Mnemosyne: An Immutable Distributed Logging Framework over Named Data Networking

Siqi Liu
siqi.liu@ucla.edu
UCLA Computer Science

Philipp Moll
mollph@cs.ucla.edu
UCLA Computer Science

Lixia Zhang
lixia@cs.ucla.edu
UCLA Computer Science

ABSTRACT

This poster describes the design of Mnemosyne, a distributed logger running over Named Data Networking. Mnemosyne utilizes proof of authenticity instead of proof of work. It assures immutability of logged events by interlocking all event records in a DAG mesh. By using a distributed design, Mnemosyne provides both a high logging throughput and system resiliency in face of network component failures.

CCS CONCEPTS

• **Networks** → *Security protocols*; • **Information systems** → *Information storage systems.*

KEYWORDS

Distributed Logging, Named Data Networking, Distributed Ledger

ACM Reference Format:
Siqi Liu, Philipp Moll, and Lixia Zhang. 2021. Mnemosyne: An Immutable Distributed Logging Framework over Named Data Networking. In *8th ACM Conference on Information-Centric Networking (ICN '21), September 22–24, 2021, Paris, France.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3460417.3483375

1 INTRODUCTION

As a proposed new Internet architecture, Named Data Networking (NDN) [7] provides secure networking primitives for building distributed data-centric applications. Examples of such applications include the distributed Data repository Hydra [6], distributed online game architecture [4], and distributed computing designs [2].

Distributed applications share a common need with single-server applications: event logging for system monitoring. However, a logger for distributed applications brings new requirements. Taking the distributed repository Hydra as an example, the logging over the network must meet the following requirements: i) the logger system should be resilient against node failure and network instability; ii) only authorized Hydra application instances are allowed to submit log events; and iii) no entity, including Hydra application instances or loggers, is allowed to remove or tamper with existing log events.

This work is licensed under a Creative Commons Attribution-ShareAlike International 4.0 License.
ICN '21, September 22–24, 2021, Paris, France
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8460-5/21/09.
https://doi.org/10.1145/3460417.3483375

This paper introduces the design of Mnemosyne, a distributed immutable logger over NDN. Mnemosyne supports the aforementioned requirements from Hydra, which we hope are generally applicable to logger designs for other distributed applications. In addition to authenticating all entities that insert log events, Mnemosyne ensures immutability by inserting log events into a DAG-based distributed ledger, taking after DLedger [8]. In this DAG, each log event is indirectly signed by subsequent log events that hold digests of previous log events. This DAG structure makes changing past log events infeasible while avoiding performance bottlenecks by allowing parallel logging.

2 THE DESIGN OF MNEMOSYNE

The Mnemosyne design takes the following steps to meet the requirements mentioned earlier. First, to ensure resiliency, Mnemosyne comprises a federation of loggers, and all loggers have the same privileges and responsibility for keeping Mnemosyne functioning. All the loggers communicate via NDN Sync protocol State Vector Sync (SVS) [3] which assures reliable dataset synchronization even under adverse conditions.

Second, the assurance of authenticity in all Mnemosyne operations is build on the NDN security framework [9], where each entity in the system, including the loggers, goes through a security bootstrapping process to obtain the Mnemosyne system's trust anchor, its own certificate, and a set of security policies that defines the criteria of legit users and their allowed operations.

Third, Mnemosyne assures the immutability of log events by borrowing the idea of linking individual log events together using cryptographic digests, as done by blockchain approaches [1]. However, instead of arranging log events as a chain, Mnemosyne adopts a directed acyclic graph (DAG) approach [5], which is used in DLedger [8]. The DAG approach enables merging partitions and allowing parallel insertion to support system resiliency and high throughput.

Mnemosyne follows a layered design, as illustrated in Figure 1. Loggers accept log events by listening to an SVS-based publish-subscribe interface [3] that reliably transports log events from an application instance to all loggers (indicated by the green boxes in the figure). Log events published by the application are encoded into NDN data packets, signed by the application entity's key (shown as a green lock in Fig. 1). Loggers authenticate each log event, then encapsulate the entire log event including the application signature in a record and insert the record into the DAG. Digests in the DAG ensure that no party can modify Hydra's events, and that events cannot be removed once inserted into the DAG. The conversion of log events to DAG records is covered in Section 3.

To mitigate the impact of node failure or network partition, all loggers receive all log events by running SVS.Keeping the log

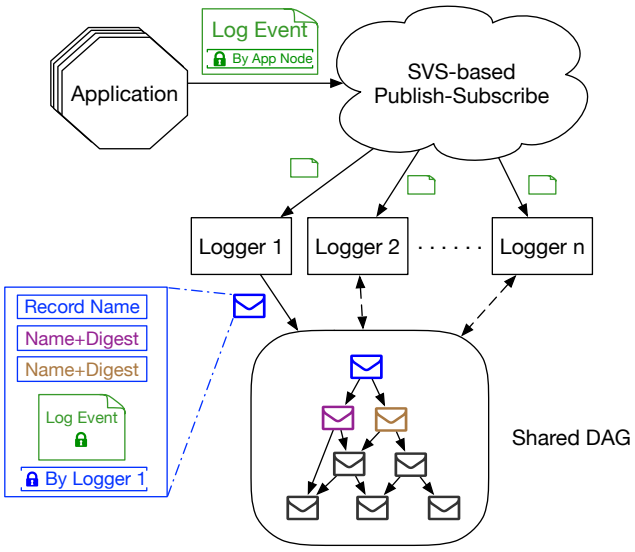

Figure 1: Overview of Mnemosyne's Design

event persistent, however, requires only a single logger to create a DAG record. Section 4 discusses a backoff mechanism to avoid storage and communication overhead resulting from redundant record generation.

3 IMMUTABLE LOGGING USING A DAG

A log event is inserted into the DAG mesh by converting it to a record, as indicated by the blue box in Figure 1. A record is a regular NDN data packet having a unique record name and encapsulates the application-generated log event. In addition, a record contains at least two[1] links to previous records in the DAG. These links are implemented as the names of the corresponding DAG records including their implicit digests (visualized as purple and orange elements). Also, each record contains a signature created by the corresponding logger as proof of authenticity, proving the record is generated by a legitimate logger. Once created, DAG records are synchronized among all loggers using the NDN sync protocol SVS [3], while each logger keeps the entire DAG, and thereby all logged events, persistent.

The usage of a DAG-based distributed ledger avoids two limitations of a chain-based structure: i) a chain-based structure only allows additions of one new entry on top of the last entry. This prevents parallel insertion and represents a performance bottleneck. A DAG offers the flexibility of appending entries in parallel. ii) Network partition cuts a chain into multiple forks, which cannot be stitched together later. The same partition can cut a DAG into multiple sub-DAGs, which can merge automatically after the network partition heals, retaining all records from the partition.

To demonstrate the immutability of the DAG, we consider a modification of a logged event in the DAG, which changes the record's digest. This change invalidates the digests of all links pointing to this record, and thus, exposes the modification. To modify a log event without being exposed to others, all subsequent records are required to be recalculated. Besides the required computational effort, the recalculation requires the signing keys of the log event, the DAG record and any subsequent records that need to be restored. Therefore, the DAG structure ensures the immutability of the logged events.

[1]Mnemosyne can work with a configurable number ≥ 2 of links. For clarity, our visualization uses exactly two links for each record.

4 INSERTING EVENTS INTO THE DAG

To increase Mnemosyne's resiliency against node or network failure, log events from the application are delivered to all loggers. This redundant delivery, however, could lead to multiple loggers generating redundant records for the same log event. Although such redundant entries do not affect Mnemosyne's functionality, but they increase the processing and storage overhead. In the following, we describe a solution that can avoid redundant record generation, so that only one logger appends the record (blue envelope) to the DAG.

We develop a timed backoff algorithm that functions in the following way. Loggers determine an order for inserting DAG records based on the log event's digest. This insertion order assigns a waiting period to every logger. During this period, the logger continues receiving new DAG records and thereby detects whether a record for the new log event is created by another logger or not. If no such record was received by the end of the waiting period, the logger inserts a new DAG record for the log event. Letting all loggers retrieve a new DAG record R ensures that R will be inserted even when some loggers might fail, or fail to retrieve R, while continued communication among the loggers makes R being inserted once only in general.

This procedure creates a single DAG record for each log event under stable network conditions, which ensures a low processing and storage overhead resulting from redundant record creation. At the same time, active loggers try inserting the event if other loggers have failed, ensuring the log event is eventually added to the DAG.

5 DISCUSSION AND FUTURE WORK

Mnemosyne uses data structure and verification techniques similar to DAG-based distributed ledgers. Mnemosyne diverges from these distributed ledgers and proof of work approach for the following reasons: i) Mnemosyne is a permissioned ledger. This design allows Mnemosyne to utilize the NDN security design for identifying loggers and application entities, simplifying the record generation and verification. ii) Mnemosyne aims to log all authenticated records. Instead of focusing on double-spending prevention as in cryptocurrencies, Mnemosyne focuses on retaining all information. Therefore, instead of having the rate-limiting capability of proof of work, proof of authenticity allows any legitimate record to be inserted.

As the next step, we plan to quantitatively evaluate Mnemosyne's performance and overhead in large-scale networks, and to explore the applicability of Mnemosyne to other distributed applications that have similar logging needs.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under award 1719403, and by Operant Networks.

REFERENCES

[1] Muhammad Nasir Mumtaz Bhutta, Amir A. Khwaja, Adnan Nadeem, Hafiz Farooq Ahmad, Muhammad Khurram Khan, Moataz A. Hanif, Houbing Song, Majed Alshamari, and Yue Cao. 2021. A Survey on Blockchain Technology: Evolution, Architecture and Security. *IEEE Access* 9 (2021), 61048–61073. https://doi.org/10.1109/ACCESS.2021.3072849

[2] Daniel Meirovitch and Lixia Zhang. 2021. *NSC – Named Service Calls, or a Remote Procedure Call for NDN.* Technical Report NDN-0074, Revision 1. Named Data Networking. 1–10 pages.

[3] 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.

[4] Philipp Moll, Sebastian Theuermann, Natascha Rauscher, Hermann Hellwagner, and Jeff Burke. 2019. Inter-Server Game State Synchronization Using Named Data Networking. In *Proceedings of the 6th ACM Conference on Information-Centric Networking* (Macao, China) *(ICN '19)*. ACM. https://doi.org/10.1145/3357150.3357399

[5] Huma Pervez, Muhammad Muneeb, Muhammad Usama Irfan, and Irfan Ul Haq. 2018. A Comparative Analysis of DAG-Based Blockchain Architectures. In *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*. 27–34. https://doi.org/10.1109/ICOSST.2018.8632193

[6] The ndn-hydra authors. 2021. ndn-hydra. https://ndn-hydra.readthedocs.io/ accessed: 2021-07-23.

[7] 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.

[8] Zhiyi Zhang, Vishrant Vasavada, Xinyu Ma, and Lixia Zhang. 2019. DLedger: An IoT-Friendly Private Distributed Ledger System Based on DAG. *CoRR* abs/1902.09031 (2019). arXiv:1902.09031 http://arxiv.org/abs/1902.09031

[9] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. 2018. An Overview of Security Support in Named Data Networking. *IEEE Communications Magazine* 56, 11 (November 2018), 62–68. https://doi.org/10.1109/MCOM.2018.1701147