UNIVERSITY OF CALIFORNIA
Los Angeles

Private Distributed Ledger over Named Data Networking

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Vishrant Nitin Vasavada

2019

ABSTRACT OF THE THESIS

Private Distributed Ledger over Named Data Networking

by

Vishrant Nitin Vasavada
Master of Science in Computer Science
University of California, Los Angeles, 2019
Professor Lixia Zhang, Chair

In this thesis, we present a private distributed ledger system, DLedger, designed for wireless meshed Named Data Networking (NDN) protocol network. DLedger utilizes lightweight Proof-Of-Authentication as gating control mechanism combining data openness among the system peers with verifiable identity within the system. The lightweight nature of Proof-Of-Authentication makes it friendly for the ledger systems consisting of even the constrained Internet of Things (IoT) devices unlike "muscle show" approaches like Proof-Of-Work, Proof-Of-Space, etc. which are storage or computation intensive and combines data openness with anonymity (or pseudonymity). Moreover, different from the popular blockchain-based ledger systems, DLedger utilizes a Directed Acyclic Graph as a fundamental data structure so that its operations can tolerate network partitions. Built over NDN, DLedger truly leverages from its data-centric nature to facilitate data dissemination in peer-to-peer heterogenous IoT networks. We conclude the thesis by reasoning our design through simulation results and discussing a real-world use case.

The thesis of Vishrant Nitin Vasavada is approved.

Peter L. Reiher

George Varghese

Lixia Zhang, Committee Chair

University of California, Los Angeles

2019

To my parents, sister and uncle ...

the hidden strength

behind my every success.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Firstly, I would like to thank Professor Lixia Zhang for believing in me and giving me opportunities to work on a couple of amazing projects under the Named Data Networking (NDN) umbrella throughout my academic career at UCLA. The door to Professor Zhang's office was always open whenever I faced any trouble or had a question regarding any part of this work and beyond. Next, I'm grateful to Professor Peter L. Reiher who served as my academic advisor since my first day at UCLA and also introduced me to the research world by giving me interesting DDoS project to work on in collaboration with a team from the University of Oregon. I am thankful to Professor Reiher for advising me to study and explore the trending Bitcoin security problems that set an initial spark, eventually resulting in this thesis. This work wouldn't have been possible without having knowledge of existing cryptocurrency protocols and the security problems within. I am also grateful to Professor George Varghese who was a constant source of inspiration and motivation. He always advised on interdisciplinary and creative thinking which completely changed the way I think and correlate different pieces of knowledge together to find a solution to a certain problem. This thesis is a result of the initial realization of the distributed ledger as a distributed dataset synchronization problem in NDN and figuring out if NDN can help develop more efficient and secure distributed ledger technology than the existing.

I am also thankful to Randy King, CEO of Operant Networks Incorporation, for being an important contributor and supporter in this work. The initial version of this work, in fact, was designed and inspired by the real-world use case of Operant Networks Inc. as proposed by Randy under the U.S. Department of Energy funding. I would also like to thank Zhiyi Zhang of Internet Research Laboratory (IRL) group for being an amazing mentor and active contributor to this work. The design of the work proposed in this thesis wouldn't have been so robust if it weren't for Zhiyi to realize loopholes and security issues in the initial prototype. I would also like to thank Xinyu Ma of the IRL group for being another active contributor to this work, especially for his code debugging help and producing simulation results.

CHAPTER 1

Introduction

This work is inspired by and formerly designed for an experimental solar mesh network developed by Operant Networks Incorporated. This solar network composes of rooftop Operant's solar gateway devices. Each device communicates using LoRa wireless channel over Named Data Networking (NDN) [ZAB14] protocol with the Operant server. Each such device records customer energy consumption and sends it to the Operant server for accounting.

The customers of such business service providers as well as the other financial parties and partners involved have to blindly trust business service providers about the data record integrity since cloud servers are fully in business service provider's control. If a business service provider makes any errors or data records that are centrally stored gets corrupted, it may go unnoticed by the customers and other parties and sometimes even to the business service providers themselves who may then mistakenly charge customers more. There is little to no proof available regarding the existence of original data records. Also, such centralized control may even allow a malicious business provider to alter the data records for their personal gains. For example, solar network business operator can alter customer energy consumption data to falsely charge them more. Moreover, server outages can also make records temporarily unavailable to the customers and other parties involved. The integrity, authenticity, and availability of data records is key to business reputation and integrity. Hence, it is necessary for the business service provider to maintain consensus backed replication of data records not just between their own servers but also customer nodes (devices) and with servers of every other party involved. With data records replicated among all the entities and consensus established, it is increasingly hard for the errors or corruption to go unnoticed as well as leaves some form of surveillance on business service provider so that

they cannot cheat and alter the data. Replication, on the other hand, provides availability on demand. The customers and other financially involved parties, hence, would trust such a system, ultimately attracting more of them and profiting businesses.

Cryptocurrencies like Bitcoin [Nak08] have shown that financial transaction records can be stored securely using consensus backed replicated storage through a decentralized network of peers using a distributed ledger. However, Bitcoin implements the Proof-Of-Work (PoW) as a gating control mechanism which determines who can add new records into the ledger (successful miners). This hashcash based approach is computation intensive and is infeasible for constrained devices such as Operant's solar gateway device. Other approaches such as Proof-Of-Stake (PoS) and Proof-Of-Space (PSpace) are also "muscle show" by brute force which is unfit for ledger system consisting of resource-constrained IoT devices. Moreover, widely used Blockchain as a data structure for distributed ledger won't be helpful here as it doesn't have support for network partitions. A partitioned and independent Blockchain cannot be reintegrated into the main chain, thus discarding old records from consensus backed replicated storage.

To address these problems, we designed DLedger, a private distributed ledger over NDN protocol, that uses Proof of Authentication (PoA) as a gating control mechanism which is lightweight and efficient even for the constrained Internet of Things (IoT) devices. The Directed Acyclic Graph (DAG) is used as a fundamental data structure allowing for simpler ledger integration after a network partition as the forked chains or graphs can simply be attached to the main DAG. Moreover, NDN's data-centric paradigm and built-in security in protocol makes record distribution and ledger synchronization efficient as well as improves security in DLedger. Since any data is fetched from the network as a whole rather than from a specific location/node in data-centric networks, DLedger, coupled with its lightweight PoA and smart storage solutions, removes the concept of a full node and light node present in traditional distributed ledger systems today over TCP/IP architecture and treats all nodes equally. This solves a major security loophole in existing distributed ledger systems where a light node is fully dependent on a specific full node for retrieving and relaying data, exposing it to malicious behavior of full nodes or DNS hijack attacks on the full nodes.

We conclude this thesis by discussing in detail a real-world use case of this DLedger in Operant's solar network. In fact, DLedger design discussed in this thesis is improved from proof-of-concept prototype [VK18] designed and developed for Operant Networks Incorporation.

The rest of this thesis is organized as follows. Chapter 2 gives an overview of the NDN architecture and discusses how distributed ledger was realized as a distributed dataset synchronization and gives a brief overview of the existing sync protocols in NDN that could be potentially used for ledger synchronization or to leverage ideas. We also introduce IOTA's Tangle [Pop18], a distributed ledger technology built for IoT devices. In Chapter 3 we describe the design of DLedger and present reasoning of our design choices. This chapter explains the Proof of Authentication and the underlying sync protocol in DLedger. We also discuss necessary verification procedures to be followed by each peer in the network whenever they receive or add a new record into the system to ensure system security and integrity. Chapter 4 discusses the motivation behind using distributed ledger in Operant Networks' use case. In Chapter 5, we identify related works. Finally, Chapter 6 concludes this thesis.

CHAPTER 2

Background

In this chapter, we give an overview of Named Data Networking (NDN) architecture and discuss the realization of the distributed ledger as a distributed dataset synchronization (sync) in NDN. We also introduce IOTA's Tangle.

2.1 Named Data Networking (NDN)

Named Data Networking (NDN), a recently proposed Internet architecture, makes the named data a first-class entity in the network architecture. It shifts the communication paradigm in today's TCP/IP architecture from being *host-centric* push model where data is pushed from one end host to another to *data-centric* pull model where data is fetched from the network via a request/response exchange. A request called an *Interest* packet in NDN, bears the name of data piece consumers would like to fetch. On the other hand, a response, called a *Data* packet, carries the actual data content (figure 2.1). It is important to note that none of these packets contain any IP address.

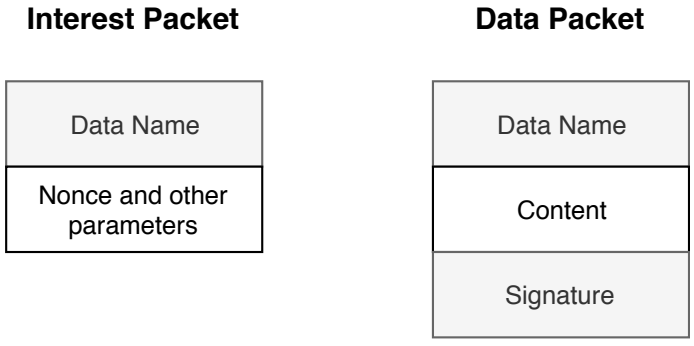


Figure 2.1: Interest and Data packets

As noted in figure 2.1, each NDN Data packet contains a signature field to ensure the integrity and authenticity of the data. The sensitive content in Data packet can also be encrypted if needed. NDN thus secures the data directly and places security building block right at the narrow waist as shown in figure 2.2.

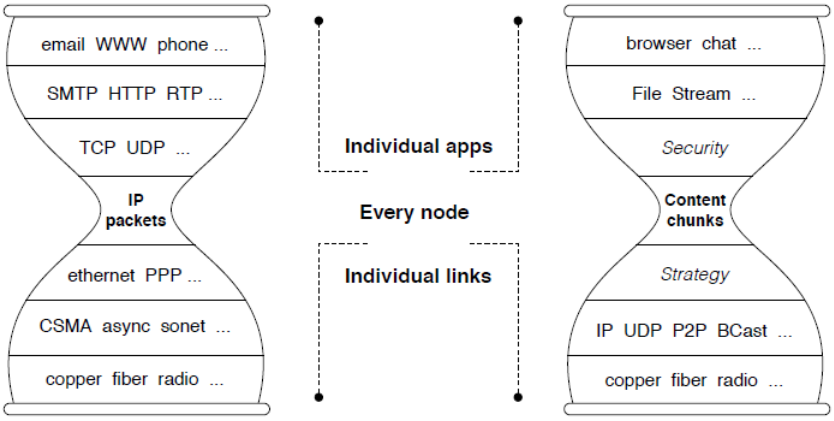


Figure 2.2: NDN Architecture Stack

Since NDN deals with application namespace directly at the network layer, the path of an Interest packet being forwarded has to be recorded in the network for the corresponding Data packet to reversely follow back as otherwise there is no way data can get back to the consumer (client) unlike in TCP/IP where destination IP and source IP could simply be swapped to send back a reply. NDN thus utilizes a stateful forwarding plane. Each Interest packet has a lifetime after which it expires if there is no Data satisfying it and the record is deleted from the network. Moreover, multiple Interest packets targeting the same piece of data content can be aggregated in the network. Also, the fetched Data packets can be cached along the path to satisfy future Interest packets asking for the same data.

2.2 Distributed Dataset Synchronization in NDN

A significant shift in the Internet world today has led us to an era where almost all applications are built around a massive scale distributed system requiring multi-party communication. Typically, many applications such as group chats, file sharing, and joint document edit-

ing, require knowledge about *shared* dataset such as chat messages and their order, changes to shared folders, and file revisions as well as edit actions. While top-notch applications like Google Drive and Messenger has effectively addressed synchronization in multi-party communication systems, they still are based on the centralized paradigm and introduce a single point of failure and centralized control of the data. To achieve dataset synchronization in a decentralized manner, a number of different peer-to-peer solutions [AS04, FSK14] have emerged. However, they require complex peer-to-peer network overlay structure. This is because TCP/IP naturally doesn't support broadcast/multicast which is essential in peer-to-peer systems.

NDN, on the other hand, through its name enabled inherent multicast support brings new opportunities to solve this problem in completely distributed fashion. A number of *sync* protocols in NDN [SYW17] were developed including ChronoSync [ZA13], PSync [ZLW17], and VectorSync [SAZ17]. NDN *sync* provides an important abstraction for multi-party communication where applications running on top of this sync protocol can directly publish and consume messages in a local copy of shared dataset that is synchronized across a group of distributed nodes. It is important to note that distributed ledger technology is exactly the same as distributed dataset synchronization. Here a node attaches a new record to its local ledger which is then propagated to all the other nodes in the system to be added to their local ledgers. At any given time, the local view of ledgers at any two nodes may vary because of network latency and partition and so they need to be constantly synchronized. This leads us to consider building DLedger applications over one of the designed NDN sync protocols mentioned above. However, each of these has shortcomings for our use case.

ChronoSync and PSync use "long-lived" Interest packets to pre-establish the return path of the data produced which carries information about dataset state changes so that other nodes can update. However, this causes the overhead of maintaining soft-state of Interests in the network and requires periodic re-transmission of the Interests on their expiry. Also, since each long-lived Interest can only bring one Data packet back as per NDN protocol convention, both ChronoSync and PSync face issues with simultaneous data publishing from different nodes and require additional Interests (with exclude filter selectors) to retrieve

simultaneous updates from multiple nodes. Since there could be multiple nodes producing records simultaneously in a distributed ledger system, both ChronoSync and PSync prove to be inefficient sync protocols for DLedger application.

VectorSync addresses these problems by multicasting Notification Interest in the system to advertise the summary of state changes in a local dataset rather than using long-lived Interests. However, synchronization in VectorSync is carried out using managed group memberships. Moreover, it utilizes the leader-driven process to synchronize the view among all the nodes in the system and leverage sequential dataset naming to synchronize dataset using *version vectors*. While VectorSync can serve as an underlying sync protocol for DLedger, it introduces the overhead of leader selection and group membership list maintenance which is not required in DLedger application unlike in applications such as chat room. This motivated us to create entirely new simple sync protocol for our DLedger application leveraging ideas from VectorSync. We discuss DLedger sync protocol in section 3.3.

2.3 IOTA: Tangle

A Blockchain data structure isn't network partition friendly; more specifically, since a consensus has to be reached on a single chain, two chains formed locally in different partitions can't be merged together. As a result, all records appended within the smaller component will be discarded (because the longest chain wins!). The businesses need to maintain a consensus backed replicated storage of records and can't lose this storage as a result of a network partition. One obvious way to solve this problem would be to consider using the Directed Acyclic Graph (DAG) as a fundamental data structure which allows the merge of multiple chains after recovery from the network partition. This led us to explore and leverage ideas from IOTA's Tangle [Pop18].

IOTA is a cryptocurrency using a distributed ledger technology called Tangle at its backbone. The underlying data structure is DAG unlike conventional cryptocurrency systems like Bitcoin who use Blockchain. IOTA claims to work for IoT because of their feeless transactions and low resource requirements as they use lighter weight PoW compared to

Bitcoin. However, in fact, it takes several minutes even for a modern PC to compute this PoW [IOT], thus proving itself not so ideal for resource-constrained IoT devices. Different from the current practice where dedicated nodes called *miners* incorporate transactions into blocks and append them to the ledger, nodes in IOTA behave as both creators and miners of blocks. Whenever a node wants to add a new block to the ledger, it must verify two previous blocks as well as perform lightweight PoW computation.

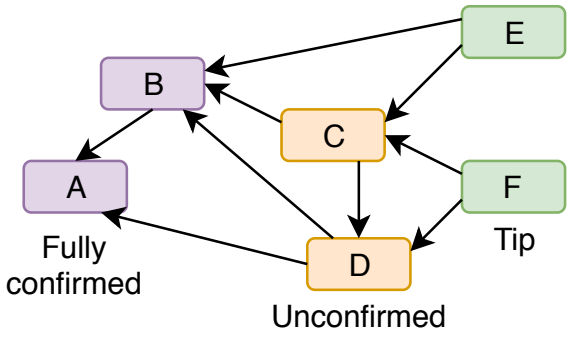


Figure 2.3: IOTA Tangle

In IOTA's Tangle, each vertex in the DAG represents a block and edges represent the approval relation between the blocks. For instance, block F in Figure 2.3 approves blocks C and D directly while blocks A and B indirectly. Each block in Tangle carries weight, in addition to two pointers to previous blocks and payload (block contents), which is simply the amount of work done towards PoW summed with the weights of blocks approving it. When a block is approved by all the recent tips in the system, it is said to be fully confirmed and accepted by the system. Otherwise, it is not yet accepted by the system and could be abandoned forever from acceptance in the future if it stops getting approvals from new incoming blocks. Note that blocks F and E in the figure 2.3 are called *tips*: they have not been approved by any block (that is, there is no incoming edge to them).

The *Tip Selection Algorithm* is used to decide which two tips to approve when a new block is appended into the ledger. IOTA takes a weighted random walk called Markov Chain Monte Carlo (MCMC) to select tips. It uses the weight of the blocks to perform this walk. The walk starts from some ancient blocks in the ledger (for example, *genesis* or root blocks) and ends at a tip block.

As we'll see in Chapter 3, our DLedger design leveraged from IOTA's Tangle is still quite different from IOTA except for the basic data structure (DAG). We explain the reasons for this in section 3.6.

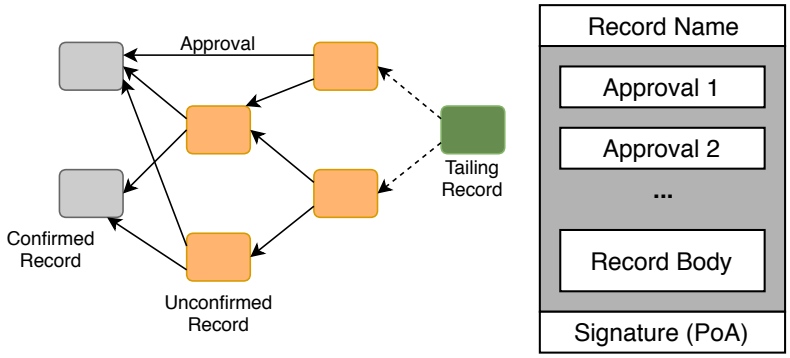
CHAPTER 3

DLedger: Private Distributed Ledger over NDN

3.1 Design Overview

In DLedger, we assume a private distributed system that is made of a wireless meshed peer-to-peer network over NDN. Peers could involve business service provider nodes, customer nodes and any other parties interested in the data records. Identity manager, appointed by the business service provider, also belongs to this peer-to-peer network. Note that there could be more than one identity manager, in which case, cross-certifications among them is required. These identity managers provide each peer with a digital certificate to bind their identity name with the public/private key pair. Each record generated by a peer bears a peer's signature. Using the certificates issued by the identity manager, other peers can then verify the authenticity of records. Moreover, these identity certificates issued are appended to the ledger system just like any other record.

In a nutshell, DLedger works in a way where each peer generates a new record, which is either event-triggered or periodic depending upon system design and requirement and appends it into the DLedger. At the same time, it also cryptographically signs this generated record so that the other peers can verify its authenticity and integrity using the certificates issued by the identity manager. This signature that is verified with the public key certified by the identity manager is called *Proof of Authentication (PoA)*. Also, each newly generated record *approves* old records in the ledger by verifying them against the system security rules and policies. When a record gains enough number of approvals from different peers, it is fully accepted by the DLedger system, that is, the system is said to have reached consensus on this piece of data.



(a) A DAG

(b) Record Format

Figure 3.1: DLedger Data Structure

We describe the design of DLedger from five dimensions in the following subsections.

3.1.1 Data Structure and Record

Similar to IOTA, DLedger uses a Directed Acyclic Graph (DAG) as an underlying data structure for storing all the records. As shown in figure 3.1a, each vertex in a graph is a record carrying peer's data while each edge represents an *approval*. Different from IOTA where each block only has two approvals, each record in DLedger can have n ($n \geq 2$) such approvals adjacent, with at least two approvals being distinct out of n, to n previous records (they could coincide) in the DAG ($n = 2$ in figure 3.1a).

Note that as in IOTA, these approvals can be direct or indirect, that is, when a record approves n previous records in the DAG, it implies that it is indirectly approving all the records beneath these approved record in the DAG (recursive approval). This means that while approving n records with a newly generated record, a peer should verify all the records being, directly and indirectly, approved as a result. The incentive to do this lies in the fact that if a peer appends its newly generated record R_{new} by approving n records without verifying the old records beneath, there could be some invalid record(s) $R_{invalid}$ in the DAG that may be indirectly approved by peer's record R_{new}. When in future some other good peer (abiding by system rules and policies) will generate a new record R_{other}, it will verify all old records beneath its chosen n approvals that include R_{new} before appending R_{other} to

the ledger. At this time, they will recognize $R_{invalid}$ to be violating system rules and policies and further abandon approving all the records that directly or indirectly approved $R_{invalid}$, including R_{new}, considering them to be malicious to ensure the system security. Thus, to prevent its records from getting abandoned from being approved by other peers, a good peer should always verify all the records in the DAG, directly and indirectly, being approved before appending the new record into the system.

In DLedger, a record is identified by a unique name as shown in figure 3.1b. This name carries a hash digest of the record generated from record content. As contents, a record carries n approvals, that is names of the previous records it approves, as well as the body (actual payload). This is similar to a block in IOTA. However, each record in DLedger bears a signature or PoA, computed using record name and contents.

There are three different status of a record in DLedger similar to IOTA.

- New Record: this is a record not known to any other peers in the system; it is a newly generated record by some peer
- Unconfirmed Record: after a peer generates and advertises its record to the network, other peers acknowledge it and validate it against system security rules and policies; after validation, they add it to their local ledgers. Before this record gains enough approvals from a pre-decided number of different peers, it is called an unconfirmed record.
- Confirmed Record: when a record gains approvals by the later appended records generated by enough number of different peers (called *entropy*, denoted by E), the record is confirmed and fully accepted by the system. Note that this is different from IOTA where each block has a *weight* which directly affects *Tip Selection algorithm*, a weighted random walk to select blocks for approval. Once a block is directly or indirectly approved by all the recent tips, it's status is changed to confirmed in IOTA. We reason this difference from IOTA in the section 3.6.

  Entropy in DLedger is explained by figure 3.2. Note that entropy of record $A1$ is 2 because it has been approved by **two different** peers B and C. When record $C2$ is appended to the system, it increases the entropy of record $A2$ but the entropy of $C1$ remains the same.

This is because record $A2$ was approved by record from a different generator than itself, thus adding to its entropy.

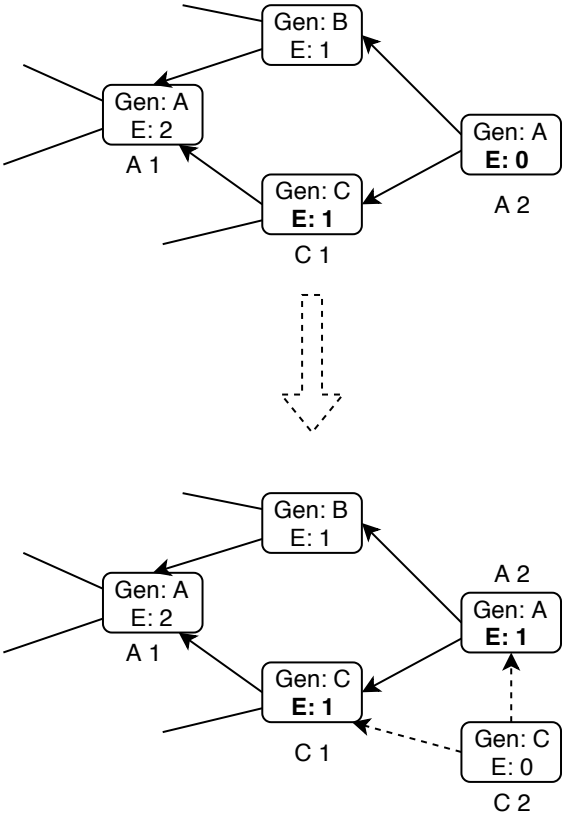


Figure 3.2: Entropy in DLedger

The goal of a record is to become confirmed. This means that record has been accepted by DLedger's peer-to-peer system and consensus has been reached on its validity. The threshold for entropy required for a record to become confirmed is denoted in this thesis by $E_{confirm}$. Note that the records which are last in the chain in DAG, that is, they don't have any records chaining to them are called *tailing* records (figure 3.1a) which is synonymous to *tips* in IOTA.

Note that when appending a newly generated record, since the system has already reached consensus on confirmed records, it doesn't need to validate those records being indirectly approved while choosing and making n approvals. As shown in figure 3.3, a new record needs to validate only the highlighted unconfirmed records while making two approvals.

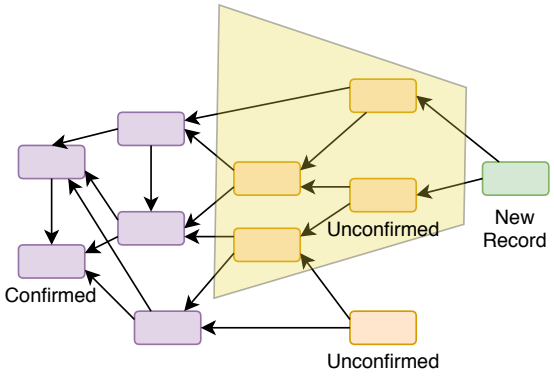


Figure 3.3: Validating Records at the approval

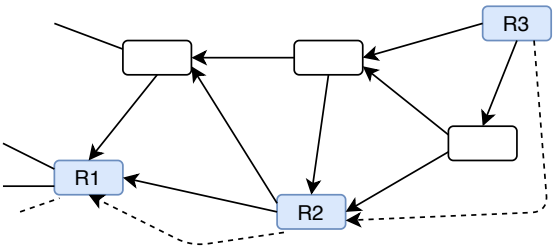


Figure 3.4: DLedger Revocation Records

Besides the usual records, DLedger system has special records called *revocation records*. Just like designated identity managers append the issued certificates to the ledger system, they also append notice of revocation to the ledger for any node that is suspected and found to be malicious. Different from other records, these records have one extra pointer apart from approvals that point to previous revocation record in the ledger. In figure 3.4, $R1$, $R2$ and $R3$ are revocation records and the dashed lines are the extra pointers. These pointers help nodes to do a fast search for revocation notices for any node rather than walking through entire ledger during PoA verification to figure out if the node's certificate is still valid or revoked. In the example provided, a node will only have to store the location of $R3$ since it is the latest revocation record. Once it knows where to find $R3$ in the ledger, it can easily follow back to previous revocation records in the ledger using these pointers. Note that to differentiate a usual record from revocation record so as to store the location of most recent revocation record, a node can simply look at the record content or signature field. Different from the other records, revocation records will bear the signature of identity manager themselves.

3.1.2 Network

In DLedger, a peer needs to multicast announcement regarding the new record it generates with a hint to allow other nodes to fetch it using correct names. Moreover, the local ledger (DAG) needs to be periodically synced between nodes to keep the system in consensus after network partition or latency. The current network architecture, TCP/IP, achieves multicast in mesh topology by having nodes relay (re-broadcast) packets for each other so that they reach all nodes in the network eventually. This is unfit in the IoT environment because of following reasons. Some of these have even been realized in [SYD16].

1. IoT devices are known to go into energy saving mode frequently and might miss multicast announcements. More important than that, whenever a certain node misses a multicast packet and wants to retrieve data after waking up after realizing the missing piece of data, it is quite possible that the node it puts in its target IP address to fetch the data from is sleeping, thus making data unavailable on demand. This means it'll then have to flood the network with hope to retrieve missing data from some other active node, thus making inefficient use of network bandwidth.

2. Because IP packet doesn't hint to nodes in any way about the contents it carries, packet suppression is hard or impossible to achieve at the network layer.

3. The most common model in TCP/IP for security is the channel-secure model where data communication channels are protected and secured. However, this incurs the overhead of setting up a secure channel (for example, TLS handshake), maintaining connection states, etc. which isn't friendly for constrained IoT devices.

NDN, on the other hand, provides inherent data resiliency and replication, thus making it always available through intermediate node caching. This is also especially useful in poor network conditions where re-transmissions are frequent as node doesn't have to go all the way to the producer of the record to fetch it (as in case of TCP/IP where target IP address will be of the producer), but rather can fetch it from intermediate node cache. Moreover, nodes may also benefit from Interest aggregation reducing network overhead. To be more

specific, multiple Interests pulling the same record from the network will be aggregated at intermediate nodes in NDN. Since packets in NDN carry names rather than meaningless IP addresses, it helps with packet suppression. To be more specific, in case of single multicast relayed over multiple nodes, even though collision avoidance will apply in TCP/IP, it still won't suppress duplicated multicast packet. In case of NDN, nodes can learn from the name of multicast packet that their neighbor already sent out the packet and so void from sending the same data out.

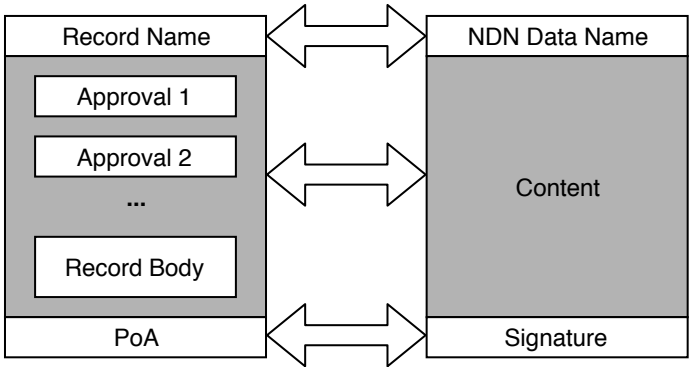


Figure 3.5: DLedger Record and NDN Data Packet

Note that in a distributed ledger system, peers ultimately care only about data and data security. Instead of making communication secure, NDN secures data packets itself at the network level, thus avoiding overhead incurred in *channel-secure* mechanism. Note that constructing DLedger over NDN, a record is simply represented by an NDN Data packet (figure 3.5). This is different from traditional distributed ledger systems over TCP/IP where a record is an application-layer block which is then encapsulated in an IP packet at the network layer for its transmission into peer-to-peer network. Being a Data packet, a record in DLedger could utilize all NDN's built-in security mechanisms.

3.1.3 Security

The DLedger system provides security through publishing and achieving consensus: all the records are kept in a data structure that is shared and synchronized among all peers. This consensus backed replication, along with PoA, provides data integrity, authenticity and avail-

ability. We also provide some security policies 3.2.2 to ensure the system's robustness to prevent potential threats like spam attack, collusion attack, and lazy peers.

3.1.4 Archiving ledger

Note that the size of the local ledger will keep increasing as new records are constantly generated as well as peers in the system increase. Since DLedger is designed essentially for constrained IoT devices, we need to take into account the restricted storage capacity.

When a peer device is about to run out of storage capacity, a peer could copy the confirmed records from its device into a backup server controlled by them. Note that even though this backup server is controlled by the peer, it cannot modify the backed up records and fool any other peer in the network. Since these are confirmed records, it means the system already reached consensus on these records (and hence, that part of a DAG) and their validity. Not just this but altering a record would change its hash digest causing an avalanche effect where all records directly or indirectly approving this record will require a corresponding change in their approvals field which basically carries the names of approved records. Moreover, this will also imply generating a new signature or PoA for all these parent records which can never be forged by a malicious peer as it doesn't have these record generators' private keys. We explain this further in section 3.2.2.2.

Note that a snapshot mechanism may also be used locally at each node for the data that can be summarized into a single or few records from multiple records. For example, Operant Networks stores energy consumption data for every measuring cycle into the records. When a peer device is about to run out of space, energy consumption data for each peer in multiple records (for multiple measuring cycles) could be summarized as a balance (total consumption till date) and stored in a single record, thus retaining the required information but summarizing it into a single record, thus reducing the space formerly consumed by multiple records.

3.2 Proof of Authentication and Security Policies

In this section, we discuss how DLedger achieves security through publishing and consensus and robustness through well-defined security policies.

Note that the primary goal of DLedger is to ensure integrity and availability of the records. As discussed earlier, it achieves integrity through Proof of Authentication (PoA) and availability by replicating record across all the peers in the system. Through such replication, every peer's action is observed and recorded in the ledger system.

However, just PoA and replication doesn't ensure robustness in the system. Some possible vulnerabilities are:

- **Lazy Peers** Peers may be selfish and may not keep their ledger updated. This means they approve very old (already confirmed or near to confirmation) records in the ledger system at the time of new record generation and do not verify newer records by other peers.
- **Self approvals** Some malicious peers can form an indefinitely long chain of records by appending records with approval to their own existing records in the ledger. Such behavior would increase the depth of ledger increasing the verification work of other peers as well as exploiting their storage capacity.

In section 3.2.2, we define security policies to prevent such vulnerabilities in DLedger.

3.2.1 Proof of Authentication

As mentioned in section 3.1.2, a record in DLedger is an NDN Data packet. Hence, PoA is realized as a Data packet's digital signature. This means validating a record against PoA adds no more application-layer complexity. The PoA process can be described as follows:

1. A peer signs a newly generated record R_{new} using its private key.
2. When another peer receives this new record R_{new}, it verifies this signature using the public key certified by the shared designated identity manager. As mentioned in sec-

tion 3.1, each identity manager appends the certificates issued to the system. Hence, peer checks if the certificate is present in the system. It also needs to check if there was any revocation record added to the system for this peer (generator of R_{new}). Only after these two checks, it accepts PoA attached with the record R_{new}.

3.2.2 Security Policies

Every entity in DLedger system should follow the proposed security policies to protect the system from abuse.

3.2.2.1 Contribution Policy

This policy prevents peers from behaving lazily, that is, approving already confirmed or near confirmed record in the DLedger system. Lazy behavior brings three disadvantages to the system:

1. By not approving unconfirmed records, the size of the unconfirmed record set will keep increasing indefinitely. Soon they will outnumber the number of records non-lazy peers can even approve in a finite time, thus expanding the width of the ledger indefinitely and ultimately crashing the system.

2. When a system doesn't enforce peers to only select from unconfirmed records, lazy peers will not bother to synchronize their local ledgers to the latest state since they can do away with approving confirmed records in the system and still get their records verified and accepted by other peers. This means lazy peers won't actively verify newly generated records by other peers and contribute to the well-being of the system.

3. Not just this but lazily approving near confirmed (but still unconfirmed) records is also not good for the system as these peers then won't really actively verify other peer's records. This is because there won't be many records beneath the record being approved if its near confirmation, thus not needing to verify many other peers' records.

Hence, to prevent DLedger system from such vulnerability, *Contribution Policy* enforces

the peers to only approve *tailing* records (and hence, unconfirmed) from its local ledger whenever it generates a new record. Note that this simply means that all n approved records which are essentially tails have an entropy of 0.

$$\forall i \in range(1, n), E_i = 0$$

However, note that because of network latency or delay, it is possible that a tailing record in a local ledger is actually not a tail anymore in other peers' local ledgers. To handle this scenario, we propose another threshold of $E_{contribution}$.

$$0 < E_{contribution} < E_{confirm}$$

Whenever a peer fetches a new record, it should check whether $E_i > E_{contribution}$ $\forall i \in range(1, n)$ for all n approvals before accepting the record along with PoA and other rules. Whenever peer generates a new record and approves n records, all the approved records should have entropy $E = 0$ to be in safety region accounting for network latency and delays. However, sometimes, when a peer appends a new record to the ledger, it is quite possible that the number of tailing records available is less than n. In such a case, a peer can select recent unconfirmed records, even though they are not tails, with $E < E_{contribution}$.

Moreover, as mentioned in section 3.1, to avoid a peer from making all n approvals to the same tailing record, DLedger enforces peers to have at least two approvals to be distinct out of the n. This will ensure that peers verify a reasonable number of old records when appending a new record. A peer fetching a new record must also verify this along with $E_{contribution}$ threshold.

3.2.2.2 Interlock Policy

Based on *Contribution Policy*, a peer would be enforced to approve n *tailing* records whenever it generates a new record. However, DLedger system needs the policy to reject self-approvals by the peer, that is, a peer shouldn't be allowed to approve their own records. We call this an *Interlock Policy*. Using self-approvals, a peer can easily make the depth of ledger increase

indefinitely causing more work for legit peers during verification of new records as well as exploiting storage capacity. The *Interlock Policy* disallows any adjacent records to be from the same peer.

$$\forall R_i, R_j \quad R_i \text{ and } R_j \text{ are adjacent} \Rightarrow P_i \neq P_j$$

where P_i and P_j are the generators of R_i and R_j, respectively.

Note that not allowing self-approvals through *Interlock Policy* also helps towards maintaining immutability of the records in the ledger. As explained in section 3.1.4, changing a record's hash (and hence, record) is nontrivial because a malicious peer $P_{malicious}$ will then have to update all the records directly or indirectly approving this record since this record's name changes. *Interlock Policy* guarantees that no two adjacent records are generated by the same peer. Hence, to change a record, $P_{malicious}$ will have to change at least one other record R_{other} directly approving it which is impossible as $P_{malicious}$ wouldn't know other peer's private key to recompute signature after changing R_{other}.

As shown in figure 3.6, A is a malicious peer who wants to alter a record with hash "`0a12cd`". Because of *Interlock Policy*, A cannot create another record to directly approve this record. Records by peers B and C directly approve record "`0a12cd`" while record "`q4qedq`" generated by peer A then approve these records. Making alterations marked in **blue** is easy since they are controlled by peer A. However, altering record "`0a12cd`" changes its hash to "`12fh45`" which breaks approvals made by the records "`8dqdcd`" and "`1dd355`". These records marked in **red** and their approvals are nontrivial to change. Since a record's PoA is computed from its name (and hence, record hash) and content, peer A will need peer B and C's private keys to alter their records. This restricts peer A to make any changes in the record, thus making ledger immutable by *interlocking* records.

3.3 Ledger Synchronization

DLedger utilizes data-centric sync protocol leveraged over NDN's inherent multicast support through NDN Forwarding Strategies [YAM13]. We now discuss the communication protocols

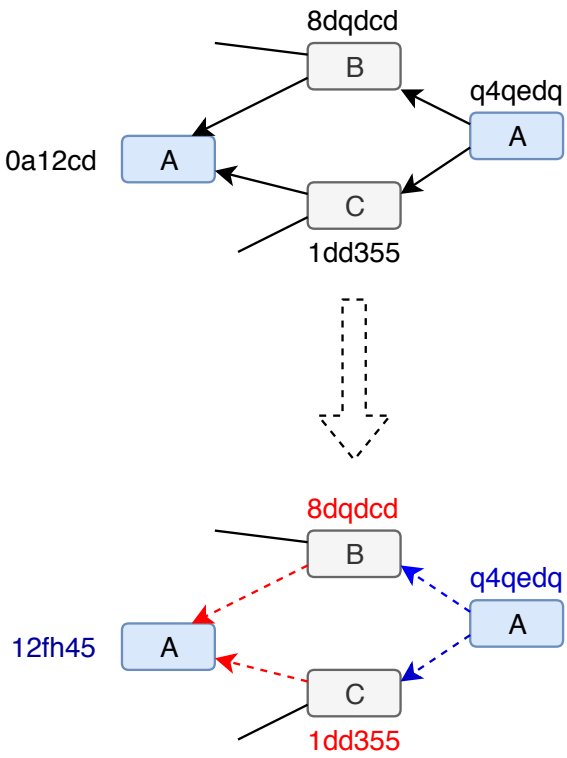


Figure 3.6: Immutability through Interlock

in DLedger.

3.3.1 New Record Notification Protocol

Whenever a peer generates a new record, it needs to relay it to other peers in the system so that they can update their local ledgers and system can achieve consensus. However, since NDN utilizes *pull* model instead of *push* model as in TCP/IP, a peer in DLedger cannot simply push newly generated record into the network. There needs to be a mechanism to notify other peers about this new record so that they can issue an Interest to fetch it. As mentioned in section 2.2, DLedger leverages ideas from VectorSync protocol. DLedger uses the New Record Notification Protocol to achieve this. Whenever a peer generates a new record R_{new}, it multicasts Notification Interest I_{notif} in the network. The structured meaningful name of I_{notif} bears hint for other peers to successfully enumerate the name of the newly generated record. Other peers then issue a unicast Interest to fetch the record

from the network (generator of the record or intermediate router cache). Once this new record is obtained, a peer will validate it against system security rules and policies.

3.3.2 DAG Sync Protocol

The network is unreliable and the parameters (latency, RTT, etc.) are constantly varying. Because of factors like network partition, delays, latency, etc., the local ledger at a node could go out-of-sync from local ledger at other nodes. This is also true when the node comes back online after a brief sleep (being offline). Hence, a node has to regularly check for the latest state of the ledger and keep updating the local copy to the most recent known state. The DAG Sync Protocol to synchronize ledger states is as follows.

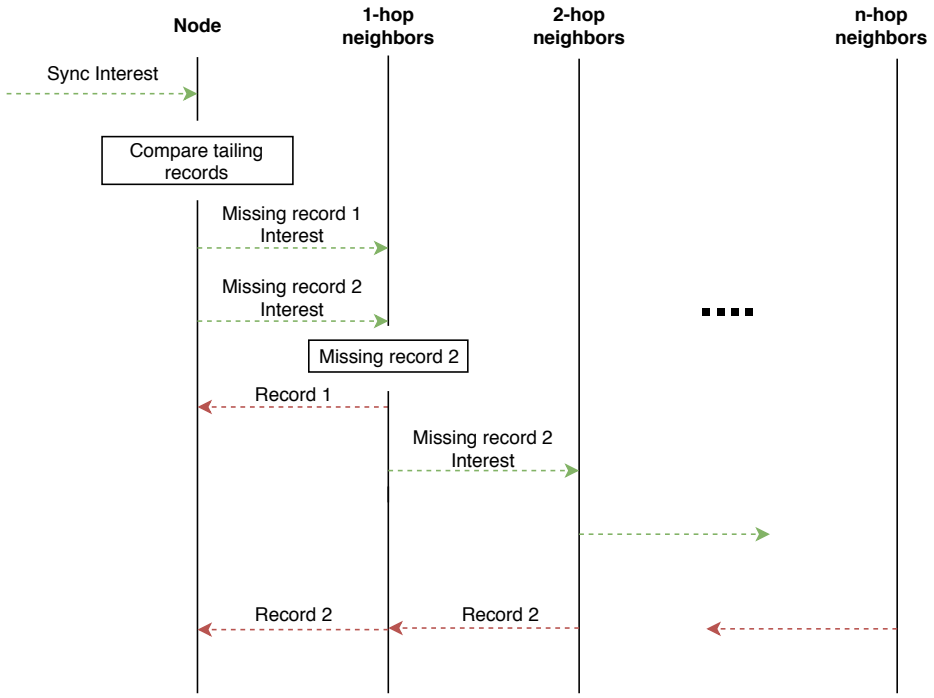


Figure 3.7: DLedger Synchronization

A Sync Interest I_{sync} is multicasted in the network to trigger sync process. This I_{sync} carries a list of tailing records in current local view of the ledger as Interest parameter. When a node receives I_{sync}, they compare the tailing record list with the tailing records in their own current local view. Once it figures out the tailing records that are missing in its local ledger, it issues one-hop multicast Interests to fetch them from its neighbors

(figure 3.7). Tracing the approvals in tailing records and thereby in each fetched record, a node can recursively fetch all the missing records on its ledger and update it to the latest known state. When a neighboring node receives such one-hop record fetching Interest, it first checks whether it has the desired record in its local ledger. If it does, it replies back (record 1 in figure 3.7). Note that there could be multiple neighboring nodes having this desired record. To avoid the collision as a result of simultaneous replies, a network would need some Channel Activity Detection. If none of the neighbors have desired record, a node will retry sending one-hop record fetching multicast Interest after waiting for some time. Meanwhile, the neighboring nodes that miss the desired record multicast one-hop record fetching Interest to their neighbors (record 2 in figure 3.7). Eventually, the original requester of the record as well as its n hop neighbors, all will receive this missing record updating their local ledgers.

This I_{sync} is sent periodically as well as triggered by two events: when a node comes back online after sleep, network partition, or any such conditions and when node realizes that it has a more recent state of ledger for some part of the DAG, that is, some of the tailing records in I_{sync} are not tails in its local ledger anymore.

3.3.3 Naming Convention

In this section, we introduce the naming structure for Interests in DLedger and record name.

Using a predefined naming structure automates the DLedger processing. Using such a predefined naming structure would help peer know (i) how to construct a Notification Interest so that other peers can successfully enumerate the name of a newly generated record, and (ii) how to construct a Sync Interest so that a peer can extract the tailing records list and assemble individual tailing record names to construct a record fetching Interest for the missing record.

Each record in DLedger is named as:

"`/<multicast prefix>/<creator prefix>/<record hash digest>`"

For example, the record name in the solar system network prototype would look like

"/operant-dledger/solar-gw-device1/axb12...yxz3". The "/<multicast prefix>" component in the name ("/operant-dledger") is a network-level predefined multicast prefix in NDN that each peer in the system will register to receive multicast Interests. The "/<creator prefix>" component ("/solar-gw-device1") denotes the generator of this record. The "/<record hash digest>" component ("/axb12...yxz3") is a unique identifier for each record. Besides the multicast prefix, each peer also registers "/<multicast prefix>/<cretor prefix>".

Note that in DLedger, a record is essentially a Data packet. Hence, the record name mentioned above will be used as record fetching Interest to retrieve newly generated or missing (during sync) records. Since record fetching Interest starts with multicast prefix, it can be received by all the nodes in the system to satisfy it. Moreover, a network can use the longest prefix match to forward Interest to the generator of the record if Interest cannot be satisfied by neighboring peers.

A Notification Interest follows this structure.

"/<multicast prefix>/NOTIF/<creator prefix>/<record hash digest>"

On receiving Notification Interest, peers can simply rip off the "NOTIF" component to compose the name of the record and fetch it using unicast Interest. As a simple example, the Notification Interest for the solar system network record example above would be "/operant-dledger/NOTIF/solar-gw-device1/axb12...yxz3".

A Sync Interest carries a list of tailing records in peer's local ledger and has a format as follows.

"/<multicast prefix>/SYNC/<digest of tailing records"

3.3.4 Merging of DLedgers

Whenever there is a network partition, nodes in different partitions will have significantly different copies of ledgers. This is because nodes in different partitions won't be able to fetch each other's records. This also means that a peer in one partition can approve an unconfirmed record which in fact has already reached confirmed status in another partition. Hence, after

the partition is resolved, and peers recursively fetch each others missing record, they can't really verify the *Contribution Policy*. We, therefore, only enforce *Contribution Policy* on the newly generated advertised records and not on the fetched records while syncing.

Note that one may argue that nodes can then deliberately form network partition and show lazy behavior by intentionally approving confirmed records and still get their records confirmed as *Contribution Policy* doesn't apply to fetched records while syncing. Not just that but these lazy peers don't even need to fully synchronize their ledgers since they can do with old confirmed records. However, note that DLedger's certificate revocation design would prevent such lazy behavior as explained in section 3.4.7.

3.4 Security Assessmnet

In this section, we propose a threat model (identify potential attacks) and do security assessment to indicate how our policies and security mechanisms protect DLedger from these threats.

3.4.1 Records Spam Attack

Given that PoA is lightweight and efficient even for the constrained IoT devices, spamming the peer-to-peer network of DLedger system is easy. Spamming the network and ledger system using fake records with a valid PoA will incur significant network overhead increasing latency and lowering throughput, keep legit peers busy with verifying all these spammed records as well as let the size of ledger grow indefinitely huge, eventually crashing the system.

We think that this problem could be solved at the application level by having well-defined application semantics. For example, abnormality in the number of records produced in unit time could be detected if historic averages are known and further actions towards malicious spamming node could be taken. The rate of incoming Notification Interests from a single peer could be measured to detect such abnormality. Note that without legit peers fetching the spam records and appending to the system after verification, just generating new records

in bulk locally doesn't abuse the system much except Notification Interest flooding. Having such abnormality detection based on the rate of Notification Interests would thus save ledger from getting indefinitely huge because of such spam attack as well as save peers' resources from useless verification of spam records.

3.4.2 Denial-of-Service and Reflection Attack

A malicious peer may carry out reflection attack in DLedger by generating false Notification Interest to non-existent new record carrying the name of some other peer, that is, forging "`/<creator prefix>`" component in Notification Interest. In this case, all other peers will send new record fetching Interests to this other peer. Such an attack could be diminished by making peers attach PoA also to the Notification Interest. Note that PoA is calculated using a record's name and content. Since record's name consists of record hash and record producer's prefix, carrying out reflection attack means forging other (victim) peer's PoA which is impossible as the attacker wouldn't have victim's private key.

3.4.3 Collusion Attack

In a distributed ledger system, two or more peers may collude to approve and verify each other's malicious record given that *Interlock Policy* rejects peers from making self-approvals. DLedger protects itself from this abuse by having record statuses, that is, a record is only confirmed and the consensus is said to reach in the system after it gains entropy $E_{confirm}$. Hence, unless more than k ($= E_{confirm}$) malicious peers out of total N peers collude to surpass this entropy threshold, DLedger avoids any rational attack as a result of such collusion. We say that DLedger follows (k, N) $k < N$ threshold scheme.

3.4.4 Malicious Identity Manager Bot Attack

A designated trust manager could go rogue and create virtual nodes in the peer-to-peer system and issue valid identity certificates to them. Such an attack would go unnoticed with simply PoA verification. However, in DLedger system, as noted earlier all the certificates

issued by designated trust managers are appended to the system. Besides PoA, a peer also checks if the certificate is in the system, otherwise rejects PoA. This doesn't prevent virtual nodes in the system but at least malicious designated trust manager will leave behind the traces (certificates) in the ledger for further examination whenever intrusion or misbehavior is detected.

3.4.5 Data Confidentiality

Note that since DLedger is built over NDN, data confidentiality can be achieved in a way where only the peers of the P2P network can read record contents and no outsider can learn about it. This can be done by encrypting records and distributing keys only to the system peers so that only they can read record contents. Ideas about automated key distribution could be leveraged from the Named Based Access Control [YAZ15] technique used in NDN to achieve encryption keys in multi-party communication.

3.4.6 Full Node vulnerability

The traditional distributed systems over TCP/IP such as Bitcoin which usually use hashcash (for example, PoW) based consensus and gating control mechanisms are infeasible for all the peers in the network. Not all participating peers possess resources to mine (append) a new record, neither they have enough storage capacity to store entire ledger for record verification. These reasons lead to a differentiation between nodes based on their resource and storage capacity into the full node and light nodes. A light node would simply use an API to communicate with the full node to invoke record retrieval, dispatching and verification functions. It doesn't directly communicate with the peer-to-peer network. As a result of this full dependence on a full node for the system participation and block validity, they might be fooled by malicious full node or become a victim of DNS hijack attacks on full node thus resulting in communication with the malicious node.

Our DLedger system is protected from such vulnerability for the following reasons.

1. The PoA mechanism used by DLedger is lightweight even for constrained IoT devices.

Hence, a node doesn't have to lease out record validation to any other node.

2. The archiving mechanism described in section 3.1.4 allows for the efficient use of space.

3. Nodes in DLedger built over NDN's data-centric paradigm retrieve data from the network as a whole rather than a specific location. Hence, a node doesn't need to be dependent on a single specific node for data.

3.4.7 Laziness vulnerability

The *Contribution Policy* already prevents lazy behavior to some extent by enforcing peers to approve unconfirmed records and thus save system ledger from expanding indefinitely and ultimately crashing. Moreover, since identity managers also add revocation notices to the ledger, it works as a motivation to a node to actively sync their local ledgers to keep them updated to the most recent known state. If they don't, they might approve records of a peer whose certificate was revoked, thus approving an invalid record which in turn will result in abandonment of their own records by other peers. Moreover, by syncing ledger, a node will also be able to figure out whether its previous records got any approvals from other peers and, if not, debug the reason for abandonment. Hence, DLedger system design and security policies enforce peers to actively contribute to the system by verifying other peers' records as well as always trying to keep their local ledgers up-to-date by actively syncing.

3.5 Implementation

We started developing initial prototype [VK18] for Operant Networks in NodeJs. This was just a proof-of-concept implementation to test Ledger Synchronization protocol (section 3.3) and PoA (section 3.2.1). We decided to keep other things same as in IOTA's Tangle; to be more specific, even we used *Tip Selection Algorithm* (weighted random walk) and record weights instead of entropy as in new design.

The figure 3.8 describes pseudocode for the DLedger synchronization procedure for $n = 2$ (n is required approvals). The *OnReceivingSyncReq* function describes the tailing records

list comparison and sending out record fetching Interests for the missing tails. If the local ledger is found to be more updated, it calls *sendSyncRequests* function to trigger a sync with a list of its own tailing records. The function *OnReceivingMissingRecord* describes the recursive fetching of all the records (traced through approvals as described earlier). When a node receives missing tail T_{miss} from its neighbors, it checks whether the two ($n = 2$) approved records by T_{miss} are in its local ledger. If not, it needs to fetch them through multicast one-hop Interests as discussed. It adds the records it receives to pendingAppends stack. When it is done receiving all the missing records recursively, it will pop each record from this stack and append to its local ledger.

```
function onReceivingSyncReq(receivedTails)
    for each tail in receivedTails do
        if !(ledger.contains(tail)) then
            fetchRecord(tail)
        else if !(localTails.contains(tail)) then
            isOutDatedSync = true
        end if
    end for
    if isOutDatedSync then
        sendSyncRequest(localTails);
    end if
end function

function onReceivingMisstingRecord(record)
    if !(ledger.contains(record.approval1)) then
        fetchRecord(record.approval1)
    end if
    if !(ledger.contains(record.approval2)) then
        fetchRecord(record.approval2)
    end if
    pendingAppends.add(record)
end function
```

Figure 3.8: DLedger Synchronization Pseudocode

To simulate network partitions and achieve scalability in our tests, we decided to migrate to C++ and use ndnSIM [MAZ17], an open-source NS-3 based network simulator. This is because NDN doesn't have a network simulator written in NodeJS making it increasingly hard for us to test our initial prototype with multiple peers as well as simulate link failure scenarios. Another reason to migrate to C++ was that DLedger is meant to be used by IoT devices in future and C++ implementation would be more efficient. Moreover, the migration also helped us to pick out loopholes in our initial prototype design and account for them

with a better newer version [VMZ18]. We discuss why our initial prototype needed changes in the next section.

3.6 Reasoning the design

In this section, we reason why DLedger design, except for the core data structure, is different from IOTA's Tangle. Next, we also provide simulation results to show that the number of unconfirmed records in the DAG converges. Moreover, we also simulate the network partition to show two DLedgers merging.

3.6.1 Why change from IOTA's Tangle?

It is important to note that besides underlying network architecture, on an application level, IOTA and DLedger differs in the following manner.

1. IOTA uses a weighted random walk (MCMC) from confirmed blocks to the tips. It does so to avoid *lazy* behavior by the peers who might only approve very old blocks, thus not contributing to the system. A weighted random walk would increase the probability of tips approving more recent blocks to be selected over the *lazy* tips. DLedger on the other hand uniformly randomly selects the tailing records for approval at the time of new record generation. We decide to not use MCMC because of its inefficiency. Note that to perform MCMC, since the weighted random walk is performed from old records towards new records while the DAG edges point from new records towards the old records, entire DAG will have to be fetched first to start the walk. To avoid this, one can make DAG bi-directional. More specifically, one can store not only approvals made by each record but also store a list of directly approving records for each record. This will then allow a walk from old records to new records easily. However, as ledger becomes huge, this will prove to be space inefficient. One might argue that MCMC isn't enforced upon all the peers directly in IOTA. However, by game theoretic principles discussed in [PSF17], all nodes theoretically should converge to using the same tip

selection algorithm which will be MCMC since it'll be followed by a majority of the peers. DLedger completely avoids this by just maintaining a list of tailing records and selecting tails in uniformly random fashion to make approvals.

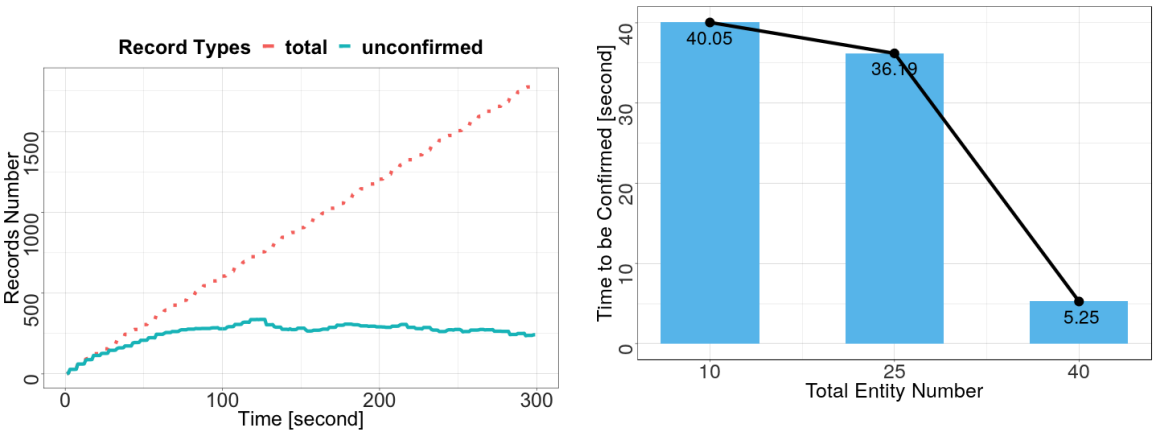
2. In IOTA, the gating control mechanism slows down appending of new blocks by the peer into the system. However, a spam attack as the one discussed in section 3.4.1 is quite possible in DLedger because of the lightweight PoA. This means a node can constantly keep approving its own records and increase the depth of ledger. Hence, for this reason, DLedger introduced *Interlock Policy* to disallow such self-approval spam by the nodes in the system. IOTA wouldn't need such policy as the rate of such self-approvals is low because of the PoW. Not just that but it is impossible for IOTA to have such policy given that nodes use one time signatures making transactions completely anonymous from one another. Hence, other peers in the system can't verify self-approvals even if they are present.

3. Moreover, again because of lightweight PoA, we cannot use weight as a threshold for consensus since gaining enough weight to change the status of the record to confirmed is easy through collusion between peers. To avoid this, DLedger records thus have an entropy instead of weight, different from IOTA. Also, since we don't perform a weighted random walk, there's further no need for weights in DLedger.

4. While IOTA avoids *lazy* behavior by doing a weighted random walk, DLedger enforces *Contribution Policy* to achieve the same.

3.6.2 Simulation

One obvious question about the design of DLedger would be to ask if, given the security mechanisms and policies, it ever converges. To prove this, we carried out a simulation using 30 nodes in a peer-to-peer network with $E_{confirm}$ set to 10. Each peer in the simulation generates a new record every 5 seconds. Figure 3.9a shows the plot which indicates that the number of unconfirmed records doesn't grow as the size of the ledger (the DAG) grows.



(a) Unconfirmed Records As DAG grows

(b) Average Time taken for Confirmation as nodes increase

Figure 3.9: Simulating DLedger

Specifically, after a certain point, the number of unconfirmed records will be around a constant number.

Figure 3.9b shows that as the number of nodes increases, with $E_{confirm}$ (set to 5 for this simulation) being same, the average time taken for nodes to be confirmed decreases. This is because more nodes generate more records overall and thus increase number and rate of approvals. This, in turn, increases the entropy of a record at a faster rate.

Figure 3.10 shows two DLedgers merging after a network partition created in the peer-to-peer network of 15 nodes. Network partition happens at $t = 20s$ and rejoins at $t = 120s$ with 9 nodes in one partition and 6 nodes in other. This simulation validates our Ledger Sync Protocol showing that recursive fetching of missing records does work.

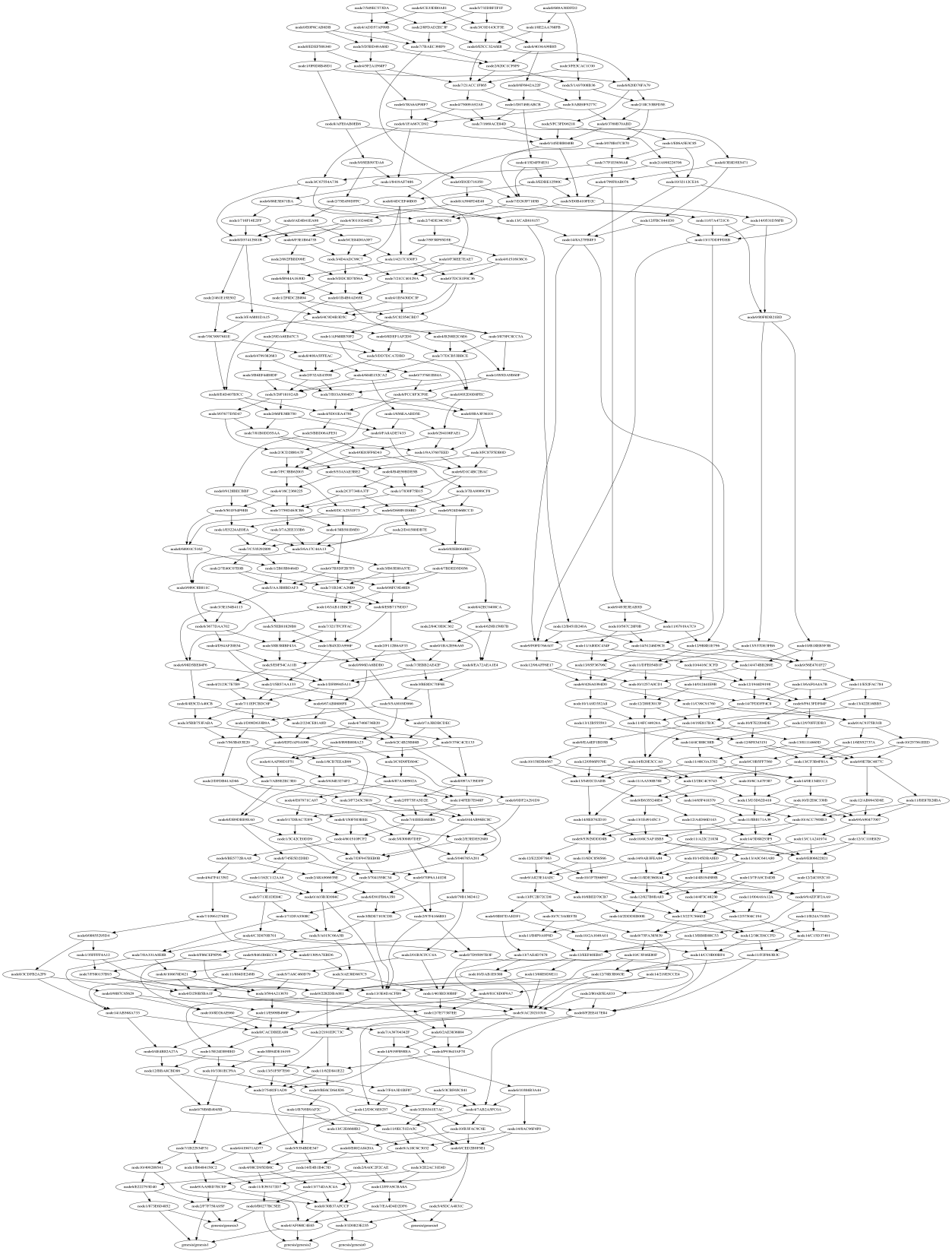


Figure 3.10: Merging of DLedgers after partition

CHAPTER 4

Use Case: Initial Prototype for Real World Solar Network

In this chapter, we talk about the motivation behind designing and developing a distributed ledger system for Operant Network's solar network. We also discuss DLedger's initial prototype developed specifically to Operant's use case.

4.1 Motivation: Security Concerns in Utility Scale Solar

Utility-scale solar is deeply integrated with the national electric grid today and is a critical infrastructure to provide utility to the sectors. Cyber-attacks on such utility-scale solar are becoming major threats today [SBL14, SK17]. Historically, these utility-scale solar were set up on their private independent network, not connected to the public Internet. However, as power generation has shifted from centralized power plants to massively scaled Distributed Energy Resources (DER), communication now takes places via the Internet. It is now impossible to physically separate solar networks to establish a secure private network and defend against cyber-attacks. Hence, utility control signals and site alarms could be hijacked and tampered damaging the equipment and personnel. Moreover, energy consumption and production records could go missing and tampered as well. Today, there is no record of such missing values or control signals and little evidence if they were corrupted. This would damage business integrity and authenticity of records provided by business such as Operant Networks since it is hard for financial parties involved to trust the data. It is thus necessary to log communication of records, control signals, and site alarms right at the point of generation. Not just that but they should be replicated across several nodes for fault-tolerance

and integrity as well as immutability through consensus. This is where distributed ledger comes into the picture. Each inverter node will maintain a local ledger storing all energy production and consumption records it generates as well as the ones generated by other nodes making an entire system a decentralized shared dataset.

4.2 Security Enabled by NDN and Distributed Ledger

Operant Networks has developed a rooftop solar network of Operant's solar gateway devices which communicate with Operant's server using LoRa wireless channel utilizing NDN protocol. Each of these devices records customer's energy production and consumption data and send back to the Operant server for accounting.

This existing communication over the NDN protocol already achieves most of the security attributes. Some of them are further enhanced by distributed ledger. We list them in Table 4.1.

Security Attribute	**Existing NDN Communication**	**Enhanced by distributed ledger**
Confidentiality	Encryption and Distribution of keys	
Integrity	Data Packet Signatures during transmission	Post transmission, PoA and consensus makes accepted ledger records immutable
Availability		Decentralized Replication
Authorization	Certificates	

Table 4.1: Security Attributes achieved together by NDN and Distributed Ledger

4.3 Prototype

We now discuss the initial prototype designed and developed for Operant's use case.

It is assumed that in Operant's solar network, the system operator sets up a trust anchor, and each rooftop solar gateway device trust this operator's digital certificates, and has obtained them. Therefore all the devices can authenticate each other, while an outsider without a valid private key cannot pass this verification.

In Operant's DLedger system, there are two main types of entities.

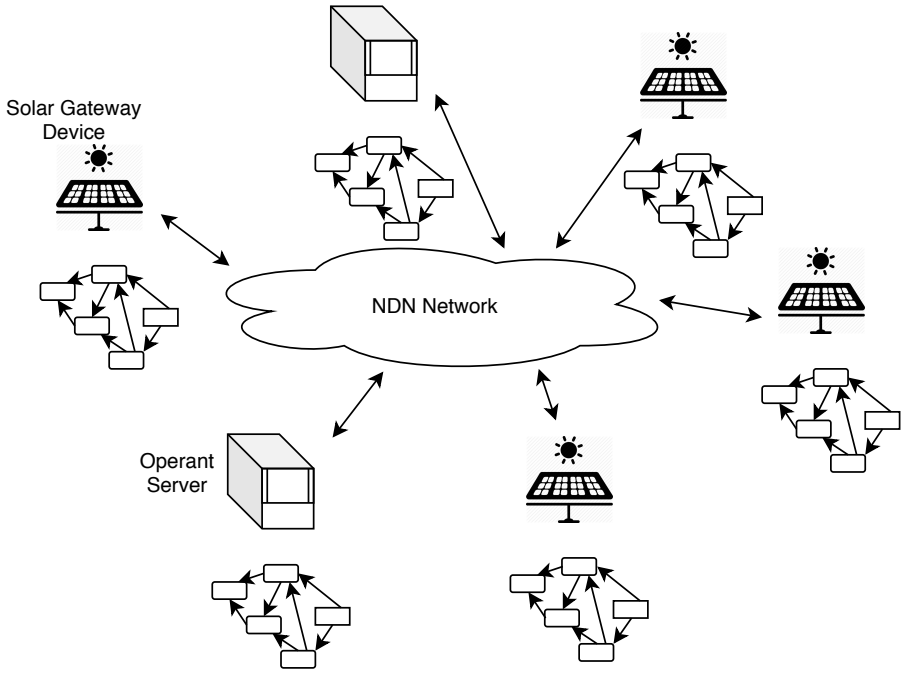


Figure 4.1: An Overview of Operant's DLedger Prototype

- **Customer Nodes:** these customer nodes consist of peer-to-peer network formed by rooftop solar gateway devices as shown in figure 4.1. Every node can append a new record into the ledger system as well as verify other peer's record.

- **Operant Network Operator:** this serves as a trust anchor of the system and also issues certificates. As shown in figure 4.1, the operator can also deploy some servers in the peer-to-peer ledger system network to improve data redundancy of the system by keeping the latest copies of the ledger. Note that, however, these servers just act as "listeners" and do not intervene in operations of DLedger. The operator also bootstraps the distributed ledger by generating a few initial genesis blocks.

The basic structure of DLedger and the design of DLedger Sync Protocol and PoA has remained the same since its initial stage and as described in Chapter 3. However, record appending, verification and acceptance rules were different in this initial prototype developed for Operant compared to DLedger's current implementation. These were similar to IOTA's Tangle, that is, Operant's prototype used the MCMC algorithm for tip selection and accepted a block into the system based on weight threshold and PoA verification. As discussed in section 3.6, this prototype had many security flaws which led to the newer version of DLedger discussed in Chapter 3.

CHAPTER 5

Related Works

5.1 Works over TCP/IP

Besides IOTA described in section 2.3 designed for IoT, there are other works done to achieve distributed ledger technology in IoT network consisting of constrained devices as well as address issues of security and privacy.

Enigma [ZNP15], developed by researchers at MIT, uses Blockchain to create an access-control system to allow users to own and have complete control and privacy over the data. Smart contracts are placed in blocks in Enigma which contain access policies to user's personal data. The data itself is stored and secret-shared among the off-the-blockchain powerful nodes thus allowing scalability and taking off storage load from the Blockchain network participants. To achieve consensus, it uses a Proof-Of-Stake (PoS) [Eni] instead of the popular PoW. Proof-Of-Space (PSpace) [DFK15] is also gaining popularity in IoT distributed ledger systems which uses memory-hard functions instead of computation intensive CPU bound PoW. For example, [YLT18] proposes a new IoT blockchain framework using Ethereum's smart contracts and using PSpace. [PKF15] also leverages PSpace. [DKJ16] uses Beta Reputation System [JI02] to establish decentralized trust model and use trust score to achieve consensus rather than using Proof-Of-X (X = work, space, stake, etc.), thus making less expensive choices for their ledger technology smart home device network. A cryptocurrency IOTW [ANA] built over blockchain claims to be IoT-friendly by using a lightweight Proof-Of-Assignment as the gating control mechanism, getting rid of PoW. A miner, in Proof-Of-Assignment, is selected in two ways. Either a centralized server can randomly pick a miner or each peer use their own keys and predefined seed to calculate randomness. The

randomness with a specific format (for example, based on characteristics of n least significant bits) is then selected.

Moreover, many other works [Van14, ZW15, HCK17, WB14] use the free bytes available in Bitcoin transactions reserved from arbitrary data and build their system over crude Blockchain without any tweaks to required to support IoT (for example, network partition friendly, less expensive gating control mechanism, efficient storage/archiving).

Nano [Nan] and Byteball [Byt] are similar works to IOTA building ledgers over the Directed Acyclic Graph. Nano uses PoW as anti-spam protection while Byteball achieves consensus by forming a single chain called "main chain" within the DAG. The main chain is selected by trusted third-party users called "witnesses".

5.2 Works over NDN

To our best knowledge, not much work has been done in NDN to develop distributed ledgers, especially for the IoT network. Among a few, [JZL17] is one such work which utilizes ChronoSync discussed in Chapter 2 to develop a Bitcoin-like ledger system. As argued in the same chapter, utilizing ChronoSync is a bad choice for a distributed ledger system given its inefficiency (long-lived interest and poor result with simultaneous data generation). [LZQ18] describes public key management infrastructure in NDN over Blockchain.

5.3 Observations

As observed, most of the mentioned related works rely on ""muscle show" consensus and gating control mechanisms such as Proof-Of-Work, Proof-Of-Stake, Proof-Of-Space, etc. Proof-Of-Assignments seems to be IoT-friendly apparently; however, it relies on a central server introducing a single point of failure. The other technique of selecting miner can easily be compromised by leasing out randomness calculation to a modern PC to win the selection.

Moreover, the works relying on the Blockchain at its backbone are not network partition friendly. The DAG based works such as IOTA, Nano and Byteball are not IoT-environment

friendly. For example, IOTA and Nano use lightweight PoW for anti-spam protection. However, malicious IoT device can simply lease out mining operation to a modern PC, thus able to achieve transaction flooding. Note that leasing out mining to modern PC is possible because of anonymity in these systems; peers cannot know who exactly mined the block into the system. Byteball, on the other hand, still ultimately relies on a single chain and claims that network partitions are very unlikely in public network. However, in private IoT network, partitions are very common and the assumption doesn't hold anymore.

CHAPTER 6

Conclusion

In this thesis, we presented an IoT-friendly distributed ledger system, DLedger, developed over Named Data Networking (NDN). DLedger utilizes lightweight Proof-Of-Authentication (PoA) to achieve an IoT-friendly gating control mechanism. Moreover, even with such a lightweight gating control mechanism, it is still bullet-proof against spam attacks. Together with security policies and PoA, DLedger is robust against many security threats and challenges over the current existing distributed ledger systems. Furthermore, NDN's data-centric model allows DLedger to achieve data distribution in a truly efficient and distributed manner. We try to show a different approach to distributed ledger where we combine data openness among the system peers with verifiable identity within the system instead of combining openness with anonymity.

Note that the DLedger system we proposed is designed to be deployed over wireless NDN mesh network which is feasible today in local private area network. Moreover, it can also be deployed in a wide area network with NDN as an overlay over TCP/IP routers. In this case, DLedger won't be able to utilize NDN's network support but is expected to work in similar way as how Bitcoin system works by utilizing Internet Relay Chat (IRC) where each node needs to maintain a list of IP addresses as first-hop neighbors and follow "`gossip`" protocol for achieving communication with all other peers in the system.

REFERENCES

[ANA] ANAPP BLOCKCHAIN TECHNOLOGIES LIMITED. "A Scalable Blockchain Proof of Assignment Protocol." Online; Available at https://iotw.io/docs/IOTW-Whitepaper.pdf.

[AS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. "A survey of peer-to-peer content distribution technologies." *ACM computing surveys (CSUR)*, **36**(4):335–371, 2004.

[Byt] Byteball Team. "An open cryptocurrency platform ready for real world adoption." Online; Available at https://obyte.org/.

[DFK15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. "Proofs of space." In *Annual Cryptology Conference*, pp. 585–605. Springer, 2015.

[DKJ16] Ali Dorri, Salil S Kanhere, and Raja Jurdak. "Blockchain in internet of things: challenges and solutions." *arXiv preprint arXiv:1608.05187*, 2016.

[Eni] Enigma Project. "Secret Nodes: Exploring Staking, Stakeholders, and ENG." Online; Available at https://blog.enigma.co/secret-nodes-exploring-staking-stakeholders-and-eng-d69a68e3d0fd.

[FSK14] Jason Farina, Mark Scanlon, and M-Tahar Kechadi. "Bittorrent sync: First impressions and digital forensic implications." *Digital Investigation*, **11**:S77–S86, 2014.

[HCK17] S. Huh, S. Cho, and S. Kim. "Managing IoT devices using blockchain platform." In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pp. 464–467, Feb 2017.

[IOT] IOTA Team. "Making a Transaction." Online; Available at https://iota.readme.io/docs/making-a-transaction.

[JI02] Audun Josang and Roslan Ismail. "The beta reputation system." In *Proceedings of the 15th bled electronic commerce conference*, volume 5, pp. 2502–2511, 2002.

[JZL17] Tong Jin, Xiang Zhang, Yirui Liu, and Kai Lei. "Blockndn: A bitcoin blockchain decentralized system over named data networking." In *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*, pp. 75–80. IEEE, 2017.

[LZQ18] Junjun Lou, Qichao Zhang, Zhuyun Qi, and Kai Lei. "A Blockchain-based key Management Scheme for Named Data Networking." In *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, pp. 141–146. IEEE, 2018.

[MAZ17] Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. "On the Evolution of ndnSIM: an Open-Source Simulator for NDN Experimentation." *ACM Computer Communication Review*, July 2017.

[Nak08] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system." 2008.

[Nan] Nano Team. "Digital currency for the real world the fast and free way to pay for everything in life." Online; Available at https://nano.org/en.

[PKF15] Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gai, Jol Alwen, and Krzysztof Pietrzak. "SpaceMint: A Cryptocurrency Based on Proofs of Space." Cryptology ePrint Archive, Report 2015/528, 2015. https://eprint.iacr.org/2015/528.

[Pop18] S Popov. "The Tangle, IOTA Whitepaper.", 2018.

[PSF17] Serguei Popov, Olivia Saa, and Paulo Finardi. "Equilibria in the Tangle." *arXiv preprint arXiv:1712.05385*, 2017.

[SAZ17] Wentao Shang, Alexander Afanasyev, and Lixia Zhang. "VectorSync: distributed dataset synchronization over named data networking." In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pp. 192–193. ACM, 2017.

[SBL14] Kallisthenis I Sgouras, Athina D Birda, and Dimitris P Labridis. "Cyber attack impact on critical smart grid infrastructures." In *Innovative smart grid technologies conference (ISGT), 2014 IEEE PES*, pp. 1–5. IEEE, 2014.

[SK17] Julia E Sullivan and Dmitriy Kamensky. "How cyber-attacks in Ukraine show the vulnerability of the US power grid." *The Electricity Journal*, **30**(3):30–35, 2017.

[SYD16] Wentao Shang, Yingdi Yu, Ralph Droms, and Lixia Zhang. "Challenges in IoT networking via TCP/IP architecture." *Technical Report NDN-0038. NDN Project*, 2016.

[SYW17] Wentao Shang, Yingdi Yu, Lijing Wang, Alexander Afanasyev, and Lixia Zhang. "A Survey of Distributed Dataset Synchronization in Named Data Networking." Technical report, Technical Report NDN-0053, NDN, 2017.

[Van14] David Vandervort. "Challenges and opportunities associated with a bitcoin-based transaction rating system." In *International Conference on Financial Cryptography and Data Security*, pp. 33–42. Springer, 2014.

[VK18] Vishrant Vasavada and Randy King. "ndn-ledger." https://github.com/vvasavada/ndn-ledger, 2018.

[VMZ18] Vishrant Vasavada, Xinyu Ma, and Zhiyi Zhang. "ndnSIM-DLedger." https://github.com/Zhiyi-Zhang/ndnSIM-DLedger, 2018.

[WB14] Dominic Wörner and Thomas von Bomhard. "When your sensor earns money: exchanging data for cash with Bitcoin." In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pp. 295–298. ACM, 2014.

[YAM13] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. "A case for stateful forwarding plane." *Computer Communications*, **36**(7):779–791, 2013.

[YAZ15] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. "Name-based access control." *Named Data Networking Project, Technical Report NDN-0034*, 2015.

[YLT18] Yong Yu, Yannan Li, Junfeng Tian, and Jianwei Liu. "Blockchain-Based Solutions to Security and Privacy Issues in the Internet of Things." *IEEE Wireless Communications*, **25**(6):12–18, 2018.

[ZA13] Zhenkai Zhu and Alexander Afanasyev. "Let." In *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10. IEEE, 2013.

[ZAB14] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. "Named data networking." *ACM SIGCOMM Computer Communication Review*, **44**(3):66–73, 2014.

[ZLW17] Minsheng Zhang, Vince Lehman, and Lan Wang. "Scalable name-based data synchronization for named data networking." In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pp. 1–9. IEEE, 2017.

[ZNP15] Guy Zyskind, Oz Nathan, and Alex Pentland. "Enigma: Decentralized computation platform with guaranteed privacy." *arXiv preprint arXiv:1506.03471*, 2015.

[ZW15] Yu Zhang and Jiangtao Wen. "An IoT electric business model based on the protocol of bitcoin." In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pp. 184–191. IEEE, 2015.