{new}). As an example, syncps performs this encapsulation, albeit using IBF as the transport identifier, in the following way: when a producer P1 receives a Sync interest I_{P2} from P2, P1 encapsulate all the application data packets that P2 misses in a reply D_{new}. Once receiving D_{new}, P2 extracts the encapsulated application Data packets and passes them to the application process for standard NDN Data verification. Since the application Data packets carry their own signatures, syncps fills the signature field of D_{new} with an integrity checksum by default, with an option of using different signing policies defined in an encapsulation security manager.

In summary, our analysis suggests that mapping application data names to sequential naming for transport is a viable direction and that more research is needed to examine the name mapping and data encapsulation security designs. An initial design including such mappings is discussed for a publish-subscribe overlay on top of SVS [32]. In this design, data with application names is encapsulated in data packets named by SVS. Participants publish and retrieve the mapping data using so-called Mapping Interests. While SVS uses sequence numbers as transport identifiers, these sequence numbers are transparent to applications, which handle a publish-subscribe API.

6.2 Scaling to Large Sync Groups

We now examine the impact of state encoding on scaling to large groups, focusing on Sync protocols that adopt the sequential naming convention. PSync encodes the dataset namespace using IBFs. Normally, an IBF with d cells can store $d/1.5$ elements with a low decoding failure probability [13]. However, since PSync decodes the *differences* between two IBFs rather than decoding each IBF directly, its IBF size should instead be proportional to the number of producers with *new* data items since the last synchronization between a node pair. When the network is stable and the publishing rate is low, the number of differences should be 1; when there are high losses or network partitions, the number of differences can be as large as the number of producers. Since the IBF size is limited by the Interest packet size, this leads to an upper bound on the number of producers PSync can support effectively in a highly dynamic environment.

SVS carries the dataset namespace directly in Sync Interests, making the size of Sync Interests likely to exceed network's MTU limit when a Sync group becomes large in size. A promising direction to scale SVS is utilizing SVS's unique feature: every entry in the state vector is independent of others, making it possible for a Sync Interest to carry as many, or as few, [producer-name, seq#]-pairs as needed. This removes any specific limit on the group size that SVS can support, and turns the question to which entries to put into state vectors in each Sync Interest. Example choices for selecting partial state vectors include *n* randomly selected entries, with *n* being the number that fits network MTU limit, or combining random selection with those of the most recent data production. In a highly dynamic environment, in order to avoid a latecomer needing to collect multiple Sync Interests to learn the full dataset state, one could also provide "bootstrapping" support: a latecomer may pick one participant *P* listed in the first Sync interest it receives and fetch the entire namespace from *P*.

6.3 Multicast Routing Support

As we reasoned in §5.2, a Sync protocol needs to multicast Sync Interests to the group members. One perceived hurdle in rolling out IP multicast is the concern regarding its scalability, a similar concern might arise for NDN Interest multicast, challenging the viability of NDN Sync protocols in general. We believe that multicast routing scalability is beyond the scope of this SoK paper, and that the main hurdle impeding IP multicast rollout is not the scalability barrier, as evidenced by the relevant literature, standards, and vendor implementations [5, 7, 8, 21, 25, 55].

6.4 Use of the NDN Protocol

All Sync protocols discussed in this work comply with the NDN protocol specification and do not require protocol modifications. At first sight, however, some Sync protocol's use of NDN seems to not follow the original CCN/NDN proposal, as presented in 2009 [24]. In the following, we discuss two sync protocol design choices that might lead to confusion:
Interests do not solicit replies: NDN's well-known communication model is retrieving named content chunks using Interest-Data exchange. Some Sync protocols use Sync Interests that do not retrieve Data replies. These Sync Interests' carry the name of the dataset to synchronize and are forwarded using the multicast strategy to all participants interested in the dataset. When using replies, participants having different dataset states, might reply the same Interest with different state updates, as visualized in Fig. 1a. This behavior interferes with NDN's protocol design principles in two ways: i) multiple response Data packets with the same name carry different dataset state updates, resulting in non-unique data names; ii) multiple Data items sent in response to a single Interest interfere with flow balance. As a result, some Data packets get dropped as unsolicited. Considering these potential conflicts of multicast Sync Interest replies, not soliciting Sync Interest replies seems appropriate.
Signed Interests: The idea of Interest packets is to request a named content chunk, which by design should not disclose the Interest sender's identity. Some use-cases, e.g., the NFD forwarder configuration [36] require interest sender authentication. To allow for the authentication of Interest senders, the *Signed Interest* format [37] was introduced. We want to highlight that using interest signatures does not have to disclose the signer's identity to third parties. The signature's key locator field identifies the used signing key; necessary information when verifying the signature. When carrying the signing key name, the key locator disclosed the signers identity and thereby removes the Interest sender's anonymity. Following the example of [38], carrying the public key fingerprint [16] as key locator enables signature verification for those entities that already know the signers key, while not disclosing the signer's identity to others.

6.5 ALF, NDN Sync, and Message Queues

The concept of Application-Level Framing (ALF) [9] allows applications to define semantically meaningful Application Data Units (ADU). These self-contained ADUs may become fragmented for delivery over the network by lower protocol layers, but need to be reassembled and kept intact for being processed by applications. Both, NDN Sync protocols and TCP/IP-based message queueing frameworks operate based on the concept of ADU. Although having this commonality, the two protocol families are entirely different. NDN Sync protocols synchronize a dataset's state among distributed applications. Each dataset entry corresponds to an ADU. Thereby, NDN Sync is taking the role of signaling the existence of ADUs.

In distributed applications, requirements for data delivery vary in different aspects. Among them, different data consumption patterns, latency, and reliability requirements. A Sync protocol can not fulfill all application demands for data retrieval, which is why higher-layer protocols build on top of NDN Sync, as proposed in [43], realize data delivery. Similarly are TCP/IP-based message queuing frameworks, such as MQTT, ZeroMQ, RabbitMQ, or Kafka [20, 34, 35, 50], build on top of TCP/IP transport protocols. TCP/IP's transport protocols transport bitstreams between two endpoints, and hence, are not directly suited for the exchange of ADUs in distributed applications. Message queuing frameworks patch application semantics in terms of ADUs on communication channels. This enables the use of high-level communication patterns, such as publish-subscribe, request-reply, or communication pipelines over host-centric TCP/IP transport protocols.

To summarize: while both NDN Sync and TCP/IP-based message queuing frameworks have ADUs as the granularity of data, their position in the protocol stack differs. NDN Sync operates on the transport layer; since TCP/IP lacks semantic application information in the protocol header, message queuing frameworks need to operate on the application layer. Due to the use of semantic names in NDN throughout the entire protocol stack, higher-layer protocols built on top of Sync may benefit from network-level features, such as data multicast, in-network caching, and name-based forwarding. Frameworks build on top of a TCP/IP transport, however, suffer from TCP's endpoint-oriented packet header and have to deploy additional components, such as MQTT's broker nodes, in the network when requiring network support.

7 Related Work

This section briefly surveys the research areas of file synchronization, distributed consensus protocols, and reliable multicast, which are closely related to distributed dataset synchronization.

File and Folder Synchronization: This area of research has a long history. The rsync algorithm [1] synchronizes files and folders between two nodes. Setting up a central server extends file sharing to multiple participants, as provided by public cloud providers [12, 17] as well as private clouds [41, 49]. Peer-to-peer protocols [10, 26] move file sharing towards decentralized architectures by creating application layer overlays. All the above mentioned systems synchronize files via *pairwise* host-to-host communication; multicast, if used at all, is implemented at application layer. NDN Sync moves the synchronization process among multiple parties down to transport layer and increases network efficiency by fully utilizing NDN's built-in multicast data delivery and in-network caching. NDN's security support further provides integrity and authenticity of files even when not retrieved from origin publishers or trusted servers.

Distributed Consensus Protocols: One of the most well-known consensus algorithms is Paxos [27] which performs a two-phase commit process among the participants of distributed systems to achieve consistency. A centralized controller can orchestrate all participants in the process 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 $n \times n$ communications, with n = number of participants, and can be achieved by setting up $n \times n$ TLS channels, a costly solution. The fundamental issue is that the application layer's focus on data cannot be effectively mapped to the network level service of connecting nodes. While implementations of Paxos such as Multi-Ring Paxos [29] can leverage IP multicast [11], achieving multicast in today's Internet is an inherently challenging exercise as explained next.

Reliable Multicast: The concept of IP multicast [11] was introduced around the same time as Stonebraker predicted the need for distributed DB systems [42]. Different from the latter, however, issues including multicast congestion control and reliable multicast delivery hampered its wide deployment [44]. Many reliable multicast solutions [40], including SRM [14], are based on the concept of Application Level Framing (ALF) [9], 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 may be multicast only to predefined groups of nodes, and this constraint makes achieving effecient dissemination of data to only the processes that require it very difficult. This mismatch also makes efficient loss recovery difficult, as reliable multicast desires retransmissions of specific missing ADUs from nearby members[28], while multicast-capable 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 above-mentioned mismatch between network and upper layers. Sync allows organizing data transport according to application needs, which renders Sync as a framework for efficient reliable multicast solutions via NDN. 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

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

We believe we are still in an early stage in the Sync protocol development, but we are confident in the important role a well-designed Sync protocol will play in future distributed applications developed 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 resulting from this work is available under open-source licensing [6].

References

[1] 1998. *The rsync algorithm.* Technical Report.

[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, 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.*

[5] Rahul Aggarwal and Eric C. Rosen. 2012. Multicast in MPLS/BGP IP VPNs. RFC 6513. https://doi.org/10.17487/RFC6513

[6] Anonymous ICN Authors. 2021. Repoducibility Set for "The Evolution of Distributed Dataset Synchronization Solutions in NDN". https://link-added-in-cam.ready

[7] T. Bates, R. Chandra, D. Katz, and Y. Rekhter. 2007. *RFC4760: Multiprotocol Extensions for BGP-4.* Technical Report.

[8] Cisco Systems. 2004. Cisco IOS IPv4 Multicast Technologies. https://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/ip-multicast/product_data_sheet0900aecd8031080b.pdf accessed: 2021-05-17.

[9] 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

[10] Bram Cohen. 2008. The BitTorrent Protocol Specification. https://web.archive.org/web/20140208002821/http://bittorrent.org/beps/bep_0003.html accessed: 2021-05-17.

[11] Steve Deering. 1989. *RFC1112: Host extensions for IP multicasting.* Technical Report.

[12] Dropbox, Inc. 2021. Dropbox Homepage. https://www.dropbox.com/ accessed: 2021-05-17.

[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] Peter Gusev and Jeff Burke. 2015. NDN-RTC: Real-Time Videoconferencing over Named Data Networking. In *Proceedings of the 2Nd International Conference on Information-Centric Networking (ICN '15).* ACM, New York, NY, USA.

[19] GÉANT project. 2018. GÉANT topology map. https://www.geant.org/Networks/Pan-European_network/Pages/GEANT_topology_map.aspx accessed: 2021-05-10.

[20] Pieter Hintjens. 2013. *ZeroMQ: messaging for many applications.* O'Reilly Media, Inc.

[21] Hugh Holbrook and Storigen Systems. 2006. Source-Specific Multicast for IP. RFC 4607. https://doi.org/10.17487/RFC4607

[22] 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).

[23] 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

[24] 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

[25] Juniper Networks, Inc. 2021. Multicast Protocols User Guide. https://www.juniper.net/documentation/us/en/software/junos/multicast/index.html accessed: 2021-05-17.

[26] Patrick Kirk. 2003. Gnutella – A Protocol for a Revolution. http://rfc-gnutella.sourceforge.net/

[27] 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

[28] 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

[29] 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

[30] Mini-NDN Authors. 2021. Mini-NDN: A Mininet-based NDN emulator. minindn.memphis.edu/ accessed: 2021-05-10.

[31] Philipp Moll, Varun Patil, Nishant Sabharwal, and Lixia Zhang. 2021. *A Brief Introduction to State Vector Sync.* Technical Report NDN-0073. NDN.

[32] Philipp Moll, Varun Patil, and Lixia Zhang. 2021. Resilient Brokerless Publish-Subscribe over NDN. In *Under review for: Military Communications Conference.* IEEE.

[33] 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.

[34] mqtt.org. 2020. MQTT: The Standard for IoT Messaging. https://mqtt.org/ accessed: 2021-07-19.

[35] 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.

[36] NDN Project team. 2018. NFD Management protocol. (2018). https://redmine.named-data.net/projects/nfd/wiki/Management accessed: 2021-07-29.

[37] 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.

[38] 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.

[39] 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

[40] Katia Obraczka. 1998. Multicast transport protocols: a survey and taxonomy. *IEEE Communications Magazine* 36, January (1998), 94–102.

[41] ownCloud GmbH. 2021. ownCloud – share files and folders, easy and secure. https://owncloud.com/ accessed: 2021-05-17.

[42] 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

[43] Varun Patil, Philipp Moll, and Lixia Zhang. 2021. Higher-level transport APIs on top of NDN Sync. In *Under review for: 8th ACM Conference on Information-Centric Networking*. ACM.

[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] Wentao Shang, Alexander Afanasyev, and Lixia Zhang. 2018. *VectorSync: Distributed Dataset Synchronization over Named Data Networking*. Technical Report NDN-0056. NDN.

[49] 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.

[50] VMware Inc. 2021. Messaging that just works – RabbitMQ. https://www.rabbitmq.com/ accessed: 2021-07-19.

[51] 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)*.

[52] 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.

[53] 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

[54] 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)*.

[55] Zhaohui Zhang, Lenny Giuliano, Eric C. Rosen, Karthik Subramanian, and Dante J. Pacella. 2015. Global Table Multicast with BGP Multicast VPN (BGP-MVPN) Procedures. *RFC* 7716 (2015), 1–22. https://doi.org/10.17487/RFC7716

[56] 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

[57] Zhenkai Zhu, Sen Wang, Xu Yang, Van Jacobson, and Lixia Zhang. 2011. ACT: audio conference tool over named data networking. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*.