Decentralizing IoT Management Systems Using Blockchain for Censorship Resistance

Songlin He, *Student Member, IEEE*, Qiang Tang, *Member, IEEE*, Chase Qishi Wu, *Senior Member, IEEE*, and Xuewen Shen

***Abstract*—Blockchain technology has been increasingly used for decentralizing cloud-based Internet of Things (IoT) architectures to address limitations faced by centralized systems. While many existing efforts are successful in decentralization with multiple servers (i.e., full nodes) to handle faulty nodes, an important issue has arisen that external clients have to rely on a relay node to communicate with the full nodes in the blockchain. Compromization of such relay nodes may result in a security breach and even a blockage of IoT sensors from the network. In this article, we propose blockchain-based decentralized IoT management systems for censorship resistance, which include a "diffusion" function to deliver all messages from sensors to all full nodes and an augmented consensus protocol to check data losses, replicate processing outcome, and facilitate opportunistic outcome delivery. We also leverage public key aggregation to reduce communication complexity and signature verification. The experimental results from proof-of-concept implementation and deployment in a real distributed environment show the feasibility and effectiveness in achieving censorship resistance.**

***Index Terms*—Blockchain, censorship resistance, cryptography, Internet of Things (IoT).**

I. Introduction

SPAWNED from machine-to-machine technology [2], Internet of Things (IoT) is becoming a dynamic global network infrastructure with self-configuring capabilities where physical and virtual "things" with identities, physical attributes, and virtual personalities are seamlessly integrated into the information network [3]. According to Gartner, the number of IoT devices worldwide is increasing by 30%–40% per year and will reach 20.8 million by 2020 [4]. IoT is recognized as one of the most important areas of future technology and is gaining vast attention from a wide range of industries [5].

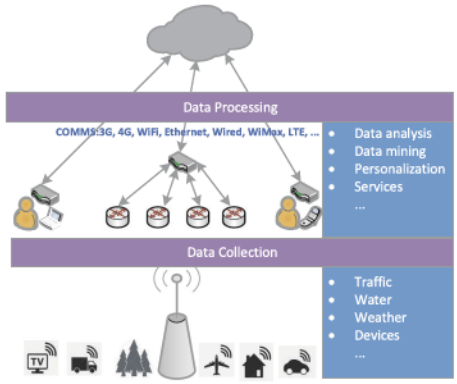

Fig. 1. Centralized IoT system architecture.

As a subset and natural evolution of IoT, industrial IoT (IIoT) shares common technologies (sensors and cloud platforms) with IoT but has higher requirements on security, scalability, reliability, and resilience. One example of the IIoT vision is the Industrial Internet of Things Services and People (IoTSP) platform [6]. The rapid development of IIoT is facilitated by the capability of data generation, collection, aggregation, and analysis over the Internet to maximize the efficiency of machines and the throughput of operations. This brings about significant challenges since data may flow across various network and administrative boundaries at the risk of attacks or failures.

Specifically, existing IoT systems (including IIoT) mainly rely on centralized clouds, where sensors collect and send data directly to a central server on the cloud for analysis, as shown in Fig. 1. This model has several drawbacks. For example, the cloud server may present a single point of failure; clouds are typically vendor specific and may not be compatible with each other, thus adversely affecting data sharing between them. Also, existing centralized IoT solutions are expensive due to a high cost in infrastructure and maintenance. Among these shortcomings, security is of primary concern. According to Van Der Meulen [4], by 2022, half of the security budgets for IoT will be allocated to fault remediation, recalls, and safety failures rather than protection. Therefore, a distributed trust technology ensuring security is regarded as a cornerstone for the continual growth of such IoT solutions. The blockchain technology is

Manuscript received May 24, 2019; accepted August 9, 2019. Date of publication September 6, 2019; date of current version January 4, 2020. This work was supported in part by U.S. National Science Foundation under Grant CNS-1828123 and Grant CNS-1801492, in part by the Google Faculty Award with the New Jersey Institute of Technology, and in part by the Key Research and Development Program of Zhejiang Province, China under Grant 2019C03138 with the Communication University of Zhejiang, China. Paper no. TII-19-2005. This paper was presented in part at the 1st Workshop on Distributed Ledger of Things, New York, NY, USA, November 2018. *(Corresponding author: Chase Qishi Wu.)*

S. He, Q. Tang, and C. Q. Wu are with the Department of Computer Science, Ying Wu College of Computing, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: sh553@njit.edu; qiang@njit.edu; chase.wu@njit.edu).

X. Shen is with the Communication University of Zhejiang, Hangzhou 310018, China (e-mail: shenxuewen@cuz.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2019.2939797

under rapid development and has proved to be an effective solution to realizing such goals due to its intrinsic security [7].

Blockchain is typically viewed as an immutable *ledger* for recording *transactions*, maintained in a distributed network of mutually untrusting *peers* [8]. Any participating peer can submit data (sometimes also referred to as a transaction), which is eventually broadcasted and replicated at all participating peers executing some consensus protocols. As an abstract layer, blockchain technology provides a reliable delivery of messages to ensure that all participating peers have a consistent copy of the ledger. This is referred to as "transparency," a property frequently mentioned about blockchain; on the other hand, once a message is written to the ledger and replicated at all peers, each peer can only modify its local ledger and the data would remain intact in other peers' ledgers. This is referred to as "immutability," another important property of blockchain. A more unique function of blockchain is to support "smart contract,"[1] which is a piece of program code that implements a predefined application logic and deterministically[2] runs on all participating peers.

The aforementioned properties of blockchain technology have facilitated its widespread applications to the IoT domain. For example, the "immutability" property of blockchain brings resistance to unauthorized modification. Since the entire history of device configuration is stored in the blockchain, recovery from incidents is straightforward. Depending on whether or not peers need to be authorized, blockchain technology is divided into two main categories: permissioned and permissionless. In this article, we focus on *permissioned blockchain* where participating nodes are all certified and known to others. In a more visionary level, IBM laid out a blueprint for "device democracy" [10], which employs blockchain to distributively manage transaction processing and coordination among hundreds of billions of interacting devices. Such an ambitious goal might take time to come to life, but on the other hand, decentralizing local management systems via permissioned blockchain to improve robustness and availability is much more viable [11].

A. Problems

Although blockchain technology offers a promising way to decentralize IoT management systems, such decentralizations cannot be realized completely based on *existing* blockchain platforms, such as the popular permissioned blockchain and hyperledger fabric (fabric for short) [8]. Note that blockchain technology (in particular, the consensus protocol) itself only concerns how to replicate data across peers consistently. Many practical issues, such as data input from external sources and data output from the ledger are not considered by the consensus protocol. These problems are currently subject to ad hoc designs and could potentially become a bottleneck in revealing the full power of a decentralized system.

[1]The "scripts" in Bitcoin is a predecessor of smart contract, whereas in Ethereum [9], it is a collection of code (functions) and data (state) that reside at a specific address.

[2]Here, "deterministically" means that the program code always produces the same output with the same input.

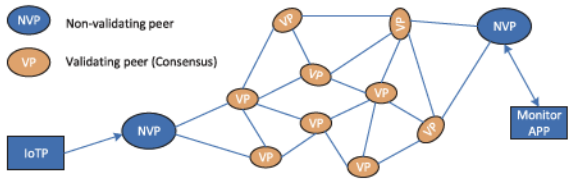

Fig. 2. Data flow in a hyperledger-fabric-based IoT management system [12]. Internet of things platform (IoTP) is the IBM Watson IoT platform, Monitor APP is for querying the ledger, which runs in read-only mode and can only see the committed world state with no ability to change it. VP (also referred to as "full node") is a validating peer in the blockchain network. NVP is a nonvalidating peer, which is responsible for forwarding or providing an interface for querying the ledger. The combination of permissioned blockchain hyperledger fabric with IoT presents a censorship problem with a single entry or exit point.

Normally, multiple sensors are connected to one server (referred to as *gateway node*, which is one of the nonvalidating nodes and whose goal is to settle with the heterogeneity between different sensor networks and the cloud [13] and effectively retrieve data from sensor networks [14]), and the server is responsible for forwarding on behalf of the sensors and participating in the consensus protocol to post the collected data to the distributed ledger. Obviously, if this gateway node is corrupted, sensor messages cannot be even transmitted to any of the blockchain's full nodes, thus the sensor simply loses the ability of "writing" to the ledger. In fact, such kind of architecture is common in existing systems, for example, Fig. 2 shows the data flow when building IoT application on top of hyperledger fabric [12].

Problem 1: The gateway node, i.e., the nonvalidating peer (*NVP*) node in fabric, could be censored. Consider a (potentially decentralized) IoT management system for environmental monitoring, where interested departments control the gateway node. The notorious *Flint water crisis* is a practical example and lesson. Flint authorities insisted for months that the city water was safe to drink, but finally it was reported that the Michigan Department of Environment Quality and the city of Flint discarded two of the collected samples containing a dangerous level of lead to avoid high cost and lawsuit.

Problem 2: The query result from the blockchain network could be censored. As an IoT management system, besides writing data into the ledger, sometimes actuators/devices may also need to read or receive instructions from the ledger. Similarly, at present, such message passing out of the blockchain is still carried out via an external nonvalidating node, which connects to one or several full nodes[3] in the blockchain network. If this external node or its connected full nodes are hacked/censored to be malicious, e.g., critical control commands are dropped, serious consequences may occur.

Consider the application of decentralized energy IoT management, the sensors continuously send real-time environmental measurements to the ledger, and the management servers analyze these measurements and send instructions back to the actuators. For example, if the temperature or pressure reaches

[3]Full nodes execute, validate, and commit transactions to the ledger in a blockchain network. Each of them maintains a copy of the ledger state.

a threshold, the servers need to instruct the actuators to shut the valve or reduce the amount of oxygen pumped into the combustion facility. If the forwarding node is compromised, such instructions may be dropped or modified on purpose to create a disaster.

These problems motivate us to consider how to build a *censorship resistant* decentralized IoT management system.

B. Contributions of This Article

We design a protocol that decentralizes the message passing module for sending and receiving data from a distributed ledger, thus avoid the single point of failure at the gateway node; moreover, this is done in a way that is compatible with existing consensus protocols so that our method can be plugged into existing platforms, as detailed in the following.

1) First, we propose to replace the traditional gateway node in IoT scenarios with several *seed nodes*, which perform the same function as gateway node but also participate in blockchain network as *full nodes*. Then, we introduce a message "diffusion" mechanism to realize censorship resistance considering the single entry point problem and propose an augmented consensus protocol to achieve reliable data delivery.
2) Furthermore, we propose the protocol to deal with the single exit point problem and the case that data on at most one-third of all full nodes are maliciously modified.
3) Finally, we propose to leverage the cryptographic tool of public key aggregation to reduce communication overhead and complexity of verification.

The rest of this article is organized as follows. Section II conducts a survey of related work. Section III provides a formal definition of the problem with the corresponding security requirements. Section IV presents an overview of the protocol. Section V provides the protocol details. Section VI conducts performance evaluation through proof-of-concept implementation. Finally, Section VII concludes this article.

II. Related Work

A. Integration of IoT and Blockchain

Billions of connected devices in future IoT networks face significant technical challenges in security, privacy, and interoperability, which are not taken into consideration during the design phase of IoT products [7]. The blockchain technology under rapid development emerges as a viable solution to addressing these challenges in decentralizing IoT systems.

Many challenges confronted by current IoT architectures may be addressed by blockchain. In [15], Kshetri presented a positive attitude toward strengthening IoT with blockchain and provided insights into how blockchain enhances IoT security, such as leveraging blockchain-based identity and access management systems or improving the overall security in supply chain networks. Cha *et al.* investigated data confidentiality and authentication based on blockchain [16]. Novo proposed to utilize blockchain as the access control layer for better security and privacy [17]. Panarello *et al.* conducted a survey of the integration of blockchain and IoT, where different application domains are categorized, including *smart home*, *smart city,* and *smart energy* [7].

B. Gossip Protocol

A *gossip protocol* [18] is a procedure where a data item is routed to all members in a distributed network similar to epidemics spreading. Gossiping has been traditionally used for reliable information dissemination, but its applicability goes far beyond in distributed systems. Uber implemented a gossip protocol variation called SWIM [19] to allow independent workers to discover each other. Cassandra [20] used a gossip protocol for peer discovery and metadata propagation. *Docker*'s multihost networking [21] employed a gossip protocol to exchange overlay network information. *Hyperledger fabric* [8] implemented a gossip data dissemination protocol to ensure data integrity and consistency among different roles of nodes.

Kermarrec and Van Steen [22] provided the general organization of a gossip protocol and discussed one of its most successful applications for *dissemination*, which is achieved by letting peers forward messages to each other. Eugster *et al.* [23] elaborated the gossiping dissemination process with three parameters: the number of messages stored in a node's local cache, the number of selected peers for message forwarding, and the upper bound of times that a message is forwarded. The shuffle protocol in [24] is designed to disseminate information among a collection of wireless devices in a mesh network, but it only considers a synchronous model where the transmission duration among peers is constant. Andrew and Antonios [25] improved this model by taking into account the dynamics of a real network and employed exponential distribution and hyperexponential to simulate various transmission durations among peer nodes. In this article, we use gossip to realize robust message dissemination from sensor networks to blockchain networks and conduct experiments in real distributed environments.

C. Censorship Resistance

Censorship resistance in IoT data communication is made possible by the decentralization and immutability nature of blockchain network. The study in [26] and [7] pointed out that the decentralization of IoT on top of blockchain is censorship resistant because *inside* the blockchain network, there is no controller and entities only trust the quality of the cryptographic algorithms that govern the operations. Obviously, the censorship problem still exists in the components of the blockchain network that communicate with *external* devices. Hence, we provide a formal definition of "censorship resistance" in blockchain-based IoT and propose an effective solution.

III. Problem Formulation

In this section, we formulate the problem and describe security requirements. The notations are provided in Table I for the convenience of reference.

Fig. 3 illustrates the current blockchain-based IoT management model. Typically, in a blockchain-based IoT management

TABLE I
KEY NOTATIONS RELATED TO THE PROTOCOL

Notation	Represent for
L_{alive}	the list maintaining live nodes in blockchain network
$sensor_{ID}$	the unique ID of a sensor
δt_{sensor}	the time period when a sensor sends data
δt_{seed}	the time period when seed nodes process data
δt_{dif}	the time period when data diffusion is completed in a synchronous network
$\mathcal{Z}$	a physical zone including sensors and gateway nodes
$\mathcal{Z}_{ID}$	the unique ID of a zone $\mathcal{Z}$
d	the data collected by a sensor
l	the number of sensors in a certain zone $\mathcal{Z}$
k	the number of selected neighbor nodes in gossip protocol
n	the number of data items collected during δt_{sensor}
n'	the number of data items during B_{seed}
s	the number of seed nodes
c	command/instruction sent to actuators from blockchain network
σ	signature from the message sender
γ	the local cache size for gossip protocol
$\mathcal{N}$	the number of full nodes in blockchain network
C	the local cache on a peer node for gossiping
BUF	the buffer that a full node uses to receive data from sensors
ts	time stamp
$SEED$	message from sensors to seed nodes
B_{seed}	a constructed batch of $SEEDs$ maintained on seed nodes
$node_{ID}$	the unique ID of a full node in blockchain network
GM	gossip message from seed nodes to all full nodes

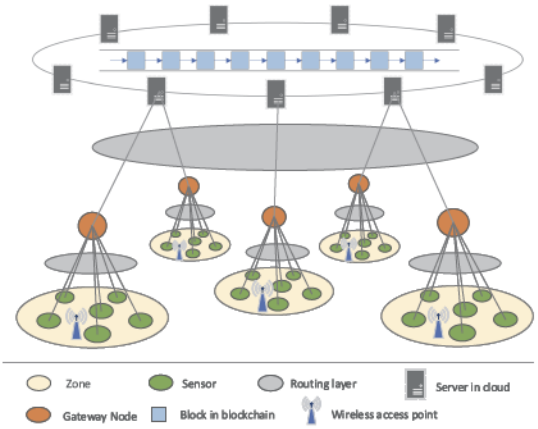

Fig. 3. Blockchain-based IoT management system model.

system, multiple sensors are deployed in a certain area (e.g., a power plant) for data collection (e.g., temperature measurements). The collected data are sent to the nearest gateway node and forwarded by routers through a wireless network to a server in the blockchain network, which starts to execute the consensus protocol and replicate the data across all participating servers (also known as "peers" or "replicas"). Such consistent data items stored in blocks are appended to the blockchain as "transactions."

To investigate the security issues in blockchain-based IoT, we first provide the following definitions.

Definition 1 (Consensus): A consensus protocol has the following properties.

1) *Termination:* Each participating peer outputs something locally within a limited amount of time.
2) *Agreement:* All honest peers in the network agree on the same value.
3) *Validity:* If all honest peers receive the same value v, then the agreed result should be equal to v.

From this definition, we know that consensus only considers ledger replication while disregarding how inputs are received from and outcomes are delivered to external clients. External clients are not full nodes of the distributed ledger and, thus, have to rely on some servers to relay. As such, existing architectures assume "trusted" relay, which is vulnerable in practice as the relay server could either be hacked or simply be malicious. To realize the properties of "reliable message delivery" and "transparency" of a distributed ledger, we provide the following definition.

Definition 2 (Censorship Resistance): Consider a sequence of data items $(d_1, d_2, \ldots, d_n)$ sent from an IoT network to a blockchain network. The system is censorship resistant if it meets the following two conditions.

1) The ledger records a permutation of the vector without any data loss.
2) The corresponding actuator in the IoT network is guaranteed to eventually receive the value of $y = \mathcal{F}(d_1, \ldots, d_n)$, which is also stored in the ledger, where $\mathcal{F}$ is a predefined processing function.

We now introduce the security issues in the current blockchain-based IoT model.

A. Security Against Entry Point Censorship

A malicious or hacked node[4] relaying messages from the sensor network to the blockchain network may act arbitrarily, e.g., drop messages, infinitely delay messages, or modify message contents. Meanwhile, even correct data are disseminated to the blockchain network, it may get lost during the process of reaching a consensus among all peers. We define the security in these two cases as follows.

1) An adversary $\mathcal{A}$ corrupts the gateway node g in a zone $\mathcal{Z}$ including $\text{sensor}_1, \ldots, \text{sensor}_l$. The message sent from the sensor network to g is denoted as $m = (d_1, \ldots, d_n)$. We consider a bad event B_1 as follows.
   a) The number of data forwarding from g to blockchain network is less than n.
   b) There is no data forwarding from g to blockchain network since g blocks all the messages.
   c) Some data items in message m are modified before sending to blockchain network.
2) An adversary $\mathcal{A}$ corrupts f nodes in the blockchain network to execute a consensus protocol. The message sent to the blockchain network from the sensor network is denoted as $m = (d_1, \ldots, d_n)$. We consider a bad event B_2 as follows.
   a) Not all nodes in the blockchain network update m to their ledgers.
   b) All nodes update m to their ledgers but on some nodes, the number of data items in m is less than n, i.e., $|m| < n$.

[4]Generally, a cluster with a master–slave architecture is constructed for the gateway node to tolerate crash fault, but it still acts as a single node since only the master node is responsible for providing services. Our proposed solution tolerates both crash fault and Byzantine fault.

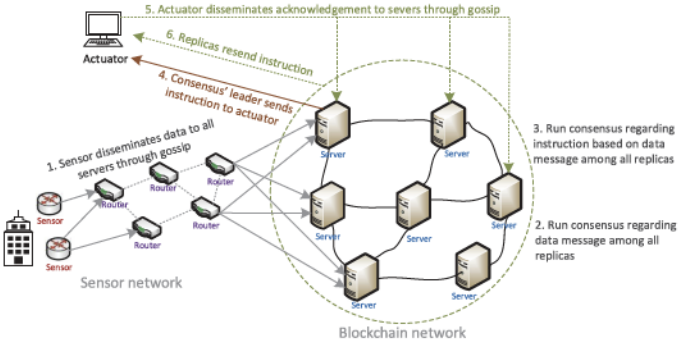

Fig. 4. Data flow in the improved architecture to resist censorship.

c) All nodes update m to their ledgers and on all nodes $|m| = n$, but the data items in m are out of order.

The security of our proposed protocol requires that for every polynomial time adversary $\mathcal{A}$, the probability $\Pr[B_1]$ and $\Pr[B_2]$ is negligibly small.

B. Security Against Exit Point Censorship

When querying from the blockchain network, a corrupted node may perform malicious actions to actuators to cause a disaster. We define the security in this case as follows: an adversary $\mathcal{A}$ corrupts f nodes in the blockchain network. We consider a bad event B_3 as follows: when querying from the ledger, the instruction y is modified to y' and sent to actuators. Again, the security of our proposed protocol requires that for every polynomial time adversary $\mathcal{A}$, the probability $\Pr[B_3]$ is negligibly small.

IV. System Overview

To defend against potential threats of censorship, we augment the current blockchain network architecture and the consensus protocol. Fig. 4 illustrates the data flow in the improved architecture.

A. Censorship Resistant Message Delivery

Our protocol carries out messages delivery as follows.

1) The sensors disseminate data to $f + 1$ gateway nodes (referred to as "seed nodes"), which are full nodes, not just forwarding messages from the sensor network to the blockchain network.
2) The seed nodes disseminate data to all other peer nodes in blockchain network through gossip-based diffusion mechanism.
3) A leader node starts the Byzantine consensus protocol to replicate the data.
4) Each replica performs filtering validation (FV) to check if there is any data loss after consensus.

Specifically, to defend against censorship at the data entry point, we need to make sure that each sensor is connected to multiple servers instead of just one single gateway node. In the proposed scheme, the conventional single gateway node is replaced with multiple full nodes in the blockchain network (i.e., *seed nodes*), which perform not only the same function as the original gateway node but also a set of blockchain operations, such as reaching consensus and updating ledger, hence eliminating the crash fault of the original gateway node. Furthermore, the number of seed nodes is at least $f + 1$ to tolerate the Byzantine fault as discussed later. We propose to use a gossip-based protocol to achieve message diffusion among all peers for better robustness. Moreover, we enhance the underlying consensus protocol (e.g., BFT-SMART [27]) such that each honest participating peer further checks whether the block being replicated has dropped some data before updating the local ledger. If a sufficient number of peers observe data missing, the consensus process is restarted (e.g., a view change type of subprotocol is triggered). We would like to point out that this enhancement is generic and could be applied to any BFT protocols.

After this round, the data are appended to the local ledger of each full node. To further enhance the protocol to support basic data analysis and instruction delivery, we propose the following.

B. Data Processing and Censorship Resistant Instruction Delivery

Our protocol carries out data processing and instruction delivery as follows.

1) For a predefined processing function $\mathcal{F}$, another round of consensus is initiated using the outcome of $\mathcal{F}(\cdot)$ as the data to be replicated. Such agreement is the same as the third step in the aforementioned message delivery round, the consensus content is instruction instead of message.
2) Once the value of $\mathcal{F}(\cdot)$ is written in each local ledger, the leader forward the value of $\mathcal{F}(\cdot)$ with the peer nodes' signatures to the corresponding actuators/devices.
3) Actuators receive an instruction and send feedback containing an acknowledgement to all full nodes.
4) All replicas maintain a timer and wait for the acknowledgment for each sent instruction; if the acknowledgment is not received within a predefined time period, they all resend the instruction to actuators, details are elaborated in Section V.

After the data are written to the ledger, the nodes run the analysis program $\mathcal{F}$ that is predefined and deployed in a smart contract (an example about $\mathcal{F}$ is provided in Section V), and use the output y of $\mathcal{F}$ as input to run another consensus protocol. At the end of this consensus protocol, there are a sufficient number of signatures on the same y, and an honest leader node forward the output y together with the signatures to the actuator. The actuator simply broadcasts an acknowledgment to all servers if it receives instructions from the leader server and successfully verifies their signatures. The peer nodes wait for a predefined period of time, and if there is no feedback from the corresponding actuator, they all send value y to the actuator. This feedback mechanism achieves an opportunistic efficiency: when the leader is honest, only one single message is sent to the actuator and this single message contains the signatures of most peers (specifically $2f + 1$, where f is the largest number of malicious nodes) in the blockchain network; only when the leader node drops the outgoing instruction, the

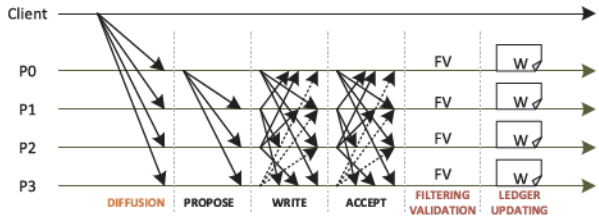

Fig. 5. Message pattern for dealing with the single entry.

other peers jointly inform the actuator. Although we assume that the diffusion from the sensor network to the blockchain network (and vice versa) be completed in a fixed amount of time (i.e., as a synchronous network), the network that connects the peers (i.e., the blockchain network itself) could be partially asynchronous and we are still able to deploy such consensus protocols as PBFT [28].

V. Protocol Design

In this section, we provide a detailed description of protocol design to realize censorship resistance on top of blockchain-based IoT. The main idea is to further decentralized data entry and exit point. For simplicity, we denote the number of all replicas (full nodes) as $\mathcal{N}$, $|\mathcal{N}| = 3f + 1$[5], where f is the maximum number of faulty nodes. In addition, BFT-SMART, which implements a modular state machine replication protocol atop a Byzantine consensus algorithm [29], is used as an example for the underlying consensus protocol to explain our protocol process.

A. Dealing With the Single Entry

The data are sent from the sensor network to the blockchain network, which is referred to as "inbound flow." Fig. 5 illustrates the normal operation on inbound flow in our protocol, as detailed in the following three phases.

1) Phase I for Message Diffusion: The goal in this phase is to ensure that data from each sensor be quickly diffused to every full node in a more robust way instead of relying on the single gateway node, so that when the consensus protocol is invoked, all full nodes have a copy of the sensor data in place.

Blockchain network is essentially a decentralized point-to-point network. To broadcast data from each sensor to the blockchain network in an efficient and robust way, we propose a gossip-based diffusion protocol, as detailed in the following.

1) Initialization. Initially, all peers in the blockchain network execute the discovery service and message exchange to maintain a dynamic list $L_{\text{alive}}^{\text{node}_{\text{ID}}}$ of live peers they can connect to. Such a list contains IP address, port, and public key of peer nodes.
2) Data multicasts to seed nodes. We call those nodes that participate in blockchain network and also initially being connected by sensors to receive data as *seed nodes*. Sensors periodically send collected data $d_1^{\text{sensor}_j}, \ldots, d_n^{\text{sensor}_j}, j = 1, \ldots, l$ to s seed nodes, the message is in the form of $\text{SEED} = \langle Z_{\text{ID}}, \text{sensor}_{\text{ID}}, d, ts \rangle_{\sigma_{\text{sensor}_{\text{ID}}}}$.
3) Processing on seed nodes. Each seed node maintains a local buffer BUF for received data from sensors and always check the signature validity before caching sensor data to BUF. Every δt_{seed}, each seed node accumulates SEEDs in BUF as a batch B_{seed}, and counts the number of data items in B_{seed} as $\tilde{n}$, which is used to check *data loss* later. SEED_i in this batch is sorted sequentially according to ts. It is practical to ensure that $\delta t_{\text{seed}} < \delta t_{\text{sensor}}$. Therefore, the unpredictable network impact is eliminated and all these s seed nodes have the same state of B_{seed} ready.
4) Gossip diffusion. We consider a synchronous network where message diffusion can be completed in δt_{dif}. In order to reduce the complexity incurred by message mixing and ensure the same number of data items on each seed node, we set $\delta t_{\text{dif}} < \delta t_{\text{seed}}$. The seed nodes then disseminate the gossip message $\text{GM} = \langle \text{node}_{\text{ID}}, B_{\text{seed}}, \tilde{n}, ts \rangle_{\sigma_{\text{node}_{\text{ID}}}}$ to all peers in the blockchain network through *gossip-based diffusion algorithm*, as shown in Algorithm 1. Note that as the system tolerates up to f malicious nodes, the number of seed nodes $k \geq f + 1$ ensures that malicious actions can be detected and, hence, not updated to the ledger.

The gossip-based diffusion algorithm in Algorithm 1 is divided into three steps as follows.

1) Topology construction: Practically, each peer node in the blockchain network maintains a list of its direct and indirect *neighbor nodes* whose information is stored in $L_{\text{alive}}^{\text{node}_{\text{ID}}}$. An *overlay network* is formed with virtual links from each peer node to its corresponding neighbor nodes. It is worth mentioning that the gossip-based algorithm can also be utilized to maintain the gossip network itself: peer nodes periodically exchange and update $L_{\text{alive}}^{\text{node}_{\text{ID}}}$ with each other so the network topology can be dynamically maintained when some nodes leave or join.
2) Peer selection: Each peer node in the blockchain network is initialized with a number of gossip parameters during the topology construction step, including: a local cache $C^{\text{node}_{\text{ID}}}$ with size γ; the maximum number t of times a message can be forwarded; and the number k of neighbor nodes a peer node selects to forward messages each time. Among these parameters, k plays a critical role in diffusion efficiency since the value of k and the selected nodes affect the dissemination speed. Previous study shows that constructing a gossip-based topology on top of a *peer sampling service* [30] can ensure a uniform and random selection of peers.
3) Data dissemination: Those k uniformly and randomly selected nodes are called *passive nodes* and the node starting to send messages is an *active node*. Their interaction is described as follows.
   a) Each seed node acts as an active node, and uniformly and randomly selects k nodes as passive nodes from its local cache $C^{\text{node}_{\text{ID}}}$. A gossip message GM is retrieved from BUF.

[5]Castro and Liskov [28] proved that a minimum of $3f + 1$ replicas/peers are needed to tolerate at most f faulty/Byzantine replicas/peers.

Algorithm 1: Gossip-based Message Diffusion.

Input: $GM = \langle node_{ID}, B_{\text{seed}}, \tilde{n}, ts \rangle_{\sigma_{\text{node}_{\text{ID}}}}$

Output: $true$ or $false$

1: **Initialization**: the number s of seed nodes; the number k of selected neighbor nodes; the maximum number t of times a message can be forwarded; the information about neighbor nodes stored in $L_{\text{alive}}^{\text{current\_node}_{ID}}$; the local cache $C^{\text{node}_{ID}}[\gamma]$ of size γ as a buffer for received messages;
2: $m, \sigma_{\text{node}_{ID}} \leftarrow parse(GM)$, where $m = \langle node_{ID}, B_{\text{seed}}, \tilde{n}, ts \rangle$;
3: $r \leftarrow verify(\sigma_{\text{node}_{ID}}, node_{pk} \leftarrow L_{\text{alive}}^{\text{current\_node}_{ID}}, m)$;
4: **if** $r == true$ **then**
5: **if** $C^{\text{node}_{ID}}$ *does not contain* m **then**
6: forward m with ts and $\sigma_{\text{current}_{\text{node}}}$ to k neighbor nodes selected from $L_{\text{alive}}^{\text{current}_{\text{node}_{ID}}}$;
7: add quadruple $(key = m,\ counter = 1,\ flag = false,\ integrity = false)$ to $C^{\text{node}_{ID}}$;
8: **while** $C^{\text{node}_{ID}}.size \geq \gamma$ **do**
9: remove those items whose $counter \geq t$ and $integrity$ is $true$;
10: **else**
11: **if** ($counter$ for m) $< t$ **then**
12: forward m with ts and $\sigma_{\text{current}_{\text{node}}}$ to k neighbor nodes selected from $L_{\text{alive}}^{\text{current}_{\text{node}_{ID}}}$;
13: $counter++$ for the item whose $key == m$;
14: **else**
15: **if** $flag$ for $m == false$ **then**
16: update $flag$ for m as $true$;
17: **return** true;
18: **else**
19: **return** false;

b) Each active node sends message GM to all of its corresponding passive nodes.
c) All passive nodes act as active nodes to repeat this process by randomly and uniformly choosing k nodes from their local cache and forwarding message GM.
d) Each node (including seed nodes) maintains a set of quadruples $(key, counter, flag, integrity)$ in the local cache $C^{\text{node}_{\text{ID}}}$, where the key is the gossip message GM, and *counter* is to count how many times GM is forwarded by the node. For efficiency, the hash value of GM is computed to quickly determine if the current node has already forwarded such a message. If the received message is already in its cache, we increase the counter; otherwise, the new item is added to the cache. If $counter \geq t$, we stop forwarding this message and consider it as *stable* by setting $flag$ to be $true$. In a synchronous network, all nodes are able to reach a *stable* status within a reasonable time period δt_{dif}. The $integrity$ is set to be $false$ by default, indicating whether or not this message is checked in the later *data loss* phase. If the total number of messages exceeds the cache size γ on the node, we remove those items whose $counter \geq t$ and $integrity$ is $true$.

In a gossip network with $\mathcal{N}$ nodes, a message sent from a seed node is relayed by a set of randomly selected k nodes in every round and is expected to reach all other nodes after θ rounds, i.e., $\sum_{i=0}^{\theta} k^i = \mathcal{N}$, and hence, $\theta = \lceil \log_k(1 - \mathcal{N}(1-k)) - 1 \rceil$. Especially when $k = 2$, the process turns to be a binary tree and the complexity of rounds becomes $O(\log \mathcal{N})$.

2) Phase II for Byzantine Consensus: We augment the consensus protocol to support censorship checking during the consensus process. Note that all full nodes have the input data in place after the *message diffusion* phase.

The consensus' leader node first sends a *PROPOSE* message containing $\mathcal{B}_{\text{seed}}$ to other replicas. All other replicas receive the *PROPOSE* message and then check the validity of the proposed batch and the sender's leadership: if both are true, then register $\mathcal{B}_{\text{seed}}$ and send a *WRITE* message containing a cryptographic hash of the batch, denoted as $H(\mathcal{B}_{\text{seed}})$, to all other replicas. If a replica receives $\lceil \frac{|R|+f+1}{2} \rceil$ *WRITE* messages with an identical hash, it sends an *ACCEPT* message containing this hash to all other replicas.

3) Phase III for Data Loss Check: If a replica receives $\lceil \frac{|R|+f+1}{2} \rceil$ *ACCEPT* messages for the same hash, it performs *FV* to detect data loss by comparing the number $\tilde{n}$ of messages in *PROPOSE* with the number of messages received from the sensors, i.e., n. If FV passes, it appends the new data $\mathcal{B}_{\text{seed}}$ to the ledger; otherwise, a *view change*[6] may take place to elect a new leader and all replicas are required to converge to the same consensus execution. More details can be found in [29].

We would like to point out that it might be more efficient to perform *FV* right after *PROPOSE* to avoid *WRITE* and *ACCEPT* if filtering was noticed. However, this would require modifying the original consensus, e.g., the BFT-SMART protocol. Our design only involves adding a few phases after the protocol finishes and, hence, facilitates quick implementation and convenient deployment.

B. Dealing With the Single Exit

In many blockchain-based IoT scenarios, sensors collect and send data to the ledger and meanwhile actuators receive instructions for further actions. These instructions could be the outcomes of some data analysis procedures applied to the collected data. Thus, we also need to ensure that the instruction from the blockchain network to the sensor network (referred to as "outbound flow") is "legitimate," i.e., the instruction is the consensus of the participants rather than a single node, and the instruction is successfully delivered to the intended actuator. Fig. 6 shows the normal operation on the outbound flow, as detailed in the following two phases.

1) Phase I for Decision Consensus: After the data batch is written to the ledger, each honest node executes a data analysis

[6]If all nodes executing the consensus protocol have the same leader, they are in the same view. Views are numbered consecutively, and the leader of a view is a replica p such that $p = v \bmod \mathcal{N}$, where v is the view number. Hence, when the leader is considered to fail, a view change is carried out by setting the new leader to be $p = (v+1) \bmod \mathcal{N}$ to continue consensus execution.

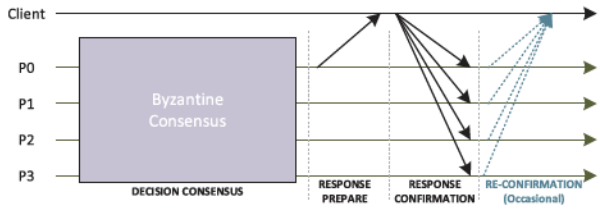

Fig. 6. Message pattern for dealing with the single exit.

program in the form of $y = \mathcal{F}(\text{GM})$. Program $\mathcal{F}$ is typically known *a priori* as it is application-specific and may vary in different scenarios. Algorithm 2 gives a simple example of how function $\mathcal{F}$ works: the input is the data batch that updated to ledger, which contains sensor data from different zones, by calculating the average value of sensors from the same zone and comparing with the threshold $\mathcal{T}$, corresponding instruction from a predefined set $\mathcal{Y}$ is returned, otherwise no action is needed by returning $\perp$. The decision consensus phase executes the same steps as the aforementioned Byzantine consensus phase. The only difference is the content to be agreed on, which is the output y instead of data batch $\mathcal{B}_{\text{seed}}$. We split it into two rounds for different consensus contents, because in some cases, such as the data collection system, only the data needs to be recorded in the ledger, whereas in other cases, it may need both. At the end of the decision consensus phase, the output based on the sensor data is updated to the ledger as well. Then, the execution of *RESPONSE* phase is triggered.

2) Phase II for Response: The response phase includes prepare, confirmation, and occasional reconfirmation. In *RESPONSE PREPARE*, the honest consensus' leader forward command/instruction y to the actuator if it is the agreed outcome, which means that it has collected sufficient[7] signatures from peer nodes on the same y. Once the actuator receives an instruction and verifies all signatures, it enters into the *RESPONSE CONFIRMATION* phase, in which the actuator simply broadcasts the signed acknowledgment ack_σ to all servers: this is done the same way as in the DIFFUSION phase via gossip. Other nonleader replicas wait for ack_σ after updating y to the ledger. If they do not receive an acknowledgment within a predefined time period, they all resend y to the actuator by themselves, also via gossip.

C. Security Discussion

1) Security Against Entry Point Censorship: We first analyze the security against entry point censorship, i.e., data flow from the sensor network to the blockchain network. Existing blockchain-based IoT management systems rely on a single gateway node to relay messages. We propose to replace the single gateway server with $f + 1$ full nodes (i.e., "seed nodes") in the blockchain network, which not only participate in the blockchain operations but also act as conventional gateway nodes for message forwarding. As defined in our security

[7]For crash fault tolerance, sufficient refers to at least $f + 1$ peer nodes, whereas for Byzantine fault tolerance, sufficient refers to at least $2f + 1$ peer nodes, proof can be found in [28].

Algorithm 2: An Example of Analysis Program $\mathcal{F}$.

Input: $\mathcal{B}_{\text{seed}}$
Output: $y/\perp$

Initialization: the instruction set $\mathcal{Y}$; a temperature threshold: $\mathcal{T} \in \mathbb{Z}$; the sum of temperatures collected by all sensors in a specific zone: $\mathcal{S}_{Z_{ID}} \leftarrow 0$; the counter that keeps track of the number of times the sensor data is counted: $t_{Z_{ID}} \leftarrow 0$
$SEED_i, i \in [n] \leftarrow parse(\mathcal{B}_{\text{seed}})$;
for $SEED_i$ **do**
  $\langle Z_{ID}, d \rangle \leftarrow parse(SEED_i)$;
  if Z_{ID} exists **then**
    $\mathcal{S}_{Z_{ID}} \leftarrow \mathcal{S}_{Z_{ID}} + d$;
    $t_{Z_{ID}} + +$;
  else
    $new(\langle Z_{ID}, \mathcal{S}_{Z_{ID}}, t_{Z_{ID}} \rangle)$;
    $\mathcal{S}_{Z_{ID}} \leftarrow d$;
    $t_{Z_{ID}} \leftarrow 1$;
for all Z_{ID} **do**
  $\langle Z_{ID}, AVG_{Z_{ID}} \rangle \leftarrow \langle Z_{ID}, \frac{\mathcal{S}_{Z_{ID}}}{t_{Z_{ID}}} \rangle$;
  if $AVG_{Z_{ID}} > \mathcal{T}$ **then**
    return $y \leftarrow \mathcal{Y}$;
return $\perp$;

model, the security against entry point censorship requires that the probability of bad events B_1 and B_2 is negligibly small. Specifically, B_1 includes following three cases.

i) The malicious gateway node may drop some messages.
ii) The malicious gateway node may infinitely delay messages without relaying to the blockchain network.
iii) The malicious gateway node may modify message contents and send modified messages to a subset of peer nodes.

B_2 also includes following three cases.

i) Some peer nodes cannot receive messages from the sensor network, therefore failing to update the ledger.
ii) All nodes update messages to the ledger successfully, but the number of messages on some nodes is less than what have been sent from the sensor network.
iii) All nodes successfully update a correct number of messages in their ledger, but in a different order.

We now discuss how our protocol prevents the aforementioned cases in a synchronous network between sensors and full nodes. For consensus, we can still handle a partially synchronous network among the full nodes. To tolerate Byzantine fault, the total number of peer nodes in the blockchain network is expected to be at least $3f + 1$, where f is the maximum number of faulty nodes. For Case i) in B_1, having $f + 1$ seed nodes instead of one single gateway node ensures that at least one honest node be selected. With a more robust gossip-based diffusion mechanism, the honest node relays a correct number of messages to all peer nodes in the blockchain network. For Case ii) in B_1, since at most f nodes can infinitely delay the messages, having $f + 1$ seed nodes ensures that at least one honest node be selected and then gossip messages to all other peer nodes. For Case iii)

in B_1, even though malicious nodes may modify messages, the signature verification ensures that the modified data be rejected and never updated to the ledger. For Cases i) and ii) in B_2, we propose an augmented consensus protocol, where an FV phase, which is added to the regular BFT consensus protocol, checks data loss before updating the ledger by comparing the number of data messages after consensus with the number of data items received in the gossip-based diffusion process. If equal, they are updated to the ledger; otherwise, a view change is triggered to reach consensus again. After at most f rounds, data are correctly updated to the ledger since an honest node is selected to be the leader. Case iii) in B_2 is addressed by the Byzantine consensus with proved security. In sum, our proposed protocol satisfies the security requirements on *entry point censorship resistance*.

2) Security Against Exit Point Censorship: We now analyze the security against exit point censorship, i.e., data flow from the blockchain network to the sensor network. Unlike permissionless blockchain, such as Ethereum, where all participants maintain one public chain, in permissioned blockchain, such as hyperledger fabric, each peer node maintains a copy of the ledger, which can be modified locally. The blockchain-based IoT management system not only stores data in the ledger, but also is expected to make proper decisions and communicate with corresponding actuators. Generally, only one "leader" node is responsible for sending instructions to actuators for efficiency since actuators are typically equipped with limited computing power. If the "leader" node is compromised, it may send modified instructions to actuators; and infinitely delay the instructions.

Our protocol solves the first problem by verifying the aggregated signatures from most (specifically, at least $2f + 1$) of the peer nodes: if failed, no action is taken by the actuator; otherwise, a correct instruction is executed. In the second problem, the actuator does not receive any instruction from the malicious leader. In both of these cases, if the actuator takes no action, no acknowledgment is sent back to the peer nodes, and then an occasional *reconfirmation* is triggered, where all peer nodes jointly inform the actuator.

An alternative solution is to initiate a view change to select a new leader. Since there are at most f malicious nodes, eventually an honest node is selected to send instructions to the actuator. However, this process may be repeated for f times in the worst case and is not suitable for time-critical IoT scenarios. In our protocol, the leader is honest in most cases, so *reconfirmation* is rare. Even if it happens, all nodes just need to send *once* and the actuator is guaranteed to receive the instruction even though it may need to communicate with more peer nodes instead of only the "leader" as in the alternative solution. Hence, our protocol satisfies the security requirement on *exit point censorship resistance*.

D. Reducing Verification Complexity via Public-Key Aggregation

In blockchain-based IoT systems, some processes have similar properties, e.g., sensors send collected data to the blockchain network, and peer nodes send instruction back to actuators. More specifically, in the former, $\text{sensor}_1, \ldots, \text{sensor}_k$ possessing their public keys $\text{pk}_{s_1}, \text{pk}_{s_2}, \ldots, \text{pk}_{s_k}$ send messages $m_1, m_2, \ldots, m_k$ and corresponding signatures $\sigma_{s_1}, \sigma_{s_2}, \ldots, \sigma_{s_k}$ to nodes in the blockchain network and get verified. Similarly, in the latter, all nonleader replicas $r_1, r_2, \ldots, r_n$ who owns public keys $\text{pk}_{r_1}, \text{pk}_{r_2}, \ldots, \text{pk}_{r_n}$ send instructions/commands $c_1, c_2, \ldots, c_n$ together with signatures $\sigma_{r_1}, \sigma_{r_2}, \ldots, \sigma_{r_n}$ to the leader for forwarding. In these cases, it is important to exploit a more efficient and secure way to verify signatures. By leveraging the work in [31], we propose to leverage the modified BLS multisignature aggregation scheme (referred to as *public key aggregation*) to reduce the communication and verification complexity, based on the following considerations.

1) Multiple sensors of the same type are typically deployed in a region for improved fault tolerance and sensing accuracy, and some of them may very likely collect identical measurements. Using a signature aggregation mechanism is efficient but may suffer from *rouge public-key attack* [31].
2) Prepending the sensor's public key to the collected data before signing defends against the aforementioned attack, but would not be able to make full use of the advantages brought by aggregating identical messages.

The adoption of *public-key aggregation* defends against *rouge public-key attack* while achieving efficiency. We take the second scenario as an example to explain the application of this scheme, which contains the following components.

1) A bilinear pairing $e : G_0 \times G_1 \to G_T$. The pairing is efficiently computable and nondegenerated. All three groups have prime order q. Let g_0 and g_1 be the generator of G_0 and G_1, respectively.
2) Two hash functions $H_0 : \mathcal{M} \to G_0; H_1 : G_1^n \to R^n$ where $R := 1, 2, \ldots, 2^{128}$ and $1 \le n \le \tilde{N}$. These two hash functions are treated as random oracle in the security analysis.

With these components, the scheme works as follows.

1) $KenGen()$: Choose a random $\alpha \xleftarrow{R} Z_q$ and set $h \leftarrow g_1^\alpha \in G_1$, output $pk := (h)$ and $sk := (\alpha)$.
2) $Sign(sk, c_i)$: Sign command c_i and output $\sigma_i \leftarrow H_0(c_i)^\alpha \in G_0$, where $i = \{1, 2, \ldots, n\}$ denotes different replicas in the blockchain network.
3) $Aggregate((pk_{r_1}, \sigma_1), \ldots, (pk_{r_n}, \sigma_n))$.
   a) $compute$: $(t_1, \ldots, t_n) \leftarrow H_1(pk_{r_1}, \ldots, pk_{r_n}) \in R^n$.
   b) $output$: $\sigma \leftarrow \sigma_1^{t_1} \cdots \sigma_n^{t_n} \in G_0$.
4) $Verify(pk_{r_1}, \ldots, pk_{r_n}, c_i, \sigma_i)$: To verify the multisignature σ_i on c_i, we do
   1) $compute$: $(t_1, \ldots, t_n) \leftarrow H_1(pk_{r_1}, \ldots, pk_{r_n}) \in R^n$.
   2) $compute$: $apk \leftarrow pk_1^{t_1} \cdots pk_n^{t_n} \in G_1$.
   3) if $e(g_1, \sigma_i) = e(apk, H_0(c_i))$, output "accept;" otherwise, output "reject."

The aforementioned scheme is for verifying multiple signatures on one message. If messages keep flowing, it is more efficient to verify as a batch. Specifically, consider a triple (m_i, σ_i, apk_i) for $i = 1, 2, \ldots, b$, where b is the number of messages in one batch. If m_i are all distinct, then

1) $compute$: $\tilde{\sigma} \leftarrow \sigma_1 \cdots \sigma_b \in G_1$.

TABLE II
TIME COST (μS) FOR GOSSIPING 10 DATA ITEMS AMONG 20 NODES WHEN THE NUMBER OF RANDOMLY SELECTED NODES IS $k = 2$ ($f = 1$)

data item	$node_1$ $node_8$ $node_{15}$	$node_2$ $node_9$ $node_{16}$	$node_3$ $node_{10}$ $node_{17}$	$node_4$ $node_{11}$ $node_{18}$	$node_5$ $node_{12}$ $node_{19}$	$node_6$ $node_{13}$ $node_{20}$	$node_7$ $node_{14}$
$data_1$	60.50 136.93 80.00	117.07 N/A 133.98	51.71 409.86 96.62	144.50 51.39 69.79	98.26 101.74 68.38	134.14 1344.76 128.41	80.63 63.62
$data_2$	84.41 139.02 147.82	80.53 N/A 59.28	101.59 43.71 142.48	74.35 185.52 83.53	106.51 69.14 73.02	67.05 213.43 119.98	200.39 39.30
$data_3$	82.43 75.90 150.59	73.03 N/A 79.74	67.87 48.69 85.01	126.23 60.80 93.48	93.82 106.11 59.63	90.11 158.98 51.75	82.26 101.95
$data_4$	72.71 93.34 64.69	69.62 N/A 93.54	61.69 76.44 56.40	50.74 278.56 63.61	90.80 76.21 96.47	54.72 75.67 70.70	86.53 51.99
$data_5$	93.55 55.45 125.75	51.58 N/A 96.41	62.91 52.57 89.15	138.72 69.84 53.09	82.08 204.22 99.21	57.58 102.44 72.90	77.37 108.16
$data_6$	68.89 84.82 72.95	138.12 N/A 126.46	135.25 71.65 137.50	107.57 163.25 72.12	119.76 141.01 54.61	87.59 93.53 192.35	125.79 90.69
$data_7$	81.70 97.63 91.89	224.80 N/A 129.23	143.63 91.69 94.73	102.91 503.61 188.51	218.64 121.75 70.89	41.81 136.32 149.41	66.95 301.45
$data_8$	148.93 116.07 76.55	62.49 N/A 77.04	65.33 59.75 59.52	92.26 129.75 90.74	64.11 117.55 122.95	76.71 99.19 98.50	97.23 46.46
$data_9$	121.29 131.41 105.10	133.56 N/A 69.21	57.62 107.72 145.50	88.09 327.55 158.08	70.90 115.38 70.53	199.61 88.92 100.76	151.89 68.05
$data_{10}$	102.90 61.27 134.91	141.17 N/A 71.56	54.24 128.00 131.73	99.74 348.11 74.40	104.74 78.43 126.71	46.35 147.39 167.69	74.01 92.11

TABLE III
TIME COST (μS) FOR GOSSIPING 10 DATA ITEMS AMONG 20 NODES WHEN THE NUMBER OF RANDOMLY SELECTED NODES IS $k = 3$ ($f = 2$)

data item	$node_1$ $node_8$ $node_{15}$	$node_2$ $node_9$ $node_{16}$	$node_3$ $node_{10}$ $node_{17}$	$node_4$ $node_{11}$ $node_{18}$	$node_5$ $node_{12}$ $node_{19}$	$node_6$ $node_{13}$ $node_{20}$	$node_7$ $node_{14}$
$data_1$	72.16 73.03 46.54	81.20 45.34 47.20	50.68 41.27 68.30	101.66 N/A 45.10	58.62 71.08 87.00	N/A 46.92 47.00	76.00 52.28
$data_2$	79.63 52.24 54.60	69.18 66.42 31.34	85.13 45.30 101.54	90.27 N/A 39.41	69.96 35.01 41.14	N/A 62.02 30.03	148.03 59.51
$data_3$	153.47 67.15 177.58	55.83 98.18 79.86	71.87 274.19 160.13	48.58 N/A 110.08	214.83 93.82 119.05	N/A 78.51 80.61	60.77 47.99
$data_4$	100.29 93.59 61.52	92.14 79.76 52.32	112.21 49.42 51.36	83.30 N/A 59.81	49.52 166.90 135.62	N/A 74.64 93.50	102.98 188.76
$data_5$	167.31 75.94 67.40	88.71 174.55 63.96	102.54 63.22 1275.38	94.04 N/A 69.11	62.50 136.31 49.55	N/A 101.63 57.92	1331.69 113.70
$data_6$	93.42 153.27 51.51	73.39 60.94 101.60	80.91 182.91 79.96	86.55 N/A 262.72	1281.28 1354.23 111.56	N/A 203.38 83.46	229.66 116.79
$data_7$	213.23 167.77 135.81	139.07 107.18 302.83	58.76 70.49 156.99	59.35 N/A 76.32	144.79 133.13 73.53	N/A 65.66 175.72	98.59 147.38
$data_8$	193.83 188.89 183.37	1237.54 1366.60 50.98	125.09 257.06 126.24	71.34 N/A 100.72	81.56 54.04 255.26	N/A 186.03 74.88	157.84 85.08
$data_9$	83.83 116.10 1409.63	111.79 1306.69 107.64	1260.80 88.61 1279.95	211.28 N/A 175.80	72.46 77.56 130.17	N/A 151.64 128.47	106.79 306.70
$data_{10}$	87.49 264.33 230.98	170.90 108.63 219.01	121.01 163.98 145.66	229.18 N/A 117.19	108.43 100.71 123.27	N/A 1252.64 190.19	95.98 185.87

2) Accept all b tuples as valid iff $e(g_1, \tilde{\sigma}) = e(apk_1, H_0(m_1)) \cdots e(apk_b, H_0(m_b))$.

If there are identical messages in m_i, then

1) $obtain : \rho_1, \ldots, \rho_b \xleftarrow{R} 1, 2, \ldots, 2^{64}$.
2) $compute : \tilde{\sigma} \leftarrow \sigma_1^{\rho_1} \cdots \sigma_b^{\rho_b} \in G_1$.
3) Accept all b tuples as valid iff $e(g_1, \tilde{\sigma}) = e(apk_1^{\rho_1}, H_0(m_1)) \cdots e(apk_b^{\rho_b}, H_0(m_b))$.

Thus, verifying b messages requires only $b + 1$ instead of $2b$ pairings if verifying one at a time. Therefore, such a batch-based mechanism can further improve verification efficiency.

VI. IMPLEMENTATION AND EVALUATION

To shed some light on the behavior of how the gossip-based diffusion mechanism resist the censorship of the single entry point (for the single exit problem, it also relies on gossip-based message dissemination to send instructions to actuators/devices, we do not repeat the redundant evaluation of such process), we implement the gossip-based message dissemination process[8] with following settings.

1) We create a gossip network testbed using 21 *Google Cloud* virtual machine (VM) instances,[9] each of which is equipped with 3.75 GB memory and 1 vCPU and has JRE 1.8.0_181 installed on Ubuntu 16.04.
2) The bandwidth between nodes in the same zone is 1.96 Gb/s, whereas across different zones, it is at least 700 Mb/s. Since the message size is relatively small, these bandwidths make the data transfer time negligible compared with the protocol execution time.
3) We gossip 10 data items from one client to the other 20 peer nodes. Each data item is a short string of about 6 B, and each peer node has a local cache large enough to buffer 10 data items.

Tables II–VII tabulate the time cost for gossiping all ten data items to 20 peer nodes with different numbers of randomly selected peer nodes in each round of the gossip protocol. Fig. 7(a)–(f) plots the number of received data items on each peer node during a certain period of time, where a red line represents the maximum number of tolerated malicious nodes, whereas a blue line represents the behavior of honest nodes. These results illustrate censorship resistance where the gossip-based diffusion mechanism guarantees that all data items sent from the client be delivered to all honest peer nodes in the blockchain network.

We calculate the average time cost and standard deviation for gossiping one data item from the "sensor network" (client node) to all other peer nodes in the blockchain network, as shown in Fig. 8 and Table VIII. We have the following observations and explanations.

1) The variation in the average time cost with different selected neighbor nodes in each gossip round is caused by the nonuniformity when randomly selecting

[8]The implementation source code of the gossip-based message dissemination can be found online. [Online]. Available: *https://github.com/Blockchain-World/gossip-based-diffusion.git*

[9]In our gossip testbed, one VM acts as a client, whereas the other 20 VMs act as peer nodes. These VMs are located in different zones: $node_1$ to $node_8$ reside in the same zone (*us-east1-b*), $node_9$ to $node_{15}$ reside in *u-east4-c*, and $node_{16}$ to $node_{20}$ reside in $us - central1 - c$.

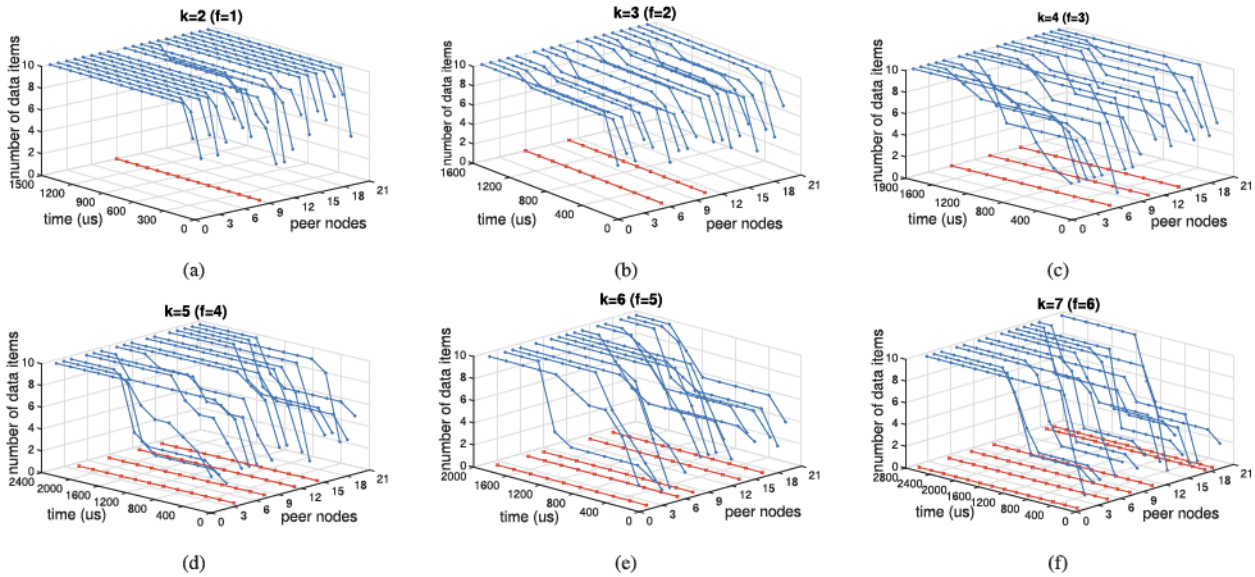

Fig. 7. Diffusion time of gossiping ten data items among 20 peer nodes when the number of selected nodes is (a) $k = 2$ (i.e., at most one faulty node), (b) $k = 3$ (i.e., at most two faulty nodes), (c) $k = 4$ (i.e., at most three faulty nodes), (d) $k = 5$ (i.e., at most four faulty nodes), (e) $k = 6$ (i.e., at most five faulty nodes), and (f) $k = 7$ (i.e., at most six faulty nodes).

TABLE IV
TIME COST (μS) FOR GOSSIPING TEN DATA ITEMS AMONG 20 NODES WHEN THE NUMBER OF RANDOMLY SELECTED NODES IS $k = 4$ ($f = 3$)

data item	$node_1$ $node_8$ $node_{15}$	$node_2$ $node_9$ $node_{16}$	$node_3$ $node_{10}$ $node_{17}$	$node_4$ $node_{11}$ $node_{18}$	$node_5$ $node_{12}$ $node_{19}$	$node_6$ $node_{13}$ $node_{20}$	$node_7$ $node_{14}$
$data_1$	49.61	54.29	49.59	42.05	46.90	N/A	103.73
	46.95	73.14	42.19	N/A	35.62	44.54	74.65
	N/A	29.00	48.11	49.16	58.21	49.56	
$data_2$	70.92	39.75	63.78	34.51	54.41	N/A	221.73
	71.91	55.97	53.46	N/A	40.60	75.32	56.94
	N/A	33.20	58.19	75.02	41.98	60.33	
$data_3$	125.02	86.54	91.96	113.19	61.11	N/A	171.00
	173.21	45.86	149.43	N/A	51.58	80.06	195.30
	N/A	108.79	83.75	83.72	66.93	77.13	
$data_4$	369.22	159.84	288.61	1235.96	1355.85	N/A	83.25
	258.82	129.93	81.25	N/A	87.56	71.44	88.11
	N/A	1209.44	156.07	68.27	49.62	108.74	
$data_5$	561.57	228.60	132.52	1247.68	303.08	N/A	1436.17
	104.32	106.26	1252.60	N/A	200.62	60.51	138.66
	N/A	87.84	74.76	284.95	125.78	114.32	
$data_6$	444.75	166.68	224.24	1239.37	1347.34	N/A	109.89
	459.40	159.35	102.45	N/A	48.14	179.36	250.17
	N/A	1231.11	104.85	61.08	206.51	53.74	
$data_7$	660.45	540.41	251.05	1224.38	250.90	N/A	186.79
	76.27	1269.52	1246.57	N/A	171.92	1298.89	223.85
	N/A	132.07	1231.56	184.51	237.51	87.30	
$data_8$	64.75	452.97	317.85	149.46	199.93	N/A	1596.44
	110.13	111.02	112.19	N/A	126.10	82.48	81.83
	N/A	95.90	1272.23	147.35	1364.31	182.67	
$data_9$	328.57	184.45	1634.37	135.30	274.91	N/A	158.08
	384.93	148.61	97.04	N/A	115.40	1253.69	149.83
	N/A	69.68	173.44	352.46	136.82	280.59	
$data_{10}$	730.07	296.96	1476.63	82.42	1532.96	N/A	1636.20
	472.78	123.94	1399.96	N/A	1383.01	1247.79	135.83
	N/A	1223.67	121.18	430.11	168.66	139.13	

TABLE V
TIME COST (μS) FOR GOSSIPING TEN DATA ITEMS AMONG 20 NODES WHEN THE NUMBER OF RANDOMLY SELECTED NODES IS $k = 5$ ($f = 4$)

data item	$node_1$ $node_8$ $node_{15}$	$node_2$ $node_9$ $node_{16}$	$node_3$ $node_{10}$ $node_{17}$	$node_4$ $node_{11}$ $node_{18}$	$node_5$ $node_{12}$ $node_{19}$	$node_6$ $node_{13}$ $node_{20}$	$node_7$ $node_{14}$
$data_1$	55.87	36.39	50.92	N/A	47.22	42.31	40.93
	N/A	52.76	38.34	36.71	N/A	65.65	73.09
	N/A	58.79	40.93	42.26	92.79	44.45	
$data_2$	45.27	51.13	117.17	N/A	224.54	66.23	51.51
	N/A	71.97	46.51	31.61	N/A	59.10	46.58
	N/A	76.26	77.72	55.37	73.22	58.32	
$data_3$	80.84	159.70	71.77	N/A	85.53	47.87	60.50
	N/A	129.23	55.10	77.80	N/A	138.85	117.45
	N/A	64.15	116.20	78.34	84.52	48.97	
$data_4$	105.93	98.34	1315.75	N/A	290.46	130.91	242.06
	N/A	98.35	205.84	102.34	N/A	126.82	178.52
	N/A	99.85	94.33	175.46	108.43	170.60	
$data_5$	601.42	1438.03	1365.96	N/A	90.77	495.21	298.85
	N/A	215.95	69.87	58.29	N/A	94.83	309.71
	N/A	348.68	112.16	164.90	165.64	89.70	
$data_6$	298.15	1347.24	1339.27	N/A	492.82	262.78	264.83
	N/A	239.60	119.43	148.51	N/A	303.59	266.85
	N/A	218.26	80.72	259.66	1312.92	1350.29	
$data_7$	385.64	1266.17	1498.72	N/A	319.04	197.40	563.89
	N/A	465.00	110.75	428.42	N/A	131.63	243.92
	N/A	185.48	1334.29	138.77	1300.00	69.35	
$data_8$	1083.20	1395.76	1430.80	N/A	974.97	207.28	2034.26
	N/A	1752.41	246.07	221.54	N/A	423.38	413.10
	N/A	417.54	1246.26	129.29	1233.45	1313.09	
$data_9$	1176.69	1386.14	1644.69	N/A	878.08	706.89	1951.65
	N/A	392.67	339.96	425.37	N/A	324.68	193.75
	N/A	470.53	1319.52	265.23	1471.77	1244.40	
$data_{10}$	1123.53	1432.25	1206.90	N/A	1087.05	747.34	190.38
	N/A	376.03	614.53	175.25	N/A	218.43	154.91
	N/A	356.40	1355.53	484.60	1298.50	1320.61	

neighbor nodes. In our implementation, we use Java *Math.random()* with pseudorandomness to generate k random numbers as the selected indices of neighbor nodes. In some rounds, it is possible that the same target nodes are selected by different peer nodes and some nodes are not covered until after a few rounds and eventually receive messages. This process could be optimized using *peer sampling service* as mentioned in Section V, but we do not focus on gossip optimization in this article.

2) The standard deviation is relatively large. The measured time cost includes message transmission, neighbor selection, and consecutive writing (I/O) operations for recording timestamp. The network condition and the randomness in peer selection affect the time cost of

TABLE VI
TIME COST (μS) FOR GOSSIPING TEN DATA ITEMS AMONG 20 NODES WHEN THE NUMBER OF RANDOMLY SELECTED NODES IS $k = 6$ ($f = 5$)

data item	$node_1$ $node_8$ $node_{15}$	$node_2$ $node_9$ $node_{16}$	$node_3$ $node_{10}$ $node_{17}$	$node_4$ $node_{11}$ $node_{18}$	$node_5$ $node_{12}$ $node_{19}$	$node_6$ $node_{13}$ $node_{20}$	$node_7$ $node_{14}$
$data_1$	72.90	N/A	48.40	111.91	59.63	N/A	78.31
	N/A	88.40	75.64	49.43	37.31	53.01	N/A
	38.95	52.37	N/A	61.63	48.80	41.42	
$data_2$	39.99	N/A	122.83	58.82	174.98	N/A	54.71
	N/A	62.92	52.10	39.29	105.01	68.89	N/A
	63.66	64.75	N/A	96.14	87.35	50.09	
$data_3$	146.65	N/A	1238.31	204.31	131.86	N/A	171.87
	N/A	135.06	64.73	95.74	73.38	133.15	N/A
	46.06	84.14	N/A	141.97	1249.26	80.16	
$data_4$	273.48	N/A	1328.33	325.70	317.29	N/A	619.28
	N/A	98.62	444.27	150.34	382.90	113.64	N/A
	92.50	1200.88	N/A	127.94	126.34	128.25	
$data_5$	262.95	N/A	88.81	875.24	351.08	N/A	543.84
	N/A	446.03	202.89	230.57	176.73	1276.05	N/A
	121.93	165.45	N/A	84.82	1277.71	1403.73	
$data_6$	121.92	N/A	1455.44	633.24	412.15	N/A	685.17
	N/A	513.21	289.91	221.19	168.03	1505.37	N/A
	1445.69	1269.85	N/A	1491.26	1287.82	70.76	
$data_7$	466.80	N/A	1322.30	862.75	205.44	N/A	778.32
	N/A	467.96	331.99	139.22	96.24	1477.77	N/A
	1206.49	1337.19	N/A	1290.92	1477.27	1236.83	
$data_8$	737.75	N/A	1445.36	678.37	248.30	N/A	421.39
	N/A	208.55	256.65	105.52	86.90	165.05	N/A
	154.74	1237.12	N/A	1351.64	129.78	1219.45	
$data_9$	324.22	N/A	1433.01	396.86	465.32	N/A	293.46
	N/A	272.17	460.72	358.83	391.82	1407.55	N/A
	1497.40	1577.14	N/A	1371.77	170.80	178.70	
$data_{10}$	958.87	N/A	1404.70	748.53	629.36	N/A	699.68
	N/A	504.40	244.48	505.99	471.85	1627.26	N/A
	1214.22	1246.75	N/A	1405.41	1244.37	1482.71	

TABLE VII
TIME COST (μS) FOR GOSSIPING TEN DATA ITEMS AMONG 20 NODES WHEN THE NUMBER OF RANDOMLY SELECTED NODES IS $k = 7$ ($f = 6$)

data item	$node_1$ $node_8$ $node_{15}$	$node_2$ $node_9$ $node_{16}$	$node_3$ $node_{10}$ $node_{17}$	$node_4$ $node_{11}$ $node_{18}$	$node_5$ $node_{12}$ $node_{19}$	$node_6$ $node_{13}$ $node_{20}$	$node_7$ $node_{14}$
$data_1$	N/A	52.69	33.85	37.09	N/A	42.30	50.19
	N/A	45.66	59.25	N/A	37.69	52.24	70.85
	43.00	51.20	29.90	N/A	N/A	36.30	
$data_2$	N/A	166.74	126.15	52.04	N/A	55.86	90.42
	N/A	90.68	118.43	N/A	46.45	102.84	49.16
	105.94	107.65	120.37	N/A	N/A	109.78	
$data_3$	N/A	102.84	1422.05	115.60	N/A	153.74	82.62
	N/A	1326.52	88.76	N/A	128.37	136.39	81.18
	74.88	114.59	1226.92	N/A	N/A	87.00	
$data_4$	N/A	254.60	72.63	1244.16	N/A	344.83	1527.74
	N/A	1261.40	75.97	N/A	114.40	110.87	57.96
	1221.59	1213.46	125.18	N/A	N/A	1303.74	
$data_5$	N/A	367.17	1277.81	1293.56	N/A	377.69	1261.73
	N/A	111.56	1230.87	N/A	1415.93	379.12	1201.91
	1203.99	2454.67	2410.35	N/A	N/A	1174.64	
$data_6$	N/A	264.14	1285.27	1319.37	N/A	178.47	1396.43
	N/A	1372.35	1189.78	N/A	1252.64	192.50	1408.86
	1200.44	171.72	148.90	N/A	N/A	1427.69	
$data_7$	N/A	466.12	1496.56	1559.58	N/A	458.95	1381.26
	N/A	1249.34	1361.33	N/A	151.86	189.12	1196.91
	1209.80	2324.99	1220.54	N/A	N/A	1332.62	
$data_8$	N/A	395.74	1264.71	1471.02	N/A	606.01	1336.43
	N/A	1292.83	1182.83	N/A	1441.96	348.94	1279.97
	1216.36	2314.28	2359.55	N/A	N/A	1254.99	
$data_9$	N/A	411.55	1199.05	1526.10	N/A	449.34	1330.48
	N/A	1432.40	1240.37	N/A	120.43	400.28	1180.35
	1277.42	1270.88	140.94	N/A	N/A	1206.55	
$data_{10}$	N/A	459.34	1330.47	1396.47	N/A	144.30	1258.29
	N/A	1228.21	1186.25	N/A	1204.61	256.10	146.64
	1213.28	2344.26	2375.10	N/A	N/A	1309.11	

message delivery, i.e., some nodes may receive messages sooner than others.

3) There exists a slowly increasing trend in the average time cost, as more selected neighbor nodes result in more traffic in the network. In some rounds, the randomly selected neighbor nodes may have already been selected in previous rounds, also contributing to the increase in the average time cost.

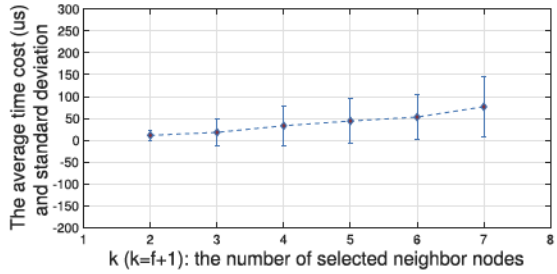

Fig. 8. Average time cost and standard deviation for gossiping one data item among 20 peer nodes with different numbers of selected neighbor nodes in each round of the gossip protocol.

TABLE VIII
AVERAGE TIME COST (μS) FOR GOSSIPING ONE DATA ITEMS AMONG 20 NODES WITH DIFFERENT NUMBERS OF RANDOMLY SELECTED NEIGHBOR NODES IN EACH ROUND

k (k=f+1)	The Average Time Cost (us)	Standard Deviation
k=2 (f=1)	11.43	10.86
k=3 (f=2)	18.21	29.33
k=4 (f=3)	33.32	44.79
k=5 (f=4)	44.16	51.24
k=6 (f=5)	53.13	51.54
k=7 (f=6)	76.73	67.58

VII. CONCLUSION

In this article, we proposed a protocol to tackle the censorship problem, which is important but remains largely unexplored in blockchain (hyperledger fabric) based IoT systems. For data flows from a sensor network to a blockchain network, we overcame potential censorship on a gateway node by employing gossip-based diffusion protocol to achieve guaranteed message delivery. Moreover, we improved the consensus protocol by checking data loss before writing to the ledger and replicating process outcome to facilitate opportunistic outcome delivery. Finally, we leveraged the cryptographic tool of public key aggregation to reduce communication and verification complexity, and analyzed the security of our protocol. We implemented the proposed gossip-based diffusion algorithm and illustrated message delivery with censorship resistance in the presence of faulty nodes. The proposed protocol has potential to improve the security of blockchain-based IoT management systems.

REFERENCES

[1] S. He, Q. Tang, and C. Wu, "Censorship resistant decentralized IoT management systems," in *Proc. 1st Int. Workshop Distrib. Ledger Things, Conjunction MobiQuitous*, New York, NY, USA, Nov. 5–7, 2018, pp. 454–459.

[2] D. Boswarthick, O. Elloumi, and O. Hersent, *M2M Communications: A Systems Approach*. Hoboken, NJ, USA: Wiley, 2012.

[3] R. V. Kranenburg, *The Internet of Things: A Critique of Ambient Technology and the All-Seeing Network of RFID*. Amsterdam, The Netherlands: Inst. Netw. Cultures, 2008.

[4] R. Van Der Meulen, "Gartner says 6.4 billion connected 'things' will be in use in 2016, up 30 percent from 2015," Gartner, Stamford, CT, USA, Nov. 10, 2015. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2015-11-10-gartner-says-6-billion-connected-things-will-be-in-use-in-2016-up-30-percent-from-2015

[5] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Bus. Horizons*, vol. 58, no. 4, pp. 431–440, 2015.
[6] K. Landernas, "Industrial Internet of Things, services, and people," ABB Corporate Research, 2015. [Online]. Available: http://www.processitinnovations.se/Sve/Nyheter/Filer/IndustrialIoTSP.pdf
[7] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and IoT integration: A systematic survey," *Sensors*, vol. 18, 2018, Art. no. 2575.
[8] E. Androulaki *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proc. 13th ACM EuroSys Conf.*, 2018, pp. 30:1–30:15.
[9] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.
[10] P. Brody and V. Pureswaran, "Device democracy: Saving the future of the Internet of Things," Executive Report, IBM Inst. Business Value, Cambridge, MA, USA, Tech. Rep. GBE03620-USEN-04, Sep. 2014.
[11] J. Kang, R. Yu, X. Huang, S. Maharjan, Y. Zhang, and E. Hossain, "Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains," *IEEE Trans. Ind. Inform.*, vol. 13, no. 6, pp. 3154–3164, Dec. 2017.
[12] "Introduction to hyperledger smart contracts for IoT best practices and patterns," IBM Watson IoT, 2017. Accessed: Dec. 11, 2017. [Online]. Available: https://github.com/ibm-watson-iot/blockchain-samples/blob/master/docs/iotcp/HyperledgerContractsIntroBestPracticesPatterns.md
[13] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "IoT gateway: Bridging-wireless sensor networks into Internet of Things," in *Proc. IEEE/IFIP Int. Conf. Embedded Ubiquitous Comput.*, 2010, pp. 347–352.
[14] L. d. T. Steenkamp, S. Kaplan, and R. H. Wilkinson, "Wireless sensor network gateway," in *Proc. IEEE AFRICON*, 2009, pp. 1–6.
[15] N. Kshetri, "Can blockchain strengthen the internet of things?" *IT Professional*, vol. 19, no. 4, pp. 68–72, Aug. 2017.
[16] S.-C. Cha, J.-F. Chen, C. Su, and K.-H. Yeh, "A blockchain connected gateway for BLE-based devices in the Internet of Things," *IEEE Access*, vol. 6, pp. 24639–24649, 2018.
[17] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.
[18] A. Demers *et al.*, "Epidemic algorithms for replicated database maintenance," *ACM SIGOPS Oper. Syst. Rev.*, vol. 22, pp. 8–32, 1988.
[19] L. Lozinski, "How ringpop from Uber engineering helps distribute your application," 2016. [Online]. Available: https://eng.uber.com/intro-to-ringpop/
[20] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, 2010.
[21] Docker, "Docker swarm mode overlay network security model," 2017. [Online]. Available: https://docs.docker.com/v17.09/engine/userguide/networking/overlay-security-model/
[22] A.-M. Kermarrec and M. Van Steen, "Gossiping in distributed systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 2–7, 2007.
[23] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, "Epidemic information dissemination in distributed systems," *Computer*, vol. 37, pp. 60–67, 2004.
[24] D. Gavidia, S. Voulgaris, and M. Van Steen, "A gossip-based distributed news service for wireless mesh networks," in *Proc. 3rd Annu. Conf. Wireless On-Demand Netw. Syst. Serv.*, 2006, pp. 59–67.
[25] P. Andrew and I. Antonios, "Asynchronous gossip-based data propagation protocol," in *Proc. 5th IEEE Int. Conf. Digit. Inf. Commun. Technol. Appl.*, 2015, pp. 7–12.
[26] A. Ahmad, "Integration of IoT devices via a blockchain-based decentralized application," Master's thesis, Dept. Inst. Architecture Appl. Syst., Univ. Stuttgart, Stuttgart, Germany, 2017.
[27] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with BFT-SMART," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Atlanta, GA, USA, Jun. 23–26, 2014, pp. 355–362.
[28] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. Operating Systems Design and Implementation*, 1999, vol. 99, pp. 173–186.
[29] J. Sousa and A. Bessani, "From Byzantine consensus to BFT state machine replication: A latency-optimal transformation," in *Proc. 9th IEEE Eur. Dependable Comput. Conf.*, 2012, pp. 37–48.
[30] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, 2007, Art. no. 8.
[31] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," Cryptology ePrint Archive, Report 2018/483, 2018. [Online]. Available: https://eprint.iacr.org/2018/483

Songlin He (S'18) is currently working toward the Ph.D. degree in computer science with the New Jersey Institute of Technology, Newark, NJ, USA.

His main research interests include blockchain, Internet of Things, and distributed computing.

Qiang Tang (M'16) received the M.S. degree in information security from the Graduate University of Chinese Academy of Sciences, Beijing, China, in 2009, and the Ph.D. degree in computer science from the University of Connecticut, Storrs, CT, USA, in 2015.

He was with Cornell University, Ithaca, NY, USA, as a Postdoctoral Research Associate and was also affiliated with the Initiative of CryptoCurrency and Contract. He is currently an Assistant Professor of Computer Science with the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, and also the Director of JingDong – New Jersey Institute of Technology – Institute of Software, Chinese Academy of Sciences (JD-NJIT-ISCAS) Joint Blockchain Laboratory. His research is supported by National Science Foundation (NSF), Department of Energy (DOE), Air Force Research Laboratory (AFRL), Google, JD.com, and multiple blockchain foundations. His current research interests include applied and theoretical cryptography and blockchain technology.

Prof. Q. Tang was a recipient of Google Faculty Research Award, in 2019.

Chase Qishi Wu (SM'19) received the Ph.D. degree in computer science from Louisiana State University, Baton Rouge, LA, USA, in 2003. He completed the Ph.D. dissertation with Oak Ridge National Laboratory, Oak Ridge, TN, USA.

He was a Research Fellow with Oak Ridge National Laboratory during 2003–2006 and an Associate Professor with the University of Memphis, Memphis, TN, USA, during 2006–2015. He is currently a Professor of Computer Science and the Director of the Center for Big Data, New Jersey Institute of Technology, Newark, NJ, USA. His research interests include big data, parallel and distributed computing, high-performance networking, sensor networks, and cybersecurity.

Xuewen Shen received the M.S. degree in applied mathematics from Hangzhou Normal University, Hangzhou, China, in 2003.

She is currently an Associate Professor with the College of Media Engineering, Communication University of Zhejiang (CUZ), Hangzhou, China. She was the Faculty Member with CUZ and was promoted to the rank of Associate Professor in 2008. She was a Visiting Scholar with the New Jersey Institute of Technology, Newark, NJ, USA, in 2019. She is currently a Faculty Member with Media Technology Research Institute, CUZ. Her research interests include applied mathematics, mathematical education, and media big data.