Sketches for Blockchains Systems

Ori Rottenstreich
Technion - Israel Institute of Technology

March 1, 2021

Grant 2029201 - Progress Report

I am glad to share that a conference paper that summarizes our first results in the project named '*Sketches for Blockchains*' has appeared in the 2021 International Conference on Communication Systems and Networks (COMSNETS), January 2021 (Virtual Conference).

The conference paper is attached to the report and can also be found at https://ieeexplore.ieee.org/document/9352944. We plan to continue the research on the project in the upcoming months.

I would like to thank the Polak Fund for Applied Research at the Technion for the generous support.

Sketches for Blockchains

Ori Rottenstreich
Technion

***Abstract*—Blockchains suffer from a critical scalability problem where traditionally each network node maintains all network state, including records since the establishment of the blockchain. Sketches are popular hash-based data structures used to represent a large amount of data while supporting particular queries such as on set membership, cardinality estimation and identification of large elements. Often, to achieve time and memory savings, sketches allow potential inaccuracies in answers to the queries. The design of popular blockchain networks such as Bitcoin and Ethereum makes use of sketches for various tasks such as summarization of transaction blocks or declaring the interests of light nodes. Similarly, they seem natural to deal with the size of the state in blockchains. In this paper, we study existing and potential future applications of sketches in blockchains. We first summarize current blockchain use cases of sketches. Likewise, we explore how this coupling can be generalized to a wider range of sketches and additional functionalities. In particular, we explain how sketches can detect anomalies based on efficient an summary of the state or traffic.**

I. Introduction

The blockchain technology recently received huge public attention as a distributed secure ledger for cryptocurrencies with no central authority behind it. It became popular as the technology behind Bitcoin [1], a trust-free cryptocurrency, and since then has been expanded to other areas of applications such as supply chain communications, online voting, advertising and medical informatics.

A blockchain is a decentralized ledger, structured as an immutable chain of data blocks. Starting from the first block, the *genesis block*, each block includes the output of a cryptographic hash function computed over the content of the previous block, making it impossible to alter a block without changing all subsequent blocks. Blocks are appended to the chain and become available for read-only and (typically) cannot be deleted or modified. Blocks are lists of transactions, ledger updates, that are bundled together due to the high cost of consensus protocols and the block meta-data overhead.

Sketches often represent data structures and algorithms used for summarizing massive data sets while enabling answering queries on the data [2]. Typically the sketch allocated memory is much smaller than the memory of the original data (e.g., logarithmic in its size). Accordingly, an error is permitted in the query answers and a tradeoff exists between the accuracy and memory efficiency. A sketch is designed for answering a particular query or a range of them.

Sketches can be a natural fit to support an increasing state size and transaction rate in a blockchain network. For instance, the processing of a new proposed block requires transaction validation based on the network state as implied by previous blocks. While it can be impractical to go over all previous blocks in each block validation, a sketch can be used as a summary for the historical data or as an indication for particular that are required. Similarly, sketches can summarize network state so that it is easy to verify that particular information is part of it. In this paper we explore *applications of sketches for blockchains*. We study various existing applications implementing different functionalities in multiple blockchains. Likewise, we describe additional blockchain applications in which we envision sketches can be useful in the near future. Our focus is on anomaly detection for which we are not aware of existing sketch-based solutions.

Paper organization: In Section II we overview counting tasks and existing sketches that solve them. We begin with the relatively simple task of set representation and continue with more advanced tasks. Then, in Section III we overview applications in blockchain of the mentioned sketches. In Section IV we discuss potential advanced future applications of sketches in blockchain, focusing on anomaly detection. We also refer to recent advances in the field of sketches and describe how they be used to enhance their blockchain applicability. Conclusions can be found in Section V.

II. Functionality and Counting Tasks

In this section we describe fundamental counting tasks and their solution bases on sketches. Later in the paper, we explain their various use cases in existing blockchain implementations. We start with the simple task of set representation and present sketch data structures that refer to sets while supporting various functionality. We then generalize the discussion to more advanced counting tasks and overview existing solutions to each of them.

A. Set Representation

Definition 1 (Set Representation)**.** *For a set S be a set. Representation of S as a sketch can support one or more of the following tasks:*

- *Insertion (addition) of an element x to S*
- *Deletion (removal) of an element $x \in S$*
- *Answering for an element x a membership query of the form $x \in S$?*
- *Listing the elements in S*
- *Proving to a sketch holder for an element x that $x \in S$*

We overview existing sketch data structures for set representation and refer to the various tasks they support.

Bloom filter. The Bloom filter [3] is a popular data structure widely used in many networking device algorithms [4, 5], in fields as diverse as packet classification, routing, filtering, caching, and accounting, as well as beyond

networking in areas like verification and spell checking. The Bloom filter is used for set representation, supporting element insertion and answering membership queries. There are two kinds of errors in membership queries: a false positive (when an element $x \notin S$ is reported as a member of a represented set S) and a false negative (when an element $x \in S$ is reported as a nonmember of S).

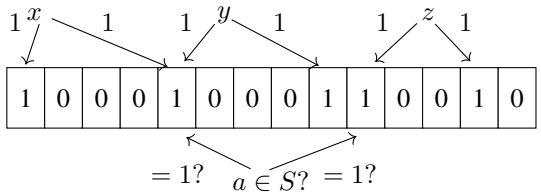
The Bloom filter encounters false positives and has no false negatives. It is built as an array of bits, where hash functions are used to map elements to locations in the array. With initial values of zero bits, the elements of S are first inserted to the filter, setting all bits pointed by the hash functions. Upon a query, the bits mapped by the queried element are examined and a positive answer is returned only when the bits are all set. The Bloom filter is illustrated in Fig. 1(a).

The probability for a false positive error (ratio of non-member elements reported as members) decreases when more memory is allocated for the data structure and increases when a larger set S is represented. Let m describe the size in bits of the Bloom filter, k be its number of hash functions and n be the size of the represented set S. A false positive occurs when a non-member element maps to k bits which are all 1 due to other elements. The false positive probability is $\left(1-(1-\frac{1}{m})^{n\cdot k}\right)^k$ such that $(1-\frac{1}{m})^{n\cdot k}$ is the probability of a bit to have a value of 0 after the insertion of the n elements, each mapped to k bits.

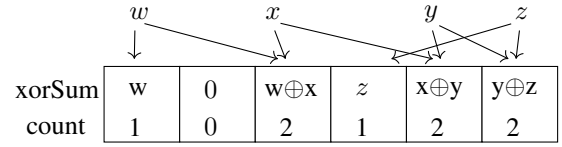
On one hand, increasing the number of hash functions allows checking more bits in the array but on the other hand, each inserted element sets more bits to have the value of 1. The optimal number of hash function is given by $k \approx \frac{m}{n} \cdot \ln 2$. This implies a probability of 0.5 for each bit to have the value of 1 following the insertion of $|S| = n$ elements. Accordingly, the false positive probability is given by $0.5^{\frac{m}{n}\cdot \ln 2} \approx (0.6185)^{m/n}$.

The Bloom filter cannot support removal (deletions) of elements from the represented set. This is since resetting bits to 0s results in forbidden false negatives of all remaining elements that map to these bits. The Counting Bloom filter (CBF) generalizes the basic Bloom filter with the support of element deletions [6]. This is a crucial functionality when the represented set is dynamic and its size can decrease over time. The CBF maintains an array of counters instead of an array of bits. Upon an element insertion, counters mapped by the hash functions are increased by one and these counters are decreased upon an element deletion. Upon a query, the same counters are examined. In case they are all non-zero a positive answer is returned for the query. In the CBF, besides the fact whether the counter is positive or not, the exact value of the counter is not considered during the query process.

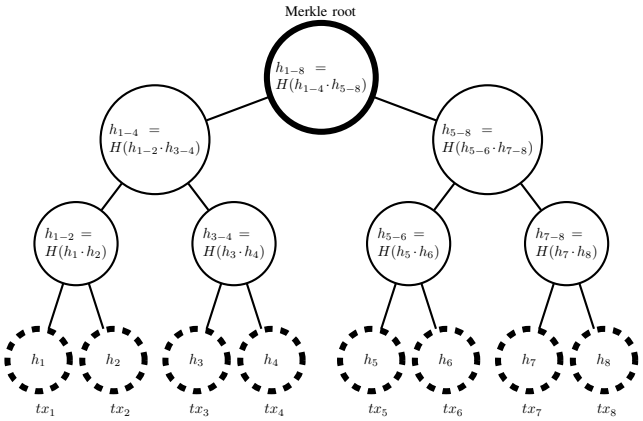
Another alternative to the Bloom filter is the Cuckoo filter [7, 8] in which an element is represented by a signature. Cuckoo filter answers in constant time membership queries and supports deletions. The filter is implemented as an array that can store multiple signatures in each of its buckets. Based on hash functions, Cuckoo filter derives two potential buckets



(a) Bloom filter, supporting membership queries



(b) Invertible Bloom filter (IBLT), allowing listing of the elements



(c) Merkle tree, enabling proofs on element membership

Fig. 1: Illustration of popular sketch data structures for set representation: The Bloom filter, the Invertible Bloom filter (IBLT) and the Merkle tree. They support various queries and operations.

for an element and the element can be stored in one of them. An element is identified as a set member if its signature can be found in one of its two buckets. If in the insertion both candidates are occupied, the Cuckoo filter randomly kicks out a fingerprint in one of the two buckets and reinserts the victim in its other candidate bucket. This reallocation ends successfully when a sampled bucket has available space and fails when the number of such reallocations reaches a given threshold max. The false positive probability increases with the bucket size and reduces for longer fingerprints.

Invertible Bloom filter Table (IBLT). In many data structures, the use of hashing eliminates the option to recover the identity of the represented elements from the data structure. The invertible Bloom lookup table (IBLT) generalizes Bloom filter functionality by supporting not only element insertion, deletion and membership queries but also a complete listing of the elements with high probability [9]. The listing property is probabilistic. An IBLT is illustrated in Fig. 1(b).

Assume that each element is associated with a (unique) key. The IBLT keeps an array of cells, each with two fields:

(i) xorSum field - stores the cumulative XOR of the keys mapped to the cell; *(ii)* count field - stores the number of entries mapped to the cell. We refer to a pure cell as a cell with a single inserted element. Such a cell can be identified based on the count field. The listing procedure of the IBLT repeatedly looks for a pure cell. Whenever such a cell is found, an element is identified based on the keySum field which is necessarily that of the element. Then, we can remove the element from other cells it is hashed to, thus reducing the number of remaining elements in these cells. The process fails if no pure cells can be found in some step, earlier to the listing of all elements. Its success probability is affected by the number of inserted elements and the number of cells.

Merkle Tree. The Merkle tree is a known tool in cryptography, first suggested by Merkle [10] which enables efficiently proving membership of a data element in a set, without revealing the entire set. In a Merkle tree, every node has a *Merkle label*. For the leaves, this label is the hash of a data block, and for every non-leaf node this label is the hash of the concatenation of the labels of its children (or the label of its child in case it only has one child). The Merkle tree is illustrated in Fig. 1(c).

In order to verify that some data is included in a Merkle tree T, one needs to obtain from a trusted source a label $M(T)$ of the root of the tree, called the Merkle root. A Merkle proof for the containment of some data v, which corresponds to a leaf in the tree, consists of the sibling path of the leaf, which contains the labels of all the siblings of the nodes in a path from the leaf to the root. This can show that the known Merkle root $M(T)$ was computed for a set of data elements with v among them. We assume Merkle trees are second preimage resistant, making it impossible to reproduce a Merkle root label. With the labels along the path, the verifier can compute the label of the Merkle root and check that it is indeed equal to $M(T)$.

B. Set Synchronization

Definition 2 (Set Synchronization)**.** *Let A, B be two hosts with corresponding sets S_A, S_B, both taken from some universe U. The symmetrical difference of the two sets is $\Delta = (A \setminus B) \cup (B \setminus A)$. The goal is to compute the union of the two sets $A \cup B$ through deriving elements in the symmetrical difference. Often, the communication complexity of the synchronization process is of concern.*

A naive solution is through letting each node expresses its set. This clearly takes communication complexity of $O(|A| + |B|)$. Interestingly, there are algorithms allowing to perform set synchronization with communication complexity of $O(|\Delta|) = O(|A \setminus B| + |B \setminus A|)$.

Guo and Li suggested a scheme based on counting Bloom Filters (CBFs) [11]. Each host simply represents its set by a CBF (with common parameters) and transmits the filter to the other host. Then a host computes the difference of its own filter and the received filter by computing the difference of counters in identical locations of the filters. The set difference is estimated as those elements mapped to non-zero counters in the computed difference. An outlier counter is defined as having a value of zero in the difference although different elements contributed to the identical non-zero values in the two filters. The scheme suffers from both kinds of potential errors: *(i)* false positives - elements identified as belonging to the set difference Δ although they appear in both sets or in none of them. False positives occur when an element maps to some counters with non-zero values due to other elements in Δ. and *(ii)* false negatives - elements in the set difference Δ that are not identified as such. False negatives occur due to outlier counters. The probability for each kind of error reduces when more memory is allocated for the filters.

C. Computing Set Similarity

Definition 3 (Set Similarity)**.** *Let A, B be two hosts with corresponding sets S_A, S_B, both taken from some universe U. The goal is to compute the Jaccard index of the two sets which is defined as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.*

Note that the Jaccard similarity of two sets equals 0 when they are disjoint and 1 when they are equal. It has a value between 0 and 1 otherwise.

The MinHash scheme (also known as the min-wise independent permutations locality sensitive hashing scheme) was suggested by Broder in 1997 [12]. It was initially used in the AltaVista search engine to detect duplicate web pages and eliminate them from search results.

The MinHash scheme works as follows. Consider a hash function h applied on set elements. Let $h_{min}(A) = min(\{h(x)|x \in A\})$ denote the minimal hash value computed over the elements of the set A. Let $h_{min}(B)$ be the corresponding value for the set B. With high probability the hash function computes distinct values for different elements. Compare the values $h_{min}(A), h_{min}(B)$. If they are equal, the minimal observed hash values appeared for both sets. If the two values differ the minimal value appeared in only one of the sets. Thus the indicator variable $I(h_{min}(A) = h_{min}(B))$ has a value of 1 with probability of the Jaccard similarity $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. Thus the variable is an unbiased estimator for $J(A, B)$. To estimate $J(A, B)$ with a reduced variance, each of the two hosts refers to k hash functions $h_1, \ldots, h_k$ and computes a vector of length k based on its set v_A, v_B with the minimal values for each function. The two vectors are compared index by index. Let $\frac{1}{k} \cdot \sum_{i=1}^{k} I(v_A(i) = v_B(i))$ be the number of ratio of indices where an equality is observed. The value serves as an estimator for the Jaccard similarity J. By Chernoff bound, it can be shown that the expected error is $O(1/\sqrt{k})$.

D. Count (Streaming model)

Sketches are also used to summarize a stream of elements, trying to reveal its essence within small memory. Algorithms have been designed for particular counting tasks, each keeping its particular sketch and updating it upon the arrival of stream items. Finally, some computation is performed

over the sketch data to report an estimated answer to the counting task. While the range of such tasks can be large, we mention two dominant tasks together with corresponding popular solutions. Later, in Section IV we explain *how such sketch-based solutions can be the basis for anomaly detection in a blockchain network*. The problems measure the number of distinct elements in the stream and those elements of the highest number of appearances.

Definition 4 (Count Distinct)**.** *Let $A = a_1, \dots a_n$ be a stream of elements. The count distinct task is to measure the number of unique elements, namely the set size $|\{a|a \in A\}|$.*

Common solutions to the count distinct problem are the LogLog [13] and HyperLogLog [14], [15] algorithms. The algorithms apply a hash function over an element and measure the number of leading zeros in the hash value. Since a repeated element implies the same hash value, the number of distinct elements can be estimated based on properties of such measures.

Definition 5 (Element Frequency Estimation)**.** *Let $A = a_1, \dots a_n$ be a stream of elements. The task includes estimating for an element a its number of appearances in the stream $\sum_i I(a_i = a)$ where I is the indicator function.*

There can be two possible errors in the estimation of element frequency: Overestimation and underestimation. The state-of-the-art data structure for element frequency estimation is the Count-Min Sketch (CM) suggested by Cormode and Muthukrishnan in 2005 [16] that can observe overestimations. The CM relies on a two-dimensional array of counters initialized to values of 0s. A set of k hash functions are used to map an element to k counters, one in each of its rows. Upon the arrival of an element, each of these counters is incremented by one. To estimate the frequency of an element, its k counters are considered and the size is estimated as the minimal value among the k counters. Since multiple elements can contribute to the same counter, the computed value can be larger than the exact one in case other elements contributed to all k counters implying overestimations. On the other hand, the CM completely avoids underestimations. A tradeoff exists between the level of accuracy and the amount of allocated memory such that more allocated memory reduces collisions among elements. Similarly, reducing the number of active elements improves accuracy.

Definition 6 (Heavy Hitters)**.** *Let $A = a_1, \dots a_n$ be a stream of elements. The heavy hitter detection task is to indicate the frequent elements, namely some top-k elements with the highest number of appearances or those elements that hold at least a minimal portion $\phi \in (0, 1)$ of the total traffic.*

The Space Saving algorithm [17] solves the heavy hitters problems by keeping an array of a bounded number of pairs of key (element id) with its number of appearances. If a new element appears in the table, its counter is updated. Otherwise, it replaces the elements with a minimal value in the table and is initialized with a number of appearances based on that of the evicted element. Finally, the largest elements in the table are reported as the estimated most frequent elements.

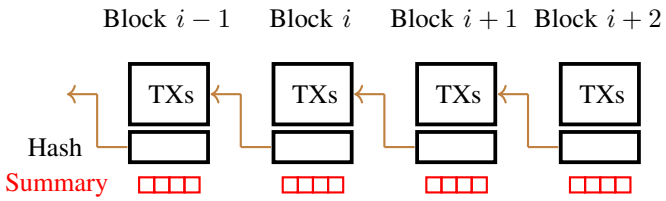


Fig. 2: Block summarization. Each block includes besides the hash of a previous block, a summary of current block often implemented as a Bloom filter or a Merkle tree.

III. Blockchain Applications

In this section we overview existing sketch applications in blockchain networks. The applications rely on the sketches ability to support various functionality such as described earlier in Section II. Table I summarizes such use cases.

Method	Block summary	Light clients	Unique addresses	Pool sync.	Fairness enfor.
Bloom filter	[1, 18, 19]	[20]	[21]	[22]	[23]
Cuckoo filter	-	-	[24]		-
IBLT	-	-	-	[22]	-
Merkle tree	[1]	[20]	-	-	-

TABLE I: Blockchain applications for sketche-based set representations.

A. Block summaries

Summarizing block information can be helpful to indicate whether a particular transaction can be found in a block or the block as a whole can be skipped in a search for a transaction. Typically blocks are associated with Bloom filter for such functionality. Moreover, to verify transaction output its log can be located. Then, the block transactions are reexecuted and their output can be compared with the log. Fig. 2 illustrates block summaries with Bloom filters. The Ethereum block contains a field called logsBloom holding a Bloom filter of 2048 bits (256 bytes) [18]. The filter makes use of $k = 3$ hash functions so that each element is mapped to three bits of the filter. It contains log entries from the receipt of each transaction of the block. The receipt includes information such as cumulative gas used in a block till the transaction and logs created through the execution of the transaction. Each of three bit selections can be described by $\log_2 2048 = 11$ bits and they are based on the low-order 11 bits of each of the first three pairs of bytes in a Keccak-256 hash (proposed SHA-3 implementation by Bertoni et al. [25]) of the inputs. A similar approach was suggested by Rush for a general blockchain [19].

B. Bitcoin Simplified Payment Verification (SPV)

The increasing amount of memory required to maintain the full Bitcoin state (more than 310GB by December 2020) together with rapid growth in the volume of transactions that have to be processed imply a large overhead on full Bitcoin nodes. A light client is a node variant that can verify only part of a block without locally maintaining the complete network state through a method called Simplified Payment Verification (SPV). A light client is connected to a full node. An implementation for the light client to report its set of relevant transactions by representing it as a Bloom filter was described [20].

Among the transactions it receives, a full node only forwards to the light client those transactions that match the filter, potentially with some false positives (namely transactions beyond the interest of the light client). Using the Bloom filter allows the light client not to express in an exact manner the set of addresses he is interested in. Together with the particular transactions of interest, the full node also provides Merkle tree based proofs demonstrating their inclusion in the block. The length of the filter can be selected based on a required false positive probability. Fig. 3 illustrates this process.

The privacy of light clients was studied in [26]. Informally, it can be viewed as the ability to hide addresses in the sets among other addresses that yield false positives. The reported Bloom filter by a node certainly leaks information on the addresses of interest for the light client and one can guess them based on the filter. On the one hand, a low false positive rate reduces privacy by allowing one to guess more accurately the identity of the represented set. On the other hand, higher false positive rate increases the redundant transaction information sent by the full node to the light client. This implies a tradeoff between the privacy and the communication overhead. The privacy can be measured by a metric describing the probability to correctly guess multiple addresses as included in the interest set. Let N be the number of addresses of interest for which the filter was computed and let F be its number of false positives. The probability to guess that a single address with a positive indication belongs to the set is $\frac{N}{N+F}$. For a parameter j, the metric computes this probability for guessing correctly j such transactions given by $P_{h(j)} = \prod_{k=0}^{j-1} \frac{N-k}{N+F-k} = \frac{N}{N+F} \cdot \frac{N-1}{N+F-1} \cdot \ldots$.

Similarly, in [27] a metric called γ-deniability was presented. They refer to a set member $x \in S$ as deniable if for $i \in \{1, \ldots, k\}$ there is a non member y_i such that $H(x) = H(y_i)$. Then, a Bloom filter is γ-deniable if an address is deniable with probability γ. That work indicates that in addition to the false positive rate of the Bloom filter, the privacy is affected by the number of real addresses as well as by the number of those with a false positive. Given a false positive probability, the last is of course determined by the size of address space for the examined addresses. They describe a method for estimating the number of active addresses through a linear regression model. A similar observation about the efficiency of Bloom filters was previously described in a more general context [28].

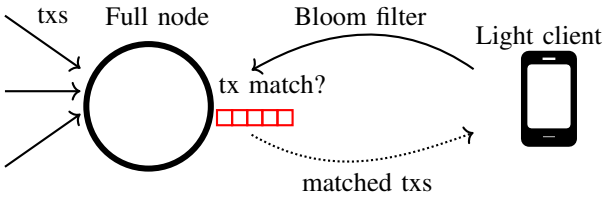


Fig. 3: Bitcoin SPV (Simplified Payment Verification) Clients: Light clients report their set of relevant addresses through a Bloom filter.

Recently, an alternative method for representing light client addresses without Bloom filters was suggested by Qin et al. [29]. The approach leverages Private Information Retrieval (PIR) to provide deterministic client privacy. This comes with modest additional overhead in the required bandwidth it takes the light node to verify a set of transactions.

C. Unique Addresses Checking in DAG-based Blockchains

IOTA is a distributed ledger protocol designed to serve secure communication and payments among Internet of Things (IoT) devices [30]. Its core is a Directed Acyclic Graph (DAG) named Tangle that stores transaction history. IOTA makes use of the Winternitz One-Time Signature (WOTS) scheme for signatures [31, 32]. Each address should be used in only a single transaction. Otherwise, information on a private key associated with the address is observed and can be used by an adversary to forge a signature for the address. Privacy is also weakened with by ability to analyze habits of the participants. Methods for identifying repeating addresses have been suggested through short summaries of previously used addresses. This can be challenging due to the large size of the address space. Wang et al. proposed to do so based on a Bloom filter [21] and Shafeeq et. al [24] based on a Cuckoo filter [7] due to its improved accuracy.

D. Efficient Pool Synchronization and Block Declaration

Blockchain protocols implement set synchronization among nodes for two major objects: *(i)* Pools of pending transaction (each can be viewed as a set) of various nodes so a node can learn about a pending transaction it misses *(ii)* Sharing information of new proposed blocks (earlier as well as following their approval based on the particular consensus protocol) where such a block is viewed as a set of transactions. Network performance is affected by the amount of bandwidth required for these common tasks. It can benefit from not sending a transaction to a node already aware of it or avoiding sending the complete block content to a node familiar with many of the block transactions. An illustration of a set synchronization protocol is shown in Fig. 4.

A node can synchronize its pool of transactions through communicating with the other network nodes. Identifying similar pairs of pools in a blockchain network can be done through the MinHash scheme described in Section II-C.

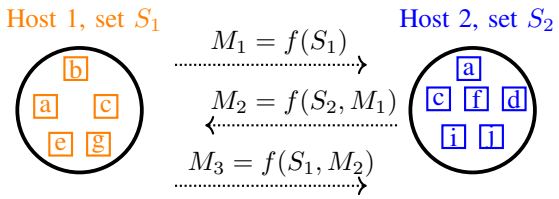


Fig. 4: Set synchronization protocol (illustration) between two hosts with sets S_1, S_2. The required accuracy and the number of transmitted messages vary among protocols. Synchronization is typically implemented between two transaction pools or between a declared block and a local transaction pool.

Consider k pools. A baseline method involves running the MinHash scheme between any pair of nodes so that there are in total $\binom{k}{2}$ computations for the pool similarity between all pairs of pools. Let P_i denote the pool of transactions for node i. A node i identifies a node $j \neq i$ for which the Jaccard similarity $J(P_i, P_j)$ is maximized. Then node i can run a pool synchronization protocol with that node j. Since each node reports the same information to all other nodes, the process can be simplified through reducing the amount of transmitted information. A shared process for which each node reports as multicast the vector of length k computed over its set with the minimal value of its pool following the k hash functions. This again allows each node to find a node with a pool maximizing the similarity to its local pool. We are unaware of an existing approach using this tool as a first step prior to pool synchronization.

Compact blocks [33] is an early approach developed by Bitcoin Core developer Matt Corallo to reduce bandwidth. It takes advantage of the fact that a receiving node has chances to be familiar with some of the block transactions and thus avoids including in the block declaration the complete 256-bit hash information for every transaction. As a side effect, the technique also reduces the block transfer latency. Following a block declaration, a receiving node sends a request for the block information in case it is not familiar with the block. The block issuer then sends only five bytes for each transaction as its identifier together with the complete version of some particular transactions it predicts the receiving node has not obtained yet. Then, the receiving node can send an additional request for the full information of transactions it is still missing.

Graphene [22] also tackles the synchronization problem in block proposal and among pools. It relies on Bloom filter and the invertible Bloom filter (IBLT) [9] (see Section II-A). For block proposals, it achieves asymptotically better performance of reduced bandwidth such that the relative improvement increases for larger blocks. Graphene refers to two protocols: The first protocol applies when transactions of a declared block can all be found in the pool of the receiver. The second refers to a general scenario where the block proposal may include transactions that do not appear in the pool of the receiver. The second protocol starts when the first lighter protocol fails to synchronize. Let n dentoe the block size and m the transaction pool size by the receiver.

The highlights of the first protocol are as follows: A sender node declares on a block and the receiver replies with a request for the block and an indication of its pool size. The sender creates a Bloom filter and an IBLT such that the false positive probability of the Bloom filter is set as $\frac{a}{m-n}$. The parameter a is set such as the IBLT allows with high probability element recovery (listing) when it holds up to a^* elements for some $a^* > a$. In the last step of the protocol, the receiver identifies the transactions from its pool that pass the Bloom filter and computes an IBLT based on them. It computes the difference between the two IBLTs to identify redundant elements and finally tries to decode the IBLT it received using such information. When the second protocol is in action, the subtraction of the two IBLT fails and more iterations among the two players follow. Similarly, for synchronizing transaction pools the protocols are generalized so that both parties finally computes the union of the two pools. The two filters here are computed for the pools rather than a block and based on the symmetric difference of the filters particular transactions are identified as appearing in only one of the pools.

E. Block Selection Fairness

When a blockchain network is shared among multiple applications, they often compete for a higher quality of service affected by the time it takes transactions to be included in a block after their issuance. When one among the nodes proposes a block, it should do that in a fair way considering other applications [34]–[37]. This can be done fairly through a random selection of the block transactions among the set of pending transactions the proposing node is aware of. On the contrary, a dishonest node includes more of its transactions at the expense of those of others.

Orda et al. [23] suggested a mechanism to enhance such fairness through periodical declarations on the transactions a node is aware of. While reporting the complete list of transactions can require high amount of communication, it was suggested to use a Bloom filter for that goal. A declaration is statistically examined by other nodes through checking that many of their pending transactions are included in the declaration. Only nodes with recently approved declarations can propose a block. Then, upon a block proposal, fairness is achieved since it is difficult for a node to ignore transactions of other applications if they were included in one of its earlier declaration. For a node to claim a transaction was not part of an early declaration but appeared to be so due to a Bloom filter false positive, it has to indicate other particular transactions that yield the false positive.

F. Transaction and State Organization with Merkle trees

Bitcoin organizes block transactions in a Merkle tree [10, 38] that its root is included in the block header [1]. As was mentioned in Section III-B, this allows proving (e.g. to light clients) the inclusion of a particular transaction within a block without the complete information of all the block

transactions. On the other hand, it can be difficult in Bitocin to prove to light clients the status of a particular account.

Ethereum makes use of Merkle trees for several purposes:

(*i*) For each block, transactions are organized in a Merkle tree and the transactions root (called transactionsRoot) is included in the block header.

(*ii*) The receipts for the block transactions are also organized in a Merkle tree and the receipts root (called receiptsRoot) is also included in the block header.

Ethereum has two kinds of accounts: Externally owned accounts (EOA) and smart contract accounts. Regular EOA accounts can send tokens to one another. A contract account refers to the deployment of a smart contract. Two main fields in the state of an EOA account are the balance (amount of Ether) and nonce (number of sent transactions or created contracts). Two additional trees are used to organize the Ethereum state:

(*iii*) A tree holds the global network state such as a mapping of addresses.

(*iv*) For each smart contract, a tree is used to organize all data associated with the contract.

Unlike transaction trees, the state trees frequently change, e.g. with any change of an account balance or nonce update as well as with the creation of a new account. In Ethereum trees are implemented as Merkle-Patricia trees.

IV. Vision for blockchain-sketches Engagement

With an eye to the future, in this section we explore potential sketch applications in the blockchain domain. First, we explain how particular sketch data structures and count algorithms can be used for the detection of anomalies of various types. We also discuss recent developments in the field of sketches and detail how they can increase the accuracy and efficiency of blockchain sketch applications.

A. Proposal: Sketches for Network Analysis and Anomaly Detection

Analyzing network behavior can be crucial towards its understanding. Such analysis can include identifying dominant players as well as irregular or suspicious behavior. The analysis can rely on data available within the blockchain as part of block information but also on additional information that can be observed by nodes and is not recorded on the chain. Suggested tools for fraud and anomaly detection in blockchain typically take a different approach and rely on machine learning methods [39]–[44]. For instance, Pham and Lee in [39] used k-means clustering and Support Vector Machine (SVM) to detect suspicious users and transactions in the Bitcoin blockchain network. Ostapowicz and Zbikowski compared capabilities of Random Forests, Support Vector Machines and XGBoost classifiers in the detection of fraudulent accounts in Ethereum [42].

Identifying main players can be based on detecting accounts with a large number of transactions or with high total volume, accounts with a large degree making transactions with many other accounts (potentially representing exchanges). Typically such values are measured in a window of time expressing recent history (rather than since the beginning of network history) and accordingly can be computed based on information of blocks produced within that time range. Consider for instance identifying those accounts with a high amount of traffic. A simple implementation of that measurement would include maintaining a counter per each such address and processing the blockchain while updating counters corresponding to addresses of each transaction. The number of required counters can be very large due to many active addresses although most of such addresses would observe only small traffic volume. Sketches implementing counting tasks naturally support such capabilities with the basic counting tasks detailed in part in Section II-D as building blocks.

As an example, sketch-based solutions for heavy hitter detection (such as the space saving algorithm) can identify in high accuracy the k most active addresses (top k) while maintaining a number of counters that grows linearly with k rather than with the number of active addresses. Similarly, computing the number of unique active addresses or finding an address of a high degree can be implemented with the help of the count distinct algorithms.

For many of the sketches and counting algorithms, computing values for a whole stream of data can be done based on the computations for disjoint subsets of the stream (with the easy to achieve assumption that the separated computations used the same sketch parameters). For instance, a Bloom filter computed for the set of elements in a stream $A = a_1, \dots a_n$ can be computed as the bitwise OR of two Bloom filters (of same lengths and hash functions) computed for two disjoint stream subsets $a_1, \dots a_i$ and $a_{i+1}, \dots a_n$. This property also holds for the computation of the main array in the implementation of the HyperLogLog algorithm [14] for the count distinct problem. Similarly, in the detection of heavy hitters with the Space Saving algorithm [17] we expect dominant values for the whole stream to frequently appear among values computed for the stream subsets.

The will to simplify such advanced computations motivates us to suggest considering including within blocks additional sketches in addition to the Bloom filter which is already available in Ethereum blocks. Since such sketches can be slightly more memory intensive, we can include them periodically once in a multiple blocks so that they serve as an approximated summary of recent blocks since the last block with a corresponding sketch. The allocated memory for a sketch and the frequency it appears can be based on the importance and required accuracy in a particular counting task. Moreover, we can balance this additional memory over multiple blocks. An illustration is shown in Fig. 5 where in addition to the Bloom filter that appears every block the chain includes two additional sketches for particular tasks. The first sketch shown in circles appears once in two blocks and is computed for data of two blocks. The second sketch is shown in diamonds and appears only once in four blocks

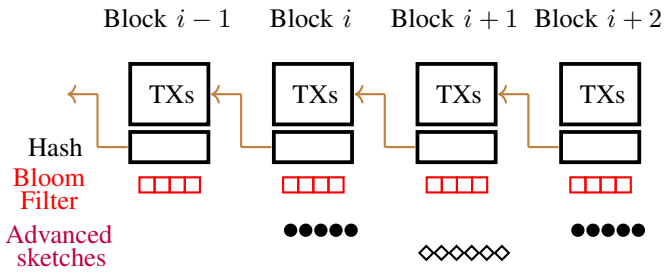


Fig. 5: Advanced Block summarization. In addition to the simple block summary in a Bloom filter, multiple consecutive blocks are summarized by periodic advanced sketches.

since it is more memory intensive or since it is useful for a task where lower accuracy is satisfying.

The occurrence of anomalies expressed as unusual transactions or a special network state can express undesired network phenomena. For instance, repeated time differences between transactions (of the same wallet) can express non-human behavior executed by bots [45]. Likewise, sharp peaks in the transaction volume can reveal DDoS (distributed denial of service) attacks while jumps in the transaction confirmation time can reflect double spending attacks [43].

B. Recent Sketch Advances with Blockchain Applicability

We detail some recent developments in the field of sketches and discuss their potential usecases to improve or extend some of the above-mentioned blockchain applications.

Sketches for multiple tasks - Typically, a sketch serves a particular counting task (such as those presented in Section II-D). UnivMon [46] and Elastic sketch [47] are recent approaches trying to provide stream summaries which are general and answers to *a wide range of counting tasks* can be computed over them. For a stream of elements $A = a_1, \ldots a_n$ let $f_1, \ldots, f_k$ be the element frequencies (number of appearances) where k is the number of distinct elements. Several functions can be computed by summing the results of some function g computed over the frequencies $\sum_{i=1}^{k} g(f_i)$. For instance, the count distinct value is obtained with a function $g(x) = x^0$ and the stream length with $g(x) = x$. UnivMon identifies particular elements with a dominant contribution to the frequency vector as the stream summary and relies on them to estimate with high accuracy multiple counts.

Accordingly, the proposal from Section IV-A to attach blocks periodically with summaries of various sketches for anomaly detection can be enhanced through including a general summary based on such computations that can be helpful in computing multiple counting tasks.

Accurate Bloom filters - A recent approach [48, 49] presented the notion of the Bloom filter false positive free zone which enhances a Bloom filter to completely avoid false positives when the number of set elements taken from a *finite* universe is at most a given bound d. The filter construction is applicable when the universe U is by nature small and makes use of simple carefully-designed hash functions while relying on prime numbers and their properties. The insertion and lookup operations are similar to those in the original Bloom filter. An m bit array is composed of segments, each in the length of a prime number. Functions map an element (exactly) once to each of the segments. Each function simply calculates the modulo of an element with the prime of each segment. For a finite universe, the required filter length to completely avoid false positives is affected by the universe size and the maximal set size that still avoids false positives. Namely, a limited size filter implies a tradeoff and can describe without false positives larger sets of elements from a small universe or alternatively smaller sets selected from a large universe.

This approach can be used for accurate block summaries such that false positives are avoided and in order to find a particular transaction a deep lookup within a block is required only in the single block in which the transaction appears.

Learned Bloom filters - A recent learn-based model suggests an alternative perspective for Bloom filters [50, 51]. Correct answers to membership queries can be learned as a binary function and use a Bloom filter to avoid undesired types of errors such as false negatives. If the sets of elements are not selected arbitrary and for instance most of them are selected from a small range of values then it can be characterized by the learned function. Higher accuracy can then be achieved by reducing Bloom filter length and allocating that memory to representing the learned model.

This approach can improve some of the Bloom filter blockchain use cases. In particular, if the set of addresses relevant to an SPV Bitocoin client can be easily characterized, the full node can hold an alternative data structure composed of a Bloom filter and a learned model to further improve accuracy and reduce the redundant traffic sent to a light client.

V. Conclusions and Future Work

Data sketches are memory-efficient tools for high-speed computations over massive datasets. In many existing blockchain implementations sketches serve various functionalities such as block summary, light client representation and synchronization of transaction pools. We expect increase of their popularity in the blockchain context due to the growing memory and processing demands of popular blockchains and presented relevant wider sketch functionalities. In particular, we explained how sketches can be helpful in network analysis and anomaly detection. We also discussed how recent main advances in sketches can be applicable to blockchains.

VI. Acknowledgment

This work was partially supported by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel National Cyber Directorate, by the Gordon Fund for System Engineering, by the Alon fellowship, by German-Israeli Science Foundation (GIF) Young Scientists Program, by the Taub Family Foundation as well as by the Polak Fund for Applied Research at the Technion.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[2] G. Cormode, "Data sketching," *Commun. ACM*, vol. 60, no. 9, pp. 48–55, 2017.

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

[4] A. Z. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2003.

[5] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing Bloom filter: Challenges, solutions, and comparisons," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2019.

[6] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.

[7] B. Fan, D. G. Andersen, M. Kaminsky, and M. Mitzenmacher, "Cuckoo Filter: Practically better than Bloom," in *ACM CoNEXT*, 2014.

[8] L. Luo, D. Guo, O. Rottenstreich, R. T. B. Ma, X. Luo, and B. Ren, "The consistent Cuckoo filter," in *IEEE INFOCOM*, 2019.

[9] M. T. Goodrich and M. Mitzenmacher, "Invertible Bloom lookup tables," in *Allerton Conference on Communication, Control, and Computing*, 2011.

[10] R. C. Merkle, "Secrecy, authentication, and public key systems," *PhD thesis, Stanford*, 1979.

[11] D. Guo and M. Li, "Set reconciliation via counting Bloom filters," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2367–2380, 2013.

[12] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of SEQUENCES*, 1997.

[13] M. Durand and P. Flajolet, "LogLog counting of large cardinalities," in *European Symposium on Algorithms (ESA)*, 2003.

[14] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm," in *Conference on analysis of algorithm (AofA)*, 2007.

[15] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Joint EDBT/ICDT Conferences*, 2013.

[16] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The Count-Min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[17] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *ICDT*, 2005.

[18] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014. [Online]. Available: http://gavwood.com/Paper.pdf

[19] T. J. Rush, "Adaptive enhanced Bloom filters for identifying transactions of interest in a blockchain," 2017.

[20] M. Hearn and M. Corallo, "BIP37: Connection Bloom filtering," 2012. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki

[21] T. Wang, W. Zhu, Q. Ma, Z. Shen, and Z. Shao, "ABACUS: Address-partitioned Bloom filter on address checking for uniqueness in IoT blockchain," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020.

[22] A. P. Ozisik, G. Andresen, B. N. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: Efficient interactive set reconciliation applied to blockchain propagation," in *ACM SIGCOMM*, 2019.

[23] A. Orda and O. Rottenstreich, "Enforcing fairness in blockchain transaction ordering," in *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019.

[24] S. Shafeeq, S. Zeadally, M. Alam, and A. Khan, "Curbing address reuse in the IOTA distributed ledger: A Cuckoo-filter-based approach," *IEEE Trans. Engineering Management*, vol. 67, no. 4, pp. 1244–1255, 2020.

[25] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "The KECCAK SHA-3 submission," 2011. [Online]. Available: https://keccak.team/

[26] A. Gervais, S. Capkun, G. O. Karame, and D. Gruber, "On the privacy provisions of Bloom filters in lightweight Bitcoin clients," in *ACM Annual Computer Security Applications Conference (ACSAC)*, 2014.

[27] K. Kanemura, K. Toyoda, and T. Ohtsuki, "Design of privacy-preserving mobile Bitcoin client based on γ-deniability enabled Bloom filter," in *Int. Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017.

[28] O. Rottenstreich and I. Keslassy, "The Bloom paradox: When not to use a Bloom filter," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 703–716, 2015.

[29] K. Qin, H. Hadass, A. Gervais, and J. Reardon, "Applying private information retrieval to lightweight Bitcoin clients," in *Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2019.

[30] S. Popov, "The Tangle," 2016. [Online]. Available: https://www.iota.org/foundation/research-papers

[31] S. Rohde, T. Eisenbarth, E. Dahmen, J. Buchmann, and C. Paar, "Fast hash-based signatures on constrained devices," in *International Conference on Smart Card Research and Advanced Applications (CARDIS)*, 2008.

[32] J. Buchmann, E. Dahmen, S. Ereth, A. Hülsing, and M. Rückert, "On the security of the Winternitz one-time signature scheme," in *International Conference on Cryptology in Africa*, 2011.

[33] M. Corallo, "BIP152: Compact block relay," 2016. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

[34] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The Honey Badger of BFT Protocols," in *ACM CCS*, 2016.

[35] A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, R. Tamari, and D. Yakira, "A fair consensus protocol for transaction ordering," in *IEEE International Conference on Network Protocols (ICNP)*, 2018.

[36] K. Lev-Ari, A. Spiegelman, I. Keidar, and D. Malkhi, "FairLedger: A Fair Blockchain Protocol for Financial Institutions," in *International Conference on Principles of Distributed Systems (OPODIS)*, 2019.

[37] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, "Order-Fairness for Byzantine Consensus," *IACR Cryptology ePrint Archive, 2020:269*, 2020.

[38] A. Mizrahi, N. Koren, and O. Rottenstreich, "Optimizing Merkle proof size for blockchain transactions," in *International Conference on Communication Systems and Networks (COMSNETS)*, 2021.

[39] T. Pham and S. Lee, "Anomaly detection in Bitcoin network using unsupervised learning methods," *CoRR*, vol. abs/1611.03941, 2016.

[40] ——, "Anomaly detection in the Bitcoin system - A network perspective," *CoRR*, vol. abs/1611.03942, 2016.

[41] P. Monamo, V. N. Marivate, and B. Twala, "Unsupervised learning for robust Bitcoin fraud detection," in *Information Security for South Africa (ISSA)*, 2016.

[42] M. Ostapowicz and K. Zbikowski, "Detecting fraudulent accounts on blockchain: A supervised approach," in *Web Information Systems Engineering - (WISE)*, 2019.

[43] S. Sayadi, S. B. Rejeb, and Z. Choukair, "Anomaly detection model over blockchain electronic transactions," in *International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2019.

[44] B. Podgorelec, M. Turkanovic, and S. Karakatic, "A machine learning-based method for automated blockchain transaction signing including personalized anomaly detection," *Sensors*, vol. 20, no. 1, p. 147, 2020.

[45] M. Zwang, S. Somin, A. Pentland, and Y. Altshuler, "Detecting bot activity in the Ethereum blockchain network," *CoRR*, vol. abs/1810.01591, 2018.

[46] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *ACM SIGCOMM*, 2016.

[47] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *ACM SIGCOMM*, 2018.

[48] S. Z. Kiss, É. Hosszu, J. Tapolcai, L. Rónyai, and O. Rottenstreich, "Bloom filter with a false positive free zone," in *IEEE INFOCOM*, 2018.

[49] O. Rottenstreich, P. Reviriego, E. Porat, and S. Muthukrishnan, "Constructions and applications for accurate counting of the Bloom filter false positive free zone," in *ACM Symposium on SDN Research (SOSR)*, 2020.

[50] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *ACM SIGMOD*, 2018.

[51] M. Mitzenmacher, "A model for learned Bloom filters and optimizing by sandwiching," in *NeurIPS*, 2018.