Tamperproof Provenance-Aware Storage for Mobile Ad Hoc Networks

Danny Adams, Gloire Rubambiza, Pablo Fiori, Xinwen Wang, Hakim Weatherspoon, Robbert van Renesse
Department of Computer Science, Cornell University

***Abstract*—This paper presents a middleware for providing a mobile ad hoc network with tamperproof provenance-aware storage, even when some fraction of devices can be Byzantine. Important considerations include fast propagation of updates, data consistency, and low power consumption. Leveraging entanglement techniques from blockchain protocols but carefully avoiding high power consumption and reliance on continuous network connectivity, we design new distributed data structures that can support useful distributed applications such as emergency response and IoT networks. Using both trace-based simulations and experiments with an Android-based prototype, we demonstrate the practicality of such a middleware.**

I. Introduction

Recently, there is renewed interest in mobile ad hoc networks, given new applications that cannot rely on continuous internet availability. Such application areas include emergence response, self-driving cars (Vehicular Autonomous Networks), distributed robotic applications, digital farming, and the Internet-of-Things (IoT). Such applications require a convenient and secure communication and storage infrastructure. It is also often desirable that such systems support *accountability*, so that events can be traced the causes of actions can be attributed [34].

Blockchains are attracting considerable research attention as a distributed computing model with the potential to revolutionize finance, transport, and supply chain management [3]–[5], [7], [20], [22], [29], [32]. Blockchains are desired for their decentralization, consistency, and tamperproof properties and seem at the face of it ideal to address the concerns raised above. However, existing blockchain protocols heavily use power and networking resources by design. The heavy use of power prevents deployment on IoT devices while the heavy use of networking prevents deployment in partitionable networks.

This paper introduces a new blockchain design called *Wayfinder* designed specifically for mobile ad hoc networks. The blockchain's integrity is based on a combination of Conflict-free Replicated Data Types (CRDTs) [28] and a guaranteed "happens-before" order on transactions that operate on one or more CRDTs. CRDTs allows continuous availability despite network partitions. Moreover, Wayfinder guarantees that transactions are tamperproof—once a correct device has acted on transaction it can no longer be lost.

Wayfinder is not suitable to supporting cryptocurrencies: it can detect but not prevent double-spending. However, we believe that for many applications the CRDT consistency properties combined with tamperproofness is sufficient. While Wayfinder only provides a partial "happens-before" ordering on operations, Wayfinder can be used to hold devices and their users *accountable* because blocks are tamperproof. Through accountability, Wayfinder incentivizes users to behave well. Also, the properties of CRDTs prevent inconsistencies in the data structures used by applications.

The contributions of this paper include the following:

- Design of a middleware for providing manet devices in low-power and partitionable network environments with tamperproof provenance-aware storage, even when some fraction of devices can be Byzantine.
- Analysis of the system design that allows Wayfinder to operate in low-power and network partition-tolerant environments on many different hardware configurations.
- Experiments evaluating performance differences between a mining-based protocol and Wayfinder in low-power and network partitionable environments.
- Discrete event time simulations using realistic traces that evaluate the scalability of Wayfinder.

The remainder of the paper is organized as follows. In Section III, we describe the design of Wayfinder, especially its tamperproof provenance-aware storage that can tolerate a fraction of Byzantine devices. We describe Wayfinder's implementation in Section IV and applications built and deployed using Wayfinder. In Section V, we evaluate the scalability of Wayfinder and compare its performance to a mining based approach. We discuss related work in Section VI and conclude in Section VII.

II. Motivation

Today's blockchain protocols are resource-hungry and manet devices generally are resource-constrained. So-called permissionless blockchains use much energy. Individual participants ("miners") have to continuously work to attempt to solve so-called cryptopuzzles. Bitcoin currently consumes 66.7 TeraWatt-Hours in aggregate per year, about the rate of consumption of the Czech Republic, a modern country with over 10 million citizens. 60-80% of revenue from Bitcoin goes back to paying the electricity that it consumes and is pure waste, not to mention the greenhouse gases produced. Even if manet devices had access to a cheap source of continuous energy (say wind or solar) and could compete with the ASIC devices that Bitcoin miners deploy, their participation in a permissionless blockchain would further explode the amount of energy waste and possibly pollution.

Both permissionless and permissioned blockchains require significant networking resources. Typically every transaction has to be stored by at least half of the participants before it can become finalized and actionable. But this knowledge dissemination requires a quadratic communication load, as each participant has to learn if most of the other participants has stored the information. This is slow and scales terribly. Permissionless blockchains typically use a gossip network for this, while permissioned blockchains, based on Byzantine consensus or voting protocols, will have to use some kind of dissemination tree to be able to scale to more than a few hundred participants. But even so, the protocols would likely not scale to more than a few thousand participants, well shy of the potential number of manet devices.

Blockchains do not only consume a lot of network bandwidth and require that the network is available continuously. In the case of permissionless blockchains, network partitions could lead to so-called "forks" in which the blockchain splits into multiple separate chains and inconsistencies would arise that are difficult to reconcile. Also, such forks could drastically reduce the rate of transactions, as in each partition there would be fewer participants to solve the cryptopuzzles while the difficulty of those puzzles take significant time to adjust. In the case of permissioned blockchains, a network partition that isolates more than a third of participants would bring the blockchain to a halt.

Unfortunately, manet devices are often deployed in challenging conditions and may even be mobile, and their connectivity may be intermittent. In a blockchain, the rate of transactions is generally independent of the number of participants in the blockchain. In Bitcoin, if you double the number of miners, the transaction rate remains constant but the difficulty of the cryptopuzzles increases. So even if that doubles the amount of energy waste and quadruples the load on the network, the rate of storage growth remains the same. However, each manet device may generate data, and all this data can be tremendously useful. If you double the number of manet devices, you double the rate of data generated. Ideally all this data should be stored for longitudinal studies. Moreover, the data may be needed long-term for chain of custody.

This paper demonstrates a design, implementation, and initial evaluation of a blockchain specifically for the low-connectivity, low-power, high data rate manet setting. The blockchain techology should be tamperproof, but it should also tolerate network partitions well and use a low-power consensus mechanism. Instead of resolving forks, it will need to permit them, resulting in a Directed Acyclic Graph (DAG) structure of the blockchain rather than a linear one. There are other blockchain designs that have embraced a DAG structure. However, they do so to increase the transaction rate, not to tolerate partitions. The cost of this partition tolerance is that the types of applications that can be implemented with the blockchain are limited to ones that only require a partial ordering of logged events. To this end, we will explore applications based on Conflict-free Replicated Data Types (CRDT) that can work with partial orders.

III. System Design

In Wayfinder, devices store state and communicate through a shared storage layer. Because the network is partitionable but consistency is still desired, Wayfinder embraces Conflict-free Replicated Data Types (CRDTs) [28]. A CRDT has the property that two states of a CRDT can be merged into a new CRDT with intuitive semantics. A good example of a CRDT is an append-only set. Elements can be added concurrently in partitions of a network, and upon reconciliation the sets can be merged by taking the union of the sets. There is an extensive variety of CRDTs defined in the literature.

Wayfinder maintains a partial ordering between operations and can define CRDTs that exploit this partial ordering. A Wayfinder CRDT (WCRDT) is an object whose state is uniquely determined by a partially ordered set of operations. Wayfinder maintains a *block DAG*, that is, a Directed Acyclic Graph of blocks. Each block contains a transaction, which is a sequence of operations on possibly multiple WCRDTs. The order of operations within a block and the directed edges between blocks encode the partial order between all operations. More precisely, an operation o_1 on a WCRDT is before another operation o_2 if and only if:

- o_1 and o_2 are part of the same transaction (and therefore in the same block) and o_1 comes before o_2 in the transaction; or
- o_1 and o_2 are in different transactions and there is a path of the block containing o_2 to the block containing o_1.

Therefore, the Wayfinder block DAG (WBD) exactly determines the state of each WCRDT stored in it.

The WBD is maintained by a collection of devices, but not all of them are expected to behave correctly—they may crash or even deviate from prescribed protocols arbitrarily (i.e., Byzantine behavior). Maintaining the integrity of the WBD is therefore a major challenge. Devices can try to create cycles in the block DAG or add new blocks relentlessly and create a block DAG that wildly branches instead of approximating a linear blockchain. Another complexity is how to ensure *tamperproofness*: blocks should never get dropped from the WBD.

To manage the complexity of Wayfinder, it has three tiers of abstraction. From top to bottom, these tiers are:

1) **The Application Tier** provides an interface for applications to create and use smart contracts (Section III-A2). This layer contains a CRDT library including a new variant of 2P sets (Section III-A1).
2) **The Block DAG Tier** maintains the WBD with byzantine fault tolerance.
3) **The Reconciliation Tier** gives an efficient reconciliation algorithm among devices.

A. Application Tier

CRDTs enable devices to independently update their states without any remote synchronization and guarantee consistency as long as their concurrent operations commute.

1) 2P+ Sets: A simple example of a CRDT is a *2P-Set* [28]. It is implemented by two append-only sets: an ADD set and a REMOVE set. The set difference between these two sets determines the current state of the CRDT state machine. Note that in a 2P-set, once an element is removed, it can never be added again.

Leveraging causal relationships between operations, we refine the notion of a 2P set and introduce a new CRDT, a *2P+* set. When an add and delete on the same element are executed concurrently in normal 2P sets, then delete wins. However, if an add happens causally after a delete of the same element, the element is added back to the set, unlike a regular 2P set. This can be formalized as follows: with each $+x$ (add x) and $-x$ (remove x) operation in the causal graph of 2P+ operations (where each operation may depend on a set of other operations), we associate an *add* set and a *delete* set. They are defined as follows:

$$+x.add = \bigcup_{t \in +x.deps} t.add \cup \{x\}$$

$$+x.delete = \bigcup_{t \in +x.deps} t.delete \backslash \{x\}$$

$$-x.add = \bigcup_{t \in -x.deps} t.add \backslash \{x\}$$

$$-x.delete = \bigcup_{t \in -x.deps} t.delete \cup \{x\}$$

The first of these specifies that the add set after some particular `add(x)` operation is the union of the add sets of its ancestors and x itself. The others are defined similarly. The application-visible content of the 2P+ set after an operation is then the difference between its add set and its delete set.

The 2P+ can be further generalized in various ways. For example, it is trivial to define a 2P$-$ set in which addition wins instead of delete to resolve concurrent conflicting operations. Going further, we can define nP+ and nP$-$ sets for values of n $\geq$ 2. One can think of these as prioritized sets, with n priorities. The priority of elements can be changed simply by moving them. We have found these very useful for building applications (Section IV).

We envision doing similar generalizations for a variety of other existing CRDT objects.

2) Smart Contracts: One defining feature of blockchains is the smart contract. Wayfinder can also support a smart contract mechanism. Wayfinder smart contracts are programs that act upon WCRDTs and are themselves stored in WCRDTs in the WBD. Each smart contract tracks the state of certain other WCRDTs maintained by the WBD and takes actions such as accessing a local database on the device or operating physical actuators. A challenge is how to implement such secure interactions between the WBD and the physical environment in which it is deployed. To solve this, we plan to leverage Trusted Execution Environments (TEE). A TEE running on an manet device can provide secure sensor readings, including possibly GPS location and time. We show two toy applications built on top of Wayfinder smart contracts in Section IV.

B. Block Tier

In this section we describe the defenses built into the WBD design to keep it manageable and to encourage good behavior by devices.

1) Ensuring an Acyclic WBD: The WBD is maintained by a collection of devices, each identified by a public key. Each block in the WBD is created and signed by a device and is uniquely identified by a cryptographic hash. Each block contains the hashes of its *ancestor blocks*, exactly forming the edges in the DAG. *The properties of cryptographic hashes prevent cycles in the WBD.* A correct device will only accept (i.e., store) a block if it has all the ancestor blocks. One or more blocks do not have ancestor blocks—it is up to each device to decide which of those blocks it will accept. Typically a device is configured with one such block, called the *genesis block*, and thus there is a path from each block on the device to the configured genesis block.

2) Permissioned Growth: The WBD contains an authorization mechanism. Each WCRDT defines an Access Control Matrix (ACM) that itself must be a WCRDT. Its state is uniquely determined by the partially ordered set of operations on the ACM contained in the WBD. In its simplest form, an ACM is a *2P+* set that determines which devices are allowed to operate on the WCRDT and its ACM. A device is authorized to delegate rights that it has to other devices, rather than create new rights.

There are special WCRDTs called "registries" that describes a set of WCRDTs maintained by the WBD. A registry defines operations to add and remove WCRDTs from that set and also contains an ACM specifying the devices that may execute those operations. A registry is always created in a genesis block.

In regards to WCRDT operations, an operation is permissible in a block if the author has a valid standing within the ACM and the operation is in the set of allowable operations. A block with a disallowed operation is considered invalid. Since an operation is signed by its owner, this is considered a *Proof-of-Misbehavior* (PoM) of that owner.

Note that while it may appear that the WBD is a *permissioned* block DAG, any device can create a new self-signed genesis block and participate. Two disconnected WBDs can be connected by creating transactions that depend on blocks in both WBDs. However, each individual WCRDT is protected through the relevant ACMs.

3) Limiting Branching: Ideally, the WBD would be strictly linear like a conventional blockchain, providing a total order on all operations. Given the partitionable nature of the environment, this goal is not achievable while allowing progress at all times. However, we impose various constraints to encourage the WBD to conform to a shape which is "long and thin". In particular, for any two blocks of a correct device, there is always a path from one of the blocks to the other. In other words, a correct device does not generate concurrent transactions.

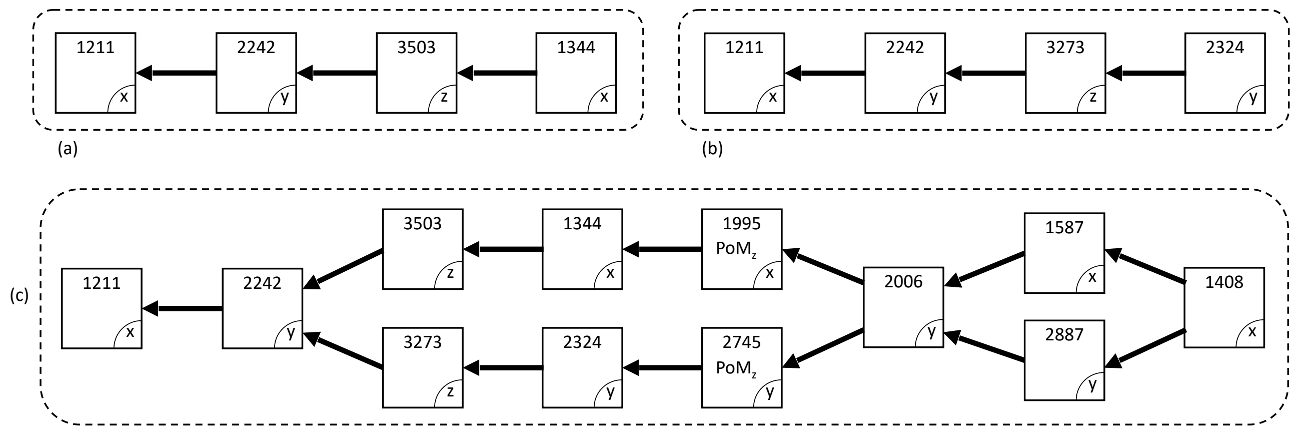


Fig. 1: Three valid WBDs. Each block displays a 4 digit unique hash identifier and the owner device in the lower right corner. (a) and (b) are two snapshots that show that device z is misbehaving because it has created two blocks (3503 and 3273) that do not depend on one another. As a result, the WCRDTs in the WBDs may have diverged. In order to merge the WBDs, both x and y have to add a PoM for z before they can create blocks that depend on both blocks from z. (c) shows a merged WBD with additional blocks from x and y added. Neither x nor y has concurrent blocks. z is no longer able to add blocks to the end.

On the other hand, a Byzantine device might create concurrent transactions. Concurrent blocks are another form of PoM. The evidence of this misbehavior is traceable back to the signature the device used to sign these blocks. A PoM can be used as an operation to remove a device from all ACMs in the WBD. See Figure 1 for an illustration.

We impose the following additional constraints on the WBD to force devices to recognize PoMs and prevent further operations from the delinquent devices:

- each device is only allowed to add one PoM operation for any other device;
- a block can only have concurrent ancestor blocks of some device if the path to each such block contains a PoM operation for that device.

A PoM creates a chain of events within the system. First, the access rights to any WCRDT that the device had are revoked. This includes any CRDTs that the device created. Additionally, the PoM revokes all delegated rights the device has granted to other devices. PoMs incentivize correct behavior by devices and their effect cannot be circumvented by devices delegating their rights to other, possibly colluding, devices. Devices may, however, have received the same rights from other devices and those rights will remain intact.

We rejected an alternative design in which blocks from misbehaving devices are automatically removed from the WBD. The reason for our choice is that correct devices may already have acted upon operations from those devices, and those actions may be difficult or even impossible to undo.

4) Tamperproving: Each device maintains an instance of a WBD and opportunistically gossips with other devices to disseminate blocks. A device will not try to reconcile with another device if it has a PoM for the other device. A correct device never drops blocks, but Byzantine devices may drop blocks from their WBDs. Also, even devices that do not drop blocks may crash or break in some unrecoverable fashion. This poses a problem, as applications may rely on blocks on the Wayfinder block graph to be *tamperproof*. Digital signatures prevent blocks from being modified, but not from being dropped. A block is tamperproof only if it is stored on a correct device. Note that because correct devices only store blocks if it stores all its ancestors blocks, a tamperproof block implies that its ancestors are tamperproof as well.

The *system WBD* is defined to be the collection of tamperproof blocks. It is a strictly growing, but virtual WBD. Applications typically will take actions only on the basis of what they know about the state of the system WBD, not the WBD instances in the devices themselves. But an application cannot know exactly which devices are correct, and thus cannot directly determine which blocks belong to the system WBD.

To this end, Wayfinder defines a set of so-called *Survivor Sets* [14], which are subsets of devices that are assumed to have at least one correct device. An additional assumption is that at least one survivor set consist of only correct devices. For example, if there is a fixed set of devices and there is an assumption that at most t devices are faulty and the majority of devices is correct (i.e., there are at least $2t + 1$ devices total), then the survivor sets could be exactly those sets that have more than t devices. However, we do not require that survivor sets intersect. Such a requirement would be unnecessary and make it harder to make progress when network partitions occur. The collection of survivor sets can either be a globally configured entity or maintained by a WCRDT in the WBD itself.

A "witness" is a cryptographic hash of a block signed by a device, certifying that that device stores a copy of that block and all its ancestor blocks. A witness therefore acts not only on the identified block but also on all its ancestor blocks. A "Proof of Witness" (PoWi) for a block is a set of witnesses for the block, one from each device in a survivor set. (Note that PoMs for any of those devices do not invalidate the PoWi because at least one device in the survivor set is correct by assumption.) A PoWi guarantees, by the assumptions on survivor sets, that the block and its ancestors are tamperproof and therefore in the system WBD. Also, by the assumption that at least one survivor set consists of only correct devices and further assuming that blocks and witnesses eventually disseminate to all correct devices (Section III-C), it is always possible to construct a PoWi for a tamperproof block.

Each correct device maintains a monotonically growing set of witnesses for its blocks. Note that a device needs to maintain only one witness for each other correct device as there are no concurrent blocks for correct devices. There can be multiple "concurrent witnesses" for blocks from the same Byzantine device. A correct device only stores witnesses for blocks that it stores. When two devices communicate they reconcile not only their local WBDs but also their witness sets (Section III-C). Note that both the WBD and the witness set of a correct device grow monotonically.

5) Checkpointing: So far we have assumed that all devices have unlimited storage. Besides the problem that this might be an unrealistic assumption, it also leads to potentially excessive amounts of communication needed to reconcile WBDs.

To solve this problem, a WCRDT can support *checkpoint* operations that summarizes its state in a way that still allows partitionable operation. In particular, checkpointing preserves the ability to compute a unique state from the partially ordered operations on the WCRDT. Most WCRDTs already natively support this as they do not need to maintain the history of operations in order to compute the state. For example, in 2P+ sets, it is only necessary to store the add set and the remove set—not their histories. The problem is how to ensure that a Byzantine device does not create invalid checkpoints.

PoWis can also solve this problem. A correct device can verify checkpoint operations locally and will not accept a block that contains an invalid checkpoint operation (although such blocks can be used as PoM). So a block containing a checkpoint operation with a PoWi can be trusted. This allows devices to potentially garbage collect blocks to reduce storage and communication. In particular, blocks that are no longer needed to compute the state of any WCRDT can be safely dropped.

6) Rate Limiting: A Byzantine device can try to create WCRDTs and/or operations on those WCRDTs at an unlimited rate. Even if we limited the rate at which a device can add operations to the WBD, it could authorize a never ending stream of new devices to add more operations (aka a *Sybil attack*).

Taking advantage of the fact that applications usually create transactions at a known maximum rate, we employ the following solution currently: each WCRDT specifies at what rate each device is allowed to add operations to the WBD. When a device authorizes another device, it must split its rate with the new device. Therefore, the two devices together cannot increase the rate at which the WBD grows.

To support this, we assume that devices have clocks that are loosely synchronized with real time. Each block has a timestamp. If b_1 is an ancestor block of b_2, then the timestamp on b_1 must be before that of b_2. Also, correct devices only consider blocks that have timestamps before the time on their clocks. Each device can compute the rate at which other devices are adding operations to a WCRDT and can use this to detect devices exceeding their rates. Such detections can be used as PoMs as well because they can be verified by any device.

7) Summary: Byzantine devices can try to mount three types of attacks: equivocation by submitting concurrent operations, Denial-of-Service by overwhelming the system with new blocks, and removing blocks that correct devices have already accepted by acting as witnesses but then dropping the blocks. Conventional blockchains prevent these either by making blockchains permissioned—controlling who can participate—or using incentive-based mechanisms such as Proof-of-Work and Proof-of-Stake.

Wayfinder is a permissionless blockchain and allows any device to join, although it places restrictions on who can do certain operations and at which rates on certain WCRDTs. Wayfinder cannot prevent Byzantine devices from *equivocating* by creating concurrent blocks because of our requirement to maintain availability in partitioned networks. However, such concurrent updates from the same device are detectable and result in such devices losing their access rights, providing incentive not to mount such attacks. Sybil attacks by delegating rights to other existing devices or even fake devices can prevent detection. However, WCRDTs remain consistent even in the presence of concurrent operations. Also, taken together, a Byzantine device and its Sybils cannot exceed the rate of block creation assigned to that device. Proofs-of-Witness and the properties of Survivor Sets prevent Byzantine devices from deleting blocks that correct devices have already accepted.

C. Reconciliation Tier

Wayfinder's Reconciliation Tier is responsible for reconciling block DAGs and witness sets between devices. In a mobile ad hoc network, reconciliation is initiated every time two devices are within each other's communication radius. This is different from a static network infrastructure where devices gossip, or periodically synchronize updates, with one another. As the period when two devices are within communication range may be limited, reconciliation must be fast. In the following section, we discuss approaches we took to address this challenge.

1) Reconciling WBDs: One approach to reconcile the WBDs of two devices is to exchange all blocks. This *Send All Blocks Protocol* (SABP) can be improved by devices also exchanging acknowledgments to prevent sending blocks that have already been acknowledged. While simple and reliable, it is inefficient as devices are generally expected to have many of the same blocks already and thus the protocol may end up sending blocks that the peer device already stores.

Wayfinder could use an existing *set reconciliation protocol* [21] to reconcile blocks. However, there are two reasons why we have rejected this for WBD reconciliation. First, set reconciliation protocols do not exploit existing structure in WBDs that can simplify and optimize reconciliation. Second, we exchange blocks in oldest-first order to ensure that the communicating nodes can make progress. Otherwise, some transferred blocks would not be able to be applied to a device's WBD upon receipt and would incur delays as devices waited until all of the block's ancestors have been received as well. The second issue could easily be exploited by Byzantine devices to waste precious communication bandwidth.

One approach that exploits the structure of WBDs is what we call the *Frontier Set Reconciliation Protocol* (FSRP). In this protocol, the devices start out with sending only the source blocks in their WBDs, that is, those blocks that no other blocks depend on. On receipt of such a block b, there are three cases:

1) the device has b in its WBD. In this case, it sends to the peer device all blocks that depend on b.
2) the device does not have b but it has all the blocks that b depends on. In this case, the device adds b to its WBD.
3) the device is missing blocks that b depends on. In this case it buffers b and requests from the peer the missing blocks.

The FSRP protocol avoids most of the duplication that occurs with the SABP protocol. The protocol can be further optimized by, in the first round, only sending the cryptographic hashes of the source blocks, thus completely eliminating sending blocks to a device that already stores them.

A problem with FSRP is that missing blocks are sent in most-recent-first order. On receipt those blocks generally cannot be accepted until a block for which the device already stores the ancestors. At that point all blocks can be added to the WBD. But should the communication link break before then, the buffered blocks are useless and communication was potentially wasted. While perhaps unlikely in practice, this effect can be exploited by Byzantine devices to send endless strings of useless blocks to a device. We solve this by exchanging *block manifests* that only contain their hash and dependencies. On receipt of a block manifest for which the device stores the dependent blocks, the device requests the content of the block from its peer.

FSRP is slow because it goes through phases rather than streaming blocks one after another. The final protocol that we will describe is the *Hashed Vector Timestamp Protocol* (HVTP). In this protocol, we exploit that correct devices do not generate concurrent blocks, and therefore blocks from a correct device can be identified by a simple counter. In particular, if there were no Byzantine devices, devices could simply exchange *Vector Timestamps* consisting of one such counter per device. On receipt, a device can easily determine which blocks its peer is missing and send it the missing blocks in oldest-first order, allowing the peer to accept each block on arrival without the need for buffering.

With Byzantine behavior we have two problems to deal with. One is that if two correct devices have identical Vector Timestamps, this does not imply that they have the same WBD. The second problem is that, upon receipt of a block, it is no longer guaranteed that the device stores the ancestors of the block.

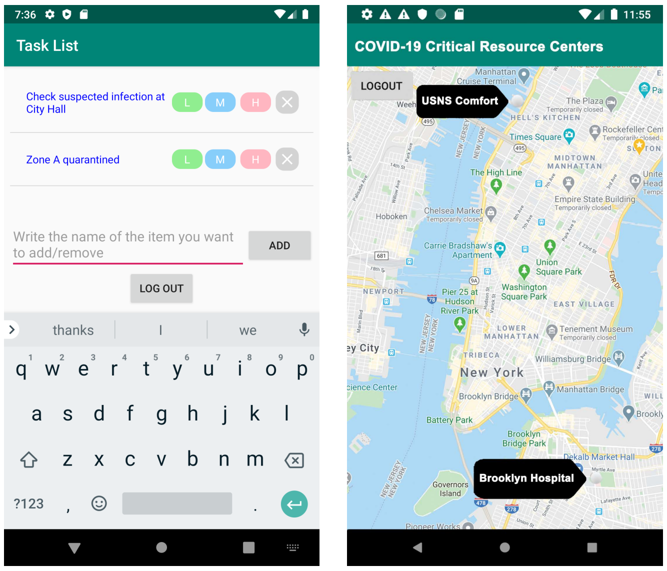
The first problem is resolved by enhancing a vector timestamp with a cryptographic hash in each entry. The hash for a device is computed by collecting the most recent blocks for that device (there may be multiple if the device is Byzantine), sorting the blocks into a list, and computing a hash over the list of blocks where both the list and hash can be computed and maintained incrementally. Now, if two correct devices have the same *Hashed Vector Timestamps* they are guaranteed to have the same WBD.

If devices detect that they do not have the same WBD either because they have different Hashed Vector Timestamps or because receiving a block for which they have no ancestor, they could revert to one of the other protocols. The current implementation of Wayfinder uses Hashed Vector Timestamp but reverts to FSRP if the protocol fails to reconcile on its own.

2) Reconciling Witness Sets: If a device maintained all witnesses, witness sets could be efficiently reconciled with existing set reconciliation protocols. However, a witness for a block also counts as a witness for all ancestor blocks, greatly reducing the storage that is needed for witnesses and making set reconciliation protocols redundant. Today, we simply exchange the witness sets. This means that some devices may end up storing witnesses for blocks that they do not store. If space is tight, the device can drop such witnesses. Correct devices never send witnesses for blocks they do not store.

IV. Wayfinder Prototype

In order to implement Wayfinder into a framework that could be installed on manet devices as well as on servers, we chose Java for our code base. One goal was to demonstrate that two concurrently running applications could rely on the same underlying WBD. So, we developed two applications for emergency first response: a *Task List* for command and control and an *Annotative Map* (see Figure 2). The *Task List* app allows users to post and complete prioritized jobs on a shared list. The *Annotative Map* app allows users to place prioritized annotations on a shared map. Both applications use 2P+ sets (Section III-A1) and run with or without network infrastructure in a completely disconnected environment. These applications run on smartphones, servers, desktops, and potentially on IoT sensor devices.


Task List
Check suspected infection at City Hall
L M H
Zone A quarantined
L M H
Write the name of the item you want to add/remove
ADD
LOG OUT
COVID-19 Critical Resource Centers
LOGOUT
USNS Comfort
Brooklyn Hospital

(a) Task List (b) Annotative Map

Fig. 2: Screenshots of sample applications.

In the remainder of the section, we discuss the layered framework that supports these applications while also minimizing power consumption. First, we will describe the interfaces applications depend on in Section IV-A, followed by the summary of the Publish-Subscribe layer that separates the Application and Block layers in Section IV-B. Next, we present the Block layer and conclude with Wayfinder's network layer in Section IV-D which introduces the techniques used to communicate information in disconnected, intermittent, and low-bandwidth environments.

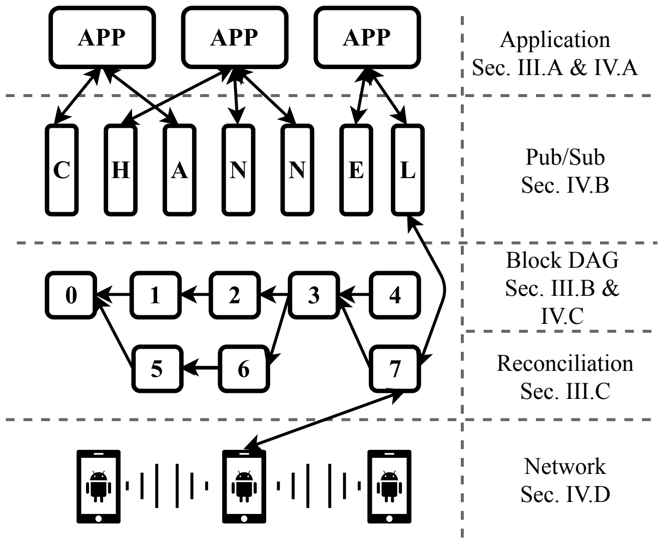

APP
APP
APP
Application Sec. III.A & IV.A
C
H
A
N
N
E
L
Pub/Sub Sec. IV.B
0
1
2
3
4
5
6
7
Block DAG Sec. III.B & IV.C
Reconciliation Sec. III.C
Network Sec. IV.D

Fig. 3: Layer Structure for Wayfinder System

A. Application Layer

To interact with Wayfinder, we expose four methods:

- `registerApplicationDelegator`: enrolls an application into a subscription service. The function takes an application context object containing channel (Section IV-B) information and a subscription handler to be called for appropriate transactions.
- `addTransaction`: loads a transaction into the subscription service. The method takes an application context, the set of concerned channels, the transaction payload in bytes, and dependent transactions within the application scope. Next, the information is translated into a Wayfinder transaction. Lastly, the method serializes and sends the transaction to the underlying block layer.
- `getWitnessesForTransaction`: given a transaction identifier, returns the set of device identifiers that are known to have a copy of the transaction.

A developer connects the application to the Publish-Subscribe layer. To begin the process, a developer invokes the `registerApplicationDelegator()` to enroll a handler for transaction notifications based on subscription channels of interest. Similar to basic publish-subscribe models, clients are notified when channel-specific transactions arrive.

To publish transactions to other applications, clients use `addTransaction()`. The transaction is sent to the appropriate channels with a list of dependent transactions. The function also triggers an arrived event callback within the subscription service. This optimization reduces client-side transaction latency and improves the user experience.

Applications vary in security and safety requirements for distributed environments. Therefore, we expose a method that allows a client to know which witnesses have verified a particular transaction by calling `getWitnessForTransaction()`. With this method, a developer can leverage the notion of Survivor Sets to devise what witnesses are required before a transaction can be acted upon. Moreover, this method can also help determine how far sensitive information has traveled within a particular network.

Additionally, the application layer has the responsibility of correctly interpreting the data contained within transactions. Therefore, a CRDT library was created for client use. The library presents a set of data structures and the allowable operations upon them. Entries contained therein have a guaranteed consistency after reconciliation and a correct causal relationship between local transactions.

B. Pub-Sub Layer

The Publish-Subscribe, or Pub-Sub, layer is a multiplexing layer between the application layer and the underlying block layer. One advantage of having this layer is that multiple applications now can cohabitate on a device while using the same WBD. The application and Pub-Sub layer communicate via channels and transactions. A channel is similar to a *read & write* queue, where application can publish (write) new transactions to and subscribe (read) incoming transactions from that channel. The mapping from applications to channels is many-to-many. An application can subscribe multiple channels.

Channel	Transaction ID	Dependencies	Timestamp	Payload

Fig. 4: Structure of a Transaction Message

A Transaction Message is a data structure defined using Google's Protocol Buffers [12], which allows applications to be written in other languages outside of Java. Wayfinder defines transactions to contain 5 fields: *Channel*, *Transaction ID*, *Dependencies*, *Timestamp*, and *Payload*. All application-specific information is placed in the *Payload* section of the transaction message as shown in Figure 4. Encryption, if needed, can be performed by the application prior to the data being placed in the payload. The dependencies among transactions must be visible to the Pub-Sub layer. This is necessary in order to eliminate invalid, dangling transactions. Pub-Sub is responsible for ensuring only transactions with fulfilled dependencies are published on their associated channels.

We have implemented Pub-Sub as an event loop. Whenever a new block arrives, the following actions are taken:

1. All transactions are extracted from the block.
2. Transactions are validated by confirming the satisfaction of its dependence set.
3. Valid transactions are hashed and placed in a mapping from transaction hash to block hash.
4. Subscribed applications are notified of the presence of new transactions.

C. Block Layer

A Wayfinder block contains three segments:

- The *header*, which includes the *owner* (currently the public key of the device that created the block), the *timestamp*, the *height* of the block (this is the number of blocks that the *owner* device created thus far), and the *hash* of each of the parent blocks.
- The list of WCRDT *operations*.
- The cryptographic *signature* of the block header and operations generated using the private key of the owner.

A device stores each block in a hash map using the hash of the block as key. Additionally, each device maintains, for each other peer device, a *chain object*. A chain object for a device is a doubly linked list of blocks generated by that device. To enable efficient search of witnesses, a device also keeps the set of last Hashed Vector Timestamps for all devices.

D. Network Layer

In order to increase code portability and to enhance system modularity, we use Google's Protocol Buffers [12] as our serialization framework. Currently, Wayfinder has two options for network communication: a regular TCP-based solution and Google Nearby [11]. When a WiFi connection to the Internet is available, devices use randomized gossip with known devices over TCP. In the absence of a Wi-Fi base station, Google Nearby leverages Bluetooth and WiFi Direct to establish an ad-hoc network and allow opportunistic peer-to-peer connections among devices.

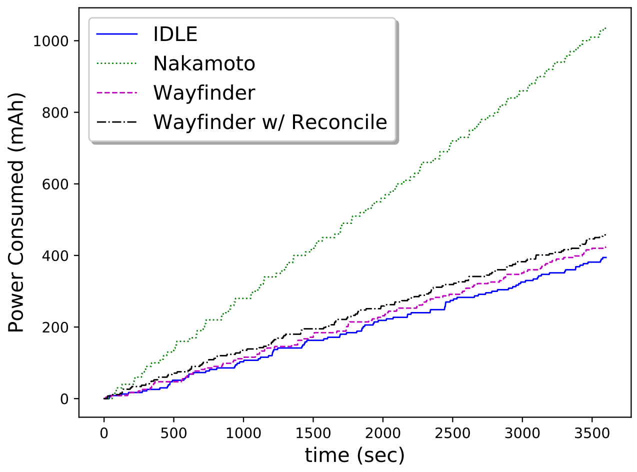


Fig. 5: Power consumed over time for Wayfinder and Nakamoto mining.

Using Google Nearby, a device can be in one of two modes: advertiser or discoverer. An advertiser makes its presence known by broadcasting connection that discoverers can use to send connection requests. While in theory a device can be in both modes at the same time, in practice this does not work well, and it is better to have a device toggle between one or the other. However, discoverers cannot find other discoverers while advertisers cannot similarly find other advertisers. To solve this issue, a device switches modes if they have not communicated with another device after a randomized timeout.

V. Evaluation

Through experimentation and trace-based simulations, we seek to answer two basic questions regarding Wayfinder:

- What is the power consumption of Wayfinder compared to mining-based blockchain protocols?
- Does Wayfinder scale to hundreds of devices in realistic settings?

In the first section we address the first question. For the second question, we use a simulation based on traces obtained from a taxi company. However, to calibrate the simulation, we also made measurements using a prototype of Wayfinder deployed in a much smaller setting.

The Wayfinder prototype devices used for experiments were six Samsung SM-T510 Galaxy Tab A (Android 9) devices with Li-Ion 6150 mAh batteries. These devices have eight cores, 2GB RAM, and 128GB of internal storage and are equipped with VHT80 antennae that can operate on 802.11 a/b/g/n/ac 2.4G+5GHz. The devices were not connected to a Wi-Fi base station during any of the experiments.

A. Power Consumption

In this section, we evaluate the power efficiency of Wayfinder's in comparison to the Nakamoto consensus protocol based on Proof-of-Work. To measure and compare power

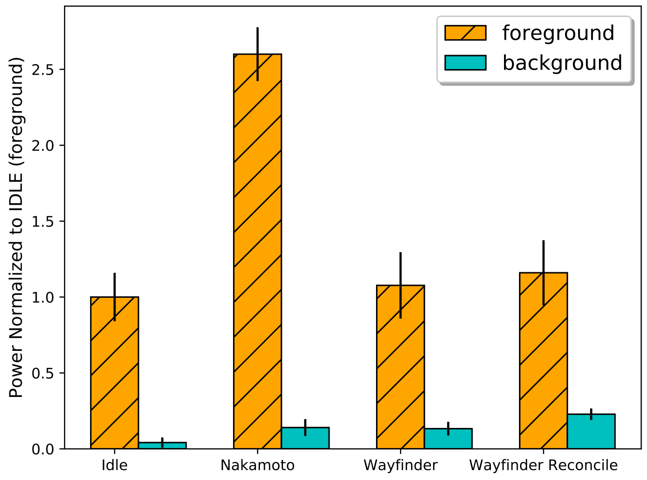


Fig. 6: Average power consumed and standard deviation by Wayfinder and Nakamoto normalized by an idle device running in the foreground.

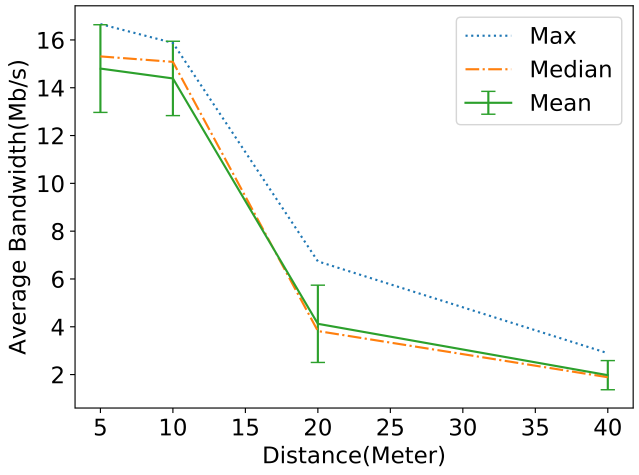


Fig. 7: Maximum, median, and mean bandwidth with standard deviation experienced by two Android devices running Wayfinder reconciliation as a function of distance.

consumption, we ran each protocol on the Samsung devices for an hour and measured power consumed at regular intervals.

For all power experiments, we measured the power consumed of the Wayfinder protocol both with and without reconciliation (i.e., with and without networking), and the Nakamoto protocol without networking to isolate the cost of solving cryptopuzzles only. We set the cryptopuzzle difficulty to produce a block every two minutes on average (i.e., 30 blocks an hour). We configured Wayfinder experiments to produce a block each second (i.e., 3600 blocks an hour). We also measured the performance of running experiments in the foreground (screen on) and background (screen off) since Android devices curb application performance running in the background [10].

Figure 5 shows the power consumed over time when experiments run in the foreground. The figure shows that, at the end of the hour, Wayfinder consumed 424.3 mAh without reconciliation and 457.5 mAh with reconciliation. Respectively, this represents 7.6% and 16.0% more power than an idle device that consumes 394.3 mAh. On the other hand, Nakamoto mining consumed 1,027.5 mAh, 160.6% more power than an idle device. Wayfinder uses 0.008 mAh to produce a block, while Nakamoto uses 21.3 mAh.

Figure 6 shows the power consumed while running the protocols for both the foreground and background configurations, normalized to the power consumed by an idle device running in the foreground (i.e., with the screen on), averaged over 10 runs. The blue bars on the left show the same data as Figure 5 at the end of the hour. The orange bars on the right show the amount of power consumed while running the protocols in the background. For Nakamoto, the block creation rate was reduced from 30 blocks an hour to 4 going from foreground to background, whereas the block rate for Wayfinder remained the same, 3600 per hour. Wayfinder running in the background consumed nearly an order of magnitude less power than in the foreground: 60 mAh without reconciliation and 90 mAh with.

B. Network Measurements

In order to calibrate our simulation experiments, we performed network measurements using our prototype and devices. The experiments used Google Nearby for peer-to-peer communication.

We measured out distances of 5, 10, 20, and 40 meters. In the experiments we ran the Wayfinder reconciliation protocol between devices, which transferred 32 MB. The effective bandwidth at each distance was measured 5 times with different pairs of devices. Figure 7 shows the average maximum bandwidth, the overall average bandwidth, and the overall median bandwidth that we measured at various distances. Bandwidth decreases as distance increases because the strength of the signal decreases logarithmically with distance. Measured bandwidth had a significant variance, particularly at small distances. The individual maximum and minimum bandwidths recorded in the experiments were 42.37 Mb/s and 0.01 Mb/s at 5 meters, respectively. We only measured the bandwidth between two devices because, in both prototype and simulation, devices running Wayfinder only talk with one peer at a time.

C. Scalability

Performing experiments in emergency scenarios is infeasible, and we do not have communication traces at our disposal. Instead, we used a full month of taxicab location traces in the city of San Francisco [24] to run discrete-time event simulation experiments. The dataset is comprised of 490 cabs dispersed throughout the city. The simulation goal is to measure amount of time for a newly generated block to propagate through the peer-to-peer network and confirm the ability of an manet device to make progress in a partitioned network. The speed at which blocks spread between devices has implications for the tamperproofness of blocks.

The data transfer rates between nearby cabs is determined by linear regression functions deduced from our real-world bandwidth experiments. Specifically, we sampled an appropriate bandwidth from a Gaussian distribution built from the mean and standard deviation functions for the bandwidth data in Figure 7. As a result, the functions, where X is the distance in meters, are defined as:

$$\textit{mean-bandwidth}(MB) = -0.39 * X + 16.24$$

$$\textit{standard-deviation}(MB) = -0.29 * X + 13.13$$

As a baseline, we used an ideal scenario where every block exchange between nearby devices occurs instantaneously and successfully. We denote these in the results as *Oracle*.

When two devices are within a sufficiently small range of one another they are able to communicate. While larger blocks lead to better communication efficiency in general, with mobile devices the exchange of a block is more likely to fail with block size because devices may move out of range during the communication. Thus, large blocks could lead to bandwidth being wasted. Using the taxicab traces, we determined the reconciliation success ratios for various block sizes averaged over 20 experiments (Figure 8).

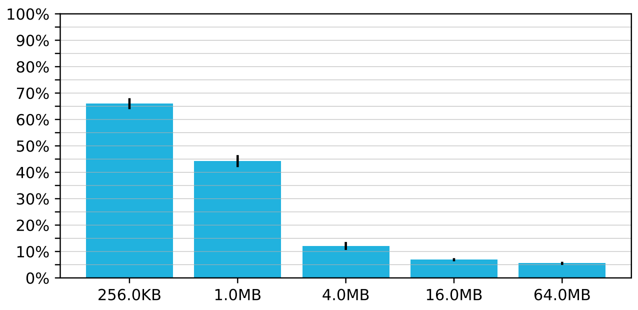


Fig. 8: Average reconciliation success ratios for various block sizes.

Next, we measured the duration for a single block (generated by a random cab) to reach all cabs in the system using the HVTP protocol. As shown in Figure 9, the baseline (Oracle) delivers a block to 90% of nodes within 1 hour. For the smallest block size (256KB), Wayfinder delivers similar infection rates as the baseline. For the largest block size (64MB), it takes 2.5 hours to propagate a block to 90% of devices.

Finally, with each taxicab generating a single block at the onset of simulation, we measured the duration for each block to reach all other taxicabs. Note that the number of blocks that a taxicab can disseminate increases over time. Figure 10 captures the average spread of 490 blocks circulating in the system. Notably, a 64MB block reaches 90% of nodes within 3 hours.

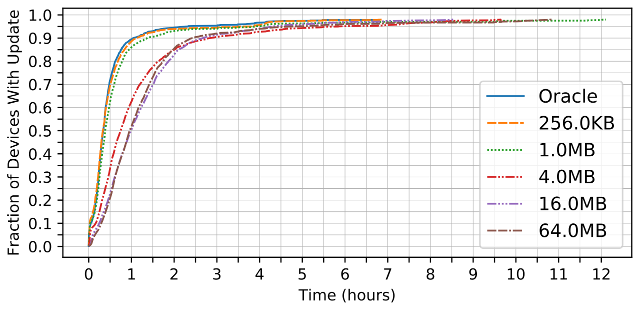


Fig. 9: CDF of block spread for various block sizes.

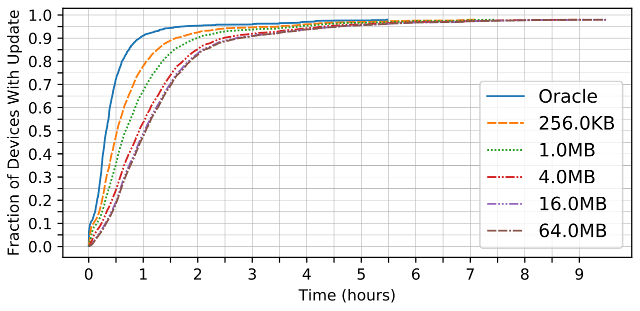


Fig. 10: CDF of the dissemination duration from a random block's generation to its ultimate receipt at the most remote participants in the system.

VI. Related work

Gossip protocols originated in the field of distributed databases [9] and has seen a resurgence coinciding with the proliferation of cloud computing [2], [13], [19], [33] and more recently within the blockchain community [4], [15], [20], [31]. Gossip protocols assume full network connectivity and can therefore not be directly applied in manet environments.

While Wayfinder uses gossip if internet connectivity is available, Wayfinder relies only on opportunistic communication between devices when not. Such protocols have been shown to work well for reliable multicast in unreliable networks [6]. Bayou [23] is a peer-to-peer storage and communication system based on opportunistic communication. Unlike Wayfinder, Bayou does not have provisions to deal with Byzantine behavior. Also, it does not use CRDTs but relies instead on ad-hoc application-dependent merging protocols that requires applications to actively detect and resolve conflicts.

CRDTs [28] provide theoretical consistency guarantees. These data structures include versions of registers, counters, sets, graphs, and maps [16], [27], and can be combined and composed to create more sophisticated data structures such as key-value stores [26]. Applications include collaborative editing [18] and distributed databases [8].

COPS [17] is a causally consistent storage system intended for wide-area networks with high latencies. Like Bayou, it uses application-dependent merging strategies and has no defenses to Byzantine behavior.

To deal with Byzantine behavior, blockchains were first introduced in 2008 as part of the Bitcoin cryptocurrency system [22]. Since then, the blockchain field has seen explosive growth with many variants. However, most blockchain designs have a linear structure and rely on a proof-of-work consensus mechanism that requires solving a computationally expensive crypto puzzle. These characteristics make them poorly suited to manet environment where computational power is limited and continuous network connectivity is not guaranteed.

Some blockchain variants use a DAG structure similar to Wayfinder. The GHOST protocol is a modification to the Bitcoin blockchain that uses a DAG structure to improve security [30]. The intuition behind this modification is to enable a more robust method for decision making in fork selection. By keeping track of all forks, a node can choose a fork based on the heaviest-subtree-wins rule (the subtree with the largest number of blocks) as opposed to the longest-chain-wins rule, wasting less work and thus eliminating certain forms of attacks. While this approach reduces the computational requirements, it is still much too high for manet devices. Also, the approach still requires continuous network connectivity.

The recently proposed SPECTRE [29] and MeshCash [5] blockchains also use a DAG structure along with a protocol to reach consensus in the case of conflicts. Both blockchains use Proof-of-Work and require continuous network connectivity, which eliminates them from consideration for our use cases.

Several blockchains have proposed using DAGs and *entanglement* to reduce the reliance on Proof-of-Work. Like Wayfinder, IOTA [1], [3] is a cryptocurrency established to operate within the IoT space. In order to create a new transaction, devices are required to verify a certain number of prior transactions. Consequently, IOTA uses witnesses to determine when a transaction is valid within a system. IOTA uses Markov Chain Monte Carlo sampling to verify unapproved transactions [25]. Additionally, double spends are resolved through a consensus algorithm that determines which transactions to keep based on the number of descendant transactions. In order to accomplish this, IOTA requires continuous strong network connectivity. IOTA also uses a weak form of Proof-of-Work to reduce Sybil attacks.

Other DAG-based protocols such as HashGraph [4] Byteball [7], and Avalanche [35] do not rely on Proof-of-Work. The DAG structure in the aforementioned blockchains is not designed to provide partition tolerance like our case, but rather to exploit available parallelism for increased throughput of transactions by only ordering transactions that are dependent. As such, these blockchains expect strong network connectivity and are therefore unsuitable in our use cases.

VII. Conclusion

In this paper, we present Wayfinder, a middleware for a Byzantine mobile ad hoc network (manet). We designed a new distributed tamperproof data structure, the Wayfinder DAG, with opportunistic update propagation. The data structure leverages entanglement techniques from blockchains combined with keeping track of *witnesses* to make blocks tamperproof. Wayfinder uses provable detection of misbehavior to incentivize good behavior. Use of CRDTs combined with happens-before ordering allows applications concurrent and continuous access to consistent data structures. Wayfinder supports applications running under intermittent network connectivity with low power consumption and scales to at least hundreds of devices.

Acknowledgments

This work was performed under financial assistance award 70NANB17H181 from the U.S. Department of Commerce, National Institute of Standards and Technology (NIST), as well as support from Cisco, Intel, Facebook, and Infosys.

References

[1] Alexander, R. *Iota-introduction to the Tangle technology: Everything you need to know about the revolutionary blockchain alternative*. Independently published, 2018.

[2] Babaoglu, O., Marzolla, M., and Tamburini, M. Design and implementation of a p2p cloud system. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (2012), pp. 412–417.

[3] Baek, B., and Lin, J. Iota: A cryptographic perspective.

[4] Baird, L. The Swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds, Inc. Technical Report SWIRLDS-TR-2016 1* (2016).

[5] Bentov, I., Hubácek, P., Moran, T., and Nadler, A. Tortoise and Hares Consensus: the Meshcash framework for incentive-compatible, scalable cryptocurrencies.

[6] Chandra, R., Ramasubramanian, V., and Birman, K. Anonymous gossip: Improving multicast reliability in mobile ad-hoc networks. In *Proceedings 21st International Conference on Distributed Computing Systems* (2001), IEEE, pp. 275–283.

[7] Churyumov, A. Byteball: A decentralized system for storage and transfer of value. *URL https://byteball. org/Byteball. pdf* (2016).

[8] Cihan, B. Getting started with active-active geo-distribution for Redis applications with CRDTs, 2017.

[9] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of Distributed Computing* (1987), pp. 1–12.

[10] Developers, G. Android 11 developer preview. Tech. rep. Background Execution Limits.

[11] Google. Nearby.

[12] Google. Protocol buffers.

[13] Jelasity, M. Gossip. In *Self-organising software*. Springer, 2011, pp. 139–162.

[14] Junqueira, F., and Marzullo, K. Designing algorithms for dependent process failures. In *Future Directions in Distributed Computing*. Springer, 2003, pp. 24–28.

[15] Karlsson, K., Jiang, W., Wicker, S., Adams, D., Ma, E., van Renesse, R., and Weatherspoon, H. Vegvisir: A partition-tolerant blockchain for the internet-of-things. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (2018), IEEE, pp. 1150–1158.

[16] Kleppmann, M., and Beresford, A. R. A conflict-free replicated JSON datatype. *IEEE Transactions on Parallel and Distributed Systems 28*, 10 (2017), 2733–2746.

[17] Lloyd, W., Freedman, M. J., Kaminsky, M., and Andersen, D. G. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), pp. 401–416.

[18] Lv, X., He, F., Cai, W., and Cheng, Y. A string-wise CRDT algorithm for smart and large-scale collaborative editing systems. *Advanced Engineering Informatics 33* (2017), 397–409.

[19] Matos, M., Sousa, A., Pereira, J., Oliveira, R., Deliot, E., and Murray, P. Clon: Overlay networks and gossip protocols for cloud environments. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (2009), Springer, pp. 549–566.

[20] MIKALSEN, J. F. Firechain: An efficient blockchain protocol using secure gossip. Master's thesis, UiT Norges arktiske universitet, 2018.

[21] MINSKY, Y., AND TRACHTENBERG, ARI ANDZIPPEL, R. Set reconciliation with nearly optimal communication complexity. In *Transactions on Information Theory* (2003), vol. 49.9, IEEE, pp. 2213–2218.

[22] NAKAMOTO, S., ET AL. A peer-to-peer electronic cash system. *Bitcoin.–URL: https://bitcoin.org/bitcoin.pdf* (2008).

[23] PETERSEN, K., SPREITZER, M., TERRY, D., AND THEIMER, M. Bayou: replicated database services for world-wide applications. In *Proceedings of the 7th ACM SIGOPS European workshop* (1996), pp. 275–280.

[24] PIORKOWSKI, M., SARAFIJANOVOC-DJUKIC, N., AND GROSSGLAUSER, M. A parsimonious model of mobile partitioned networks with clustering. In *The First International Conference on COMmunication Systems and NETworkS (COMSNETS)* (January 2009).

[25] POPOV, S. The Tangle. https://iota.org/IOTA Whitepaper.pdf.

[26] SANFILIPPO, S. Redis. *http://redis. io* (2009).

[27] SHAPIRO, M., PREGUIÇA, N., BAQUERO, C., AND ZAWIRSKI, M. A comprehensive study of convergent and commutative replicated data types.

[28] SHAPIRO, M., PREGUIÇA, N., BAQUERO, C., AND ZAWIRSKI, M. Conflict-free Replicated Data Types. In *Symposium on Self-Stabilizing Systems* (2011), Springer, pp. 386–400.

[29] SOMPOLINSKY, Y., LEWENBERG, Y., AND ZOHAR, A. Spectre: A fast and scalable cryptocurrency protocol.

[30] SOMPOLINSKY, Y., AND ZOHAR, A. Secure high-rate transaction processing in Bitcoin. In *International Conference on Financial Cryptography and Data Security* (2015), Springer, pp. 507–527.

[31] VAN RENESSE, R. A blockchain based on gossip? – a position paper. In *Distributed Cryptocurrencies and Consensus Ledgers (DCCL 2016)* (July 2016).

[32] WOOD, G., ET AL. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper 151*, 2014 (2014), 1–32.

[33] WUHIB, F., STADLER, R., AND SPREITZER, M. Gossip-based resource management for cloud environments. In *2010 International Conference on Network and Service Management* (2010), IEEE, pp. 1–8.

[34] XIAO, Y. Accountability for wireless lans, ad hoc networks, and wireless mesh networks. *IEEE Communications Magazine 46*, 4 (2008), 116–126.

[35] YIN, M., SEKNIQI, K., VAN RENESSE, R., AND SIRER, E. G. Scalable and probabilistic leaderless BFT consensus through metastability. *CoRR abs/1906.08936* (2019).