Proof-of-Authentication for Private Distributed Ledger

Zhiyi Zhang
UCLA
zhiyi@cs.ucla.edu

Vishrant Vasavada
UCLA
v.vasavada@hotmail.com

Randy King
Operant Networks
randy.king@operantnetworks.com

Lixia Zhang
UCLA
lixia@cs.ucla.edu

***Abstract*—Over the last few years, blockchain-based technologies have flourished in many application areas. One of them is the creation of distributed ledgers where records of immutable objects are widely replicated for both transparency and availability. However, the Proof-of-Work (PoW) approach, a popular gating control that determines who can add new records into a ledger, is deemed infeasible for IoT devices with resource constraints.**

In this paper, we present the design of DLedger, a private distributed ledger system designed for an experimental solar network developed by Operant Networks. DLedger records both individual customers' solar energy production/consumption as well as all other noteworthy system events, such as certificate issuance and revocations. Compared to today's centralized record keeping solutions, DLedger brings the benefits of information transparency and availability to both customers and the system vendor. Operant's solar network uses the Named Data Networking (NDN) protocol, based on which DLedger controls the addition of new records using a lightweight Proof-of-Authentication (PoA). PoA leverages the properties of NDN where (i) every entity in the system possesses a name and a digital certificate, and (ii) they share the same trust anchor and thus can authenticate each other. DLedger further leverages NDN's data-centric design to keep the ledger synchronized in a truly distributed and efficient manner.

I. Introduction

This work is inspired by, and designed for, an experimental solar network developed by Operant Networks Incorporated (hereafter called Operant). In Operant's solar network, customer's appliances are equipped with the solar gateway device, which communicates through LoRa wireless channels and uses Named Data Network (NDN) [9] as its network layer protocol. Solar gateway devices generate the records of energy production and consumption of individual customers, and customers desire to see their data not only be reliably and securely recorded, but also be available for their own viewing.

Cryptocurrencies like BitCoin have shown that financial transactions can be stored in a trusted and consented way through a decentralized network of peers using a public distributed ledger. However, the Proof-of-Work (PoW) approach, a popular gating control that determines who can add new records into a ledger, is considered infeasible for Internet of Things (IoT) such as the Operant's solar gateway devices. Furthermore, synchronization of a truly distributed ledger requires the devices exchange secured application records among each other *directly*, yet the existing TCP/IP protocol stack lacks adequate support for group communications; consequently overlay protocols like Internet Relay Chat (IRC) [6] are used for group communication, introducing extra overhead and configuration complexity.

In this paper, we present a preliminary design of DLedger, a private distributed ledger system designed for Operant's NDN-based solar network. Compared with the blockchain-based distributed ledger used in BitCoin:

- DLedger utilizes the Tangle data structure inspired by IOTA [7] instead of the blockchain.
- DLedger takes Proof-of-Authentication (PoA) as the gating function for records in DLedger and PoA is readily realized over the built-in security support by NDN [10].
- DLedger leverages NDN's data-centric solutions to keep the ledger synchronized in a truly distributed manner.

In the rest of this paper, we introduce the background in Section II, present the design of DLedger in Section III, describe the proof-of-concept implementation for Operant Network in Section IV, discuss the DLedger's design in Section V, list the unsolved issues and future work in Section VI, and conclude our work in Section VII.

II. Background

A. IOTA

IOTA [4] is a cryptocurrency formed over a distributed ledger technology called Tangle, which is different from conventional cryptocurrency systems like Bitcoin where underlying data structure is based on blockchain. Tangle is fundamentally a *Directed Acyclic Graph (DAG)* used for storing transactions. IOTA claims to be able to work with Internet of Things (IoT) because of their feeless transactions and low resource requirement. Rather than following a current practice that miners incorporate transactions into blocks and attach them to the ledger, users in IOTA behave both as creators and miners of blocks. Whenever a user wants to add a new block to the ledger, they must verify two previous blocks in the system as well as perform a small Proof-Of-Work (PoW) computation.

In IOTA's Tangle, each vertex represents a block in the ledger system, while an edge represents the approval relation

Workshop on Decentralized IoT Systems and Security (DISS) 2019
24 February 2019, San Diego, CA, USA
ISBN 1-891562-56-8
https://dx.doi.org/10.14722/diss.2019.23003
www.ndss-symposium.org

between blocks. For instance, block A in Figure-1 approves blocks B and C directly, and approves D, $E$and F indirectly. Block F is the root of the DAG, and is called a *genesis* block. A and other tailing blocks are called *tips*: they have not been approved by any block (i.e., there is no incoming edge to these blocks). A block is confirmed when it has been referenced by a special block called milestone. Milestone blocks are generated by a centralized entity called coordinator, which is run by the IOTA foundation[1].

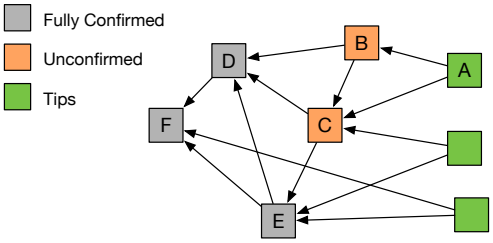

Fig. 1: IOTA Tangle

In addition to containing two pointers to previous blocks and its own payload (actual block content), a block also carries a weight. A block's weight is the total number of both direct and indirect approvals it has. Block weight is used in IOTA's *Tip Selection Algorithm* which decides which two tips to approve. To be specific, IOTA takes a weighted random walk algorithm called Markov Chain Monte Carlo (MCMC): the walk starts from some ancient blocks (e.g., genesis blocks) and ends at a tip block, where in each step, a linked following block with higher weight gets a bigger chance to be the next hop.

B. Named Data Networking

Named Data Networking (NDN) makes the named data a first class entity in the network architecture. To be more specific, applications name their data at the application layer and NDN uses application data names directly for network layer delivery. In an NDN network, routing and forwarding are based on data name prefixes.

Instead of *pushing* packets to destination addresses as in an IP network, NDN supports data *pull* model where consumers fetch data from the network via request/response exchange - a request, called an *Interest* packet, carries the name of a data piece consumers would like to fetch; the response, called a *Data* packet, carries the actual response data (Figure 2).

NDN utilizes a stateful forwarding plane: when forwarding an Interest packet, the network will record the path of the Interest; the fetched Data packet will reversely follow the Interest patch back to the consumer. Since NDN packets identify data instead of locations, (i) multiple Interest packets targeting the same piece of data can be aggregated in the network, and (ii) the fetched data can be cached along the path to satisfy future Interest packets asking for the same data.

To ensure the integrity and authenticity of retrieved data regardless from where the data is returned (e.g., from the original producer or from in-network cache), NDN builds security in network architecture by requiring data producers to cryptographically sign all data packets at the time of production. Data carrying sensitive content can also be encrypted as needed.

[1]IOTA foundation claims that they will remove the coordinator in the future. Without milestone blocks, a transaction will be confirmed when it is referenced by all tips.

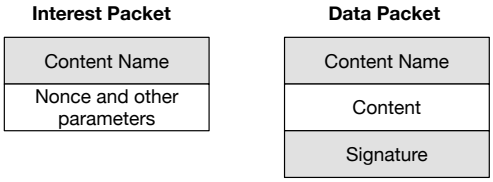

Fig. 2: Interest and Data packets, neither carries source or destination address.

III. DLedger

DLedger is designed to work over an NDN network, where we assume that individual entities in the system have already performed security initialization [10], that is, they have established their trust on a shared anchor node and each of them has obtained an NDN certificate issued by the trust anchor. In the Operant's solar network, a system operator sets up a trust anchor, and each solar device trusts system operator's digital certificate, and obtains an NDN certificate issued by the anchor. Therefore all the solar devices can authenticate each other, while any outsider without a valid private key cannot pass the verification.

DLedger aims to provide a secure and highly available ledger to keep permanent, immutable and complete records of the solar network's operations. Any and all operations that are related to the updates of the ledger will be recorded in the system.

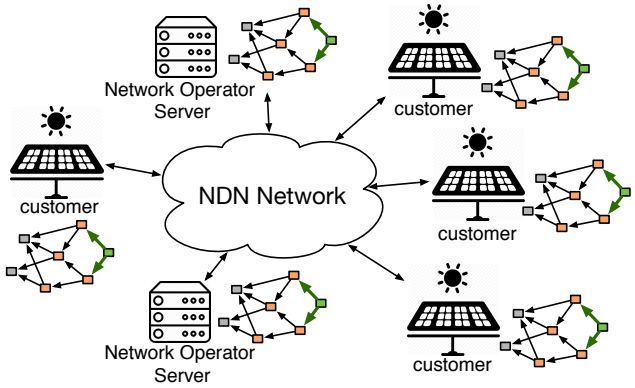

Fig. 3: An Overview of DLedger

As shown in Figure 3, in DLedger, there are two main types of entities.

- **Customer nodes**: all customer nodes form a peer-to-peer network. Every node can append new record data into the ledger system and at the same time, verify other peers' record.
- **Network Operator**: A network operator serves as the trust anchor of the system. Certificates issued by the operator to each customer will also be inserted into the ledger as

a block. The system operator will bootstrap the distributed ledger by creating genesis blocks into the Tangle. Moreover, the system operator can also deploy servers into the peer-to-peer network to improve the data redundancy of the system by keeping the latest copies of the ledger.

Note that after the bootstrapping, network operator's servers do not intervene the operations of DLedger – they act as pure "listeners".

DLedger's design leverages IOTA's idea of storing each and all data records in the Tangle. Each peer in the P2P network maintains a copy of the tangle. However, different from IOTA's tangle where each block is an application-layer data block which is then encapsulated in an IP packet when transmitting at the network layer, in DLedger, each block is an NDN Data packet, which is created by the application and then passed to the NDN network layer for delivery as is, with no further encapsulation. As shown in Figure 4, DLedger uniquely identifies each block by the block name, which also names the NDN Data packet; a block name follows the naming convention as described in Section III-C. The references that contain the names of two tip blocks being approved and the payload of this new block being created make the content of this Data packet. As an NDN Data packet, one can fetch any block by sending an Interest packet carrying the block's name.

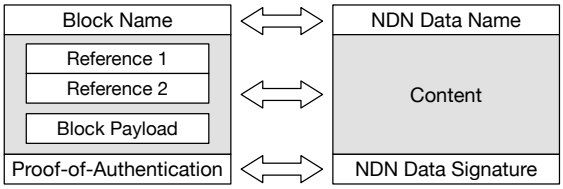

Fig. 4: Block and NDN Data packet

The following steps show how a new block is appended into the distributed ledger system. (i) When a peer generates a new block B_{new}, it first runs the MCMC tip selection algorithm to select two existing tip blocks and verifies their validity. (ii) It then packs its data record into an NDN Data packet, together with the two reference pointers, and adds a PoA. (iii) The peer then utilizes DLedger's notification protocol to notify other peers in the system about this new block B_{new}. Other nodes in the system will fetch and check B_{new} and accept it into their own copy of the Tangle. (iv) When a sufficient number of blocks have successfully verified B_{new}, B_{new} is considered to be accepted by the whole system.

A. Proof-of-Authentication

In DLedger, instead of using PoW or other hashcash-based mechanisms, we use the PoA for peers to decide whether a block is valid or not. Each block's PoA is a digital signature. The PoA works as follows:

1) When a peer generates a new block B_{new} as described above, it appends a PoA by signing B_{new} using its private key.
2) After other peers fetch B_{new}, they will verify the PoA by validating the digital signature and consider B_{new} to be valid only if (i) it can be verified using an anchor-certified public key and (ii) the certificate of the public key is recorded in the distributed ledger.

As we described in Section II, NDN requires that every data producer cryptographically sign each Data packet it generates at the time of the production. As shown in Figure 4, since a block is essentially an NDN Data packet, its signature sufficed as a *PoA*. Therefore, the PoA is in fact already provided by the basic NDN data packet design.

B. Distributed Synchronization over NDN

Another design feature that set DLedger apart from other existing distributed ledger systems is its data-centric synchronization protocols.

When a node generates a new block, it needs to notify other nodes so that they can get this new block and update their local ledgers. In DLedger, we leverage NDN's inherit multicast data delivery feature to facilitate the above process:

1) The B_{new} generator multicasts an Interest packet I_{notif} called a Notification to notify all the peers who have registered the same multicast prefix (i.e. expressed interest in receiving Notification of new block generations). As shown in the next subsection, this Notification Interest name bears a hint for other nodes to compose the name of the newly generated block.
2) The receivers of I_{notif} will send out an unicast Interest packet to fetch the new record block back. After fetching the B_{new}, the node will verify its PoA and decide whether to put it into the local ledger.

Now and then there can be transient network failures, nodes falling into sleep to save power, or other events that may eventually cause the local state of peer nodes out of sync. Therefore they need to synchronize their local state with the latest version of the Tangle structure. DLedger utilizes a data-centric synchronization protocol over NDN to facilitate this process.

1) A node multicast a Sync Interest I_{sync} to the network to trigger a synchronization process. I_{sync} carries a list of tip block names in the node's current local ledger view.
2) When another node receives I_{sync}, it compares the tips carried in I_{sync} with the tips in its local ledger, identifies what blocks it may be missing by recursively tracing the references in each block. If it misses any tips or even ancestor blocks of tips, it will retrieve the missing blocks by sending Interest packets carrying the names of the missing blocks.
3) When a block fetching Interest packet reaches a neighbor node, if that neighbor has the desired block, it will send a reply with the requested Data packet. Operant's NDN Geo-forwarding plane [2] has built-in Channel Activity Detection and collision avoidance, so that not all the nodes who have the missing block will reply at the same time, and one reply can suppress others. When none of the neighbors has the missing block, they will forward the request Interest, and the eventually retrieved Data block will also update their own local ledgers.

A peer sends an Sync Interest periodically, or when any of the following events happen: (i) The node is recovered from

a network failure, a sleep, or any other conditions that may cause the Tangle out of sync. (ii) The node receives another Sync Interest carrying a outdated tip list.

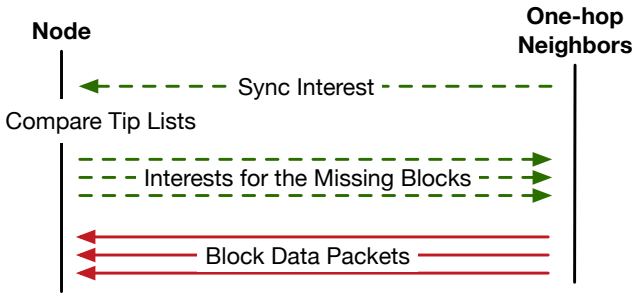

Fig. 5: Synchronization

When a peer-to-peer network is small in size, the Sync Interest can be multicast to all the peers. In a large-scale network, instead of multicasting Sync Interest globally, DLedger can set a *hop limit* on multicast Sync Interest; if one set the limit to one, then only direct neighbor nodes will hear the Interest and will not re-multicast the packet. In this way, as shown in Figure 5, the synchronization process only happens among one-hop neighbors, and all the changes eventually propagate to all the nodes in a staged manner of one-hop at a time.

C. Naming Conventions

DLedger utilize naming conventions to automate the system processing. To be specific, following the pre-defined naming convention, a peer knows how to (i) construct a Notification Interest, (ii) construct a Sync Interest, and (iii) assemble a block name using the information extracted from the tip list.

Each block in DLedger has a name like:

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

An example of a block name could be "`/ndn-dledger/solar-gtw-123/35a...5cc598`". The producer prefix (e.g., "`/solar-gtw-123`") help peers to know the generator of the new block. The hash (e.g., "`/35a...5cc598`") is the digest of the record, adding uniqueness to the block name. In DLedger, each peer is supposed to register a prefix "`/<multicast prefix>`" and a prefix "`/<multicast prefix>/<peer prefix>`". Therefore, when a peer requests a missing block using the block name as the Interest packet, this Interest can be accepted by all the peers. Moreover, the network can use longest prefix and forward the Interest to block's original generator when the Interest cannot be satisfied by the neighbor peers.

Notification Interest follows the naming convention:

"`/<multicast prefix>/NOTIF/<creator prefix>/<hash>`"

By extracting the producer prefix and the record hash, other nodes can directly compose the name of the new block by removing the "`NOTIF`" component. The name of the block is sufficient for a peer to generate an Interest packet and fetch the block back. As an simple example, a Notification Interest whose name is "`/ndn-dledger/NOTIF/solar-gtw-123/35a...5cc598`" indicates that the name of the new block is "`/ndn-dledger/solar-gtw-123/35a...5cc598`". A Notification Interest packet is a mulitcast packet at the network layer: all the nodes will eventually receive the packet and issue an unicast Interest to fetch the new block.

A Sync Interest carries a list of tip names as the Interest parameters [5]. The name of the Sync Interest packet is in the format of:

"`/<multicast prefix>/SYNC/<digest of tip name list>`"

IV. Implementation

We used Node.js for the proof-of-concept implementation of DLedger system. This DLedger system prototype was built over NDN's ndn-js library [8], which is also written in Node.js. As next step in this project, we will perform simulations to evaluate the data structure and algorithms used in DLedger. We also plan to integrate the system into NDN IoT package [12] and quantitatively evaluate our system performance with constrained IoT devices.

A. System Bootstrapping

Before the system starts, a system operator creates a number of genesis records into the ledger and all these genesis records are signed directly by the network operator's trust anchor key. These genesis blocks can be saved in the first several peers and in the servers. Whenever a new node joins the system, the node first obtains a certificate issued by the operator either manually or through automated certificate issuance mechanisms such as NDNCERT [11]. After that, the new node will copy the current Tangle to the local by sending out a Sync Interest packet which carries an empty tip list. As mentioned in the Section 3, nodes who have the latest copy of the Tangle will send out a Sync Interest carrying the latest tips.

B. Tangle Synchronization Process

We implemented DLedger's synchronization algorithm in our proof-of-concept implementation. When a node receives list of tips from other node as a result of Sync Interest, it compares with the tips in its local ledger as discussed in Section III. LedgerSync function in Algorithm 1 describes the procedure. For each tip that is in the list received but not the local ledger, the receiver will compose a Interest using the block name to fetch this missing block. If the tips received are in the local ledger but are no longer tip blocks, which means the sender of the tips has an obsolete Tangle, and the receiver will send out a Sync Interest carrying the local tips.

The recursive fetching of missing blocks is explained by onReceivingMissingBlock function in Algorithm 1. When a node receives a missing block B_{miss} that it requested for, it checks whether two blocks B_{miss} references are present in ledger. If not, it sends out Interest to fetch these missing referenced blocks. It adds the block it received to pendingAttaches stack. When it is done fetching all the missing blocks recursively, it will pop each block from this stack, verify and attach to the ledger.

Algorithm 1 Ledger Synchronization

```
function onReceivingSyncReq(receivedTips)
    for each tip in receivedTips do
        if !(ledger.contains(tip)) then
            fetchRecord(tip)
        else if !(localTips.contains(tip)) then
            isOutDatedSync = true
        end if
    end for
    if isOutDatedSync then
        sendSyncRequest(localTips);
    end if
end function

function onReceivingMisstingBlock(block)
    if !ledger.contains(block.reference1) then
        fetchRecord(block.reference1)
    end if
    if !ledger.contains(block.reference2) then
        fetchRecord(block.reference2)
    end if
    pendingAttaches.add(block)
end function
```

C. System Overhead

Overhead of Tip Selection. Note that the approvals in Tangle go from newly generated blocks towards the existing ones. However, in MCMC tip selection algorithm, we need to walk from genesis towards tips where there are no pointers to follow. Hence, it will be necessary to parse the entire chain from tip to genesis first in memory so that it could be walked backwards. Our proof-of-concept implementation mitigates this overhead by maintaining list of approvers for each record in the database. Hence, during tip selection algorithm, we directly start walking from genesis block and for each block, get this list of approvers and choose an approver with *weighted* probability.

Overhead of PoA. In our implementation, a peer needs to generate PoA whenever a new block is generated and verify a PoA every time a new block is receiving. Since PoA is essentially Data signature of the NDN Data packet, the PoA of the DLedger does not add extra overhead in terms of cryptographic operations.

Overhead of The Network Protocols. Since there are small number of nodes in our proof-of-concept system, we simply multicast all the Interest packets (i.e., Notification Interest, Sync Interest, and block fetching Interest) to the whole P2P network. Note that a block fetching Interest stops when it hits a matching block (from cache of intermediate nodes or from the block producer). As mentioned in Section III, during the Tangle Synchronization process, Sync Interest and block fetching Interests can be limited to one-hop scope in order to reduce network overhead.

V. Discussion

A. Proof-of-Authentication versus Proof-of-Work

PoW provides a way for nodes in an *anonymous* peer-to-peer network to reach consensus on who can add new blocks into the blockchain. The huge expenditures of PoW also greatly lower the chance of denial of service (DoS) attacks and increase the difficulty of cheating (e.g., double spend). At the same time, PoW leads to huge consumption of energy and limits the scalability of the system: the speed of PoW calculation becomes the bottleneck when a large number of pending blocks need to be added quickly. Although the difficulty of PoW in IOTA has already been largely reduced, it can still be infeasible for a constrained device to directly finish a PoW task; a recommended practice of using IOTA in IoT is to deploy a dedicated node that constrained devices can delegate the PoW calculation to.

In Operant's use case and other similar scenarios where nodes must be identified (e.g. to enable power companies to attribute energy credits to users), it is much cheaper to use PoA as the gating function. DLedger allows any legitimate node to append new blocks into the tangle and PoA utilizes the public key cryptography to ensure the block is from a legitimate member of the system. PoA makes DLedger feasible to deploy in IoT systems because digital signatures can be efficiently generated and verified even by constrained devices.

B. Distributed Ledger over NDN versus over TCP/IP

Building peer-to-peer network over TCP/IP is nontrivial since TCP/IP network provides point-to-point communication *channels* while the nature of peer-to-peer network benefits from broadcast/multicast. To enable multicast, BitCoin and other distributed ledger systems pay a high overhead; for instance, BitCoin adopts a gossip mode for the purpose of broadcast, where each peer needs to maintain i) a list of IP addresses that currently connects to the network [1] and ii) a number of TCP connections required by the gossip protocol [3].

Since an NDN Interest packets do not carry destination information, it can be easily forwarded to multiple next hops as needed. Therefore, Interests can be broadcast or multicast easily supported with forwarding strategies [9], without using overlay protocols (e.g., the gossip protocol). Instead of pushing a new block to all the peers, DLedger's notification protocol works in a pulling way–all the peers who received the notification issue an Interest to fetch the block, and retrieve blocks are cached at intermediate nodes, and later Interest packets directly fetch the blocks from the intermediate node. Moreover, multiple Interest packets asking for the same data will be aggregated as a single Interest, reducing the network overhead.

C. Preventing Intervention of the Anchor Node

Network operators deploy trust anchor nodes of the system, controls who can join the system by issuing them a certificate. One potential attack scenario is that the anchor node can issue certificates to many virtual nodes controlled by the anchor node. These bots can then easily approve an invalid block and let the block gain enough weight to be accepted.

DLedger can mitigate this attack with data publicity. As described in Section III, all the certificate issuance events will be recorded in the system. Certificates issued by the trust anchor but not recorded in the ledger will not be accepted during PoA verification. In this way, a network operator

cannot manipulate the distributed ledger by hiring bots without recording them into the ledger. Any system intervention by the anchor node will be recorded in the ledge and used for later examination.

VI. Remaining Issues and Future Work

The DLedger design is still in a preliminary stage, a number of issues that related to system security and robustness are identified and yet to be resolved. In this section we listed these issues and discuss potential solutions as our future work.

A. The Size of the Tangle

The size of the Tangle is linearly porportion to the number of records stored in it. As the system runs over time, the size of the Tangle will go larger, requiring more memory for tip selection process (walking from genesis to tips) and more storage to keep the entire Tangle. This can potentially lead to high overhead for constrained devices with limited memory and storage.

IOTA utilizes a centralized mechanism for snapshot: a global trusted party called coordinator will lead the effort of snapshot and create a new block containing the current status of the whole system, after which all clients in the system need to manually claim the coins from the coordinator. However, this solution is insufficient for DLedger where IoT devices like solar boards are equipped without direct user interfaces and is inconvenient to operate.

In our future work, we will introduce snapshot into the system to reduce the size of the Tangle but in a automatic manner without relying on a centralized coordinator.

B. Tip Selection Algorithm Efficiency

MCMC, the current tip selection algorithm used in DLedger, is inefficient in terms of memory use and time consumption. This is due to the fact that in Tangle, the directed edge is pointed from new blocks to old blocks while the MCMC algorithm walks from old blocks towards tips. Therefore, to realize MCMC, the system either needs to maintain the approvers of each block by updating the database whenever there is a new arriving block or to parse the whole Tangle into the memory before performing MCMC. We leave it as our future work to develop a new tip selection algorithm which can reduce this cost. A possible direction is to maintain the tip list and perform random selection among tips instead of walking from the genesis blocks to tips.

C. Denial of Service

Since signature signing is efficient even for constrained devices (e.g., ECDSA hardware chip ATECC508A is less than one dollar), it is possible that a malicious peer can append large number of blocks into the ledger system. Such an attack may lead to an invalid block gaining enough approvals and become accepted by the whole system and ruin the data integrity of the ledger. We leave it as our future work to mitigate such attack. A possible solution is to add checking rules on newly generated blocks after receiving a Notification Interest; for example, a peer can refuse to fetch a new block if it found the new block and its approved block are generated by the same node.

D. Collusion of Peers

The current design cannot prevent the collusion attack: two or more peers can collude and help approve each other's invalid blocks to get those blocks accepted by the ledger system. The system is supposed to accomplish a k/n $(k < n)$ level of security: if no more than k peers collude, the collusion will not hurt the system security. In our future work, we plan to apply security rules to the block confirmation process. Peers in the system can also perform periodic tangle walk through and check the block additions against the known system properties, for example, the limitation on how much energy one can produce in a given time period.

VII. Conclusion

This paper presents the design of DLedger, a private distributed ledger systems for an experimental solar network system. Different from the popular distributed ledger system used for cryptocurrenies, DLedger adopts efficient PoA instead of the PoW to confirms the validity of the blocks. Our proposed ledger system adopts a data-centric design: (i) We build the notification protocol and synchronization protocol over NDN to utilize NDN's inherent multicast support and data distribution feature, reducing the content fetching latency and improving the bandwidth utilization. (ii) NDN's Data packet format confirms DLedger's block data structure and we directly use NDN's Data signature as the PoA's implementation.

Our work is still in its early stage. However, our prototype implementation already shows that PoA is sufficient for a private ledger system and data-centric network infrastructure provides better network support for the distributed ledger system.

References

[1] Bitcoin Wiki Authors. (2018) Bitcoin network protcols. [Online]. Available: https://en.bitcoin.it/wiki/Network

[2] G. Grassi, D. Pesavento *et al.*, "Navigo: Interest forwarding by geolocations in vehicular named data networking," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE*. IEEE, 2015, pp. 1–10.

[3] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

[4] Members from IOTA Foundation. (2018) Iota foundation. [Online]. Available: https://www.iota.org/

[5] NDN Developers. (2018) Ndn packet format specification 0.3. [Online]. Available: http://named-data.net/doc/NDN-packet-spec/current/

[6] J. Oikarinen and D. Reed, "Internet relay chat protocol," Internet Requests for Comments, RFC 1459, 1993, http://www.rfc-editor.org/rfc/rfc1459.txt.

[7] S. Popov, "The tangle," *cit. on*, p. 131, 2016.

[8] W. Shang, J. Thompson *et al.*, "Ndn. js: A javascript client library for named data networking," in *Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 399–404.

[9] L. Zhang, A. Afanasyev *et al.*, "Named Data Networking," *ACM SIGCOMM Computer Communication Review*, 2014.

[10] Z. Zhang, Y. Yu *et al.*, "An overview of security support in named data networking," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 62–68, November 2018.

[11] Z. Zhang, A. Afanasyev, and L. Zhang, "Ndncert: universal usable trust management for ndn," in *Proceedings of the ACM ICN, 2017*. ACM, 2017, pp. 178–179.

[12] Z. Zhang, E. Lu *et al.*, "Ndnot: A framework for named data network of things," *Proceedings of the ACM ICN, 2018*, 2018.