SREP: Out-Of-Band Sync of Transaction Pools for Large-Scale Blockchains

Novak Boškov, Sevval Simsek, Ari Trachtenberg, and David Starobinski
Department of Electrical and Computer Engineering
Boston University, Boston, Massachusetts, USA
{boskov,sevvals,trachten,staro}@bu.edu

***Abstract*—Synchronization of transaction pools (*mempools*) has shown potential for improving the performance and block propagation delay of state-of-the-art blockchains. Indeed, various heuristics have been proposed in the literature to this end, all of which incorporate exchanges of unconfirmed transactions into their block propagation protocol. In this work, we take a different approach, maintaining transaction synchronization outside (and independently) of the block propagation channel. In the process, we formalize the synchronization problem within a graph theoretic framework and introduce a novel algorithm (*SREP - Set Reconciliation-Enhanced Propagation*) with quantifiable guarantees. We analyze the algorithm's performance for various realistic network topologies, and show that it converges on any connected graph in a number of steps that is bounded by the diameter of the graph. We confirm our analytical findings through extensive simulations that include comparison with *MempoolSync*, a recent approach from the literature. Our simulations show that *SREP* incurs reasonable overall bandwidth overhead and, unlike *MempoolSync*, scales gracefully with the size of the network.**

***Index Terms*—Blockchains, Overlay networks, Peer-to-peer computing**

I. Introduction and Related Work

Block propagation represents a fundamental aspect of many blockchain networks in which blockchain nodes forward newly created blocks to their neighbors. Historically, block propagation has been performed by sending all the transactions belonging to the block alongside the block's metadata. Often, a substantial number of the block's transactions are present on the receiving end, resulting in unnecessarily high *bandwidth overhead*. To cope with such overhead, more advanced block propagation protocols such as *CompactBlock* [1], *Xtreme Thin Blocks* [2], *Graphene* [3], and *Gauze* [4] have been introduced.

Yet, it has recently been demonstrated through *in-situ* measurements in live blockchains, including Bitcoin, that the performance of these advanced block propagation protocols can significantly degrade when transaction pools go out of sync [5]–[8]. One approach to prevent such performance degradation is to have neighboring nodes regularly synchronize their pools of unconfirmed transactions. Toward this end, the recent work in [6] proposes a heuristic, called *MempoolSync*, that is shown to reduce the average block propagation delay by 50% in the Bitcoin network. Yet, *MempoolSync* does not provide any quantifiable *guarantees* on overall communication or delay performance.

In this work, we study the problem of transaction pool synchronization (*sync*) from a fundamental, graph-theoretic perspective, which allows us to analyze synchronization performance metrics in various network topologies. Our main contributions are as follows:

- We introduce a novel transaction pool sync algorithm, called *SREP*, which functions in an assistive capacity outside of the existing block propagation protocols.
- We analyze the performance of *SREP* in general network topologies, including a more specialized model that captures topological properties of actual blockchains (*e.g.,* the "small-world" property) as well as the statistics of transaction pools.
- We develop a simulation approach based on realistic transaction pool data from measurement campaigns, and confirm our analytical findings through simulations.
- We show that *SREP* has significantly lower bandwidth overhead than *MempoolSync*.

The rest of this paper is organized as follows. In Section II, we overview the related work. In Section III, we introduce *SREP*. In Section IV, we analyze the properties of *SREP* and validate our findings through simulations in Section V. We compare *SREP* with a transaction pool synchronization approach from the literature in Section V-C. Finally, we give a conclusion and propose future work in Section VI.

II. Background

To the best of our knowledge, *SREP* is a unique distributed algorithm that explicitly tackles the problem of network-wide synchronization of unconfirmed transactions — *transaction pools* [9]. To achieve its goals, *SREP* relies on *communication-efficient* solutions to the *set reconciliation* problem [10], which is defined as follows. Given two remote parties with their corresponding data sets S_A and S_B, each party needs to discover the elements local to the other. Communication-efficient solutions to this problem exchange only messages of size proportional to the number of *mutual differences* defined as $(S_A \setminus S_B) \cup (S_B \setminus S_A)$ and often denoted as $S_A \oplus S_B$.

In fact, there has been several communication-efficient set reconciliation algorithms proposed in the literature including Characteristic Polynomial Interpolation [11] (CPI), BCH

979-8-3503-1019-1/23/$31.00 ©2023 IEEE

codes [12], and Invertible Bloom Lookup Tables (IBLT) [13]–[15]. For instance, CPI incurs a communication cost *equal* to the number of mutual differences plus a small constant, which makes it nearly *optimal* in communication [11]. On the other hand, IBLT-based solutions typically offer better *computational* complexity at the cost of increasing their communication cost by a constant factor. To further reduce this communication overhead, Lázaro and Matuz [15] have recently proposed an IBLT-based solution that brings the communication cost closer to that of CPI while keeping the computational complexity low.

On the other hand, when it comes to our analytical model and simulations, we make use of the findings from the blockchain topology-discovering literature. In particular, Wang *et al.* [16] and Gao *et al.* [17] independently verified that the Ethereum network exhibits "small-world" property. Recently, Shahsavari *et al.* [18] used a random graph model to simulate Bitcoin network and Ma *et al.* [19] proposed a topology generation based on Watts-Strogatz [20] random graph model to capture the Bitcoin network in their *CBlockSim* simulator.

III. SREP Algorithm

We propose a novel distributed algorithm for network-wide transaction pool synchronization called *SREP* (*Set Reconciliation-Enhanced Propagation*). The core building block of *SREP* is a concept that we denote as *primal* sync — a set reconciliation protocol with communication complexity linear in the number of symmetric differences (*e.g.,* CPI [11]). Given the local transaction pool as a set of globally unique identifiers [21], *SREP* invokes one primal sync per each neighbor in parallel.

One way to support many parallel invocations of primal syncs is to create one transaction pool *replica* per each neighbor. Then run primal syncs in parallel using the corresponding replicas to avoid write collisions. Upon the completion of all parallel tasks, we can reuse the primal sync to incorporate new elements into the local transaction pool. We describe *SREP* in Algorithm 1 using S_n to denote the transaction pool at node n, d_{in} to denote the differences between S_i and S_n that reside in S_i, and **Sync** to denote a primal sync. As an illustration, in Fig. 1, we depict one iteration of *SREP*'s main loop (line 2), assuming that each node n holds only one transaction whose hash is also n.

Avoiding Full Replication

SREP from Algorithm 1 has a significant memory overhead caused by transaction pool replication for each neighbor. However, certain primal syncs allow us to implement *SREP* without replication, thus mitigating this memory overhead. In particular, multiple set reconciliation algorithms mentioned in Section II use data set *sketches* to perform synchronization and modify the underlying data sets only at the end of the protocol.

For instance, CPI reads from the set only once, at the beginning of the protocol, and writes to it only once at the end of the protocol. Suppose that we choose CPI as the primal sync in *SREP*. Then we can construct the characteristic polynomial [10] of S_n as the very first step in each iteration (after line 2 in Algorithm 1). Instead of using the neighbor replicas, we can now use the same characteristic polynomial in all neighbor threads. As no thread will modify the polynomial, the procedure is thread-safe and the threads can now write directly to the underlying set. Although the write operation will need to acquire the corresponding lock, since set union is commutative and associative, the order in which the threads update the set does not matter. As we now avoid replication, the local synchronization step can be safely eliminated altogether.

Note that this implementation improvement does not change the functional properties of *SREP*. That is, each thread still operates on its own version of the sketch and will update its sketch only at the beginning of the subsequent iteration. Hence, a difference that arrives in iteration i via some neighbor thread will only get acknowledged by other threads in iteration $i+1$. For that reason, we use the notion of "replicas" in the subsequent analysis.

Algorithm 1: *SREP* Algorithm.

Input: Network $G = (V, E)$ as adjacency list.
1 **At each node** $n \in \{0, |V| - 1\}$
2   **Loop**
3   **for** i **in** $G[n]$ **do** `// Neighbors of n`
4       $S_n^i \leftarrow S_n$; `// Replicate data set`
5   **Do in parallel**
        `// Network sync`
6         $d_{in} \leftarrow$ **Sync** (S_n^i, S_i) ;
7         $S_n^i \leftarrow S_n^i \cup d_{in}$;
8   **for** i **in** $G[n]$ **do**
      `// Local sync`
9       $S_n^i \setminus S_n \leftarrow$ **Sync** (S_n, S_n^i) ;
10       $S_n \leftarrow S_n \cup (S_n^i \setminus S_n)$;

$G = (V, E)$	Network of $\lvert E\rvert$ edges and $\lvert V\rvert$ nodes
S_n	Transaction pool at node $n \in \{0..\lvert V\rvert - 1\}$
$d_{ij} = S_i \setminus S_j$	Differences between i and j that reside in i
$\overline{deg}$	Average node degree
t_n	Time node n spends to synchronize with all its neighbors once
$T_{x\%}$	Time until $x\%$ of G is synchronized
$\Sigma_{x\%}$	Number of primal sync invocations
$C_{x\%}$	Overall communication cost

TABLE I: Summary of notation.

IV. *SREP* Performance Analysis

Several aspects affect the performance of *SREP*, including the network topology and the statistics of transaction pools. To aid our analysis, we first define an explicit network model, and then analyze *SREP* in a step-by-step fashion. In each stage of our analysis, we describe a *SREP* variant with the corresponding set of *simplifying assumptions* and analyze

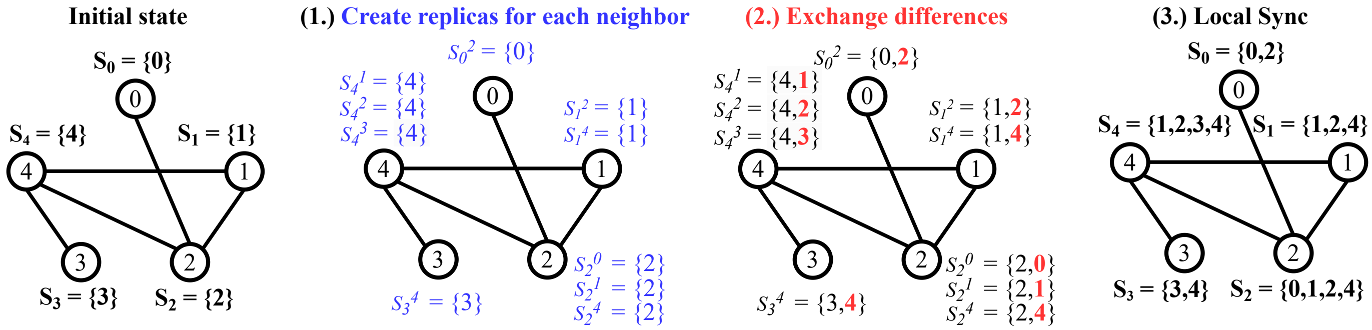

Fig. 1: One iteration of *SREP* on a tractably small network.

its performance. By successively relaxing these assumptions, we arrive at the final version of *SREP*. Table I summarizes notation used throughout this work.

Definition 1: We use $T_{x\%}$, $\Sigma_{x\%}$, and $C_{x\%}$ to denote time, total number of primal sync invocations, and total communication cost until $x\%$ of transaction pools in the network are equal. When $x = 100$, we say that *full network* synchronization is achieved — the ultimate goal of *SREP*.

A. Network Model

Watts-Strogatz [20] random graphs allow us to describe a wide range of realistic blockchain network topologies reasonably well [16], [17], [19], [22]. A typical set of parameters to Watts-Strogats model are the number of nodes in the network $|V|$, average node degree $\overline{deg}$, and rewire probability p [20].

For instance, each Bitcoin node selects 8 random neighbors upon joining the network [23]–[25], which has been shown to yield an unstructured random graph [18]. We can capture this in the Watts-Strogatz model by setting $\overline{deg} = 8$ and $p = 1$. Ethereum's neighbor selection mechanism, on the other hand, relies on a Kademlia distributed hash table (DHT) [26], and yields a network with more structure [17]. Notwithstanding this, multiple recent measurement results have independently confirmed that the generated network exhibits the "small world" property and fits the Watts-Strogatz model [16], [17], [22]. That is, the average shortest path between any two nodes can be reasonably approximated by $O\left(log_{\overline{deg}}|V|\right)$, and the diameter of the network is small [27].

Besides the graph topology, our network model also captures the states of transaction pools across the network. In particular, we define the *pool assignment* A as a collection of sets $S_0..S_{|V|-1}$ where set S_i represents the transaction pool at node i. We model the statistical properties of A through the following *pool parameters*:

- $\mathcal{S}$: *sizes distribution*. A discrete random variable describing the sizes of transaction pools S_i for $i \in \{0...|V|-1\}$,
- s: *sizes vector*. A $|V|$-size vector where elements are drawn from $\mathcal{S}$,
- $\mathcal{P}$: *differences distribution*. A discrete random variable describing the sizes of mutual differences between the pairs of transaction pools (*i.e.,* $|S_i \oplus S_j|$),
- M: *mutual differences matrix*. A $|V| \times |V|$ upper triangular matrix of mutual differences. For the given topology $G = (V, E)$, the elements of the matrix are defined as:

$$m_{ij} = \begin{cases} |S_i \oplus S_j| & \text{when } (i,j) \in E \text{ and } i < j, \\ 0 & \text{otherwise.} \end{cases}$$

  Non-zero elements are drawn from $\mathcal{P}$.
- $\mathcal{U}$: *universe*. A discrete random variable from which we draw transaction IDs. We choose $\mathcal{U}\{0, u\}$ to be a uniform random variable for some $u \geq |V|$.

B. Elementary SREP *(*E-SREP*)*

The starting point for our build up of *SREP* is called *elementary SREP* (Algorithm 2). We summarize its simplifying assumptions as follows:

- (A_1) All nodes have global view of the network.
- (A_2) Initially, the transaction pools at each node contain only one element (transaction) that is unique across all network nodes (*e.g.,* index of the node). Strictly speaking, we set the pool parameters as: $\mathcal{S} = 1$, $\mathcal{P} = 2$, and $u \gg |V|$.
- (A_3) No new transactions arrive to the network after the initialization.
- (A_4) In one iteration of elementary *SREP* (line 1), nodes take turns to perform their synchronization duties such that no two nodes invoke primal sync at the same time. For instance, nodes with smaller indices go first. An iteration ends when all nodes have invoked synchronization once for all their neighbors.
- (A_5) Nodes synchronize with their neighbors sequentially. For instance, the neighbors with smaller indices get synchronized first (line 3).
- (A_6) All synchronizations are two-way (lines 7 and 8), meaning that the differences are exchanged in both directions.
- (A_7) All synchronizations take equally long.

In the context of *E-SREP*, the following special case is particularly significant for the analysis.

Lemma 1: For *E-SREP* over a complete graph $G = (V, E)$, the communication cost to sync the entire network is

$$C_{100\%}(G) = |V| \cdot (|V| - 1).$$

Algorithm 2: Elementary *SREP*.

```
Input: Network G = (V, E) as adjacency list.
while network is not fully synchronized do
    for n ← 0 to {0..|V| − 1} do
        neighbors ← sort ( G[n] ) ;
        for i in neighbors do
            d_in ← Sync ( S_n, S_i ) ;
            d_ni ← Sync ( S_i, S_n ) ;
            S_n ← S_n ∪ d_in ;
            S_i ← S_i ∪ d_ni ;
```

C. *Elementary Parallel* SREP *(*EP-SREP*)*

The main aim of the *elementary parallel SREP* is to relax (A_1), (A_4) and (A_5). Instead of invoking synchronization in order, *EP-SREP* invokes synchronization for all neighbors at once (*i.e.,* Algorithm 1). In addition to that, we also relax (A_7). The synchronization between nodes u and v now takes time *equal* to the number of their mutual differences (*i.e.,* $|d_{uv} \cup d_{vu}|$). As discussed earlier in Section II, this is a reasonable assumption to make (*e.g.,* CPI has such a property).

Theorem 1: In *EP-SREP* and for any connected network $G = (V, E)$, we have the following bounds on the overall communication cost until the network is fully synchronized:

$$|V| \cdot (|V| - 1) \leq C_{100\%} < |V| \cdot (|V|^2 - 1).$$

Proof: The lower bound is obtained similarly as in Lemma 1. The least amount of communication to achieve full synchronization is equivalent to each node sending its element to all the other nodes directly. On the other hand, we get the upper bound by observing that there cannot be more than $|V|^2 \cdot (|V| - 1)$ *redundant* element transmissions on top of the lower bound. Redundant transmissions happen when a node receives an element via multiple replicas in the same iteration. To count all redundant transmissions, we observe that, in each iteration, each node either receives some new elements or does not receive any. In the latter case, obviously, no redundant transmissions happen. Otherwise, if there are some new elements received, the following holds: (1) there will be no more than $|V|$ new elements arriving at the node across all iterations, as there is only that much elements in the network, and (2) for each element, there cannot be more than $|V| - 1$ redundant transmissions, as there cannot be more than that much replicas at any node. Thus, there cannot be more than $|V|^2 \cdot (|V| - 1)$ redundant transmissions at all nodes in all iterations. ■

As in Watts-Strogatz networks we have $\overline{deg}$ replicas at each node on average, the same counting argument from above applies in the following form.

Corollary 1: For *EP-SREP* in Watts-Strogatz networks:

$$C_{100\%} < |V| \cdot (|V| \cdot \overline{deg} + |V| - 1).$$

On the other hand, to infer the upper bound on the time that *EP-SREP* needs to complete a full sync ($T_{100\%}$), we rely on following definition.

Definition 2: $I_{x\%}(G)$ is the maximal number of *EP-SREP* iterations (line 2 in Algorithm 1) at any node to achieve x% network synchronization.

Theorem 2: In *EP-SREP* and for any connected network $G = (V, E)$, with the shortest path between nodes u and v denoted as $dist(u, v)$, the maximum number of iterations required for a full network synchronization is equal to the diameter of the network:

$$I_{100\%}(G) = \max_{u,v \in V} dist(u, v).$$

Proof: By the definition of full synchronization, all elements need to reach every other node. Without a loss of generality, suppose that we follow the propagation of some element $i \in V$ during the execution of *EP-SREP*. Since the graph is connected, in each iteration of *EP-SREP*, i will progress exactly one step further through the network. The number of iterations required to synchronize the entire network is then equivalent to the maximum distance between any two nodes in the network (*i.e.,* diameter). ■

Lemma 2: In *EP-SREP* over complete graphs $G = (E, V)$:

$$I_{100\%}(G) = 1 \text{ and } C_{100\%} = |V| \cdot (|V| - 1).$$

The former holds as the diameter of complete graphs is 1. The latter is a consequence of the former; as no element traverses more than one edge, there cannot be any redundant transmissions.

Corollary 2: For *EP-SREP* and Watts-Strogatz networks, the maximal number of iterations at any node to synchronize the entire network ($I_{100\%}$) is logarithmic in the size of the network.

Counting the number of nodes that have heard about an element $n \in V$ in iteration i of *EP-SREP* over a Watts-Strogatz network, we get the following sum:

$$1 + \overline{deg} + \overline{deg}^2 + \ldots + \overline{deg}^i.$$

By equating it to $|V|$, we can express i, the number of iterations until all nodes have heard of n, as a logarithmic function of $|V|$ [27]. Practically speaking, *EP-SREP* will complete in logarithmically small number of iterations ($\approx 4 \log_{\overline{deg}}(10))$) for the blockchain networks of realistic sizes (*e.g.,* Blockchain and Ethereum [24], [25]).

Theorem 3: In general graphs $G = (V, E)$, the following holds for *EP-SREP*:

$$T_{100\%} \leq I_{100\%}(G) \cdot \max_{i \in V} t_i < I_{100\%}(G) \cdot |V|,$$
$$\Sigma_{100\%} \leq I_{100\%}(G) \cdot |E|.$$

Proof: Since synchronizations happen in parallel, the overall elapsed time is proportional to the number of iterations. Any sync invocation at any node will take strictly less than $|V|$, as no two data sets can differ in more than $|V| - 1$ elements (each data set keeps exactly one element at the beginning). Since in each iteration nodes sync with all their neighbors and each sync is two-way by (A_6), there will be no more than $|E|$ syncs in each iteration. ■

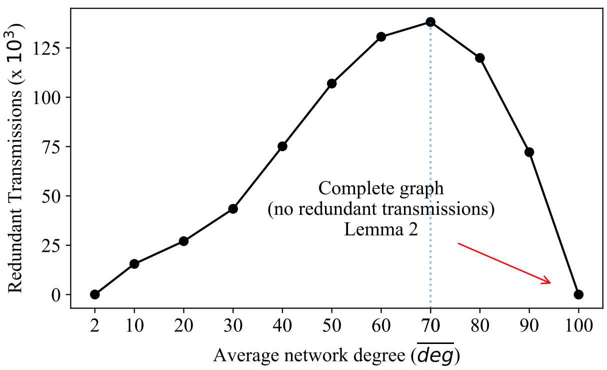

Fig. 2: Amount of redundant transmissions in *EP-SREP* over a network of 100 nodes ($p = 0.24$).

The $\overline{deg}$ Dilemma: Due to the counting argument from Theorem 1, the upper bound on overall communication cost is *not* tight; there must be at least some elements that will *not* generate redundant transmissions in any connected network. On top of that, the topology of the network plays a complex role in generating redundant transmissions. Intuitively speaking, the impact of $\overline{deg}$ in Watts-Strogatz networks is twofold, and conflicting: (1) the larger $\overline{deg}$, the larger the average number of replicas per node, which may cause redundant transmissions, and (2) the larger $\overline{deg}$, the shorter the average pair-wise shortest path among the nodes in the network, which makes each element traverse less intermediate nodes to reach the entire network, thus reducing the probability of redundant transmissions. We plot this non-monotonic effect that $\overline{deg}$ has on the amount of redundant transmissions in Fig. 2 for a tractably small network. Up to a point, the first effect (replicas count) prevails and drives the overall communication cost up. After that point, the second effect (path shortening) prevails and drives the overall communication cost down all the way to the point when the network becomes a complete graph and there is no redundant transmissions at all.

D. *Multi-element* SREP

The final stage in building *SREP* is *multi-element* SREP. We build it by relaxing (A_2) — transaction pools can now initially contain multiple elements. In terms of our network model, this means that our $\mathcal{S}$ (sizes distribution) and $\mathcal{P}$ (differences distribution) are no more constant. Thus, *SREP* is a *generalization* of *EP-SREP*.

Definition 3: Function $f : (G, A) \mapsto \mathbb{Z}$ maps a pair of a topology G and a pool assignment A to a non-negative integer via first constructing the corresponding mutual differences matrix M, then computing $\sum m_{ij}$.

Definition 4: Function $g : (G, A) \mapsto (G, A_{(next)})$ maps a pair of a topology G and a pool assignment A to the same topology G and a transformed pool assignment $A_{(next)}$. We define the transaction pools in the transformed pools assignment $A_{(next)}$ as:

$$S_{(next)i} = S_i \cup (\bigcup_{j \in G[i]} S_j).$$

We use $\bigcup_{j \in G[i]} S_j$ to denote the union of all transaction pools S_j corresponding to the neighbors of node i in the previous iteration.

Definition 5: For some function h, we write $h^{(n)}(x)$ to denote the composition of function h with itself n times, starting with argument x:

$$h^{(n)}(x) = \underbrace{h \circ h \cdots h}_{n}(x).$$

Definition 6: $A_{(n)}$ is the assignment resulting from n compositions of g with itself starting with the initial pool assignment that we denote as $A = A_{(0)}$.

Lemma 3: For a network model (G, A) where G is a connected graph and A the initial pool assignment, the number of *SREP* iterations to achieve the full network synchronization $I_{100\%}(G, A)$ is given as a solution to the following equation:

$$f(g^{(I_{100\%}(G,A))}(G, A)) = 0.$$

Note that by Definition 4, g exactly corresponds to one iteration of *SREP*. That is, the transformed pool assignment $A_{(next)}$ reflects the state of the transaction pools after an iteration of *SREP* at all nodes in the network. Composing g with itself n times corresponds to repeating an iteration of *SREP* at all nodes n times. By a similar argument as in Theorem 2, all elements will reach all nodes after some number of iterations. Since this implies that no two sets have any differences, M will be an all-zeros matrix. That is, $(f \circ g^{(n)})(G, A)$ has at least one zero. Thus, the number of times we need to compose g with itself until $f(G, A_{(n)}) = 0$ gives us the maximal number of *SREP* iterations to achieve full network synchronization.

Theorem 4: For a connected graph $G = (V, E)$ and an initial pool assignment A, the number of *SREP* iterations to achieve the full network synchronization is bounded by the diameter of the network:

$$I_{100\%}(G, A) \leq \max_{u,v \in V} dist(u, v).$$

Proof: As *SREP* is a generalization of *EP-SREP*, the argument here is similar to that of Theorem 2. To achieve the full network synchronization, elements need to traverse at most the diameter of G. As opposed to *EP-SREP*, in *SREP* each element may initially appear at more than one node, dictated by the differences distribution $\mathcal{P}$. Thus the diameter is an upper bound on *SREP* iterations. ■

Lemma 4: For a connected graph $G = (V, E)$ and initial pool assignment A with the corresponding mutual differences matrix M, the communication cost of *SREP* is:

$$C_{100}(G, A) = \sum_{i=0}^{I_{100\%}(G,A)} f(G, A_{(i)})$$
$$< I_{100\%}(G, A) \cdot \max\{f(G, A), \ldots, f(G, A_{(I_{100\%}(G,A))})\}.$$

In ith iteration of *SREP*, we transmit exactly as much elements as there are in the differences matrix that corresponds to $A_{(i)}$. Given $I_{100\%}(G, A)$ from Lemma 3, we get the overall communication cost of *SREP*.

Lemma 5: In *SREP* over a connected network $G = (V, E)$ with the given initial pool assignment A and the largest order statistics of differences distribution $\mathcal{P}$ denoted as $\mathcal{P}_{(n)}$:

$$T_{100\%} \leq I_{100\%}(G, A) \cdot \max_{i \in V} t_i = I_{100\%}(G, A) \cdot \mathcal{P}_{(n)},$$
$$\Sigma_{100\%} \leq I_{100\%}(G, A) \cdot |E|.$$

The argument is similar to that of Theorem 3.

Finally, note that the assumptions in our analysis such as (A_3) — no new transactions arrive after *SREP* starts, are artificial in that they simplify our analysis, but they do not constrain *SREP* in practice. The properties such as the overall communication cost ($C_{100\%}$) and time ($T_{100\%}$) to sync the entire network relate to the transactions that have arrived before *SREP* begins.

V. SIMULATIONS

To validate our analytical findings about *SREP*, we construct an event-based simulator called *SREPSim* [28] that shares the topology generation procedure with *CBlockSim* of Ma *et al.* [19] and adds the other parameters of our network model described in Section IV-A.

In the rest of this section, we first describe a method to parameterize our network model. Then, we use such parameterized model to validate the main analytical properties of *SREP*. We then compare the overall communication cost of *SREP* with a similar approach from the literature. At the end, we present a *SREPSim* optimization that allows for easy *SREP* communication cost calculation over large-scale networks.

A. *Configuring Network Model Parameters*

Unlike the simulation approaches from the literature (*e.g., SimBlock* [29]), our network model can seamlessly integrate real-world transaction pool data. For instance, the empirical distributions of $\mathcal{S}$ and $\mathcal{P}$ can be generated for some small subset of all nodes in the network using the measurement software such as *log-to-file* of Imtiaz *et al.* [30], [31]. This software instruments adjacent Bitcoin nodes and periodically serializes the snapshots of their transaction pools. From these transaction pool snapshots, we can measure transaction pool sizes and their mutual differences to construct the empirical distributions for $\mathcal{S}$ and $\mathcal{P}$.

For the purpose of this work, we have conducted a 3-day long measurement campaign on two time-synchronized Bitcoin nodes and requested the transaction pool snapshots each minute. Fig. 3 depicts the results that we obtained. Roughly speaking, the set sizes fit the Maxwell distribution reasonably well, while the number of mutual differences fits the Hyperbolic distribution. Next, given the empirical distribution of $\mathcal{S}$, we need to configure the rest of our network model's pool parameters[1]. Ultimately, we need to construct a pool assignment A that conforms to the differences distribution $\mathcal{P}$.

In *SREPSim*, we construct such assignments through Procedure 1. For the given network topology $G = (V, E)$ and

[1]Direct usage of $\mathcal{P}$ is also possible but perhaps harder.

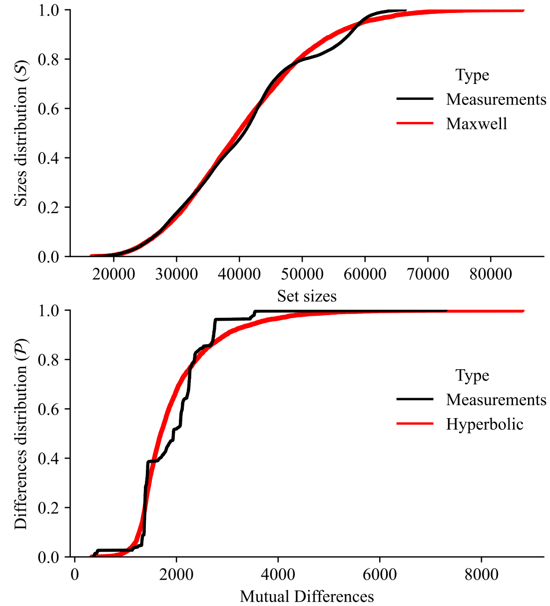

Fig. 3: Empirical distributions of transaction pool sizes $\mathcal{S}$ for two adjacent Bitcoin nodes (up) and their mutual differences $\mathcal{P}$ (down). Best distribution fits in red (using Error Sum of Squares).

the sizes distribution $\mathcal{S}$, we need to configure the parameter ψ such that the resulting assignment A produces a differences distribution that resembles $\mathcal{P}$. As shown in Fig. 4, $\psi = 0.35$ works reasonably well with our empirical sizes distribution. Note that by increasing ψ, we can decrease the average similarity among the transaction pools (*i.e.,* increase the number of their mutual differences).

Procedure 1: Network parameterization in SREPSim.

Input: Network $G = (V, E)$.
Input: Sizes distribution $\mathcal{S}$.
Input: Parameter ψ.
Output: Pool assignment A.

$u \leftarrow \lceil \psi \mathbb{E}[\mathcal{S}] \rceil$;
$\mathcal{U}\{0, u-1\}$; `// Uniform distribution`
sizes $\leftarrow$ **sample** $|V|$ **elements from** $\mathcal{S}$;
A $\leftarrow$ [ ] ;
for $i \leftarrow 0$ **to** $|V| - 1$ **do**
    $S_i \leftarrow$ **sample** sizes[i] **elements from** $\mathcal{U}$;
    A.append (S_i) ;

B. SREP *Properties Validation*

The main analytical properties that we want to validate through simulations are *SREP*'s communication cost to achieve full network sync ($C_{100\%}$) and the time required to achieve this state ($T_{100\%}$). In particular, we want to show how these two quantities change as a function of the network topology and the measure of difference among the transaction pools.

In Fig. 5, we plot the maximal number of *SREP* iterations $I_{100\%}$ and the network diameter as functions of the average network degree $\overline{deg}$. In Fig. 6, we plot the communication

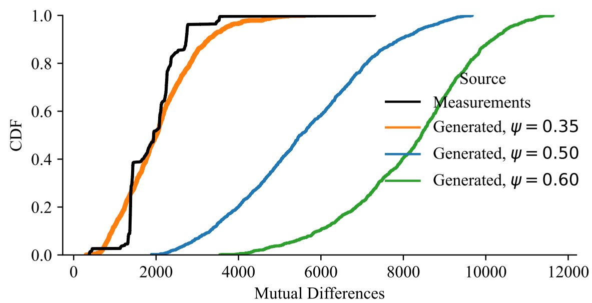

Fig. 4: Empirical differences distribution for two adjacent Bitcoin nodes versus the differences distribution generated by Procedure 1 for various ψ. Watts-Strogatz network with 100 nodes ($\overline{deg} = 19$ and $p = 0.24$).

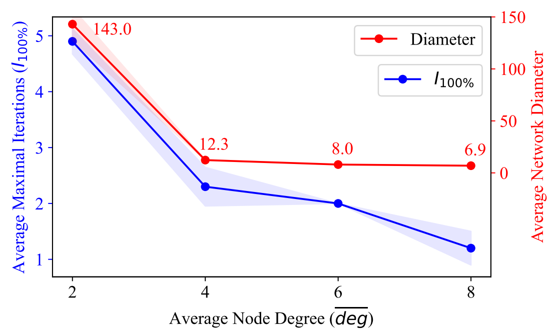

Fig. 5: Maximal number of *SREP* iterations at any node ($I_{100\%}$) bounded by the network diameter for Watts-Strogatz graphs with 1000 nodes ($p = 0.24$). 95% confidence intervals.

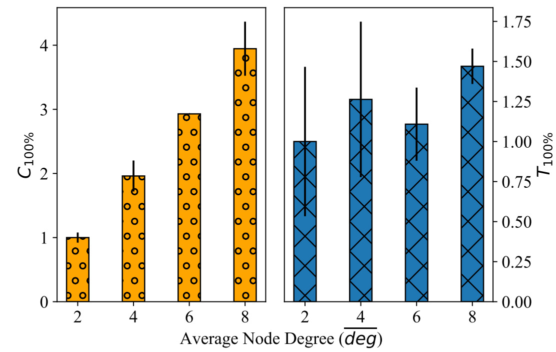

Fig. 6: Relative communication cost ($C_{100\%}$) and time to fully synchronize the network ($T_{100\%}$). Network with 1000 nodes ($p = 0.24$).

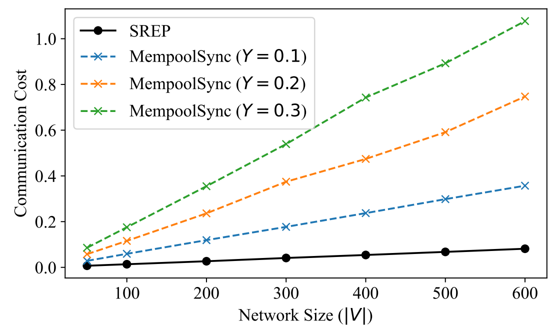

Fig. 7: Normalized overall communication cost of *SREP* ($C_{100\%}$) and *MempoolSync* as a function of network size. Data from Section V-A. $DefTXtoSync = 1000$. Y is the *MempoolSync* heuristic constant.

cost and time to full network sync as a function of $\overline{deg}$. The main observation is that the overall communication increases with the average node degree as a consequence of using more replicas per node, which increases the number of redundant transmissions (see Fig. 2). On the other hand, the time to achieve full network sync does not exhibit such a trend. Since primal syncs run in parallel, it is the maximal number of differences among any two nodes in the network that dominates the total time to sync the network (see Lemma 5).

C. *Comparison with* MempoolSync

MempoolSync of Imtiaz *et al.* is a transaction pool synchronization protocol that can improve the average transaction propagation delay by 50% in the event of churn in the Bitcoin network [6]. Here we describe this protocol and compare its communication efficiency with our newly proposed *SREP* through simulations.

As pointed out in [6], the main reason for slow block propagation times is a large number of missing transactions in the transaction pools of the block-receiving nodes. This effect occurs in the legacy block propagation protocols such as *CompactBlock* [1] and the more recent improvements such as *Graphene* [3], [5]. Thus, the goal of *MempoolSync* is to supply the nodes with potentially missing transactions, and it does so through an *ancestor score*-based heuristics [32]. The protocol uses a small constant `DefTXtoSync` as the default number of transaction hashes that the transmitting node will select from its transaction pool in descending order of ancestor score. The transmitting node will send exactly `DefTXtoSync` selected transaction hashes *unless* one of the following holds:

1) Transmitting node's transaction pool is much larger than `DefTXtoSync` (*e.g.,* 10 times). In this case, the node will send $Y \times$ `DefTXtoSync` top rated transactions, where Y is a constant between 0 and 1, or
2) Transmitting node's transaction pool is smaller than `DefTXtoSync`. In this case, the node will send its entire transaction pool. Because `DefTXtoSync` is a small constant, this is a quite rare event. It occurs only when the node has just joined the Bitcoin network or has just propagated a large block that triggered a massive transaction pool cleanup [6].

In Fig. 7, we compare the overall communication costs of *MempoolSync* and *SREP*. For *SREP*, we plot the communication cost to sync the entire network ($C_{100\%}$). For *MempoolSync*, we plot the communication cost that *MempoolSync* incurs until *SREP* would achieve a full sync.

Note that this kind of comparison gives an advantage to *MempoolSync*. While *SREP*'s $C_{100\%}$ implies that the network is fully synced, *MempoolSync*'s communication cost does not.

$\overline{deg}$	ψ	**Diameter** *average*	$I_{100\%}$ *average*	$C_{100\%}$ **(GB)** *average*
	0.355		**2.5**	**1.214397**
4	0.5	**16**	**3.0**	**3.165879**
	0.6		**3.1**	**4.801665**
	0.355		**1.7**	**2.428649**
8	0.5	**9**	**2.0**	**6.317304**
	0.6		**2.0**	**9.569259**
	0.355		**1.0**	**3.642738**
12	0.5	**7**	**1.5**	**9.485572**
	0.6		**2.0**	**14.347242**
	0.355		**1.0**	**4.876714**
16	0.5	**6**	**1.0**	**12.649385**
	0.6		**1.0**	**19.135943**
	0.355		**1.0**	**6.065679**
20	0.5	**5**	**1.0**	**15.804836**
	0.6		**1.0**	**23.886079**
	0.355		**1.0**	**7.294909**
24	0.5	**5**	**1.0**	**18.966694**
	0.6		**1.0**	**28.672272**
	0.355		**1.0**	**8.465624**
28	0.5	**5**	**1.0**	**22.156316**
	0.6		**1.0**	**33.446278**

TABLE II: *SREP* over a 10,000 nodes network. $p = 0.24$.

Procedure 2: SREPSim's analytical module.

Input: Network $G = (V, E)$.
Input: Initial pool assignment A as $S_0..S_{|V|-1}$.
Output: Overall network communication cost $C_{100\%}$.
Output: Maximal number of iterations $I_{100\%}$.

function `CalculateM(`A`)`**:**
  $M \leftarrow$ zeros($|V| \times |V|$) ; `// Zero matrix`
  for $i \leftarrow 0$ **to** $|V| - 1$ **do**
    for $j \leftarrow i + 1$ **to** $|V| - 1$ **do**
      if $i \in G[j]$ **then** `// i neighbor of j`
        $M[i][j] \leftarrow |S_i \oplus S_j|$;
  return M;
$C_{100\%} \leftarrow 0$;
$I_{100\%} \leftarrow 0$;
$M \leftarrow$ `CalculateM` (A) ;
while $\sum m_{ij} > 0$ **do**
  for $i \leftarrow 0$ **to** $|V| - 1$ **do**
    $S_i' \leftarrow S_i$; `// New assignment`
    for $j \in G[i]$ **do**
      $S_i' \leftarrow S_i' \cup S_j$;
  $C_{100\%} = C_{100\%} + \sum m_{ij}$;
  $I_{100\%} \leftarrow I_{100\%} + 1$;
  $A \leftarrow A'$;
  $M \leftarrow$ `CalculateM` (A) ;

In fact, *MempoolSync* has no guarantees about the communication (or time) needed to sync the entire network. Note also that *MempoolSync* uses Bitcoin internals to calculate the ancestor score of the transactions and later uses this score to determine which transactions to transmit. As opposed to *MempoolSync*, *SREP* is a general approach that does not rely on any Bitcoin internals and can be seamlessly integrated into other blockchains that keep transaction pools.

D. Communication Cost in Large-Scale Networks

Event-based simulators such as *SREPSim* may consume prohibitive amounts of memory and take a long time to complete simulations when the simulated network is large [19]. To address this issue, we designed a *SREPSim* module that computes *SREP*'s performance metrics analytically. In particular, we implement the functions from Definitions 3 and 4, and rely on the results from Lemma 4 to compute $C_{100\%}$ and $I_{100\%}$. We describe the *SREPSim*'s analytical module in Procedure 2. Using this module, we can easily compute the desired performance metrics for the networks of realistic sizes (*e.g.,* Bitcoin and Ethereum) [24], [25].

In Table II, we summarize the results for a 10,000 nodes network with various average node degrees ($\overline{deg}$) and the measure of similarity among transaction pools (ψ). As we report the communication cost, we assume that the transaction pools represent each transaction as a 32-byte long globally unique hash [21]. All simulations complete in tens of minutes.

VI. Conclusion

In this work, we have developed and analyzed *SREP*, an independent protocol that assists block propagation in large-scale blockchains. This new protocol synchronizes transaction pools of nodes in the blockchain network using communication-efficient set reconciliation approaches from the literature. However, rather than inserting itself directly into the block propagation process, as previous works have done, *SREP* operates in a distributed manner *outside* the block propagation channels of the network. As a result, it is easier to formally analyze its performance, and, indeed, we have shown that it completes in time bounded by the network diameter (or logarithmic in network size for the "small-world" networks that reasonably model blockchain networks).

We have also validated our analytical findings against a novel event-based simulator that we have developed. We run the simulator on real-world transaction pool statistics drawn from our own measurement campaign. In our simulations, *SREP* incurs only tens of gigabytes of overall bandwidth overhead to synchronize networks with ten thousand nodes, which is several times better than the current approach in the literature.

For future work, we propose to consider *multi-party* set reconciliation [33], [34] in the context of transaction pool sync. Though the main benefit may be further reduction in overall communication cost, it is not clear whether an advantage over pairwise approaches can be achieved when an average pairwise intersection is large compared to the total intersection ($\cap_i S_i$) [33].

Acknowledgments

The authors would like to thank Red Hat, the Boston University Red Hat Collaboratory (award # 2022-01-RH03), and the US National Science Foundation (award # CNS-2210029) for their support.

REFERENCES

[1] Matt Corallo, "Compact block relay protocol," https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki, 2016, (Accessed 2022-12-02).

[2] Peter Tschipper, "BUIP010 Xtreme Thinblocks," https://bitco.in/forum/threads/buip010-passed-xtreme-thinblocks.774/, 2016, (Accessed 2022-12-02).

[3] 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 *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 303–317.

[4] X. Ding, L. Zhao, L. Luo, J. Xie, D. Guo, and J. Li, "Gauze: Enabling Communication-Friendly Block Synchronization with Cuckoo Filter," *Frontiers of Computer Science*, vol. 17, no. 3, p. 173403, Sep 2022. [Online]. Available: https://doi.org/10.1007/s11704-022-1685-5

[5] M. A. Imtiaz, D. Starobinski, and A. Trachtenberg, "Empirical Comparison of Block Relay Protocols," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.

[6] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis, "Churn in the bitcoin network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1598–1615, 2021.

[7] ——, "Churn in the Bitcoin Network: Characterization and Impact," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 431–439.

[8] S. G. Motlagh, J. Mišić, and V. B. Mišić, "Impact of Node Churn in the Bitcoin Network," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 3, pp. 2104–2113, 2020.

[9] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A Survey of Distributed Consensus Protocols for Blockchain Networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[10] Y. Minsky and A. Trachtenberg, "Practical set reconciliation," in *40th Annual Allerton Conference on Communication, Control, and Computing*, vol. 248, 2002. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.456.7200

[11] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," in *Proceedings. 2001 IEEE International Symposium on Information Theory (IEEE Cat. No.01CH37252)*, 2001, pp. 232–, doi: https://doi.org/10.1109/ISIT.2001.936095.

[12] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 523–540, ISBN: 978-3-540-24676-3.

[13] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2011, pp. 792–799, doi: https://doi.org/10.1109/Allerton.2011.6120248.

[14] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the Difference? Efficient Set Reconciliation without Prior Context," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 218–229, doi: https://doi.org/10.1145/2018436.2018462.

[15] F. Lázaro and B. Matuz, "A Rate-Compatible Solution to the Set Reconciliation Problem," 2022. [Online]. Available: https://arxiv.org/abs/2211.05472

[16] T. Wang, C. Zhao, Q. Yang, S. Zhang, and S. C. Liew, "Ethna: Analyzing the Underlying Peer-to-Peer Network of Ethereum Blockchain," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2131–2146, 2021.

[17] Y. Gao, J. Shi, X. Wang, Q. Tan, C. Zhao, and Z. Yin, "Topology Measurement and Analysis on Ethereum P2P Network," in *2019 IEEE Symposium on Computers and Communications (ISCC)*, 2019, pp. 1–7.

[18] Y. Shahsavari, K. Zhang, and C. Talhi, "A Theoretical Model for Block Propagation Analysis in Bitcoin Network," *IEEE Transactions on Engineering Management*, vol. 69, no. 4, pp. 1459–1476, 2022.

[19] X. Ma, H. Wu, D. Xu, and K. Wolter, "CBlockSim: A Modular High-Performance Blockchain Simulator," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2022, pp. 1–5.

[20] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun 1998. [Online]. Available: https://doi.org/10.1038/30918

[21] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Analysis of the Bitcoin UTXO Set," in *Financial Cryptography and Data Security*, A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 78–91.

[22] L. Kiffer, A. Salman, D. Levin, A. Mislove, and C. Nita-Rotaru, "Under the Hood of the Ethereum Gossip Protocol," in *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 437–456. [Online]. Available: https://doi.org/10.1007/978-3-662-64331-0_23

[23] Bitcoin developers, "Bitcoin referential implementation," https://github.com/bitcoin/bitcoin, 2022, (Accessed 2022-12-02).

[24] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee, "TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions," in *Financial Cryptography and Data Security*, I. Goldberg and T. Moore, Eds. Cham: Springer International Publishing, 2019, pp. 550–566.

[25] M. Grundmann, M. Baumstark, and H. Hartenstein, "On the Peer Degree Distribution of the Bitcoin P2P Network," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2022, pp. 1–5.

[26] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65.

[27] F. Chung and L. Lu, "The diameter of sparse random graphs," *Advances in Applied Mathematics*, vol. 26, no. 4, pp. 257–279, 2001.

[28] N. Boškov, "SREPSim," http://www.github.com/nislab/SREPSim, (Accessed 2023-02-02).

[29] R. Banno and K. Shudo, "Simulating a Blockchain Network with SimBlock," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 3–4.

[30] M. A. Imtiaz, D. Starobinski, and A. Trachtenberg, "Characterizing Orphan Transactions in the Bitcoin Network," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020, pp. 1–9.

[31] ——, "Investigating Orphan Transactions in the Bitcoin Network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1718–1731, 2021.

[32] Bitcoin developers, "Ancestor Score Sorting," https://github.com/bitcoin/bitcoin/blob/master/src/txmempool.h, 2022, (Accessed 2022-12-02).

[33] M. Mitzenmacher and R. Pagh, "Simple multi-party set reconciliation," *Distributed Computing*, vol. 31, no. 6, pp. 441–453, Nov 2018. [Online]. Available: https://doi.org/10.1007/s00446-017-0316-0

[34] A. Boral and M. Mitzenmacher, "Multi-party set reconciliation using characteristic polynomials," in *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2014, pp. 1182–1187.