Scaling State Vector Sync

Varun Patil, Sichen Song, Guorui Xiao, Lixia Zhang
{varunpatil,songsichen,grxiao,lixia}@cs.ucla.edu
UCLA
Los Angeles, USA

ABSTRACT

State Vector Sync (SVS) is a Distributed Dataset Synchronization (Sync) protocol designed to support distributed applications running over NDN. SVS encodes raw dataset state in its messages to achieve resilient synchronization with low latency. As a result, the SVS message size grows linearly with the number of data producers in the same communication group, raising concerns about its scalability. This poster proposes a solution to improve SVS's scalability through the use of partial state vectors (p-SVS), and presents the results from our preliminary evaluation. Our results show that p-SVS has similar performance to vanilla SVS with improved scalability.

CCS CONCEPTS

• **Networks → Network protocol design**; **Transport protocols**; *Network performance evaluation.*

KEYWORDS

Named Data Networking, Distributed Dataset Synchronization, NDN Transport, State Vector Sync, Scaling

ACM Reference Format:
Varun Patil, Sichen Song, Guorui Xiao, Lixia Zhang. 2022. Scaling State Vector Sync. In *ACM ICN 2022 Demos and Posters (ICN '22), September 19–21, 2022, Osaka, Japan.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3517212.3559485

1 OVERVIEW

Distributed Dataset Synchronization protocols provide the transport service in the Named Data Networking (NDN) [5] architecture. State Vector Sync (SVS) [2] is one of the newer Sync designs, which uses raw state information directly for encoding. In this poster we address the scalability of SVS as the number of producers increases.

SVS provides *data namespace* synchronization[1] among all participants in a *Sync group*. Each data producer in the group uses sequential naming to identify all data pieces it produces; it increases this local sequence number every time it generates a new piece of data. SVS propagates the changes in sequence numbers at all producers to all participants in the group. Thus, the group's dataset state can be represented by a list of [producer, seq#] tuples. To achieve synchronization, participants encode this State Vector in the name of a single *Sync Interest* and multicast this Interest to all participants in the Sync group.

One unique aspect in the SVS design is letting Sync Interests carry the state vector *directly*, instead of a highly compressed form as some other NDN Sync protocols do, that require additional state to decode [3]. Therefore, a single Sync Interest I_S informs its receiver R the latest dataset state of its sender S, independent from how many previous Sync Interests, either from S or anyone else, that R may have missed. Upon reception of I_S, R compares its local state vector with that carried in I_S: if the latter has new information, R updates its local state vector; or if R notices S falling behind, it will multicast another Sync Interest containing its state vector. To prevent multiple receivers of I_S from reacting simultaneously, R sets a random wait timer before sending its Sync Interest, and cancels its transmission if it receives another Sync Interest with more or the same information compared to its own before the timer expires. This is referred to as Sync Interest suppression in the SVS design.

SVS achieves resilient synchronization with low delay by carrying the raw state vector in Sync Interests. Unfortunately, doing so results in the size of a Sync Interest growing linearly with the number of producers in the group, therefore the number of producers would be upper-bounded by the size of network MTU. However, we note that each tuple [producer, seq#] has no dependency on any others in the state vector, therefore one may put any *partial set* of the vector in a Sync Interest. In this work, we investigate the use of partial state vector to improve the scalability of SVS.

2 PARTIAL STATE VECTOR

By encoding a subset of the state vector in the Sync Interest, SVS can synchronize a Sync group of any size. However, this scalability brings a potential cost of increased Sync latency, because a receiver needs to collect multiple Sync Interests to learn the latest dataset state of the group. Such a design converts the problem of SVS scalability to the question of how to choose an ideal subset of state vector to send in next Sync Interest. On careful inspection, we recognize two conflicting goals in the creation of a Sync Interest carrying a partial state vector.

(1) We note that in real distributed applications, not all producers may actively produce data at all times. Therefore, one may only include in the Sync Interest the tuples of producers that have produced data most recently, which would increase efficiency since the Sync Interests carry less redundant information. We refer to this approach as the ***Recent*** strategy.
(2) At the same time, we must take into account the possibility of packet losses and network partitions, which may cause some participants to miss some Sync Interests. As a result, if Sync Interests only carry information for the most recently changed producers, the state of some producers will not be propagated

[1]Synchronization of data can be achieved using higher level APIs [4]

ICN '22, September 19–21, 2022, Osaka, Japan
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9257-0/22/09.
https://doi.org/10.1145/3517212.3559485

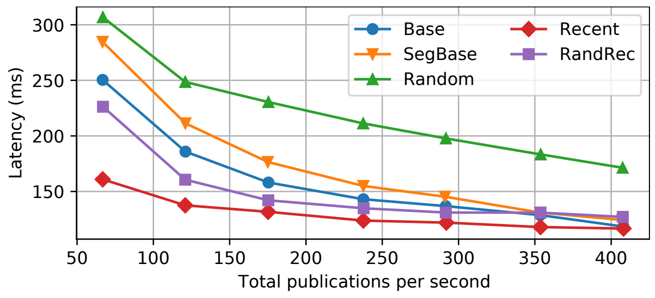

Figure 1: Latency Comparison

to all participants until their next update. In other words, the *Recent* strategy does not provide any time-bound on eventual consistency. One solution to this problem is to include the states of a random subset of producers in every Sync Interest, thus providing a statistical guarantee that each producer's state will be periodically included in a Sync Interest. We refer to this as the ***Random*** strategy.

As a middle ground, we also devise a third strategy, by including the states of both a subset of recently updated and a subset of randomly selected producers in the Sync Interests, which we call the ***Random-Recent*** (*RandRec*) strategy.

It must be noted here that the aforementioned list of competing factors for creating a partial state vector is by no means exhaustive, and serves only as a starting point for developing more strategies. We also note that when using partial state vectors, to let a newly joined participant quickly catch up the latest group dataset state, we need to design a bootstrapping mechanism that allows a new participant to fetch the complete dataset state as Data packets.

3 EVALUATION

We conducted preliminary comparison of the three aforementioned partial state vector strategies, together with vanilla SVS (a Sync Interest carrying the complete state vector) and segmented SVS (segmenting the complete state vector into multiple Sync Interests). Using ndnSIM [1], we set up a grid topology of 8×8 nodes, with all the 64 nodes participating in a single SVS Sync group. Each node produces new pieces of data every 1sec, while a few nodes are randomly selected to produce new data every 100ms. We vary the aggregate group publication rate by varying the number of the *fast* producers from 0 to 38. Each network link has 10ms propagation latency and a 50% packet loss rate[2]. All the nodes start data production at the start of each simulation which ran for 10sec.

We use the 95%tile latency of synchronization and the total network traffic as the metrics for the comparison; 95%tile latency is defined as the delay for 95% of participants to learn the latest data production. We generate partial state vectors whose size is 30% of the entire state vector (simulating an artificially small MTU). As a baseline, we also evaluate (1) Vanilla SVS (***Base***), and (2) Vanilla SVS with segmented Sync Interests, where the complete state vector is segmented into 4 Sync Interests (***SegBase***).

[2]Because our grid topology creates rich connectivity and multicast Sync Interests are sent along all links, we use this artificially high loss rate to tear apart each strategy's performance.

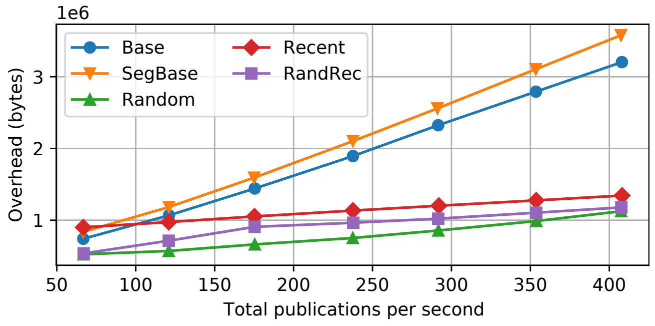

Figure 2: Overhead Comparison

Fig. 1 shows a comparison of the dataset state synchronization latencies among different strategies. Note that, overall, the latency goes down as data production rate goes up, because a node sends a Sync Interest as soon as it produces a new piece of data, thus higher data rate leads to more frequent Sync Interests.[3] We also note that segmented Sync Interests perform marginally worse than base SVS, since it has lower probability of informing latest data production than base SVS. Performance of the random strategy worsens with increasing frequency of publication, because the randomly selected product tuples are more likely to miss some of the latest data production. The latency of Random-Recent strategy, on the other hand, follows closely to that of base SVS, and is even lower at low publication rate. The Recent strategy has shortest latency at all publication rates; however this strategy may not do well in the presence of network partitions, which we did not simulate.

Fig. 2 compares the total number of bytes transmitted across all links in the network. The Sync Interest sizes of the three partial state vector strategies are approximately 30% of the base SVS Sync Interests, thus they all have a lower overhead than the base SVS by a similar factor. The low overhead also indicates that Sync Interest suppression continues to perform well when using partial state vectors, since we used the same suppression mechanism as base SVS for all strategies. Together, these results demonstrate the potential of using partial state vectors to achieve similar latency with improved scalability and lower overhead, as compared to vanilla SVS.

4 SUMMARY AND FUTURE WORK

Although our preliminary results suggest that partial state vector represents a a promising direction to address SVS scalability problem, our simulation setting is overly simplified. Further evaluation efforts need to use more realistic network topology and workload conditions, and further reduced partial state vector size, e.g. 10% or lower, to see where the three proposed strategies in shortening state vector can offer similar results as reported in this poster. More innovative approaches may also exist that can achieve both high scalability and low latency.

REFERENCES

[1] Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. 2017. On the Evolution of ndnSIM: an Open-Source Simulator for NDN Experimentation. *ACM Computer Communication Review* (July 2017).

[2] Philipp Moll, Varun Patil, Nishant Sabharwal, and Lixia Zhang. 2021. *A Brief Introduction to State Vector Sync.* Technical Report NDN-0073, Revision 2. Named Data Networking. 1–4 pages.

[3]This same reason lets ***Recent*** have lowest latency.

[3] Philipp Moll, Varun Patil, Lan Wang, and Lixia Zhang. 2022. SoK: The Evolution of Distributed Dataset Synchronization Solutions in NDN. https://doi.org/10.1145/3517212.3558092 To appear in ACM Information Centric Networking Conference, 2022.

[4] Varun Patil, Philipp Moll, and Lixia Zhang. 2021. Supporting Pub/Sub over NDN Sync. In *Proceedings of the 8th ACM Conference on Information-Centric Networking* (Paris, France) *(ICN '21)*. Association for Computing Machinery, New York, NY, USA, 133–135. https://doi.org/10.1145/3460417.3483376

[5] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *ACM SIGCOMM Computer Communication Review (CCR)* 44, 3 (July 2014), 66–73.