Strategic Latency Reduction in Blockchain Peer-to-Peer Networks

WEIZHAO TANG, Carnegie Mellon Univeristy & IC3, USA
LUCIANNA KIFFER, ETH Zurich, Switzerland
GIULIA FANTI, Carnegie Mellon Univeristy & IC3, USA
ARI JUELS*, Cornell Tech, IC3, & Chainlink Labs, USA

Most permissionless blockchain networks run on peer-to-peer (P2P) networks, which offer flexibility and decentralization at the expense of performance (e.g., network latency). Historically, this tradeoff has not been a bottleneck for most blockchains. However, an emerging host of blockchain-based applications (e.g., decentralized finance) are increasingly sensitive to latency; users who can reduce their network latency relative to other users can accrue (sometimes significant) financial gains.

In this work, we initiate the study of strategic latency reduction in blockchain P2P networks. We first define two classes of latency that are of interest in blockchain applications. We then show empirically that a strategic agent who controls only their local peering decisions can manipulate both types of latency, achieving 60% of the global latency gains provided by the centralized, paid service bloXroute, or, in targeted scenarios, comparable gains. Finally, we show that our results are not due to the poor design of existing P2P networks. Under a simple network model, we theoretically prove that an adversary can always manipulate the P2P network's latency to their advantage, provided the network experiences sufficient peer churn and transaction activity.

CCS Concepts: • **Security and privacy** → *Network security*; • **Networks** → *Network measurement*; **Network algorithms**.

Additional Key Words and Phrases: blockchains, P2P networks, strategic manipulation

ACM Reference Format:
Weizhao Tang, Lucianna Kiffer, Giulia Fanti, and Ari Juels*. 2023. Strategic Latency Reduction in Blockchain Peer-to-Peer Networks. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 2, Article 32 (June 2023), 33 pages. https://doi.org/10.1145/3589976

1 INTRODUCTION

Most permissionless blockchains today run on peer-to-peer (P2P) communication networks due to their flexibility and distributed nature [20, 32]. These benefits of P2P networks typically come at the expense of network performance, particularly the latency of message delivery [18, 20, 46]. Historically, this has not been a bottleneck because most permissionless blockchains are currently performance-limited not by network latency, but by the rate of block production and transaction confirmation, both of which are superlinear in network latency and dictated by the underlying consensus mechanism [12, 18, 25, 31, 51]. For this reason, network latency has not traditionally

*The author contributed to this work in his Chainlink Labs role.

Authors' addresses: Weizhao Tang, Carnegie Mellon Univeristy & IC3, USA, wtang2@andrew.cmu.edu; Lucianna Kiffer, ETH Zurich, Switzerland, lkiffer@ethz.ch; Giulia Fanti, Carnegie Mellon Univeristy & IC3, USA, gfanti@andrew.cmu.edu; Ari Juels*, Cornell Tech, IC3, & Chainlink Labs, USA, juels@cornell.edu.

2476-1249/2023/6-ART32 $15.00
https://doi.org/10.1145/3589976

been a first-order concern in blockchain P2P networks, except where substantial latency on the order of many seconds or even minutes raises the risk of forking [15].

Emerging concerns in blockchain systems, however, are beginning to highlight the importance of *variations in fine-grained network latency*—e.g., on the order of milliseconds—among nodes in blockchain P2P networks. These concerns are largely independent of the underlying consensus mechanism. They revolve instead around strategic behaviors that arise particularly in smart-contract-enabled blockchains, such as Ethereum, where decentralized finance (DeFi) applications create timing-based opportunities for financial gain. Key examples of such opportunities include:

- **Arbitrage:** Many blockchain systems offer highly profitable opportunities for *arbitrage*, in which assets are strategically sold and bought at different prices on different markets (or at different times) to take advantage of price differences [34]. Such arbitrage can involve trading one cryptocurrency for another, buying / selling mispriced non-fungible tokens (NFTs), etc., on blockchains and/or in centralized cryptocurrency exchanges. Strategic agents performing arbitrage can obtain an important advantage through low latency access to blockchain transactions.
- **Strategic transaction ordering:** In many public blockchains, strategic agents (who do not validate blocks) can profit by ordering their transactions in ways that exploit other users—a phenomenon referred to as *Miner-Extractable Value* (MEV) [10, 16]. For example, strategic agents may *front-run* other users' transactions, i.e., execute strategic transactions immediately before those of victims, or *back-run* key transactions—e.g., oracle report delivery or token sales—to gain priority in transaction ordering. Small network latency reductions can allow an adversary to observe victims' transactions before competing strategic agents do. With more adversarial nodes and links available, it may even attempt to choose peers in order to frontrun as many pairs of sources and destinations as possible.
- **Improved block composition:** Miners (or validators) rely on P2P networks to observe user transactions, which they then include in the blocks they produce. Which transactions a miner includes in a block determines the fees it receives and thus its profits. Therefore the lower the latency a miner experiences in obtaining new transactions, the higher its potential profit.[1]
- **Targeted attacks:** The security of nodes in blockchain P2P networks may be weakened or compromised by adversaries' discovery of their IP addresses. For example, a node may issue critical transactions that update asset prices in a DeFi smart contract. Discovery of its IP address could result in denial-of-service (DoS) attacks against it. Similarly, a user who uses her own node to issue transactions that suggest possession of a large amount of cryptocurrency could be victimized by targeted cyberattacks should her IP address be revealed.

We show that small differences in transaction latencies can help an adversary distinguish between paths to a victim—and ultimately even discover the victim's IP address. There is a strong incentive, therefore, for agents to minimize the latency they experience in P2P networks. **In this work, we initiate the study of strategic latency reduction in blockchain P2P networks**.

1.1 Types of network latency

In our investigations, we explore two types of latency that play a key role in blockchain P2P networks: *direct latency* and *triangular latency* (formal definitions in §3). We demonstrate these types of latencies in Table 1.

Direct latency refers to the latency with which messages reach a listener node from one or more vantage points—in other words, source latency. We consider two variants:

[1]Receiving blocks from other miners quickly is also beneficial. To reduce forking risks, however, miners or validators often bypass P2P networks and instead use cut-through networks to communicate blocks to one another [7].

- *Direct targeted latency* is the delay between a transaction being pushed onto the network by a specific target node (the *victim*) and a node belonging to a given agent receiving the transaction, averaged across all transactions produced by the victim. We can view targeted latency either in terms of an absolute latency, or a relative latency compared to other nodes receiving the same transaction; we focus in this paper on relative latency.
- *Direct global latency* for a given agent's node refers to targeted latency for that node averaged over all source nodes in the network. In other words, there is no single target victim.

Triangular latency is a second type of latency that corresponds to a node's ability to inject itself between a pair of communicating nodes. Low triangular latency is motivated by (for example) a node's desire to front-run another node's transactions. We consider two forms of triangular latency:

- *Triangular targeted latency* refers to the ability of an agent's node v to “shortcut" paths between a sender s (e.g., the creator of a transaction) and a receiver r (e.g., a miner or miners). That is, suppose s creates a transaction m that is meant to reach a (set of) target nodes r. Triangular targeted latency measures the difference between the total delay on path $s \to v \to r$ and the smallest delay over all paths from $s \to r$ not involving v. Given negative triangular targeted latency, an agent can front-run successfully by injecting a competing transaction m' into v upon seeing a victim's transaction m.
- *Triangular global latency* refers to the triangular relative targeted latency averaged over all source-destination pairs in the network.

Direct latency and triangular latency are related, but not identical, as we explain in §3 and §6.

1.2 Latency-reduction strategies

High-frequency trading (HFT) firms in the traditional finance industry compete aggressively to reduce the latency of their trading platforms and connections to markets—in some cases, by nanoseconds—using technologies like hollow-core fiber optics [43] and satellite links [42].

Such approaches are not viable for blockchain P2P networks, in which *network topology* matters more than individual nodes' hardware configurations. These networks are also permissionless and experience high churn rates, and are controlled by different parties.

In practice, nodes may rely on proprietary, *cut-through* networks to learn transactions or blocks quickly. Miners maintain private networks [7]. There are also public, paid cut-through networks, such as bloXroute [3]. An agent in a blockchain P2P network, however, can also act strategically by means of *local actions, namely choosing which peers a node under its control connects to.*

Peri: A recent protocol called Perigee [35] leverages agents' ability in blockchain P2P networks to control their peering to achieve network-wide latency improvements. In the Perigee protocol, *every* node favors peers that relay blocks quickly and rejects peers that fail to do so. We observe that Perigee-like strategies can instead be adopted by *individual strategic agents*. We adopt a set of latency-reduction strategies called ***Peri***[2] that modify Perigee with optimizations tailored for the individual-agent setting. In particular, we show that agents using Peri can gain a latency advantage over other agents in both direct and triangular latency, including targeted and global variants.

1.3 Our contributions

We explore techniques for an agent to reduce direct measurement latency and triangular latency in a blockchain P2P network using only *local peering choices*, i.e., those of node(s) controlled by the agent. **Overall, we find that strategic latency reduction is possible, both in theory and in practice.** Our contributions are as follows.

[2]A mischievous winged spirit in Persian lore.

- **Practical strategic peering:** We demonstrate empirically and in simulation that the strategic peering scheme Peri can achieve direct latency improvement compared to the current Ethereum P2P network.[3] We instrument the `geth` Ethereum client to measure direct measurement latency and implement Peri. We observe direct global latency and direct targeted latency reductions each of about 11 ms—over half the reduction observed for bloXroute for direct latency and comparable to that of bloXroute's paid service for targeted latency. We additionally show that Peri can discover the IP address of a victim node 7× more frequently than baseline approaches.
- **Hardness of triangular-latency minimization:** We explore the question of strategic triangular-latency reduction via a graph-theoretic model. We show that solving this problem optimally is NP-hard, and the greedy algorithm cannot approximate the global solution. These graph-theoretic results may be of independent interest. We show experimentally in simulations, however, that Peri *is* effective in reducing such latency.
- **Impossiblity of strategy-proof peering protocols:** Within a theoretical model, we prove that *strategy-proof P2P network design is fundamentally unachievable*: For any default peering algorithm used by nodes in a P2P network, as long as the network experiences natural churn and a target node is active, a strategic agent can always reduce direct targeted latency relative to agents following the default peering algorithm. Specifically, with probability at least $1 - \varepsilon$ for any $\varepsilon \in (0, 1)$, an agent can connect directly to a victim in time $O(\varepsilon^{-1} \log^2(\varepsilon^{-1}))$.

2 BACKGROUND AND RELATED WORK

Many of the latency-sensitive applications discussed in §1 arise in decentralized finance (DeFi) and require a blockchain platform that supports general smart contracts. We therefore focus on the Ethereum network (in particular, the eth1 or the execution layer); however, the concepts and analysis apply to other blockchain P2P networks.

Network Formation. Ethereum, like most permissionless blockchains, maintains a P2P network among its nodes. Each node is represented by its enode ID, which encodes the node's IP address and TCP and UDP ports [49]. Nodes in the network learn about each other via a node discovery protocol based on the Kademlia distributed hash table (DHT) protocol [36]. To bootstrap after a quiescent period or upon first joining the network, a node either queries its previous peers or hard-coded bootstrap peers about other nodes in the network. Specifically, it sends a `FINDNODE` request using its own enode ID as the DHT query seed. The node's peers respond with the enode IDs and IP addresses of those nodes in their own peer tables that have IDs closest in distance to the query ID. Nodes use these responses to populate their local peer tables and identify new peers.

Transaction Propagation. When a user wants to make a transaction, they first cryptographically sign the transaction using a fixed public key, then broadcast the signed transaction over the P2P network using a simple flooding protocol [6]. Each node v, upon seeing a new transaction m, first checks the transaction's validity; if the transaction passes validity checks, it is added to v's local `TxPool` of unconfirmed transactions. Then, v executes a simple three-step process: (1) It chooses a small random subset $\{u_i\}$ of its connected peer nodes; (2) v sends a transaction hash $H(m)$ to each peer u_i; (3) If u_i has already seen transaction m, it communicates this to the sender v; otherwise it responds with a `GetTx` request, and v in turn sends m in full [49]. There are two main sources of latency in the P2P network: (1) network latency, which stems from sending messages over a P2P overlay of the public Internet, (2) node latency, which stems from local computation at each node before relaying a transaction. We treat node latency as fixed and try to manipulate network latency.

[3]We explicitly do not consider the design of Peri to be a main contribution, as the design is effectively the same as Perigee. We make a distinction between the two purely to highlight their differing goals and implementation details.

Transaction Confirmation. After transactions are disseminated over the P2P network, *miner* nodes confirm transactions from the `TxPool` by compiling them into blocks [40, 49]. When forming blocks, miners choose the order in which transactions will be executed in the final ledger. This ordering can have significant financial repercussions (§2.0.1). Miners typically choose the ordering of transactions in a block based on a combination of (a) transactions' time of arrival, and (b) incentives and fees associated with mining a transaction in a particular order. In Ethereum, each block has a base fee, which is the minimum cost for being included in the block. In addition, transaction creators add a tip (priority fee), which is paid to the miner to incentivize inclusion of the transaction in a block [8, 45]. These fees are commonly called *gas fees* in the Ethereum ecosystem.

2.0.1 Example: The role of latency in arbitrage. Among the applications in §1, the interplay between network latency and arbitrage is particularly delicate. To perform successful arbitrage, the arbitrageur must often make a front-running transaction f immediately before the target transaction m, i.e., the transactions f and m are mined within the same block, but f is ordered before m. Techniques for achieving this goal have shifted over time.

Before 2020, arbitrage on the Ethereum blockchain happened mostly via priority gas auctions (PGAs), in which arbitrageurs would observe a victim transaction, then publicly broadcast front-running transactions with progressively higher gas prices [16]. The arbitrageur with the highest gas price would win. It benefits an arbitrageur to have a low triangular targeted latency (Definition 3.3) between target source nodes (e.g., the victim, other arbitrageurs) and a destination miner, or a low direct targeted latency (Definition 3.1) to a miner. This allows swifter reactions to competing bids and more rebidding opportunities before the block is finalized.

In 2020, mechanisms for arbitrage began to shift to private auction channels like Flashbots [10], in which an arbitrageur submits a miner-extractable-value (MEV) bundle with a tip for miners to a public relay, which privately forwards the bundle to miners. A typical bundle consists of a front-running transaction and the target transaction, where the order of transactions is decided by the creator and cannot be changed by the miner of the bundle. Among competing bundles (which conflict by including the same victim transaction), the one with the highest tip is mined.

The key difference from PGA is that an arbitrageur is no longer aware of the tips of its competitors; tips are chosen blindly to balance cost and chance of success. However, the tip is upper-bounded for a rational arbitrageur, because the arbitraging gain is determined by the victim transaction itself. Competing arbitrageurs may set up a grim trigger [4] on the percentage of the tip compared to the arbitrage profit [16]. For instance, they may agree that the tip must not exceed 80% of the profit, so that 20% of profit is guaranteed for the winner. Assuming that every arbitrageur is paying the miner with the same maximum tip, the competition collapses to one of speed. Therefore, even with MEV bundles, network latency remains a critical component of arbitrage success.

2.1 Related Work

Reducing P2P network latency has been a topic of significant interest in the blockchain community, including (1) decentralized protocol changes, and (2) centralized relay networks.

Decentralized Protocol Changes. Various decentralized protocols have been proposed to reduce the latency of propagating blocks and/or transactions. Several of these focus on reducing bandwidth usage, and reap latency benefits as a secondary effect. For example, Erlay [41] uses set reconciliation to reduce bandwidth costs of transaction propagation, and Shrec [26] further designs a new encoding

[4]Grim trigger is a trigger strategy in which a player begins by cooperating in the first period, and continues to cooperate until a single defection by her opponent, after which the player defects forever.

of transactions and a new relay protocol. These approaches are complementary to our results, which are not unique to the broadcast protocol or compressive transaction encoding scheme.

Another body of work has attempted to bypass the effects of latency and peer churn on the topology of the P2P network by creating highly-structured networks and/or propagation paths while maintaining an open network; examples include Overchain and KadCast [11, 44, 47]. Such highly-structured networks would be resilient to simple peering-choice strategies like Peri.

Centralized Relay Networks. There have been multiple efforts to build centrally-operated, geographically-distributed relay networks for blockchain transactions and blocks. These include bloXroute [3], Fibre [7], Bitcoin Relay Network [48], and the Falcon network [48]. In this work, we evaluate the latency reduction of centralized services, using bloXroute as a representative example that operates on the Ethereum blockchain. Nodes in the Ethereum P2P network connect to the bloXroute relay network via a gateway, which can either be local or hosted in the cloud [2]. BloXroute offers different tiers of service, which affect the resources available to a subscriber.

3 MODEL

We begin by modelling the P2P network $\mathcal{N}$ as a (possibly weighted) graph $(\mathcal{V}, \mathcal{E})$. Edge weights represent the physical distance traversed by a packet traveling between a pair of nodes (e.g., this can be approximated in practice with traceroute). In §6, we consider an unweighted, simplified model for analytical tractability. Let $d_{\mathcal{N}}(s, t)$ denote the shortest distance between s and t over the graph $\mathcal{N}$. Each individual node $v \in \mathcal{V}$ can create a message m and broadcast it to the network. The message m traverses all the nodes in $\mathcal{V}$ following the network protocol, which allows an arbitrary node to forward the message m upon receipt to a subset of its neighbors. We assume the source nodes of these messages follow a distribution $\mathbb{S}$ with support $\mathcal{V}$, i.e., $\mathbb{S}$ specifies a probability distribution over transaction emissions by nodes in $\mathcal{V}$.

A participant of the P2P network has two classes of identifiers:

Class 1. **(Network ID)** An ID assigned uniquely to each of its nodes. For example, each node in the Ethereum P2P network is assigned a unique enode ID including the IP address. For simplicity, we let $\mathcal{V}$ denote the ID space, and a node $v \in \mathcal{V}$ represents an ID instance.

Class 2. **(Logical ID)** An ID that is used to identify the creator or owner of a message (e.g., the public key of a wallet). We let $\mathcal{W}$ denote the ID space.

In blockchain networks, a participant may own multiple instances of both classes. In our analysis, we assume there exists a mapping $\texttt{NID} : \mathcal{W} \to \mathcal{V}$ from a logical ID to a network ID for simplicity.

3.1 Adversarial Model

We consider an adversary who aims to reduce latency in order to gain profit and/or threaten network security, as mentioned in §1. As a starting point, we assume the adversary is capable of inserting one agent node a into the network, which can maintain at most k peer connections at a time.[5] In addition to receiving and sending messages like a normal node, the node a can behave arbitrarily during relaying; for instance, it can block messages when the protocol requires it to broadcast them. The adversary observes the network traffic through a consisting of transactions, where each transaction has a signer, i.e., the logical ID of its sender. The adversary is assumed to not know the mapping NID, and cannot peer to a transaction sender $v = \texttt{NID}(w)$ with only knowledge of the logical ID w. With the network ID v, however, the adversary may establish a peer connection between a and the node.

[5]A motivated attacker can cheaply connect to many peers. However, in practice, we find empirically that maintaining too many peer connections is difficult due to issues including peer discovery and message processing, to name a few [14].

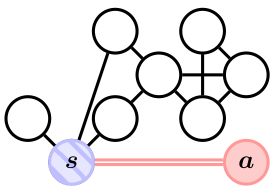
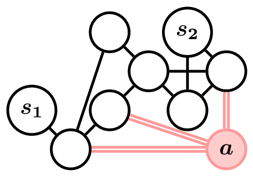
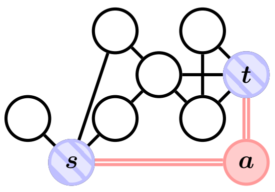
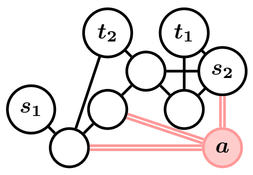
▨ Target node(s) ● Agent node(s) ═ Agent-initiated connections		
	Targeted	**Global**
Direct Latency		
Optimized by	Directly connecting to target (§3.2)	SS-ASPDM approximation algorithm [37]
Triangular Latency		
Optimized by	Directly connecting to targets (§3.3)	Unknown (§6.1)
Main Challenge	Don't know the IP addresses of targets (§7.2.1)	Don't know the graph topology, computational hardness of algorithms (§3.2, §6.1)

Table 1. Challenges and algorithms for optimizing direct and triangular latency, both targeted and global.

We consider multiple adversarial models with the above capabilities, but different goals. In particular, they aim to minimize different types of latencies, which we term direct and triangular latencies, with both global and targeted variants. We start by precisely defining these latencies.

In Table 1, we show the relationship between these metrics, and summarize what is currently known (and unknown) about how to optimize them. In this figure, solid red nodes represent a strategic agent. Blue striped nodes represent target nodes in targeted latency metrics. Red thick lines represent the edges (peer connections) that can be formed by the agent to attempt to optimize its network latency.

We let the random variable $\Lambda(a)$ denote the end-to-end traveling time of a message to a from a random source $S \sim \mathbb{S}$; the randomness is over the source and (possibly) the network relay protocol. Further, we let $\Lambda_v(a)$ denote the random variable $\Lambda(a)|S = v$, representing end-to-end traveling time of a message from a single source v to a. The randomness of $\Lambda_v(a)$ is over the network relay protocol. We measure the distributions of both random variables in §5.

3.2 Direct Latency

Direct targeted latency (Def. 3.1) is defined as the expected single-source message travel time Λ_v.[6]

Definition 3.1. In a P2P network $\mathcal{N}$, the **direct targeted latency** of node a with respect to a target node v is defined as:

$$L_v(a) \triangleq \mathbb{E}\left[\Lambda_v(a)\right].$$

[6]Expectation is only one metric of interest; practitioners may care about other metrics, e.g., quantiles. While we include such metrics in our experiments, we find that expectation captures the desired properties, while also facilitating basic analysis.

Reducing direct targeted latency can help with applications such as targeted attacks or front-running a specific set of victim nodes. We define direct global latency as follows.

Definition 3.2. In a P2P network $\mathcal{N}$, the **direct global latency** of node a is defined as:

$$L(a) \triangleq \mathbb{E}\left[\Lambda(a)\right].$$

Note that $L(a) = \mathbb{E}_{S\sim\mathbb{S}}[L_S(a)]$, which means the direct global latency equals the average direct targeted latency weighted by traffic source distribution $\mathbb{S}$.

3.2.1 Optimizing Direct Latency. We can consider the problem of latency reduction by an agent as an optimization problem. As a simplification to provide basic insights, we assume the network topology is fixed, and the end-to-end delay of an arbitrary message from source s to destination t is proportional to $d_{\mathcal{N}}(s,t)$, which denotes the graph distance between s and t on the network (i.e., number of hops, weighted by edge weights). We also assume the agent has a budget of peer links $k \geq 1$. Then, we aim to solve the following problem to optimize direct targeted latency:

$$\begin{aligned} \text{minimize} \quad & L_v(a) \\ \text{subject to} \quad & |\{y \mid (a,y) \in \mathcal{E}\}| \leq k. \end{aligned} \tag{1}$$

If we simplify the problem by assumption $d_{\mathcal{N}}(s,t) \leq d_{\mathcal{N}}(s,r) + d_{\mathcal{N}}(r,t)$ for all $r,s,t \in \mathcal{V}$, the solution to optimization (1) becomes trivial. The agent only needs to add the target v as a peer in order to achieve the minimum targeted latency, as stated in Table 1. However, achieving this is not necessarily trivial: an agent may not know v's network address (e.g., IP address), even if v's logical address is known. We discuss how to circumvent this challenge in §7.2 and what happens in practice when the simplifying assumption does not hold in §5.2.

The situation is very different with the optimization problem for global latency reduction, which we formulate as follows. The optimization problem is:

$$\begin{aligned} \text{minimize} \quad & L(a) = \sum_{v \in \mathcal{V}-\{a\}} d(v,a) \cdot \mathbb{P}_{S\sim\mathbb{S}}[S=v] \\ \text{subject to} \quad & |\{y \mid (a,y) \in \mathcal{E}\}| \leq k. \end{aligned}$$

This problem is called single-source average shortest path distance minimization (SS-ASPDM) [37]. It is NP-hard, but has an α-approximate algorithm [37].

3.3 Triangular Latency

Triangular latency is motivated by applications related to front-running in P2P networks. It is defined with respect to one or more pairs of target nodes, where a pair of target nodes includes a source node s and destination node t. For example, consider Table 1 (Triangular Targeted Latency), and let s represent the creator of a transaction and t a miner. The goal of agent node a is to establish a path $s \rightarrow a \rightarrow t$ with lower travel time than any path on $\mathcal{N}$ from s to t excluding a. It can do so by adding edges to the P2P network, which are shown in red in Table 1; note that the (shortest) path from s to a to t is four hops, whereas the shortest path from s to t excluding a is five hops. The existence of such a path enables a to front-run s's transactions that are mined by t.

We let L'_u denote the direct targeted latency with respect to u on the network excluding the agent a and its incident edges. As a front-runner, a aims to satisfy the following condition:

$$L'_s(t) > L_s(a) + L_t(a). \tag{2}$$

Here we assume the symmetry of targeted latency, i.e., $L_t(a) = L_a(t)$. The left-hand side $L'_s(t)$ is a constant, while the right-hand side depends on the neighbors of a over the P2P network $\mathcal{N} = (\mathcal{V}, \mathcal{E})$; we define this quantity as triangular targeted latency.

Definition 3.3. In a P2P network $\mathcal{N}$, the **triangular targeted latency** of node a with respect to a target pair (s,t) is defined as

$$L_{s,t}(a) \triangleq L_s(a) + L_t(a).$$

We again optimize this subject to a constraint on the number of agent peer connections:

$$\begin{aligned} \text{minimize} \quad & L_{s,t}(a) \\ \text{subject to} \quad & |\{y \mid (a,y) \in \mathcal{E}\}| \le k. \end{aligned} \tag{3}$$

The optimal strategy for solving (3) is again trivial: the agent should connect to both s and t (Table 1). Hence, to optimize for a single source and destination, it is key to find s and t on the network and to ensure (2) holds. We discuss this further in §7.

We next consider an agent that tries to manipulate the *global* triangular latency, capturing front-running opportunities over the entire network. We start with a simple (and unsatisfactory) definition of triangular global latency. Let Q denote the set of source-destination pairs of network traffic. A front-running agent might try to optimize the following:

$$L_Q(a) \triangleq \sum_{(s,t)\in Q} L_{s,t}(a) = \sum_{(s,t)\in Q} L_s(a) + L_t(a). \tag{4}$$

This is a variation of the SS-ASPDM problem discussed previously in §3.2. Note, however, that an agent with minimal aggregated pairwise triangular latency L_Q does *not necessarily gain maximum profit from front-running*. To profit, the agent needs to ensure the inequality (2) holds for pairs (s,t) not necessarily that right-hand side in (4) is minimized.

We thus define the following proxy metric for triangular global latency, which instead measures the quality of peering choices made by front-runners. Intuitively, the metric counts the number of node pairs (s,t) that an adversarial agent can shortcut, and the agent wants it to be as high as possible. For example, in Table 1 (Triangular Global Latency), the agent a is allowed to add three edges to the network, and its goal is to select those edges so as to shortcut the maximum number of node pairs (s,t) from network $\mathcal{N}$.

Definition 3.4. Let $a \notin \mathcal{V}$ denote an adversarial agent node, and $U \subseteq \mathcal{V}$ the set of a's peers. $\mathcal{N}' = (\mathcal{V}', \mathcal{E}')$ represents the network modified by the agent a, where $\mathcal{V}' = \mathcal{V} \cup \{a\}$ and $\mathcal{E}' = \mathcal{E} \cup \bigcup_{u\in U}(u,a)$. Let $\mathcal{S}, \mathcal{T} \subseteq \mathcal{V}$ be the sets of sources and destinations, respectively. The agent node a has a static distance penalty τ, which means that it can only successfully front-run a source-destination pair (s,t) if the path $s \longrightarrow a \longrightarrow t$ is at least τ units shorter than the shortest path $s \longrightarrow t$ on $\mathcal{N}$ that does not pass through a. The **adversarial advantage** is defined as

$$A_{\mathcal{N}}(U) = \sum_{s\in\mathcal{S}, t\in\mathcal{T}} \left(\mathbb{I}\left[d_{\mathcal{N}'}(s,t) + \tau < d_{\mathcal{N}}(s,t)\right] + \frac{1}{2} \cdot \mathbb{I}\left[d_{\mathcal{N}'}(s,t) + \tau = d_{\mathcal{N}}(s,t)\right] \right). \tag{5}$$

Note that we avoid defining the advantage with random end-to-end latencies such as $\Lambda_s(t)$ in order to keep the problem theoretically tractable. We also assign equal weights to each source-destination pair following the assumption that the adversary is unable to predict the profit to shortcut each particular pair of sources and destinations. In practice, $\mathcal{T}$ is the set of miners, and $\tau \in [0, \infty)$ is a parameter that depends on the computational capabilities of the agent, as well as randomness in the network. Ideally, we assume $\mathcal{N}$ is uniformly weighted. We also assume the weights of adversarial links $d_{\mathcal{N}'}(u,a) = 0$ for all $u \in U$, so that we have maximum flexibility in controlling the time costs over these links with parameter τ.

Summary. The proposed latency metrics express natural strategic agent goals. They are difficult to optimize directly, however, due to a combination of strong knowledge requirements about the P2P network, high computational complexity, and/or assumptions about the stationarity of the underlying network. For example, to optimize global latency, we require two key assumptions that are unrealistic for P2P networks: the network topology should be static and the agent should know both the network topology and traffic patterns. To optimize targeted latency, we require knowledge of the target node's network ID, i.e., its IP address; this is often unknown in practice. These challenges motivate the Peri algorithm in §4, which avoids all the above assumptions.

4 DESIGN

Optimizing network latency requires knowledge of network topology and/or node network addresses (§3), which are unknown for typical P2P networks. Instead, an agent is typically only aware of its own peers. In this section, we present the Peri peering algorithm to account for this and achieve *reduced* (if not necessarily optimal) latency. Peri is a variant of the Perigee [35] algorithm.

4.1 Perigee

Perigee was introduced in [35] as a decentralized algorithm for generating network topologies with reduced broadcast latency for transactions and blocks. The Perigee algorithm is presented in Alg. 1. Lines that are highlighted in red are specific to Peri (§4.2).

```
Input: Number of peers $K$ kept after each iteration, maximum peer count $N$, length of period $T$, score
       function $\phi$
Result: Peer set $P$ at each period $h = 1, 2, \cdots$
1  $P \leftarrow \emptyset$, $B \leftarrow \emptyset$ ;                          // $B$ is a blocklist of evicted peers
2  Run Thread peer_manager():
3      while true do
4          Hang and wait for next peer;
5          $v \leftarrow$ Random sampled node from $\mathcal{V} - B$;
6          if $|P| < N$ then
7              $P \leftarrow P \cup \{v\}$ ;                        // add peer when a slot is available
8  for $h \leftarrow 1, 2, \cdots$ do
9      Sleep $T$ ;                        // peer_manager adds peers to $P$ when sleeping
10     $e \leftarrow N - K$ ;                                 // $e$ is the number of peers to evict
11     Init score map $\Phi$;
12     for $p \in P$ do
13         if not is_excused($p$) then
14             $\Phi(p) \leftarrow \phi(p)$ ;
15         else
16             $e \leftarrow e - 1$;
17     if $e > 0$ then
18         $P \leftarrow P - \{e$ keys with largest values in $\Phi\}$;
19         $B \leftarrow B \cup \{e$ keys with largest values in $\Phi\}$;
20     Output $(h, P)$;
```

Algorithm 1: Perigee [35]/ Peri. Red text denotes parts that are specific to Peri. Note: is_excused is a predicate that is true when peer-delay information is insufficient to make a peering decision. $\phi(v)$ equals the average transaction-delivery delay of v with respect to the best peer, as defined in Eqn. (6). It incorporates design choices highlighted in §4.2.

At a high level, Perigee requires every node in the network to assign each of its peers a latency score and periodically tears down connections to peers with high scores. In our context, the latency score represents the latency with which transactions are delivered; we want this score to be as low as possible. Over time, Perigee causes nodes to remain connected to low-latency peers, while replacing other peers with random new ones. Roughly speaking, [35] shows that Perigee converges to a topology that is close to the optimal one, in the sense of minimizing global broadcast latency.

More precisely, Perigee divides time into periods. Let u denote a given node in the network and M_v denote the set of all transactions received by u in the current period from a current peer v. For a given transaction m, let $T(m)$ denote the time when m is first received by u from any of its peers and $T_v(m)$ denote the time when m is received by u specifically from v. We define $T_v(m) = \infty$ if v did not deliver m during the current period.

In every period, each node u evaluates a score function ϕ over each of its peers v, defined as

$$\phi(v) \triangleq \sum_{m \in M_v} \frac{1}{|M_v|} \min\left\{T_v(m) - T(m),\ \overline{\Delta}\right\}. \tag{6}$$

$\overline{\Delta}$ is a parameter used by Perigee as an upper bound on measured latency differences. It bounds the influence of outliers on score-function computation. For a node u, the score function $\phi(v)$ captures the average over all transactions m of the difference in latency between delivery of m by v and that by the peer from which u first received m. In other words, $\phi(v)$ may be viewed as the average slowdown imposed by v with respect to the fastest delivery of transactions to u.

The procedure `peer_manager` is an asynchronous thread (or set of threads) that handles peer connections on the P2P network, including accepting incoming peer requests, requesting nodes for connection and dropping peers. Ideally, it can randomly sample nodes from the entire network, gradually add peers when the peer count is under the maximum, and keep connections with specific nodes (targets). Hence, while the main thread sleeps, `peer_manager` expands the peer set P until it reaches its maximum size N.

4.2 Peri

Although Perigee was designed to be deployed by *all* network nodes to improve broadcast latency, we observe that the same ideas can be applied by a single agent to improve their observed direct and triangular latency. We next describe Peri, a slight modification of Perigee enabling an agent to control their direct and triangular latency. Although Peri does not functionally differ from Perigee, we give them different names to differentiate their usage and implementation choices. Again, the steps unique to Peri are highlighted in Red in Alg. 1. The main differences are:

(a) **Goal:** Peri is meant to be applied by a single node to advantage it over other nodes, whereas Perigee was proposed as a protocol to optimize systemic network performance.
(b) **Relevant transactions:** In Perigee, nodes measure the latency of all received transactions. In Peri, the score function $\phi(v)$ enforces that only *relevant transactions* participate in scoring peers. In Peri, for direct global latency reduction, all transactions are considered relevant, while for direct targeted latency reduction, only transactions made by the target are relevant.
(c) **Handling silent peers:** Particularly when optimizing targeted latency, the function ϕ may be undefined for some peers in some periods. For example, if a peer p is connected near the end of a given period, there will not be sufficient data to compare p with other peers, which means $\phi(p)$ may be undefined. Instead of evicting p in such cases and possibly losing a good peer, we forego eviction of p. In Alg. 1, the predicate `is_excused`(v) is true if node v should be excluded from eviction for the current period.
(d) **Blocklists.** The Perigee [35] algorithm advocates for selecting a new set of peers at random. However, this increases the likelihood of a peer tearing down connections, then connecting to

the same node(s) shortly thereafter, particularly since some cryptocurrency clients (including `geth`) favor previously-visited nodes during peer selection [27, 28]. To handle this, in Peri we use *blocklists*: if a node tears down a connection to peer v in a Peri period, it refuses to re-connect to v in future periods. In Alg. 1, we maintain blocklist B for this purpose.

(e) **Sampling relevant transactions:** We cannot relay relevant transactions; otherwise, the "late" peers who do not deliver a relevant transaction first to our node will *never* deliver it to our node (§2), thus removing them from Peri's latency comparison. If all transactions are relevant, our node will hence act as a black hole, and may impact the P2P network. We avoid this by sampling 1/4 of all relevant transactions for global latency measurement, by redefining relevant transactions as those with hashes divisible by 4 when computing $\phi(\cdot)$ in Line 14 of Alg. 1.

5 DIRECT LATENCY EVALUATION

In §5.1 and §5.2, we show the practical latency reduction effects of Peri with experiments on the Ethereum P2P main network (mainnet) and Rinkeby test network (testnet).

5.1 Evaluation: Direct Global Latency

5.1.1 Methods. We evaluate four approaches.

(a) **Baseline.** Our experimental control node uses the default settings of the Go-Ethereum client, version 1.10.16-unstable [9][7]. This was the latest version when we started the experiments.

(b) **BloXroute.** We compare against a centralized, private relay network, using bloXroute as a representative example. We use the bloXroute Professional Plan [3]; at the time our experiments were run, this plan cost \$300 per month. [8] We ran the bloXroute gateway locally to avoid incurring additional latency (§2.1).

(c) **Peri**. We modify the Go-Ethereum client [9] to implement the Peri algorithm for peer selection. We set the period to 20 minutes, and replace at most 25 peers every period.

(d) **Hybrid.** We implement a hybrid method that combines bloXroute and Peri by applying Peri to a node with access to a bloXroute relay. For correctness, we ensure that the gateway connecting to the relay, which acts as a peer of the node, cannot be removed by the Peri algorithm.

5.1.2 Experimental Setup. We establish 4 EC2 instances in the us-east-1 AWS data center, where a public bloXroute relay is located. On each instance, we deploy a full node on the Ethereum P2P main network (the mainnet), which is implemented by a customized Go-Ethereum (geth) client. Each node has at most 50 peers, which is the default setting of Go-Ethereum. For a node running Peri or Hybrid, we set the proportion of outbound peers (peers dialed by the node itself) to 80% so that the node actively searches for new peers. For a node running other baselines, we keep the proportion at 33%, which is also default for Go-Ethereum.

We ran 63 experiments from Feb 18, 2022 to March 16, 2022, each following the procedure below. First, we assign each latency reduction method (bloXroute, Peri, Hybrid and Baseline) exclusively to a single node, so that all four nodes use different comparison methods. Then, we launch the nodes simultaneously. When a packet arrives, the node checks if the packet contains a full relevant transaction or its hash; if so, it records the timestamp. At the end of the experiment, we stop the nodes and collect the arrival timestamps of transactions from their host instances.

Bias reduction. We take the following steps to control for systematic bias in our experiments. We prohibit the 4 measurement nodes from adding each other as peers. We reset all the enode IDs (unique identifiers of the Ethereum nodes including IP address) before each experiment. This

[7]The customized client source code can be found at https://github.com/WeizhaoT/geth_peri.

[8] bloXroute also offers more expensive and powerful plans, which we did not test due to financial constraints.

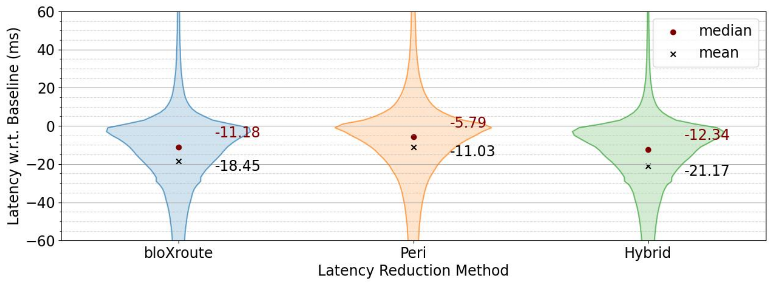


Fig. 1. PDFs of global latency distributions (difference in ms from Baseline latency).

prevents the other nodes from remembering our nodes' IP addresses and making peering decisions based on activity from previous experiments. We record the clock differences with periodic NTP queries between each pair of hosts to fix systematic errors in the timestamps recorded locally by the hosts. Despite being the same type of AWS instance, the host machines may also introduce biases because they may provide different runtime environments for the Ethereum node program. To eliminate these biases, we rotate the assignment of latency reduction methods after every experiment, so that for every successive 4 experiments, each node is assigned each method exactly once. We attempt to control for temporal biases due to diurnal transaction traffic patterns by running each experiment every 8 hours, so that the assignment of methods to nodes rotates 3 times a day. Because the number of possible method assignments is 4, a co-prime of 3, each node experiences every combination of latency reduction methods and time-of-day. We did not control for contention (e.g., of network bandwidth, computational resources) among EC2 instances by running experiments on dedicated hardware due to financial constraints.

5.1.3 Results. Each node is allowed to warm up for 2.5 hours; after this, we collect all transactions that are received by each of the nodes for 3.5 hours. Although it is infeasible to measure the transaction propagation time directly—as this would require sender timestamps—the time difference of arrival allows us to measure the ability of a method to *reduce* propagation time. For each transaction m and each node y with a latency reduction method other than Baseline, we compute the difference between the timestamp of m at y and the timestamp at the baseline node b, which is effectively a sample of random variable $\Lambda(y) - \Lambda(b)$. The smaller the time difference (i.e., the more negative), the earlier y delivers m, and the more effective the latency reduction method is. We gather the latency differences over 63 experiments, and plot their distributions for each node in Fig. 1. In total, we analyzed the latencies of 6,228,520 transactions.

Finding 1: ***Peri alone is at least half as effective as bloXroute private networks in reducing average direct global latency. The Hybrid node (Peri with bloXroute) achieves an additional 15% improvement in latency reduction over bloXroute alone.***

In the figure, all the latency differences are distributed with negative means and medians, showing an effective latency reduction. We find these results to be statistically significant; our statistical tests are detailed in App. A.4. BloXroute reduced this latency by 18.45 milliseconds on average. In comparison, Peri reduced it by 11.09 milliseconds. Although Peri's ability to reduce latency is expected, it is perhaps unexpected that Peri is more than half as effective as bloXroute (and cost-free). On the other hand, by exploiting access to bloXroute services, Hybrid nodes can achieve an *additional* 15% reduction in latency. This suggests the potential of peer-selection algorithms to further boost latency reduction techniques over private relay networks.

Note that these experiments naturally experienced network dynamics (e.g., traffic congestion) over months on the live Ethereum P2P network, suggesting that the results are robust to realistic conditions. On the other hand, it is difficult to empirically determine how network dynamics (cross traffic, BGP, etc.) exactly affect our results, because we cannot control or even measure them comprehensively in the network. As a compromise, we study the effects of changes in local network conditions, including the peer count and bandwidths of the attacker. Roughly, we found that the higher the adversary's peer count limit, the lower the advantage afforded by Peri; above a peer count of 100, the benefits of Peri appear to plateau, with median reductions between 2-5 ms (App. A.6). Similarly, we find that if the attacker has a low-bandwidth connection to the network, the benefits of Peri are significantly amplified. For example, if we throttle the adversary's bandwidth to 1.2 Mbps, the median latency reduction of Peri compared to the baselins is 37 ms—7× larger than the reduction in Fig. 1, which uses a 10 Gpbs link (App. A.7). Overall, these results suggest that **the benefits of strategic latency reduction are most significant for nodes with comparatively low network resources.**

5.2 Evaluation: Direct Targeted Latency

Experimentally evaluating targeted latency reduction techniques on the Ethereum mainnet can be costly and time-consuming. This is because to evaluate latency distributions, we need to observe many transactions with a known ground truth IP address. A natural approach is to generate our own transactions from a single node and measure their latency; unfortunately, nodes in the P2P network do not forward transactions that are invalid or unlikely to be mined due to low gas fees. Hence, we would need to create valid transactions. For example, at the time of writing this paper, the recommended base fee of a single transaction was about \$2.36 per transaction [4]. Collecting even a fraction of the 6+ million transactions analyzed in §5.1 would be prohibitively costly.

We ran limited experiments on direct targeted latency, where the setups and results are shown in details in Appendix A.5. The findings of these experiments are summarized below.

Finding 2: ***Peri reduces direct targeted latency by 14% of the end-to-end delay, which is as effective as bloXroute and Hybrid.***

We did not collect a sufficient amount of data for Peri and Hybrid to show a statistically significant ability to connect directly to a victim and learn its IP address. We attribute this to the low transaction frequency and large size of the mainnet. The testnet experiments we describe next, however, *did* result in frequent victim discovery of this kind—provided a sufficiently large number of Peri periods or a high frequency of victim transactions.

5.2.1 *Targeted Latency on Ethereum testnets.*

Due to the financial cost of latency reduction experiments on the Ethereum mainnet, we also performed experiments on the Rinkeby testnet. [9] Our goal in these experiments is to simulate a highly active victim in the network and compare the Peri algorithm's ability to find and connect to the victim against a baseline default client.

In these experiments we run a victim client on an EC2 instance in the ap-northeast-1 location and the Peri and Baseline clients in the us-east-2 location. Note that we cannot evaluate bloXroute or Hybrid on the test network, because bloXroute does not support test network traffic. Due to our observations of the Rinkeby network's smaller size, and to emulate a long-running victim whose peer slots are frequently full, we set the peer cap of the victim to 25 peers. As in previous

[9]Though the Ropsten testnet is generally thought to be the test network that most faithfully emulates the Ethereum mainnet [5], its behavior was too erratic for our experiments during our period of study. This included high variance in our ability to connect to peers from any machine and constant deep chain reorganizations, causing lags in client synchronization.

experiments, the Baseline node is running the default geth client with peer cap of 50. The Peri node is also running with a peer cap of 50 but with the Peri period shortened to 2 minutes.

Method	Successful Victim Connections
Peri	47/70 = 67.1%
Baseline	6/70 = 8.6%

Table 2. Rate of connection to victim node in Rinkeby testnet experiments.

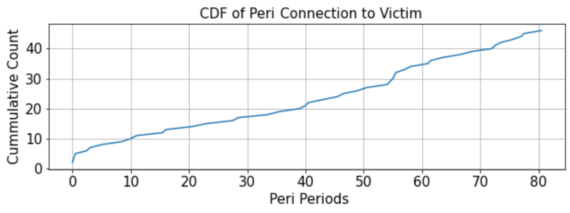


Fig. 2. CDF of number of Peri periods before finding victim. Each Peri period lasts 2 minutes.

In each experiment, we start the victim client first and give it a 30 minute warm-up period to ensure its peer slots are filled. We then start the Peri and Baseline nodes and begin transmitting transactions from multiple accounts on the victim node. We set the frequency to 3 transactions per minute to control the variance of timestamps and Peri scores. We then run the experiment for just under 3 hours, enabling us to run 7 experiments per day and alleviate time-of-day biases. After each experiment, we say the Peri node found the victim if the victim's Peri score is best (lowest) among the currently-connected peers. Since the baseline does not compute Peri scores, we say it found the victim if the baseline node ever connected to the victim. As with the global latency experiments in §5.1, we alternate which us-east-2 machine is running Peri v.s. Baseline across runs in order to mitigate potential machine-specific biases. We ran these experiments from April 16 - 26, 2022, for a total of 70 runs.

Finding 3: ***In testnet experiments, Peri is able to identify the IP address and connect to a target (victim) node with a frequency more than 7× that of a Baseline node.***

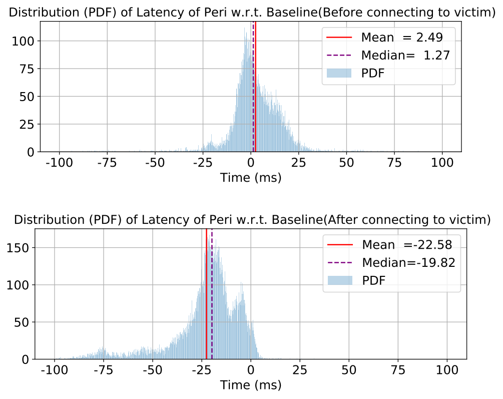


Fig. 3. PDFs of distributions of targeted latency for runs when Peri found the victim split between before either found the victim, and after Peri finds the victim. The x axis represents the difference of timestamp at the Peri node and at the baseline node for each single transaction.

The number of connections established by the Peri and Baseline clients to our victim are listed in Table 2. We find that Peri gives a notable advantage when connecting to the victim node. Fig. 2 shows a CDF of the number of Peri periods until Peri connected to the victim, with an mean/median of 39/45 periods (~ 1.3/1.5 hrs). Fig. 3 shows the delay PDF for the runs when Peri finds the victim for all transactions before and after Peri connects to the victim. We observe a mean latency advantage of 22ms over the Baseline client once Peri connects to the victim. This is 30% of the end-to-end delay between the victim and both the Peri and Baseline clients. Unlike our mainnet experiments, Peri has no significant latency advantage before it connects to the victim. We attribute the difficulty of establishing a network advantage prior to direct connection to the victim to the Rinkeby network's smaller size (1.8K unique IPs we came across over the period of our study compared to 13.6K in the mainnet). Hence, there may be relatively few peers in geographical proximity to the the Tokyo data center compared to our mainnet victim node in Germany. There are significant differences in size and topology between the Rinkeby network and the Ethereum mainnet, and it is unclear how our experiments extrapolate to the mainnet.

Though it is interesting to extend our results to different settings (different peer count, etc.), we encounter limitations for both the testnet and the mainnet. On the testnet, there are not enough active nodes to support a peer count of even 50, and the latency data is much noisier than on the mainnet. On the mainnet, the victim node from Chainlink is no longer available.

6 TRIANGULAR LATENCY EVALUATION

A front-runner may be interested in targeted and/or global triangular latency. As with targeted latency, the optimal strategy for an agent to reduce triangular targeted latency is to connect directly to the target nodes (§3.3). Since this procedure is identical to the experiments in §5.2, we do not run additional experiments to demonstrate its feasibility.

The more complicated question is how to optimize triangular global latency. Recall from §3.3 that we study triangular global latency via a proxy metric, which we call *adversarial advantage* $A_{\mathcal{N}}(U)$, where $\mathcal{N}$ denotes the network and U denotes the set of strategic or adversarial agents. In the remainder of the section, we first show that directly optimizing adversarial advantage is computationally infeasible even if the agent knows the entire network topology (§6.1). However, we also show through simulation that a variant of Peri can be used to outperform baselines (§6.2).[10]

6.1 Hardness of Optimizing Adversarial Advantage

Generally, we are interested in the following problem.

Problem 1. *Given a network $\mathcal{N}$, sets of sources and destinations $\mathcal{S}, \mathcal{T}$, and budget k (number of edges the agent can add), we want to maximize $A_{\mathcal{N}}(U)$ (Definition 3.4) subject to $|U| \leq k$, where U is the set of peers to which the agent node connects.*

Theorem 6.1. *Problem 1 is NP-hard.*

(Proof in Appendix A.3.1) The proof follows from a reduction from the set cover problem. Not only is solving this problem optimally NP-hard, we next show that it is not possible to *approximate* the optimal solution with a greedy algorithm. A natural greedy algorithm that solves the advantage maximization problem is presented in Algorithm 2. [11] It is *unable* to approximately solve Problem 1.

Proposition 6.2. *The output of Alg. 2 is* not *an α-approximate solution to Problem 1 for any $\alpha > 0$.*

(Proof in Appendix A.3.2) We show this by constructing a counterexample for which the greedy algorithm achieves an adversarial advantage whose suboptimality gap grows arbitrarily close to 1 as the problem scales up. Whether a polynomial-time approximation algorithm exists for maximization of $A_{\mathcal{N}}$ is an open problem.

[10] *Note on evaluation:* Adversarial advantage is more difficult to evaluate empirically than direct latency. For a single source-destination pair, we would need to observe transactions from a known source (e.g., a front-running victim), which end up at a known target destination (e.g., an auction platform [10]); we would also need to verify that our agent node can reach the target node *before* the victim's transaction reaches the target destination. Setting up mainnet nodes to measure this would be twice as costly as evaluating direct targeted latency, which we deemed infeasible in §5.2. Measuring adversarial advantage globally would further require visibility into every pair of nodes in the network. We therefore evaluate triangular latency reduction theoretically and in simulation.

[11] It includes a "bootstrapping" step in which two nodes are initially added to the set U of agent peers, as no advantage is obtainable without at least two such nodes.

```
Input: Number of peers $k \geq 2$, graph $\mathcal{N}$
Output: Set of peers $U$
1 $U \leftarrow \text{argmax}_{\{x,y\}:(x,y)\in\mathcal{V}^2} A_{\mathcal{N}}(\{x,y\})$ ;          // bootstrapping
2 for $j \in \{3, 4, \cdots, k\}$ do
3     $\bar{z} \leftarrow \text{argmax}_{z:z\in\mathcal{V}-U} A_{\mathcal{N}}(U \cup \{z\})$;
4     $U \leftarrow U \cup \{\bar{z}\}$;
5 return $U$;
```

Algorithm 2: Greedy Algorithm for Maximizing $A_{\mathcal{N}}$.

6.2 Approximations of Advantage Maximization

Although the greedy algorithm (Alg. 2) is provably suboptimal for maximizing adversarial advantage, we observe in simulation that for small, random network topologies (up to 20 nodes), it attains a near-optimal adversarial advantage (results omitted for space). We also show in this section that the greedy strategy achieves a much higher advantage (2-4×) than a natural baseline approach of randomly adding edges. Hence, the greedy strategy may perform well in practice. However, the greedy strategy *requires knowledge of the entire network topology.*

We next evaluate the feasibility of local methods (i.e., Peri) for approaching the greedy strategy.[12]

Graph topology. We consider four models of random graph topologies: Erdös-Rényi, random regular graphs, Barabási-Albert (scale free) graphs, and Watts–Strogatz (small world) graphs—each with 300 nodes. In addition, we study a snapshot of the Bitcoin P2P network [38] and the Lightning Network (LN) [17] (sampling details in App. A.8). The Ethereum P2P network has been shown to exhibit properties of both small world and scale-free networks [49]. The average degree is set to 9 for the Erdös-Rényi model to ensure connectivity. The average degree is set to 4 for the other 3 models to make sure the average distance between nodes is high enough for shortcuts to exist. For each model, we sample 25 different graph instances. To model the observed existence of hubs in cryptocurrency P2P networks [19, 21, 23, 38], we introduce 20 new hub nodes, each connecting to 30 nodes randomly sampled from the original graph.

On each instance, the set of sources $\mathcal{S}$ is equal to the set of all the nodes $\mathcal{V}$, and the set of destinations $\mathcal{T}$ is a random subset of $\mathcal{V}$ with $|\mathcal{T}| = 0.1 \cdot |\mathcal{V}|$. Note that $\mathcal{T} \subset \mathcal{S}$. We let the static front-running penalty $\tau = 0$, a minimum path advantage, and assign a unit weight to all the edges.

Peri Modification. To make Peri optimize adversarial advantage, we alter the scoring function and the peer-selection criterion; this allows us to simulate the (modified) Peri algorithm without simulating individual messages. First, we make the following simplifying assumptions: (a) Peer replacement is completed immediately at the beginning of a new Peri period, and no peer is added or dropped during the period; (b) The full set M of messages sent during the current Peri period is delivered to the adversarial agent during the Peri period by every peer; (c) The end-to-end delay of each message is proportional to the distance between its source and destination; (d) Message sources are uniformly distributed over the network and message destinations send messages at the same frequency; and (e) The agent can tell whether the source of any message belongs to set $\mathcal{T}$.

Let $d_v(m)$ denote the distance from $s(m)$, the source of m, to peer v of the agent node a, and $S(m)$ the time when message m departs from the source. As in (6), we define $T_v(m)$ as the time when m is received from peer v, and $T(m)$ as the time when m is first received from the fastest peer. Recall that we assume $d_{\mathcal{N}'}(v, a) = 0$ for any peer v of a in §3.3, so for such a v, $d_v(m)$ equals the distance from the source of m to a. We assume the agent assigns a weight in the score function

[12]The simulator code can be found at https://github.com/WeizhaoT/Triangular-Latency-Simulator.

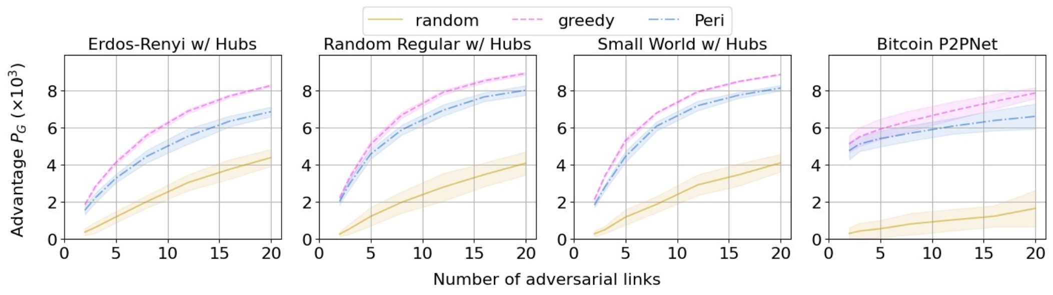


Fig. 4. Advantage-Peer-count curves on hub-enriched graph models. The mean of the curves are shown by solid lines, and the standard deviations are shown by the transparent color zones centered at the mean.

$\lambda_{s(m)}$ by the source $s(m)$. The score function $\phi(v)$ for peer v can be transformed with constants C_1, C_2, C_3 with respect to v for a set topology.

$$\phi(v) = \sum_{m\in M} \frac{T_v(m) - S(m)}{|M|\lambda_{s(m)}^{-1}} + \frac{S(m) - T(m)}{|M|\lambda_{s(m)}^{-1}} = C_1 \sum_{m\in M} \lambda_{s(m)} d_v(m) + C_2 = C_3 \sum_{s\in\mathcal{S}} \lambda_s d_{\mathcal{N}}(s, v) + C_2.$$

Since Peri's choice of peers to drop is invariant to C_2 or C_3, we can further simplify the score: $\phi(v) = \sum_{s\in\mathcal{S}} \lambda_s d_{\mathcal{N}}(s, v)$. Recall that $\mathcal{T} \subset \mathcal{S}$, which implies that by properly controlling λ_s, we can let the agent pay equal attention to approaching sources and destinations. Eventually,

$$\phi(v) = \frac{1}{|\mathcal{S}|} \sum_{s\in\mathcal{S}} d_{\mathcal{N}}(s, v) + \frac{1}{|\mathcal{T}|} \sum_{t\in\mathcal{T}} d_{\mathcal{N}}(t, v). \tag{7}$$

For a given peer budget k, we replace $r = \left\lceil \frac{k}{3} \right\rceil$ peers and keep k peers. We only consider the kept peers for evaluation of the advantage metric. In total we execute 800 Peri periods.

Results. We sweep the peer count budget $|U| = k$ of the agent over 7 values ranging from 2 to 20. For each maximum peer count, we obtain a resulting peer set U^* with advantage $A_{\mathcal{N}}(U^*)$ using the greedy algorithm, the Peri algorithm, and random sampling, in which each agent chooses to peer with nodes selected uniformly at random. The performance of each method is represented by its advantage-peer-count ($A_{\mathcal{N}}$-$|U|$) curve. For each graph model, we plot the mean and standard deviation of the curves over 25 different hub-enriched graph instances in Fig. 4.

Finding 4: ***Peri is competitive with the greedy algorithm for maximizing adversarial advantage when the underlying network has many hubs, or nodes of high degree.***[13]

Regardless of original topology, the existence of hub nodes enables the Peri algorithm to place shortcuts almost as effectively as the greedy algorithm. We notice that Peri worked significantly better on models with more hubs such as scale-free than the other topologies. It is also notable that over 70% of the resulting peers of the Peri algorithm are hubs. Our results on Bitcoin and the Lightning Network (App. A.8, A.9), however, show, that in practice, the gap between Peri and the Greedy baseline grows with the number of adversarial links. Peri appears to be too aggressive in selecting hubs, which are often clustered in real-world networks; hence connecting to multiple hubs gives less advantage. Nontheless, Peri still outperforms the random baseline by 3× - 5×.

[13]Precisely, we define a hub as a node whose degree is at least 10% the total number of nodes in the graph.

7 IMPOSSIBILITY OF STRATEGY-PROOF PEERING PROTOCOLS

Peri and its variants, e.g., Hybrid, can significantly improve direct and triangular latency. A natural question is whether this is because most nodes are currently running a poor peer selection strategy (Baseline). If *every* node were running Peri or Hybrid, would it still be possible for an agent to strategically improve their targeted latency? We show that no matter what peering strategy nodes use, if the P2P network has some natural churn and target node(s) are active, a strategic agent can *always* manipulate their targeted latency.

7.1 Model

We start with some additional modeling assumptions regarding the temporal dynamics of our network. Consider a time-varying network $\mathcal{N}(t) = (\mathcal{V}(t), \mathcal{E}(t))$ where $\mathcal{V}(t)$ denotes the set of nodes and $\mathcal{E}(t)$ denotes the set of undirected edges of G at time t. In the network, any party can spawn a set V' of nodes and add them to $\mathcal{V}(t)$. ($\mathcal{V}(t^+) = \mathcal{V}(t) \cup V'$, where t^+ denotes the time infinitesimally after t). We assume there is an upper bound $\bar{V}$ on the total number of nodes in the network; that is, $\mathcal{V}(t) \leq \bar{V}$ for all $t \geq 0$, due to the limited address space for nodes.

If a node u knows the network (Class 1) ID of another node v, u can attempt to add an edge, or a peer connection, (u, v) at time t (that is, $\mathcal{E}(t^+) = \mathcal{E}(t) \cup \{(u, v)\}$) to the network. This peering attempt will succeed unless all of v's peer connections are full. Nodes have an upper bound of h on their total number of peer connections: $\deg(u) \leq h \; \forall u \in \mathcal{V}(t), \; \forall t \geq 0$. Among these, each node has at least $F \in (0, h]$ **dynamic peer slots**. A connection from u to v that occupies one of either u's or v's dynamic peer slots is called *dynamic* and will be periodically torn down (see Definition 7.1). Additionally, a dynamic peer slot is permissionless and is filled first-come-first-serve (FCFS).

In practice, permissionless network nodes find each other via queries to distributed databases of network IDs in a process called peer discovery [50]. We model such peering databases as an oracle that responds to peer queries from any node. In response to a query, the oracle independently draws the network ID of a node in the network from some (unknown) probability distribution, where each node may be drawn with a non-zero probability. Specifically, there exists a universal constant $q > 0$ where the probability of drawing an arbitrary node is lower-bounded by q. This is based on the assumption that the number of nodes is upper bounded. We assume the oracle can process a fixed number of queries per unit time, so each query (across all nodes) takes constant time. Upon processing a query, the oracle responds with the network ID of an existing node.

Each edge $(u, v) \in \mathcal{E}(t)$ has an associated *link distance* $w(u, v)$, which operationally represents the end-to-end latency from u to v. We assume edge latency $w(u, v)$ is affected by the physical length of the path $u \longrightarrow v$ on the Internet (i.e., including routing) and processing delays at the sender or recipient. Hence, we assume $w(u, v)$ is a constant over time and satisfies the triangle inequality:

$$w(u, v) \leq w(u, z) + w(z, v), \quad \forall u, v, z \in \mathcal{V}(t), t \geq 0. \tag{8}$$

This is distinct from graph distance $d_{\mathcal{N}}(u, v)$ introduced in §3. The triangle inequality often does not hold over the Internet [33]. However, we conjecture that triangle inequality violations may be less common in cryptocurrency P2P networks, since end-to-end latency is significantly impacted by processing delays at router nodes, which scales with the number of hops in a route.

Transaction dissemination. A node can broadcast an arbitrary message (transaction) at an arbitrary time through the entire network. When a message is sent from u to its neighbor v at time t, v will receive the message at time $t + w(u, v)$, where $w(u, v)$ is the link distance between u and v. If v is not adversarial, it will immediately forward the message to each of its neighbors.[14] We assume the P2P network is connected at all times $t \geq 0$. We additionally assume that the latency between any

[14]This is a special case of randomized flooding protocols like diffusion [24].

pair of agent nodes is negligible, such that an agent connecting to multiple targets from multiple agent nodes, is equivalent to these targets peering with one node.

Liveness. In our analysis, we will assume that nodes are sufficiently active in terms of producing transactions and that the network experiences some amount of peer churn. We make both of these assumptions precise with the following definition.

Definition 7.1. A (λ, ν)-live network has the following properties:

(I) All dynamic connections have a random duration $\text{Exp}(\lambda)$.

(II) For each node $u \in \mathcal{S}$, where $\mathcal{S}$ denotes a set of target nodes, we let $\{M_u(t),\ t \in [0, \infty)\}$ denote the counting process of messages that are generated and broadcast by u. $M_u(t)$ is a Poisson process with rate ν.

7.2 Optimization of Targeted Latencies

7.2.1 Inferring Network ID. We argue in §3 that we can optimize targeted latency (both direct and triangular) by connecting directly to the target node(s). However, in practice, an agent typically only knows the targets' logical (Class 2) IDs (e.g., public key of wallet), whereas it needs their network (Class 1) IDs (e.g., IP address) to connect. In the following, we show when an adversarial agent can learn the network ID(s) of one or more targets. Our first result states that in a live network (Def. 7.1), an agent can uncover the network ID(s) of a set of target nodes with probability $1 - \varepsilon$ given only their logical IDs in time quasilinear in ε^{-1} and quadratic in the number of target nodes.

Theorem 7.2. *Consider a live network $\mathcal{N}(t) = (\mathcal{V}(t), \mathcal{E}(t))$. Let $\mathcal{A}$ denote an adversarial agent capable of identifying the logical ID of the sender of any message it received from the network. Given a set of target nodes U, with probability at least $1 - \varepsilon$ for any $\varepsilon \in (0, 1)$, it takes the agent $O(|U|^2 \varepsilon^{-1} \log^2 \varepsilon^{-1})$ time to find the network ID of any message sender in $\mathcal{N}(t)$.*

(Proof in Appendix A.3.3) The proof shows that a variant of Peri achieves the upper bound. In particular, to optimize triangular targeted latency, for a target pair of nodes (s, t), the agent must connect to both s and t, so $|U| = 2$ in Theorem 7.2. Additionally, the following proposition shows that regardless of the agent's algorithm, we require time at least logarithmic in $1/\varepsilon$.

Proposition 7.3 (Lower Bound). *Consider a live network $\mathcal{N}(t) = (\mathcal{V}(t), \mathcal{E}(t))$. Let $\mathcal{A}$ denote an adversary defined in Theorem 7.2. For any $\varepsilon \in (0, 1)$, to achieve a probability at least $1 - \varepsilon$ that the agent is connected to its target, $\mathcal{A}$ must spend at least $\Omega(\log \varepsilon^{-1})$ time, regardless of algorithm.*

(Proof in Appendix A.3.4) Proposition 7.3 suggests that a node aiming to prevent agents from learning its network ID with probability at least $1 - \varepsilon$, should cycle its logical ID on a timescale of order $\log \varepsilon^{-1}$. Together, Theorem 7.2 and Proposition 7.3 show how and when an agent can find a message source in a P2P network. They are an important missing piece in the solution to optimization problems (1) and (3) for targeted latency. Next, we discuss how the agent ensures successful and persistent peer connections with targets after finding them.

7.2.2 Connecting to Targets. By assumption, the target must have at least $f > 0$ dynamic peer slots, which are permissionless and thus can be accessed by the agent. Since these slots are on a FCFS basis, the agent may request an occupied peer slot at an arbitrarily high frequency, such that it immediately gets it when the slot becomes available after the old connection is torn down. [15] In this way, an agent can effectively make a dynamic peer slot of any target *no longer dynamic*, and constantly dedicated to itself.

[15]To evade detection due to the frequency of requests, an agent can spawn many nodes and split the requests among them.

7.2.3 *Latency Manipulation.*

Even if an agent peers with a target, its latency is lower-bounded due to the physical distance between the agent and the target. Agents may be able to relocate their nodes geographically to manipulate triangular latency. Currently, over 25% of Ethereum nodes are running on AWS [1]. An agent can deploy nodes on co-located cloud servers to reduce the direct targeted latency to the level of microseconds. For triangular targeted latency where the source and the destination are at different geolocations, the agent can deploy node a_1 near the source s and node a_2 near the destination t, where a_1 connects to s and a_2 connects to t, with a_1 and a_2 connected via a low-latency link. This increases the odds of path $s \longrightarrow a_1 \longrightarrow a_2 \longrightarrow t$ being shortest, which increases the chance of success in front-running messages between s and t.

8 CONCLUSION

Motivated by the increasing importance of latency in blockchain systems, we have studied the empirical performance of strategic latency reduction methods as well as their theoretical limits. We formally defined the notions of direct and triangular latency, proposed a strategic scheme Peri for reducing such latencies, and demonstrated its effectiveness experimentally on the Ethereum network. Our results suggest it is not possible to ensure strategy-proof peering protocols in unstructured blockchain P2P networks. An open question is how to co-design consensus protocols that account for potential abuse at the network layer.

ACKNOWLEDGEMENTS

We wish to thank Lorenz Breidenbach from Chainlink Labs and Peter van Mourik from Chainlayer for their gracious help in the setup for our direct-targeted-latency experiments.

G. Fanti and W. Tang acknowledge the Air Force Office of Scientific Research under award number FA9550-21-1-0090, as well as the generous support of Chainlink Labs and IC3. L. Kiffer contributed to this project while under a Facebook Fellowship.

REFERENCES

[1] [Accessed Apr. 2022]. Blockchain on AWS. https://aws.amazon.com/blockchain/. ([Accessed Apr. 2022]).
[2] [Accessed Apr. 2022]. bloXroute Documentation. ([Accessed Apr. 2022]). https://docs.bloxroute.com/.
[3] [Accessed Apr. 2022]. BloXroute website. https://bloxroute.com/. ([Accessed Apr. 2022]).
[4] [Accessed Apr. 2022]. Eth Gas Station. https://ethgasstation.info/. ([Accessed Apr. 2022]).
[5] [Accessed Apr. 2022]. Ethereum Networks Documentation. ([Accessed Apr. 2022]). https://ethereum.org/en/developers/docs/networks/.
[6] [Accessed Apr. 2022]. Ethereum Wire Protocol (ETH). https://github.com/ethereum/devp2p/blob/master/caps/eth.md. ([Accessed Apr. 2022]).
[7] [Accessed Apr. 2022]. Fast Internet Bitcoin Relay Engine (FIBRE) . https://bitcoinfibre.org/. ([Accessed Apr. 2022]).
[8] [Accessed Apr. 2022]. Gas and Fees. https://ethereum.org/en/developers/docs/gas/. ([Accessed Apr. 2022]). Accessed on April 29, 2022.
[9] [Accessed Apr. 2022]. Go Ethereum. https://geth.ethereum.org/. ([Accessed Apr. 2022]).
[10] [Accessed Apr. 2022]. MEV-Explore v1. https://explore.flashbots.net/. ([Accessed Apr. 2022]).
[11] Vijeth Aradhya, Seth Gilbert, and Aquinas Hobor. 2022. OverChain: Building a robust overlay with a blockchain. *arXiv preprint arXiv:2201.12809* (2022).
[12] 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.* 585–602.
[13] C.E. Bonferroni. 1936. *Teoria statistica delle classi e calcolo delle probabilità.* Seeber. https://books.google.com/books?id=3CY-HQAACAAJ
[14] Lighthouse Book. [Accessed Jan. 2023]. Advanced Networking. https://lighthouse-book.sigmaprime.io/advanced_networking.html. ([Accessed Jan. 2023]). Accessed on January 30, 2023.
[15] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. 2016. On scaling decentralized blockchains. In *International conference on financial cryptography and data security.* Springer, 106–125.

[16] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 910–927.

[17] Christian Decker. [Accessed Jan. 2023]. Lightning Network Research; Topology Datasets. https://github.com/lnresearch/topology. ([Accessed Jan. 2023]). https://doi.org/10.5281/zenodo.4088530 Accessed on January 10, 2023.

[18] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*. IEEE, 1–10.

[19] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. 2019. Txprobe: Discovering bitcoin's network topology using orphan transactions. In *International Conference on Financial Cryptography and Data Security*. Springer, 550–566.

[20] Sergi Delgado-Segura, Cristina Pérez-Solà, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joan Borrell. 2018. Cryptocurrency networks: A new P2P paradigm. *Mobile Information Systems* 2018 (2018).

[21] Varun Deshpande, Hakim Badis, and Laurent George. 2018. BTCmap: mapping bitcoin peer-to-peer network topology. In *2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*. IEEE, 1–6.

[22] Olive Jean Dunn. 1961. Multiple Comparisons among Means. *J. Amer. Statist. Assoc.* 56, 293 (1961), 52–64. https://doi.org/10.1080/01621459.1961.10482090 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1961.10482090

[23] Jean-Philippe Eisenbarth, Thibault Cholez, and Olivier Perrin. 2021. An open measurement dataset on the Bitcoin P2P Network. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 643–647.

[24] Giulia Fanti and Pramod Viswanath. 2017. Deanonymization in the bitcoin P2P network. *Advances in Neural Information Processing Systems* 30 (2017).

[25] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 281–310.

[26] 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 *Proceedings of the 11th ACM Symposium on Cloud Computing*. 238–252.

[27] 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)*. 129–144.

[28] Sebastian Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. 2019. Eclipsing ethereum peers with false friends. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 300–309.

[29] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

[30] William H. Kruskal and W. Allen Wallis. 1952. Use of Ranks in One-Criterion Variance Analysis. *J. Amer. Statist. Assoc.* 47, 260 (1952), 583–621. http://www.jstor.org/stable/2280779

[31] Chenxin Li, Peilun Li, Dong Zhou, Zhe Yang, Ming Wu, Guang Yang, Wei Xu, Fan Long, and Andrew Chi-Chih Yao. 2020. A decentralized blockchain with high throughput and fast confirmation. In *2020 {USENIX} Annual Technical Conference ({USENIX} {ATC} 20)*. 515–528.

[32] Jinyang Li. 2005. *Routing tradeoffs in dynamic peer-to-peer networks.* Ph.D. Dissertation. Massachusetts Institute of Technology.

[33] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. 2009. Triangle inequality variations in the internet. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. 177–183.

[34] Igor Makarov and Antoinette Schoar. 2020. Trading and arbitrage in cryptocurrency markets. *Journal of Financial Economics* 135, 2 (2020), 293–319.

[35] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. 2020. Perigee: Efficient peer-to-peer network design for blockchains. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*. 428–437.

[36] Petar Maymounkov and David Mazieres. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*. Springer, 53–65.

[37] Adam Meyerson and Brian Tagiku. 2009. Minimizing average shortest path distances via shortcut edge addition. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 272–285.

[38] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2015. Discovering bitcoin's public topology and influential nodes. *et al* (2015).

[39] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2015. Discovering bitcoin's public topology and influential nodes. *et al* (2015).

[40] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. 2019. A survey of blockchain from the perspectives of applications, challenges, and opportunities. *IEEE Access* 7 (2019), 117134–117151.

[41] 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. 817–831.
[42] A. Osipovich. 1 Apr. 2021. High-Frequency Traders Eye Satellites for Ultimate Speed Boost. *Wall Street Journal* (1 Apr. 2021).
[43] A. Osipovich. 15 Dec. 2020. High-Frequency Traders Push Closer to Light Speed With Cutting-Edge Cables. *Wall Street Journal* (15 Dec. 2020).
[44] Elias Rohrer and Florian Tschorsch. 2019. Kadcast: A structured approach to broadcast in blockchain networks. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 199–213.
[45] Tim Roughgarden. 2020. Transaction fee mechanism design for the Ethereum blockchain: An economic analysis of EIP-1559. *arXiv preprint arXiv:2012.00854* (2020).
[46] Avron Shane. 2021. Advantages and Disadvantages of a Peer-to-Peer Network. *Flevy Blog* (2021).
[47] Bhavesh Toshniwal and Kotaro Kataoka. 2021. Comparative Performance Analysis of Underlying Network Topologies for Blockchain. In *2021 International Conference on Information Networking (ICOIN)*. IEEE, 367–372.
[48] Aaron Van Wirdum. 2016. HOW FALCON, FIBRE AND THE FAST RELAY NETWORK SPEED UP BITCOIN BLOCK PROPAGATION (PART 2). *Bitcoin Magazine* (2016).
[49] Taotao Wang, Chonghe Zhao, Qing Yang, Shengli Zhang, and Soung Chang Liew. 2021. Ethna: Analyzing the Underlying Peer-to-Peer Network of Ethereum Blockchain. *IEEE Transactions on Network Science and Engineering* 8, 3 (2021), 2131–2146.
[50] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. 2019. A survey on consensus mechanisms and mining strategy management in blockchain networks. *Ieee Access* 7 (2019), 22328–22370.
[51] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. 2020. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 90–105.

A APPENDIX

A.1 Table of Notations

We list the table of notations in Table 3.

Notation	Description
$\mathcal{N}$	Network $\mathcal{N} = (\mathcal{V}, \mathcal{E})$
$\mathcal{V}$	Set of nodes in the network
$\mathcal{E}$	Set of edges in the network
$\mathcal{S}$	Set of source nodes in the network
$\mathcal{T}$	Set of destination nodes in the network
$\mathcal{W}$	The space of logical node IDs (e.g., pubkey of a wallet in Ethereum)
$d_{\mathcal{N}}(a, b)$	Distance between a pair of nodes a and b over network $\mathcal{N}$
$w(a, b)$	Link distance between a and b (dominated by their physical distance)
$\mathbb{S}$	Distribution of network traffic sources with support $\mathcal{V}$
$\Lambda(a)$	End-to-end traveling time of a message to a from a random source $S \sim \mathbb{S}$
$\Lambda_v(a)$	End-to-end traveling time of a message to a from a fixed source v
$L(a)$	Direct global latency of agent node a over network $\mathcal{N}$
$L_v(a)$	Direct targeted latency of agent node a w.r.t. target v over network $\mathcal{N}$
$L_{s,t}(a)$	Triangular targeted latency of agent node a w.r.t. target pair (s, t) over network $\mathcal{N}$
$L_Q(a)$	Triangular global latency of agent node a w.r.t. set of target pairs Q
$A_{\mathcal{N}}(U)$	Adversarial advantage when the agent chooses set of peers $U \subseteq \mathcal{V}$
$\phi(v)$	Score of peer v of agent a during a Peri period
k	Budget of number of peers available to the agent node

Table 3. Table of Notations.

A.2 An Extension of Theorem 7.2

We present Lemma A.1 which shows a necessary condition that a Poisson process always satisfies. It is proved in §A.3.5. Note that the Poisson process is not the only type of counting process that satisfies (9). For instance, a process with constant inter-arrival time also satisfies (9).

Lemma A.1. *A Poisson process* $\{M_u(t),\ t \geq 0\}$ *with rate* v *satisfies that there exist constants* $\gamma, \mu > 0$, *such that*

$$\mathbb{P}\left[M_u(t+\Delta) - M_u(t) < \gamma\Delta\right] < \frac{\mu}{\Delta}, \quad \forall t, \Delta > 0. \tag{9}$$

Then, we relax the notion of live network to that of *quasi-live* network by relaxing the Poisson process assumption.

Definition A.2. A (λ, v)-*quasi-live* network has the following properties:

(I) All dynamic connections have a random duration $\mathrm{Exp}(\lambda)$.

(II) For each node $u \in \mathcal{S}$, where $\mathcal{S}$ denotes a set of target nodes, we let $\{M_u(t),\ t \in [0, \infty)\}$ denote the counting process of messages that are generated and broadcast by u. $M_u(t)$ is a counting process, where there exist constants $\gamma, \mu > 0$, such that

$$\mathbb{P}\left[M_u(t+\Delta) - M_u(t) < \gamma\Delta\right] < \frac{\mu}{\Delta}, \quad \forall t, \Delta > 0.$$

Finally, we extend Thm. 7.2 to Thm. A.3, relaxing the Poisson process condition to its necessary condition above. The proof of Thm. A.3 is in

Theorem A.3. *Consider a* quasi-live *network* $\mathcal{N}(t) = (\mathcal{V}(t), \mathcal{E}(t))$. *Let* $\mathcal{A}$ *denote an adversarial agent capable of identifying the logical ID of the sender of any message it received from the network. Given a set of target nodes* U, *with probability at least* $1 - \varepsilon$ *for any* $\varepsilon \in (0, 1)$, *it takes the agent* $O(|U|^2 \varepsilon^{-1} \log^2 \varepsilon^{-1})$ *time to find the network ID of any message sender in* $\mathcal{N}(t)$.

A.3 Proofs

A.3.1 Proof of Theorem 6.1. It is known that the set cover problem is NP-complete [29]. We take an arbitrary instance of the set cover problem:

Problem 2. *Given a finite set of elements* $\Sigma = \{\sigma_1, \cdots, \sigma_p\}$ *and its subsets* $\Gamma_1, \cdots, \Gamma_q$. *We aim to find the fewest collection of subsets from* $\Gamma_{1:q}$, *whose union equals* Σ.

We construct a graph $G = (\mathcal{V}, \mathcal{E})$ as below. Each element σ_i in Σ corresponds to a unique node of the same name. Each subset Γ_j corresponds to 2 nodes γ_j^+, γ_j^-. Finally, a fresh node c is added. In other words, $\mathcal{V} = \{c\} \cup \bigcup_{i=1}^{p} \{\sigma_i\} \cup \bigcup_{j=1}^{q} \{\gamma_j^+, \gamma_j^-\}$.

For each pair $(i, j) \in [p] \times [q]$, edge $(\sigma_i, \gamma_j^-) \in \mathcal{E}$ if and only if $\sigma_i \in \Gamma_j$. Besides these, $\mathcal{E}$ contains (γ_j^-, γ_j^+) and (γ_j^+, c) for each j. Fig. 5 illustrates an example of topology of G.

We take set of sources $\mathcal{S} \triangleq \{\sigma_1, \sigma_2, \cdots, \sigma_p\}$ and set of destinations $\mathcal{T} \triangleq \{c\}$. As expected, the original distance $d_G(s, c) = 3$ for each $s \in \mathcal{S}$. Now we claim that if we can solve Problem 1 in polynomial time for $G, \mathcal{S}, \mathcal{T}$ where $\forall e \in \mathcal{E}\ \big(w(e) = 1\big)$, $\tau = 1.99$ (breaking ties) and $k \in [2, 1+q]$, then we can also solve the original set cover problem in polynomial time.

When $k \in [2, 1+q]$, we are able to select k spy nodes amongst nodes in $\mathcal{V}$. The optimal choice must

- contain c: Otherwise, there must exist a spy node in Γ^+, or the placement will not be effective at all. Replacing this spy node with c will not decrease the advantage.

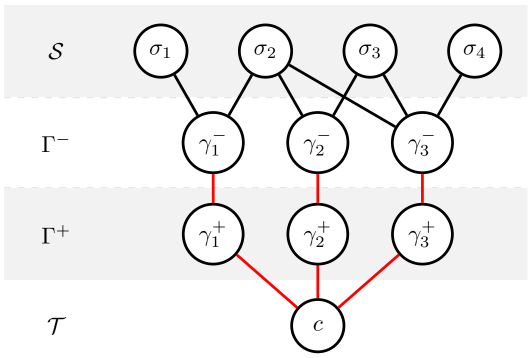


Fig. 5. Topology of G given a set cover instance where $\Gamma_1 = \{\sigma_1, \sigma_2\}, \Gamma_2 = \{\sigma_2, \sigma_3\}$ and $\Gamma_3 = \{\sigma_2, \sigma_3, \sigma_4\}$.

- contain only nodes in Γ^- other than c: If σ_i is chosen, then this spy node serves no other pair than (σ_i, c). We pick j where $\sigma_i \in \Gamma_j$ and choose γ_j^- instead. This will not decrease the advantage. If γ_j^+ is chosen, then it benefits the advantage by moving it to γ_j^-, even when this creates duplicates.

Therefore, to solve the advantage maximization, we are essentially picking nodes in Γ^-, which corresponds to picking subsets among $\Gamma_1, \cdots, \Gamma_q$. If we can solve the advantage maximization in polynomial time for all $k \in [2, q+1]$, we can enumerate all solutions and pick the smallest k where the advantage reaches p, the total number of elements in $\mathcal{S}$, and also the best advantage we have between $\mathcal{S}$ and $\mathcal{T} = \{c\}$. This also solves the minimum set cover problem known to be NP-hard, which is a contradiction. Therefore, Problem 1 is NP-hard, and such polynomial-time algorithm exists only if P = NP. ■

A.3.2 Proof of Proposition 6.2. We present the counter-example in Fig. 6, which is a tree with $2k+1$ branches. In this tree, the initial pair of nodes chosen by the greedy algorithm must be g and h_i for some $i \in [k]$, because a shortcut between them puts 3 pairs at maximum of sources and destinations $\{(g, t_{3i-j}) | j \in \{0, 1, 2\}\}$ under the risk of being front-run. At each of the following steps, the greedy algorithm will have to continue choosing an additional peer from $\{h_i | i \in [k]\}$, which further increases the advantage by 3. Hence, with 2ℓ peers where $\ell < k/2$ is an integer, the total advantage equals $6\ell - 3$.

However, these 2ℓ peer connections can be put to better uses. If we choose $\{s_1, \cdots, s_\ell\} \cup \{r_1, \cdots, r_\ell\}$ instead, then the shortcut pairs of sources and destinations can be described by $P = \{(s_i, r_j) | i \neq j, i \in [\ell], j \in [\ell]\}$, where $|P| = \ell(\ell-1)$. Therefore, the ratio of the greedy algorithm solution to the maximum advantage is at most $\frac{6\ell-3}{\ell(\ell-1)} \sim \frac{6}{\ell}$, which can be arbitrarily close to 0 as ℓ, k become arbitrarily large. Equivalently, the greedy algorithm cannot guarantee an α-approximately optimal solution for any $\alpha > 0$. ■

A.3.3 Proof of Theorem A.3. We first show the result for a single target, i.e., $|U| = 1$. Let a denote an adversarial node and x denote the message sender (that is, $U = \{x\}$). We let the adversary perform the following 3-step strategy iteratively.

Step 1. Wait Δ_1 for the first arrival of a new message M sent by x.
Step 2. Keep only the peer that contributed to the first arrival, and get $d-1$ random new addresses from the oracle to replace the other peers. The probability of x belonging in these addresses is lower-bounded by $q > 0$. Send peering requests repeatedly until a peer slot becomes available.

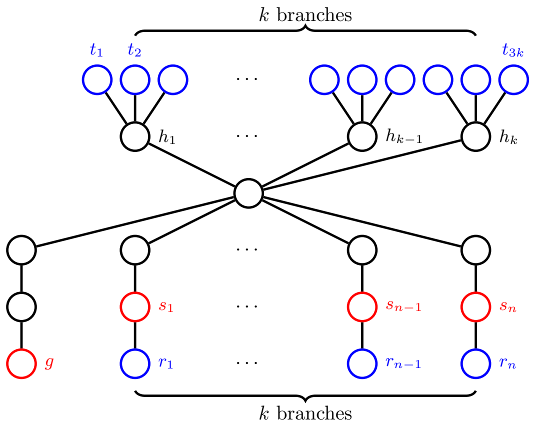


Fig. 6. Greedy algorithm cannot maximize advantage with proximity greater than 0. $\tau = 3.99$. Red nodes are sources and blue nodes are destinations.

If x is already one of the neighbors at Step 1, it will surely be kept in Step 2 because $w(x, a)$ is already the shortest path length between x and a on the network, implied by the triangle inequality (8). In the end, it will remain a peer of the adversary. Because the samplings for replacement nodes are independent across K iterations, we can geometrically decrease the probability that x is missed in the entire procedure by increasing K. This intuitively explains why K is of order $O(\log \varepsilon^{-1})$. Next, we make this intuition precise.

Let

$$K \triangleq \frac{\log(\varepsilon/2)}{\log(1-q)}, \quad \varepsilon_1 = \varepsilon_2 \triangleq \frac{\varepsilon \log(1-q)}{4\log(\varepsilon/2)}.$$

In Step 1, it is desired that the sender sends at least 1 message during the time window of length Δ_1 with probability at least $1 - \varepsilon_1$.

By Definition A.2, there exist constants μ, γ such that

$$\mathbb{P}\left[M_x(t + \Delta_1) - M_x(t) \geq \gamma\Delta_1\right] \geq 1 - \frac{\mu}{\Delta_1}.$$

This assumption essentially states that it is highly probable that during a sufficiently long window of time, a node broadcasts at least γ messages every unit of time on average.

Here by taking $\Delta_1 = \max\{\gamma^{-1}, \mu\varepsilon_1^{-1}\}$, we obtain the lower-bound probability that x broadcasts at least 1 message during any time interval of length at least Δ_1.

In Step 2, we would like to wait Δ_2 for all the $d - 1$ new nodes being connected to the adversary. This requires each new node to have at least 1 free slot. We consider the worst case where none of the slots are free in the beginning. For each target peer p, each one of the F slots will become free after a period of time T_0, where $T_0 \sim \text{Exp}(\lambda)$. Since one slot is already sufficient, the probability the peer p becomes available after Δ_2 is lower-bounded by $1 - e^{-\lambda\Delta_2}$. This expression is further lower-bounded by $1 - \dfrac{\zeta}{\Delta_2}$ for $\zeta = \dfrac{1}{e\lambda}$.

Then, considering all the $(d - 1)$ peers, we may use the union bound to derive a lower bound of the probability P_3 that all the $d - 1$ peers are available.

$$P_3 \geq 1 - \frac{(d-1)\zeta}{\Delta_2}.$$

We want P_3 to be at least $1 - \varepsilon_2$. This can be achieved by letting

$$\Delta_2 = \frac{(d-1)\zeta}{\varepsilon_2}.$$

After taking K iterations of these steps, the probability of finding x equals $1 - (1-q)^K$, while the probability that all executions of Steps 1 & 3 are successful is at least $1 - K\varepsilon_1 - K\varepsilon_2$. The overall probability equals

$$\begin{aligned}\left[1 - (1-q)^K\right](1 - K\varepsilon_1 - K\varepsilon_2) &\geq 1 - (1-q)^K - K(\varepsilon_1 + \varepsilon_2) \\ &= 1 - \frac{\varepsilon}{2} - \frac{\varepsilon}{2} \\ &= 1 - \varepsilon.\end{aligned}$$

On the other hand, the total time consumption equals

$$\begin{aligned}K\Delta_1 + K\Delta_2 &= O(\log \varepsilon^{-1})\left[O\left(\varepsilon^{-1}\log\varepsilon^{-1}\right) + O\left(\varepsilon^{-1}\right)\right] \\ &= O\left(\varepsilon^{-1}\log^2(\varepsilon^{-1})\right). \end{aligned} \tag{10}$$

Next, we show how to extend this result to an arbitrary U via a union bound.

From (10), we know that with probability $1 - \varepsilon/|U|$, the adversary is able to find the network ID of an arbitrary node $u \in U$ within time

$$O\left((\varepsilon/|U|)^{-1}\log^2(\varepsilon/|U|)\right) = O\left(|U|\varepsilon^{-1}\log^2\left(\varepsilon^{-1}\right)\right).$$

Regardless of how the adversary allocates time to the tasks of finding each $u \in U$, the probability of finding all of them is at least $1 - |U| \times \varepsilon/|U| = 1 - \varepsilon$ by the union bound. As for the time consumption, we consider the worst case where the agent has to run the algorithms for finding each $u \in U$ sequentially. In this case, the total time consumption is the time consumption above of each single task multiplied by number of tasks $|U|$, which equals

$$O\left(|U|^2\varepsilon^{-1}\log^2\left(\varepsilon^{-1}\right)\right).$$ ■

A.3.4 Proof of Proposition 7.3. First of all, we assume N, the number of nodes, to be upper-bounded so that the probability q that the oracle returns a target after a single draw satisfies $0 < q_1 \leq q \leq q_2 < 1$ for some constants q_1, q_2. In order to connect to the target, it is necessary for the adversary to draw it using the oracle. Temporarily, we ignore the other necessary steps (such as judging if an existing peer is the target) and consider only drawing nodes. Then, the probability P that the target is drawn within K steps satisfies

$$1 - (1-q_1)^K \leq P \leq 1 - (1-q_2)^K.$$

To let $P = 1 - \varepsilon$, we should equivalently have $K = \Theta(\log\varepsilon^{-1})$ because

$$\frac{\log\varepsilon}{\log(1-q_2)} \leq K \leq \frac{\log\varepsilon}{\log(1-q_1)}.$$

Considering that each drawing takes $\Theta(1)$ time, it takes at least $\Theta(K) = \Theta(\log\varepsilon^{-1})$ time to find the target with the oracle w.p. at least $1 - \varepsilon$. As argued above, drawing the target is a necessary condition for the final peer connection to it. Hence, $\Omega(\log\varepsilon^{-1})$ is a lower bound of time consumption. ■

A.3.5 Proof of Lemma A.1. Let ν denote the arrival rate. For any Δ, we assign $k = \nu\Delta/2$, and obtain

$$\begin{aligned}\mathbb{P}\left[M_u(t+\Delta) - M_u(t) < \frac{\nu\Delta}{2}\right] &= e^{-\nu\Delta} \sum_{i=0}^{\lfloor \nu\Delta/2 \rfloor} \frac{(\nu\Delta)^i}{i!} \\ &\sim e^{-2k} \sum_{i=0}^{k} \frac{(2k)^i}{i!} \\ &\lesssim ke^{-2k} \sup_{t\in[0,k]} \frac{(2k)^t}{t!} \\ &\lesssim ke^{-k} \qquad (*) \\ &\lesssim \frac{1}{k} \sim \frac{1}{\nu\Delta}.\end{aligned}$$

This already justifies the claim. It remains to prove (*). By Stirling's approximation, let

$$f(t) = \log \frac{(2k)^t}{t!} \approx t\log(2k) - \frac{\log t}{2} - t\log t + t.$$

As a result,

$$f'(t) \approx \log(2k) - \frac{1}{2t} - \log t.$$

As $\log t + 1/(2t)$ monotonically increases in $(1/2, \infty)$, we may assert that the maximizer τ satisfies

$$f'(\tau) = 0 \iff 2k = \tau e^{\frac{1}{2\tau}}.$$

It can be confirmed that $\tau \in [0, k]$. Hence,

$$\sup_{t\in[0,k]} \frac{(2k)^t}{t!} \sim \frac{(\tau e^{\frac{1}{2\tau}})^\tau}{\sqrt{\tau}\tau^\tau e^{-\tau}} \lesssim e^\tau \lesssim e^k. \qquad \blacksquare$$

A.4 Statistical Significance of Direct Global Latency Measurements

To establish the statistical significance of the mean difference among our direct global latency distributions measured in Section 5.1 over the Ethereum mainnet, we first perform the Kruskal-Wallis H test [30]. The resulting p-value equals 0, so a subsequent Dunn's test is recommended. We perform Dunn's test [22] with Bonferroni correction [13], and obtained a 0 p-value between each pair of distributions. This test supports statistically significant differences in the reduction of direct global latency effected by the three methods, ordered by the means of their corresponding distributions.

A.5 Evaluation of direct targeted latency over Ethereum Mainnet

We ran limited experiments on targeted latency by measuring transactions from a target / victim node in Germany operated by Chainlink, an organization providing widely used blockchain services ; the node sends 2 transactions per hour on average. These transactions originate from an account (Ethereum address) exclusive to the victim node, i.e., all transactions are sent only by the victim. The node kept running on the network during our experiments.

We evaluate each of the four methods from §5.1.1, measuring both their ability to reduce targeted latency and their ability to connect to (i.e., infer) the IP address of our target node on the Ethereum main P2P network. To this end, we establish 4 EC2 instances in the ap-southeast-1 AWS data center in Singapore, which have a 160 ms round-trip time to Germany. They are located within the same subnet as a public bloXroute relay. As in §5.1, we deploy a full Go-Ethereum node on each host with at most 50 peers. The proportion of outbound peers remain the same. We conduct the experiments under the same procedure and take similar anti-biasing measures, except for three key differences. First, we define the relevant transactions as only those sent by the target's account. Second, we set the Peri period to 30 minutes to match the frequency of source transactions. Third, the duration of each experiment is extended from 8 hours to 16 hours to permit a larger number of Peri periods.

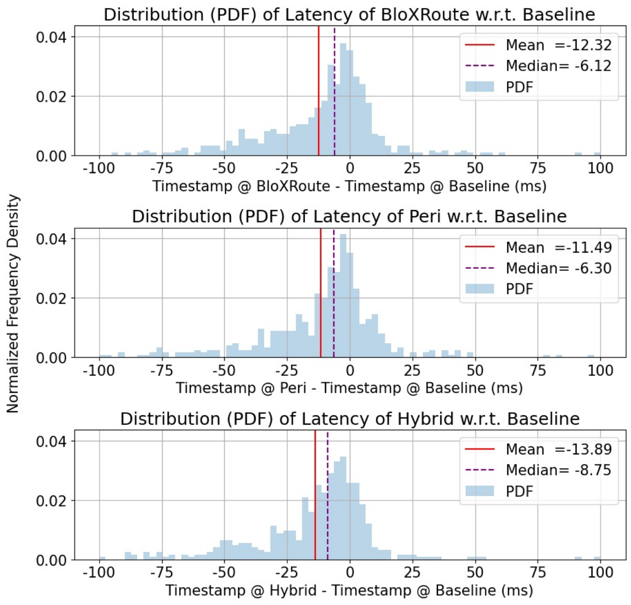


Fig. 7. PDFs of distributions of targeted latency.

We ran 23 experiments from March 19, 2022 to April 19, 2022. The Hybrid and Baseline clients were able to establish connections to the victim node once each. The distributions of targeted latency of each method are displayed in Fig. 7. In total, we analyzed the latencies of 654 transactions.

Over distributions in Fig. 7, we performed Kruskal-Wallis test and obtained p-value = 0.0235, which allows us to reject the null hypothesis at significance level 5% and continue with a Dunn's test. Then, we performed Dunn's test with Bonferroni correction over these distributions. The p-values are listed in Table 4. At significance level 5%, we cannot assert a difference between the means of any pair of distributions. This indicates that all the methods share a similar ability to reduce direct targeted latency. We can further observe this phenomenon from Fig. 7, where they share a similar ability to reduce direct targeted latency by 14% to 18% of the end-to-end delay, and the hybrid method outperforms the other two methods with a slight advantage. Unlike reduction of direct global latency, the Peri algorithm is no longer significantly worse than the bloXroute relay network at reducing direct targeted latency, which makes it a free replacement of bloXroute services for agents with specific targets. An agent with bloXroute services can also further boost the latency reduction by an additional 13% by stacking the Peri algorithm and turning hybrid. In addition, Peri can potentially help an agent identify the IP address of a victim, while bloXroute, which intermediates connections, cannot support such functionality.

A.6 Evaluation of Relation Between direct global latency and Peer Count

We ran additional experiments for comparing direct global latency when we vary the peer count of the adversary. We use 2 EC2 (i3.xlarge) instances in the us-east-1 AWS data center, where each

Pair	(B, P)	(B, H)	(P, H)
p-value	1.0	0.0532	0.0529

Table 4. Pairwise p-values of Dunn's test over distributions of targeted latency in Fig. 7, comparing bloXroute (B), Peri (P), and Hybrid (H).

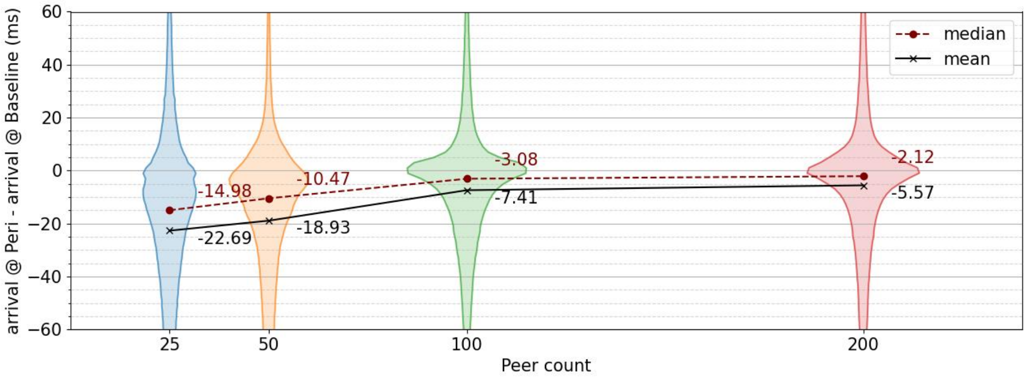


Fig. 8. PDFs of global latency with maximum peer counts 25, 50, 100 and 200. The PDFs are normalized by the same factor for legibility.

instance deploys a full node with a customized Go-Ethereum (geth) client, built on top of the nemata (1.10.25) release. To correctly run the nodes after the Ethereum merge in September 2022, we ran a lighthouse client[16] at default settings in parallel. We vary the maximum peer count in the range from 25 to 200. For each peer count, we ran 12 experiments in January 2023 with each following the procedure below. Each experiment lasted 8 hours. We adaptively selected the period of Peri for each peer count to let the Peri node accumulate enough candidates to choose from. The experiments were configured by the same workflow as described in §5.1. For financial reasons, we did not test bloXroute and Hybrid methods with 2 additional AWS nodes.

Each node is allowed to warm up for 2.5 hours; after this, we collect all transactions that are received by each of the nodes for 3.5 hours. As we did in §5.1, we collected the samples of random variable $\Lambda(y) - \Lambda(b)$, which represents the arrival time difference of one transaction m between at the Peri node y and the baseline node b. The smaller the time difference (i.e., the more negative), the earlier y delivers m, and the more effective Peri is. For each peer count, we gather the latency differences over 12 experiments, and plot their distributions for each node in Fig. 8. Each distribution is estimated over 500,000 transactions.

In Fig. 8, we observe that Peri maintains an advantage over the baseline, regardless of the peer count. However, the more peers the node maintains, the less advantage Peri has, and the more concentrated the distribution is around 0. It is still notable that Peri has a 5.5ms advantage with a large peer count of 200.

A.7 Performance of Peri under Limited Bandwidth

We ran additional experiments for comparing direct global latency under different networking conditions. Our nodes were deployed in the same way as in App. A.6. We set the peer count to 50, the default option of Go-Ethereum. We limited the bandwidth of both nodes by setting the bandwidth limit of both inbound and outbound traffic. In all experiments where the bandwidths

[16] https://github.com/sigp/lighthouse/releases/tag/v3.3.0

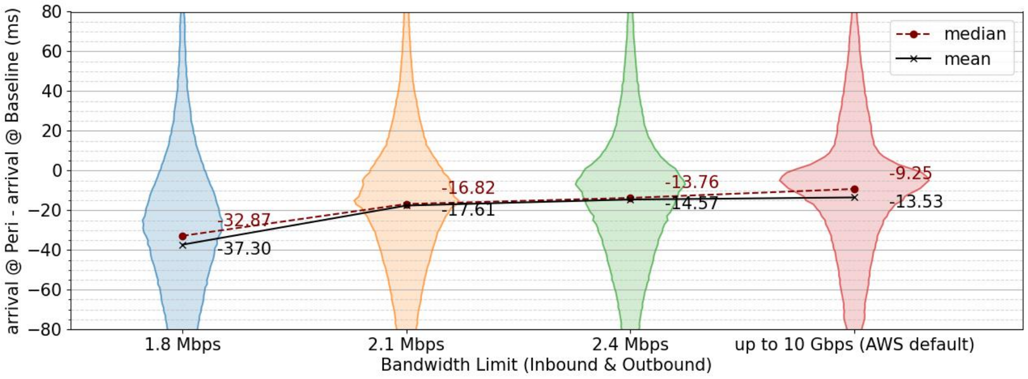


Fig. 9. PDFs of distributions of global latency under limited bandwidth.

were unlimited, the machines had "up to 10 Gbps" bandwidth. [17] For each bandwidth, we ran 10 experiments in January 2023 with each lasting 4.8 hours. The experiments were configured by the same workflow as described in §5.1. For financial reasons, we did not test bloXroute and Hybrid methods with 2 additional AWS nodes.

Each node is allowed to warm up for 2.5 hours; after this, we collect all transactions that are received by each of the nodes for 2.3 hours. The data is collected in the same way as we did in §5.1 and App. A.6. For each bandwidth, we gather the latency differences over 10 experiments, and plot their distributions for each node in Fig. 9. Each distribution is estimated over 350,000 transactions.

In Fig. 9, we observe that the lower bandwidth an agent has access to, the higher advantage Peri gains over the baseline.

A.8 Full Result of Simulations of Advantage Maximization Algorithms on Hub-Enriched Topologies

We present the extended results of §4 and a full description of our simulation setups, which we lack space to show in the main text. Cryptocurrency P2P topologies are notoriously difficult and expensive to measure [19], and to our knowledge, there are no public datasets of recent cryptocurrency P2P topologies [20, 21, 23, 39]. As a compromise, we took 2 snapshots of cryptocurrency topologies in real life – the Bitcoin P2P network topology on 9/4/2015 [38] and the Lightning network topology in 8/23/2022 [17]. The Bitcoin data is not publicly available, and was obtained by contacting the authors of [38]. The LN snapshot was directly collected [17]. Although the Lightning network is a layer-2 network where latency wars do not typically take place, it is still a P2P network and its topology may have some similarities to layer-1 P2P networks. Therefore, we add simulation results on the Lightning network topology for reference. The major component of the Bitcoin P2P network consists of 4,654 nodes and 18,467 edges, and that of the Lightning network consists of 36,553 nodes and 296,589 edges. For a fair comparison against the random topologies, we generate 25 sub-topologies of each topology by the snowball-sampling algorithm. All subgraphs consist of 300 nodes, but their average node degree differ. The average degrees of Bitcoin subgraphs range from 2.3 to 4.9, while those of Lightning subgraphs range from 3 to 35. Fig. A.8 shows the performance in advantage metric of 3 different methods over 4 random synthetic topologies and 2 topologies in practice.

[17]The speed tests typically report 1 to 2 Gbps.

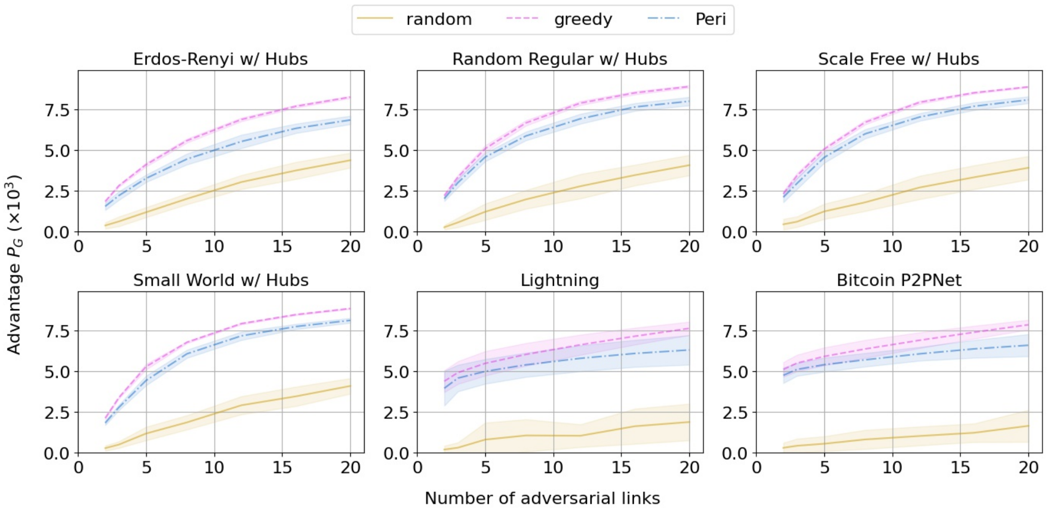


Fig. 10. Full version of Fig. 4. The mean of the curves are shown by solid lines, and the standard deviations are shown by the transparent color zones centered at the mean.

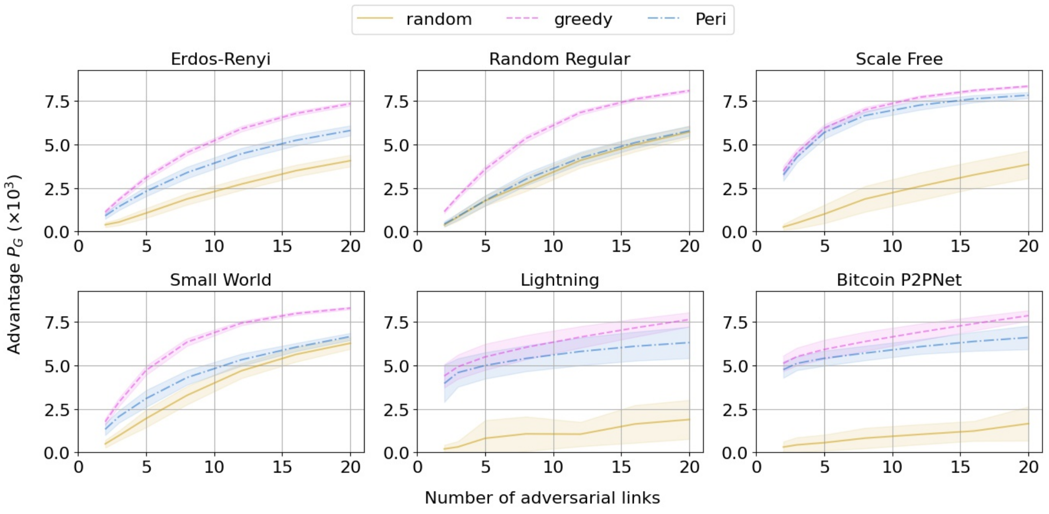


Fig. 11. Advantage-Peer-count curves on original random graph models. The mean of the curves are shown by solid lines, and the standard deviations are shown by the transparent color zones centered at the mean.

A.9 Simulations of Advantage Maximization Algorithms on Original Topologies

We synthesize the network models as in §6.2 without centralizing them by introducing hub nodes. We reuse other setups in the original simulation, and plot the advantage-peer-count curves in Fig. 11.

For all the graph models, both Peri and Greedy achieve a higher advantage A_N than the random baseline, with Greedy outperforming Peri by varying amounts. For the most decentralized models, such as the random regular and small world, the advantage of Peri is much closer to that of random baseline than the greedy algorithm. On the other hand, for models with a few high degree nodes (i.e., hubs), Peri inserts shortcut peering connections almost as well as the greedy algorithm, in spite of its limited knowledge of the graph. Therefore, the performance of Peri is likely to be stronger on networks with many hubs. This is consistent with our conclusion in §6.2.

Received August 2022; revised February 2023; accepted April 2023