Shrec: Bandwidth-Efficient Transaction Relay in High-Throughput Blockchain Systems

Yilin Han[*]
Shanghai Tree-Graph Blockchain Research Institute
yilin@conflux-chain.org

Chenxing Li[†]
Tsinghua University
li-cx17@mails.tsinghua.edu.cn

Peilun Li
Tsinghua University
lpl15@mails.tsinghua.edu.cn

Ming Wu
Shanghai Tree-Graph Blockchain Research Institute
ming@conflux-chain.org

Dong Zhou
Tsinghua University,
Shanghai Qi Zhi Institute
dongz@mail.tsinghua.edu.cn

Fan Long
University of Toronto
fanl@cs.toronto.edu

Abstract

The success of Bitcoin and Ethereum has attracted many efforts to build high-throughput blockchain systems. This paper focuses on transaction dissemination — a rather overlooked issue in these systems. We argue that efficient transaction dissemination is the key for a blockchain system to sustain at high-throughput — usually thousands of transactions per second — and the existing solutions fell short at doing so.

This paper presents Shrec, a novel transaction relay protocol for high-throughput blockchain systems built around a hybrid transaction hashing scheme that has a low hash collision rate, is resilient to collision attacks, and is fast to construct. Our experiments demonstrate that when propagating transactions, Shrec utilizes network efficiently: compared to alternative designs, Shrec reduces the bandwidth consumption by 60% at modest CPU overhead and improves the system throughput by up to 90%.

CCS Concepts

• **Networks** → **Peer-to-peer protocols**; *Network performance analysis*; • **Computer systems organization** → *Fault-tolerant network topologies*; *Distributed architectures*.

[*]Yilin Han was a master student at University of Toronto when conducting this work.
[†]The first two authors contributed equally.

SoCC '20, October 19–21, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8137-6/20/10.
https://doi.org/10.1145/3419111.3421283

Keywords

blockchain, transaction relay, network bandwidth, hash collision, high-throughput

ACM Reference Format:
Yilin Han, Chenxing Li, Peilun Li, Ming Wu, Dong Zhou, and Fan Long. 2020. Shrec: Bandwidth-Efficient Transaction Relay in High-Throughput Blockchain Systems. In *ACM Symposium on Cloud Computing (SoCC '20), October 19–21, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3419111.3421283

1 Introduction

Originated from Bitcoin [42], blockchain powers distributed transaction ledgers at Internet scale and has evolved to fuel innovations in industries like financial systems, supply chains, and health care [16, 17, 26]. To guarantee the security under adversarial conditions in decentralized public environments, blockchain systems with classical Nakamoto Consensus like Bitcoin and Ethereum [1] have very limited throughput, e.g., 7~30 transactions per second (TPS), by applying low block-generation rate or small block size restrictively.

To overcome the throughput bottleneck of Nakamoto Consensus, many consensus protocols have been proposed in recent years [5, 22, 24, 28, 29, 32–34, 38, 41, 46, 47, 51, 52, 56, 57]. These protocols explore alternative structures to organize blocks [5, 22, 33, 34, 46, 51, 52, 57], use Byzantine Fault Tolerant to fully or partially replace Nakamoto Consensus [24, 28, 38, 41, 47], or even partition the blockchain state into multiple shards [29, 32, 56]. For example, both Conflux [34] and OHIE [57] are able to process more than 5000 TPS, several orders of magnitudes faster than the original Nakamoto Consensus. Such high consensus throughput imposes a challenging but largely ignored requirement on the peer-to-peer network: given the average size of a typical Bitcoin transaction, ~ 300 bytes, a consensus throughput of

5000 TPS requires at least 10 Mbps network bandwidth for transaction dissemination. Under 20 Mbps per-node available bandwidth configuration (which is the typical setup of commodity public network links), there is little room left for other protocol messages.

Disseminating transactions across the entire blockchain network in a timely manner is crucial to the confirmation latency of blockchain systems. In order for a transaction T to be accepted by such systems, its *author* must broadcast (or *disseminate*) the transaction to a group of participating nodes via the peer-to-peer network and wait for one of them to generate a valid block B containing T. The node which generates B (the *miner*) will then broadcast the block across the network, hoping that other nodes will accept B based on the consensus protocol. If B is indeed accepted, all of its transactions, including T, are confirmed. Because each node generates block independently and probabilistically, the more nodes a transaction reaches, the faster it will be packed in a confirmed block. Although some sharding techniques [56] limit the spread of transactions within each shard, such systems sacrifice the security guarantee since the state produced by execution of a transaction can only be verified by the nodes in the same single shard.

Perhaps the most straightforward approach to propagate transactions within a peer-to-peer network is by *flooding*, i.e., every transaction received by a participating node is transmitted to all of its neighbors. Because each node is unaware of which transactions its neighbors have already received, this naive approach leads to significant redundancy rate in transactions each node receives, causing poor network utilization. Bitcoin optimizes the network efficiency via *transaction announcements*. A transaction announcement (or digest) is an approximate representation of a set of transactions. Instead of broadcasting transactions, each node in Bitcoin only announces the digest of all the transactions it is aware of. A node X will request a transaction T from its neighbor Y if and only if X does not know T and T is present in the digest sent from Y to X. Without false positives, using transaction announcements completely eliminates redundant transaction transmission.

However, recent work [43] has shown that disseminating transaction digests could be expensive. In fact, in today's Bitcoin network, they have taken up 30-50% of the traffic. This leads us to somewhat contradictory requirements upon the digest design. On the one hand, the blockchain systems benefit from long digests for low false-positive rate; on the other hand, digests need to be extremely compact to save bandwidth. Moreover, given our targeted transaction throughput, these digests must also be fast to compute. Last but not least, recent attacks on transaction censoring [8, 20, 50] renders the necessity of robustness against collision attacks.

In this paper, we present the design, theoretical analysis, and empirical evaluation of Shrec, a novel transaction relay protocol for high-throughput blockchain systems. Shrec encodes each transaction by using only 4 bytes. It adopts a *hybrid hashing* scheme to reduce the false positive rate and defend against collision attacks. We implemented Shrec in Conflux, and our evaluation shows that compared to alternative designs, Shrec uses up to 60% less bandwidth to propagate transactions; Shrec improves the system throughput by up to 90%.

2 Background and Problem Statement

Blockchain systems. Bitcoin [42] and Ethereum [1] are the two prominent and representative peer-to-peer public blockchain systems that account for over 70% of the total market capitalization of cryptocurrencies. They both employ Nakamoto Consensus, and apply low block-generation rate (10 minutes an 1MB block in Bitcoin) or small block size (a block about 100KB per 15 seconds in average in Ethereum) to guarantee the security in decentralized public environments. This leads to a very limited throughput of about 7∼30 TPS that they can only achieve.

In these systems, nodes are connected and communicate through a peer-to-peer gossip network where the block and transaction-related information are disseminated. Specifically, each node can initiate several outgoing connections and accept a number of incoming connections. The number of both the outgoing and the incoming connections are limited through configurations. For example, a typical Bitcoin node can have up to 8 outgoing connections and 117 incoming connections [25], while an Ethereum node may have up to 13 outgoing connections and up to 26 total peers [36]. Prior work [6, 14] has shown that the security of the systems depends on adequate network connectivity and recommended increasing the number of connections between nodes to make the network more robust.

Flooding. In current Bitcoin and Ethereum implementation, transactions are disseminated among nodes using variations of flooding. Flooding is a protocol where each node announces every transaction it receives to each of its peers. Announcements can be sent on either inbound and outbound connections. In Bitcoin, when a node receives a transaction, it advertises the transaction to all of its peers except for the node that sent the transaction in the first place and other nodes from which it already received an advertisement. A node injects a random delay before advertising a received transaction to its peers, which helps reduce the probability of in-flight collisions, i.e., two nodes simultaneously announce the same transaction between them. To advertise a transaction, a node sends a hash of the transaction within an inventory, or INV message. If a node hears about a transaction hash for the first time, it will request the full transaction by sending a GETDATA message to the node that sent it the INV message. The transaction hash is typically a big integer with 32 bytes whose conflict rate is extremely low and negligible. In contrast, in Ethereum, a node floods the original transaction rather than the hash to a fraction of its peers, and maintains the information about which transaction has already been sent for each peer.

Flooding is very inefficient since it introduces a lot of redundant transaction announcements. To see how significant this overhead can be on the network bandwidth consumption, let us first consider the case in Bitcoin. Assuming each node only maintains 8 peers and announces transactions through all its peers. Even if each announcement only contains a 32-byte transaction hash, the total size of announcements that will be propagated for each transaction is 32×8, i.e., 256 bytes. This is already similar to the size of a typical Bitcoin transaction. Ideally and optimally, each node should only receive each transaction exactly once without receiving any other auxiliary information. That is, the transaction hash announcements use almost the same network traffic needed for effective transaction dissemination which may take a significant portion of the overall system traffic. Some empirical study [43] shows that transaction announcements account for 30–50% of the overall Bitcoin traffic. In Ethereum, this situation can be even worse, since a node floods the whole transaction instead of its hash value.

Erlay. Erlay [43] tried to mitigate the bandwidth consumption issues of flooding-based transaction dissemination protocol in Bitcoin. It combines a low-fanout flooding and a set reconciliation mechanism. This is essentially a concrete extension and application of the similar idea of epidemic algorithms [18]. In the low-fanout flooding, each node announces transactions only to a small subset of its peers. This reduces the bandwidth consumption of transaction announcements in flooding accordingly. To ensure all transactions reaching the entire network, in the set reconciliation, nodes periodically conduct an interactive protocol with peers to discover transaction announcements that were missed and request missing transactions. The set reconciliation in Erlay employs PinSketch algorithm [19] to encode a local transaction set of each node for each one of its peers. The set consists of short ids of transactions that the node has not sent to the peer. Through exchanging the sketches of these transaction sets between two connected peers, they can recover the symmetric difference of the sets that represent the transactions missed from the two peers respectively.

However, although Erlay transaction relay protocol demonstrates its effectiveness in Bitcoin network, it cannot be directly applied in high-throughput blockchain systems whose expected throughput can be thousands of transactions per second in practice. In Erlay, the period of initiating set reconciliation is important for performance. If the period is too short, the sketch transfer happens too frequently, which harms the batching effect and reduces the bandwidth utilization of transaction dissemination. On the other hand, if the period is long, the size of the symmetric difference of the transaction set sketches can be very large. For example, empirically, given the 1 second set reconciliation period suggested by Erlay, in a situation with 3000 tps, the size of the difference set between peers can easily go beyond thousands on average. This results in significant computation cost of decoding the sketch which is quadratic with the difference size. Furthermore, since a sketch of the symmetric difference between the two sets is obtained by XORing the bit representation of sketches of those sets, the procedure is very likely to fail if the size of the difference is too large and exceeds the capacity of the original input sketches. As a result, the high-throughput blockchain scenarios make the set reconciliation period hard to be set appropriately.

Goals. In high-throughput public blockchain systems that are bottlenecked at network bandwidth, an ideal transaction relay protocol should have the following properties. First, the transmission effectiveness should be maximized. Ideally, each node should receive each transaction exactly once, this is the effective transmission. Any other transmitted information related to transaction relay is auxiliary or redundant and should be minimized. Therefore, the effectiveness is defined as the size ratio of the effective transmission in the total transferred data related to transaction relay. Secondly, the transaction propagation latency should be maintained as reasonably small. Low transaction transfer delay is essential to user experience since it affects how fast a transaction can be packed into a block. It also leads to better efficiency in block relay with compact block [12] enabled. Thirdly, the relay protocol should not introduce much computation cost per transaction. Since the throughput is high, per-transaction cost can easily accrue to be substantial to shift the system bottleneck to computation and result in low utilization of available network bandwidth, which in turn will negatively impact the expected throughput that the system can finally achieve. And fourthly, the relay mechanism should not incur extra vulnerability to existing threat models, e.g., DoS, timing analysis [45], eclipse [25, 36], or transaction censoring [8, 20, 50] attacks, etc.

3 Shrec Design

The goal of the design of Shrec is to achieve the best trade-off among the above mentioned properties, which hinges on how the transaction announcement is encoded and propagated. The content of a transaction announcement serves the following two purposes. First, for the node receiving the announcement, it is used to check whether the transaction has already been received and processed, and hence does not need to be fetched again from the peer node and be relayed again to others. This can be achieved through a *received pool* that maintains the set of received transaction announcements. Secondly, the content of announcement is also used as the key to fetch the corresponding transaction from the peer and check to avoid duplicated fetching requests to peers, i.e., if an inflight request for a transaction has already been issued to a peer, the node should not issue another request for the same transaction to a different peer to avoid duplicated transmissions of the same transaction. This often requires to maintain an *inflight pool* that consists of the outstanding transaction requests. In summary, the purpose of transaction announcement is to minimize unnecessary and redundant transaction transmissions.

However, the announcement itself consumes network bandwidth. An encoding scheme is required to make a good trade-off between the announcement size and the effectiveness of the resulting transaction dissemination. Shrec encodes each transaction into a 4-byte short id and announces it through a low-fanout flooding, i.e., the announcement is broadcasted to a fixed number of peers (e.g., 8) no matter how many total peers that a node connects to.

To better understand how Shrec encodes the transaction announcement and why it employs such a scheme, we start from describing a strawman approach and evolve it towards our final solution.

3.1 A Strawman Encoding Approach

A naive encoding scheme is to simply assemble the transaction short id by arbitrarily picking 4 bytes (e.g., the last 4 bytes) from the 32-byte SHA-3 [21] hash of the transaction. However, this simple approach confronts high collision rate in a high throughput blockchain system. For example, consider a system with throughput about 5,000 tps (e.g., Conflux). Assume the system forgets transactions that were received two minutes ago, then the *received pool* in each node may constantly contain about 600,000 transactions on average. Therefore, the probability of collision when receiving a new transaction announcement is $600,000/2^{32}$, which leads to an expectation about one collision per 7000 transactions. If the collision happens, the transaction may only be propagated to a few nodes since most of the nodes may have already received the collided announcement and falsely think that this is an already received transaction. In this case, the client that generates the transaction has to resend it after the system forgets it, which introduces much longer extra transaction latency.

In addition, this strawman approach is vulnerable to hash collision attacks that can be conducted in behaviors like censoring transactions [8, 20, 50]. Specifically, an attacker may generate forged transactions offline to collide all possible short ids with 4 bytes, i.e., search for a set of transactions so that it can readily produce one that matches any given 4-byte short id. This requires $n \times log(n)$ times of transaction signing and SHA-3 hashing in expectation, where n is 2^{32}. In our experience, this would take less than 2 weeks with an Intel Xeon E5-2673 processor. To generate the random transactions, the attacker can simply randomly produce an 8-byte integer in the data field of each transaction. Therefore, the major information that needs to be stored are these random integers that consume the space about 32GB. To attack an observed victim transaction, the attacker picks a transaction with colliding short id from the set of pre-generated forged transactions and quickly disseminates it across the entire network. This attack can be repeatedly applied for any specific target transaction, so even resending at client side cannot mitigate the issue.

3.2 Improved Encoding with Random Nonce

In order to decrease the collision rate and defend against the collision attack, a more advanced encoding approach is to introduce a random nonce for communication with each peer. Specifically, the 4-byte transaction short id is constructed by taking the last 4 bytes of the SipHash [4] over the transaction SHA-3 hash. The SipHash is a fast pseudorandom function and can be regarded as a keyed collision-resistant hash function. It takes a short message and a nonce as inputs, and outputs an 8-byte hash value. For each pair of peers, a random nonce is first decided before the transfers of transaction announcements between them, and the nonce is used together with the transaction SHA-3 hash to produce the corresponding SipHash short id. A similar method is also used in the compact block [12] mechanism of Bitcoin.

After introducing the random nonce, the collision becomes independent for different peers. Since a transaction may only be blocked from normal propagation if collisions happen for its announcement from all the peers, the collision rate is exponentially reduced with the number of peers compared to the strawman approach and can be ignored. In addition, it is impossible to attack specific transactions since the attacker cannot predict the nonce used for the channel between a pair of peers.

However, this improved approach introduces another problem. Due to the random nonce per peer, the same transaction has different short ids for different peers. Therefore, for a transaction announcement received from a peer, there is no way to check in the *inflight pool* whether there is an inflight transaction request for the same transaction already being sent to some other peer. This would introduce duplicate requests for the same transaction and incur redundant transaction transmissions. We evaluate the effect of such duplication in Section 5.

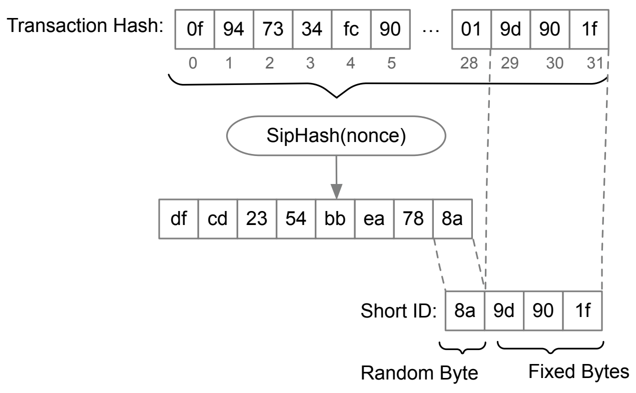


Figure 1: Transaction short id encoding in Shrec.

3.3 The Shrec Encoding Approach

To address the aforementioned issues in the previous two solutions, Shrec employs a novel encoding scheme that combines the values of SipHash and SHA-3 hash. Specifically, to construct the 4-byte short id for a transaction, it takes 3 bytes (e.g., the last three) from the SHA-3 hash of the transaction and 1 byte (e.g., the last one) from the SipHash over the transaction SHA-3 hash, and concatenates them together. An example of this scheme is shown in Figure 1. The 1 byte from the SipHash is called *RandomByte* and the other 3-byte is called *FixedBytes*.

In order to check whether an announcement that is just received from a peer refers to an already received transaction, Shrec looks up in the *received pool* to see whether there is a transaction whose 4-byte short id is the same as that in the announcement. It does this lookup through 2-level indices. It first searches for a set of transactions whose last 3 bytes are same as the *FixedBytes* part in the received transaction short id, and then, from the set, looks for the transaction that can produce the same byte as the *RandomByte* in the short id through applying the SipHash with the corresponding nonce for the peer.

If there is no transaction in the *received pool* that matches the received announcement, Shrec needs to check whether a request for the transaction associating with the announcement has already been issued to another peer by looking at the *inflight pool*. To do this, the outstanding requests in the *inflight pool* in Shrec are indexed with the *FixedBytes* part of short id which has the same representation for different peers. This can mostly get rid of the duplicated transaction transmissions described in Section 3.2. In case of the collision on the *FixedBytes* of short id when querying *inflight pool*, Shrec does not immediately assume the transaction is being requested. Instead, it waits for the result of the in-flight request and re-builds the entire short id of the responded transaction with respect to the peer where the querying short id was received. It then checks whether this re-built short id is same as the querying short id. If they are same, Shrec thinks the transaction is already received, otherwise, it issues the request then.

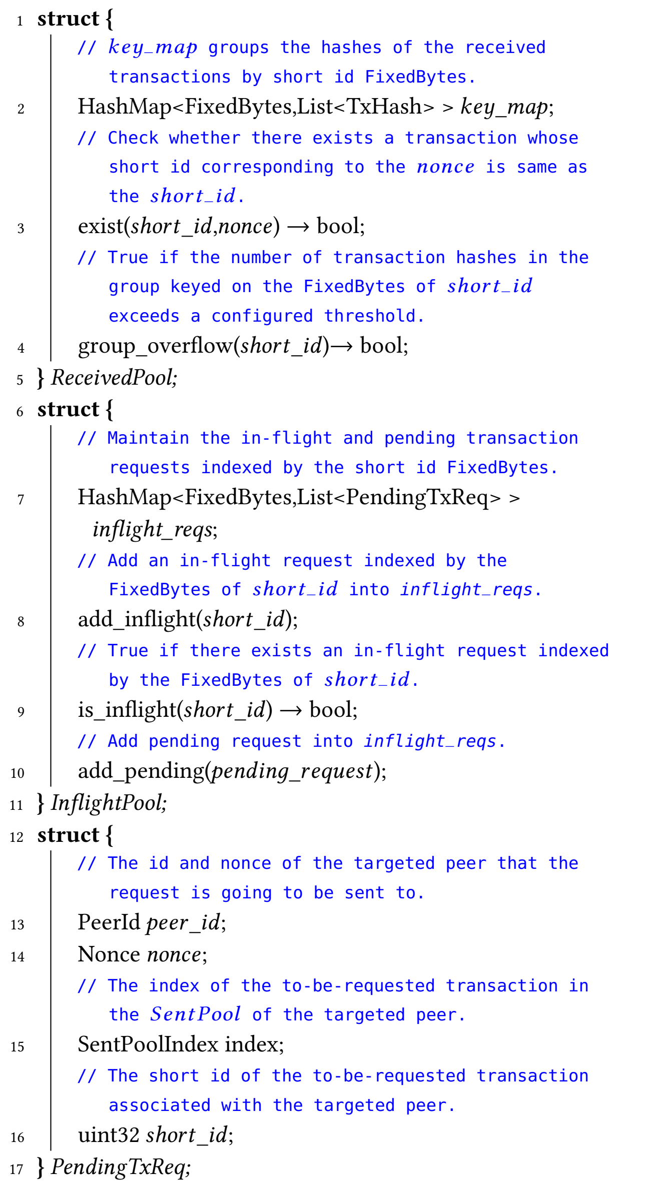

```
struct {
    // key_map groups the hashes of the received
       transactions by short id FixedBytes.
    HashMap<FixedBytes,List<TxHash> > key_map;
    // Check whether there exists a transaction whose
       short id corresponding to the nonce is same as
       the short_id.
    exist(short_id,nonce) → bool;
    // True if the number of transaction hashes in the
       group keyed on the FixedBytes of short_id
       exceeds a configured threshold.
    group_overflow(short_id)→ bool;
} ReceivedPool;
struct {
    // Maintain the in-flight and pending transaction
       requests indexed by the short id FixedBytes.
    HashMap<FixedBytes,List<PendingTxReq> >
      inflight_reqs;
    // Add an in-flight request indexed by the
       FixedBytes of short_id into inflight_reqs.
    add_inflight(short_id);
    // True if there exists an in-flight request indexed
       by the FixedBytes of short_id.
    is_inflight(short_id) → bool;
    // Add pending request into inflight_reqs.
    add_pending(pending_request);
} InflightPool;
struct {
    // The id and nonce of the targeted peer that the
       request is going to be sent to.
    PeerId peer_id;
    Nonce nonce;
    // The index of the to-be-requested transaction in
       the SentPool of the targeted peer.
    SentPoolIndex index;
    // The short id of the to-be-requested transaction
       associated with the targeted peer.
    uint32 short_id;
} PendingTxReq;
```

Figure 2: The definition of struct ReceviedPool, InflightPool, and PendingTxReq.

One issue of this encoding scheme is that merely relying on this encoding scheme does not entirely fix the vulnerability to the hash collision attack which is similar to the one targeting the strawman approach as described in Section 3.1. Still consider the attacking strategy that prepares many forged transactions offline for colliding the specific observed transaction online later. The attacker may group all the generated transactions based on the *FixedBytes* of their short ids. The total number of groups then is 2^{24}. As long as there are enough number of transactions (e.g., several times of 256, i.e., the number of values of a byte) in each group, for an arbitrarily chosen transaction and nonce, the group of transactions which has the same short id *FixedBytes* as the chosen transaction may contain transactions that collide with it on the *RandomByte* of their short ids with high probability. The cost of this offline process would be a small factor of the cost in attacking the strawman approach. When attacking an observed transaction online, the attacker picks the group of transactions with the same short id *FixedBytes* as the victim one and quickly disseminates them across the entire network.

To address this issue, Shrec uses transaction short id to check whether a transaction was received before only when a small number of transactions with the same *FixedBytes* of their short ids co-exist in the *received pool*. The rationale is that the transaction hashes should be uniformly distributed in the normal case without such attack, so the number of transactions in the set with same short id *FixedBytes* should be small. For each received transaction announcement, if Shrec observes, in the *received pool*, that the set of transactions indexed by the *FixedBytes* of the short id in the announcement contains a larger number of transactions than a threshold and there is a transaction in the set that matches the announcement, it requires the peer to re-announce the entire SHA-3 hash of the transaction. In our implementation, we set this threshold as 9 so that the false positive rate of this mechanism is less than 10^{-14}.

3.4 Shrec System and Protocol

In Shrec, the major components in each node include a *received pool*, an *inflight pool*, and a *sent pool*. The *sent pool* is used to buffer the transactions whose announcements are already sent to some other peers and to wait to serve their requests for the transactions. The main data structures in these components are shown in Figure 2. At line 7, the field *inflight_reqs* of `struct InflightPool` tracks the outstanding transaction requests and also the pending requests when collisions happen. If the *FixedBytes* of some short id exists in *inflight_reqs*, it means that an in-flight request for the corresponding transaction has been issued. When some new transaction is going to be requested and its short id *FixedBytes* collides with that of some in-flight request, a pending request is created and inserted into the list associating with the *FixedBytes* in *inflight_reqs*. The field *index* (at line 15) in `struct PendingTxReq` is used to identify the transaction in the *sent pool* of the peer that announces this transaction.

Figure 3 shows a visualized illustration on the data structure of *sent pool* and how the announcement and transaction request messages interact with it. To better utilize the network bandwidth in the transaction relay process, Shrec buffers newly received transactions and periodically batches

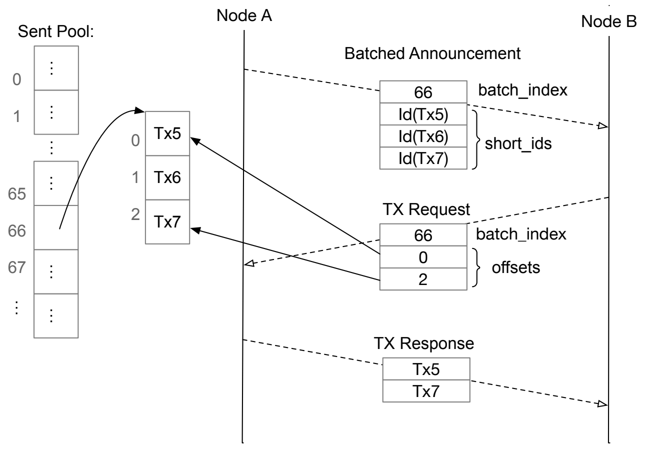


Figure 3: The normal flow of node interaction in Shrec.

```
Input  : received_pool, inflight_pool, the peer information
         of the announcement sender including the peer_id
         and nonce, and the batched announcement which
         consists of batch_index and a list of short_ids.
if short_ids is empty then
    return;
// Stores the SentPool offsets of to-be-requested
   transactions in a batch.
request_offsets =[];
offset=0;
for s in short_ids do
    if received_pool.exist(s,nonce) then
        if received_pool.group_overflow(s) then
            // Request the announcement sender to
               re-send the full hash of the transaction.
            ...
        offset=offset+1;
        continue;
    if inflight_pool.is_inflight(s) then
        pending_request = PendingTxReq(peer_id, nonce,
          (batch_index, offset), s);
        inflight_pool.add_pending(pending_request);
    else
        inflight_pool.add_inflight(s);
        request_offsets.push(offset);
    offset=offset+1;
request_transactions(peer_id,batch_index, request_offsets);
```

Figure 4: The pseudo code for handling a batched announcement.

```
Input : inflight_pool and a list of newly received transactions.
to_request = [];
for tx in transactions do
    fixed_bytes = get_fixed_bytes(tx);
    inflight_reqs = inflight_pool.inflight_reqs;
    pending_reqs = inflight_reqs.get(fixed_bytes);
    if pending_reqs.is_empty() then
        // There is no pending request. Remove the
           in-flight transaction from inflight_reqs.
        inflight_reqs.remove(fixed_bytes);
    else
        // Filter out pending requests that match this
           received transaction.
        for r in pending_reqs do
            if short_id(tx, r.nonce) != r.short_id then
                to_request.push_back(r);
        // Re-issue the first remaining pending request
           and update the in-flight request status.
        if not to_request.is_empty() then
            req = to_request.pop_front();
            idx = req.index;
            request_transactions(req.peer_id,
              idx.batch_index, [idx.offset]);
            inflight_reqs.set(fixed_bytes, to_request);
    // Process the received transaction.
    ...
```

Figure 5: The pseudo code for handling transaction response.

their short ids into a single announcement message to propagate to a random set of peers. The *sent pool* therefore organizes the indices, which are used to associate the announcements and the transactions, in a batched way. It is essentially an append-only buffer with a 2-level index. The first level is the batch index indicating which batch the announcement of the transaction belongs to. The second level index is its offset inside the batch. Similarly, all the transaction requests (excluding the pending ones) corresponding to a batched announcement are also batched in a single request message. This 2-level index makes the announcement and request messages compact. In these messages, the batch index only appears once since all the transactions in a batched message share the same batch index. Figure 4 shows the algorithm handling the received batched announcement to generate the corresponding transaction request packet. Figure 5 shows the algorithm that processes the responded transactions to consume colliding pending requests.

Note that both the data in *received pool* and *sent pool* are managed in a time sliding window fashion to allow the system to forget those information arrived long ago. In our implementation, we set the time-to-live as 5 minutes which is much longer than the latency for a disseminated transaction to reach most of the nodes in the network. A previous report [44] shows that it takes 18 seconds to propagate a transaction to 90% nodes in Bitcoin.

4 Collision and Security Analysis

Our model in analyzing Shrec approach. We start our analysis with finding the collision rate of sending one newly generated transaction, with SHA-3 hash x as its original identifier, to m peers that do not have such transaction in their received pools. Usually, all the participants share similar *received pools*. For simplicity, we assume all the m peers share the same and static *received pool* during the propagation of this new transaction. Let S be the set of SHA-3 hashes of transactions in *received pool*.

In Shrec, the short id of x consists of a 3-byte *FixedBytes* and a 1-byte *RandomByte*. *FixedBytes* is extracted from x and denoted by function $\mathsf{E} : \{0,1\}^{256} \rightarrow \{0,1\}^{24}$. *RandomByte* is computed from SipHash function with 128-bit key (random nonce) over x and is denoted by function $\mathsf{PRF} : \{0,1\}^{256} \times \{0,1\}^{128} \rightarrow \{0,1\}^{8}$. In the following part of this section, *FixedBytes* and *RandomByte* of a SHA-3 hash refer to the corresponding part of the short id. For any $h \in \{0,1\}^{24}$, let S_h group the SHA-3 hashes in S with the *FixedBytes* h in their short id.

Collision rate for a normal case. When announcing a transaction to other peers, Shrec picks m different keys k_i ($i \in [m]$) randomly and sends different short ids to m peers. The following analysis studies the collision rate which refers to the probability that all m peers falsely consider the newly generated transaction as received.

Section 3.3 describes that if S contains too many SHA-3 hashes with the same *FixedBytes* as x, all the participants should request an announcement with its entire SHA-3 hash of x. We denote the threshold by t. When $|S_{\mathsf{E}(x)}| > t$, the participant will forward the SHA-3 hash x instead of its short id, and the collision rate will be 0.

When $|S_{\mathsf{E}(x)}| \leq t$, each peer will falsely skim the announcement for short id of x only if it finds a hash in $S_{\mathsf{E}(x)}$ with the same *RandomByte* as x. Due to the pseudo-randomness of PRF, the *RandomByte* collision events under different SHA-3 hashes or nonce are independent. So the collision rate for a given fixed x can be estimated:

$$\begin{aligned}&\Pr_{k_1,\cdots,k_m}[\forall i \in [m], \exists x_i' \in S_{\mathsf{E}(x)}, \mathsf{PRF}(x,k_i) = \mathsf{PRF}(x_i',k_i)]\\ =&\prod_{i\in[m]}\Big(1-\prod_{x_i'\in S_{\mathsf{E}(x)}}\Pr_{k_i}[\mathsf{PRF}_{k_i}(x)\neq \mathsf{PRF}_{k_i}(x_i')]\Big)\\ =&\big(1-(1-2^{-8})^{|S_{\mathsf{E}(x)}|}\big)^m\end{aligned}$$

Let $p(s) := 1-(1-2^{-8})^s$. We assume SHA-3 hash x is picked uniformly random from $\{0,1\}^{256}$ and independent with received pool S. So for any $h \in \{0,1\}^{24}$, $\Pr[\mathsf{E}(x) = h] = 2^{-24}$. The previous equation claims that the collision rate will be $p(|S_h|)^m$ if $\mathsf{E}(x) = h$ and $|S_h| \leq t$. When taking into account all the possible h, the collision rate for random x will be

$$2^{-24}\times\sum_{h\in\{0,1\}^{24}\wedge|S_h|\leq t}p(|S_h|)^m.$$

In the following computation, we assume the shared received pool contains less than 1, 500, 000 transactions (assuming 5 minutes sliding window under 5000 tps throughput) and each node announces the short id to $m = 8$ peers. In normal case, we can assume S contains independent and uniformly random SHA-3 hashes of transactions. So each group S_h will contain more than eight elements with a small probability ($< 10^{-15}$). Since $\max_{1\leq s\leq 8} p(s)^8/s < 2^{-43}$, the collision rate can be upper bounded by

$$\begin{aligned}2^{-24}\times\sum_{h\in\{0,1\}^{24}\wedge 0<|S_h|\leq t}\frac{p(|S_h|)^8}{|S_h|}\cdot|S_h| &< 2^{-67}\times\sum_{h\in\{0,1\}^{24}}|S_h|\\ &= 2^{-67}\times|S|\end{aligned}$$

This rate means one collision will take at least 600 years to happen if 5000 transactions are processed per second.

Targeting transaction attack. Section 3.1 describes an attack which prevents the propagation of a target transaction with a pre-generated forged transaction pool. This attack is modelled as follows. The attacker is allowed to generate polynomial number of transactions with uniformly random SHA-3 hashed id and store them. After that, the attacker is given a victim transaction with a random SHA-3 hash x. It picks several transactions from its storage based on x and sends these transactions to all the full nodes in order to make them falsely consider the victim transaction already received. The SHA-3 hashes of these transactions are denoted by set C.

In Shrec solution, only the SHA-3 hash with the same *FixedBytes* as x may have a colliding short id with x. So we assume C only contains the SHA-3 hashes x' with $\mathsf{E}(x') = \mathsf{E}(x)$. Further, C should contain no more than t transactions, in order to avoid triggering the mechanism making all the nodes require entire SHA-3 hash in the announcement.

The effort of this attack is measured by the collision rate that a random picked nonce k has a colliding short id with a SHA-3 hash in C. Applying the Boole's inequality and our assumptions for C, this collision rate is bounded by the sum of *RandomByte* collision rate with x for each individual SHA-3 hash in C, a.k.a.

$$\sum_{x'\in C}\Pr_k[\mathsf{PRF}(x,k) = \mathsf{PRF}(x',k)]$$

The collision rate $\Pr_k[\mathsf{PRF}(x,k) = \mathsf{PRF}(x',k)]$ will be $1/256$ for a random SHA-3 hash x'. Due to pseudo-randomness of PRF, the attacker can not find a transaction with SHA-3 hash $\bar{x}'$ in polynomial time such that $\bar{x}'$ has non-negligible advantage in this collision rate compared with random picked x'. So in the best effort, the collision rate with one peer will be $t/256$. The sender has $1-(t/256)^8$ probability in sending out this transaction to at least one out of 8 peers.

Propagation coverage under targeting attack. We care more about how many full nodes will receive a transaction under targeting attack. Let the network contain n full nodes and we study the propagation coverage of transaction Tx.

When focusing on one transaction, the low-fanout forwarding policy is approximately equivalent to make each node choose m peers randomly among all the full nodes and send the transaction with failure probability $t/256$ (the best effort of attacker in targeting attack). Our model for propagation process initials with a forwarding plan list A of $n \cdot m$ items. Each item in A consists of a receiver node picked randomly from n nodes and a successful bit which will be set with probability $1 - t/256$. The model also maintains a *receiving queue* which is initialized with one random node.

In each round, the model pops the first node from *receiving queue*. If this node has never been popped from *receiving queue* before, it receives transaction Tx. At this time, the model simulates process for forwarding transaction to peers by popping m elements from list A, filtering out the receivers with unset successful bit, and pushing the others to *receiving queue*. The propagation terminates when the *receiving queue* is empty.

If the propagation terminates when exactly i nodes have received transaction Tx, the first $m \cdot i$ elements in A must contain no more than i unique successful receivers. We estimate the probability of this event as follows. For a random list A, the number of set successful bits in the first $m \cdot i$ element is a random variable in binomial distribution $B(m \cdot i, 1 - t/256)$, which is denoted by Z. Given $Z = z$, z random nodes picked randomly with replacement from n nodes will contain no more than i unique nodes with probability at most $\binom{n}{i}(i/n)^z$. Taking on all the possible z, the probability that the propagation terminates when i nodes received transaction is upper bounded by

$$q(n,i) := \sum_{z=0}^{m\cdot i}\Pr[Z=z]\cdot\binom{n}{i}\cdot\left(\frac{i}{n}\right)^z.$$

Thus, transaction Tx will be propagated to at least u nodes with failure probability $\sum_{i=1}^{u-1} q(n,i)$, which equals to 1.22×10^{-8} when $m = 8, n = 5000, u = 4980, t = 9$. It implies that transaction Tx will be propagated to 99.6% full nodes even under targeting attack.

5 Evaluation

We implemented Shrec transaction relay protocols in Conflux [34] with Rust. We also implemented three other different transaction propagation mechanisms on the same code base for comparison, including BTCFlood, Erlay [43], and the SipHash protocol described in Section 3.2. BTCFlood and SipHash protocols are flooding-based solutions. BTCFlood takes SHA-3 hash of a transaction as its id to announce, while SipHash scheme uses the last 4 bytes of SipHash over transaction hash as the id in announcement.

Note that Erlay combines the low-fanout flooding and periodic set reconciliation. In our setup, each node propagates sketches to all its peers for reconciliation in every second. To be consistent with the setup reported in Erlay paper [43], we use the 8-byte SipHash of transaction SHA-3 hash as its id. The ids of the received transactions in a one-second period are encoded by PinSketch [19] algorithm with a parameter l, which generates an $8 \times l$ bytes sketch. When a node receives a sketch, it decodes the sketch with its own local one to extract the symmetric difference. If the symmetric difference is less than l, the decoding process will always be successful, otherwise, it fails. Because of this, choosing a proper l is difficult. If l is too big, it does not save the bandwidth as expected. On the other hand, if l is too small, the failure of the decoding provides no information on which transactions a node should request. In addition, decoding sketches is computationally expensive, as it is quadratic to the size of maximum symmetric difference. Therefore, in order to find the best trade-off, we gradually increase the variable l so that PinSketch achieves at least 99% of success rate without over consuming the CPU.

5.1 Experiment Setup

We evaluate all the protocols using 1,000 m5.2xlarge virtual machine instances on Amazon EC2. Each instance has 4 cores and 16GB memory. In the experiments, we run 1 Conflux full node on each of the instances. Conflux adopts account model. Before each experiment, one million accounts are created in the system in advance. Each account is given sufficient amount of fund for transferring. During the experiment, simple payment transactions are randomly generated as the workload under a specified global throughput in the entire network, where each node contributes evenly to the transaction generation process. The average transaction size is 110 bytes, and each node produces and propagates a batched announcement every 0.2 seconds. The global workload throughput is adjusted in every experiment to reach the performance capacity of each protocol.

Conflux supports much higher block generation rate than Bitcoin and Ethereum. Similar to Bitcoin, it also employs compact block to avoid transferring transactions through block other than transaction dissemination protocol. Compact blocks may contain information of duplicated transactions which may introduce other bandwidth and execution overhead. However, this overhead is orthogonal to the impact of transaction relay protocols. In the experiments, we configure the global block generation rate as 4 blocks per second and the block size limit as 3000 transactions. This setup can satisfy the workload throughput in our experiments and introduce minimal overhead caused by duplicate transactions in blocks.

When a new Conflux node is initialized, it must discover at least one other existing node in the network to participate. Each node will issue 8 outbound connections and it can receive up to 32 inbound connections. In order to avoid isolated subgraphs, the initialization of network topology is partially randomized: the nodes are connected in a circular manner for their first outgoing connection, then each node further initialize 7 extra outgoing connections to randomly selected peers. This leads to about 13 total connections per node on average. Shrec is evaluated by setting low-fanout to 4 and 8 peers, represented with SR-4 and SR-8 respectively, and also flooding to all connected peers, noted as SR-Flood. For all the other protocols, we set the flooding fanout as 8. We emulate the network environment as in real deployment by limiting the maximum available network bandwidth of each node and injecting a 0~300ms random delay in the communication between each pair of nodes. We evaluate the systems with the bandwidth limitation at 10 Mbps and 20 Mbps, respectively, to better understand the trade-offs under different network resource constraints.

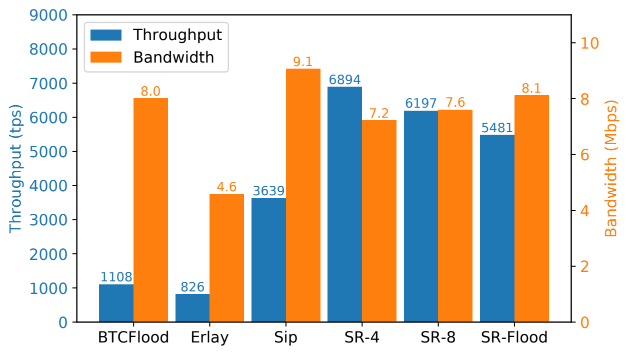
In the experiments, we are aiming to answer the following questions:

(1) How does Shrec perform compared to alternative protocols with respect to system throughput under different network constraints?
(2) How effective is Shrec on reducing bandwidth consumption of required announcement information and redundant transactions?
(3) What is the impact of the low-fanout mechanism in Shrec on transaction packing latency?

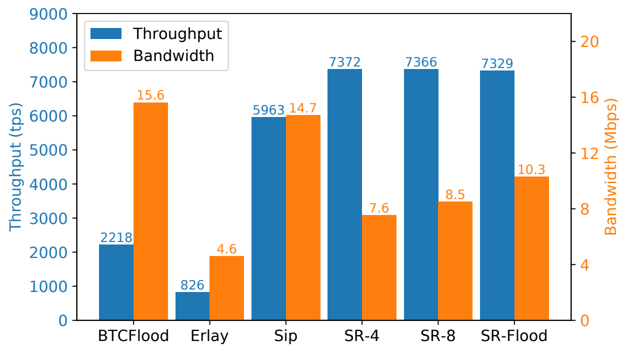
5.2 Overall System Performance

In this experiment, we compare the best achievable throughput of the system under different transaction relay protocols.

We also measure the per-node network bandwidth consumption of total outgoing messages.



(a) 10 Mbps bandwidth limit



(b) 20 Mbps bandwidth limit

Figure 6: System throughput and bandwidth consumption.

Figure 6(a) shows the results under 10 Mbps bandwidth limit. Shrec significantly outperforms all the other alternative solutions. BTCFlood and SipHash are clearly bounded at network bandwidth. BTCFlood already consumes 80% network bandwidth at throughput 1108 TPS, while SipHash saturates the bandwidth at 3639 TPS. In contrast, Erlay cannot sufficiently utilize the network bandwidth because its sketch encoding and decoding are computation intensive, which makes it CPU-bounded.

Shrec achieves the highest throughput at 6894 TPS when flooding fanout is 4. The throughput decreases when fanout grows since the announcement propagation takes more network bandwidth. This indicates that, under constrained network environment, lower fanout should be used to achieve higher throughput.

To understand how Shrec performs under relatively unconstrained network environment, we also evaluate the system with 20Mbps bandwidth limit. BTCFlood and SipHash significantly improved their throughput since they are heavily bounded at network bandwidth. However, 20Mbps is still too constrained for them, so their throughput are still significantly lower than Shrec. Erlay does not gain any improvement on its throughput with such higher bandwidth limit since it is bounded at CPU processing.

Interestingly, Shrec with different fanouts achieve similar throughput because the system is not bounded at bandwidth anymore under 20Mbps limit. According to our investigation, the system is bounded by the transaction execution throughput in this case. Prior work has shown that there exist many potentials to further improve the execution efficiency of Ethereum-like transactions which is bounded at the merkle tree[40] access, e.g., Ponnapalli et. al.[48] proposes a solution to improve the merkle tree access for 100x more efficient than Ethereum implementation. In this case, the throughput of the system with Shrec would be bottlenecked at network bandwidth again under 20Mbps limit.

5.3 Bandwidth and CPU Consumption Breakdown

To prove that the performance improvement of Shrec is achieved by reducing the CPU and network resource consumption, we conducted more detailed evaluations on these factors.

To understand the transmission effectiveness of these protocols, we rerun the experiments under 500 TPS workload throughput which can be supported by all the protocols, and analyze their bandwidth usage.

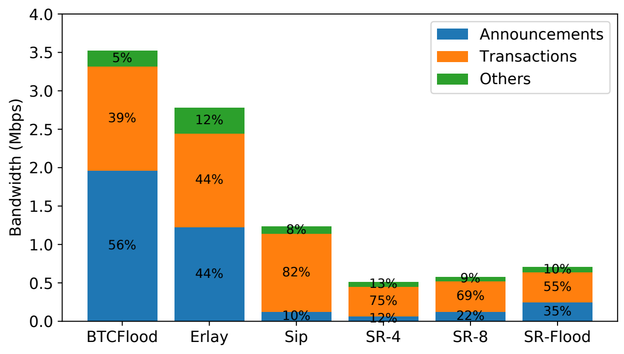


Figure 7: Bandwidth consumption breakdown at 500 TPS workload throughput and 20Mbps bandwidth limit.

Figure 7 breaks down the bandwidth usage among announcements, transactions, and other block-related messages. For BTCFlood and Erlay, where the total bandwidth consumption is 3.5 Mbps and 2.8 Mbps respectively, the announcements consume the most bandwidth as they are composed of 32-byte transaction hashes. In comparison, by

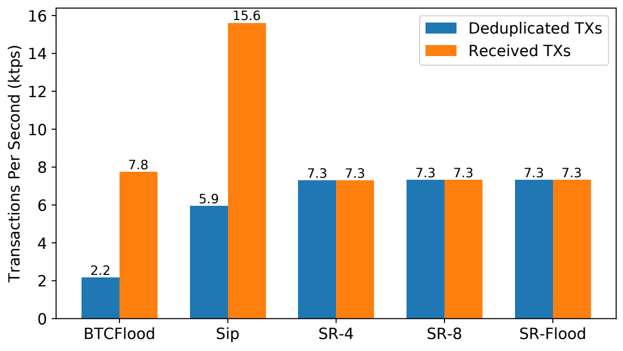


Figure 8: The total number and deduplicated number of received transactions.

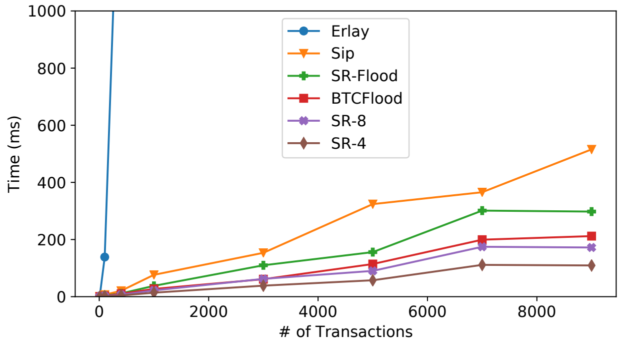


Figure 9: CPU cost involved during transaction relay.

utilizing 4-byte short ids and the low-fanout design, SR-4 achieves 75% transmission effectiveness, and reduces the total bandwidth consumption to only 0.5 Mbps under the same throughput.

Although SipHash uses the most portion of bandwidth for transaction transmission, it still costs more bandwidth compared with Shrec because most transmitted transactions in SipHash are duplicated as discussed in Section 3.2. Figure 8 compares the transaction throughput received at the network level and the actual deduplicated throughput in the experiments with 20Mbps bandwidth limit. SipHash receives more than twice the transactions as it is supposed to be, wasting these bandwidths for redundant data. Shrec improves over SipHash by utilizing *inflight pool* to prevent duplicate fetching request and did not waste any bandwidth on duplicated transaction transmission. However, SipHash does not support *inflight pool* because its short ids of the same transaction provided by different peers are completely different.

To understand the CPU cost of all the protocols, we run the logic of announcements generation and handling as benchmark on a single thread node. This experiment is driven by transactions as input. We run the experiments with different numbers of transactions. For each experiment, we run multiple times to measure the average CPU time. Figure 9 shows the results. The running time of all the protocols, except for Erlay, increases almost linearly with the number of transactions. As the figure shows, the computation cost of Erlay is significantly larger than other protocols as it is quadratic to throughput. When input transaction number is about 800, it consumes 3.52 seconds to decode sketches with 16 peers. Consider the case with its best achievable throughput (826 TPS), this would exhaust a huge portion of CPU resources on an EC2 m5.2xlarge instance. This confirms that Erlay protocol is bounded at CPU computation rather than network bandwidth. In contrast, the computation cost of Shrec is very low. Running with 9000 transactions only costs 109ms and 172ms CPU time for SR-4 and SR-8 to propagate transactions, which is quite affordable for a common 4-core machine and leaves sufficient CPU budget for other part logic of the system.

5.4 Transaction Packing Latency

We have shown that Shrec can achieve better throughput by consuming less bandwidth. Now we further analyze its latency performance when it reaches its best achievable throughput.

Figure 10 describes the transaction packing latency of SR-4 and SH-8 under 20 Mbps bandwidth cap. The system is not bounded on the bandwidth in this setup. The packing latency includes the time of transaction propagation, the wait time to be selected among all unpacked transactions, and for generating a block with Proof-of-Work. Over 80% transactions can be packed within 20 seconds for both cases. On average, reducing the number of fanout peers from 8 to 4 only increases the latency from transaction generation to packing from 10.79 seconds to 12.24 seconds, which only increased by 1.45s. We believe it is reasonable to choose fanout 4 over 8 for less bandwidth consumption, because the latency increase is almost negligible compared with the transaction confirmation time at the consensus layer, i.e., 1 hour for Bitcoin and 23s for Conflux.

Large scale simulation. To better understand the low-fanout effects in Shrec, we measured the transaction propagation and packing latency on 20,000 nodes with simulations. The block generation rate is set to 4 blocks per second. In the simulation, transactions are generated from a source node and propagated in iterations where one iteration represents one hop of transaction broadcast. The time of an iteration is set to 1 second and the size of the transaction pool is set to 100,000, same as the value we measured in the experiments

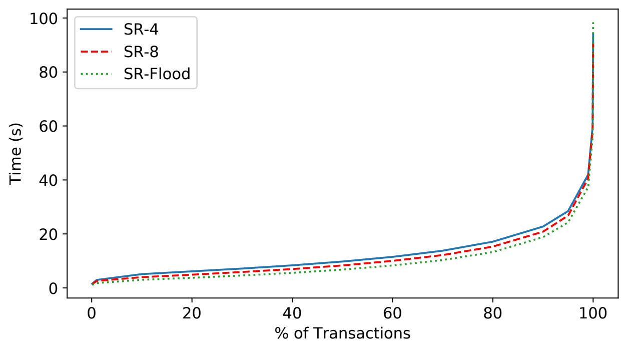


Figure 10: Transaction packing latency with 20 Mbps bandwidth limit.

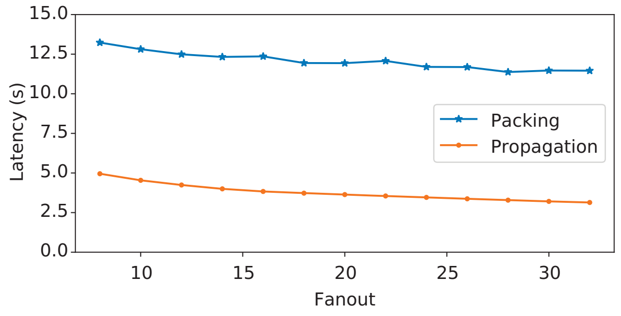


Figure 11: The average transaction propagation and packing latency of different fanouts.

of 20 Mbps bandwidth limit. Figure 11 plots the average propagation latency and packing latency, and shows that increasing the fanout becomes less and less beneficial. For example, doubling fanout from 8 to 16 only reduces the packing latency from 13.2s to 12.3s, but increases the bandwidth consumption by about 18%. This is because the number of hops required for broadcasting a transaction across the entire network is $O(log_d N)$ where d is the average degree (i.e., the fanout) and N is the number of nodes in the network.

6 Related Work

Transaction propagation in blockchain. Bitcoin [42] and Ethereum [1] pioneered decentralized public blockchain systems. One of their major functions is to provide secure transaction ledgers at internet scale in a trustless way. In such systems, transactions are needed to be disseminated across the entire network timely so that they can be packed into the newly generated blocks and get processed with low latency. Both of the systems employ variants of flooding mechanism to fulfill the transaction propagation. Since these systems can only achieve very low transaction processing throughput, which is mainly bottlenecked at their consensus protocols, the available network bandwidth typically has enough budget to tolerate the inefficiency of the naive flooding-based transaction transmission schemes.

One problem of transaction flooding in Bitcoin is that it cannot scale with the increasing number of the connected peers per node. Erlay [43] observes this issue and proposes a combination of low-fanout flooding and sketch-based set reconciliation to reduce the bandwidth consumption and keep the bandwidth use almost constant as the connectivity increases.

More recently, many new consensus protocols and ledger structures [5, 22, 24, 29, 32–34, 38, 41, 46, 47, 51, 52, 56, 57] have been proposed and applied in public blockchains to significantly improve the system throughput to thousands of transactions per second. These systems are typically not bottlenecked at the consensus protocol anymore but at the available network bandwidth. Although Erlay is effective in Bitcoin scenarios, it cannot be applied in these more advanced systems since the computation cost of its set sketch mechanism is too high and the success rate is low when the target transaction throughput is such high. In addition, without careful design of the transaction announcement encoding, even low-fanout flooding can easily consume substantial scarce network bandwidth, and hence significantly limit the achievable throughput. In contrast, Shrec optimizes the encoding of the short transaction id to achieve the best balance between the effectiveness of the transaction announcement and its overhead.

Security effects of connectivity. Many network-related security issues in Bitcoin and Ethereum have been scrutinized with the corresponding attacks [2, 3, 6, 7, 13, 15, 23, 25, 27, 30, 36, 39] published. These attacks attempt to increase the probability of double-spending or denials of service, or violate the user privacy. Many of them assume the victim nodes having limited number of connected peers. This causes repeated recommendations from recent literatures [6, 14] on increasing the number of connections between nodes to make the network more robust. Shrec sticks to a low-fanout flooding on its transaction announcement mechanism, and therefore will not introduce more transmission overhead when network connectivity increases.

Structured peer-to-peer networks. Unlike the peer-to-peer gossip networks employed in decentralized blockchain systems, structured peer-to-peer networks are also extensively used in another series of systems that leverage topology information to make efficient routing decisions, including tree-based multicast protocols [9, 10, 54] and Distributed Hash Tables (DHTs) [11, 37, 49, 53, 55]. However, these designs make the protocols leak information about the structure of the network and introduce security vulnerabilities in adversarial environments. For example, Ethereum employs

Kademlia DHT [37] to maintain its network topology, and hence suffers from a low-resource eclipse attack [36].

A typical trade-off in these DHT systems is between the bandwidth consumption in routing table maintenance and the query latency which relies on the number of required lookup hops. Kumar et. al. identify and formalize this trade-off [31], and Li et. al. propose a DHT design that can automatically optimize the trade-off according to the available network bandwidth budget [35]. Similarly, Shrec can also allow the full node to adjust the number of peers to flood transaction announcement based on the network bandwidth budget.

7 Conclusion

Recent development of public blockchain systems has resulted in significant improvement on throughput, bringing much higher requirement and more challenges on transaction transmission efficiency. Shrec helps blockchain system to achieve high transaction transmission efficiency with strong security guarantee through its novel hybrid hashing scheme for transaction announcement encoding. This leads to much higher system throughput by significantly mitigating the network bandwidth bottleneck.

References

[1] 2020. Ethereum White Paper. https://ethereum.org/en/whitepaper/.

[2] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2015. On the Malleability of Bitcoin Transactions.. In *Financial Cryptography and Data Security*.

[3] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *IEEE Symposium on Security and Privacy (SP)*.

[4] Jean-Philippe Aumasson and Daniel J. Bernstein. 2012. SipHash: A Fast Short-Input PRF. In *Progress in Cryptology - INDOCRYPT 2012*, Steven Galbraith and Mridul Nandi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 489–508.

[5] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the Blockchain to Approach Physical Limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 585–602. https://doi.org/10.1145/3319535.3363213

[6] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. 2014. Deanonymisation of Clients in Bitcoin P2P Network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 15–29. https://doi.org/10.1145/2660267.2660379

[7] Joseph Bonneau. 2016. Why buy when you can rent? Bribery attacks on bitcoin-style consensus. In *Financial Cryptography and Data Security - International Workshops, FC 2016, BITCOIN, VOTING, and WAHC, Revised Selected Papers (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9604 LNCS)*. Springer Verlag, Germany, 19–26.

[8] JP Buntinx. 2017. F2Pool Allegedly Prevented Users From Investing in Status ICO. https://themerkle.com/f2pool-allegedly-prevented-users-from-investing-in-status-ico/.

[9] Justin Cappos and John H. Hartman. 2008. San Fermín: Aggregating Large Data Sets Using a Binomial Swap Forest. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (San Francisco, California) *(NSDI'08)*. USENIX Association, USA, 147–160.

[10] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. 2003. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM symposium on Operating Systems Principles*. ACM, 298–313.

[11] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. 2001. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability* (Berkeley, California, USA). Springer-Verlag, Berlin, Heidelberg, 46–66. http://dl.acm.org/citation.cfm?id=371931.371977

[12] Matt Corallo. 2017. Compact block relay. BIP 152. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki.

[13] corbixgwelt. 2011. Timejacking and bitcoin. http://culubas.blogspot.de/2011/05/timejacking-bitcoin_802.html

[14] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the Bitcoin network. IEEE P2P 2013 Proceedings. https://doi.org/10.1109/P2P.2013.6688704.

[15] Christian Decker and Roger Wattenhofer. 2014. Bitcoin Transaction Malleability and MtGox. *CoRR* abs/1403.6676 (2014). arXiv:1403.6676 http://arxiv.org/abs/1403.6676

[16] Deloitte. 2017. 5 blockchain technology use cases in financial services. http://blog.deloitte.com.ng/5-blockchain-use-cases-in-financial-services/.

[17] Deloitte. 2018. Blockchain: Opportunities for health care. https://www2.deloitte.com/us/en/pages/public-sector/articles/blockchain-opportunities-for-health-care.html.

[18] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. 1987. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of Distributed Computing*. ACM.

[19] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. 2004. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In *Advances in Cryptology - EUROCRYPT 2004*, Christian Cachin and Jan L. Camenisch (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 523–540.

[20] DwarfPool. 2016. Why DwarfPool mines mostly empty blocks and only few ones with transactions. https://www.reddit.com/r/ethereum/comments/57c1yn/why_dwarfpool_mines_mostly_empty_blocks_and_only/.

[21] Morris J. Dworkin. 2015. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. *Federal Inf. Process. STDS. (NIST FIPS)* (2015).

[22] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol.. In *NSDI*. 45–59.

[23] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. 2015. Tampering with the Delivery of Blocks and Transactions in Bitcoin. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. ACM, New York, NY, USA, 692–705. https://doi.org/10.1145/2810103.2813655

[24] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 51–68.

[25] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *24th*

USENIX Security Symposium (USENIX Security 15). USENIX Association, Washington, D.C., 129–144. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[26] IBM. 2020. Blockchain for Supply Chain. https://www.ibm.com/blockchain/supply-chain/.

[27] Benjamin Johnson, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. 2014. Game-Theoretic Analysis of DDoS Attacks Against Bitcoin Mining Pools.. In *Financial Cryptography and Data Security*.

[28] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*. 279–296.

[29] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 583–598.

[30] Philip Koshy, Diana Koshy, and Patrick McDaniel. 2014. An Analysis of Anonymity in Bitcoin Using P2P Network Traffic.. In *Financial Cryptography and Data Security*.

[31] P. Kumar, G. Sridhar, and V. Sridhar. 2005. Bandwidth and latency model for DHT based peer-to-peer networks under variable churn. In *2005 Systems Communications (ICW'05, ICHSN'05, ICMCS'05, SENET'05)*. 320–325. https://doi.org/10.1109/ICW.2005.31

[32] Derek Leung, Adam Suhl, Yossi Gilad, and Nickolai Zeldovich. 2018. Vault: Fast Bootstrapping for Cryptocurrencies. *IACR Cryptology ePrint Archive* 2018 (2018), 269.

[33] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. 2015. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*. Springer, 528–547.

[34] Chenxing Li, Peilun Li, Dong Zhou, Wei Xu, Fan Long, and Andrew Yao. 2018. Scaling Nakamoto Consensus to Thousands of Transactions per Second. *arXiv preprint arXiv:1805.03870* (2018).

[35] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. 2005. Bandwidth-efficient Management of DHT Routing Tables. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*. USENIX Association, Berkeley, CA, USA, 99–114. http://dl.acm.org/citation.cfm?id=1251203.1251211

[36] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. 2018. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. Cryptology ePrint Archive, Report 2018/236. https://eprint.iacr.org/2018/236.

[37] Petar Maymounkov and David Mazières. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems (Druschel P., Kaashoek F., Rowstron A. (eds). Springer, Berlin, Heidelberg)*. Springer Verlag.

[38] David Mazieres. 2015. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation* (2015).

[39] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. 2016. Refund attacks on Bitcoin's Payment Protocol. *IACR Cryptology ePrint Archive* 2016 (2016).

[40] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*. Springer, 369–378.

[41] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 31–42.

[42] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.

[43] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. 2019. Erlay: Efficient Transaction Relay for Bitcoin. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 817–831. https://doi.org/10.1145/3319535.3354237

[44] Till Neudecker. 2019. Characterization of the Bitcoin Peer-to-Peer Network (2015-2018). http://dsn.tm.kit.edu/bitcoin/publications/bitcoin_network_characterization.pdf.

[45] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. 2016. Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network. In *2016 International IEEE Conference on Advanced and Trusted Computing (ATC)*. 358–367. https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070 Received Best Paper Award.

[46] Rafael Pass and Elaine Shi. 2017. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 315–324.

[47] Rafael Pass and Elaine Shi. 2017. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[48] Soujanya Ponnapalli, Aashaka Shah, Amy Tai, Souvik Banerjee, Vijay Chidambaram, Dahlia Malkhi, and Michael Wei. 2019. Scalable and Efficient Data Authentication for Decentralized Systems. *arXiv preprint arXiv:1909.11590* (2019).

[49] Antony Rowstron and Peter Druschel. 2001. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science - LNCS* 2218 (01 2001), 329–350.

[50] SECBIT. 2018. How the winner got Fomo3D prize - A Detailed Explanation. https://medium.com/coinmonks/how-the-winner-got-fomo3d-prize-a-detailed-explanation-b30a69b7813f.

[51] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. 2020. PHANTOM and GHOSTDAG, A Scalable Generalization of Nakamoto Consensus. (2020). https://eprint.iacr.org/2018/104.pdf.

[52] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 507–527.

[53] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (San Diego, California, USA) *(SIGCOMM '01)*. ACM, New York, NY, USA, 149–160. https://doi.org/10.1145/383059.383071

[54] V. Venkataraman, K. Yoshida, and P. Francis. 2006. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In *Proceedings of the 2006 IEEE International Conference on Network Protocols*. 2–11.

[55] S. Vuong and J. Li. 2003. Efa: an efficient content routing algorithm in large peer-to-peer overlay networks. In *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*. 216–217. https://doi.org/10.1109/PTP.2003.1231532

[56] Jiaping Wang and Hao Wang. 2019. Monoxide: Scale out Blockchains with Asynchronous Consensus Zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 95–112. https://www.usenix.org/conference/nsdi19/presentation/wang-jiaping

[57] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. 2020. OHIE: Blockchain Scaling Made Simple. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE.