MERCURY: Fast Transaction Broadcast in High Performance Blockchain Systems

Mingxun Zhou[*‡], Liyi Zeng[†‡], Yilin Han[§], Peilun Li[§], Fan Long[¶], Dong Zhou[‖], Ivan Beschastnikh[**], Ming Wu[§]
[*]Carnegie Mellon University [†]Institute for Interdisciplinary Information Sciences, Tsinghua University
[§]Shanghai Tree-Graph Blockchain Research Institute [¶]University of Toronto [‖]IMO Ventures [**]University of British Columbia

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228972

***Abstract*—Blockchain systems must be secure and offer high performance. These systems rely on transaction broadcast mechanisms to provide both of these features. Unfortunately, in today's systems, the broadcast mechanisms are highly inefficient.**

We present MERCURY, a new transaction broadcast protocol designed for high performance blockchains. MERCURY shortens the transaction propagation delay using two techniques: a virtual coordinate system and an early outburst strategy. Simulation results show that MERCURY outperforms prior propagation schemes and decreases overall propagation latency by up to 44%. When implemented in Conflux, an open-source high-throughput blockchain system, MERCURY reduces transaction propagation latency by over 50% with less than 5% bandwidth overhead.

***Index Terms*—Blockchain, transaction broadcast, propagation latency, P2P network.**

I. INTRODUCTION

Pioneered by Bitcoin [1] and Ethereum [2], decentralized public blockchains provide a secure transactional ledger abstraction at internet scale. As a result, blockchain systems are now being adopted in many areas, such as finance [3], [4], [5], supply-chain management [6], and health care [7].

Public blockchain systems are built around consensus logic over a peer-to-peer (P2P) gossip network. Original systems, like Bitcoin [1] and Ethereum [2], have low performance in terms of throughput and latency. For example, Bitcoin's throughput is about 7 transactions per second (TPS) with a transaction confirmation latency of about an hour. Ethereum, though better, only supports about 30 TPS and has a confirmation latency of around 10 minutes. The root cause of such low performance is the conservative mechanism in Nakamoto consensus to cope with the long broadcast latency of the P2P network. To form a chain structure in a high-latency environment, the consensus algorithms must use a low global block generation rate. This decreases throughput and increases confirmation latency. Decreasing the transaction and block propagation delay is therefore key to improving the performance of these systems.

Recently work has proposed new designs that address the *throughput* bottleneck of Nakamoto consensus, including alternative ledger structures [8], [9], [10], [11], [12], combining BFT-like mechanisms [13], [14], [15], [16], or sharding blockchain states [17], [18], [19]. Some of these protocols have pushed the throughput to thousands of TPS. These proposals drastically increase network bandwidth consumption and push the performance bottleneck toward the underlying P2P network.

Transaction dissemination is also under pressure from recently deployed low-latency systems. For example: Stellar has an average latency of 1.1s and 2.4s at the 99th percentile [14], Avalanche claims a confirmation time of 0.3s [20], and several articles claim that Solana's latency is around 0.4s [21]. To provide such low latencies in the wide area, these systems require fast transaction dissemination.

A naive solution to speed up transaction dissemination is to increase *fanout*. That is, have each node in the network relay the transactions to more peers. Unfortunately, this strategy trades off lower latency for higher transaction throughput: with high fanout, nodes will receive the same transaction multiple times and this will decrease throughput. This is the focus of our work: designing a transaction dissemination protocol that achieves low latency without high bandwidth overhead.

We present MERCURY, a new protocol for transaction dissemination in blockchain systems. MERCURY uses a broadcast strategy that relies on virtual coordinate systems (VCS) to guide the routing decision of individual nodes. The strategy significantly boosts the broadcast efficiency without wasting network bandwidth. It makes MERCURY particularly suitable for high throughput blockchain systems where the network bandwidth is a scarce resource.

MERCURY first uses a virtual coordinate system (VCS) to assign each node a coordinate. The VCS ensures the *propagation latency* between any two nodes is proportional to distance between their VCS coordinates. Unlike previous work which does not consider malicious behaviors [22], MERCURY uses a secure VCS to ensure that the assigned coordinates will remain robust to attacks. MERCURY then clusters all the nodes based on their coordinates and selects two kinds of nodes to propagate a transaction: close neighbors in the same cluster (to reduce the latency per hop), and random nodes in other clusters (to initiate propagation across clusters earlier). Finally, MERCURY uses a simple and effective optimization called *early outburst*, which dynamically configures the propagation tree structure and takes advantage of the early stage in the propagation process. MERCURY has several protection mechanisms that allow it to fallback to basic random propagation in case of an attack.

We evaluated MERCURY with simulations and compared

[‡]The first two authors contributed equally to this work.
Contacts: mingxunz@andrew.cmu.edu, zengly17@mails.tsinghua.edu.cn.

MERCURY with BlockP2P [22] and Perigee [23], two recent blockchain P2P propagation protocols optimized for low latency. Our results show that the propagation latency of MERCURY is 49% lower than BlockP2P and 30% lower than Perigee in a simulated network of 8,000 nodes. MERCURY is also robust to attacks. Even when 49% of the nodes are malicious, the propagation latency of MERCURY is still much better than BlockP2P and Perigee. We also integrated MERCURY with Conflux [10], a high throughput blockchain system. Our evaluation shows that MERCURY reduces the transaction propagation delay by up to 56% for a 1000-node network and that its bandwidth overhead does not exceed 5%.

In summary, we make the following contributions:

- We design a secure VCS-based broadcast strategy for blockchain P2P networks that improves latency without wasting bandwidth.
- We propose a simple and effective optimization called *early outburst*.
- We integrate our proposals into an end-to-end implementation of MERCURY and evaluate it in simulation and with the Conflux high-throughput blockchain system.

II. BACKGROUND

Nodes in blockchain systems like Bitcoin [1] and Ethereum [2] maintain a geo-replicated ledger and are connected via a P2P gossip network. The ledger records transactions made by users. The transactions are packed into blocks which are chained together according to some happened-before relationship.

To replicate the ledger in a blockchain system, transactions and blocks are disseminated through the P2P network. Each node will try to relay the received transactions and blocks to some connected peers. A mining node will also try to pack the received transactions into a new block which may be appended into the ledger. These nodes generate new blocks with certain probability to maintain a specific global block generation rate. Therefore, the quicker a transaction reaches many nodes, the faster it will be packed into a block, and the sooner it will be confirmed by the ledger.

In Bitcoin and Ethereum, blocks and transactions are broadcast using variations of flooding: each node announces content to each of its peers. In Bitcoin, a node announces the hash of the received block or transaction to all its peers. Subsequently, the peers that have not yet received the block/transaction respond with a GETDATA message to request it. Ethereum nodes broadcast information similarly except that a node relays a complete transaction to a small fraction of connected peers and floods the transaction hash to all other nodes [24].

Aggressive flooding squanders network bandwidth. In high-throughput blockchain systems like Conflux [10], simple flooding can saturate network bandwidth. Shrec [25] optimizes the transaction relay protocol to effectively utilize the network bandwidth by designing a hybrid transaction short id encoding that achieves a trade-off between the announcement size and the conflict rate. This pattern of a small announcement followed by content retrieval is assumed in this paper.

To effectively utilize network bandwidth in high-throughput systems, transaction relay typically requires batching. This is because the transaction announcement itself is still flooded and the announcement size is small, e.g., 4 bytes in Shrec. If the announcement is sent one-by-one, the meta-data overhead would be substantial, so batching transactions is inevitable. But, batching introduces extra latency, which is proportional to the number of hops in the dissemination. This in turn makes decreasing the number of hops a critical goal when optimizing transaction dissemination latency.

III. SYSTEM MODEL

a) Network Model: We model the P2P network of nodes in a blockchain system as an undirected graph $G(V, E)$, where V is the set of nodes and E is the set of connections between nodes. When a node joins the network, it first tries to connect to a list of IP addresses of predefined nodes. The node then enters the discovery phase and will request more node IP addresses from the connected peers. The node repeats this process and maintain the known peer addresses, `Addr`, until it learns enough IP addresses. Next, the node connects to the known nodes over TCP to synchronize block information. We denote the connected peer set of a node v as $\texttt{Peers}_v$. A node stops making new connections when it reaches the maximum number of outgoing connections. Similarly, it rejects connection requests from other nodes when the maximum number of incoming connections is reached. The default maximum numbers of incoming and outgoing connections are both 64. We assume the delay of sending a message from a node u to another node v is $\delta(u, v) + \tilde{L}$, where $\delta(u, v)$ is a fixed delay of the path and $\tilde{L}$ is a random latency due to network conditions. We model $\tilde{L}$ as a Gaussian random variable with mean of 50ms and standard deviation of 10ms.

Once a node is synchronized with the blockchain network, it starts to receive and relay messages, including blocks and transactions. We assume that blocks are disseminated using flooding. Periodically, some transactions are generated by some source nodes (in practice, nodes receive transactions from connected clients). When a node v receives a generated or relayed transaction, it stores the transaction into a transaction pool. Every $\Delta = 0.4$ seconds, the node selects unsent transactions in its pool and announces the transaction digests to its peers. When a peer receives the digests, it scans the digest list and requests unseen transactions from the announcing node, which then sends the requested transactions. Therefore, a complete transaction sending process from u to v requires $3\delta(u, v)$ time plus some random latency.

b) Threat Model: We assume the adversary can corrupt up to a certain fraction t of nodes in the network. When the connection between an honest node and a corrupted node is built, the corrupted node is allowed to send/respond arbitrary messages to the honest node. When a corrupted node relays a message originated from an honest node, it is allowed to drop or delay the message but not forge nor modify it.

Since the design of MERCURY is independent of the node discovery process and the connection setup process, we make an extra assumption that those processes are secure and the dynamic network topology [26](i.e the join and quit of nodes) does not affect the performance of MERCURY. In contrast to our work, previous work like [23] requires setting up and terminating connections regularly to improve latency, which exposes it to the attack on the connection setup process. Designing a secure node discovery protocol and protecting honest connection in P2P networks is beyond the scope of this paper. See Eclipse Attack [27] and later defense [28] for more discussion.

c) Problem Description: We consider the transaction propagation problem in the above model. When a node v starts the periodic transaction relaying, it selects a subset of its peer list $\texttt{Peers}_v$, called relay list L. We assume the peer connection list is already given and the problem is *how to select this relay list* L. We focus on designing a scheme that decreases overall transaction broadcast latency and is resilient to malicious behaviors while minimizing bandwidth usage.

IV. SYSTEM DESIGN

A. Virtual Coordinate System in Blockchain Network

Wide area latency is dominated by geography. For example, suppose there are three nodes: node A is in Asia, while nodes B and C are in Europe. A propagation path $B-A-C$ is a poor choice, because it traverses Europe and Asia twice. The path $B-C-A$ is a better one. One solution to avoid high-latency paths is to encourage nodes to connect to nearby peers. With more locality in the relay network, a node is more likely to receive a message from a nearby peer than a distant peer, preventing detours. However, inferring physical location from an IP address can be inaccurate. Latency also depends on routing decisions of Internet Service Providers (ISPs) and network conditions, and may not reflect physical distances.

We introduce a virtual coordinate system (VCS) into our system design. A VCS embeds nodes into a metric space, such that distances between nodes in this metric space correlate with the inter-node network latency. There are many types of VCS systems. For example, centralized VCS systems rely on a set of landmark nodes [29], [30], [31]. These are not suitable to a decentralized blockchain network. The *Vivaldi* VCS [32] is fully decentralized and has been adopted by Azureus [33], a widely-used BitTorrent client. Vivaldi has a minimum overhead and fits our requirements. In Vivaldi's metric space every pair of nodes is connected by an imaginary spring. The length of the spring is the measured network latency between the nodes. If the two nodes are placed too far away from each other, the spring will apply a force to pull the nodes closer. And, if the nodes are too close, the spring will push them away. If the coordinates of all the nodes are approximately stable, the distance between every pair of nodes reflects the real network latency. Reaching a globally stable position in Vivaldi is a decentralized process. Every node initializes its coordinate, $\vec{x_i}$, as a random point near the origin in the R^{Dim} space (in practice, we set Dim to 3). They also maintain an error estimator, e_i, that indicates the relative error in round trip time (RTT) prediction. Every node periodically sends 16 queries to measure the RTT $t_{i,j}$ between itself and other random nodes, then updates its own coordinate based on the measurements and others' coordinates $\vec{x_j}$. The update algorithm is listed in Algorithm 1, where c_c and c_e are pre-defined constants (set to 0.25 in our implementation). For simplicity, Vivaldi uses a simpler definition of force: when we say node j applies the force $F_{i,j}$ to node i, it means node i is pulled or pushed by the distance of $F_{i,j}$ in the direction of $\vec{x_i}-\vec{x_j}$. Just like in Azureus [33], a node only uses the median RTT value over historical data to perform a coordinate update. In MERCURY we use the median of the last 10 observations. Usually, within 40 rounds, the system can converge to a relatively stable position [32]. We piggybacked the telemetry messages of our VCS to the node discovery messages to reduce the traffic overhead. Our system evaluation V-E confirms that the overhead of the VCS is nearly negligible.

Algorithm 1: Vivaldi Update($\vec{x_i}, \vec{x_j}, e_i, e_j, t_{i,j}$)

```
Let $w = e_i/(e_i + e_j)$;
Let $\epsilon = \frac{|\|\vec{x_i}-\vec{x_j}\| - t_{i,j}|}{t_{i,j}}$ ;
// Update the relative error indicator.
$e_i \leftarrow (c_e \times w \times \epsilon) + ((1 - c_e \times w) \times e_i)$ ;
// Update the coordinate using Hooke's
    law [34]
Let $F_{i,j} = c_c \times w \times (t_{i,j} - \|\vec{x_i} - \vec{x_j}\|)$ ;
$\vec{x_i} \leftarrow \vec{x_i} + F_{i,j} \times \text{unit}(\vec{x_i} - \vec{x_j})$ ;
```

Challenges of using VCS in Blockchain. A public blockchain platform allows anyone to join, making it necessary to consider unstable network conditions and malicious nodes. Our threat model (Section III) allows a malicious node to arbitrarily report its coordinates or delay measurements to attack the coordinates of other nodes. The original Vivaldi design does not deal with such attacks.

For the VCS component we therefore combine Vivaldi with (1) Newton [35], which is a set of rules shown to protect a VCS against attack; and (2) techniques from Azureus [33] that were shown to improve the robustness of real-world VCS deployments with tens of thousands of nodes. We also introduce (3) a new rule, the stability rule, to the system. We now describe each of these enhancements to Vivaldi.

a) Stability restriction. We introduce a new restriction based on the fact that the VCS should steadily converge. Whenever a node's coordinates reach a stable state, they should not drastically change later. We implement this rule as follows: if a node's average prediction error falls below e_{stable}, its coordinate can move by no more than F_c. We set e_{stable} to 30% and F_c to 75. Here, the unit of force is a millisecond. Any peer violating this rule will be prevented from affecting other nodes' local coordinates.

b) Force restriction. Because VCS coordinates should steadily converge, we adopt a rule from Newton [35] that any force larger than $F_{max} = 100$ is ignored. Also, a node in MERCURY keeps historical data on the magnitude of the forces imposed by its peers. If the magnitude of a new force deviates from the median $\tilde{F}$, i.e., $|f_{new}| > \tilde{F} + k \times D$, it is

Data Structure	Description
`Addr`	IP address list of known nodes. Periodically discover new addresses by asking other nodes.
$\vec{x_i}, e_i$	Vivaldi coordinate and error indicator, updated by periodically measuring RTT with random nodes.
`Coord`	Coordinates of known nodes, updated by Vivaldi measurement and IP discovery process. C_{stable} is its stable node subset that $C_{\text{stable}} = \{j \in \texttt{COORD} \| e_j < 0.4\}$.
`Cluster`	Cluster results for the stable node set C_{stable}, obtained by running K-means clustering algorithm.
`Peers`	Connected peer set. If $\|\texttt{Peers}\| <$ `MAX_PEERS`, node randomly selects IP addresses in `Addr` and tries to create connections.

TABLE I: Primary data structures maintained by a node in MERCURY.

ignored. Here, D is the median absolute deviation; we also empirically found that a value of $k = 8$ works well.

c) Centroid and Gravity. Since every node starts from the origin and the system has no outside forces, the centroid of the system, i.e., the average location, should be close to the origin. A node can approximate the centroid by computing the average location of its randomly selected peer set. In Newton [35], if a node detects the approximate centroid drifts more than $t_{drift} = 50$, the node will calculate which peer generated the largest force in the direction of the drift and ban that peer from future updating. Also, a gravity force [33] will pull a node towards the origin after an update round. The magnitude of the force is calculated by $G = (\|\vec{x_i}\|/\rho)^2$, where $\rho = 500$ is a constant scaling factor.

B. VCS-Based Propagation Scheme

We start by describing a basic propagation scheme based on clustering. Then, we will introduce several optimization heuristics to further reduce the propagation latency.

We notice that clustering is an intrinsic feature of P2P networks, including blockchain networks (e.g., Figure 1). Most nodes are located in one of the large clusters. Ideally, when a node receives a new message during propagation, the nearby peers can quickly receive the message. Our basic scheme uses the K-means algorithm to cluster nodes into K clusters. Then, when a node needs to relay transactions, it relays the transactions to $d_{cluster}$ peers located in the node's cluster.

However, if a node only relays to peers in the same cluster, the propagation will fail to reach the rest of the network. Attackers can also mount partition attacks [36] in this case. We realized that we can avoid both issues if a node also relays to a certain number of random peers. From a latency standpoint, these random peers can spread the transactions to remote peers that trigger the fast local propagation in other clusters. From a security standpoint, the random peers are selected from a large set (usually 128) and are unpredictable and difficult for an attacker to exploit. We further discuss the ratio between inter- and intra-cluster peers in Section V.

To summarize, the basic scheme in MERCURY is as follows:

1) A node discovers IP addresses of other nodes and their coordinates. It periodically fetches the new coordinates of the peers and updates its local coordinates based on the new round trip time measurements.
2) When a node receives enough coordinate updates, it locally runs K-means and classifies all stable known nodes into K clusters.
3) When a node is relaying transactions, it randomly selects $d_{cluster}$ peers from its own cluster and $(d_{max} - d_{cluster})$ peers from all of its connected peers without repetition[1]. It then sends the transactions to the selected peers.

With the help of a VCS, the cluster-based propagation scheme can reduce propagation latency while preserving security. Notice that every node computes the clusters *locally*. We allow different nodes to have different clustering results. From a global view, a true set of clusters exist and the boundaries between them are blurred at the local level. We find this to be acceptable in practice.

We further improve the basic scheme with optimization heuristics. These heuristics reduce the average latency by around 5% in our simulations.

a) Reduce the in-cluster hop distances. When a node samples $d_{cluster}$ peers, we have it sample more peers and choose $d_{cluster}$ that are the closest. The motivation is simple: the closer the peers are, the faster the propagation. However, we noticed that if a node tries to find very close peers, the overall latency increases. For example, if all nodes choose the four closest peers in their cluster, many small local structures form and there is no latency reduction.

b) Fast start of local propagation. We observed that, if a node receives new transactions from a node located in another cluster, it is likely that the propagation in the local cluster has not yet started. The best way to start local propagation is to send the transactions to many nearby peers quickly and let them begin their propagation. Therefore, if a node receives the transaction from a peer outside of its cluster, it selects $d_{cluster}$ nearest peers in the cluster to relay the transactions. These peers will not send to close nodes, since they receive the transactions from an in-cluster peer. Those nodes will still spread the transactions to other parts of the cluster and the rest of the network.

C. Early Outburst

We have described how MERCURY uses a VCS to reduce latency. Another consideration is to reduce the average number of hops. This is important because every node imposes an extra delay Δ. Increasing fan-out is one simple way to accomplish this, but this trades extra bandwidth for lower latency. In our 8000-node simulation, if all nodes choose to double the peer number from 8 to 16, the average hop number will drop from 5.50 to 4.29 and the average latency will drop from 2483.23ms to 1767.01ms. However, this method requires more bandwidth. We propose a simple strategy to reduce the average number

[1]Note that the $(d_{max} - d_{cluster})$ peers must be selected independently of clustering to make sure these nodes cannot be compromised when clustering or the VCS are attacked.

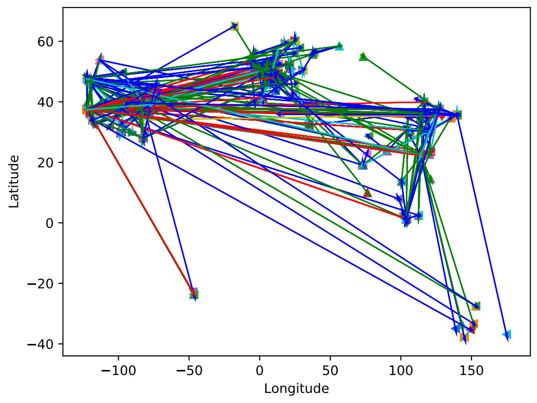


Fig. 1: An 800-node propagation tree example built by a propagation process in MERCURY. Nodes are plotted according to their physical locations [37]. The root of the transaction is located on the west coast of the United States. Red lines, blue lines and green lines denote the first hop, second hop and third hop of the propagation tree.

of hops with minimal increase in bandwidth. We start by introduce the concept of a *propagation tree*.

Progagation Tree: Given a message m, its source S and a set of nodes V, the propagation tree T is a directed tree rooted at S. Node u will be v's parent node in the tree if v receives m from u earlier than receiving it from any other nodes.

The early outburst idea builds on two observations:

a) Network traffic is dominated by the deeper layers of the propagation tree. For example, in an 8 fanout random relay network, every layer has $8\times$ more nodes than the previous layer, which means that the tree grows exponentially. Thus, the total traffic sent by a deeper layer is much larger than the traffic sent from the first few layers.

b) Nodes can choose different strategies depending on different phases in a propagation process. In other solutions, such as BlockP2P [22] and Perigee [23], a node has a small fixed relaying peer set. By contrast, we allow a node to connect to a large peer set (for example, 128 peers) and dynamically select its relaying strategy.

These two observations introduce the early outburst strategy, which can be summed as *enlarge the number of relaying peers in the early stage of the propagation*. The propagation tree height is approximately the logarithm of the total number of nodes and it is small in practice (normally around 5). When the fanout is 8, it takes two hops to reach 64 nodes (two layers in the tree). If the relay number of the source node is increased from 8 to 64, it only takes one hop to reach 64 nodes (one layers in the tree). Thus, the average hop number will be reduced by one, which immediately reduces the average latency by nearly 20% if the original tree height is around 5. More importantly, the overall network bandwidth barely increases. Suppose that the network has 8,000 nodes, then the total number of messages sent by the 8 fan-out strategy is $8 \times 8,000 = 64,000$. With the early outburst strategy, only 56 extra messages are generated, because only the root increases the fanout. It requires just 0.1% extra bandwidth as compared to the base strategy.

The easiest way to implement the early outburst strategy is to *enlarge the relaying peer set of the source node*. We will discuss the security implication of this choice in Section IV-D. Our implementation in Conflux simply lets the source node propagate the transactions to all of its connected peers (the default maximum connected peer number is 128).

The original Conflux transaction propagation protocol [25] uses a 3-phase transaction sending process that 1) a node first propagates the digest of transactions to a peer; 2) the peer returns a bitmap denoting unseen transactions; 3) the node sends those transactions to the peer. With the early outburst strategy, the source node in MERCURY can directly send the complete transactions to its peers, because it knows the transactions have not been received by the other nodes. We call this direct sending version of MERCURY as MERCURY(Direct) in evaluation (see Section V).

Algorithm 2 lists the full MERCURY propagation scheme.

Algorithm 2: MERCURY propagation in a node i.

```
Data: $\{i, j, tx\}$. Node $i$ receives $tx$ from $j$.
Result: Node $i$ relays $tx$ to all nodes in the relay list $L$.
if $j$ is a client then
    // Node $i$ receives $tx$ from a client and
       it applies early outburst strategy.
    $L \leftarrow$ Peers ;
else
    if CLUSTER$_i$=CLUSTER$_j$ then
        // If node $i$ receives $tx$ from an
           inner-cluster connection
        Randomly find at most $2d_{cluster}$ peers in Peers that
          locate in the same cluster, select at most $d_{cluster}$
          peers with smaller distances and append them to $L$
          ;
    else
        // If node $i$ receives $tx$ from a
           cross-cluster connection
        Find at most $d_{cluster}$ peers with smallest distances
          and append them to $L$ ;
    end
    Append random peers in Peers to $L$ until $|L| = d_{max}$ ;
end
```

D. Security of MERCURY

MERCURY's goal is to deliver honest nodes' transactions to as many other honest nodes as possible, while minimizing latency. We assume the adversary's goal is to prevent honest nodes from receiving transactions or delay the process.

Based on our threat model (Section III), the adversary has two ways to attack propagation: (1) disseminate new messages, and (2) drop or delay messages from honest nodes.

MERCURY can be thought of as a two-layer network. The bottom layer includes the random links and the upper layer includes the carefully selected links. A node in MERCURY will always propagate messages to at least $d_{random} = d_{max} - d_{cluster}$ totally random peers. As a result, in the worst case, the bottom layer provides the same security guarantee as a

d_{random}-degree random propagation network. That is, if the clustering mechanism is completely under attacker's control, then the system performs no worse than a random graph. This is demonstrated in our experimental results (see Section V-D).

a) Fallback Mechanism.: A massive network attack conducted by a large number of malicious nodes may break all of the VCS security mechanisms from Section IV-A. Based on the historical round-trip-time measurements, an honest node can assess the accuracy of its local coordinate by the coordinate absolute error indicator e_i. If the coordinate's accuracy is poor, i.e., $e_i > e_{\text{safe}} = 0.4$, the node falls back to the baseline random propagation scheme.

b) Validity and uniqueness check on transactions.: The attacker may also try to trigger large volume of traffic in the network and conduct a Denial-of-Service attack by exploiting the early outburst strategy. To prevent such an attack, before the propagation, a node will examine the validity and the uniqueness of the transactions. If an adversarial client sends many invalid or duplicate transactions, propagation will not be triggered. If a malicious client sends many valid original transactions to the targeted nodes, it has to pay the transaction fee, which makes the attack expensive. If an attacker is using a fake client to relay transactions it receives from other nodes, it is possible that the target node has already received the same transactions. In this case, the nodes can detect the duplication and disconnect from the attacking client.

c) Traffic monitoring.: Every node monitors its incoming and outgoing network traffic. A node maintains a throttle on every peer link. If a node suddenly receives large network traffic from a client, it refuses to perform the relaying service for the client. Also, if its outgoing traffic is abnormally high, it will temporally disable the early outburst strategy to avoid network congestion.

V. EVALUATION

We aim to answer the following questions in our evaluation:

- What is the best configuration for MERCURY?
- How does MERCURY improve the latency compared to alternative propagation strategies under the same constraint of network traffic usage?
- How does MERCURY perform under attacks?

We evaluated MERCURY with two types of experiments. First, we ran simulations to compare MERCURY with existing propagation strategies. Then, we deployed our MERCURY prototype as part of a high-throughput open-source blockchain, Conflux, to evaluate MERCURY in a real environment.

A. Simulation Setup

We implement MERCURY along with other propagation schemes, including random propagation, BlockP2P [22] and Perigee [23] on the same P2P network simulation code base in C++. Random propagation is a standard design in existing blockchain protocols. BlockP2P-EP [38] shares the propagation scheme with BlockP2P, so we only compare to BlockP2P.

We crawl the full-node list of Ethereum[37], which consists of 8,000 nodes. We convert the IP addresses from the node list to approximate geo-locations using an IP database [39]. Perigee [23] used a similar setup with a Bitcoin dataset. The latency between a node pair (u, v) is $\delta(u, v)+\tilde{L}$, where $\delta(u, v)$ is the basic latency approximated by the geo-distance and the random latency $\tilde{L}$ follows a Gaussian distribution with mean of 50ms and standard deviation of 10ms.

We run 100 rounds of simulation and report the averages. We randomly selected a full node as the source for each round. When a node receives a transaction for the first time, a delay of 200ms is added to simulate the average wait on batching interval (recall that a batch interval is 400ms). The node then disseminates the transactions. We repeat this process until no active transmission happens. We let the nodes run 100 rounds of Vivaldi updates. Then, we run K-means based on the virtual coordinates to partition the nodes into K clusters. The same preparation process is used for BlockP2P. Perigee has an extra 64-round transaction propagation warm-up phase. We choose the UCB scoring rule [23] as the default setting for Perigee as it performs the best in our simulation.

We plot latency results using CDFs. For example in Fig 2, a point with latency of 2,000ms at 0.5 *received node ratio* means that 50% of the nodes in the system received all the transaction within 2,000ms.

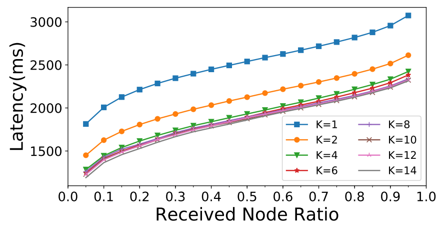
B. *Configuration for* MERCURY

We first consider the MERCURY's configuration. MERCURY has two key components: the cluster-based scheme, MERCURYCLUSTER, and the early outburst strategy. MERCURYCLUSTER has two parameters – the cluster number K and the inner cluster peers $d_{cluster}$. The default setting is $K = 8$, $d_{cluster} = 4$ and $d_{max} = 8$. To better demonstrate the impact of clustering, we disable the early outburst optimization when testing the MERCURYCLUSTER configuration. We also disable other optimizations described in Section IV-B.

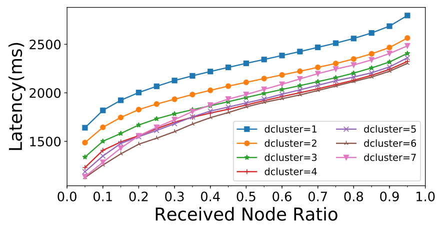
We test the latency of the coordinate-based scheme with different cluster numbers K. The results are shown in Fig 2a. By increasing the cluster number from 2 to 8, the overall latency goes down. When the cluster number is above 8, we do not observe significant improvement. This is because the nodes naturally fall into large geolocation-based clusters (e.g., Fig 1) and splitting large and dense clusters into smaller sets only brings small improvement. We set $K = 8$ as our default.

Next, we examine the influence of the inner cluster peer number $d_{cluster}$. We fixed $d_{max} = 8$. The nodes relay the transaction to $d_{cluster}$ peers that are located in the same cluster and $8 - d_{cluster}$ randomly selected peers. A higher $d_{cluster}$ induces more locality. Meanwhile, random relaying invokes cross-cluster propagation. We plot the result in Fig 2b. It shows that when $d_{cluster}$ is between 4 and 6, the overall latency is small. We choose $d_{cluster} = 4$ as our default.

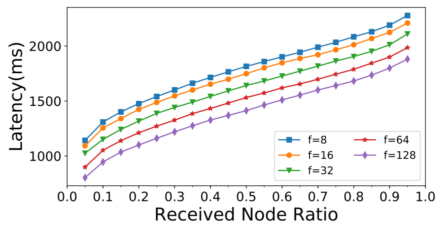
The results for early outburst strategy are shown in Fig 2c. We observe that the fanout number, f, for the source node can be as large as the maximum number of connection (128) and it achieves the lowest overall latency.



(a) Latency results for different cluster number, K, values. Latency is lowest when K is 8 or higher.



(b) Latency result for different inner cluster peer number, $d_{cluster}$, values.



(c) Latency result with different early outburst fanout f values. The large the f values, the lower the latency.

Fig. 2: Experimental results from exploring different MERCURY configuration parameter values.

	Random	BlockP2P	Perigee	MERCURYCLUSTER	MERCURY
Average latency (ms)	2483.23	2708.83	1977.74	1774.73	1391.17
Ratio vs. Random	100%	109.08%	79.67%	71.47%	56.02%

TABLE II: Comparison between propagation schemes under similar traffic usage.

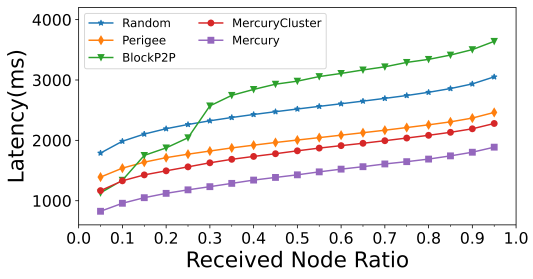


Fig. 3: Comparison with existing propagation schemes under similar traffic usage. MERCURY with different setups beats other schemes.

C. Comparison with Alternative Schemes

We compare MERCURY with random propagation, BlockP2P, and Perigee. We set the default maximum relay peer number to 8 so every approach consumes roughly equal network traffic. BlockP2P tries to build a routing structure based on clustering, but every cluster has only one "routing node" that can communicate with other clusters. Perigee uses an online learning concept to grade every peer based on their propagation performance. If a peer shows bad performance, the node will disconnect from the peer. For MERCURY, we record the results for MERCURYCLUSTER and MERCURY that combines the cluster propagation scheme and early outburst strategy. See Fig 3 and also Table II for the results.

Compared to all three prior approaches (Perigee, Random, BlockP2P), MERCURYCLUSTER and MERCURY both show an improvement in average latency. For example, MERCURY average latency improves over the three schemes by 30%, 44% and 49%. BlockP2P performs worse than Random mechanism in our simulation. We observe that it propagates to local groups quickly, but the cross-cluster communication is restricted. The reason of the discrepancy could be that topologies of the simulation networks – we use the IPs from the real Ethereum system, while the original BlockP2P experiments are conducting on artificially generated topology.

D. Performance under Attacks

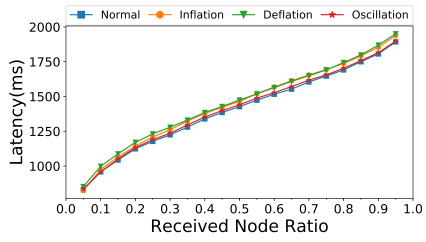
We now consider MERCURY's performance during attacks. We consider that malicious nodes can: (1) disrupt the virtual coordinate system by lying about their coordinates and delaying RTT measurements, or (2) not forward the message to attack the cluster-based propagation scheme.

First, we conduct the experiment for all three kinds of attacks introduced by [35] against the virtual coordinate system: (1) **inflation attack**: attackers lie about having very large coordinates, far away from the origin; (2) **deflation attack**: attackers lie about having small coordinates that are near the origin; and, (3) **oscillation attack**: attackers report random chosen coordinates and random delay RTT measurements. These attacks make benign nodes unable to update their coordinates accurately.

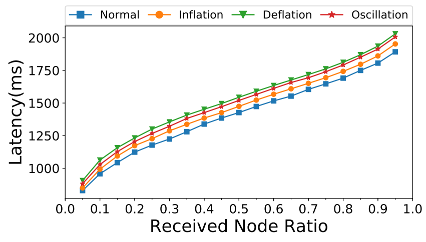
The performance of MERCURY under these coordinate-based attacks is shown in Fig 4. We bounded the number of malicious nodes to 10%, 30% and 49% of total nodes. As the percentage of attackers increases, propagation latency increases gradually. But, even with 49% malicious nodes, MERCURY continues to propagate transactions with only a slight increase in latency.

We also evaluate the MERCURY performance with different percentage of malicious nodes that do not broadcast message, namely, the **no-response attack**.

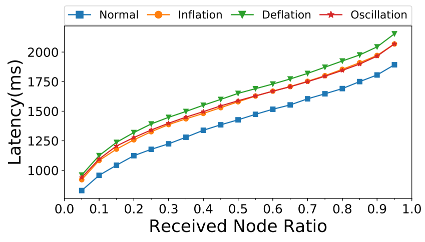
We compare MERCURY performance with that of random and Perigee. The results are shown in Fig 5. The three plots are CDFs that illustrate the fraction of honest nodes that have received the transaction within a certain time budget. When the number of malicious nodes increases, the network has more dropped messages, but MERCURY keeps the latency low. Even when 49% of the nodes are malicious, the performance of MERCURY remains much better than the other schemes. BlockP2P sets a "routing node" for every cluster. If the node is malicious, the cluster is separated. Our experiments show that BlockP2P can not operate under this attack when there are more than 15% malicious nodes since over 20% of the nodes cannot receive transactions.



(a) Latency - 10% malicious nodes

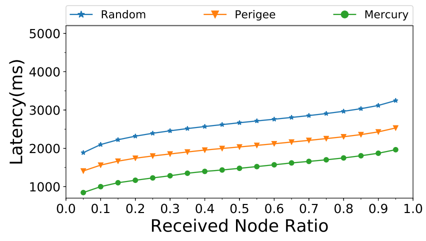


(b) Latency - 30% malicious nodes

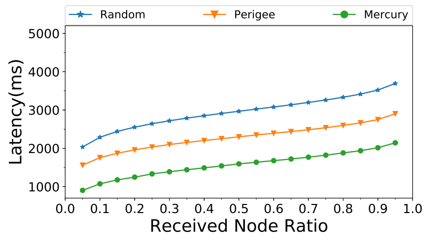


(c) Latency - 49% malicious nodes

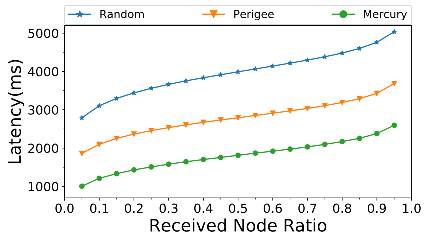
Fig. 4: Performance of MERCURY under three kinds of **coordinate-based attacks**: inflation, deflation, and oscillation.



(a) Latency - 10% malicious nodes



(b) Latency - 30% malicious nodes



(c) Latency - 49% malicious nodes

Fig. 5: Performance of random, Perigee, and MERCURY under the **no-response attack**.

Finally, we evaluate MERCURY in the worst-case scenario. In this experiment, we assume the attacker can manipulate the VCS system and fully control the clustering mechanism, such that all honest nodes cluster themselves with all the corrupted nodes, while deceiving the honest nodes that the VCS system still has high accuracy. We measure the latency under normal and attacking scenarios and compare MERCURY with random broadcasting with 4 peers or 8 peers, which is least impacted by this type of attack. Two different specifications of MERCURY are also compared: MERCURY(Cluster) with early outburst optimization disabled, and the MERCURY(dcluster=3) in which $d_{cluster}$ is set to 3 (the standard MERCURY sets $d_{cluster} = 4$).

Fig 6 plots the results. The y-axis plots latency without the attack while the x-axis plots latency with the attack. MERCURY has the lowest latency without the attack. With the attack, MERCURY's performance is ∼10% better than 4-random broadcasting. MERCURY(Cluster)'s performance is the same as the 4-random broadcasting. Surprisingly, when we set the $d_{cluster} = 3$, the performance of MERCURY is roughly the same as 8-random broadcasting under the attack and only slightly worse than standard MERCURY without the attack. This configuration may be best when the system designer is prioritizing performance in an adversarial environment.

E. Experiments for Conflux Network

To study MERCURY's behavior in a real environment, we implemented MERCURY in the Conflux full node codebase. We ran experiments on 1,000 m5.xlarge VMs on Amazon EC2. Each VM has 2 cores, 8GB memory and runs 1 Conflux full node. We randomly assign four neighbours for each node during initialization. Then, the nodes discover other nodes by exchanging known peer sets, until they reach the maximum peer number (128). Since the latency inside the AWS internal network is homogeneously low, we add extra latency to both TCP and UDP messages to emulate a real network environment. The latency for every pair of nodes is generated from the Ethereum dataset. Each node runs the secure Vivaldi VCS algorithm to obtain its coordinates. During the experiment, we randomly generate payment transactions for our workload under a specified global throughput for the entire network. Each node evenly contributes to transaction generation. The average transaction size is 100 bytes and the throughput is set to 2,000 TPS. Each node produces and propagates a batched announcement every 0.5 seconds. After the test, we use map-reduce to record the transaction arrival time in every node, then sort the recorded time. We randomly sample a node and monitor its traffic usage during the test.

We test four strategies: (1) original Conflux using random forwarding; (2) the MERCURYCLUSTER strategy; (3) the full MERCURY with early outburst optimization; and, (4) the MERCURY(Direct) that has the same strategy as MERCURY, but the source node directly sends transactions to peers. The latency results are plotted in Fig 7. We notice that the overall latency improvement of MERCURY matches our simulation results. MERCURY improves on average latency by 41% over Conflux. MERCURY(Direct) further improves the latency because the source node only sends one message to its peers.

Table III shows the "Average Tx packed to block time" results that represent the earliest time for a transaction to be verified and packed into a mined block. This metric indicates the impact of propagation schemes on the overall system performance. MERCURY and MERCURY(Direct) show a 38% and a 46% improvement in this metric, respectively.

	Original	MercuryCluster	Mercury	Mercury(Direct)
Average latency (s)	2.14	1.72	1.27	0.93
95%th percentile received time (s)	2.68	2.46	1.68	1.42
Average txn packed to block time (s)	2.58	2.08	1.62	1.40

TABLE III: Results from Conflux network experiments. *"Txn packed to block time"* denotes the earliest time when a transaction is verified by a node and is packed into a mined block.

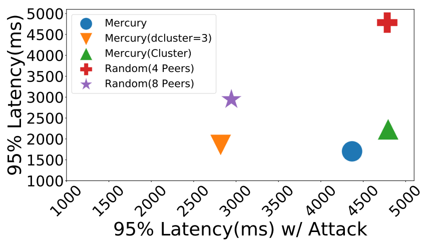


Fig. 6: Performance of Mercury when the attacker controls the VCS system.

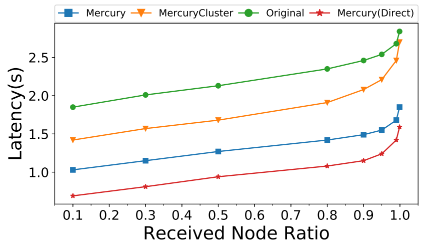


Fig. 7: Propagation Latency from the Conflux network experiments.

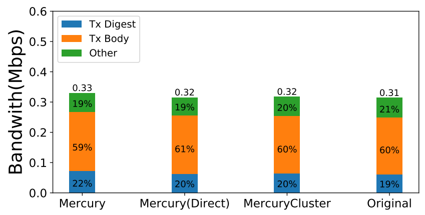


Fig. 8: Network traffic from the Conflux network experiments.

Fig 8 plots the network traffic usage. The traffic overhead is larger than the theoretical value but still modest (∼5%). The extra bandwidth is due to the header of the transaction digest (TXD) messages. Originally, a node only sends 16 transaction digest messages per second (two rounds of relaying). With early outburst, the nodes need to send more than 256 TXD messages per second and their headers induce extra traffic. The source node in Mercury(Direct) sends the transaction directly, avoiding the overhead of TXD messages.

VI. Related Work

a) Blockchain P2P Routing: Different routing strategies have been proposed to speed up the message delivery in blockchain systems. Kadcast [40] introduces structural P2P overlay network with DHT-like propagation. It does not consider the underlying node connection latency, so its performance is only slightly better than random topology [23]. BlockP2P [22], [38] uses clustering-proximity connection to accelerate block propagation. Urocissa [41] and FRING [42] share a similar static hierarchical design as BlockP2P. Those designs are susceptible to attack because once the attacker controls the important entry nodes, the network is partitioned. CougaR [43] designs a routing protocol based on simple RTT measurements, but provides no protection against adversaries. In [44], every node grades their peers by local message arrival time and attempts to select peers that have fast connections. Later, Perigee [23] generalized such strategy with online learning technique to further optimize the peer set. Our results show that Mercury is 30% faster than Perigee on average. Peri [45] discusses the phenomenon that an individual node can strategically select its peers to gain advantages in block/transaction propagation. By contrast, we focus on designing a protocol for the entire network.

b) Transaction Relay Protocols: Bitcoin uses a flooding-based protocol for transaction and block propagation that makes it hard to scale. with the increasing number of the connected peers per node. Erlay [46] proposes a combination of low-fanout flooding and set reconciliation to reduce bandwidth. Shrec [25] introduces a new relay protocol and transaction short Id encoding with a hybrid hashing scheme that has low collision rate and is resilient to collision attacks. Mercury is complementary to both protocols. For example, we implemented Mercury on Conflux, which uses Shrec as its relay protocol.

c) Block Distribution Network: Fibre [47], Falcon [48] and BloXroute [49] rely on a high-speed block relay network. These solutions are not fully decentralized and require nodes to place trust in the relay network.

d) Traditional P2P Network Multicast: Many schemes have been proposed to improve P2P multicast efficiency, including traffic usage and overall latency, in traditional P2P network. For example, DHTs can improve multicast [50]. Tree-based solutions have been proposed [51], [52] that nodes collaboratively build multiple propagation trees. Location-aware topology matching (LTM) scheme has been proposed to eliminate low efficiency connection in a P2P network [53]. These techniques cannot directly apply to blockchain networks because they are not intended for adversarial environments.

VII. Conclusion

High-performance blockchain systems require efficient and secure transaction broadcasting. Mercury combines two novel ideas — using a virtual coordinate system to organize propagation structure and the use of an early outburst strategy — to reduce the latency of transaction propagation. Our experiments show that Mercury is robust to attacks and improves transaction propagation latency by up to 56% in a real high-throughput blockchain system, while introducing less than 5% bandwidth overhead. The authors have provided public access to their code at https://github.com/wuwuz/P2PNetwork.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (2022YFB2702300). Mingxun Zhou was a research assistant at Shanghai Qizhi Institute during the project.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[2] "Ethereum White Paper," https://ethereum.org/en/whitepaper/, 2020.
[3] "Blockchain in Finance," https://home.kpmg/uk/en/home/insights/2019/03/bffb-blockchain-in-finance.html, 2019.
[4] F. Schär, "Decentralized finance: On blockchain-and smart contract-based financial markets," *Available at SSRN 3571335*, 2020.
[5] M. Casey, J. Crane, G. Gensler, S. Johnson, and N. Narula, "The impact of blockchain technology on finance: A catalyst for change," 2018.
[6] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, "Blockchain technology and its relationships to sustainable supply chain management," *International Journal of Production Research*, 2019.
[7] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *IEEE Healthcom*, 2016.
[8] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.
[9] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.
[10] C. Li, P. Li, D. Zhou, W. Xu, F. Long, and A. Yao, "Scaling nakamoto consensus to thousands of transactions per second," *arXiv preprint arXiv:1805.03870*, 2018.
[11] Y. Sompolinsky, S. Wyborski, and A. Zohar, "Phantom ghostdag: a scalable generalization of nakamoto consensus," in *ACM AFT*, 2021.
[12] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *ACM CCS*, 2019.
[13] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *ACM SOSP*, 2017.
[14] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Development Foundation*, 2015.
[15] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *ACM CCS*, 2016.
[16] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *EATCS DISC*, 2017.
[17] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *IEEE S&P*, 2018.
[18] D. Leung, A. Suhl, Y. Gilad, and N. Zeldovich, "Vault: Fast bootstrapping for cryptocurrencies," in *USENIX NDSS*, 2018.
[19] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *USENIX NSDI*, 2019.
[20] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and probabilistic leaderless bft consensus through metastability," *arXiv preprint arXiv:1906.08936*, 2019.
[21] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0. 8.13," 2018.
[22] W. Hao, J. Zeng, X. Dai, J. Xiao, Q. Hua, H. Chen, K.-C. Li, and H. Jin, "Blockp2p: Enabling fast blockchain broadcast with scalable peer-to-peer network topology," in *Green, Pervasive, and Cloud Computing*, 2019.
[23] Y. Mao, S. Deb, S. B. Venkatakrishnan, S. Kannan, and K. Srinivasan, "Perigee: Efficient peer-to-peer network design for blockchains," in *ACM PODC*, 2020.
[24] "Ethereum Transaction Exchange," https://github.com/ethereum/devp2p/blob/master/caps/eth.md#transaction-exchange, 2022.
[25] Y. Han, C. Li, P. Li, M. Wu, D. Zhou, and F. Long, "Shrec: Bandwidth-efficient transaction relay in high-throughput blockchain systems," in *ACM SoCC*, 2020.
[26] J. Augustine, G. Pandurangan, P. Robinson, S. Roche, and E. Upfal, "Enabling robust and efficient distributed computation in dynamic peer-to-peer networks," in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE, 2015, pp. 350–369.
[27] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *USENIX Security*, 2015.
[28] M. Tran, A. Shenoi, and M. S. Kang, "On the {Routing-Aware} peering against {Network-Eclipse} attacks in bitcoin," in *Usenix Security*, 2021.
[29] T. S. Eugene Ng, "Predicting internet network distance with coordinates-based approaches," in *IEEE INFOCOM*, 2002.
[30] T. S. E. Ng and H. Zhang, "A network positioning system for the internet," in *Conference on Usenix Technical Conference*, 2004.
[31] T. Harris, "Lighthouses for scalable distributed location," in *Peer-to-Peer Systems II, Second International Workshop, IPTPS*, 2003.
[32] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," *ACM SIGCOMM*, 2004.
[33] J. Ledlie, P. Gardner, and M. I. Seltzer, "Network coordinates in the wild." in *NSDI*, vol. 7, 2007, pp. 299–311.
[34] "Hooke's law," https://en.wikipedia.org/wiki/Hooke's_law, 2020.
[35] J. Seibert, S. Becker, C. Nita-Rotaru, and R. State, "Securing virtual coordinates by enforcing physical laws," *IEEE/ACM ToN*, 2014.
[36] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen, "Partitioning attacks on bitcoin: Colliding space, time, and logic," in *IEEE ICDCS*, 2019.
[37] "Ethnodes," https://www.ethernodes.org/, 2021.
[38] W. Hao, J. Zeng, X. Dai, J. Xiao, Q.-S. Hua, H. Chen, K.-C. Li, and H. Jin, "Towards a trust-enhanced blockchain p2p topology for enabling fast and reliable broadcast," *IEEE TNSM*, 2020.
[39] "Really free geoip," https://reallyfreegeoip.org/, 2021.
[40] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *ACM AFT*, 2019.
[41] Y. Zhu, C. Hua, D. Zhong, and W. Xu, "Design of low-latency overlay protocol for blockchain delivery networks," in *IEEE WCNC*, 2022.
[42] H. Qiu, T. Ji, S. Zhao, X. Chen, J. Qi, H. Cui, and S. Wang, "A geography-based p2p overlay network for fast and robust blockchain systems," *IEEE Transactions on Services Computing*, pp. 1–14, 2022.
[43] E. Kolyvas and S. Voulgaris, "Cougar: Fast and eclipse-resilient dissemination for blockchain networks," in *ACM DEBS*, 2022.
[44] Y. Aoki and K. Shudo, "Proximity neighbor selection in blockchain networks," in *IEEE Blockchain*, 2019.
[45] W. Tang, L. Kiffer, G. Fanti, and A. Juels, "Strategic latency reduction in blockchain peer-to-peer networks," *arXiv preprint arXiv:2205.06837*, 2022.
[46] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *ACM CCS*, 2019.
[47] "FIBRE," https://bitcoinfibre.org/, 2020.
[48] "Falcon," https://www.falcon-net.org/, 2020.
[49] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer, "bloxroute: A scalable trustless blockchain distribution network whitepaper," *IEEE Internet Things J.*, 2018.
[50] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, "Efficient broadcast in structured p2p networks," in *International workshop on Peer-to-Peer systems*. Springer, 2003, pp. 304–314.
[51] T. Perez, J. Solano, and I. Stojmenovic, "Lmst-based searching and broadcasting algorithms over internet graphs and peer-to-peer computing systems," in *2007 IEEE SPCOM*, 2007.
[52] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast," in *IEEE ICNP*, 2006.
[53] Yunhao Liu, Xiaomei Liu, Li Xiao, L. M. Ni, and Xiaodong Zhang, "Location-aware topology matching in p2p systems," in *IEEE INFOCOM*, 2004.