Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Out-Of-Band Transaction Pool Sync for Large Dynamic Blockchain Networks

Novak Boskov | **Xingyu Chen** | **Sevval Simsek** | **Ari Trachtenberg** | **David Starobinski**

Department of Electrical and Computer Engineering, Boston University, Boston, Massachusetts, United States

Correspondence
Corresponding author Novak Boskov,
Email: boskov@bu.edu

Present address
Novak Boskov is with Nokia Bell Labs, Murray Hill, New Jersey, United States.

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 incorporate early exchanges of *unconfirmed* transactions into the block propagation protocol. In this work, we take a different approach, maintaining transaction synchronization externally (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 static connected graphs in a time bounded by the diameter of the graph. In graphs with dynamic edges, *SREP* converges in an expected time that is linear in the number of nodes. We confirm our analytical findings through extensive simulations that include comparisons with *MempoolSync*, a recent approach from the literature. Our simulations show that *SREP* incurs reasonable bandwidth overhead and scales gracefully with the size of the network (unlike *MempoolSync*).

KEYWORDS
Blockchains, Overlay networks, Peer-to-peer computing

1 | INTRODUCTION

Block propagation represents a fundamental aspect in many blockchains, such as Bitcoin and Ethereum, in which nodes forward newly created blocks to their neighbors and maintain memory pools of unconfirmed transactions. 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 had already been present on the receiving end before transmission, 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], *Gauze*[4], and *Dino*[5] have been introduced.

At the same time, 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[6,7,8,9,10,11]. 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[7] proposes a heuristic, called *MempoolSync*, that has shown the potential to reduce the average block propagation delay by 50% in the Bitcoin network under realistic conditions. However, *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.

Abbreviations: SREP, Set Reconciliation-Enhanced Propagation; CPI, Characteristic Polynomial Interpolation;

 © 2023 Copyright Holder Name

- We analyze the performance of *SREP* in general static network topologies, including more realistic "small-world" networks.
- We analyze the performance of *SREP* in even more realistic *dynamic* networks, where connections between nodes disappear and reappear over time.
- We develop a simulation approach based on transaction pool data from measurement campaigns, and confirm our analytical findings therein.
- We show that *SREP* can have significantly lower bandwidth overhead than *MempoolSync*.

The rest of this paper is organized as follows. In Section 2, we provide some background and a brief summary of related work. In Section 3, we introduce *SREP*. In Section 4, we analyze the properties of *SREP* in large-scale networks. In Section 5, we extend our analysis to dynamic networks. We validate our analytical findings through simulations in Section 6, and compare *SREP* with a transaction pool synchronization approach from the literature in Section 6.3. Finally, we conclude with proposals for future work in Section 7.

2 | BACKGROUND AND RELATED WORK

Our *SREP* algorithm explicitly tackles the problem of distributed network-wide synchronization of unconfirmed transactions — *transaction pools*[12]. To achieve its goals, *SREP* relies on *communication-efficient* solutions to the *set reconciliation* problem[13], which seeks to identify the differences between two remote data sets S_A and S_B with minimum communication. Communication-efficient solutions to this problem are able to solve it by exchanging messages of size proportional to the number of *mutual differences*

$$S_A \oplus S_B = (S_A \setminus S_B) \cup (S_B \setminus S_A).$$

In fact, there has been several communication-efficient set reconciliation algorithms proposed in the literature including Characteristic Polynomial Interpolation[14] (CPI), BCH codes[15], and Invertible Bloom Lookup Tables (IBLT)[16,17,18]. The algorithm CPI incurs a communication cost *equal* to the number of mutual differences plus a small constant (even for very large sets!), which makes it nearly *optimal* in communication[14]. On the other hand, IBLT-based solutions typically offer better *computational* complexity at the cost of increasing their communication cost. To further reduce this communication overhead, Lázaro and Matuz[18] have recently proposed an IBLT-based solution that brings the communication cost closer to that of CPI while keeping the computational complexity low. The latest IBLT construction of Lázaro and Matuz[18] is based on Eppstein *et al.*[17] and has both communication complexity and computation (decoding) complexity that are *linear* in the number of mutual differences.

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.*[19] and Gao *et al.*[20] independently verified that the Ethereum network exhibits the "small-world" property. Recently, Shahsavari *et al.*[21] used a random graph model to simulate the Bitcoin network and Ma *et al.*[22] proposed a topology generation based on the Watts-Strogatz[23] random graph model to capture the Bitcoin network in their *CBlockSim* simulator. To analyze *SREP*'s performance in dynamic (temporal) networks[24], we consider a network model that can be viewed as a specialization of Kuhn-Oshman's *evolving graphs*[25].

3 | THE 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[14] or IBLT[18]). Given the local transaction pool as a set of globally unique identifiers[26], *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.

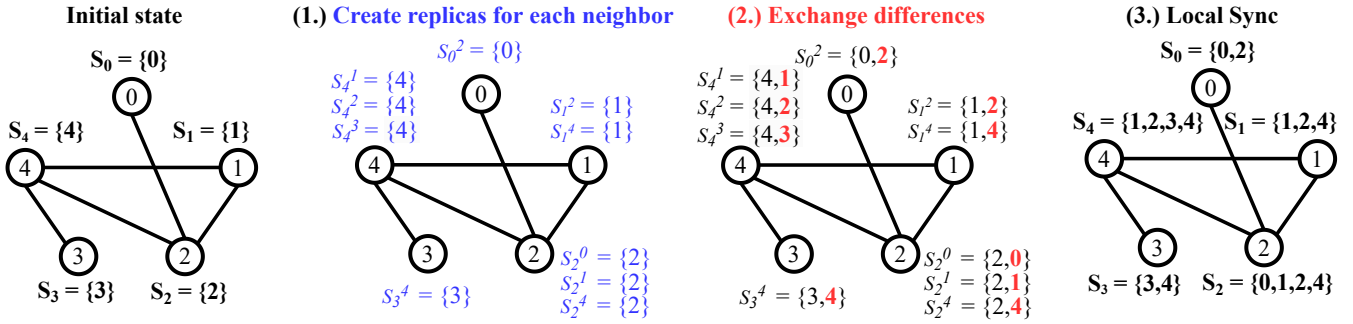

FIGURE 1 One iteration of *SREP* on a tractably small network.

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** ▷ Network Sync with all neighbors
4: $S_n^i \leftarrow S_n$
5: **Do in parallel** ▷ Syncing with neighbors is non-blocking
6: **Begin**
7: $d_{in} \leftarrow$ **Sync** (S_n^i, S_i)
8: $S_n^i \leftarrow S_n^i \cup d_{in}$
9: **End**
10: **for** $i \in G[n]$ **do** ▷ Local Sync
11: $S_n^i \setminus S_n \leftarrow$ **Sync** (S_n, S_n^i)
12: $S_n \leftarrow S_n \cup (S_n^i \setminus S_n)$

3.1 | 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 2 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[13] 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. For this purpose, the neighbor threads can utilize a write lock to coordinate the concurrent writes to the underlying data set. Upon completion of the network sync, each thread first acquires the write lock, updates the underlying data set, and releases the lock. The order in which the threads acquire the lock does not matter, because the set union operation is commutative and associative. 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.

$G = (V, E)$	Network of $\|E\|$ edges and $\|V\|$ nodes
S_n	Transaction pool at node $n \in \{0..\|V\| - 1\}$
$d_{ij} = S_i \setminus S_j$	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 1 Summary of notation.

4 | PERFORMANCE ANALYSIS IN STATIC NETWORKS

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 its performance. By successively relaxing these assumptions, we arrive at the final version of *SREP*.

Definition 1: The *communication cost C* of a transaction pool synchronization algorithm is the size of the messages it transmits over the network.

As transaction pools typically hold transaction identifiers with fixed size, we express the communication cost as the size of messages divided by that fixed size. 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*. Throughout the analysis, we assume that the total actual time taken by a primal sync is dominated by the transmission time[27]. Table 1 summarizes the notation we use in this work.

4.1 | Network Model

Our network model consists of a topology (*i.e.,* the underlying graph connecting nodes in the network) and the states of transaction pools residing at the nodes of the network.

4.1.1 | Topology

Watts-Strogatz[23] random graphs allow us to describe a wide range of realistic blockchain network topologies reasonably well[20,19,28,22]. 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[23].

For instance, each Bitcoin node selects 8 random neighbors upon joining the network[29,30,31], which has been shown to yield an unstructured random graph[21]. 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 Camellia distributed hash table (DHT)[32], and yields a network with more structure[20]. Notwithstanding this, multiple recent measurement results have independently confirmed that the generated network exhibits the "small world" property and fits the Watts-Strogatz model[20,19,28]. 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[33], as illustrated in Fig. 2.

4.1.2 | Transaction Pools

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 $i \in \{0...|V| - 1\}$,
- *sizes*: *sizes vector*. A $|V|$-size vector where elements are drawn from $\mathcal{S}$,

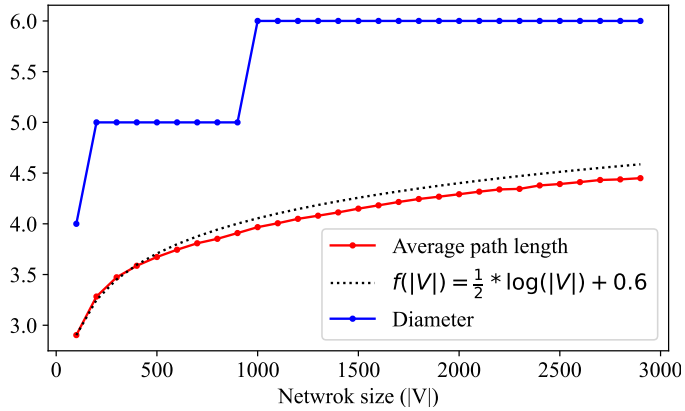

FIGURE 2 The small world property in random graphs generated through Watts-Strogatz model with $k = \overline{deg} = 19$ and rewire probability $p = 0.24$.

- $\mathcal{P}$: *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 $u \geq |V|$.

4.2 | 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.
1: while network is not fully synchronized do
2:    for n ← 0 to |V| − 1 do
3:       neighbors ← sort(G[n])
4:       for i ∈ neighbors do
5:          d_in ← Sync (S_n, S_i)
6:          d_ni ← Sync (S_i, S_n)
7:          S_n ← S_n ∪ d_in
8:          S_i ← S_i ∪ d_ni
```

4.3 | 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 2, this is a reasonable assumption to make. For instance, Eppstein *et al.* proposed a primal sync with both communication and time complexities that are linear in the number of mutual differences [17,18].

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

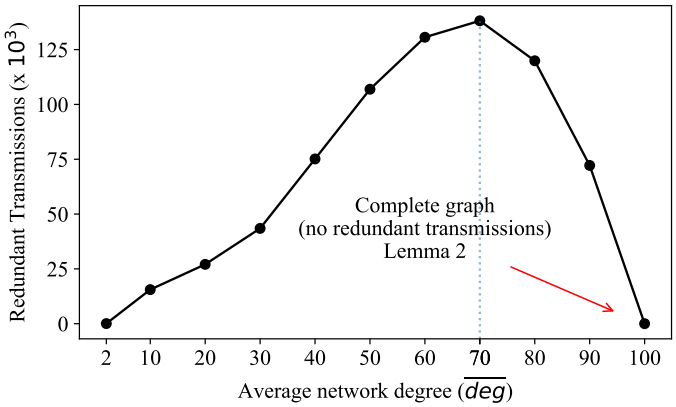

FIGURE 3 Amount of redundant transmissions in *EP-SREP* over a network of 100 nodes ($p = 0.24$).

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 $|V|$[33]. 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.,* Bitcoin and Ethereum[30,31]).

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. ■

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. 3 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.

4.4 | 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 the i-th iteration of *SREP*, we transmit exactly as many 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*.

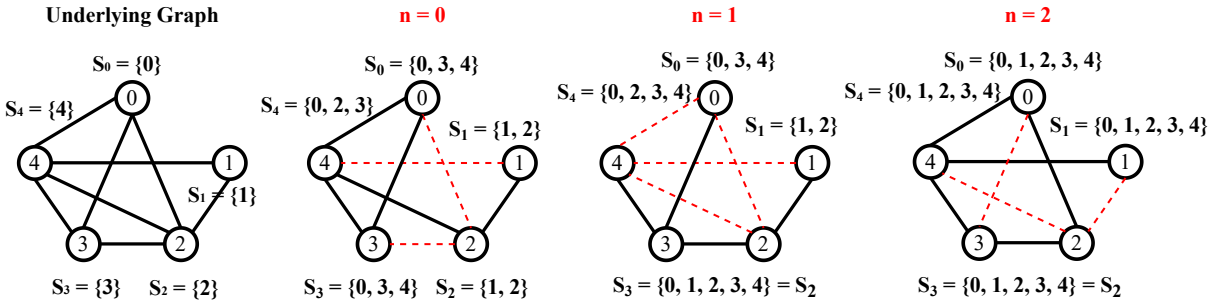

FIGURE 4 *SREP* over a time-varying graph. Red dashed edges are absent in the corresponding time intervals.

Lemma 5: In *SREP* over a connected network $G = (V, E)$ with the given initial pool assignment A and the maximum difference among any two pools $\overline{d}$:

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

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.

5 | PERFORMANCE ANALYSIS IN DYNAMIC NETWORKS

Blockchain networks often involve non-persistent connections among nodes[7]. Therefore, in this section we analyze the performance of *SREP* in dynamic (time-varying) networks, particularly focusing on the total time to synchronize ($T_{100\%}$). Due to the time-varying nature of dynamic networks, $T_{100\%}$ is now a random variable. Hence, we are interested in statistics related to this random variable, such as the expectation (mean) $\mathbb{E}[T_{100\%}]$.

We distinguish between two types of dynamic networks: slowly-varying and rapidly-varying. The case of slowly-varying networks is more straightforward. In such networks, edges remain active for long periods of time relative to the time needed to synchronize an entire network. Therefore, these networks appear static from the perspective of *SREP*. In other words, the analysis from Section 4 applies as long as the network is connected.

In rapidly-varying networks, edges appear and disappear throughout the entire synchronization process. Therefore, the degree of nodes may change over time and the analysis from Section 4 does not directly apply. The rest of this section focuses on the analysis of such rapidly-varying networks. We first formally define our model of time-varying graph and analyze the special case of a time-varying linear topology (simply referred to as a *line*). By the same token as in the proof of Theorem 4, our analysis assumes *EP-SREP*, meaning that each transaction pool initially holds only one element (*e.g.,* the one whose identifier equals the index of the node). We show an example in Fig. 4. Next, we leverage the analysis for linear topologies to provide bounds for *general* time-varying graphs. A summary of notation relevant to this section appears in Table 2.

5.1 | Network Model

To analyze the performance of *SREP* in dynamic networks, we use the transaction pools model from Section 4.1.2. However, instead of using a static topology as in Section 4.1.1, we consider *connected time-varying graphs* defined as follows.

Definition 7: A connected time-varying graph is a discrete-time graph $G = (V, E)$ where in each time slot $t = \{0, 1, 2, \ldots\}$, each edge in the graph is active (up) with some probability p_{con}, independently of other edges and other time slots. If an edge is active during a certain time-slot, the two vertices connected to the edge can fully synchronize their pools. Note that we assume that the *underlying graph* $G = (V, E)$ is connected (that is, a path exists between any two vertices when all the edges are active).

$n = \|V\|$	Number of nodes in the network
s_i	Element possessed by S_i at the beginning of synchronization
$T_{100\%}^{line_n}$	Time to sync a n-nodes time-varying line
$\tau_{j\to i}$	Time for transaction pool S_i to acquire element s_j
τ_{v_i}	Time for node v_i to acquire all elements in an n-nodes network

TABLE 2 Summary of notation for dynamic networks.

Note that by Definition 7, the time until a connection is established between two adjacent nodes in the underlying graph follows a geometric random distribution with parameter p_{con}. Such time includes the time slot when connection is established, so the minimum value of these geometric distributed random variables is one time slot. As edges appear and disappear independently, these connection times are independent and identically distributed (i.i.d.) random variables over different edges and different time slots.

5.2 | *SREP* in Linear Time-Varying Topologies

Consider a time-varying line composed of n nodes denoted as $v_1...v_n$. For any pair of nodes v_i, v_j in a time-varying line, an element that reaches one node from the other node must have already reached all the nodes that lie in between v_i and v_j. We formalize this observation in the following lemma.

Lemma 6: For a time-varying line the following holds for all i, j, and k, such that $i \leq j < k$, and all elements s_j:

$$s_i \in S_k \implies s_j \in S_k.$$

Let τ_{v_i} denote the time till node v_i acquires all elements in an n-nodes network.

Lemma 7: For a n-node time-varying line denoted as $v_1...v_n$, the time to synchronize is

$$T_{100\%}^{line_n} = \max(\tau_{v_1}, \tau_{v_n}).$$

Proof: Using Lemma 6, by the time $s_n \in S_1$, v_1 has acquired all the elements. Analogously, by time τ_{v_1}, node v_i has acquired all the elements from the nodes with higher indices:

$$\{s_i, .., s_n\} \subset S_i, \text{ for all } i \text{ s.t. } 1 \leq i \leq n.$$

Similarly, by τ_{v_n}, node v_i has acquired all the elements from the nodes with lower indices :

$$\{s_1, .., s_i\} \subset S_i, \text{ for all } i \text{ s.t. } 1 \leq i \leq n.$$

Thus, by time $\max(\tau_{v_1}, \tau_{v_n})$ all nodes have acquired all the elements. ■

Consider node v_m that is in the middle of an n-node time-varying line, such that $m = \frac{n}{2}$ for even n and $m = \frac{n+1}{2}$ for odd n. The following lemma states that node v_m will be synchronized once it gets the elements from node v_1 and node v_n. The time that it takes is the maximum between the times for transaction pool S_m to get element s_1 (denoted $\tau_{1\to m}$) and the time to get element s_n (denoted $\tau_{n\to m}$). This lemma will be used in the next subsection to provide a tight bound on the expected time to synchronize a line.

Lemma 8: The time to synchronize node v_m is

$$\tau_{v_m} = \max(\tau_{1\to m}, \tau_{n\to m}). \tag{1}$$

Proof: At time $\tau_{1\to m}$, node v_m has already acquired s_1. By Lemma 6, any elements residing in any S_j such that $1 \leq j < m$ will also reside in S_m by time $\tau_{1\to m}$. Similarly, at time $\tau_{n\to m}$ all elements residing in any S_j such that $m < j \leq n$ will reside in S_m as well. This means that at time $\max(\tau_{1\to m}, \tau_{n\to m})$, transaction pool S_m has acquired all elements $s_1, ..s_n$. Therefore, τ_{v_m} is bounded

from above by $\max(\tau_{1\to m}, \tau_{n\to m})$. On the other hand, τ_{v_m} cannot be less than $\max(\tau_{1\to m}, \tau_{n\to m})$ because that would mean that v_m would not have acquired either s_1 or s_n. Therefore, this is the lower bound as well. ■

The results in the next subsection will rely on a concept known as *stochastic ordering*[34]. Specifically, consider two random variables Y and Z. We say that Y is stochastically smaller or equal to Z, denoted $Y \leq_{\text{st}} Z$, if for all $x \in (-\infty, \infty)$ we have $\Pr(Y > x) \leq \Pr(Z > x)$. A direct consequence of this definition is the following lemma[34]:

Lemma 9: If two random variables X and Y satisfies the stochastic order $Y \leq_{\text{st}} Z$, then

$$\mathbb{E}[Y] \leq \mathbb{E}[Z].$$

5.3 | Expected Time to Sync Time-Varying Lines

We now turn to the analysis of the synchronization time in a time-varying line. We first consider the middle node and then extend the result to all the nodes on the line.

The next theorem shows that the expected time to synchronize the middle node of an n-node time-varying line is $\mathcal{O}(n)$.

Theorem 5: Consider a n-node time-varying line, then $\mathbb{E}[\tau_{v_m}] = \mathcal{O}(n)$.

Proof: In a n-node time-varying line, element s_1 will propagate to the middle node v_m as follows: first the element is stuck in vertex v_1 till a time slot when the edge between v_1 and v_2 is active. Then the process between v_2 and v_3 and so on till vertex v_{m-1} and vertex v_m. The time for an edge to become active is a geometric random variable (see Definition 7), independent of other events. Hence, the time slots that s_1 is stuck follows the sum of $m-1$ independent geometric random variables which corresponds to a negative-binomial distribution with parameters $\gamma = m-1$ and $p = p_{con}$ starting from $m-1$, denoted as:

$$\tau_{1\to m} - (m-1) \sim NB(m-1, p_{con}). \tag{2}$$

Similarly, $\tau_{n\to m}$ is a negative-binomial random variable starting from $n-m$, that is

$$\tau_{n\to m} - (n-m) \sim NB(n-m, p_{con}). \tag{3}$$

Furthermore $\tau_{n\to m}$ and $\tau_{n\to m}$ are *independent* because they use a different set of independently appearing edges.

Using Lemma 8, τ_{v_m} is the maximum between the two independent random variables, which implies that τ_{v_m} is stochastically smaller than the sum of $\tau_{1\to m}$ and $\tau_{n\to m}$. Therefore, by Lemma 9 we have:

$$\mathbb{E}[\tau_{v_m}] \leq \mathbb{E}[\tau_{1\to m}] + \mathbb{E}[\tau_{n\to m}].$$

From Eq. (2) and Eq. (3), we have:

$$\begin{aligned}\mathbb{E}[\tau_{v_m}] &\leq \mathbb{E}[\tau_{1\to m}] + \mathbb{E}[\tau_{n\to m}] \\ &= \frac{(m-1)(1-p_{con})}{p_{con}} + (m-1) \\ &+ \frac{(n-m)(1-p_{con})}{p_{con}} + (n-m) \\ &= \frac{n-1}{p_{con}}\end{aligned} \tag{4}$$

Therefore, τ_{v_m} is $\mathcal{O}(n)$. ■

Next, we derive an analytical expression for the expected time to synchronize the middle node v_m in a n-node time-varying line. The expression depends on whether n is odd or even:

- If n is odd, set $m = \frac{n+1}{2}$. As a result, both $\tau_{1\to m}$ and $\tau_{n\to m}$ conform to negative-binomial distributions with parameters $\gamma = \frac{n-1}{2}$ and $p = p_{con}$ starting from $m-1$. Their cumulative distribution can be expressed in terms of the *regularized incomplete beta function*[35]:

$$Pr(X \leq x) = I_{p_{con}}(\frac{n-1}{2}, x-m+2), \tag{5}$$

where the regularized incomplete beta function is defined as:

$$I_x(a,b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1}\,dt}{\int_0^1 t^{a-1}(1-t)^{b-1}\,dt}.$$

Using Eq. (1), we can express expectation of τ_{v_m} in terms of the maximum of two negative binomial random variables:

$$\mathbb{E}[\tau_{v_m}] = \mathbb{E}[\max(\tau_{1\to m}, \tau_{n\to m})] \tag{6}$$

Because $\tau_{1\to m}$ and $\tau_{n\to m}$ are independent, we can express the cumulative distribution function of $\max(\tau_{1\to m}, \tau_{n\to m})$ using Eq. (5):

$$\begin{aligned} &Pr(\max(\tau_{1\to m}, \tau_{n\to m}) \le x) \\ &= Pr(\tau_{1\to m} \le x) \cdot Pr(\tau_{n\to m} \le x) \\ &= (I_{p_{con}}(\frac{n-1}{2}, x-m+2))^2. \end{aligned} \tag{7}$$

Combining cumulative distribution function from Eq. (7) with Eq. (6), we express $\mathbb{E}[\tau_{v_m}]$ as:

$$\mathbb{E}[\tau_{v_m}] = \int_0^\infty (1 - (I_{p_{con}}(\frac{n-1}{2}, x - \frac{n-3}{2}))^2)\,dx. \tag{8}$$

- If n is even, we set $m = \frac{n}{2}$. Now $\tau_{1\to m}$ and $\tau_{n\to m}$ conform to the negative binomial distribution with different parameters $\gamma_{1\to m} = \frac{n}{2} - 1$ and $\gamma_{n\to m} = \frac{n}{2}$, and the same $p = p_{con}$. $\tau_{1\to m}$ starts from $\frac{n}{2} - 1$. $\tau_{n\to m}$ starts from $\frac{n}{2}$.
  Similar to Eq. (7) cumulative distribution function of $\max(\tau_{1\to m}, \tau_{n\to m})$ can be given from Eq. (5):

$$\begin{aligned} &Pr(\max(\tau_{1\to m}, \tau_{n\to m}) \le x) \\ &= Pr(\tau_{1\to m} \le x) \cdot Pr(\tau_{n\to m} \le x) \\ &= I_{p_{con}}(\frac{n}{2} - 1, x - \frac{n}{2} + 2) \cdot I_{p_{con}}(\frac{n}{2}, x - \frac{n}{2} + 1). \end{aligned} \tag{9}$$

From Eq. (9):

$$\begin{aligned} &\mathbb{E}[\tau_{v_m}] = \\ &\int_0^\infty (1 - I_{p_{con}}(\frac{n}{2} - 1, x - \frac{n}{2} + 2) \\ &\cdot I_{p_{con}}(\frac{n}{2}, x - \frac{n}{2} + 1))\,dx. \end{aligned} \tag{10}$$

The next theorem provides a bound on the expected time to synchronize a time-varying line. In Section 6.5, we will numerically show that this bound is quite tight.

Theorem 6: For a n-nodes time-varying line, the expected time to synchronize is:

$$\mathbb{E}[T^{line_n}_{100\%}] \le 2 * \mathbb{E}[\tau_{v_m}], \tag{11}$$

where $\mathbb{E}[\tau_{v_m}]$ is given by Eq. (8) when n is odd and by Eq. (10) when n is even.

Proof: Using Lemma 8, by time τ_{v_m}, both s_1 and s_n have already been propagated to S_m. It is either that one element arrived before the other or they reached S_m at the same time. Without loss of generality, we assume that $\tau_{1\to m} \le \tau_{n\to m}$.

By the time τ_{v_m}, elements $s_1, ..., s_m$ have reached all nodes on the shorter side of v_m (*i.e.,* $v_1, .., v_{m-1}$) and for *some* j such that $m \le j \le n$, we have:

$$\{s_1, .., s_n\} \subset S_j.$$

Some other $v_{j'}$ such that $j < j' \le n$ have not yet acquired s_1. It is easy to see that the time that has left until the full sync is reached is:

$$T^{line_k}_{100\%} - \tau_{v_m} = \max(\tau_{j\to n}, \tau_{m\to 1}) \tag{12}$$

Because edges between v_1 and v_m do not overlap with edges between v_j and v_n, we have that $\tau_{j\to n}$ and $\tau_{m\to 1}$ are two independent random variables conforming to negative-binomial distribution $NB(\gamma, p)$ with different γ parameters. Note that $\tau_{1\to m}$ and $\tau_{m\to 1}$ are not identical but they follow the *same* distribution, that is

$$Pr(\tau_{1\to m} > x) = Pr(\tau_{m\to 1} > x) \tag{13}$$

Note also that $\tau_{n\to m}$ and $\tau_{j\to n}$ are both negative-random random variables but with different parameters:

$$\begin{aligned} \tau_{n\to m} &\sim NB(n-m, p_{con}) \\ \tau_{j\to n} &\sim NB(n-j, p_{con}) \end{aligned} \tag{14}$$

Since $n-j \leq n-m$, we establish stochastic ordering between the two random variables from Eq. (14):

$$\begin{aligned} &Pr(\tau_{j\to n} > x) \leq Pr(\tau_{n\to m} > x) \\ &\implies \tau_{j\to n} \leq_{\text{st}} \tau_{n\to m} \end{aligned} \tag{15}$$

Combining Eq. (13) and Eq. (15), we compare the cumulative distribution functions of $\max(\tau_{1\to m}, \tau_{n\to m})$ and $\max(\tau_{j\to n}, \tau_{m\to 1})$:

$$\begin{aligned} &Pr(\max(\tau_{j\to n}, \tau_{m\to 1}) > x) \\ &\qquad \leq Pr(\max(\tau_{1\to m}, \tau_{n\to m}) > x) \end{aligned} \tag{16}$$

Hence, by Lemma 8 we have:

$$\begin{aligned} \mathbb{E}[\max(\tau_{j\to n}, \tau_{m\to 1})] &\leq \mathbb{E}[\max(\tau_{1\to m}, \tau_{n\to m})] \\ &\leq \mathbb{E}[\tau_{v_m}] \end{aligned} \tag{17}$$

We then use Eq. (17) and Eq. (12):

$$\begin{aligned} \mathbb{E}[T^{line_n}_{100\%}] &= \mathbb{E}[\tau_{v_m}] + \mathbb{E}[\max(\tau_{j\to n}, \tau_{m\to 1})] \\ &\leq 2\,\mathbb{E}[\tau_{v_m}] \end{aligned} \tag{18}$$

Finally, we combine Eq. (18), Eq. (8), and Eq. (10), to get the expression from Eq. (11). ■

As a corollary of Theorem 5 and Theorem 6, we get that the expected time to sync a n-node time-varying line is $\mathcal{O}(n)$.

5.4 | *SREP* in General Time-Varying Topologies

We now generalize our analysis for time-varying line to time-varying graphs from Definition 7. For the purpose of analysis, we define the minimum spanning tree in time-varying graphs.

Definition 8: A spanning tree of time-varying graph $G = (V, E)$ is a spanning tree in its underlying graph.

Definition 9: Given time-varying graph $G = (V, E)$ with the corresponding time to synchronize $T_{100\%}$, time $T^{tree}_{100\%}$ is the time to synchronize its spanning tree.

Next, we bound the expected time to sync a time-varying graph using its spanning tree.

Lemma 10: Expected time to sync a time-varying spanning tree is the upper bound on the expected time to sync the corresponding time-varying graph:

$$\mathbb{E}[T_{100\%}] \leq \mathbb{E}[T^{tree}_{100\%}].$$

Proof: By the definition of full sync, the total time to sync a time-varying graph is the maximum among syncing all its nodes:

$$T_{100\%} = \max(\tau_{v_1}, \tau_{v_2}, .., \tau_{v_n}). \tag{19}$$

For any node v_i, such that $1 \leq i \leq n$, τ_{v_i} is bounded by the longest time for an element to reach transaction pool S_i:

$$\tau_{v_i} = \max(\tau_{1\to i}, \tau_{2\to i}, .., \tau_{n\to i}). \tag{20}$$

Note that the spanning tree has $n-1$ edges, which is no more than the corresponding graph. Hence, the time for any element to reach node v_i in spanning tree (τ'), is stochastically larger than or equal to what it is in the corresponding graph (τ). In other words, for all $1 \leq i \leq n$, the following holds:

$$\begin{aligned} &Pr(\tau_{1\to i} > x) \leq Pr(\tau'_{1\to i} > x) \\ &\implies \tau_{1\to i} \leq_{\text{st}} \tau'_{1\to i}. \end{aligned} \tag{21}$$

Combining Eq. (21) and Eq. (20), we establish the stochastic order between τ_{v_i} and τ'_{v_i}:

$$\tau_{v_i} \leq_{\text{st}} \tau'_{v_i}. \tag{22}$$

We conclude the proof combining Eq. (19) and Eq. (22):

$$T_{100\%} \leq_{\text{st}} T^{tree}_{100\%} \implies \mathbb{E}[T_{100\%}] \leq \mathbb{E}[T^{tree}_{100\%}].$$

■

Theorem 7: The expected time to sync n-nodes time-varying graph is bounded from above by the expected time to sync a n-nodes time-varying line.

$$\mathbb{E}[T_{100\%}] \leq \mathbb{E}[T^{line_n}_{100\%}]. \tag{23}$$

Proof: Consider the spanning tree of the time-varying graph and find a random node v^*_m that splits the tree in two. Left subtree has n_l nodes, right subtree has n_r nodes. Meaning, $n_l + n_r = n - 1$.

We denote the time for v^*_m to acquire all the elements from the left and right subtrees as τ_{left} and τ_{right}, respectively.

Then, the time for v^*_m to complete synchronization is:

$$\tau_{v^*_m} = \max(\tau_{left}, \tau_{right}). \tag{24}$$

Note that τ_{left} and τ_{right} use different edges, therefore they are independent random variables. τ_{left} is bounded by the maximum time for any elements in the left subtree to reach v^*_m. The longest distance between a node in the left subtree and the split node v^*_m is n_l. Note that longest distance appeares only when the subtrees are structured into time-varying lines. Therefore, τ_{left} is stochastically smaller than $\tau_{1\to n_l+1}$ in a time-varying line. Analogously, τ_{right} is stochastically smaller than $\tau_{1\to n_r+1}$ in a time-varying line. In other words, it takes the longest in expectation to sync both the left and the right sides of the spanning tree when they are lines. Hence, we have:

$$\mathbb{E}[T^{tree}_{100\%}] \leq \mathbb{E}[T^{line_n}_{100\%}].$$

Using Lemma 10, we get the expression from Eq. (23) . ■

6 | SIMULATIONS

To validate our analytical findings about *SREP*, we construct an event-based simulator called *SREPSim*[36] that shares the topology generation procedure with *CBlockSim* of Ma *et al.*[22] and adds the other parameters of our network model described in Section 4.1.

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* in static graphs. We then compare the overall communication cost of *SREP* with a similar approach from the literature. Additionally, we present a *SREPSim* optimization that allows for easy *SREP* communication cost calculations over large-scale static networks. Finally, we validate the analytical properties of *SREP* in time-varying graphs.

6.1 | Configuring Network Model Parameters

Unlike the simulation approaches from the literature (*e.g., SimBlock*[37]), 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

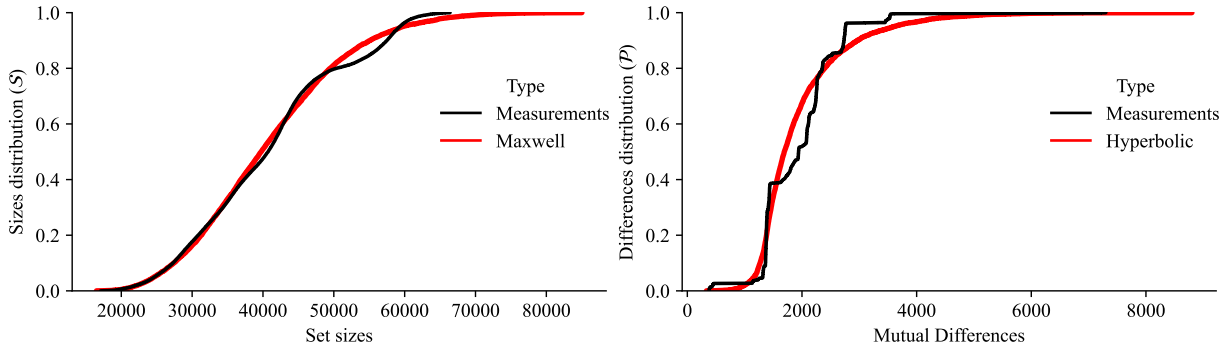

FIGURE 5 Empirical distributions of transaction pool sizes $\mathcal{S}$ for two adjacent Bitcoin nodes (left) and their mutual differences $\mathcal{P}$ (right). Best distribution fits in red (using Error Sum of Squares).

the network using the measurement software such as *log-to-file* of Imtiaz *et al.* [38,39]. 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. 5 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[‡]. 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 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. 6, $\psi = 0.35$ 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.

$u \leftarrow \lceil\ \psi\, \mathbb{E}[\mathcal{S}]\ \rceil$
$\mathcal{U}\{0, u-1\}$ ▷ Instantiate 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)

6.2 | *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.

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

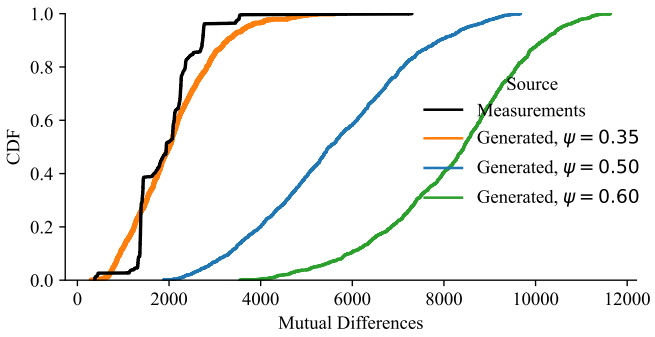

FIGURE 6 Empirical differences distribution for two adjacent Bitcoin nodes versus the differences distribution generated by Procedure 1 for various ψ. Watts-Strogatz network with 100 nodes ($\overline{deg}$ = 19 and p = 0.24).

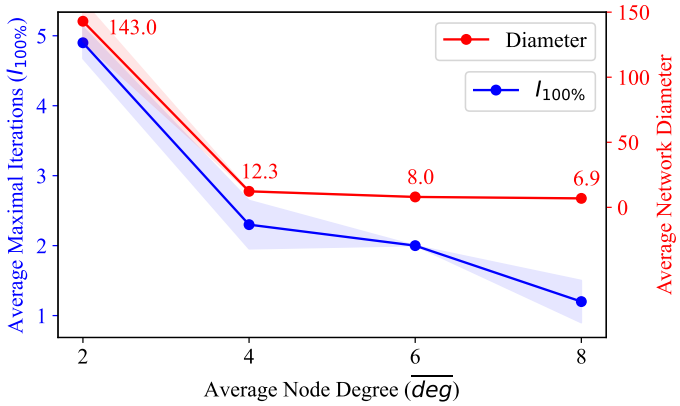

FIGURE 7 Maximal number of *SREP* iterations at any node ($I_{100\%}$) bounded by the network diameter for Watts-Strogatz graphs with 1000 nodes (p = 0.24). 95% confidence intervals.

In Fig. 7, 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. 8, we plot the communication 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. 3). 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).

6.3 | 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[7]. Here we describe this protocol and compare its communication efficiency with our newly proposed *SREP* through simulations.

As pointed out in[7], 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*[3,6]. Thus, the goal of *MempoolSync* is to supply the nodes with potentially missing transactions, and it does so through an *ancestor score*-based heuristics[40]. 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:

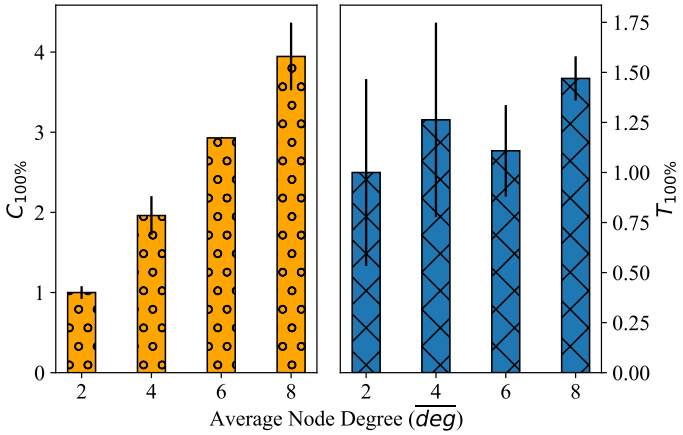

FIGURE 8 Relative communication cost ($C_{100\%}$) and time to fully synchronize the network ($T_{100\%}$). Network with 1000 nodes (p = 0.24).

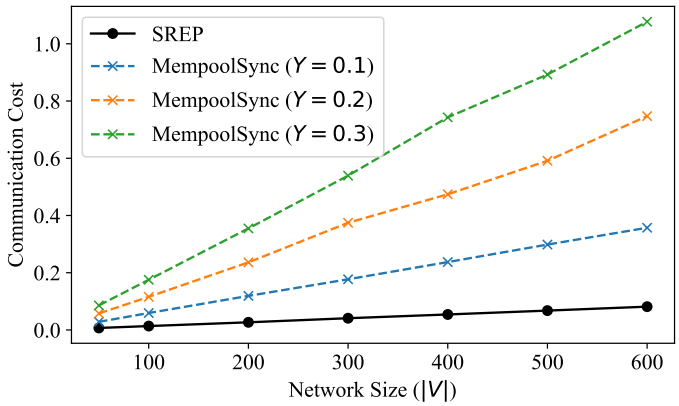

FIGURE 9 Normalized overall communication cost of *SREP* ($C_{100\%}$) and *MempoolSync* as a function of network size. Data from Section 6.1. *DefTXtoSync* = 1000. Y is the *MempoolSync* heuristic constant.

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[7].

In Fig. 9, 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. 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.

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

TABLE 3 *SREP* over a 10,000 nodes network. $p = 0.24$.

6.4 | 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[22]. 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)[30,31].

In Table 3, 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[26]. All simulations complete in tens of minutes.

6.5 | Sync Time in Time-Varying Networks

Here we validate our analytical findings about connected time-varying graphs. Following the order of our analysis, we first simulate time-varying lines and then proceed to graphs. For the purpose of simulation, we extend *SREPSim*'s topology generation module with connected time-varying graphs from Definition 7. To get the initial underlying graphs, we rely on a topology generation procedure similar to the one mentioned at the beginning of this section. We assign connection probabilities independently to each edge to get the final connected time-varying graph.

In Fig. 10, we plot the time to sync time-varying lines against the predicted upper bound. Each simulation is repeated 1000 times and the 95% confidence intervals are reported. To better illustrate the dynamics of the network, we set the connection probability as low as $p_{con} = 0.05$, which means that the average wait time for an edge to get active is 20 time slots. Our simulations confirm the $\mathcal{O}(n)$ upper bound predicted by Theorem 5 and Theorem 6.

In Fig. 11, we analyze the effect of connection probability p_{con} to the sync time. We vary p_{con} over $\{0.05, 0.1, 0.15\}$ and consider time-varying lines ranging from 2 to 20 nodes. As in the previous experiments, we repeat each simulation 1000 times and report 95% confidence intervals. Higher connection probabilities mean that the wait times for connections to get active are smaller, which makes the network change more rapidly. Note that our analytical results from Theorem 6 get tighter as the network changes more rapidly.

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
        $M[i][j] \leftarrow |S_i \oplus S_j|$
$C_{100\%} \leftarrow 0$
$I_{100\%} \leftarrow 0$
$M \leftarrow$ 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)

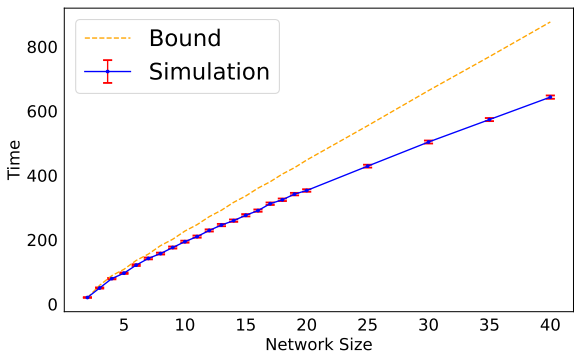

FIGURE 10 Time to sync time-varying lines. Analytical upper bound versus simulations. Connection probability $p_{con} = 0.05$.

In Fig. 12, we plot the sync time in time-varying graphs and time-varying lines with the same number of nodes. For each network size, we generate 1000 time-varying graphs and report the mean time among all 1000 graphs with the corresponding 95% confidence intervals. As predicted by Theorem 7, the time to sync n-node time-varying lines is no better than the time to sync n-node time-varying graphs. As the graph gets larger, the difference in the number of connections in the graph and its corresponding spanning tree gets larger, which widens the gap between our analysis from the proof of Theorem 7 and simulations. In Fig. 13, we plot, on a logarithmic scale, the sync time in time-varying graphs of sizes ranging from 50 to 400 nodes.

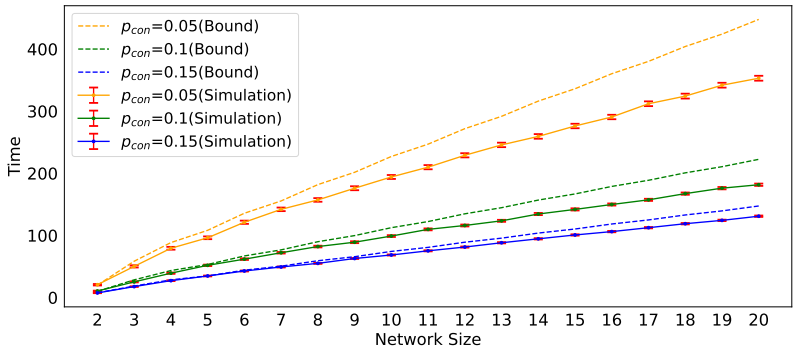

FIGURE 11 Time to sync time-varying lines for p_{con} ranging over $\{0.05, 0.1, 0.15\}$. Analytical upper bound versus simulations for each p_{con}.

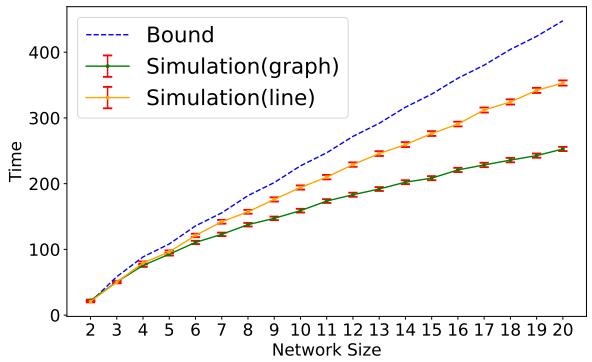

FIGURE 12 Time to sync time-varying graphs and lines with equal number of nodes. Analysis versus simulations. Connection probability $p_{con} = 0.05$.

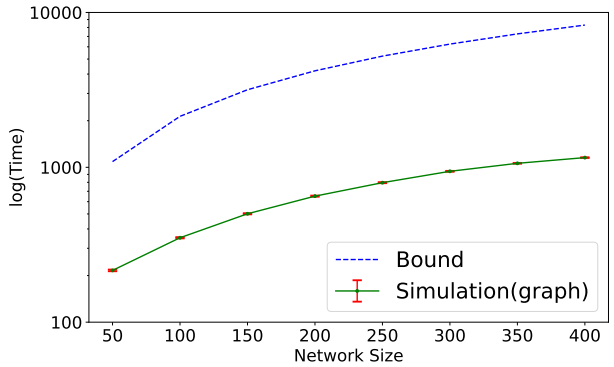

FIGURE 13 Time to sync *larger* time-varying graphs. Analysis versus simulations. Connection probability $p_{con} = 0.05$.

7 | 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 the network size for "small-world" networks, which reasonably model large-scale blockchains). In time-varying networks with constant edge connection probability, *SREP* completes in expected time that is linear in the size of the network.

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[41,42] 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 $(\cap_i S_i)$[41]. We also propose considering very frequently changing time-varying graphs with high probabilities of primal sync interruption. In such scenarios, particularly interesting are primal syncs that support partial and prioritized synchronization[43]. We further propose analyzing the benefits of SREP-like protocols in synchronizing unconfirmed transactions in directed acyclic graph-based[44] distributed ledger technologies.

Finally, we emphasize some implementation aspects. Given that the nodes in the blockchain network may not keep track of the overall network size, it may be necessary to use techniques from the network measurement research[45,20,19,28] to develop a network size estimate. Alternatively, one may consider applying a combination of linear sketches and network coding similar as Mitzenmacher and Pagh[41] to ascertain the convergence of the sync without requiring a network size estimate. Another implementation concern is the potential arrival of new blocks during the sync. In this case, one can utilize efficient probabilistic filers[46,47] to avoid reintroducing transactions that were included in the recently arrived blocks. Lastly, we note that one can choose between several available implementations of primal syncs depending on the memory and compute constraints of the blockchain nodes. For a fine-grained performance analysis of primal syncs and their implementations, we refer the reader to our previous work[27].

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. Ozisik AP, Andresen G, Levine BN, Tapp D, Bissias G, Katkuri S. Graphene: Efficient Interactive Set Reconciliation Applied to Blockchain Propagation. In: SIGCOMM '19. Association for Computing Machinery 2019; New York, NY, USA:303–317
4. Ding X, Zhao L, Luo L, Xie J, Guo D, Li J. Gauze: Enabling Communication-Friendly Block Synchronization with Cuckoo Filter. *Frontiers of Computer Science.* 2022;17(3):173403. doi: 10.1007/s11704-022-1685-5
5. Hu Z, Xiao Z. Dino: A Block Transmission Protocol with Low Bandwidth Consumption and Propagation Latency. In: IEEE. 2022:1319-1328
6. Imtiaz MA, Starobinski D, Trachtenberg A. Empirical Comparison of Block Relay Protocols. *IEEE Transactions on Network and Service Management.* 2022:1-1. doi: 10.1109/TNSM.2022.3195976
7. Imtiaz MA, Starobinski D, Trachtenberg A, Younis N. Churn in the Bitcoin Network. *IEEE Transactions on Network and Service Management.* 2021;18(2):1598-1615. doi: 10.1109/TNSM.2021.3050428
8. Imtiaz MA, Starobinski D, Trachtenberg A, Younis N. Churn in the Bitcoin Network: Characterization and Impact. In: IEEE. 2019:431-439
9. Motlagh SG, Mišić J, Mišić VB. Impact of Node Churn in the Bitcoin Network. *IEEE Transactions on Network Science and Engineering.* 2020;7(3):2104-2113. doi: 10.1109/TNSE.2020.2974739
10. Mišić J, Mišić VB, Chang X. On the Benefits of Compact Blocks in Bitcoin. In: IEEE. 2020:1-6
11. Zhao C, Wang T, Zhang S, Liew SC. HCB: Enabling Compact Block in Ethereum Network with Secondary Pool and Transaction Prediction. https://arxiv.org/abs/2212.13367; 2022.
12. Xiao Y, Zhang N, Lou W, Hou YT. A Survey of Distributed Consensus Protocols for Blockchain Networks. *IEEE Communications Surveys & Tutorials.* 2020;22(2):1432-1465. doi: 10.1109/COMST.2020.2969706
13. Minsky Y, Trachtenberg A. Practical set reconciliation. In: . 248. IEEE. 2002.
14. Minsky Y, Trachtenberg A, Zippel R. Set reconciliation with nearly optimal communication complexity. In: IEEE. 2001:232-. doi: https://doi.org/10.1109/ISIT.2001.936095
15. Dodis Y, Reyzin L, Smith A. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In: Cachin C, Camenisch JL., eds. *Advances in Cryptology - EUROCRYPT 2004*IACR. Springer Berlin Heidelberg 2004; Berlin, Heidelberg:523–540. ISBN: 978-3-540-24676-3.
16. Goodrich MT, Mitzenmacher M. Invertible bloom lookup tables. In: IEEE. 2011:792-799. doi: https://doi.org/10.1109/Allerton.2011.6120248
17. Eppstein D, Goodrich MT, Uyeda F, Varghese G. What's the Difference? Efficient Set Reconciliation without Prior Context. In: SIGCOMM '11. ACM. Association for Computing Machinery 2011; New York, NY, USA:218–229. doi: https://doi.org/10.1145/2018436.2018462
18. Lázaro F, Matuz B. A Rate-Compatible Solution to the Set Reconciliation Problem. *IEEE Transactions on Communications.* 2023;71(10):5769-5782. doi: 10.1109/TCOMM.2023.3296630
19. Wang T, Zhao C, Yang Q, Zhang S, Liew SC. Ethna: Analyzing the Underlying Peer-to-Peer Network of Ethereum Blockchain. *IEEE Transactions on Network Science and Engineering.* 2021;8(3):2131-2146. doi: 10.1109/TNSE.2021.3078181
20. Gao Y, Shi J, Wang X, Tan Q, Zhao C, Yin Z. Topology Measurement and Analysis on Ethereum P2P Network. In: IEEE. 2019:1-7

21. Shahsavari Y, Zhang K, Talhi C. A Theoretical Model for Block Propagation Analysis in Bitcoin Network. *IEEE Transactions on Engineering Management.* 2022;69(4):1459-1476. doi: 10.1109/TEM.2020.2989170
22. Ma X, Wu H, Xu D, Wolter K. CBlockSim: A Modular High-Performance Blockchain Simulator. In: IEEE. 2022:1-5
23. Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks. *Nature.* 1998;393(6684):440-442. doi: 10.1038/30918
24. Holme P, Saramäki J. Temporal networks. *Physics Reports.* 2012;519(3):97-125. Temporal Networksdoi: https://doi.org/10.1016/j.physrep.2012.03.001
25. Kuhn F, Oshman R. Dynamic Networks: Models and Algorithms. *SIGACT News.* 2011;42(1):82–96. doi: 10.1145/1959045.1959064
26. Delgado-Segura S, Pérez-Solà C, Navarro-Arribas G, Herrera-Joancomartí J. Analysis of the Bitcoin UTXO Set. In: Zohar A, Eyal I, Teague V, et al., eds. *Financial Cryptography and Data Security*Springer Berlin Heidelberg. Springer Berlin Heidelberg 2019; Berlin, Heidelberg:78–91.
27. Boškov N, Trachtenberg A, Starobinski D. GenSync: A New Framework for Benchmarking and Optimizing Reconciliation of Data. *IEEE Transactions on Network and Service Management.* 2022:1-1. doi: 10.1109/TNSM.2022.3164369
28. Kiffer L, Salman A, Levin D, Mislove A, Nita-Rotaru C. Under the Hood of the Ethereum Gossip Protocol. In: Springer Berlin Heidelberg. Springer-Verlag 2021; Berlin, Heidelberg:437–456
29. Bitcoin developers . Bitcoin referential implementation. https://github.com/bitcoin/bitcoin; 2022. (Accessed 2022-12-02).
30. Delgado-Segura S, Bakshi S, Pérez-Solà C, et al. TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions. In: Goldberg I, Moore T., eds. *Financial Cryptography and Data Security*Springer Berlin Heidelberg. Springer International Publishing 2019; Cham:550–566.
31. Grundmann M, Baumstark M, Hartenstein H. On the Peer Degree Distribution of the Bitcoin P2P Network. In: IEEE. 2022:1-5
32. Maymounkov P, Mazières D. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Druschel P, Kaashoek F, Rowstron A., eds. *Peer-to-Peer Systems*Springer Berlin Heidelberg. Springer Berlin Heidelberg 2002; Berlin, Heidelberg:53–65.
33. Chung F, Lu L. The diameter of sparse random graphs. *Advances in Applied Mathematics.* 2001;26(4):257–279.
34. Ross SM. *Stochastic processes*. John Wiley & Sons, 1995.
35. Olver F, Lozier D, Boisvert R, Clark C. *The NIST Handbook of Mathematical Functions*. Cambridge University Press, New York, NY, 2010.
36. Boškov N. SREPSim. http://www.github.com/nislab/SREPSim; . (Accessed 2023-02-02).
37. Banno R, Shudo K. Simulating a Blockchain Network with SimBlock. In: IEEE. 2019:3-4
38. Imtiaz MA, Starobinski D, Trachtenberg A. Characterizing Orphan Transactions in the Bitcoin Network. In: IEEE. 2020:1-9
39. Imtiaz MA, Starobinski D, Trachtenberg A. Investigating Orphan Transactions in the Bitcoin Network. *IEEE Transactions on Network and Service Management.* 2021;18(2):1718-1731. doi: 10.1109/TNSM.2021.3056949
40. Bitcoin developers . Ancestor Score Sorting. https://github.com/bitcoin/bitcoin/blob/master/src/txmempool.h; 2022. (Accessed 2022-12-02).
41. Mitzenmacher M, Pagh R. Simple multi-party set reconciliation. *Distributed Computing.* 2018;31(6):441-453. doi: 10.1007/s00446-017-0316-0
42. Boral A, Mitzenmacher M. Multi-party set reconciliation using characteristic polynomials. In: IEEE. 2014:1182-1187
43. Jin J, Si W, Starobinski D, Trachtenberg A. Prioritized data synchronization for disruption tolerant networks. In: IEEE. 2012:1-8. doi: https://doi.org/10.1109/MILCOM.2012.6415678
44. Dong Z, Zheng E, Choon Y, Zomaya AY. DAGBENCH: A Performance Evaluation Framework for DAG Distributed Ledgers. In: IEEE. 2019:264-271
45. Kim SK, Ma Z, Murali S, Mason J, Miller A, Bailey M. Measuring Ethereum Network Peers. In: IMC '18. Association for Computing Machinery 2018; New York, NY, USA:91–104
46. Tarkoma S, Rothenberg CE, Lagerspetz E. Theory and Practice of Bloom Filters for Distributed Systems. *IEEE Communications Surveys & Tutorials.* 2012;14(1):131-155. doi: 10.1109/SURV.2011.031611.00024
47. Fan B, Andersen DG, Kaminsky M, Mitzenmacher MD. Cuckoo Filter: Practically Better Than Bloom. In: CoNEXT '14. Association for Computing Machinery 2014; New York, NY, USA:75–88