ApproxBC: Blockchain Design Alternatives for Approximation-Tolerant Resource-Constrained Applications

Prasanna Karthik Vairam[1], Gargi Mitra[1], Chester Rebeiro[1], Byrav Ramamurthy[2], Kamakoti Veezhinathan[1]
[1]Indian Institute of Technology Madras, India
[2]University of Nebraska - Lincoln, Lincoln, USA
E-mail:{pkarthik, gargim, chester}@cse.iitm.ac.in, byrav@cse.unl.edu, kama@cse.iitm.ac.in

***Abstract*—Blockchains are known to provide verifiable tamper-resistant trails of accepted transactions. This guarantee comes at the considerable cost of storage and computational power, thereby restricting its application. Current research has focused on alternatives such as proof-of-reputation, proof-of-stake, and proof-of-elapsed-time to reduce the computational burden on the Blockchain participants. Orthogonal to this effort, we focus on a specific set of applications which cannot commit much of storage space and computational resources, yet require only reasonable guarantees on the validity of transactions. To this end, we introduce Blockchain design alternatives, collectively called ApproxBC, that can provide proof of transactions with provable confidence bounds. Consequently, ApproxBC can considerably reduce the computation and the storage resources required, making them suitable for resource-constrained Internet-of-Things (IoT) environments. We also showcase two approximation-tolerant applications that can leverage the quicker computation and smaller storage requirements.**[1]

***Index Terms*—approximate Blockchain, approximation-tolerant applications, evidence Bloom Filter**

I. Introduction

Strategic applications such as payment gateways, digital asset tracking, and supply chain management, are now moving to a Blockchain-based distributed architecture due to the lack of security in traditional database-based centralized architectures [5]. The Blockchain consists of a list of blocks, where each block contains several transactions. For instance, in a payment system, a transaction could be a request to transfer money from one electronic account to another. Since the inception of Bitcoin in 2009, Blockchains have evolved over what can be classified roughly into three generations [10]. The first two generations have focused on managing assets and enabling smart-contracts respectively. The third generation Blockchains try to solve a wide range of problems which include scalability, interoperability, privacy, and governance. While a few research works [6] have focused on improving the scalability of Blockchains, their applicability to devices with small memories and low computational power is still a challenge. This article focuses on Blockchain design alternatives that can achieve this.

[1]Accepted for publication at IEEE Communications Standards Magazine.

We first examine the types of frauds that Blockchains can protect against. These include **(1)** expending a resource (e.g. crypto-currency, digital asset, product) that is not owned by the submitter of a transaction; **(2)** expending the same resource more than once (i.e., double spending) within a short interval of the time; and, **(3)** tampering with the ledger logs to change the ownership of resources or the amount of resources owned by an entity.

To prevent the aforementioned frauds, Blockchain enforces various in-built validation checks as a part of its core architecture. While each Blockchain variant (e.g., Bitcoin, Ethereum, and IBM Hyperledger) implements its own version of these validation checks, we examine the three checks which are fundamental to almost all architectures. **(1)** First, Blockchain mandates that a transaction submitter (i.e., spender) provide *information* which proves ownership of the resource they want to expend. For example, in the public Bitcoin, the $ScriptSig$ and the $scriptPubKey$ perform this check based on the ID of the spender, ID of the receiver, and the amount to be spent. Thereafter, Blockchain participants *validate* the spender (i.e., if they are authorized to expend) and build a potential block with transactions corresponding to many such spenders. **(2)** Second, the Blockchain participants include *information* which can be used by other participants to verify that the transactions included in a potential block and their order of inclusion are indeed valid. For example, the Merkle-root hash, included in a block of the Bitcoin's Blockchain, serves this purpose. Thereafter, if a majority of the participants agree on the validity of the potential block, the block is added to the Blockchain. **(3)** Third, Blockchains also ensure that the ledger is effectively tamper-resistant i.e., modification of an accepted transaction by a disgruntled participant is nearly impossible. Particularly, when a transaction is validated, the resource being spent is cross-checked with its parent transaction. Similarly, when a block is validated, the entire trail of previous blocks leading to the genesis block is validated implicitly due to hash chaining.

The Blockchain participants have to maintain a copy of the whole Blockchain in order to validate transactions, the cost of which is non-trivial. One of the most space consuming and computationally intensive components of Blockchains is the Merkle-tree, which is used to enable quicker transaction

validation. The size of the Merkle-tree is calculated as twice the size of transaction the hash $\times$ the number of transactions. This results in 128 KB, 35 KB, and 15.6KB per block for Bitcoin, Litecoin, and Ethereum respectively [4]. In terms of CPU, the aggregate number of hashes computed per second on these platforms are 35.5×10^{18}, 317×10^{12}, and 265×10^{12} for Bitcoin, Litecoin, and Ethereum respectively [4]. The storage and computational needs of the Merkle-trees are further exacerbated by the rate of growth of Blockchain over time. For example, one block gets added every $600s$ in Bitcoin and $60s$ in BitcoinPlus [2].

A. Challenges– Devices with limited CPU and Memory

The use of Blockchains in application areas such as IoT, smart-grids, and vehicular networks, is challenging due to the inherent limitations of the computing devices they employ [6]. Typically, these devices have limited memory (about 2KB to 4MB RAM) and possess restrictive computing power (500MHz or lesser); may even be battery-powered [7]. The aforementioned constraints severely limit the use of traditional Blockchains due to the following reasons:

1) Limited Memory: The RAM may not hold the entire Blockchain, resulting in an excessive delay to access the internal or external persistent storage. This will have a serious impact on the time required to perform transaction validation.
2) Limited CPU: The CPU is time-shared between multiple applications including the Blockchain application. Further, since the CPU used is not powerful, the number of cryptographic computations that can be performed per second is limited.
3) Limited Battery: The power consumed by the Blockchain application would result in considerable power drain. An inefficient Blockchain design could drain the battery sooner than expected, forcing re-deployment.

B. Contributions

The primary objective of this work is to adapt Blockchains to cater to applications running on resource constrained devices, in which the traditional Blockchain cannot be directly applied. For this, we introduce alternative Blockchain designs, called ApproxBC, that have *almost* all the benefits of regular Blockchains but with lesser resource requirements. We present two ApproxBC design variants as a part of this article. **(1)** The first uses *Hash-tables* instead of Merkle-trees to store the fingerprint of accepted transactions. This data-structure requires about $4\times$ less space as well as $2\times$ lesser number of hash computations than the Merkle-trees. **(2)** The second design uses a novel data-structure called evidence Bloom Filter *(e-BF* [8]) instead of Merkle-trees. This data-structure requires $8\times$ less space and $2\times$ less number of hash computations compared to a Merkle-tree.

As a side-effect, ApproxBC may not provide high levels of security guaranteed by the traditional Blockchains, but can be configured to provide a level of security that is acceptable for a certain class of applications. Entropy (or randomness) of the information stored in the validation framework (e.g., Merkle-tree) is a measure of its security. An analysis on the loss of security in the two ApproxBC variants is as follows: **(1)** The Hash-table-based ApproxBC is slightly less resilient to transaction and block forgery attacks. However, under normal circumstances, a transaction that is actually valid is always reported as valid, and vice-versa. In comparison to the Merkle-tree-based Blockchain, the Hash-table-based ApproxBC provides $14\times$ less entropy. In actual terms, this amounts to an entropy of 256 bits, which is more than sufficient to ensure security of many applications. **(2)** The e-BF-based ApproxBC uses an approximate transaction validation framework. For instance, in such a framework, a forged transaction (with a valid transaction ID) which is not present in a block may sometimes (less than 4% of the time) get misreported as being present. However, it strictly ensures that a transaction that is present in a block is never flagged as not present by the Blockchain. Note that a successful forgery will require an attacker to create a forged transaction trail by taking advantage of the 4% leeway in every Block, which requires significant effort. Both the ApproxBC designs presented can, however, be customized to achieve an increased level of security as well as accuracy if more storage space is made available.

Finally, we show how ApproxBC can benefit existing Blockchain applications, namely, Vehicle Insurance and Power Utility Management. We also present an analysis of the application areas in which we expect ApproxBC to make a difference.

II. Traditional Blockchain Architectures

In this section, we present a high-level overview of the Blockchain architecture. For illustration, we use the public Bitcoin Blockchain architecture [1] in Figure 1. Blockchains accept transactions which move resources (e.g. crypto-currency) from one electronic address (e.g. Bitcoin wallet) to another. For a transaction to get accepted by the Blockchain, i) the transaction must get accepted into a potential *block* that is being built by a Blockchain participant (refer steps 1 to 4 in Figure 1); and ii) a potential *block* containing the transaction must get accepted into the Blockchain (refer steps 5a to 8 in Figure 1). The steps to achieve this are described next.

We first describe the steps involved in adding a transaction to a block of the Blockchain. Figure 1 shows an example where Bob wants to expend the resources given to it by Alice. Alice's transaction, which transferred the resource to Bob, is called the *parent transaction* and Bob's transaction is called the *spending transaction* as shown in steps 1 and 3. Step 2 of Figure 1 shows how the *scriptSig* script in the spending transaction (Bob) and the *scriptPubKey* in the parent transaction (Alice) are executed one after the other to return a Boolean value representing the success or failure of the validity check. Together, the two complementary scripts check if Bob is authorized to spend the amount of resource which it spends. In step 4, the spending transaction is broadcast to the Blockchain participants who may then decide to include it into a potential block that they create, after verifying its validity by executing the complementary scripts.

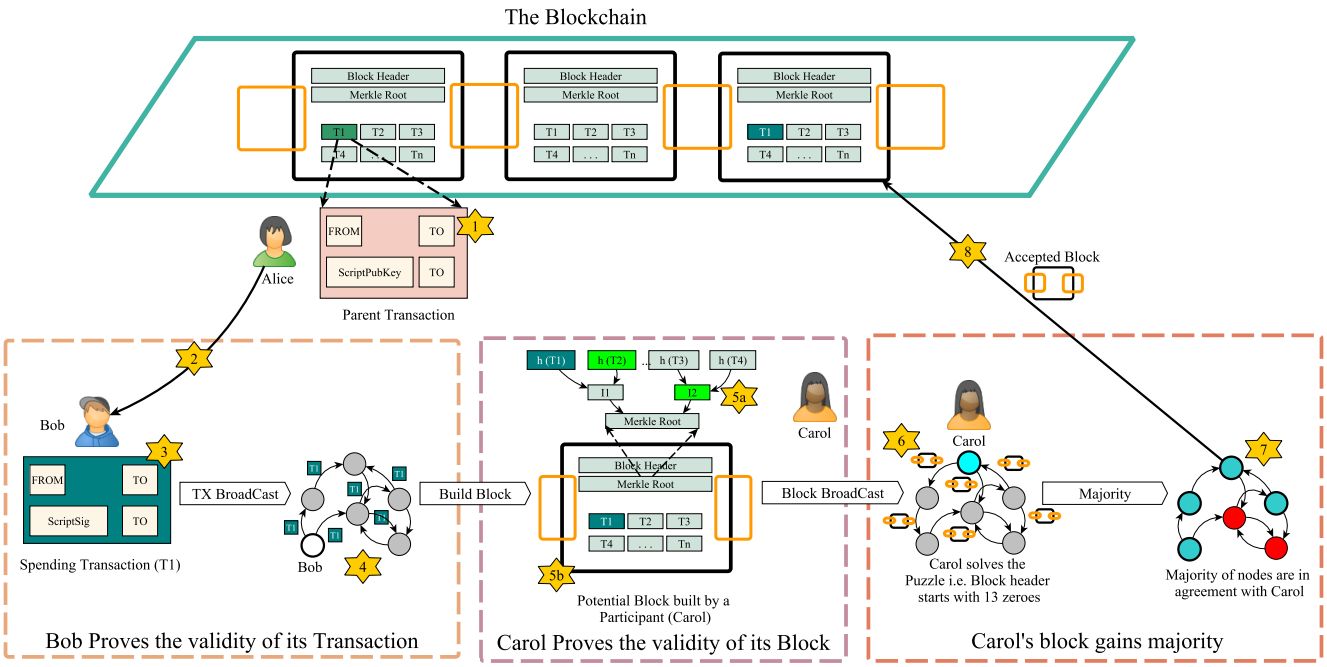

Fig. 1. The steps taken to accept a transaction into the Blockchain. Bob initiates a transaction to spend the resource given by Alice. Carol is an independent Blockchain participant who builds a Blockchain block containing Bob and many others transactions.

We now describe how a potential block is built by a participant and added to the Blockchain. Any interested Blockchain participant (say, Carol) can build a potential block with the set of outstanding transactions from spenders (similar to Bob) as shown in step 5b. A Merkle-tree of the transactions, which is built by Carol in step 5a, is also included as a part of the potential block. The importance of the Merkle-tree will be explained shortly. Similar to Carol, other Blockchain participants may also attempt to build a potential Block from the set of outstanding transactions. If two blocks are broadcasted within a short period of time, each participant may have their own view of the Blockchain depending on which block arrived first. It is this inconsistency that an attacker may exploit to double spend resources. To ensure that all participants see the same ordering of blocks, a consensus mechanism, which introduces a delay between the addition of two consecutive blocks is required. The consensus mechanism lets only one participant succeed in getting its potential block accepted over a period of time. For instance, in Bitcoin, this delay is ensured by requiring the participants to solve a hard puzzle i.e., proof-of-work. Hence, this consensus mechanism addresses the double spending problem.

Assuming that Carol wins the consensus, its potential block is added by other participants to their copy of Blockchain as shown in step 6. The other participants accept the block into their Blockchain after validating every transaction in the block as well as the Merkle-tree embedded within the block. Eventually, if a majority of the participants accept the block (shown in step 7 and 8), the block and the transactions in it are said to have been accepted into the Blockchain.

III. Merkle-Trees: Security and Overheads

Merkle-tree is a data-structure, which is maintained by the Blockchain participants to facilitate validation of transactions in logarithmic time. The Blockchain participants build a Merkle-tree with the list of outstanding transactions to be included in a block (refer step 5a of Figure 1). Hashes of all these transactions form the *leaves* of the Merkle-tree. Higher level nodes of the Merkle-tree are composed by repeatedly hashing pairs of nodes from the lower level. For instance, hashes of transactions T1 and T2 (represented by h(T1) and h(T2) in step 5a of Figure 1) are appended and rehashed to form the Merkle-tree vertex I1. The root of the tree, called the Merkle-root, captures both the contents of each of the transactions and the order in which they appear at the leaf of the Merkle-tree.

When a third party (say, Dave) wants to verify if Bob's transaction (say, T1) was included as a part of this block, it performs the following operations. First, it requests from a Blockchain participant, the minimal set of intermediate Merkle-tree vertices which are required to reconstruct the Merkle-root. For example, in step 5a of Figure 1, the two highlighted Merkle-tree nodes, namely, h(T2) and I2, are provided to Dave by the Blockchain participant. Using h(T1) and h(T2), Dave can reconstruct I1. Again, using I1 and I2, Dave can reconstruct the Merkle Root. Finally, Dave fetches the reference Merkle Root for the block from a *trusted* Blockchain participant and ensures that the reconstructed Merkle-Root matches it. If this comparison succeeds, the copy of the transaction T1 that Dave has is indeed the copy that was accepted in the Blockchain. If not, then either the transaction T1 that Dave has is forged, or the Blockchain participant who provided the Merkle-tree is not trustworthy.

Threat Model: We consider the following attack scenarios: **(1)** *Transaction forgery*, where Bob gives Dave a fake transaction which promises transfer of some resources. When Dave validates the transaction, it rebuilds the Merkle-root and compares it to the Merkle-root from a trusted participant. The validation check will pass if and only if Bob managed to find a hash collision corresponding to at least one transaction in the given block. **(2)** *Merkle-tree forgery*, where Carol and a small swarm of disgruntled Blockchain participants attempt to create a fake Merkle-tree. The fake Merkle-tree contains fake hashes corresponding to forged transactions, subject to the constraint that the Merkle-root is the same as the one maintained by the trusted Blockchain participants. During transaction validation, if Dave consults one of the disgruntled participants, the intermediate hashes from the fake Merkle-tree are given to it. The Merkle-root built by Dave will match the Merkle-root from the trusted source, thereby accepting the fake transaction.

Security: We now estimate the effort involved in performing the aforementioned attacks. For a Blockchain implementation which uses 256 bit hashes, the entropy involved in finding a hash collision for a transaction is 256 bits, which would take 13.7×10^9 years [11]. On the other hand, building a fake Merkle-tree which can allow one fake transaction to pass involves forging 14 intermediate Merkle-tree hashes, the entropy for which is 14×256 bits.

Storage and Computation Overhead: We present an analysis based on the public Bitcoin, in which the block size is currently limited to 1MB, which in-turn can contain $5,000$ transactions (each of length 200 Bytes). The proof of existence of these transactions in a block is maintained at the Blockchain participants using *Merkle-tree* [14]. The Merkle-tree constructed for $5,000$ transactions will have about $10,000$ tree vertices in total. This would consume 320 KB of space and requires $10,000$ SHA-256 hashes to be computed per block of the Blockchain. Further, every time a spending transaction needs to be validated, the Blockchain participants need to perform $\log_2(10,000)$ (≈ 14) memory accesses.

IV. Approximate Blockchain Architecture

In this section, we look at alternative Blockchain architectures, which are composed by replacing certain computation intensive and space consuming components with approximate data-structures and the corresponding computation methods. Specifically, we explore alternatives to the Merkle-trees that are at the heart of the traditional Blockchain architectures. We can abstract the functionality of Merkle-trees as an *oracle* maintained at Blockchain participants, that can be queried by the transaction verifier. This oracle must be capable of answering if a given transaction is accepted by the given block, with sufficient proof. Also, this scheme should not require information about any other transaction in the block, to ensure privacy of other spenders that share this block. Any data-structure that can function as a privacy-preserving oracle can possibly replace Merkle-trees.

In this work, we propose two alternative Blockchain designs that are based on **(1)** Hash-tables, and **(2)** evidence bloom filter (e-BF), which is a new data-structure we proposed in a recent work [8].

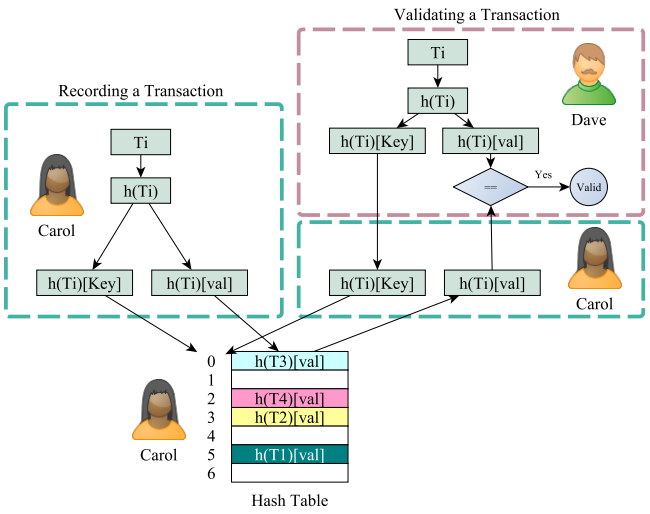

Fig. 2. Figure describing the use of Hash-tables to store and validate transactions. Carol is the block creator who builds the Hash-table. Steps followed by the transaction verifier (Dave) is also illustrated.

A. Hash-table-based Approximate Blockchain

Hash-tables have been used in a variety of applications such as caches and message digest repositories, that require privacy as well as a high level of accuracy. The use of Hash-table as an oracle is promising since it can help the participants answer if a transaction was accepted or not with sufficient proof; much like Merkle-trees. However, the working of the two data-structures are quite different. Figure 2 describes how transactions are stored in the Hash-table by a participant (say, Carol) who builds the block. The transaction Ti is first hashed (to say, 256 bits) and split into the key (represented by 128 bit h(Ti)[Key]) and value (represented by 128 bit h(Ti)[val]). The key is used as the index into the Hash-table where the value is stored. For example, if h(T4)[Key] is 3, then h(T4)[val] is stored at index 3.

When a Blockchain user (say, Dave) wants to verify if the copy of transaction T4 that was given to him by someone (say, Mallory) is valid, he first calculates h(T4) and splits them into key and value as shown in Figure 2. This value, called the *reference value*, is used to cross-check the value retrieved from the hash-table. To know if T4 is a part of the accepted block, Dave challenges Carol to produce h(T4)[val] by providing h(T4)[key]. Finally, Dave considers T4 to be valid if and only if the h(T4)[val] provided by Carol matches the reference value from the first step. However, a down-side is that two transactions that are interdependent cannot be included in the same block, since their ordering cannot be ensured.

Security: We assume a scenario where Mallory forges a fake transaction Ti' and gives it to Dave. We quantify the effort required to get Ti' accepted as a valid transaction in a particular block. This requires creating a Ti' such that it results in a key-value pair which is already present in the hash-table. Since this requires finding a 256 bit hash collision, the entropy

	Data-structure Size		Transaction Verification Effort			Security
	Size (KB)	No. of Hashes	Mem. Lookups at Participant	Communication Overhead (bits)	No. Hashes at verifier	Entropy (bits)
Merkle-Tree	320	10K	14	14x256	14	256
Hash Table	80	5K	1	128+128**	1	256
e-BF	38.5	5K	K* = 1 or 2	128+128**	1	<256

* K is the e-BF replication factor
**128 bit challenge and 128 bit response
***Values presented are for public Bitcoin Implementation of Blockchains

TABLE I
TABLE COMPARING THE THREE ORACLES, NAMELY, THE MERKLE-TREE, E-BF, AND HASH-TABLES ON THE BASIS OF STORAGE SPACE, COMPUTATIONAL EFFICIENCY AND COMMUNICATION OVERHEADS.

of this scheme is 256 bits; quite similar to that of the Merkle-trees.

Storage and Computational Overheads: For storing 5000 Bitcoin transactions in a hash-table, the number of hash computations is 5000, as summarized in Table I. Since the keys are not stored, the storage space required is only 128 bits $\times 5000 = 80$ KB. During transaction validation, one hash computation has to be performed to generate the key-value pair at Dave, and exactly one memory look-up to retrieve the value from Carol's hash-table. Also, since Dave sends the key to Carol, and Carol responds with the value, the communication overhead is 256 bits per validation.

B. e-BF-based Approximate Blockchain

In this section, we present an oracle which is more efficient than hash-tables. e-BF [8] is a data-structure that combines the space and computational efficiency of Bloom filters [13] and the usability of hash-tables (i.e., the ability to retrieve values). The e-BF is a key-value store which is represented as an array of cells, where each cell is of a fixed length. Figure 3 shows an e-BF which has 8 cells and stores the 128 bit values corresponding to transactions T1, T2, T3, and T4. Each transaction is hashed and split into a key-value pair, as in case of hash-table-based ApproxBC. The key is used to index into the cell, where the value is stored. Similarly, the value corresponding to a key can be retrieved by indexing into the appropriate cell. In order to save storage space, each cell acts as an accumulator, where the values corresponding to multiple transactions are stored. Accumulation of multiple transactions hashes in a cell may result in certain bits of the previously stored transaction hash getting over-written. This is the root-cause of the approximation. However, the usefulness of the values stored are improved with the help of a novel encoding scheme that minimizes the probability of overwriting, and a novel cell-layout that allows overlapping of adjacent cells in the e-BF. Further, the e-BF design allows a value to be stored in multiple cells (k in number), based on indices derived from key, to ensure redundancy. Additionally, e-BF can also quantify the level of approximation on the retrieved values. These features make e-BF suitable for a wide range of applications that are approximation-tolerant, provided there exists a mechanism to quantify the extent of this

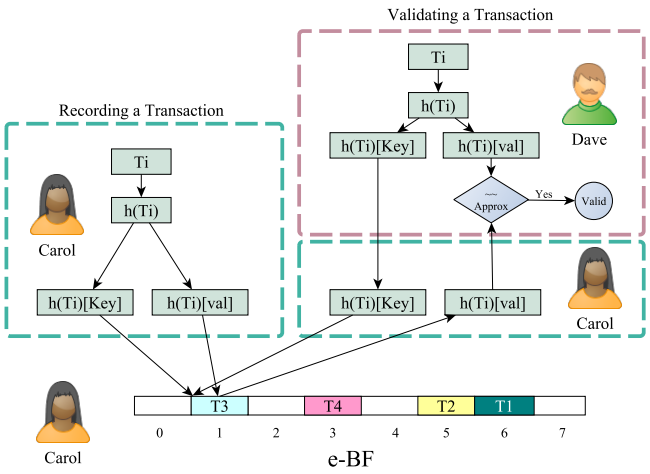

Fig. 3. Figure describing the use of e-BF with 8 cells to store and validate transactions is shown. e-BF is built by Carol who builds the Blockchain block. The transaction validator (Dave) queries Carol to provide proof of validity of a transaction.

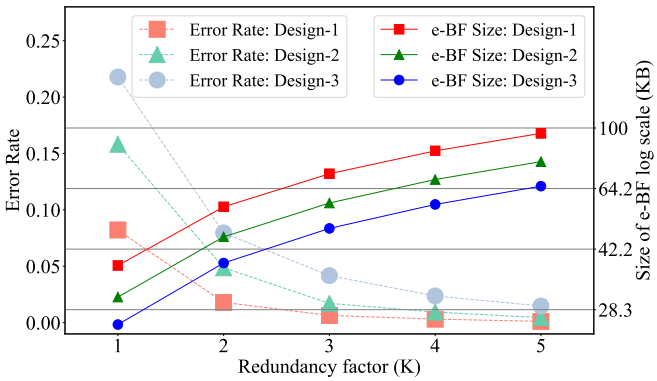

Fig. 4. Figure describing the error rate and size of e-BF corresponding to different values of redundancy (K). Three e-BF designs are shown.

approximation. A detailed description of e-BF can be found in the full paper [8].

Figure 3 describes the scheme used to store and validate transactions when using e-BF. Similar to hash-tables, the scheme requires the verifying party (Dave) to first generate a key-value pair from the transaction given to it by Mallory, then querying the Blockchain participant (Carol) with the key, and finally matching the response from Carol with the reference response. However, contrary to hash-tables wherein the key points to the hash-table entry, the key in e-BF points to the bit-offset from which the value can be stored/retrieved. It must be noted that the value provided by the e-BF may not match the reference value exactly due to the approximations. However, the e-BF can quantify the extent of approximation of the retrieved evidence by computing its hamming distance from the reference value. This can, in turn, be used to quantify Dave's confidence in the validity of the transaction.

Security: We assume a scenario where Mallory forges a fake transaction such that the e-BF flags it as valid with a high degree of confidence. Such an attack is slightly easier (about 4% easier) to perform, when compared to the hash-table-based

ApproxBC. In absolute terms, the entropy of e-BF, which is a measure of the randomness measured in bits, is 256 bits (minus the entropy loss due to approximation).

Storage and Computational Overhead: For storing 5000 Bitcoin transactions, the number of hash computations is 5000, but requires only 38.5 KB of e-BF[2] storage (determined using empirical analysis), as summarized in Table I. During validation, the communication overhead and the no. of hashes computed are the same as the hash-table-based ApproxBC. However, e-BF may require more memory accesses depending on the value of k, which dictates how many copies of value need to be retrieved.

Accuracy vs Storage Space Trade-off: The e-BF is capable of offering better transaction validation accuracy if more space is allocated. Figure 4 showcases the trade-off between the error rate and the size of three e-BF variants (Design-1, Design-2, and Design-3) obtained by varying the overlap between the e-BF cells. We can observe that for higher redundancy (K) values, the space required by e-BF is high, and consequently the error rate is low. There exists an optimal point ($K = 2$) at which the storage required is not too high, and the error rate is reasonable.

V. Approximation-Tolerant Applications

In this section, we showcase two different applications that use the approxBC architecture to record critical events. The need for Blockchains and the effect of approximation on each application is analyzed.

A. Vehicle Insurance

Vehicle insurance is an insurance for motorized vehicles that offers financial cover against accidental collisions, theft, keying, and natural disasters. The organizations that provide Vehicle insurance, called the insurance providers, are prone to attempts by disgruntled policy holders who may misconstrue the circumstances involving an accident, or falsify documentation in order to claim insurance. Such attempts to defraud the insurance providers are significant in number, and cost them billions of dollars. Conventionally, insurance providers have resorted to extensive forensic audit of a few high profile claims. However, most of the other claims do not undergo forensic audits due to the prohibitively high cost involved. Currently, researchers are focusing on using IoT in conjunction with Blockchain to record critical events into the Blockchain [9]. For instance, critical events such as application of brakes, crossing the speed limit of 60 Kmph, and the status of the car's headlight can be captured by an IoT device, which can then be recorded in a Blockchain. Figure 5 describes one such scenario where a car driver who does not stop at the red signal, and ends up colliding with a van. ApproxBC records all the critical events, thereby catching the eliminating attempts to misconstrue events. The tamper-resistance of Blockchains could help the insurance providers determine the circumstances surrounding the accident, based on which insurance payouts can be made.

[2]e-BF [8] with parameters $k = 2, g = 0.25, m = 128, p = 5000$ was used

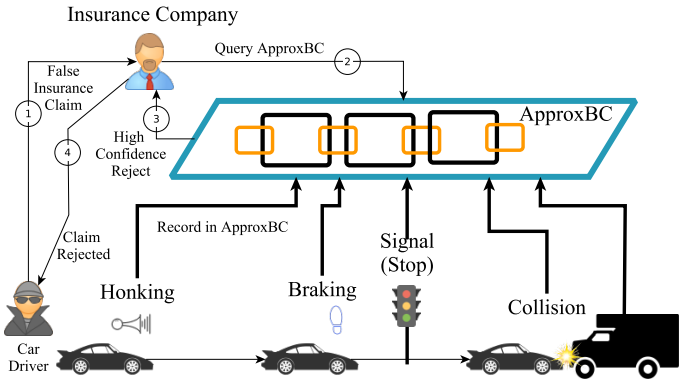

Fig. 5. Figure describing a disgruntled insurance claimant who claims insurance despite violating traffic rules. ApproxBC is able to identify this using the critical event generated by the stop traffic signal.

While this solution works on paper, most IoT devices cannot support the storage and computational demands of traditional Blockchains. While prior works [6] have used a cloud-based storage for recording IoT events, such a solution may not work when connectivity is an issue. In this context, we explore the possibility of using ApproxBC in place of the traditional Blockchains. ApproxBC records all the critical events in the Blockchain. When queried for the occurrence of a particular event, ApproxBC responds with an answer and additionally quantifies its confidence in the response. If the response has a high level of confidence, the insurance payouts can be made. However, if the response has a low confidence, further forensic audit has to be performed, in order to get a definitive answer. Therefore, ApproxBC-based Blockchain can be helpful in performing preliminary analysis to eliminate most cases of insurance frauds. Based on our analysis, ApproxBC cannot answer conclusively for a meager $3\% - 4\%$ of the claims, which we believe is a significant result.

B. Power Utility Management

The developing regions of the world and even some developed countries suffer from the problem of power-theft, which involves illegally tapping power from a power-line. For instance, the revenue loss due to power-theft in the world is estimated at US $89.3 billion annually [12]. To solve this, many countries have installed smart-meters at the end-users as well as the power distribution grids. Since the smart-meters are hosted at the premises of the end-users, trust is still an issue. For this reason, recent research works [3] have focused on employing Blockchains to record critical events related to power consumption. However, the storage and computational requirements to maintain such a Blockchain is still huge, restricting its application.

We propose ApproxBC as an alternative to address this problem. ApproxBC is capable of recording all the critical events. However, sometimes, due to its approximate nature, ApproxBC may not have a high confidence in a critical power event which happened in reality. In such cases, the end-user may claim that the power event did not happen in the first place, resulting in non-payment of bills corresponding to that

particular event. By design, the ApproxBC ensures that such discrepancies do not occur more than $3\% - 4\%$ of the time. If the power event happens to be a critical one (in terms of revenue), a forensic audit may reveal the facts, based on which payouts can be made. However, if the power utility company were to commit more storage and computation, ApproxBC can achieve better guarantees, thereby reducing the need for forensic audits.

C. High Impact Areas

ApproxBC has some desirable properties that the traditional Blockchain platforms lack, which make it suitable for a certain class of applications. The following are the conditions under which ApproxBC may be useful.

- Is there a critical event that needs to be recorded persistently?
- Is this critical event available digitally?
- Do you want to capture the trail of prior events which caused the critical event to happen?
- Are the participants untrusted? Can they tamper with the recorded critical events?
- Is there a fall-back technique such as forensic audit to verify the occurrence of a critical event?
- Is the fall-back technique costly to perform?

If the answers to most of the aforementioned questions are *yes*, then ApproxBC could be a good fit. While ApproxBC may not offer the high levels of security guarantees offered by the traditional Blockchain, it can help many organizations reduce the operational costs involved in conducting extensive audits.

VI. Conclusion

In this article, we proposed ApproxBC, an alternative Blockchain architecture, that can be supported by devices that have limited space and computational resources. This architecture can be leveraged by applications that can accommodate a certain level of approximation in their transaction validation framework, but still require good-enough provable security guarantees. To meet these requirements, we replace the storage and computation-heavy Merkle-tree-based in traditional Blockchains with two approximate data-structures: Hash-tables and e-BF. The approximate data-structures can also be configured to provide a given level of accuracy required by the application. We have observed that the approximate Blockchain has the potential to reduce the storage and computational power required to store transaction logs by $8\times$ and $2\times$ respectively. Finally, we also demonstrate the usability of the approximate Blockchain by building two sample applications and analyze the impact of approximation on their correctness. We believe that there is scope for exploring other alternative Blockchain design choices which can provide approximate guarantees, and a broad class of applications that run on devices with constrained resources which can benefit from them.

References

[1] "Bitcoin Blockchain Architecture." [Online]. Available: https://bitcoin.org

[2] "Block Size And Transactions Per Second ." [Online]. Available: https://www.bitcoinplus.org/blog/block-size-and-transactions-second

[3] "Blockchain for Power Utilities: A View on Capabilities and Adoption." [Online]. Available: https://www.cognizant.com/whitepapers/blockchain-for-power-utilities-a-view-on-capabilities-and-adoption-codex3372.pdf

[4] "Cryptocurrency statistics." [Online]. Available: https://bitinfocharts.com/

[5] "Explained: Punjab National Banks 1.8 Billion Fraud, howpublished = https://thewire.in/224314/explained-punjab-national-banks-1-8-billion-fraud/, note = Accessed: 2018-02-28."

[6] "IBM HyperLedger for IoT." [Online]. Available: https://www.ibm.com/internet-of-things/spotlight/blockchain

[7] "Memory Options for the IoT." [Online]. Available: https://www.synopsys.com/designware-ip/technical-bulletin/memory-options.html

[8] "Promises, Lies and Measuring Tape: Assessing Service Quality in an Untrusted Network Service Function Chain," http://www.cse.iitm.ac.in/~pkarthik/e-BF.pdf, accessed: 2018-02-28.

[9] "The Blockchain Imperative: The Next Challenge for P&C Carriers." [Online]. Available: {https://www.cognizant.com/whitepapers/the-blockchain-imperative-the-next-challenge-for-p-and-c-carriers-codex2360.pdf},date={2018-02-28},

[10] "Top Five Blockchain 3.0 To Watch Out For In 2018 ." [Online]. Available: https://coinsutra.com/3rd-generation-blockchain/

[11] "Why haven't any SHA-256 collisions been found yet?" [Online]. Available: https://crypto.stackexchange.com/questions/47809/why-havent-any-sha-256-collisions-been-found-yet

[12] "World Loses $89.3 Billion to Electricity Theft Annually, $58.7 Billion in Emerging Markets." [Online]. Available: https://www.prnewswire.com/news-releases/world-loses-893-billion-to-electricity-theft-annually-587-billion-in-emerging-markets-html

[13] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[14] R. C. Merkle, "Method of providing digital signatures," 1979. [Online]. Available: https://patents.google.com/patent/US4309569