Future Generation Computer Systems 92 (2019) 357–373

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

MOF-BC: A memory optimized and flexible blockchain for large scale networks

Ali Dorri [a,b,*], Salil S. Kanhere [a], Raja Jurdak [b]

[a] The School of computer science and engineering, UNSW, Sydney, Australia
[b] DATA61 CSIRO, Brisbane, Australia

HIGHLIGHTS

- Removable blockchain compatible with all existing blockchain instantiations.
- User to exercise the right to be forgotten while maintaining blockchain consistency.
- Reduces blockchain storage requirements and management costs.
- Maintains a level of auditability even if transactions are removed.

ARTICLE INFO

Article history:
Received 29 December 2017
Received in revised form 23 August 2018
Accepted 2 October 2018
Available online 17 October 2018

Keywords:
Blockchain
Auditing
Privacy
Internet of Things

ABSTRACT

The inherent immutability offered by Blockchains (BC) ensures resilience against modification or removal of the stored data. In large scale networks like the Internet of Things (IoT), however, this feature significantly increases BC storage size and raises privacy concerns. In this paper, we propose a Memory Optimized and Flexible BC (MOF-BC) that enables the IoT users and service providers to remove or summarize their transactions and age their data and thus exercise their "right to be forgotten". To increase privacy, a user may employ multiple keys for different transactions. However, to facilitate removal of the stored transactions in the future, all keys would need to be stored which complicates key management and storage. MOF-BC introduces the notion of a Generator Verifier (GV) which is a signed hash of a Generator Verifier Secret (GVS). The GV changes for each transaction to provide privacy yet is signed by a unique key, thus minimizing the information that needs to be stored. A flexible transaction fee model and a reward mechanism is proposed to incentivize users to participate in optimizing memory consumption. We propose MOF-BC as a generalized solution, which can be implemented on top of any existing or future BC instantiation. Qualitative security and privacy analysis demonstrates that MOF-BC is resilient against several security attacks. Evaluation results show that MOF-BC decreases BC memory consumption by up to 25% and the cost incurred by users by more than two orders of magnitude compared to conventional BC instantiations.

1. Introduction

BlockChain (BC) is a disruptive technology which has attracted tremendous attention from practitioners and academics in different disciplines (including law, finance, and computer science) due to its salient features which include decentralization, anonymity, security, and immutability [1]. In BC, a transaction forms the basic communication primitive that allows two nodes to exchange information with each other. A digital ledger of transactions is shared and synchronized across all participating nodes. Using a consensus algorithm which involves solving a resource demanding hard-to-solve and easy-to-verify puzzle, a secure trusted network is established among untrusted nodes. BC users may choose to employ changeable Public Keys (PK^+) as their identity which prevents them from being tracked, and thus provides a level of anonymity [2]. Multiple transactions are collated to form a block which is appended to the ledger by following the consensus algorithm. Each block includes the hash of the previous block in the ledger. Any modifications to a block (and thus transactions) can be readily detected as the hash maintained in the subsequent block will not match. The immutable nature of the BC affords auditability of all stored transactions.

BC was first introduced in Bitcoin [3], the first cryptocurrency system. Since then, it has been widely applied to non-monetary applications, e.g. securing and distributing sharing economy for

* Corresponding author at: The School of computer science and engineering, UNSW, Sydney, Australia.
E-mail addresses: ali.dorri@unsw.edu.au (A. Dorri), Salil.kanhere@unsw.edu.au (S.S. Kanhere), Raja.Jurdak@csiro.au (R. Jurdak).

https://doi.org/10.1016/j.future.2018.10.002
0167-739X/

renting properties [4] and securing vehicle to vehicle communications [5]. Recently, there has been increased interest in adopting BC to address security and anonymity challenges of large scale networks including the billions of connected devices that form the Internet of Things [6].

1.1. Motivation

Although BC offers a secure, anonymous, and auditable framework for IoT, the immutable nature of the BC raises significant challenges. First, the large number of IoT devices will undoubtedly generate an equally substantial number of transactions which in turn would significantly increase the memory footprint at the participating nodes that store the BC. To give some context, the current Bitcoin BC which comprises 324 million transactions requires 172GB of storage space [7]. Arguably, an IoT BC would rapidly outgrow the Bitcoin BC. The nodes that store transactions in the BC, known as miners, are offered a monetary reward which is paid by the transaction generators in the form of a transaction fee [3]. In IoT, the costs associated with a perpetually increasing BC would also be prohibitive for the users. Second, IoT applications are likely to have diverse storage requirements. For example, a smart device may provide its data to a Service Provider (SP) for a fixed period (e.g., a one year subscription to a service) and thus the associated transaction record may only be needed for this duration. As another example, a user may progressively install multiple IoT devices at a facility and may wish to summarize all transactions associated with these devices into a single consolidated transaction representing the entire facility. Such flexibility is not afforded by current BC instantiations wherein transactions are stored permanently and cannot be altered. Third, permanently storing transactions of all devices of a user in the public BC could compromise the user privacy by: (i) linking multiple transactions generated by the same user, which eventually allows deanonymization of their identity (referred to as linking attack [8]), or (ii) monitoring the frequency with which a user stores transaction even if the transaction content is encrypted which reveals information about the interactions between the IoT devices and/or SPs [9]. Consequently, privacy concerns may motivate users to exercise their right to be forgotten and not store records of certain IoT devices in the BC.

The right to be forgotten is one of the key requirements in General Data Protection Regulation (GDPR) of the European Union (EU) which is discussed primarily in article 17 [10]. This regulation came into action in May 2018. However, as outlined earlier, conventional BC instantiations do not address this requirement which raises questions about their widespread adoption given that all enterprises that offer services to EU citizens are now required to comply with these regulations [11–13]. Thus, there is a strong need for a way by which data stored in the BC can be removed when requested by a user.

1.2. Contributions

In this paper, we propose a Memory Optimized and Flexible BC (MOF-BC) that affords greater flexibility in storage of transactions and data of IoT devices. The aforementioned examples illustrate instances where the user must have full control over the management of the stored transactions in the BC. One can also envisage use cases where the SP needs to exert this control. For example, the power company may install sensors and smart metering equipment on the client's premises but would still wish to exert control over these devices. We introduce User-Initiated Memory Optimization (UIMO) and SP-Initiated Memory Optimization (SIMO) which allow either the user or the SP where appropriate to remove or summarize stored transactions and to age (compress) the data of an IoT device which may be either stored within the BC or off-the-chain in the cloud. However, the SP or user may not always need to be burdened with management of stored transactions for all their devices. MOF-BC thus provides a way to offload these functions to the BC network by introducing the notion of Network-Initiated Memory Optimization (NIMO). This is realized by specifying the Memory Optimization Modes (MOM) in the transaction when it is created. MOM can be of the following types: (i) temporary: the network removes a transaction after a specific period of time which conceptually is similar to removing a transaction in UIMO and SIMO, (ii) permanent: transaction is stored permanently, and (iii) summarizable: the network summarizes multiple transactions to a single summary record in the same way as the user or the SP summarizes transactions in UIMO or SIMO.

The removal of stored transactions is fundamental to implementation of the aforementioned MOM. However, the immutable nature of conventional BCs does not permit this operation since the hash of a block is computed over the contents of all transactions within the block. MOF-BC addresses this challenge by computing the hash of the block over the hashes of constituted transactions and not their contents. This allows a transaction to be removed from a block without impacting the hash consistency checks. Moreover, the existence of the hash of a transaction that is no longer present in the BC allows for limited posthumous auditability. The transaction generator must store the transaction content locally and reveal it to the participant(s) that wishes to audit the removed transaction. Using the stored hash in the BC, other participants can ensure that the transaction once existed in the BC. However, auditing is not possible if the transaction generator does not store its content locally.

A previously stored transaction can be removed only by the transaction generator, i.e., the node that knows the corresponding PK^+/Private Key (PK^-), to prevent another node from either maliciously or erroneously removing its transactions. As noted earlier, a node (user or SP) may choose to change the PK^+ used in its transactions to enhance its privacy. The node will thus have to store a large number of keys to be able to remove stored transactions at a later point in time, which complicates key management and storage. The authors in [14] proposed a Hierarchical Deterministic (HD) wallet which organizes the keys hierarchically such that child PK^+/PK^- pairs are generated using m parent PK^+ and PK^- that belong to the higher tier in the hierarchy, where m is the number of keys defined based on the specific use case. The child keys are used for signing the transactions by the user. For managing transactions generated by all child keys in the hierarchy, only the m parent PK^+/PK^- pairs need to be stored. This incurs significantly lower storage than having to store and manage each and every key used for creating transactions. By knowing master PK^+s, the child PK^+s can also be generated. However, this may also allow a malicious participant to link all the transactions generated by a user and thus compromise the user anonymity. To address the key management challenge, MOF-BC adds a *Generator Verifier (GV)* in each transaction which conceptually shares some similarities with HD. GV is a signed hash of a *Generator Verifier Secret (GVS)* that is a secret that is known only to the entity generating the transaction. It can be a string similar to a password or an image or even biometric information such as a fingerprint scan. The GVS changes for each transaction by applying a simple pattern known only to this entity, e.g. incrementing the GVS by a fixed value, which thus makes the corresponding GV unique. This affords the same level of privacy as with a changeable PK^+. There is no dependency between the value of GV in multiple transactions of the same user, which ensures that even if the GVS is revealed then the attacker cannot deanonymize the user. Moreover, the GVs in all transactions generated by a node are encrypted using a single PK^+. Thus, to verify that a node can remove a stored transaction, it only has to furnish the PK^+ and

the GVS (and the corresponding GV) for that transaction. As is evident, secure storage of one PK^+/PK^- pair and the GVS requires less storage space as compared to m key pairs in the HD wallet.

The removal of a transaction requires each participating node to locate this transaction in the BC, which can in the worst case incur a delay of O(N) where N denotes the number of transactions in the BC. To amortize this overhead, rather than removing transactions on an individual basis, MOF-BC processes transaction removals in batches over a periodic Cleaning Period (CP). To facilitate the removal process, multiple partially distributed agents are introduced which reduce the packet and processing overhead associated with the multiple MOMs available in MOF-BC. These agents are defined as a new layer on top of the BC layer, thus, the core BC functions, including transactions and block generation and verification as well as distributed trustless network, are not impacted by MOF-BC and remain purely distributed. MOF-BC is independent of the BC layer, thus it can be employed on top of any blockchain instantiation. To encourage users to free up space, MOF-BC grants rewards to users that do so and introduces a flexible transaction fee based on the duration for which a transaction is stored. The rewards can be either used to pay the storage fee of new transactions or exchanged to Bitcoin. The increased flexibility and savings in storage have an associated cost of reduced accountability as the transaction content is either removed or consolidated which leads to loss of some information. Multiple benefits and implications of MOF-BC are studied in this paper.

To analyze the security of MOF-BC, we consider seven of the most relevant attacks and outline the defence mechanisms employed to prevent them. We develop a custom implementation of MOF-BC and show that it decreases the BC memory footprint by up to 25% and the cost that the users are required to pay to store their transactions by more than two orders of magnitude compared to conventional BC instantiations. We also provide comprehensive evaluations on the effects of the CP on the BC size.

1.3. Paper overview

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 outlines the details of employed memory optimization methods. Section 4 discusses the removing process. Section 5 outlines multiple real world applications for different optimization modes introduced by MOF-BC. Section 6 presents evaluation results. Section 7 concludes the paper and discusses the future works.

2. Related work

This section presents a brief overview of BC and outlines related work. Fig. 1 represents the basic structure of a transaction in BC. Note that different instantiations of BC might have some slight variations in the transaction structure. T_{ID} represents the unique identifier of the transaction which is the hash of all other fields of the transaction. $P.T_{ID}$ denotes the ID of the previous transaction which effectively establishes the link between successive transactions created by the same node (or entity), thus forming a ledger. The first transaction in each ledger is known as the genesis transaction. It is possible that there exist dependences between transactions whereby certain fields generated in one transaction (outputs) are referenced as inputs in another transaction. The inputs and outputs are stored in the *Input* and *Output* fields. In most BC instantiations, a node changes the public key (PK^+) used as the identity in each transaction that it creates as a way to increase anonymity. The hash of this PK^+ is stored in the PK field. The reason for storing the hash of the PK^+ is to reduce the size of the transactions as well as to future proof transactions against possible attacks where malicious nodes may attempt to reconstruct the

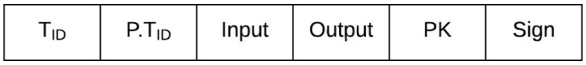

Fig. 1. The structure of a transaction.

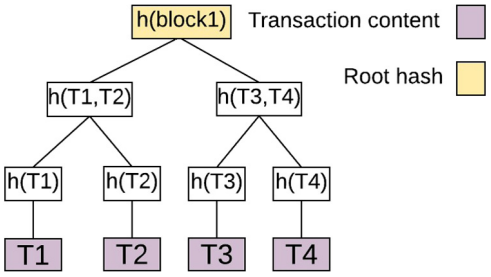

Fig. 2. The structure of the Merkle tree.

Private Key (PK^-) using the PK^+. Finally, the *Sign* field contains the signature of the transaction generator, created using the PK^- corresponding to the PK^+.

All transactions are broadcast to the network. Special nodes, known as miners, verify each transaction by validating the embedded signature using the corresponding PK^+. Next, the existence of the $P.T_{ID}$ is checked. Finally, other fields in the transaction are verified depending on the specific rules of the BC instantiation. The verified transactions are added to a pool of pending transactions. Each miner collates pending transactions into a block when the size of the collected transactions reaches to a predefined size known as the *block size*. The miner generates a Merkle tree [15] by recursively hashing the constituted transactions of the block, which are stored as the leaves of the tree, as shown in Fig. 2. The root of the Merkle tree is stored in the block header to speed up the process of verifying membership of a transaction in a block.

The miner mines, i.e., appends, the block into the BC by following a consensus algorithm, examples of which include Proof of Work (POW) [3] and Proof of Stake (POS) [16]. The consensus algorithm ensures BC consistency between participating nodes as well as randomness among the miners. The randomness prevents malicious miners from continuously mining blocks, thus increasing BC security. A mined block is broadcast to all nodes. Each node appends the new block to its local copy of the BC after validating the constituent transactions.

In recent years, there has been a growing interest in the adoption of BC technology in IoT. IBM introduced a new BC instantiation for IoT known as Hyperledger [17], which is a permissioned BC, wherein only authorized nodes can participate in BC. The authors in [18] proposed a new method to manage IoT devices using Ethereum smart contracts. Each device is managed by a contract in Ethereum. In [19], the authors proposed a BC-based Software Defined Network (SDN) architecture for IoT. A rule table which defines the rules for access permissions in the network is stored in the BC. Participating nodes can validate access requests using the BC. The authors in [20] proposed a new ledger based cryptocurrency called IoTA. By eliminating the notion of blocks and mining, IoTA ensures that the transactions are free and verification is fast. The key innovation behind IoTA is the "tangle", which is essentially a Directed Acyclic Graph (DAG). Before a node can generate a transaction, it has to verify two randomly chosen transactions generated by other nodes. Thus, IoTA throughput, i.e., the number of transactions stored in the ledger, increases as the number of participants increases.

In [21], the authors proposed a distributed secure and private IoT data management platform using BC. The data of IoT devices are stored off-the-chain, i.e., in a separate cloud storage, while the hash

of the data is stored in the BC. The access permissions for the data is stored in the BC. The authors in [22] proposed a BC-based multi-tier architecture to share data from IoT devices with organizations and people. The proposed architecture has three main components namely: data management protocol, data store system, and message service. The data management protocol provides a framework for data owner, requester, or data source to communicate with each other. The messaging system is used to increase the network scalability based on a publish/subscribe model. Finally, the data store uses a BC for storing data privately.

Despite the benefits of the BC in IoT in the above mentioned works, the BC memory footprint still remains an unsolved issue considering the large scale of IoT. As new services are introduced in IoT by SPs and with the passage of time, the number of transactions generated by the large number of connected devices that make up an IoT network will significantly increase. Consequently, the BC memory footprint will increase infinitely. The authors in [21,23] proposed to store the older blocks of the BC off-the-chain to address the BC storage challenge. Although this method reduces the resource requirement on the participating nodes, it incurs additional latency for retrieving transactions from the cloud. Moreover, the issue of ever expanding storage is simply offloaded to the cloud. Additionally, as the user data still is permanently accessible by all participants, the aforementioned privacy concerns still remain. MOF-BC tackles these challenges by affording the users flexibility to remove older transactions or consolidate multiple transactions as one and age stored data in the cloud or transactions.

As noted earlier, each user may employ multiple PK^+s as his identity that must be stored to be able to claim ownership of the generated transactions, which in turn complicates key management and storage. The authors in [14] proposed Hierarchical Deterministic (HD) wallet to address this challenge in Bitcoin. In their hierarchical approach child PK^-s (denoted as d_i) are generated using a parent PK^- (denoted as $\hat{d}$) as follow:

$$d_i = hash(i, \hat{d})$$

HD wallet is vulnerable to key leakage as knowledge of $\hat{d}$ would allow an attacker to reconstruct all d_is. To address this challenge, the authors proposed to use m parent keys in place of a single key, with the logic being that it would be difficult for someone to gain access to all m parent keys. However, their approach has the following drawbacks: (i) anyone who knows the parent PK^+s can reconstruct the subsequent child PK^+s. This reduces the anonymity offered by BC as anyone can link multiple transactions generated by a single user with different child PKs, (ii) the nodes must still store m keys locally and generate child keys using them which incurs processing and storage overhead, particularly for resource restricted devices in IoT. In this paper, we propose a GV which is a signed hash of a secret known to the transaction generator. The GV value in each transaction is independent of the other GV values which makes user de-anonymization challenging even when the two hashes used in one GV are revealed to the BC participants. The proposed GV introduces higher security as for generating a GV, the node requires not only the GVS, but also the pattern employed and the corresponding PK^+ which are both only known to the GV generator.

The authors in [24] were the first to propose a modifiable BC that allows users to modify or remove a stored transaction. They use a Chameleon hash function to generate the block hash, by hashing the block content, such that a collision can be found in the hash. The modification of BC is performed by one central node or a group of distributed nodes known as modifier(s). The modifiers are aware of a secret trapdoor key that is used for creating a collision in the hash of a block such that $H(m, \xi) = H(m', \xi')$ where $H(x)$ indicates the hash of x, m is the block content, and ξ is a *check string*. The check string is adjusted by the modifier to find a collision in the block hash.

This method faces multiple challenges in an IoT setting. To modify the content (i.e., transactions) of a chain of blocks, the chain is sent to the modifier(s). The modifier(s) broadcast the modified chain to all nodes. Each node verifies all new blocks and replaces the corresponding blocks in its copy of the BC with them. However, this approach is unlikely to scale for large IoT eco-systems, where one can envision a large number of modification requests, due to significant (processing and packet) overheads associated with the aforementioned steps. Moreover, their approach does not keep the consistency of the transactions before and after modification as the hash of the transaction after modification would not match with the stored hash in the transaction header. Finally, there is not any provision to remove information from the BC. Conversely, in MOF-BC only the transaction generator can modify its transaction. A range of memory optimizations including removal and summarization of transactions and aging of data are afforded and there is flexibility for initiating these optimizations either by the user or the SP. In addition, the responsibility of these optimizations can be offloaded to the network by generating transactions with specific optimization modes. MOF-BC introduces multiple agents and a shared indexing service that significantly reduces the associated overheads. We also propose to use a central index database to manage the removal process and introduce multiple rewards to incentivize users to free up space in BC.

3. Optimizing blockchain memory consumption

In this section, we discuss memory optimization methods employed by MOF-BC. We first provide an overview of the MOF-BC framework in Section 3.1. In Section 3.2, we introduce the notion of a storage fee as part of the transaction fee to reward the nodes that are involved in storing the BC for the amount of storage space contributed. Finally, in Section 3.3, we outline how the underlying mechanisms that form the basis of user-initiated, SP-initiated, and network-initiated memory optimizations are implemented.

3.1. An overview of the framework

Fig. 3 proposes an overview of the MOF-BC framework which consists of three layers: (i) peer-to-peer (p2p) layer: this layer is the underlying p2p network constructed by all participating nodes in the BC, (ii) BC layer: this layer is the core BC layer in which all BC functions including transaction and block generation and verification are performed. Any existing or future BC instantiation can be employed in this layer, (iii) MOF-BC layer: this layer comprises multiple agents for achieving efficient execution of several key functions of MOF-BC. The agents are partially distributed meaning that multiple replications of each agent are placed in the network. Synchronization methods such as in [25] are used to synchronize the replicas. They use checksums to ensure consistency among the replicas. Following the update of an agent, a checksum is computed and validated by comparing with the checksum of other updated agents. Using partially distributed agents prevents the agents from being a bottleneck and reduces the (processing and packet) overheads in the network compared to a fully distributed approach as in [24].

In MOF-BC layer, a Summary Manager Agent (SMA) manages all the processes related to summarization of multiple transactions into one consolidated transaction. A Reward Manager Agent (RMA) computes the rewards offered to the nodes that participate in memory optimization and free up memory. The rewards are sent to a Bank to be claimed by the user. Storage Manger Agent (StMA) collects the storage fees and distributes the collated amount proportionally among the miners. A Patrol Agent (PA) monitors the

claims made by the miners by randomly migrating to a miner and checking the storage resources expended in storing the BC. Blackboard Manager Agent (BMA) manages a central read-only database that acts as a repository of information required for bookkeeping e.g., a list of miners, and a list of transactions paid by rewards. A Service Agent (SerA) is responsible for processing transaction removals. It maintains an updated version of the BC during the removal process, which is passed on to the miners at the conclusion. A Search Agent (SA) searches newly mined blocks for particular, e.g., summarizable transactions and sends them to relevant agents such as the SMA and RMA. The aforementioned agents may need to generate transactions to action memory optimization, e.g. the SMA generates a summary transaction in place of multiple summarized transactions.

Each agent is identified by a unique PK^+ which is certified by a Certificate Authority (CA) to verify the identity of a particular agent. Note that we rely on a centralized approach (i.e., existing public key infrastructure) for this aspect of identity verification in MOF-BC layer. The rest of the functionality is achieved by the distributed BC in BC layer.

To enhance the network security against malicious agents and reduce the overhead of verifying the transactions generated by the agents, the miners employ a distributed trust algorithm as proposed in [26]. The core idea behind the distributed trust algorithm is that the stronger the evidence a node has gathered about the validity of transactions generated by an agent, fewer of the subsequent transactions received from that agent need to be verified. The miners use a distributed trust table, an example is shown in Fig. 4, which specifies the probability with which a transaction from each agent must be validated (known as Validation Probability (VP)). If the transaction is checked and is valid, then the number of validated transactions for that particular agent is incremented (i.e., this agent is now more trustworthy), thus, the VP for the next transaction from this agent will decrease. If the agent generates an invalid transaction, the number of validated transactions for that agent reduces. If the malicious agent continues with this behavior, the VP for its transactions will be increased and more transactions will be validated. The VP must never be equal to zero even for highly trusted nodes to protect the network against potentially compromised agents. A compromised agent may start adding false transactions in new blocks that it generates, which is known as appending attack. The minimum value for VP to protect against this attack depends on the total number of the miners, i.e., the transaction validators. We studied the minimum value of VP that protects network against appending attack in [8]. We simulated a BC network with 50 participating nodes using NS3 simulator. Participating nodes generate a combination of four transactions per second which are formed into blocks with block size (i.e., the total number of transactions in a block) of 10 transactions by the miners. We vary the number of miners in the network from 3 to 20. We assume that a miner with the highest trust value, i.e., lowest VP, is compromised and conducts an appending attack. We repeat the simulation 10 times and choose the minimum VP for which the appending attack is detected by at least one of the miners in all 10 runs. Simulation results are outlined in Table 1 and can be used as a guideline to configure the trust table. As can be seen from the results, the greater the number of miners in the network, the lower the minimum value of VP to prevent an appending attack. VP values smaller than the values proposed in Table 1 will make the network vulnerable to appending attacks.

Note that these results are not absolute numbers to be used for the trust table. The main aim of this study is to give an intuition on how to design the trust table based on the network configuration.

Table 1
Minimum VP for detecting appending attacks as a function of the number of miners.

Number of miners	3	5	7	10	13	15	17	20
Minimum VP	80	60	60	40	20	20	20	10

3.2. Storage fee

Since the primary goal of MOF-BC is optimizing BC memory storage, it is important to consider the associated costs and benefits of transaction mining and storage in BC. In conventional BCs, the miner transacts a fee for all transactions within a block as compensation for the resources consumed to mine the block. This fee varies across BC instantiations. However, a storage fee is not considered. There is now growing consensus that a fee must be assessed for storage, particularly for large networks like IoT [27]. Billions of devices in IoT will generate a large number of transactions that must be permanently stored in the BC and thus requires the miners to expend significant memory resources for storing the BC. To incentivize the miners for contributing memory space for the BC storage, MOF-BC considers a storage fee in addition to the mining fee. Unlike mining fee which is paid to the miner as an incentive for mining the transaction, storage fee is paid to all the nodes who have stored the BC according to the duration and amount of storage space they contributed to the BC. Thus, the storage fee is first paid by the transaction generators to a Storage Manager Agent (StMA) which pays the nodes that stored the BC according to their contribution.

The storage fee is levied based on the size of the transaction and the duration for which it is stored and is discussed in Section 3.3. In MOF-BC the minimum size of a transaction is defined as a *page*. For a string x we denote its size by $|x|$. At the minimum, a transaction must include the following fields: PK^+, Signature ($Sign$), hash of the current transaction (T_{ID}), and hash of the previous transaction ($P.T_{ID}$). Thus, $|page| = |PK^+| + |Signature| + 2|hash|$. It is assumed that the BC designer sets a pre-defined value for $|page|$. The total number of pages required for a transaction is determined by rounding up to the nearest integer number of pages that can contain the transaction contents.

The payments to the miners are made out in periodic time intervals known as *payment periods*. When miner X joins the BC network, it notifies the StMA of the amount of dedicated storage space that it is contributing to store the BC, known as $Storage_X$. The miners benefit from larger BC and thus may avoid removing transactions from their BC copy. To protect against this, the size of the most optimized version of the BC which is available by SerA is considered to be the maximum size of the BC a miner can claim, i.e.,

$$Storage_X <= Storage_{SerA}$$

Similarly, when miner X leaves the BC network it notifies the StMA. Thus, the StMA pays the miner only for the duration that it has the BC stored as discussed below. The miners may store the comprehensive BC or a specific part of the BC based on their available resources and requirements [6]. At the end of each payment period, the StMA calculates the cumulative amount of space allocated to the BC in the network ($Store_{All}$). The share of each miner of the collected storage fee ($Share_X$) is:

$$Share_X = Storage_X * \frac{Fee}{Store_{All}} * \frac{Time_X}{PaymentPeriod}$$

Where Fee is the cumulative storage fees received during the current payment period and $Time_X$ denotes the duration with the current payment period for which miner X has stored the BC. The StMA may maliciously prevent paying the storage fee to (some of)

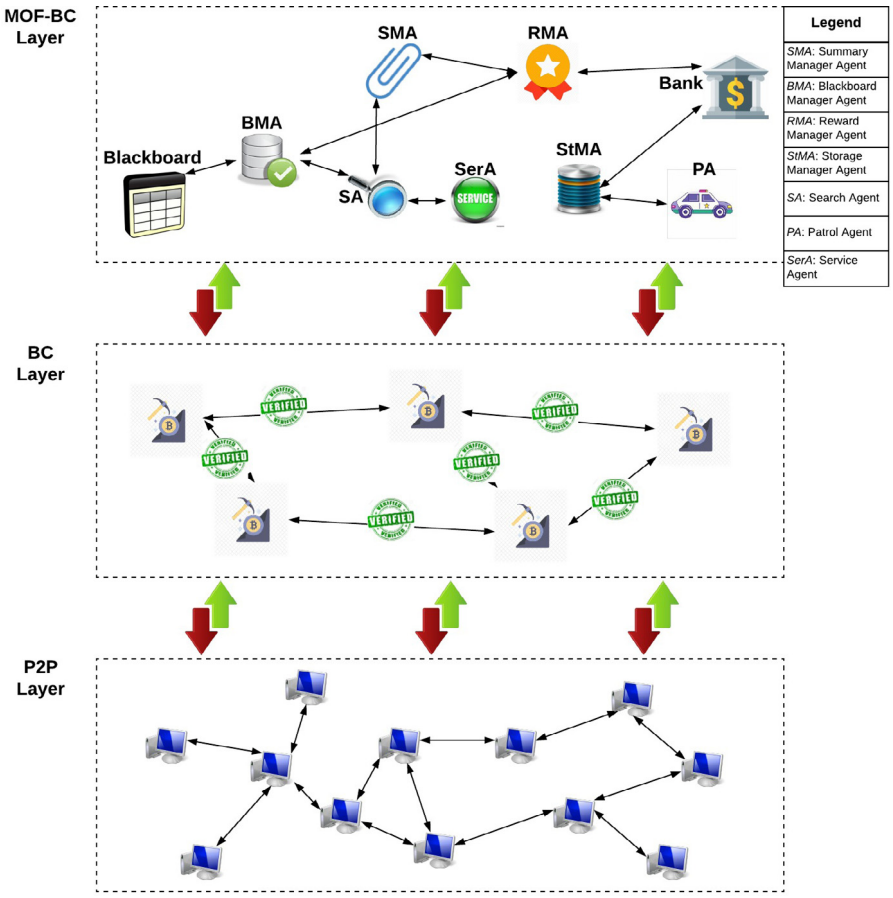

Fig. 3. An overview of the MOF-BC.

the miners. The malicious StMA can be detected by the miners as they would not receive any payment from the StMA. Consequently, the miners isolate the StMA and choose a new one for the network. All payment claims made by the miners are verified by a Patrol Agent (PA). The PA is a mobile agent meaning that it can migrate from one miner to another miner. The PA migrates randomly between miners and examines the memory footprint used at a miner for storing the BC. The PA informs the StMA if a discrepancy is observed in the claims made by a particular miner. No further payment claims are accepted from this malicious miner and subsequently no rewards are paid out.

A PA may be compromised. A malicious PA does not report the false claims made by the miners so that the miners will receive payment while they no longer have the BC stored. This can be detected by the new nodes joining the BC network. The new nodes request to download the BC from one of the miners. If the miner rejects their request, they inform the StMA. The StMA then generates a new PA to check the validity of the suspected miner as the old PA is suspected to be compromised. If the new PA detects a malicious behavior of the miner, then the old PA is removed and the miner is also isolated.

3.3. Memory optimization

In this section, we discuss multiple optimization methods employed by the MOF-BC to optimize the BC memory footprint. As was noted in the examples and arguments in Section 1, either the user or the SP must have absolute control over all stored transactions pertaining to their devices (depending on the use case). To support this, MOF-BC introduces User-Initiated Memory Optimization (UIMO) and SP-Initiated Memory Optimization (SIMO).

While the aforementioned optimizations offer immense flexibility, actioning them requires the user (in UIMO) or SP (in SIMO) to explicitly create new transactions. For example, to remove a particular stored transaction, the user (or SP) would have to initiated a new remove transaction (details to follow in Section 3.3.1). These entities may not wish to be encumbered with these actions for all transactions of each and every device under their control. An option to delegate these actions to the network is thus available via Network-Initiated Memory Optimization (NIMO). This is achieved by specifying the appropriate Memory Optimization Modes (MOM) in the transaction when it is created.

To be able to optimize the BC memory footprint, we introduce the following additional fields to the transaction format:

$$GV \parallel MOM \parallel MOM - Setup \parallel Pay - by - reward$$

GV is used by the UIMO and SIMO as discussed in Sections 3.3.1 and 3.3.2. MOM and MOM-Setup fields are used to identify the MOM and its configuration used for the NIMO as discussed in 3.3.3. The last field is used to pay the storage fee by accrued rewards as per discussions in 3.3.1.

Table 2 summarizes different optimization methods which are outlined in greater detail in the rest of this section.

3.3.1. User-Initiated memory optimization (UIMO)

Before we describe UIMO, we first discuss a new concept introduced for enabling this functionality known as *Generator Verifier (GV)*. When a user wishes to optimize a stored transaction, he must first present evidence that the transaction was created by him. The PK^+ used to generate the transaction is sufficient to prove ownership. However in the IoT context, users may choose to change the PK^+ used for different transactions to protect their privacy (as

Table 2
Summary of the different optimization options.

	Who authorizes the optimization?	Who initiates the optimization?	When is the optimization initiated?	Optimization methods
UIMO	User	User	The user can initiate this at a time of his choosing by creating a transaction.	Remove, summarize, Age (data).
SIMO	SP	SP	The SP can initiate this at a time of his choosing by creating a transaction.	Remove, Summarize, Age (data).
NIMO	User/SP	Network	The User (or SP) must set the appropriate MOM in the transaction when it is created. The network will perform the action as per the specified rules for the MOM.	Temporary, Permanent, Summarize.

Number of previously validated transactions	10	20	30	40	50
VP for next transaction	80%	60%	40%	30%	20%

Fig. 4. An example of a trust table in participating nodes.

T_{ID}	$P.T_{ID}$	Input	GV-hash1	GV-hash2	GV-PK$^+$	Sign

Fig. 5. The structure of remove transaction.

discussed earlier in Section 1). Management and storage of a potentially large number of keys is bound to become an issue. To address this challenge, we introduce the notion of a Generator Verifier (GV). The GV in all transactions generated by a user is encrypted using a unique PK$^+$ (known as GV-PK$^+$), thus eliminating the need for managing multiple keys. For transaction *i*, GV_i is calculated as the signed hash of the $P.T.ID_i$ and a *Generator Verifier Secret (GVS)*. The GVS is a secret known only to the user and can be any type of data, e.g., a string similar to a password or an image or biometric information such as fingerprint scan. To prevent an attacker from guessing the GVS, the same recommendations for creating strong passwords apply. The GV value for each transaction is unique even if the same GVS is used in multiple transactions as the P.T.ID is unique in each transaction. To further enhance the security, a user has the option to change the GVS for each transaction by applying a fixed pattern to it. For example, adding a fixed value to it. In this instance, the user would need to remember this unique pattern along with the first used value of the GVS. Once a transaction with GV is stored in the BC, the user can perform the following memory optimizations at a later time:

(1) Removal: The user can remove his stored transactions, to either optimize the BC memory or enhance his privacy, by generating a *remove transaction*. Fig. 5 illustrates the format of this transaction. T_{ID} is the ID of the transaction, and $P.T_{ID}$ is the ID of the previous transaction in the ledger. The ID of the transaction to be removed is stored in the *input* field of the remove transaction. To remove a stored transaction, the user has to prove that he has previously generated that transaction. To do so, the user must include the hashes used to generate the GV, i.e., the GVS and the P.T.ID, of the transaction to be removed and GV-PK$^+$ in *GV-hash1*, *GV-hash2*, and *GV-PK$^+$*, respectively . To prove that the user knows the PK$^-$ corresponding to the GV-PK$^+$, the user signs the remove transaction with this key and stores it in *sign* field. This transaction is subsequently broadcast to the network.

On receipt of the remove transaction (say X), a miner first locates the transaction that is marked for removal in the input field of X (say Y) in the BC. Then, the miner verifies X by verifying that the generator of X knows the hashes and the key used for creating GV stored in Y (see Section 3.3.1) as outlined in Algorithm 1. Recall that GV in Y is a signed hash, i.e., is in ciphertext, thus the miner first decrypts the GV using the provided GV-PK$^+$ in X (lines 1,2 Algorithm 1). Next, the miner verifies if the hash of the GVS and P.T.ID, extracted from GV-hash1 and GV-hash2 fields in X matches with the GV in Y (lines 4,5). Finally, the miner verifies the signature in X using the GV-PK$^+$ (lines 7,8). This ensures that the generator of X knows the corresponding GV-PK$^-$. The verified transaction is mined into the BC.

ALGORITHM 1: The process of verifying GV.

Input: remove transaction (X), transaction to be removed (Y)
Output: True or False
Verification :
1: **if** (X.GV-PK$^+$ *not decrypt* Y.GV) **then**
2: **return** False;
3: **else**
4: **if** H(X.GVS + X.P.T.ID) != Y.GV **then**
5: **return** False ;
6: **else**
7: **if** X.GV-PK$^+$ *redeem* X.Sign **then**
8: **return** True;
9: **else**
10: **return** False ;
11: **end if**
12: **end if**
13: **end if**

The removal of Y requires each miner to locate this transaction in the BC, which in the worst case incurs a delay of O(N) where N denotes the number of transactions in the BC. To amortize this overhead, the miners process transactions removal in batches over a periodic Cleaning Period (CP). A detailed discussion about batch removals is presented in Section 4.

To encourage users to optimize BC memory consumption, MOF-BC rewards users that reduce the BC memory footprint by removing their transactions. The reward allocation process is performed by the *Reward Manager Agent (RMA)*. The RMA would need to search each newly mined block to find a remove transaction. Other agents would need to similarly search for new transactions of a particular kind (e.g. the SMA would need to search for summarization transactions). To amortize these overheads, a dedicated Search Agent (SA) is designated to search newly mined blocks for transactions related to memory optimization and send references for these to the relevant agents. The agents may randomly check whether the SA sends all relevant transactions to them by searching newly mined blocks and compare the relative transactions with the ones sent by the SA. For removed transaction *Y*, the reward value is calculated as:

$$Reward = Y.pages - X.pages$$

Where *X* is the remove transaction. Each transaction should be only rewarded once, even if it is mined in multiple blocks by multiple miners. To ensure this, the RMA records the GV of

Time-Stamp	PK+	Sign	Merkle Tree Root	Inputs	Outputs	Summary-Time	TransOrder	$H_{X.1}$	$H_{X.2}$	GV-PK+

Fig. 6. The structure of summary transaction.

transactions that have been rewarded. The RMA sends the GV and the corresponding amount of reward to a Bank which collects all information of the rewards. Sending only the reward and the GV to the bank ensures user anonymity. If a node does not receive its rewards, i.e., the RMA is compromised, then the node informs the rest of the network to change the RMA as per discussions in Section 6.1.

By rewarding the users for removing their transactions, MOF-BC optimizes the BC memory footprint. The miners, on the other hand, are paid for the space they allocate to the BC, implying that they may benefit from having a larger BC. This potentially may lead to a conflict of interest about the BC size between the miners and the participating nodes. To motivate miners to be efficient about memory usage, we limit the payment received by a miner to be proportional to the size of the optimized version of the BC, which is maintained by SerA. Additionally, storage fee is very small (see Section 3.3.3) which is calculated based on the cost for purchasing or maintaining the storage space necessary for BC. Larger BC memory footprint incurs higher cost for storage purchase or maintenance (see implementation results presented in Fig. 16). Thus, larger BC size does not necessarily mean greater benefit for the miners.

The user, i.e., the rewardee, can use his accrued rewards in two ways: (i) Exchange to Bitcoin: A user can ask the bank for his rewards to be converted to Bitcoin at the current exchange rate, (ii) Pay storage fee of new transactions: Recall that the transaction header contains a pay-by-reward field. If this field is set in a transaction, then it implies that the corresponding storage fee will be paid by earned rewards. The rewardee should send a corresponding redeem transaction to the bank which contains the ID of the new transaction and GV-PK+ and hashes used to construct the GV corresponding to the accrued reward. The bank verifies the redeem transaction as discussed in Algorithm 1. If verified, the bank sends the ID of the new transaction to the BMA which in turn notifies the miners and stores the ID of the new transaction in the blackboard.

(2) Summarize: As was noted in the examples and arguments in Section 1, IoT users can summarize their transactions into a single consolidated transaction to optimize BC memory while maintain the auditability of the summarized transactions. The process of summarizing transactions is shown in Fig. 7. To summarize a selected set of transactions, the user creates a *summary transaction*, which is illustrated in Fig. 6.

In Fig. 6, the *Time-stamp* is the time when the summary transaction is generated. The next two fields are the PK+ and signature of the transaction generator. The *Merkle tree root* is the root of the Merkle tree formed by collating all the transactions that are summarized in this consolidated transaction. Recall from Section 2 that this data structure makes it possible to perform posthumous audits, i.e., check whether a transaction belonged to the original group of transactions that are now summarized. If the transactions being summarized include multiple inputs and outputs, then including all of them in the summarized transaction would significantly increase its size. As a compromise, we chose to only include the inputs/outputs which are not used as outputs/inputs of a transaction in the same summarized group. Consequently, the excluded inputs and outputs can no longer be audited. Fig. 8 illustrates this process.

The time-stamp of each original transaction is stored in the seventh field, i.e., summary-time. The summary-times are stored in order. However, the Merkle tree does not provide any information

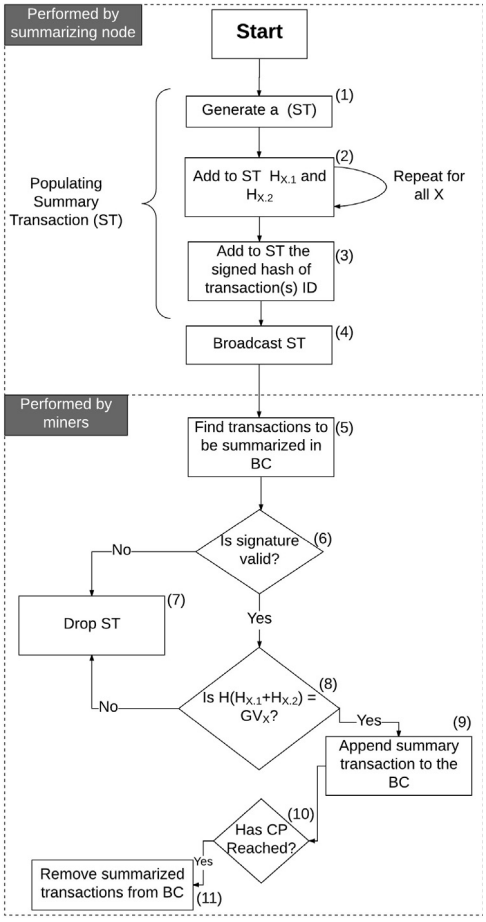

Fig. 7. The process of user-initiated summarization.

about the exact order of transactions. To address this challenge, the smallest number of distinct bytes of the *T.ID* of the summarized transactions (denoted as *d*) are stored in the *TransOrder* field. Thus, by knowing the *T.ID*, the specific order of a transaction within the pool of transactions can be found with small overhead. As an example consider the four transactions given in Fig. 9. The TransOrder field contains 3 bytes of the ID of each transaction as the first two bytes of IDs of transactions 1 and 3 are equal.

The value of d is always $1 <= d <= |T.ID - 1|$. Given the randomness in the hash outputs it is unlikely that the hashes for all transactions in the summarization pool have a large number of similar bytes which justifies using the distinct bytes to reduce the size of the summary transaction compared to storing the original T.ID. The last three fields in the summary transaction are used to verify the GV and are discussed in the rest of this section. In these fields X is the index of the transaction among transactions to be summarized.

The user initiates the summarization process by populating a summary transaction (step 1 in Fig. 7). To prove that it has previously generated the transactions that are to be summarized, the user stores the hashes used to generate the GV and the GV-PK+ for all transactions to be summarized in the summary transaction (step 2). The summarizing node must prove its identity by signing the hash of the ID of transactions to be summarized (step 3). Then, it broadcasts the summary transaction (step 4).

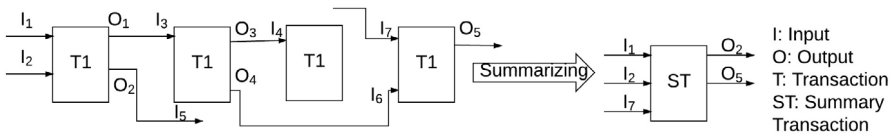

Fig. 8. The input/output of the summary transaction.

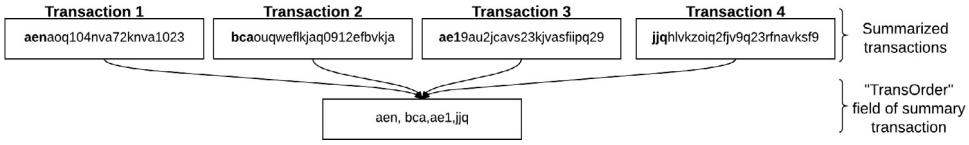

Fig. 9. TransOrder field population.

Time-Stamp	PK+	Sign	Merkle Tree Root	Inputs	Outputs	Summary-Time	TransOrder

Fig. 10. The structure of NIMO summary transaction.

On receiving the transaction, a miner must first verify if the summary transaction generator has the authority to summarize the listed transactions. This is achieved using the GV as outlined in Algorithm 1 (steps 5–9). If the transaction is verified, it is mined in the BC. As was noted in Section 3.3.1, transaction removals are handled in batches, i.e., at the end of each CP. Thus, the miner will remove all summarized transactions (discussed in detail in Section 4) from its copy of the BC in the next CP (step 10 and 11).

Recall that MOF-BC rewards users for optimizing BC memory footprint. Similar to how rewards were handled for removing a transaction, the RMA calculates the rewards value (*Reward*) for each user using the following:

$$Reward = \sum_{i=1}^{k} t_i.pages - Sum.pages$$

Where k is the total number of transactions that are summarized and *Sum* is the summarized transaction.

(3) Aging: The data of the IoT devices is either stored within transactions or in a separate storage (e.g., cloud storage) with the hash of the data stored in a transaction (i.e., the data is linked to the BC). To optimize memory consumption, a user may decide to compress the stored data which is known as aging in the literature [28]. However, applications and services that use the compressed data may be impacted depending on the extent of the compression particularly when lossy compression is used.

In MOF-BC, the users can age their data stored in or linked to a transaction. To age data either the user or a third-party, e.g. the cloud storage, passes the original data through an aging function as in [28]. If a third-party ages data, it must send the aged version of the data to the user for verification purpose. To prove that the user is the generator of the original transaction, i.e., the transaction that contains or links to the original version of the data, the GV is included in the aged transaction.

The user then broadcasts the aged transaction to the network. The miners verify the transaction by verifying the GV of the original transaction as outlined in Algorithm 1. After verification, the aged transaction is mined in BC. On receipt of the aged transaction, miners can remove the corresponding original transaction from the BC in the next CP (see Section 4). If the removed transaction is the input of another transaction, then a redirection flag is set in the hash of that transaction in the block, which implies that the corresponding transaction is updated to a new transaction (i.e. redirected). To ensure that users can find the reference to this new transaction (i.e., the redirected address), we use a shared read-only central database known as *blackboard*. Multiple replications of the blackboard exist to reduce the risk of single point of failure and ensure scalability. The blackboard is managed centrally by a Blackboard Manager Agent (BMA). The BMA populates the blackboard with the IDs of the aged transactions that have the redirection flag set. A malicious BMA may either do not store or alter the data it receives from the nodes. However, this can be detected by the node (or agent) that originally generates the data to be stored in the BC.

3.3.2. SP-Initiated memory optimization (SIMO)

The core functions offered in SIMO are very similar to those in UIMO, i.e. removal and summarization of transactions and aging of data. In SIMO, the SP is aware of the GV-PK$^+$ and GVS of the GVs used by the devices since it exerts control over them and thus the SP can initiate the aforementioned actions. The P.T.ID differs for each transaction, thus, the SP must generate a unique GV for each transaction that its devices are generating as the SP is the only entity that knows the GV-PK$^+$ and its corresponding GV-PK$^-$. This method will not scale for SPs with billions of devices as they require to response to billion of GV generation requests from their devices. To address this challenge, in SIMO the P.T.ID is excluded from the GV generation and the GV is simply generated using the hash of the GVS. Using the same hash for a number of transactions leads to the same GV value for them. However, as these transactions are created by the SP and are thus likely to reference a large number of users, the risk of privacy exposure for individual users is low.

3.3.3. Network-Initiated memory optimization (NIMO)

UIMO and SIMO afford significant flexibility to the user and SP for management of their stored transactions. However, there are certain overheads associated with this flexibility. Actioning any of the functions (summarization, removal and aging) requires the responsible entity to create a new transaction. The SP in particular could potentially be responsible for managing a large number of devices and may not want to be burdened with this overhead for all of them. Since these new transactions need to be mined into the BC, there could be an adverse impact on the BC throughput, i.e., the number of transactions stored in the BC per second. Finally, the removal of a stored transaction in UIMO and SIMO achieves zero memory savings as the corresponding remove transaction needs to be added to the BC.

To address these challenges, MOF-BC offers NIMO, whereby users and SPs can offload these functions to the network. NIMO offers the following memory optimization modes (some of which are very similar to the functions undertaken in SIMO/UIMO):

1. Temporary

2. Permanent
3. Summarizable

The MOM field in the transactions (see Section 3) indicates the optimization mode used by the transaction with a different value identifying each MOM. The MOM field must be set when the transaction is generated.

(1) Temporary: Certain transactions between IoT nodes might only need to be valid for a specified period of time known as Time To Live (TTL). For example, a home owner may grant access to the data from a sensor to the SP for a year's service. MOF-BC introduces *temporary* transactions for such cases. A temporary transaction is removed from BC (as discussed in Section 4) after TTL specified in the MOM-Setup field of the transaction.

Recall from Section 3.2 that the MOF-BC applies a storage fee to all transactions. To encourage the generation of temporary transactions, MOF-BC introduces flexible storage fee for user/SP that do so:

$$Storagefee = Pages * TTL * Rate$$

Rate is the cost of storing a page for a period of time and can be used as a weight to adjust the transaction fee. The rate could be progressively increased for transactions that are stored in the BC for a longer time. An estimation of the rate can be made based on the cost of buying storage media. A 1T SSD storage media costs around 650$. The optimal lifetime of storage media is 5 years [29]. In our experiments, a page size is 1 Kbyte (see Section 6). Thus, the cost for storing each page for 5 years is 0.00000065$. Although storage cost of an individual transaction is small, with billions of IoT devices generating transactions, these costs will add up significantly. The storage fee of a temporary transaction can be paid by any rewards accrued as per the discussion in Section 3.3.1.

(2) Permanent: These transactions are stored permanently in the BC. Note that, all current BC instantiations only support permanent storage of transactions. A fixed storage fee is applied, which is defined based on the application.

(3) Summarizable: In NIMO, the user/SP must specifically mark transactions that should be summarized when they are created. Since the network handles the summarization process, there is no need for GV as in the case of SIMO/UIMO. Subsequently the structure of the NIMO summary transaction is as shown in Fig. 10.

Summary transactions are mined as normal into the BC. At the end of each CP, a Summary Manager Agent (SMA) summarizes all summarizable transactions in a ledger in a single consolidated summary transaction. Although the summary transaction is initiated by the SMA, the user/SP first needs to permit its transactions to be summarized by setting the MOM of its transactions as summarizable. This ensures that the user/SP has full control over which transactions should be summarized, while the network specifies when to summarize the transactions to enhance the BC memory optimization. We will further elaborate on the choice of the CP in our experimental evaluations in Section 6.2.2.

Fig. 11 outlines the main steps of summarization MOM. The SA scans newly mined blocks for summarizable transactions and sends references to these to the SMA. When the current CP concludes, the SMA populates a summary transaction, as outlined in the summarizing process in Section 3.3.1, for each ledger that has summarizable transactions. Note that the summarizable transactions can no longer be removed from the BC. A comprehensive study on the impact of summarizable transactions on BC size is presented in Section 6.2.2. Next, the SMA broadcasts the transaction to the network to be mined in the BC. The miners may randomly check the summary transaction by summarizing the summarized transactions and comparing the results with the summary transaction. This protects the network against malicious SMA which generates fake summary transactions. The associated processing overhead

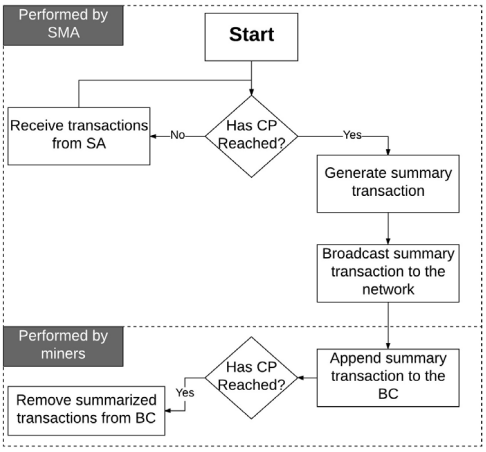

Fig. 11. The process of network-initiated summary transaction.

with verifying the summary transaction on the miners is further reduced using the distributed trust algorithm outlined in Section 3.1.

As outlined in Section 3.3.1, the RMA calculates the rewards for each user/SP that summarized its transactions in the BC. The total value of rewards (*RewardT*) offered to the owners of summarized transactions is:

$$RewardT = \sum_{i=1}^{k} t_i.pages - Sum.pages$$

Where k is the total number of transactions to be summarized and *Sum* is the summarized transaction. The share of node N (*RewardN*) in *RewardT* is:

$$RewardN = RewardT * \frac{t_N.pages}{\sum_{i=1}^{k} t_i.pages}$$

Although the summary transaction is mined into the BC, the summarized transactions still need to be removed. This process is performed at the end of the CP and is discussed in Section 4.

3.3.4. Summary

In this section, we summarize key advantages and implications of using multiple MOMs employed in the MOF-BC in Table 3. These arguments are also relevant for SIMO and UIMO, since the underlying functionality for removing and summarizing a transaction in SIMO/UIMO is similar to the temporary and summarizable MOM in NIMO.

4. Batch removal of transactions

In this section, we discuss the process for removing transactions. We also introduce removal of a ledger. Recall that in conventional BCs, removing a transaction breaks the block hash consistency as the hash is generated over the entire contents of the block as:

$$Block_{ID} = H(T_1 \| T_2 \| \cdots \| T_k \| block.header)$$

Where k is the total number of transactions in the block, T_k is the content of the transaction, and *block.header* is the content of the block header. To ensure the block hash consistency while allowing removal of transactions, in MOF-BC the block hash, i.e., $Block_{ID}$, is calculated as:

$$Block_{ID} = H(T.ID_1 \| T.ID_2 \| \ldots \| T.ID_k \| H(block.header))$$

T.ID is the hash of the transaction content, thus, the transaction content is included in the block hash generation. To remove a

Table 3
Discussion about the proposed optimizations.

Optimization mode	Benefits	Implications
Temporary/Removing	Suited for transactions that have a specific lifetime; transaction is removed after the TTL freeing up memory and enhancing privacy; Reduces memory consumption and long term fees.	The transaction is only valid for the specified time and thus can only be audited and referenced up to that time
Permanent	Suited for transactions that need to remain in the BC forever; These transactions can always be referenced and audited.	Constant memory usage and higher transaction costs.
Summarizable	Suited for transactions that are related to one another; after summarization it is still possible to verify if one of the original transactions belongs to the group of summarized transactions; memory savings and increased privacy is manifested after the summarization process; reduces monetary cost; timestamps of the individual transactions can be audited.	The content of the summarized transactions cannot be accessed.
Aging	Suited for applications where older data is not as frequently referenced; Reduces memory consumption and monetary cost.	The accuracy of the data is reduced and can thus impact the quality of service from the SP.

transaction, its content is removed from the BC, however, *T.ID* and *P.T.ID* remain stored. The *T.ID* ensures the block consistency while the transaction is removed. *P.T.ID* ensures that the chain of transactions in the ledger is not broken after removing a particular transaction.

Removing a ledger: In IoT, the users/SPs may demand to remove all information of a particular device. For example, a device installed in a user's home may break or he may wish to stop using it. The user may no longer wish to keep a record of the transactions pertaining to this device. Thus, the transaction ledger is removed using a *ledger-remove* transaction. Normally as outlined in Section 3.3.1, removing multiple transactions would require the user or SP to provide the GV of each transaction. But since all transactions being removed are chained together in a ledger, it is sufficient to only include the GVS and GV-PK$^+$ associated with the genesis transaction of that ledger. This not only reduces the size of the transaction but also simplifies transaction processing.

Cleaning Period (CP): As outlined in removal part of Section 3.3.1, the miners process the removal of transactions in batches over a periodic Cleaning Period (CP). The CP value is application based. We further elaborate on the choice of the CP and its impact on the BC size and the resources expanded at each miner in the evaluations in Section 6.2.

The removal of all transactions at the end of each CP incurs processing overhead on the miners. To reduce this overhead, MOF-BC introduces a Service Agent (SerA) that handles the removal process and makes its updated version of the BC available for all miners to download, which in turn reduces the processing overhead on the miners. The miners (or some of them) may decide to perform the removals by their own and compare the result with the updated version of the BC available from the SerA to ensure that the SerA is not compromised. Similar to SMA (see Section 3.3.3), distributed trust algorithm is used to decrease the processing overhead.

5. Applications

In this section we discuss multiple applications that can use MOF-BC to highlight its applicability and usability. BCs can be broadly categorized as: (i) permissioned: where only authorized nodes can join the BC, e.g., Hyperledger [17], and (ii) permissionless: where any node can join the BC, e.g. Bitcoin [3]. As highlighted in Fig. 3, MOF-BC is implemented as a separate layer on top of the core BC layer which makes it applicable for both permissioned and permissionless BCs.

Recall from Section 3.3 that MOF-BC limits the auditability of the removed transactions as only the hash of the affected transaction remains in the BC. This limits the applicability of the MOF-BC for BC use cases where high auditability is demanded either by the BC users or to verify transactions in a ledger. Consequently, in BC instantiations which use Unspent Transaction Output (UTXO), the removal of a transaction might cause inconsistency in the BC or open up the possibility of double spending as detecting such an attack requires validation of the corresponding input transactions, which must thus be stored in the BC in their entirety. However, for such applications the removal can be performed after a specified period of time which is long enough to ensure that the aforementioned attacks are no longer possible, e.g., one may consider X number of transaction confirmations prior to be able to remove a transaction. In case that the full history of transactions is demanded for transaction verification, the SerAs store the full version of the BC. However, this decreases the privacy benefits of MOF-BC as transaction history is still fully available. Thus, the trade-off between audibility and privacy should be considered while designing such BCs.

In the following, we provide exemplary use cases for each of the defined optimizations methods:

Removal (see Sections 3.3.1 and 4): Consider a smart thermostat installed in a home. This thermostat may have generated a number of transactions which capture various interactions with the service provider such as updates to the temperature settings initiated by the residents, maintenance updates, etc. Initially, the thermostat owner sets up the device with a GVS and GV-PK$^+$ that is used by the device to add GV in its future transactions. Recall from Section 3.3.1 that the GV enables the user to remove the transactions from the BC. Now assume that after 3 years of operation the thermostat breaks down. Consequently, no further transactions will be appended to this ledger. Thus, the owner can remove the entire ledger by issuing a remove transaction using GV of the first transaction.

The data of the IoT devices may be stored either off-the-chain or in BC. Due to significant packet and storage overheads, in most applications, only the hash of the data is stored in the BC and the data is stored off-the-chain in a separate storage. The optimizations employed by MOF-BC only require the removal of transactions in the BC and not the corresponding data (which is stored separately). Data ownership should be agreed by the user and third-party which is beyond the scope of MOF-BC. In instances where the data is stored in the BC, removal of a transaction would also remove the associated data. There is an increasing demand by the SPs of IoT devices to maintain the data of IoT devices even when the device is no longer receiving service to improve their service. On the other side, the users require the data to be removed to ensure their privacy. This conflict of interest is an open research challenge which is beyond the scope of MOF-BC.

Summarizing (see Sections 3.3.1 and 3.3.3): An electricity provider may install sensors in the customer site to monitor different metrics, e.g., temperature, to balance the load in a smart grid. The sensors frequently send transactions which capture various metrics, including home temperature, to the electricity provider. The customer may wish to summarize the transactions for the entire site from time to time in order to save cost and enhance

his privacy. Using the Merkle tree stored in the corresponding summary transaction, the user can prove the existence of one of the constituent transactions in the BC on demand.

Aging (see Section 3.3.1): Consider a company which uses cloud storage to store the data recorded by all security cameras installed on its premises. When the data is stored in the cloud, the hash of the data is signed in a transaction and stored in the BC, i.e., the data is linked to the BC, which protects data against tampering. With passage of time and depending on the frequency of updates, the amount of data stored in the repository may become very large motivating the company to compress, i.e., age, this data and create space for new data. After aging data, the company must issue an aging transaction which contains the hash of the aged data.

Temporary (see Section 3.3.3): BC enables energy producers, e.g., smart homes equipped with solar panels, and consumers to trade energy distributively and privately using smart contracts. The smart contract holds the payment of the consumer until the consumer confirms reception of the energy, then pays the energy to the producer. The smart contract can be generated as a temporary transaction which is removed after a certain time period following the trade.

6. Evaluation and discussion

In this section we provide qualitative security analyses as well as quantitative performance evaluations.

6.1. Security analysis

We first discuss MOF-BC security and fault tolerance. It is assumed that the adversary can be any node in the BC network, e.g., miner, SP, agent, or cloud storage. Adversaries are able to sniff communications, create fake transactions, attempt to change or remove stored transactions in BC, and link a user's transaction to each other to uncover the real identity of the user. However, they are not able to compromise the standard encryption algorithms that are employed.

Security: In this section we discuss the security of MOF-BC and its resilience against some possible attacks. Recall from Section 3 that MOF-BC is implemented as a separate layer on top of the underlying BC instantiation, thus the security of the underlying BC is not affected by attacks on MOF-BC. We consider the following attacks on MOF-BC:

Transaction Removal Attack: In this attack, the malicious node attempts to remove the transactions generated by other nodes. Recall that in MOF-BC, the following three entities can initiate the memory optimization: the user in UIMO (Section 3.3.1), the SP in SIMO 3.3.2, and the network in NIMO 3.3.3. To remove the transactions of a user (similar process would apply for the SP), the malicious node would have to create a remove transaction which appears to have been generated by the corresponding node. This requires the node to know: (i) The GV-PK^{+} which is used to decrypt the signature of the GV, (ii) The PK^{-} that corresponds to the GV-PK^{+} for signing the remove transaction. While the former can be obtained by eavesdropping a remove request by the same user/SP, the latter would only be known to the user/SP. In the worst case, if the attacker somehow finds the PK^{-}, it still requires the GVS to decipher the GV. A brute force attack is unlikely to succeed given that a collision resistant hash such as SHA-3 is employed.

Recall that unlike UIMO and SIMO where transactions are removed using a remove transaction and GV, in NIMO transactions can be removed only according to the MOM field populated initially by the transaction generator. The MOM field is included in the signed hash of the transaction by the transaction generator, thus no one can modify the MOM which makes NIMO resilient against this attack.

False Storage Claim: In this attack, the malicious miner claims to have a locally stored copy of the BC to receive incentive from the StMA (see Section 3.2). Recall that the PA migrates randomly between the miners that have made claims and validates their claim by examining the storage space that they have dedicated to the BC. To verify the storage claims of all miners, the PA must visit all miners at least once during the CP. The frequency with which the PA visits the miners can be defined by network designers considering the implications outlined below.

A malicious miner may decide to store the BC temporarily and then remove it after the PA visit to gain profit while not having to dedicate storage permanently. To make this viable, the profit gained by miner X must be higher than the share of X from storage fee in that particular period (denoted by $Share_X$) as shown below:

$$|P_X - (C_r + C_w)| > Share_X$$

where P_X is the profit that X gains by using its storage for other purposes, e.g., offering it as storage to a cloud service, during the period that it does not store the BC, C_r and C_w are the costs of removing and writing the BC, respectively, which includes HDD read and write and bandwidth consumption. The frequency with which the PA visits the miners is unknown. Moreover, the PA does not follow a specific pattern when it visits the miners. Thus, the precise duration for which a miner should temporarily store the BC is entirely unknown to the attacker. The more frequent and random PA visits X, the lower the chance of the successful attack.

Eclipse attack: As outlined in Fig. 3, BC nodes use a peer-to-peer network to communicate. All nodes broadcast packets to their neighboring nodes. A node may choose to be an agent by performing functionalities of one of the agents outlined in Section 3.1. An agent, regardless of its type, might be isolated from the network when all the nodes in its one-hop neighborhood collude with each other and drop all transactions or blocks to or from the agent. This attack is known as eclipse attack in the literature [30]. The aim of this attack is to prevent normal well-behaving nodes from receiving services offered by the agent. To prevent the eclipse attack, the agent replicas should be positioned such that they are physically distant from each other. This can be achieved by using the number of hops separating the replicas in the p2p network as a measure of "distance" and assuming a certain minimum inter-replica *"hop-distance"*, which can be configured based on the application and network size. Thus, in case one of them is isolated, other agents can continue to provide service. We elaborate more on the impact of the number of agents isolated due to eclipse attacks on the service provided to the nodes when we discuss fault tolerance at the end of Section 6.1.

Malicious SP: A SP may try to maliciously remove a transaction to disown responsibility. Consider the following example. Alice (the renter of a home) has sent a transaction to Bob (the home owner and thus the SP) alerting him that the fire alarm is broken and requesting servicing. Bob ignores the transaction. A fire breaks out in the home causing significant damage. Bob wishing to shirk responsibility may attempt to remove Alice's transaction and falsely alleges that she was responsible for the fire as the faulty fire alarm was not reported. However, even if Bob removed Alice's transaction, the corresponding hash is still present in the BC. Assuming that Alice has stored a copy of the transaction locally, she can readily verify this and thus implicate Bob.

Colluding attack: A group of malicious agents may collude to remove or summarize transactions which may enhance the possibility of successful attacks, e.g., double spending, in the future as the removed transactions can no longer be used to verify the chain of transactions. Recall that all the miners in the network verify the transactions and blocks, thus the attack will be detected. However, if the attackers compromise the miners in the network, they might be successful in removing or summarizing transactions

in the BC. The simulation results discussed in Table 1 show the minimum number of honest miners that must be present in the network to prevent such attacks.

Reward tracking: An attacker may eavesdrop on the communication between the users/SPs and the bank to discover the PK^+ that rewards are paid to. The attacker may subsequently monitor the rewards paid to that particular PK^+ and the expenditure of the same, which may reveal privacy sensitive information about the owner of the PK^+. The users/SPs encrypt the redeem request using the PK^+ of the bank, which ensures only the bank can read the GVs and the PK^+ to be paid to. Additionally, the users/SPs can exchange their earned coin or reward with any other user/SP. Thus, even knowing the PK^+ of the rewards corresponding to each user/SP does not compromise user/SP privacy.

Malicious agents: The agents may perform malicious activities in the network. It is assumed that agents are selected to run on nodes which have higher security, e.g. the machines with high resources to perform the security tasks. However, we conservatively assume that either initially a malicious node might be selected as an agent or an agent might be compromised. A malicious agent can create fake transactions or blocks, remove or summarize transactions without permission, stop rewarding the users, or change or remove data in the blackboard. The participating nodes in the BC can collaboratively detect all of the aforementioned misbehaviors as outlined below. Once a malicious agent is identified, the miners isolate it and choose a new agent as a replacement.

SA: Recall from Section 3.1 that SA searches newly mined blocks for particular transactions, e.g., summarizable, and sends them to the relevant agent. A malicious SA may avoid relaying (some of the) transactions to the agents, as a result of which the skipped transactions will not be processed by the agents, e.g., summarized by SMA. Consequently, the user will no longer receive reward for the amount of space he saved for summarization. To prevent this attack, the agents that receive transactions from the SA randomly search the new blocks for the relevant transactions and compare them with the transactions sent by the SA. Recall from Section 3.1 that to reduce the processing overhead on the agents, MOF-BC employs distributed trust. Accordingly, as the agents build up trust in each other, fewer transactions from the new blocks are verified. Distributed trust is also employed by all agents discussed in the rest of this section.

SMA: Recall from Section 3.3.3 that the SMA summarizes all the summarizable transactions in a ledger by creating a summary transaction at the end of each CP. A malicious SMA may: (i) summarize transactions that are not permitted to be summarized by their generator, or/and (ii) create fake summary transaction, e.g., by creating a fake Merkle tree, of summarizable transactions. To detect this attack, the miners randomly summarize the summarized transactions using the same method as the SMA and validate the received summary transaction (see Section 3.3.3).

SerA: Recall from Section 3.1 that the SerA actions the removal of transactions on its own copy of the BC and makes it available for all miners, thus reducing the processing overhead for removing the transactions on miners. A malicious SerA may modify blocks, by either removing or modifying the constituted transactions. The miners verify the received blocks by ensuring that the SerA only removed the expired transactions (see Section 4) and thus can detect a malicious serA.

RMA: Recall from Section 3.3.1 that the RMA calculates the reward for each user according to the saved storage space. A malicious RMA may avoid rewarding users or partially reward them. A compromised RMA can be detected by the nodes in BC network as they would no longer receive reward or receive partial rewards. Such nodes inform the rest of the network of the malicious behavior of the RMA (see Section 3.3.1).

BMA: As outlined in Section 3.1 BMA manages a central read-only database. A malicious BMA may modify or stop writing data to the blackboard. A compromised BMA can be detected by the participating nodes in the BC or the agents that populate the blackboard as the information that they have sent to the BMA is not stored in the blackboard (or is changed) (see Section 3.3.1).

StMA: Recall from Section 3.2 that the StMA collects the storage fees and distributes the collected amount among the miners. A malicious StMA may stop rewarding the miners for the amount of storage they allocated to the BC. This can be detected by the miners as they are not paid for their service (see Section 3.2).

PA: Recall from Section 3.2 that PA is a mobile agent which monitors the storage space of the miners dedicated to the BC. A malicious PA may collude with miners such that they submit false claims for storage fee without actually allocating space to store the BC, thus allowing them to receive incentives for no investment. The malicious behavior of the PA can be detected by the BC nodes which request to download a copy of the BC from the miner, e.g., new nodes joining the network or existing nodes wishing to start serving as a miner. If the request is rejected by the miner as it would not have a local copy of the BC, then the node informs the StMA. The StMA monitors the storage space dedicated to the BC in the suspected miner. In case that the miner does not store the BC, the StMA removes the miner from the list of miners that receive reward for storing BC (see Section 3.2).

Next, we analyze the resilience of MOF-BC against each attack and the likelihood of the attack to happen based on European Telecommunications Standards Institute (ETSI) [31] risk analysis criteria. These criteria evaluate each attack based on the following five metrics: (i) *time:* the cumulative time for an attacker to first detect a vulnerability, and subsequently plan and launch a successful attack, (ii) *expertise:* the generic expertise that the attacker must possess about the underlying principles in order to orchestrate the attack, (iii) *knowledge:* specific information that is available about the target system, e.g., security configuration, (iv) *opportunity:* the duration and nature (e.g., continuous or intermittent) of access to the system needed for launching the attack, (v) *equipment:* software and/or hardware necessary for conducting the attack. The evaluation results are presented in Table 4. Readers are referred to [31] for detailed information about measuring the aforementioned metrics.

Table 4
An analysis on attack likelihood and attack resistance of MOF-BC based on ETSI.

Attack	Resistance to attack	Attack likelihood
Transaction removal	Beyond high	Unlikely
False Storage Claim	Moderate	Possible
Eclipse attack	Moderate	Possible
Malicious SP	Basic	Likely
Colluding attack	Beyond high	Unlikely
Reward tracking	Beyond high	Unlikely
Malicious Agents	High	Unlikely

Observe that, transaction removal, colluding attack, reward sniffing, and malicious agent attacks are unlikely to happen, while eclipse attack is possible and malicious SP is likely. Although eclipse attack is possible, the redundancy offered by multiple agents ensures that properly functioning nodes can respond to the requests that were originally meant for the isolated agents. However, the latency in servicing these requests may be affected. Recall from 3.3.2 that the malicious SP knows the GV and its corresponding keys and thus is able to remove the transactions. Hence, this attack is likely and the resilience against the attack is basic which is lower compared to moderate. However, the user can later prove the malicious behavior of the SP using the remaining hash in the BC as outlined in Section 3.3.2.

Fault tolerance: Fault tolerance is a measure of how resilient an architecture is to node failures. In MOF-BC only the agents are

vulnerable to failure as the rest of the network that forms the BC operates in a distributed manner. To enhance the fault tolerance of the agents, multiple replicas of the agents work collaboratively. Thus, failure of one will not impact the network. Failure of multiple replicas of an agent may affect the fault tolerance of the MOF-BC. If the total requests generated by the participating nodes in the BC exceeds the cumulative response rate, i.e., the number of transactions being responded, of replicas, then the participating nodes experience delay in receiving service.

6.2. Performance evaluation

In this section we present extensive performance evaluations of MOF-BC. We implemented MOF-BC using C++ integrated with crypto++ library and SQLite database on a MacBook laptop (8 GB RAM, Intel core M-5y51 CPU, 1.10 GHz*4). We assume a network comprised of 900 nodes, each of which has its unique PK^{+}/PK^{-} for generating transactions. Each node generates one transaction per week. Nodes generate different types of transactions. In BC, throughput is defined as the total number of transactions that can be stored in BC per second. The BC throughput in our implementation is 90 blocks per week, i.e., one transaction for each node per week. Transactions are organized in blocks such that there are 10 transactions within one block. Since the process of mining blocks is handled by the BC layer (see Fig. 3) and is thus orthogonal to the functionality offered by MOF-BC, i.e., optimizing BC memory footprint, it has not been implemented. Each transaction contains PK^{+}, signature, T_{ID}, $P.T_{ID}$, MOM, and MOM-setup (see Section 3.3) fields.

Recall from Section 3.3.3 that in NIMO a large portion of the optimization tasks are performed by the network, thus, increases the (processing) overhead on the network compared to UIMO and SIMO where the user or SP performs most of the processing for the memory optimization. Additionally, MOMs defined in NIMO overlap with the functionality supported in UIMO and SIMO. Thus, we only evaluate NIMO since we expect that the associated overheads will be similar (or even supersede) the overheads incurred by other two optimization methods. Note that the optimized memory using the aging largely depends on type of data and aging function which are not in the scope of this paper.

We first evaluate the benefits (memory footprint optimization and monetary cost) and implications (processing time of handling memory optimization tasks) of using MOF-BC compared to a scenario where all transactions are permanently stored. We show that the benefits of MOF-BC far outweigh the small overheads. Next, we provide comprehensive evaluations on memory saved or expended by choosing different CP values.

6.2.1. BC size and transaction fee

In this section, we evaluate the impact of multiple MOMs on the BC size and the transaction fee that needs to be paid by the users. It is assumed that each node generates one transaction per week with a cumulative total of 280 transactions. All nodes generate transactions sequentially which are organized in blocks of 10 transactions resulting in a BC that contains 25200 blocks. The CP is set to 180 weeks. The evaluation metrics are computed once the BC is populated with 25200 blocks. We use 3 different instances of the BC each using a different MOM namely permanent, temporary, and summarizable, to evaluate each MOM separately. The permanent MOM represents the existing BC solutions as there is no optimization in this MOM. The CP equals to 2 weeks, i.e, 180 blocks. For temporary MOM we form three groups of nodes, each group with 300 participating nodes. All participants of a group use the same value as the TTL which is 26, 52, and 104 weeks for groups 1–3 respectively. Groups sequentially generate one transaction. To measure the transaction fee, we used the estimated storage fee in Section 3.3.3, i.e. 0.00000065$ per 5 years. The cost for storing a temporary transaction for 20 years is considered as the permanent transaction fee which is 0.0000026$. Mining fee is not considered in our study as we exclusively consider the storage fee as an evaluation metric. The implementation results are shown in Fig. 12. In Fig. 12(a) the right vertical axis shows the BC size while the left axis shows the cumulative transaction fee which is the transaction fee paid by all nodes in the network. Fig. 12(b) illustrates the cumulative processing time incurred for executing the actions associated with the 3 MOM.

Observe that, the permanent MOM is essentially similar to conventional BCs and expectedly has the highest memory footprint. The BC size of temporary MOM is the lowest as transactions are removed after their TTL expires. With the summarizable MOM, a sizeable fraction of the BC size is attributed to summarized transactions. Consequently, the BC size is greater than with temporary. However, the summarizable transaction incurs lower cost compared to the temporary. For temporary transactions, the node pays a flexible transaction fee for all its transactions. For summarizable transactions, each node receives a reward at the end of each CP which is used to cover part of the transaction fees, thus reducing the expenses incurred by the user. In our experiment at the end of the first CP each node earns 1.6 units of rewards based on the reward calculation formula given in Section 3.3.3. Following this, in each CP one summary transaction for each node is stored in the BC, thus each node can only store one transaction in the BC within the CP. The storage fee of this transaction can be paid by the earned rewards from the optimizations employed in the previous CP. Thus, each node only needs to pay for its first two transactions stored during the first CP and the rest are paid by earned rewards. As shown in Fig. 12(b), the processing time for summarizable transactions is greater than the temporary transactions. This is because the transactions that are summarized need to be removed and a new summary transaction must be mined into the BC. Note that the measured processing time is the read/write time for updating the database and thus is the HDD read/write time.

We next measure the cumulative monetary cost saved by the network participants using MOF-BC as well as the monetary cost of running MOF-BC tasks. Thus, we convert the processing time incurred by the MOF-BC to energy consumption and in turn to monetary cost. To convert the processing time to energy consumption, we base our calculations on the energy consumptions measured by the authors in [32]. The consumed energy during the load time for HDD is 8.4 W. Thus, the cumulative consumed energy for temporary and summarizable MOM is 781 and 3305 W respectively. We next measure the cost for energy. Based on the market price, the energy price is 28.52 c/Kwh [33]. The saved and incurred costs are presented in Table 5. The saved cost is the cost that each user saves in transaction fee and is measured by subtracting the cost that the user pays for using the temporary and summarizable MOM from the cost that the user would have paid for the permanent MOM, which is essentially similar to the conventional BCs, and the cost users pay using each MOM. The incurred cost includes the cost incurred at the miner for executing the removal process (see Section 4).

It is evident that the cost saved by the MOF-BC is by far higher than the cost incurred for processing transactions. As the memory footprint of MOF-BC reduces significantly compared to conventional BC instantiations, the miners are required to expend less memory to store the BC and thus can save monetary cost of purchasing and maintaining extra storage. The exact amount of saving depends on multiple factors including the storage space in the miner, the BC size, and the delayed optimizations which are application-specific, thus, we are unable to provide any estimation on this additional advantages of the MOF-BC.

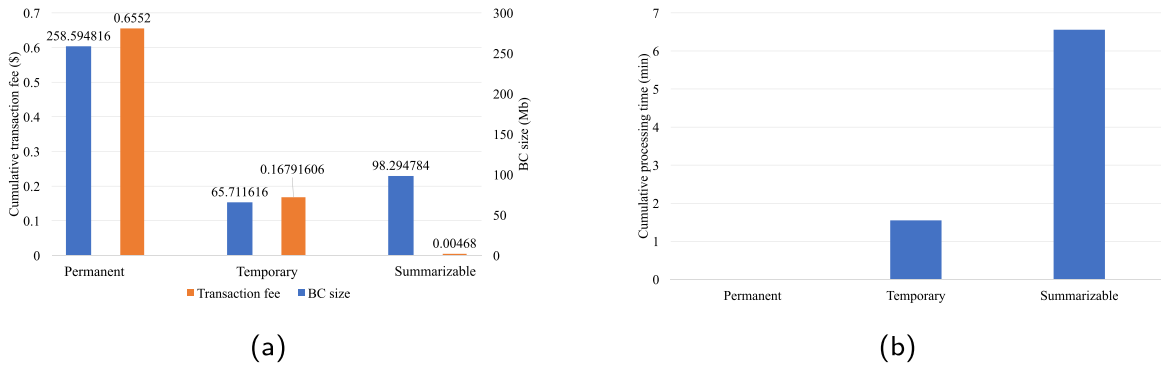

Fig. 12. Evaluation of the (a) cost and memory (b) processing time.

Table 5
The saved and incurred cost by MOF-BC ($).

	Temporary	Summarizable
Saved cost	0.48728394	0.65052
Incurred cost	0.000374948	0.001586572
Benefit/Cost ratio	1300	410

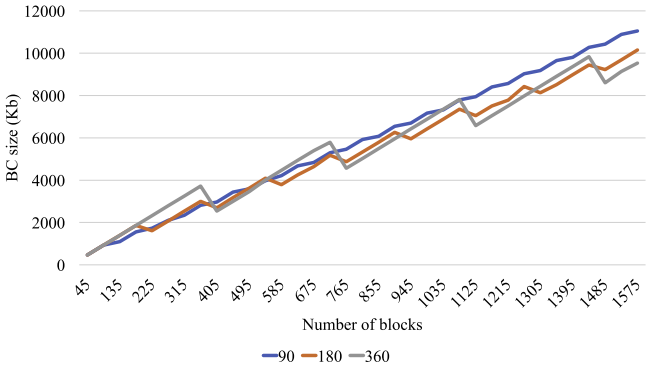

Fig. 13. The impact of the CP on the BC size.

6.2.2. Impact of varying the CP

Recall from Section 4 that in MOF-BC at the end of each CP each node that stores a copy of BC removes all removable transactions from its BC copy. Additionally, the SMA generates a summary transaction for all summarizable transactions in a ledger and broadcasts it to be mined into the BC. We disregard the delay of creating a summary transaction and the time taken for the broadcast to propagate through the network as these have no effect on the measured metrics. Thus, we assume that the summary transactions are immediately mined into the BC after generation.

The cleaning process (see Section 4) and summarization (see Section 3.3.3) tasks performed at the end of each CP directly affect the BC size. Thus, we first measure the cumulative size of the BC while varying the CP value. In this setup, the participating nodes are grouped in three groups, each with 300 nodes. The nodes in each group generate transactions with a particular MOM, i.e., permanent, temporary, and summarizable. The nodes that generate temporary transactions are further divided in three groups, each generating transactions with different TTL values as follows: 26, 52 and 104 weeks. This allows us to highlight the impact of TTL on the BC size. This configuration is referred to as the default configuration in the rest of this section. Fig. 13 plots the BC size for 3 different CP values, which are 90, 180, and 360 weeks, as a function of the total number of blocks generated and stored in the BC. As can be inferred, with larger CP more transactions are generated which increases the size of the BC before reaching the CP. Recall from Section 3.3.3 that at the end of each CP, the SMA stores new summary transactions that can no longer be optimized to free up BC memory. Thus, with smaller CPs the frequency which the summary transactions are generated increases. Consequently, the BC size is larger with a smaller CP as compared to a larger CP.

At the end of the CP, the summary transactions must be mined into the BC. Consequently, the number of transactions generated by the users that can be mined during the CP, referred to as throughput in the rest of this section, is reduced. Next, we study the impact of varying the CP value on the BC throughput. We used the default configuration and the results are shown in Fig. 14. As expected, by increasing the CP the BC throughput also increases as the number of summary transactions decreases. These affect the BC size and number of blocks in BC, as a number of blocks have to be mined into the BC to store the summary transactions. The number of blocks in the BC further impacts the packet and processing overhead on the network for mining and broadcasting the new blocks. Before we study the implementation results and to better understand the impact of the CP on BC size and the number of blocks, we formulate their relationship as follows:

$$(1)\ BCSize = \sum_{i=1}^{CP} (TT_i.pages + TS_i.pages - SS_i)$$

$$(2)\ Numberofblocks = \sum_{i=1}^{CP} \frac{TT_i + TS_i}{Block\_size}$$

where TT denotes the total number of transactions generated by the users, TS denotes the total number of summary transactions stored in the BC, and SS is the amount of space saved after CP. We measure the BC size and the number of blocks for storing a particular number of transactions, 10,000 in our implementations, in the BC. The BC size metric shows the impact of summary transactions on the BC memory footprint. We use the default configuration and measure the two metrics when 10,000 transactions generated by the users are mined into the BC. Fig. 15 presents the results. It can be seen that for storing the same number of transactions, a larger CP results in a lower memory footprint. This can be supported by (1) $TT.pages$ and SS are greater for larger CPs, while $TS.pages$ is roughly equal for all CPs (this is because for any number of summarizable transactions only one summary transaction is stored, thus only the size of summary transaction is larger for larger CPs — see Section 3.3.4). Recall from Fig. 14 that a higher CP also leads to higher throughput, implying that the total CP rounds required to store 10,000 transactions are smaller for larger values of CP.

As shown in Fig. 15 for larger CPs, fewer blocks are required for storing 10,000 transactions. This fact can also be supported by (2) and Fig. 14. Assuming that the number of ledgers that have summarizable transactions remain unchanged, for multiple values of CP a fixed number of summary transactions (presented as TS) are stored in the BC. Thus, with smaller CP, more CPs are required to mine 10,000 transactions as more summary transactions are stored. Arguably, the mining overhead as well as the bandwidth

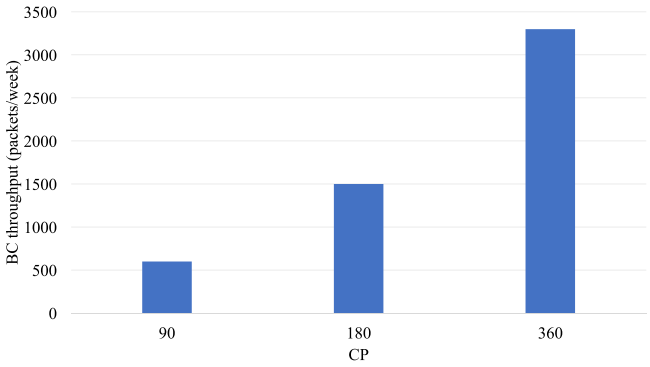

Fig. 14. The impact of CP on the BC throughput.

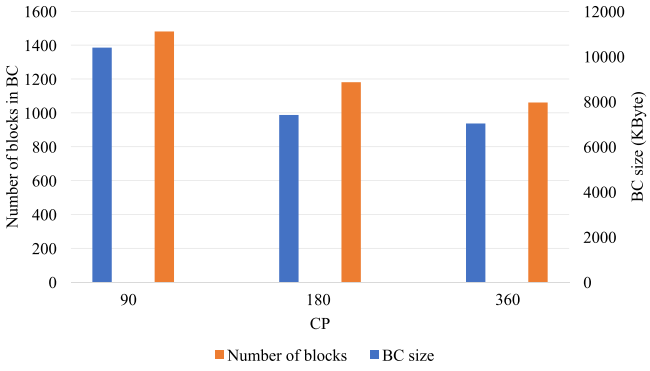

Fig. 15. The impact of CP on the size and length of BC for storing 10k transactions.

consumption are decreased and the scalability is increased as fewer blocks need to be broadcast in the network.

Our results have so far shown that larger CPs improve the BC throughput and memory requirement compared to the smaller CPs. However, using large CP incurs storage overhead at the miners for storing expired transactions including temporary transactions whose TTL has passed and summarizable transactions, which is referred to as Delayed Memory Optimization (DMO) in the rest of this paper. By delaying the process of summarizing summarizable transactions the motivative privacy and memory optimization benefits of generating the transactions as summarizable cannot be achieved. Thus, we categorize them as DMO. To measure the impact of the CP on DMO, we consider the default configuration. Fig. 16 plots the cumulative amount of DMO in each miner as a function of the number of blocks for different CPs. With a large CP, the time between when the cleaning process is executed is longer and thus a greater number of expired transactions accumulate (this is shown as *SS* in (1)). Consequently the DMO is far greater than with a smaller CP.

The DMO increases the monetary cost of the miners for storing delayed transactions which can be removed. To evaluate the impact of CP on the cost, we measure the cumulative cost each miner incurs in storing DMO. We based our measurements on the estimated cost in Section 3.3.3, i.e., 0.00000065$ for five years. The DMO cost can be measured by multiplying the DMO given in Fig. 16 by 0.000000065$ that is the cost of storing one page of data, i.e., 1 KByte, for six months (which equals with 45 blocks). The latter is the base when we measured the DMO.

In conclusion, for choosing a CP the trade-off between throughput memory footprint on the one hand and DMO and monetary cost on the other needs to be considered.

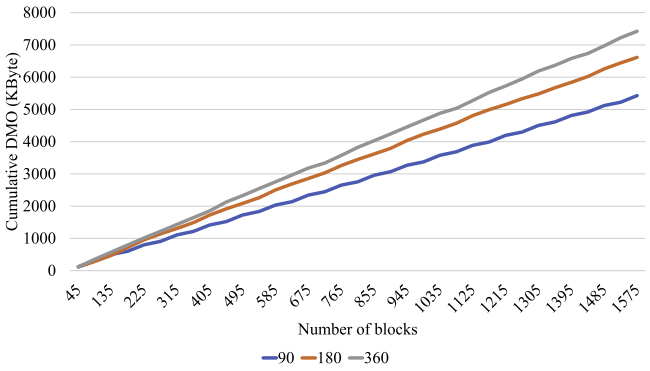

Fig. 16. Evaluation on cumulative DMO.

7. Conclusion

The immutable nature of the BC makes it impossible to modify or remove a previously stored block and thus increases BC security. However, it leads to storage, privacy, and cost challenges for BC users particularly in large scale networks like Internet of Things (IoT). This paper, proposed a Memory Optimized and Flexible BC (MOF-BC) that empowers users and Service Providers (SPs) to remove a previously stored transaction or reduce its size by summarizing transactions or aging the data in transactions. The user/SP may decide to offload the associated overheads for optimizing BC memory to the network using Network-Initiated Memory Optimization (NIMO). To encourage users/SPs to employ memory optimization, MOF-BC offers flexible transaction fees and rewards. MOF-BC introduced a Generator Verifier (GV) which addresses key management for large scale networks while maintains the user/SP privacy. Security analysis showed the robustness of the MOF-BC against several attacks. Implementation results show that MOF-BC achieved lower memory consumption while incurring a small processing overhead.

By enabling the BC users to exercise the right to be forgotten, MOF-BC is the first BC instantiation compatible with this requirement of EU General Data Protection Regulation (GDPR). Besides the right to be forgotten, GDPR requires the user data to remain anonymous and unlinked to the true identity of the users. Similar to existing BC instantiations, user deanonymization by tracking multiple identities is possible in MOF-BC. In UIMO and SIMO the user needs to issue an explicit remove transaction for removal of an existing transaction in the BC. The remove transaction contains the GV-PK^+ for the user, which could potentially be used for deanonymization of the user. However in NIMO, memory optimization is embedded in the original transaction and is specified once the transaction is generated. Thus, by removing a transaction all traces for user deanonymization are removed.

MOF-BC outlines core functions and concepts of a removable BC. We believe MOF-BC can serve as a starting point for further research investigating the storage issues concerning BC. In the following, we outline a few ideas for future research:

- *Smart contract:* MOF-BC employs multiple agents to reduce the packet and processing overhead while optimizing BC memory footprint. Some of these agents including bank, RMA, and StMA could be replaced with smart contracts to impart more distributed control within MOF-BC.
  To be able to remove a smart contract from BC, all nodes participating and affected by the contract must be in agreement. Further investigations are required into how to reach this agreement and deciding which nodes will action the smart contract removal process.

- *Implementation:* In this paper, we implemented MOF-BC in isolation of the underlying BC to evaluate its performance. In the future, we will explore a full stack implementation on top of a BC instantiation such as Ethereum and study its performance in real-world scenarios.
- *Full node:* As outlined in Section 5 in some applications the full history of the BC might be demanded by participants to verify transactions. However, making the full history available to all participating nodes reduces the privacy benefits of MOF-BC. As a future work, we aim to study the possibility of the ability of verifying the removed transactions while protecting the user privacy.

Acknowledgement

A.D.'s work is supported by a Data61 Ph.D. scholarship.

References

[1] M. Abramaowicz, Cryptocurrency-Based law, Ariz. L. Rev. 58 (2016) 359.
[2] A. Dorri, et al., BlockChain: a distributed solution to automotive security and privacy, IEEE Commun. Mag. (2017) arXiv preprint arXiv:1704.00073.
[3] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, 2018.
[4] Slock, 2017. https://slock.it/usn.html.
[5] S. Rowan, M. Clear, M. Gerla, M. Huggard, C.M. Goldrick, Securing vehicle to vehicle communications using blockchain through visible light and acoustic side-channels, 2017. arXiv preprint arXiv:1704.02553.
[6] K. Christidis, et al., Blockchains and smart contracts for the internet of things, IEEE Access 4 (2016) 2292–2303.
[7] BlockChain, 2017. https://blockchain.info/charts/blocks-size.
[8] A. Dorri, S.S. Kanhere, R. Jurdak, P. Gauravaram, LSB: a lightweight scalable blockchain for iot security and privacy, 2017. arXiv preprint arXiv:1712.02969.
[9] N. Apthorpe, D. Reisman, N. Feamster, A smart home is no castle: privacy vulnerabilities of encrypted iot traffic, 2017. arXiv preprint arXiv:1705.06805.
[10] EU, 2018. http://europa.eu/rapid/press-release_MEMO-17-1441_en.htm.
[11] medium, 2018. https://medium.com/coinmonks/the-right-to-be-forgotten-as-the-main-risk-for-blockchain-technology-4f6a7ecca14b.
[12] cointelegraph, 2018. https://cointelegraph.com/news/child-abuse-content-on-bitcoin-blockchain-can-node-operators-be-prosecuted.
[13] theindependentrepublic, 2018. https://theindependentrepublic.com/2018/03/23/bitcoins-btc-story-may-have-come-to-an-end/.
[14] G. Gutoski, D. Stebila, Hierarchical deterministic bitcoin wallets that tolerate key leakage, in: International Conference on Financial Cryptography and Data Security, Springer, 2015, pp. 497–504.
[15] R.C. Merkle, A digital signature based on a conventional encryption function, in: Conference on the Theory and Application of Cryptographic Techniques, Springer, 1987, pp. 369–378.
[16] G. Wood, Ethereum: a secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper 151, 2014.
[17] C. Cachin, Architecture of the hyperledger blockchain fabric, in: Workshop on Distributed Cryptocurrencies and Consensus Ledgers, 2016.
[18] S. Huh, S. Cho, S. Kim, Managing iot devices using blockchain platform, in: Advanced Communication Technology (ICACT), 2017 19th International Conference on, IEEE, 2017, pp. 464–467.
[19] P.K. Sharma, S. Singh, Y.-S. Jeong, J.H. Park, Distblocknet: a distributed blockchains-based secure sdn architecture for iot networks, IEEE Commun. Mag. 55 (9) (2017) 78–85.
[20] S. Popov, The tangle, cit. on, 2016, 131.
[21] H. Shafagh, A. Hithnawi, S. Duquennoy, Towards blockchain-based auditable storage and sharing of iot data, 2017. arXiv preprint arXiv:1705.08230.
[22] S.H. Hashemi, et al., World of empowered iot users, in: IoTDI, IEEE, 2016, pp. 13–24.
[23] A. Dubovitskaya, Z. Xu, S. Ryu, M. Schumacher, F. Wang, Secure and trustable electronic medical records sharing using blockchain, 2017. arXiv preprint arXiv:1709.06528.
[24] G. Ateniese, et al., Redactable blockchain–or–rewriting history in bitcoin and friends, in: Security and Privacy (EuroS&P), 2017 IEEE European Symposium on, IEEE, 2017, pp. 111–126.
[25] D.L. Fox, System and method for verifying integrity of replicated databases, US Patent 5, 765, 172 (9 June 1998).
[26] A. Dorri, et al., Towards an optimized blockchain for iot, in: IoTDI, ACM, 2017, pp. 173–178.
[27] Medium, 2017. https://medium.com/ipdb-blog/forever-isnt-free-the-cost-of-storage-on-a-blockchain-database-59003f63e01.
[28] S. Nath, Energy efficient sensor data logging with amnesic flash storage, in: Proceedings of IPSN, IEEE Computer Society, 2009, pp. 157–168.
[29] K. Croman, et al., On scaling decentralized blockchains, in: International Conference on Financial Cryptography and Data Security, Springer, 2016, pp. 106–125.
[30] A. Gervais, G.O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, S. Capkun, On the security and performance of proof of work blockchains, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 3–16.
[31] T.S. ETSI, 102 165-1 V4. 2.3 (2011-03), Technical Specification Telecommunications and Internet converged Services and Protocols for Advanced Networking, TISPAN, 2011.
[32] A. Jaiantilal, Y. Jiang, S. Mishra, Modeling cpu energy consumption for energy efficient scheduling, in: Proceedings of the 1st Workshop on Green Computing, ACM, 2010, pp. 10–15.
[33] OriginAustralia, 2017. https://www.originenergy.com.au/.

Ali Dorri is a Ph.D. student at UNSW and Postgraduate research student at CSIRO, Australia. His research interest includes blockchain applications and challenges for large scale networks including the Internet of Things, smart cities, smart grid, and e-health. He has published over 20 peer-reviewed papers. His publications in blockchain are being ranked among most popular and top-cited papers in their respective venues.

Salil S. Kanhere received his M.S. and Ph.D. degrees, both in Electrical Engineering from Drexel University, Philadelphia. He is currently an Associate Professor in the School of Computer Science and Engineering at the University of New South Wales in Sydney, Australia. His current research interests include Internet of Things, pervasive computing, crowdsourcing, embedded sensor networks, mobile networking, privacy and security. He has published over 170 peer-reviewed articles and delivered over 20 tutorials and keynote talks on these research topics. He is a contributing research staff at Data61, CSIRO and a faculty associate at Institute for Infocomm Research, Singapore. Salil regularly serves on the organizing committee of a number of IEEE and ACM international conferences (e.g., IEEE PerCom, ACM MobiSys, ACM SenSys, ACM CoNext, IEEE WoWMoM, IEEE LCN, ACM MSWiM, IEEE DCOSS). He currently serves as the Area Editor for Pervasive and Mobile Computing, and Computer Communications. Salil is a Senior Member of both the IEEE and the ACM. He is a recipient of the Humboldt Research Fellowship in 2014.

Raja Jurdak is a Senior Principal Research Scientist at CSIRO, where he leads the Distributed Sensing Systems Group. He has a PhD in Information and Computer Science at University of California, Irvine in 2005. His current research interests focus on energy-efficiency and mobility in networks. He has over 120 peer-reviewed journal and conference publications, as well as a book published by Springer in 2007 titled Wireless Ad Hoc and Sensor Networks: A Cross-Layer Design Perspective. He regularly serves on the organizing and technical program committees of international conferences (DCOSS, RTSS, Sensapp, Percomm, EWSN, ICDCS). Raja is an Honorary Professor at University of Queensland, and an Adjunct Professor at Macquarie University, James Cook University, and University of Queensland and the University of New South Wales. He is a Senior Member of the IEEE.