VectorSync: Distributed Dataset Synchronization over Named Data Networking

Wentao Shang
UCLA
wentao@cs.ucla.edu

Alexander Afanasyev
Florida International University
aa@cs.fiu.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

Distributed dataset synchronization (sync for short) provides an important abstraction for multi-party data-centric communication in the Named Data Networking (NDN) architecture. Since the beginning of the NDN project, several sync protocols have been developed, each made its own design choices that cause inefficiency under various conditions. Furthermore, none of them provides group membership management, making it difficult to remove departed nodes from the protocol state maintained at each node. This poster presents VectorSync, a new NDN sync protocol that is built upon the lessons learned so far, provides group membership management, and improves the efficiency of dataset synchronization.

CCS CONCEPTS

• **Networks → Network protocol design**;

KEYWORDS

Named Data Networking, Distributed dataset synchronization

ACM Reference Format:
Wentao Shang, Alexander Afanasyev, and Lixia Zhang. 2017. VectorSync: Distributed Dataset Synchronization over Named Data Networking. In *Proceedings of ICN '17, Berlin, Germany, September 26–28, 2017,* 2 pages. https://doi.org/10.1145/3125719.3132106

1 INTRODUCTION

The proposed Named Data Networking (NDN) architecture [5] replaces the host-oriented communication model in TCP/IP with a data-centric one. At its network layer, NDN employs a basic *Interest-Data exchange* primitive to provide best-effort retrieval of individual named and secured data objects. While this simple yet powerful primitive significantly narrows the semantic gap between the application layer and the network layer, it remains cumbersome to use directly by applications. Early on in the NDN research effort, we identified *sync* as an important abstraction to simplify the development of distributed applications over NDN. Essentially, one may view sync as playing a *transport* layer role in the NDN architecture that bridges the gap between the functionality required by applications and the delivery semantics offered by network primitives, similar to the role TCP plays in bridging the gap between applications' need for reliable data delivery and IP's datagram service.

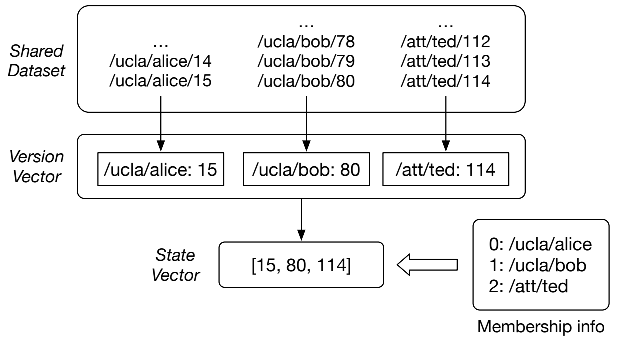

Figure 1: Representing the dataset with a state vector

Several sync protocols have been developed for the NDN architecture, each having its own design trade-offs that affect the performance under various conditions (see Table 1). In this poster we present the design and implementation of VectorSync, a new NDN sync protocol that integrates a *leader-based group membership management* to synchronize the view of the current group membership among the active participants while nodes join and leave the group over time. Maintaining the group membership information at the sync layer facilitates data authentication and access control and improves the sync protocol efficiency by removing the departed nodes from the protocol state.

2 PROTOCOL DESIGN

The design of VectorSync consists of two interdependent components: a *dataset state synchronization* mechanism for maintaining a consistent state of the shared dataset, and a *group membership synchronization* mechanism for maintaining a consistent view of the current group members.

Similar to ChronoSync [6], VectorSync adopts a sequential data naming convention of each node naming its data under a unique publishing prefix with continuous sequence numbers. This enables VectorSync to represent the state of the shared dataset efficiently using a version vector [2] (called *state vector*) that contains the latest data sequence number from each producer in the group. The order of the producers in the version vector follows the group membership list, where the producers are ordered canonically based on their prefixes. Therefore, the producers' data prefixes can be omitted from the state vector, minimizing the cost of transmitting the full vector over the network. Figure 1 illustrates an example of the state vector representing the namespace of a dataset with three producers.

Table 1: Design comparison between VectorSync and existing sync protocols

	Sync state representation	Interest frequency	Factors affecting Interest size	Min data dissemination delay
CCNx Sync [3]	Name tree (large)	Periodic	Node hash	Depending on Interest period + tree walk
iSync [1]	IBF (large)	Periodic	IBF digest	Depending on Interest period + 3.5 RTT
ChronoSync [6]	"prefix:seq#" list (small)	Long-lived Interest	State digest (with exclude filter)	1.5 RTT (+ additional RTT to fetch simultaneous data)
VectorSync	State vector (small)	One per data (with heartbeat)	View ID + data name	1.5 RTT

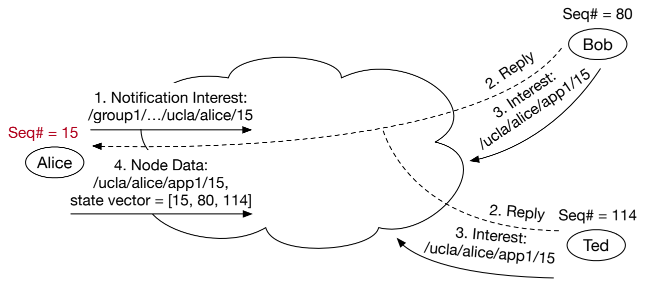
Seq# = 80
Bob
2. Reply
3. Interest: /ucla/alice/app1/15
1. Notification Interest: /group1/.../ucla/alice/15
Seq# = 15
Alice
4. Node Data: /ucla/alice/app1/15, state vector = [15, 80, 114]
2. Reply
Seq# = 114
3. Interest: /ucla/alice/app1/15
Ted

Figure 2: Publishing new data in the group

When a new data item is published, VectorSync uses a notification Interest to announce the name of the new data so that others can fetch the data immediately upon receiving the notification. The data carries the state vector of the producer at the time of the data's production; this full vector allows each receiving node to detect and reconcile inconsistency caused by various factors such as packet loss and network partition. Upon receiving the data, a node updates its local vector with the entry-wise maximum of the local and received vectors. An entry in the received vector with a higher sequence number than in the local one indicates missing data in the shared dataset. Figure 2 shows an example of a new data production and dissemination process in a group of three parties.

VectorSync utilizes a leader-based protocol to maintain a consistent *view* of the group membership among the participants. Each view is identified by a monotonically increasing view number and the leader's data prefix (to disambiguate multiple views with the same number during group partition). To maintain its membership, a node periodically publishes heartbeat packets in the shared dataset to assert its existence. The node with the highest-ordered prefix among the existing nodes is selected as the leader and monitors the heartbeats of each group member. When a node joins or leaves the group, the leader increments the view number and publishes its knowledge of the group membership in a Data packet (called *ViewInfo* packet) under the name "`/[group-prefix]/vinfo/[view-number]/[leader-prefix]`".

A notification Interest name carries the view identifier (i.e., leader prefix plus view number) of the producer's current view. Upon receiving a notification Interest with a higher view number, a node will fetch the corresponding ViewInfo and adjust the state vector by adding and/or removing the entries according to the new membership list. If multiple views (i.e., subgroups) are created by different leaders due to network partition, the leader with the highest-ordered prefix is responsible for merging those views and creating a new view that contains the members from all previous views.

VectorSync requires all participants to obtain a public key certificate from the application-defined trust anchor before joining the sync group. After verifying the certificates, the leader may put the members' public keys together with their data prefixes in the ViewInfo packet, making a certificate bundle for the current view. The group members can directly use the public keys in the ViewInfo packet to authenticate the data published in the shared dataset. To achieve access control, the leader may periodically generate a symmetric data encryption key and distribute the key to every node on the membership list [4].

3 CONCLUSION AND FUTURE WORK

Leveraging the lessons learned from our previous efforts, VectorSync represents our latest result from pursuing an effective and efficient sync protocol. It utilizes version vectors to reconcile the state of the shared dataset and supports group membership management. Table 1 compares VectorSync with a few previous sync protocols on several key design aspects. Our plan for next step is to conduct extensive evaluation of VectorSync through large-scale simulations and to integrate VectorSync with real-world applications to demonstrate its utility.

ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation under awards CNS-1345318, CNS-1629922, and CNS-1719403.

REFERENCES

[1] Wenliang Fu, H. Ben Abraham, and P. Crowley. 2015. Synchronizing Namespaces with Invertible Bloom Filters. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 123–134.
[2] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. 1983. Detection of Mutual Inconsistency in Distributed Systems. *IEEE Transactions on Software Engineering* SE-9, 3 (1983).
[3] ProjectCCNx. 2012. CCNx Synchronization Protocol. CCNx 0.8.2 documentation. (2012).
[4] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. 2016. *Name-Based Access Control*. Technical Report NDN-0034, Revision 2. NDN Project.
[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 Computer Communication Review* 44, 3 (July 2014).
[6] Zhenkai Zhu and Alexander Afanasyev. 2013. Let's ChronoSync: Decentralized Dataset State Synchronization in Named Data Networking. In *Proc. of IEEE International Conference on Network Protocols (ICNP)*.