**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
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.

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

<table>
<thead>
<tr>
<th>Algorithm 1: SREP Algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Network $G = (V, E)$ as adjacency list.</td>
</tr>
<tr>
<td><strong>At each node</strong> $n \in {0,</td>
</tr>
<tr>
<td><strong>Loop</strong></td>
</tr>
<tr>
<td><strong>for</strong> $i$ in $G[n]$ <strong>do</strong> // Neighbors of $n$</td>
</tr>
<tr>
<td>$S_n' \leftarrow S_n$: // Replicate data set</td>
</tr>
<tr>
<td><strong>Do in parallel</strong></td>
</tr>
<tr>
<td>// Network sync</td>
</tr>
<tr>
<td>$d_{in} \leftarrow$ Sync $(S_n, S_i)$;</td>
</tr>
<tr>
<td>$S_n' \leftarrow S_n' \cup d_{in}$;</td>
</tr>
<tr>
<td><strong>for</strong> $i$ in $G[n]$ <strong>do</strong></td>
</tr>
<tr>
<td>// Local sync</td>
</tr>
<tr>
<td>$S_n \leftarrow S_n \cap (S_n' \setminus S_n)$;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE I: Summary of notation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G = (V, E)$</td>
</tr>
<tr>
<td>$S_n$</td>
</tr>
<tr>
<td>$d_{ij} = S_i \setminus S_j$</td>
</tr>
<tr>
<td>$\text{deg}$</td>
</tr>
<tr>
<td>$t_n$</td>
</tr>
<tr>
<td>$T_x$</td>
</tr>
<tr>
<td>$\Sigma_x$</td>
</tr>
<tr>
<td>$C_x$</td>
</tr>
</tbody>
</table>
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 models 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 $S_0 \ldots S_{|V| - 1}$ where set $S_i$ represents the transaction pool at node $i$. We model the statistical properties of $A$ through the following pool parameters:

$S$: sizes distribution. A discrete random variable describing the sizes of transaction pools $S_i$ for $i \in \{0 \ldots |V| - 1\}$, $s$: sizes vector. A $|V|$-size vector where elements are drawn from $S$,

$P$: 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 $P$.

$U$: universe. A discrete random variable from which we draw transaction IDs. We choose $U\{0, u\}$ 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: $S = 1$, $P = 2$ , 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

$$C_{100\%}(G) = |V| \cdot (|V| - 1).$$
C. Elementary Parallel SREP (EP-SREP)

The main aim of the elementary parallel SREP is to relax (A1), (A4) and (A5). 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 (A7).

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 communications, 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: (1) 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 elements in the network, and (2) for each element, there cannot be more than $|V| - 1$ redundant communications, as there cannot be more than that much replicas at any node. Thus, there cannot be more than $|V|^2 \cdot (|V| - 1)$ redundant communications 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 $\text{dist}(u, v)$, the maximum number of iterations required for a full network synchronization is equal to the diameter of the network:

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

**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 $G = (E, V)$:

$$I_{100\%}(G) = 1$$ 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 (A6), there will be no more than $|E|$ syncs in each iteration.
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 $x$:

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

**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:

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

**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 $P$. 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:

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

In $i$th 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 $\mathcal{P}_{(n)}$:

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

$$\Sigma_{100\%} \leq I_{100\%}(G, A) \cdot |E|.$$ 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 $S$ 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 $S$ 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 $S$, 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 the sizes distribution $S$, we need to configure the parameter $\psi$ 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 $\psi$, 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.

\begin{verbatim}
Input: Network $G = (V, E)$.
Input: Sizes distribution $S$.
Input: Parameter $\psi$.
Output: Pool assignment $A$.
1. $\psi \leftarrow \psi E[S] \}$ ;
2. $\mathcal{U}(0, u - 1) ;$ // Uniform distribution
3. sizes $\leftarrow$ sample $|V|$ elements from $S$ ;
4. $A$ $\leftarrow$ {} ;
5. for $i \leftarrow 0$ to $|V| - 1$ do
6. \hspace{1em} $S_i \leftarrow$ sample sizes[i] elements from $\mathcal{U}$ ;
7. \hspace{2em} A.append ($S_i$) ;
\end{verbatim}

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 $\bar{d}$. In Fig. 6, we plot the communication
cost and time to full network sync as a function of $\overline{\text{deg}}$. 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 $Y \times \text{DefTXtoSync}$ 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.
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 ancestor score of the transactions and later uses this score also that communication (or time) needed to sync the entire network. Note

In Table II, we summarize the results for a 10,000 nodes network. $p = 0.24$.

**Procedure 2: SREPSim’s analytical module.**

<table>
<thead>
<tr>
<th>$\text{deg}$</th>
<th>$\psi$</th>
<th>Diameter</th>
<th>$I_{100%}$</th>
<th>$C_{100%}$ (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>16</td>
<td>2.0</td>
<td>3.165879</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>9</td>
<td>2.0</td>
<td>6.317304</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>7</td>
<td>1.5</td>
<td>9.485572</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>6</td>
<td>1.0</td>
<td>1.064714</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>2</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>1</td>
<td>1.0</td>
<td>4.876714</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td></td>
<td>1.0</td>
<td>6.065679</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>5</td>
<td>1.0</td>
<td>15.048386</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>1</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>5</td>
<td>1.0</td>
<td>18.966694</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>1</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>5</td>
<td>1.0</td>
<td>22.156316</td>
</tr>
<tr>
<td>0.355</td>
<td>0.5</td>
<td>1</td>
<td>1.0</td>
<td>33.446278</td>
</tr>
</tbody>
</table>

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

### 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


